# **WEEK 5: Microservices Architecture using ASP.NET Core Web API**

## **6. (Web API) Kafka Chat Application**

### **Code:**

**KafkaChatApp/Program.cs**

using System;

using System.Threading;

using System.Threading.Tasks;

using Confluent.Kafka;

class Program

{

    private const string bootstrapServers = "localhost:9092";

    private const string topicName = "chat-messages";

    static async Task Main(string[] args)

    {

        Console.WriteLine("Kafka Chat Application");

        Console.Write("Enter your name: ");

        string? userName = Console.ReadLine();

        if (string.IsNullOrEmpty(userName))

        {

            Console.WriteLine("User name cannot be empty.");

            return;

        }

        var cts = new CancellationTokenSource();

        var consumerTask = Task.Run(() => StartConsumer(cts.Token));

        await StartProducer(userName!, cts.Token);

        cts.Cancel();

        await consumerTask;

    }

    static async Task StartProducer(string userName, CancellationToken token)

    {

        var config = new ProducerConfig { BootstrapServers = bootstrapServers };

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

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

        while (!token.IsCancellationRequested)

        {

            string? message = Console.ReadLine();

            if (string.IsNullOrEmpty(message))

                continue;

            if (message.ToLower() == "exit")

                break;

            try

            {

                var kafkaMessage = new Message<Null, string> { Value = $"{userName}: {message}" };

                await producer.ProduceAsync(topicName, kafkaMessage, token);

            }

            catch (Exception ex)

            {

                Console.WriteLine($"Error producing message: {ex.Message}");

            }

        }

    }

    static void StartConsumer(CancellationToken token)

    {

        var config = new ConsumerConfig

        {

            BootstrapServers = bootstrapServers,

            GroupId = "chat-consumer-group-" + Guid.NewGuid(),

            AutoOffsetReset = AutoOffsetReset.Earliest

        };

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

        consumer.Subscribe(topicName);

        try

        {

            while (!token.IsCancellationRequested)

            {

                try

                {

                    var cr = consumer.Consume(token);

                    if (!string.IsNullOrEmpty(cr.Message.Value))

                    {

                        Console.WriteLine($"[Received] {cr.Message.Value}");

                    }

                }

                catch (ConsumeException e)

                {

                    Console.WriteLine($"Error consuming message: {e.Error.Reason}");

            }

        }

        catch (OperationCanceledException)

        {

consumer.Close();

        }

    }

}

**KafkaChatConsumer/Program.cs:**

using Confluent.Kafka;

using System;

class Program

{

    static void Main(string[] args)

    {

        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-messages");

        Console.WriteLine("Kafka Consumer started. Waiting for messages...");

        while (true)

        {

            var consumeResult = consumer.Consume();

