**Exercise 1:6.WebApi\_HandsOn**

**//chat\_producer.py**

from confluent\_kafka import Producer

import sys

KAFKA\_TOPIC = 'chat\_messages'

def delivery\_report(err, msg):

if err is not None:

print(f"Message delivery failed: {err}")

else:

print(f"Message delivered to topic '{msg.topic()}' [{msg.partition()}] @ offset {msg.offset()}")

def run\_producer():

producer\_conf = {

'bootstrap.servers': KAFKA\_BROKER

}

producer = Producer(producer\_conf)

print(f"Kafka Producer started. Type messages and press Enter to send. (Topic: {KAFKA\_TOPIC})")

print("Press Ctrl+C to exit.")

try:

while True:

message = input("> ")

if not message:

continue

producer.produce(KAFKA\_TOPIC, value=message.encode('utf-8'), callback=delivery\_report)

producer.poll(0)

except KeyboardInterrupt:

print("\nExiting producer...")

finally:

producer.flush()

print("Producer flushed and closed.")

if \_\_name\_\_ == "\_\_main\_\_":

run\_producer()

**//chat\_consumer.py**

from confluent\_kafka import Consumer, KafkaException, KafkaError

import sys

KAFKA\_BROKER = 'localhost:9092'

KAFKA\_TOPIC = 'chat\_messages'

CONSUMER\_GROUP\_ID = 'my\_chat\_group'

def run\_consumer():

consumer\_conf = {

'bootstrap.servers': KAFKA\_BROKER,

'group.id': CONSUMER\_GROUP\_ID,

'auto.offset.reset': 'earliest'

}

consumer = Consumer(consumer\_conf)

try:

consumer.subscribe([KAFKA\_TOPIC])

print(f"Kafka Consumer started. Listening for messages on topic '{KAFKA\_TOPIC}'...")

print("Press Ctrl+C to exit.")

while True:

msg = consumer.poll(1.0)

if msg is None:

continue

if msg.error():

if msg.error().code() == KafkaError.\_PARTITION\_EOF:

sys.stderr.write(f"%% {msg.topic()} [{msg.partition()}] reached end offset {msg.offset()}\n")

elif msg.error():

raise KafkaException(msg.error())

else:

print(f"Received message: {msg.value().decode('utf-8')}")

except KeyboardInterrupt:

print("\nExiting consumer...")

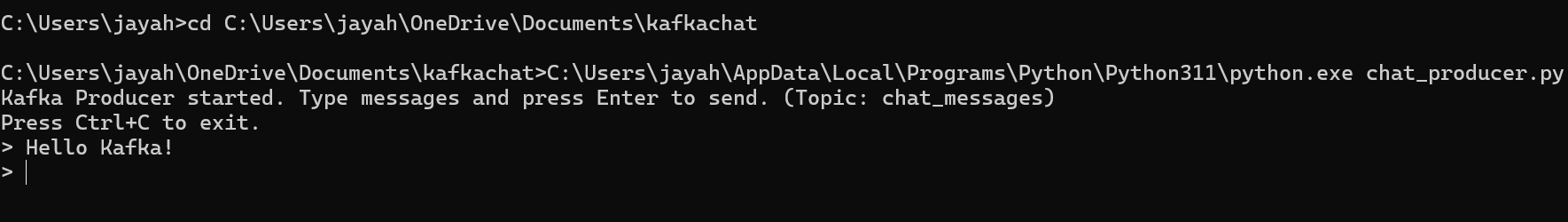
finally:

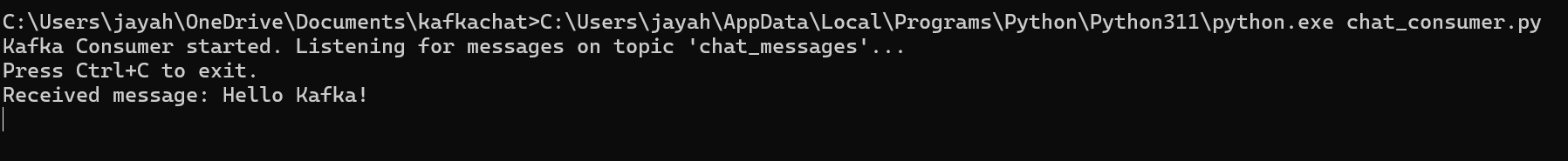
consumer.close()

print("Consumer closed.")

if \_\_name\_\_ == "\_\_main\_\_":

run\_consumer()





**//Form1.cs**

using System;

using System.Drawing;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using Confluent.Kafka;

namespace KafkaChatAppCSharp

{

public partial class Form1 : Form

{

private TextBox txtUsername;

private TextBox txtMessage;

private Button btnSend;

private ListBox lstChat;

private string topic = "chat-messages";

private string bootstrapServers = "localhost:9092";

private IProducer<Null, string> producer;

public Form1()

{

InitializeUI();

var config = new ProducerConfig { BootstrapServers = bootstrapServers };

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

StartConsumer();

}

private void InitializeUI()

{

this.Text = "Kafka Chat App";

this.Size = new Size(520, 420);

this.StartPosition = FormStartPosition.CenterScreen;

txtUsername = new TextBox { Location = new Point(10, 10), Width = 200 };

txtMessage = new TextBox { Location = new Point(10, 50), Width = 350 };

btnSend = new Button { Text = "Send", Location = new Point(370, 48), Width = 80 };

lstChat = new ListBox { Location = new Point(10, 90), Width = 460, Height = 250 };

btnSend.Click += btnSend\_Click;

this.Controls.Add(txtUsername);

this.Controls.Add(txtMessage);

this.Controls.Add(btnSend);

this.Controls.Add(lstChat);

}

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

{

string username = txtUsername.Text.Trim();

string message = txtMessage.Text.Trim();

if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(message))

{

MessageBox.Show("Please enter both username and message.");

return;

}

string fullMessage = $"{username}: {message}";

try

{

await producer.ProduceAsync(topic, new Message<Null, string> { Value = fullMessage });

txtMessage.Clear();

}

catch (Exception ex)

{

MessageBox.Show("Error sending message: " + ex.Message);

}

}

private void StartConsumer()

{

Task.Run(() =>

{

var config = new ConsumerConfig

{

BootstrapServers = bootstrapServers,

GroupId = Guid.NewGuid().ToString(),

AutoOffsetReset = AutoOffsetReset.Earliest

};

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

consumer.Subscribe(topic);

try

{

while (true)

{

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

Invoke(new Action(() =>

{

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

lstChat.TopIndex = lstChat.Items.Count - 1; // auto-scroll

}));

}

}

catch (OperationCanceledException)

{

consumer.Close();

}

});

}

}

}

**//Program.cs**

using System;

using System.Windows.Forms;

namespace KafkaChatAppCSharp

{

static class Program

{

[STAThread]

static void Main()

{

Application.Run(new Form1());

}

}

}



**Exercise 2: Implement JWT Authentication in ASP.NET Core Web API**

**//Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

using System.Security.Claims;

using System.IdentityModel.Tokens.Jwt;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

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();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

**//AuthController.cs**

using Microsoft.AspNetCore.Mvc;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Microsoft.IdentityModel.Tokens;

using JwtAuthApi.Models;

using Microsoft.Extensions.Configuration;

namespace JwtAuthApi.Controllers

{

[ApiController]

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

public class AuthController : ControllerBase

{

private readonly IConfiguration \_configuration;

public AuthController(IConfiguration configuration)

{

\_configuration = configuration;

}

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

if (IsValidUser(model))

{

var token = GenerateJwtToken(model.Username);

return Ok(new { Token = token });

}

return Unauthorized("Invalid credentials.");

}

private bool IsValidUser(LoginModel model)

{

return model.Username == "testuser" && model.Password == "password123";

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())

};

var jwtKey = \_configuration["Jwt:Key"];

var jwtIssuer = \_configuration["Jwt:Issuer"];

var jwtAudience = \_configuration["Jwt:Audience"];

var jwtDurationInMinutes = Convert.ToDouble(\_configuration["Jwt:DurationInMinutes"]);

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

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

var token = new JwtSecurityToken(

issuer: jwtIssuer,

audience: jwtAudience,

claims: claims,

expires: DateTime.Now.AddMinutes(jwtDurationInMinutes),

signingCredentials: creds

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

**//LoginModel.cs**

namespace JwtAuthApi.Models

{

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**//TestController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthApi.Controllers

{

[ApiController]

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

[Authorize] // This secures the entire controller

public class TestController : ControllerBase

{

[HttpGet]

public IActionResult GetSecuredData()

{

var username = User.Identity.IsAuthenticated ? User.Identity.Name : "Anonymous";

return Ok($"Hello {username}! You have accessed secured data.");

}

[HttpGet("public")]

[AllowAnonymous]

public IActionResult GetPublicData()

{

return Ok("This is public data, accessible without a token.");

}

}

}

**//appsettings.json**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"Jwt": {

"Key": "ThisIsAVeryLongAndSecureSecretKeyForMyJwtTokenAuthenticationDemo",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

}

}