Tanu Singh Superset ID: 6362191

**Kafka Integration with C#**

Codes:

KafkaConsumer.cs

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

    <OutputType>Exe</OutputType>

    <TargetFramework>net9.0</TargetFramework>

    <ImplicitUsings>enable</ImplicitUsings>

    <Nullable>enable</Nullable>

  </PropertyGroup>

 <ItemGroup>

    <PackageReference Include="Confluent.Kafka" Version="2.11.0" />

  </ItemGroup>

  </Project>

Program.cs

using System;

using System.Threading;

using Confluent.Kafka;

class Program

{

    static void Main(string[] args)

    {

        var config = new ConsumerConfig

        {

            BootstrapServers = "localhost:9092",

            GroupId = "chat-consumer-group",

            AutoOffsetReset = AutoOffsetReset.Earliest

        };

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

        consumer.Subscribe("chat-messages");

        Console.WriteLine("🎧 Kafka Chat Consumer Started!");

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

        var cts = new CancellationTokenSource();

        Console.CancelKeyPress += (\_, e) => {

            e.Cancel = true;

            cts.Cancel();

        };

        try

        {

            while (!cts.Token.IsCancellationRequested)

            {

                try

                {

                    var consumeResult = consumer.Consume(cts.Token);

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

                }

                catch (ConsumeException e)

                {

                    Console.WriteLine($"❌ Consume error: {e.Error.Reason}");

                }

            }

        }

        catch (OperationCanceledException)

        {

            Console.WriteLine("🛑 Consumer canceled.");

        }

        finally

        {

            consumer.Close();

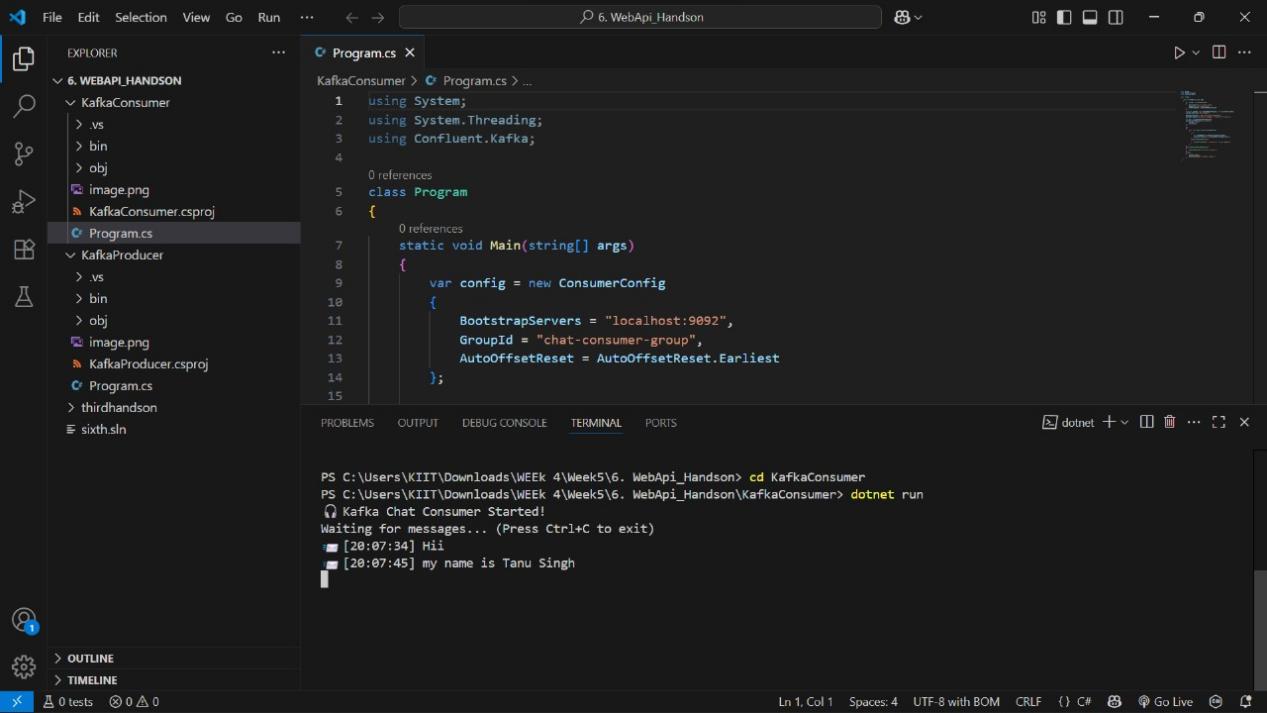
            Console.WriteLine("✅ Consumer closed.");

