**WEEK 5**

**ASP.NET Core 8.0 Web API**

* Superset ID - 6362284
* Agrima Singh

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

**KafkaProducer:**

**Code – Program.cs**

using Confluent.Kafka;

var config = new ProducerConfig

{

BootstrapServers = "localhost:9092"

};

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

Console.WriteLine("Kafka Producer started. Type messages below. Type 'exit' to quit.\n");

while (true)

{

Console.Write("You: ");

var input = Console.ReadLine();

if (input?.ToLower() == "exit") break;

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

Console.WriteLine(" Message sent.");

}

**KafkaConsumer:**

**Code – Program.cs**

using Confluent.Kafka;

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("Kafka Consumer started. Waiting for messages...\n");

try

{

while (true)

{

var cr = consumer.Consume();

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

}

}

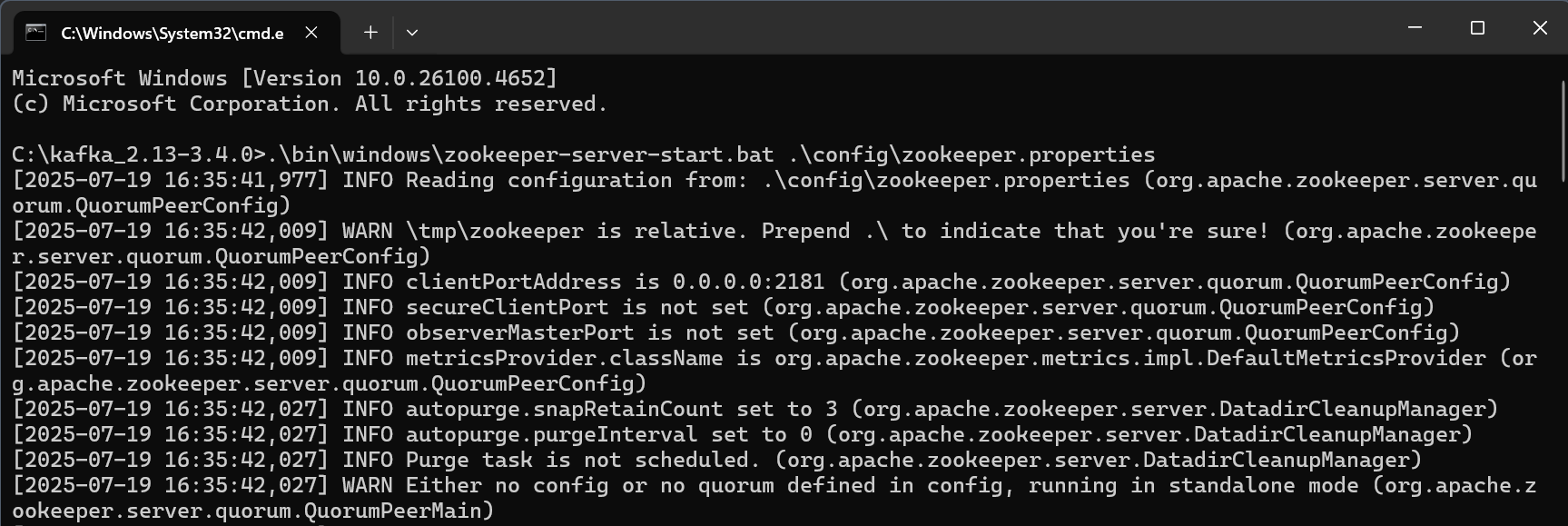
catch (OperationCanceledException)