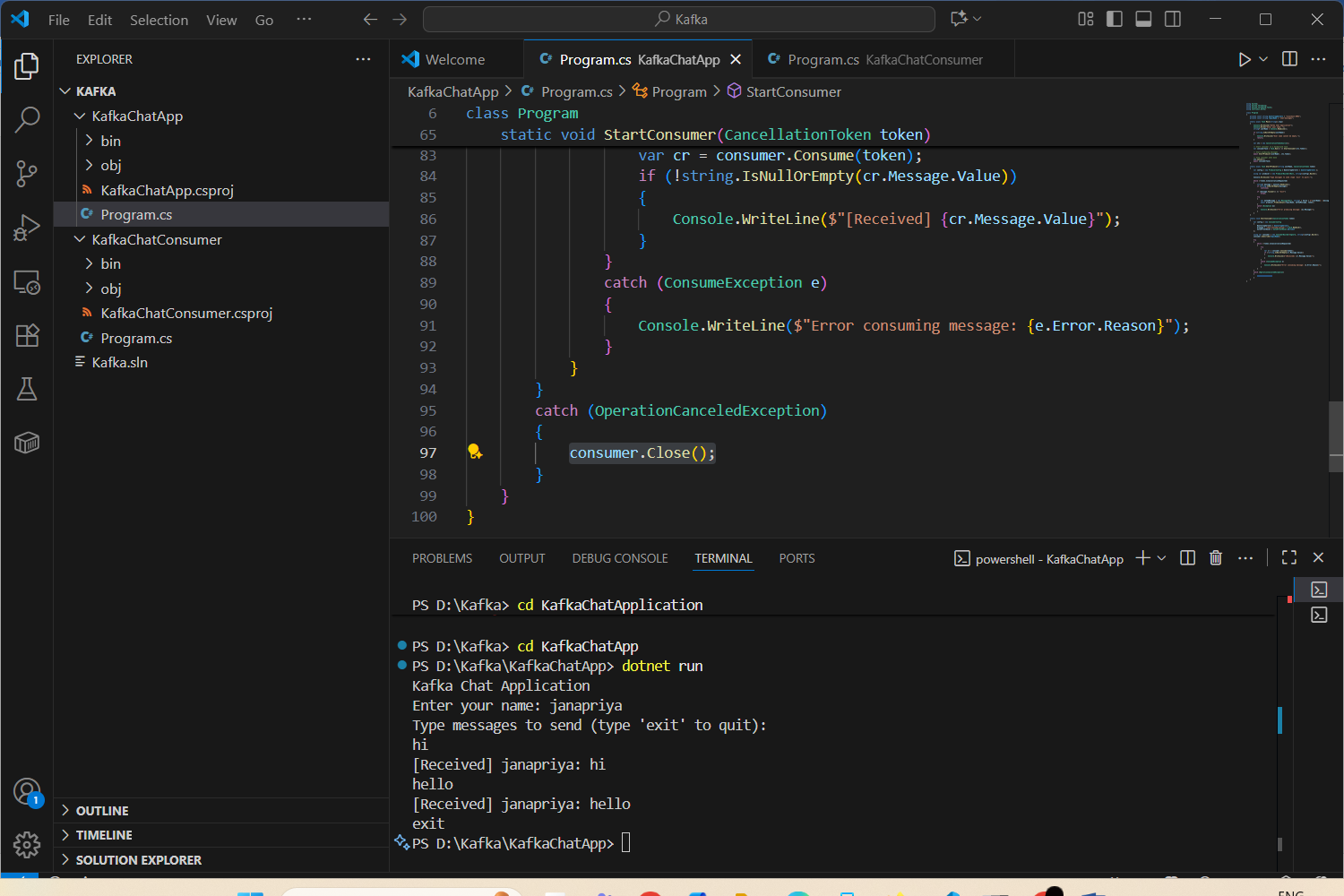
            Console.WriteLine($"Received: {consumeResult.Message.Value}");

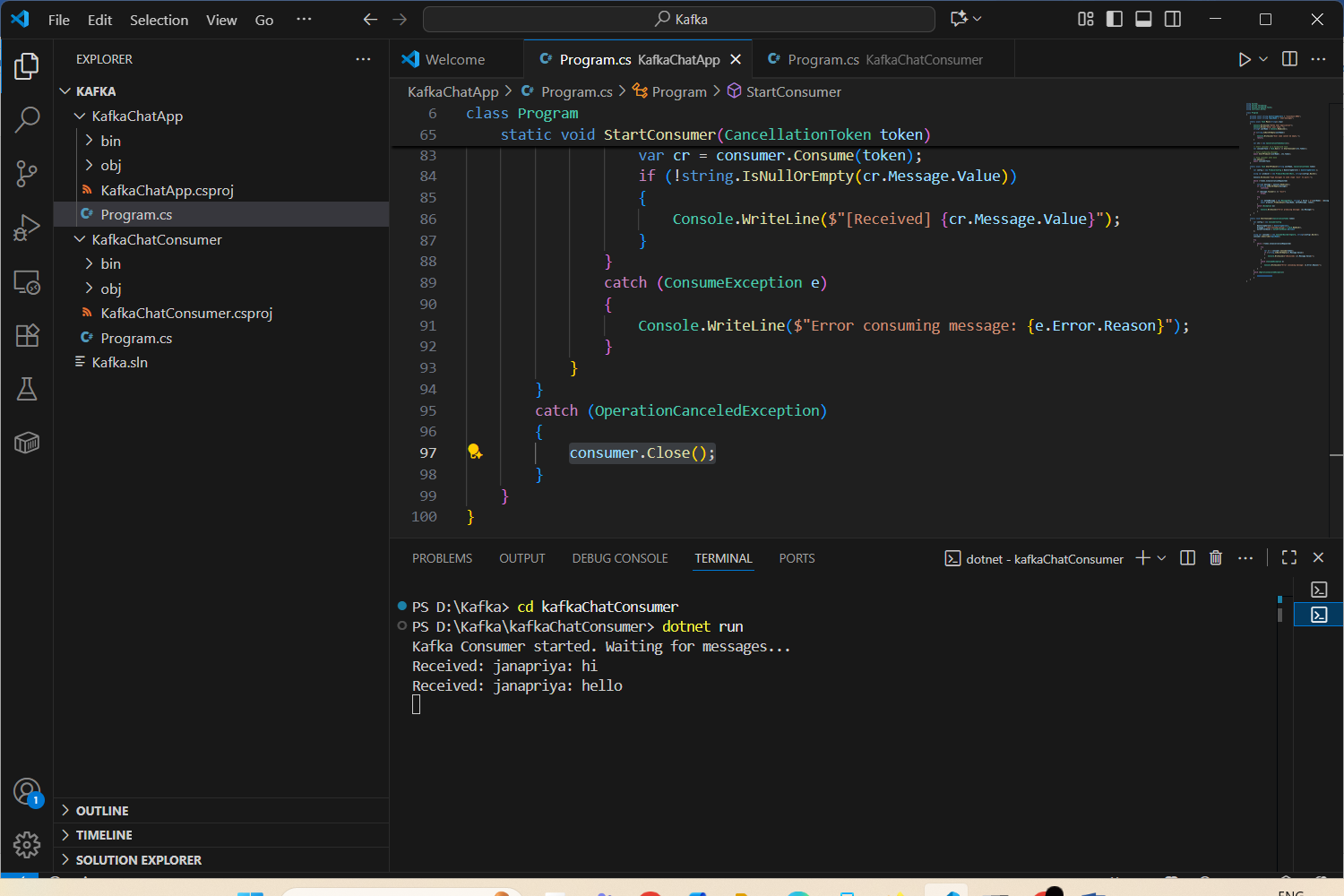
        }

    }

}

### **Output:**





## **MicroServices**

### **Code:**

**Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

var jwtKey = builder.Configuration["Jwt:Key"];

var jwtIssuer = builder.Configuration["Jwt:Issuer"];

var jwtAudience = builder.Configuration["Jwt:Audience"];

if (string.IsNullOrWhiteSpace(jwtKey) || string.IsNullOrWhiteSpace(jwtIssuer) || string.IsNullOrWhiteSpace(jwtAudience))

    throw new Exception("JWT settings are missing in appsettings.json");

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

    .AddJwtBearer(options =>

    {

        options.TokenValidationParameters = new TokenValidationParameters

        {

            ValidateIssuer = true,

            ValidateAudience = true,

            ValidateLifetime = true,

            ValidateIssuerSigningKey = true,

            ValidIssuer = jwtIssuer,

            ValidAudience = jwtAudience,

            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtKey))

        };

        options.Events = new JwtBearerEvents

        {

            OnAuthenticationFailed = context =>

            {

                if (context.Exception.GetType() == typeof(SecurityTokenExpiredException))

                {

                    context.Response.Headers["Token-Expired"] = "true";

                }

                return Task.CompletedTask;

            }

        };

    });

builder.Services.AddAuthorization();

var app = builder.Build();

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.MapGet("/", () => "API is running!");

app.Run();

**Controller/AdminController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

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

public class AdminController : ControllerBase

{

    [HttpGet("dashboard")]

    [Authorize(Roles = "Admin")]

    public IActionResult GetAdminDashboard()

    {

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

    }

}

**Controller/AuthController.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace JwtDemoApi.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)

        {

            return (model.Username == "admin" && model.Password == "123") ||

                   (model.Username == "user" && model.Password == "123");

        }

        private string GenerateJwtToken(string username)

        {

            var claims = new[]

            {

                new Claim(ClaimTypes.Name, username),

                new Claim(ClaimTypes.Role, username == "admin" ? "Admin" : "User")

            };

            var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_config["Jwt:Key"]!));

            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"])),

                signingCredentials: creds

            );

            return new JwtSecurityTokenHandler().WriteToken(token);

        }

    }

    public class LoginModel

    {

        public string? Username { get; set; }

        public string? Password { get; set; }

    }

}

**Controller/TestController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

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

public class TestController : ControllerBase

{

    [HttpGet("protected")]

    [Authorize]

    public IActionResult GetProtectedData()

    {

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

    }

}

### **Output:**