        }

    }

}

Output:



KafkaProducer.cs

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

    <OutputType>Exe</OutputType>

    <TargetFramework>net9.0</TargetFramework>

    <ImplicitUsings>enable</ImplicitUsings>

    <Nullable>enable</Nullable>

  </PropertyGroup>

  <ItemGroup>

    <PackageReference Include="Confluent.Kafka" Version="2.11.0" />

  </ItemGroup>

  </Project>

Program.cs

using Confluent.Kafka;

using System;

using System.Threading;

using System.Threading.Tasks;

class Program

{

    private static readonly CancellationTokenSource cts = new CancellationTokenSource();

    static async Task Main(string[] args)

    {

        Console.CancelKeyPress += (\_, e) =>

        {

            e.Cancel = true;

            cts.Cancel();

            Console.WriteLine("\nShutting down...");

        };

        var config = new ProducerConfig

        {

            BootstrapServers = "localhost:9092",

            EnableDeliveryReports = true,

            Acks = Acks.All,

            MessageTimeoutMs = 5000,

            SocketTimeoutMs = 30000,

            MessageSendMaxRetries = 3,

            RetryBackoffMs = 1000

        };

        try

        {

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

                .SetLogHandler((\_, logMessage) =>

                {

                    Console.WriteLine($"[Kafka] {logMessage.Level}: {logMessage.Message}");

                })

                .SetErrorHandler((\_, error) =>

                {

                    Console.WriteLine($"[Kafka Error] {error.Reason}");

                    if (error.IsFatal) cts.Cancel();

                })

                .Build();

            Console.WriteLine("Kafka Chat Producer Started");

            Console.WriteLine("Type messages (press Ctrl+C or type 'quit' to exit):");

             while (!cts.IsCancellationRequested)

            {

                Console.Write("You: ");

                var message = await ReadLineAsync(cts.Token);

                if (string.IsNullOrWhiteSpace(message)) continue;

                if (message.Equals("quit", StringComparison.OrdinalIgnoreCase)) break;

                var messageWithTimestamp = $"[{DateTime.Now:HH:mm:ss}] {message}";

                try

                {

                    var deliveryResult = await producer.ProduceAsync(

                        "chat-messages",

                        new Message<Null, string> { Value = messageWithTimestamp },

                        cts.Token

                    );

                    Console.WriteLine($"Message delivered to partition {deliveryResult.Partition}, offset {deliveryResult.Offset}");

                }

                catch (ProduceException<Null, string> ex)

                {

                    Console.WriteLine($"Failed to deliver message: {ex.Error.Reason}");

                }

                catch (OperationCanceledException)

                {

                    Console.WriteLine("Message delivery cancelled");

                    break;

                }

                catch (Exception ex)

                {

                    Console.WriteLine($"Unexpected error: {ex.Message}");

                }

            }

        }

        catch (Exception ex)

        {

            Console.WriteLine($"Fatal error: {ex.Message}");

        }

        finally

        {

            cts.Dispose();

            Console.WriteLine("Producer shut down completed");

        }

    }

    private static async Task<string> ReadLineAsync(CancellationToken cancellationToken)

    {

        string input = null;

        var task = Task.Run(() =>

        {

            input = Console.ReadLine();

        }, cancellationToken);

        try

        {

            await task;

            return input;

        }

        catch (TaskCanceledException)