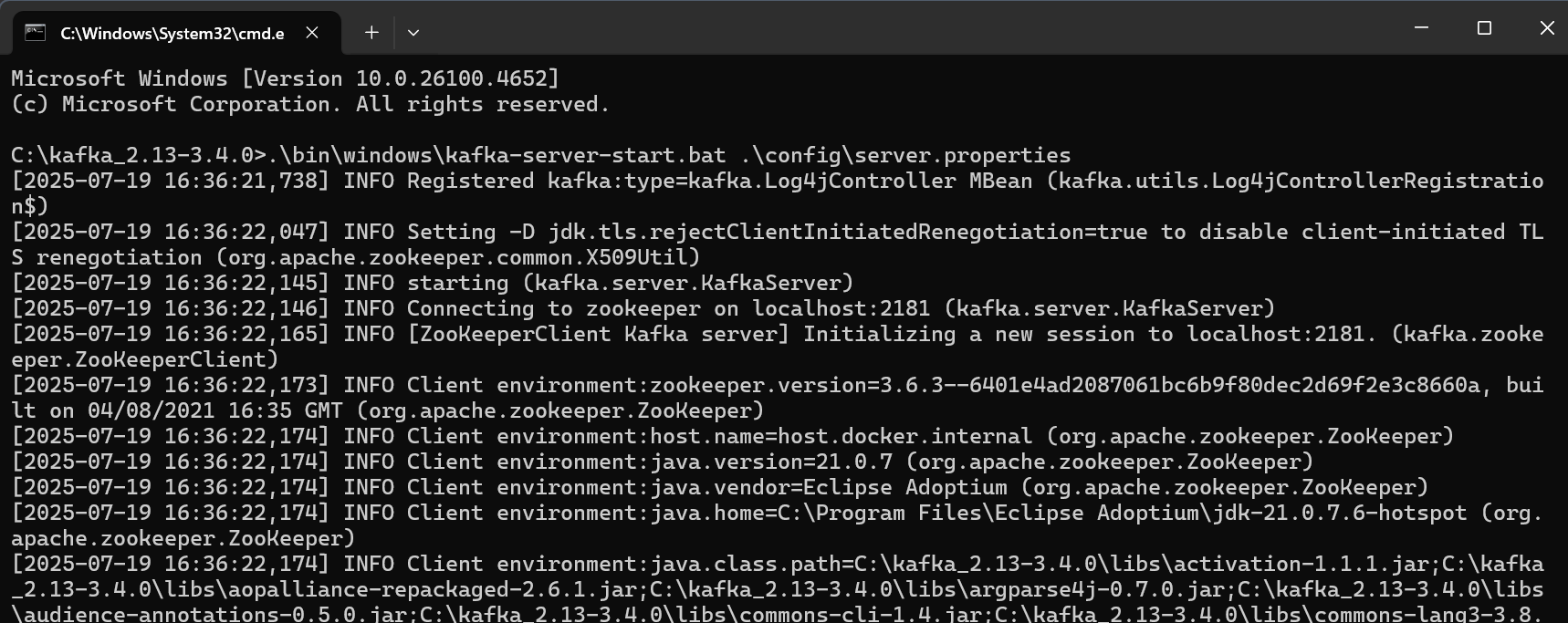
{

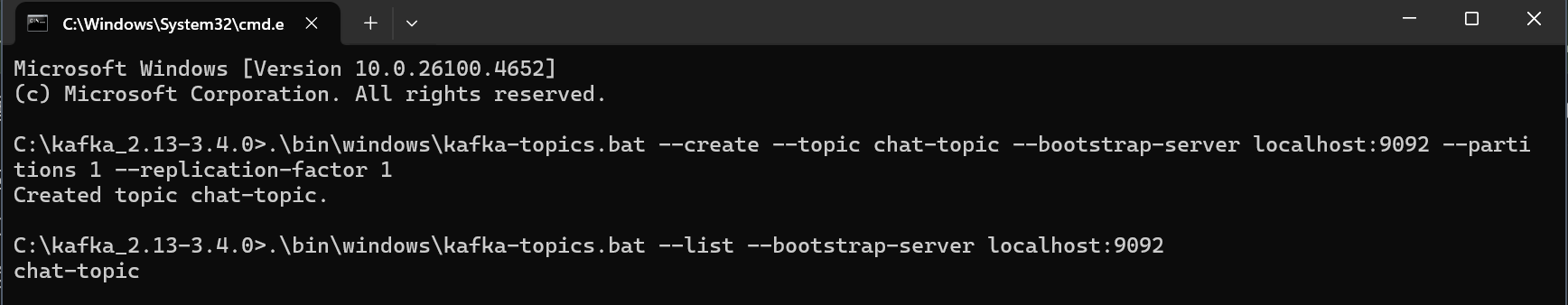
consumer.Close();

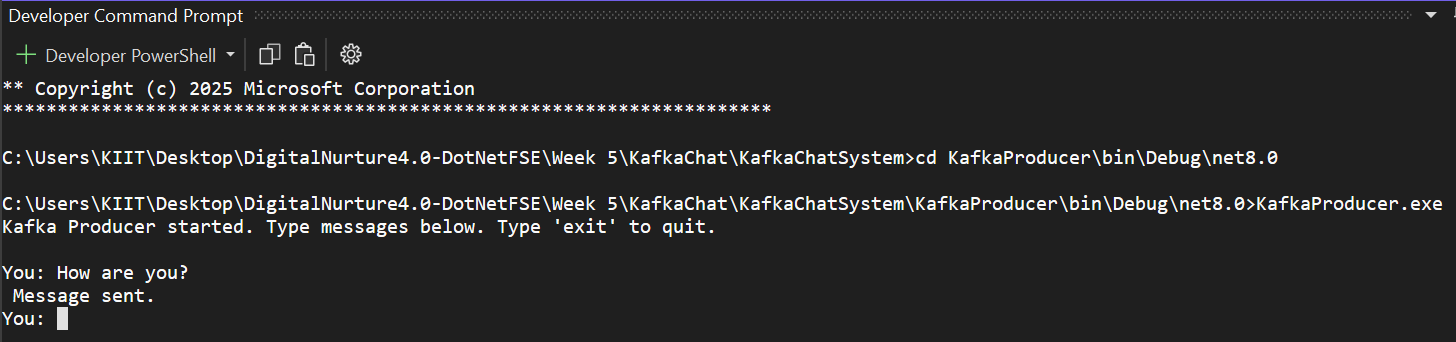
}

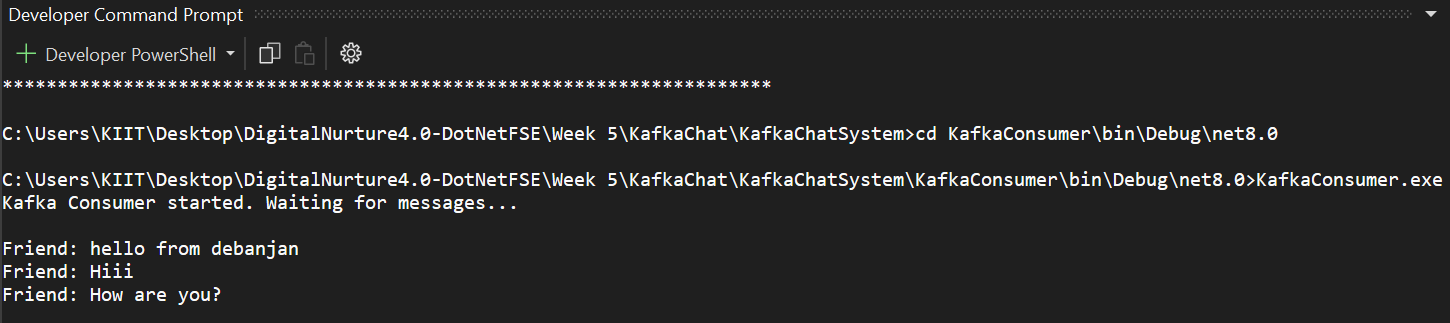
**Output –**











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

**Code –**

**Filename –Form1.cs**

using Confluent.Kafka;

using System;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace KafkaChatWinApp

{

public partial class Form1 : Form

{

private IProducer<Null, string>? producer;

private IConsumer<Null, string>? consumer;

private readonly string topic = "chat-topic";

private readonly string clientId = "User-" + Guid.NewGuid().ToString().Substring(0, 4);

private bool isRunning = true;

public Form1()

{

InitializeComponent();

this.Text = $"Kafka Chat - {clientId}";

InitializeKafka();

}

private void InitializeKafka()

{

try

{

var producerConfig = new ProducerConfig

{

BootstrapServers = "localhost:9092"

};

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

var consumerConfig = new ConsumerConfig

{

BootstrapServers = "localhost:9092",

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

AutoOffsetReset = AutoOffsetReset.Latest

};

consumer = new ConsumerBuilder<Null, string>(consumerConfig).Build();

consumer.Subscribe(topic);

Task.Run(() => ListenForMessages());

}

catch (Exception ex)

{

MessageBox.Show("Kafka initialization failed: " + ex.Message);

}

}

private void ListenForMessages()

{

try

{

while (isRunning)

{

try

{

var consumeResult = consumer?.Consume();

if (consumeResult?.Message?.Value != null)

{

AppendMessageToList(consumeResult.Message.Value);

}

}

catch (ConsumeException ex)

{

AppendMessageToList($"Kafka error: {ex.Error.Reason}");

}

}

}

catch (Exception ex)

{

AppendMessageToList("Error in consumer thread: " + ex.Message);

}

}

private void AppendMessageToList(string message)

{

if (lstMessages.InvokeRequired)

{

lstMessages.Invoke(new Action(() => lstMessages.Items.Add(message)));

}

else

{

lstMessages.Items.Add(message);

}

}

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

{

string userMessage = txtMessage.Text.Trim();

if (!string.IsNullOrEmpty(userMessage) && producer != null)

{

string fullMessage = $"{clientId}: {userMessage}";

try

{

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

txtMessage.Clear();

}

catch (ProduceException<Null, string> ex)

{

AppendMessageToList($"Failed to send: {ex.Message}");

}

}

}

protected override void OnFormClosing(FormClosingEventArgs e)

{

isRunning = false;

consumer?.Close();

consumer?.Dispose();

producer?.Flush(TimeSpan.FromSeconds(5));

producer?.Dispose();

base.OnFormClosing(e);

}

private void lstMessages\_SelectedIndexChanged(object sender, EventArgs e)

{

}

}

}

**Output –**