        {

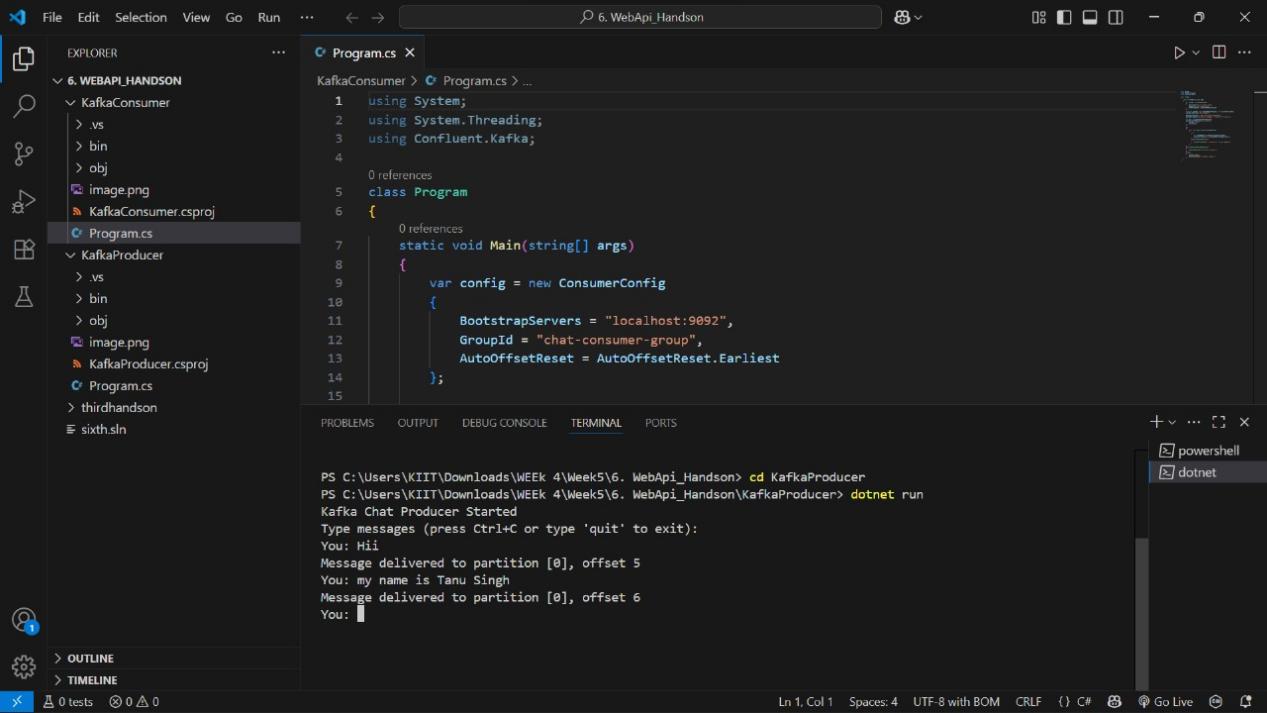
            return string.Empty;

        }

    }

}

Output:



Authentication and Authorization in ASP.NET Core Web API Microservices

Codes:

fifth.cs

<Project Sdk="Microsoft.NET.Sdk.Web">

<PropertyGroup>

    <TargetFramework>net9.0</TargetFramework>

    <Nullable>enable</Nullable>

    <ImplicitUsings>enable</ImplicitUsings>

  </PropertyGroup>

  <ItemGroup>

    <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="9.0.7" />

    <PackageReference Include="Microsoft.AspNetCore.Cors" Version="2.3.0" />

    <PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="9.0.6" />

    <PackageReference Include="Swashbuckle.AspNetCore" Version="9.0.3" />

    <PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="8.12.1" />

  </ItemGroup>

<Project>

appsettings.json

{

  "Jwt": {

    "Key": "YourSuperSecretKey1234567890",

    "Issuer": "https://localhost:5001",

    "Audience": "https://localhost:5001",

    "DurationInMinutes": 30

  }

}

appsettings.Development.json

{

  "Logging": {

    "LogLevel": {

      "Default": "Information",

      "Microsoft.AspNetCore": "Warning"

    }

  }

}

Program.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAuthentication(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();

var app = builder.Build();

app.UseAuthentication();

app.UseAuthorization();

app.Run();

Output:

