**SUPERSET ID - 6364957**

**Kafka Integration with C#:**

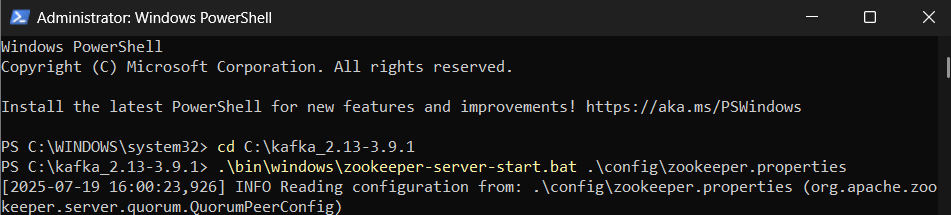
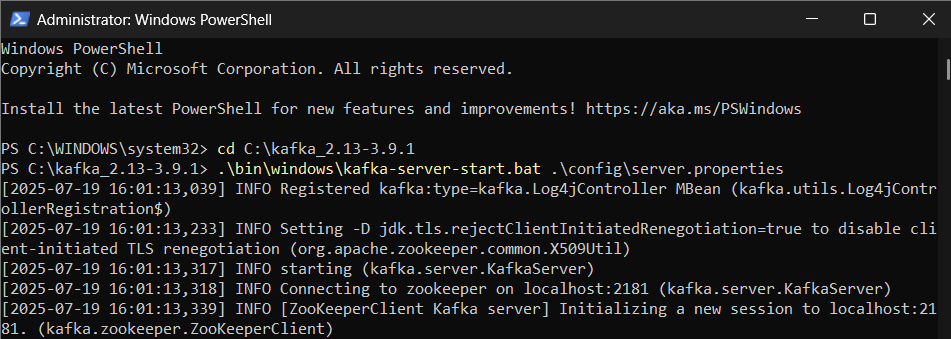
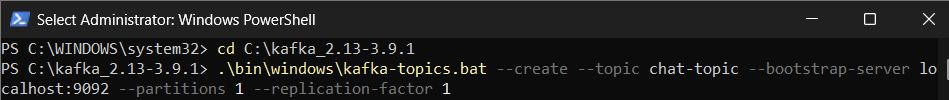
**Outline:**

* Introduction to Kafka
* Kafka Architecture
* Topics
* Partitions
* Brokers
* Kafka plug in .NET
* Installation of Kafka
* Basics of Zookeeper
* Demo

**Hands On:**

1. Create a Chat Application which uses Kafka as a streaming platform and consume the chat messages in the command prompt.
2. Create a Chat Application using C# Windows Application using Kafka and consume the message in different client applications.

**Kafka Setup for Services after Installation and Changing Required Config Contents with correct Path :**

* **Starting Zookeper in cmd Prompt:**
* Starting kafka Server in another cmd Prompt:  
  
* **Creating a Topic in another cmd Prompt:**
* Kafka Topic Creation:  
    
  kafka-topics.bat --create --topic chat-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

**HANDSON-1**

**Project Setup  
  
1. Creating New Console Project**

# Creating project directory  
mkdir KafkaChatApp  
cd KafkaChatApp  
  
# Initializing new console application  
dotnet new console  
  
# Adding Kafka client package  
dotnet add package Confluent.Kafka

**Implementation Files**

**File 1: Producer.cs**

* **Purpose:** Sends messages to a Kafka topic.
* **Key Concepts:** ProducerConfig, ProduceAsync, message serialization.

using Confluent.Kafka;

using System;

using System.Threading.Tasks;

namespace KafkaChatApp

{

public class ChatProducer

{

private readonly string \_bootstrapServers;

private readonly string \_topicName;

public ChatProducer(string bootstrapServers, string topicName)

{

\_bootstrapServers = bootstrapServers;

\_topicName = topicName;

}

public async Task StartProducing()

{

var config = new ProducerConfig

{

BootstrapServers = \_bootstrapServers,

ClientId = "chat-producer"

};

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

Console.WriteLine("Chat Producer Started. Type messages (type 'exit' to quit):");

Console.WriteLine("Format: username: message");

string input;

while ((input = Console.ReadLine()) != "exit")

{

if (string.IsNullOrWhiteSpace(input)) continue;

try

{

var message = new Message<string, string>

{

Key = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"),

Value = input

};

var result = await producer.ProduceAsync(\_topicName, message);

Console.WriteLine($"✓ Message sent: {result.TopicPartitionOffset}");

}

catch (ProduceException<string, string> ex)

{

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

}

}

}

}

}

**File 2: Consumer.cs**

* **Purpose:** Receives and processes messages from Kafka.
* **Key Concepts:** ConsumerConfig, Subscribe, Consume, offset management.

using Confluent.Kafka;

using System;

using System.Threading;

namespace KafkaChatApp

{

public class ChatConsumer

{

private readonly string \_bootstrapServers;

private readonly string \_topicName;

private readonly string \_groupId;

public ChatConsumer(string bootstrapServers, string topicName, string groupId)

{

\_bootstrapServers = bootstrapServers;

\_topicName = topicName;

\_groupId = groupId;

}

public void StartConsuming()

{

var config = new ConsumerConfig

{

BootstrapServers = \_bootstrapServers,

GroupId = \_groupId,

AutoOffsetReset = AutoOffsetReset.Earliest,

EnableAutoCommit = true,

ClientId = "chat-consumer"

};

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

consumer.Subscribe(\_topicName);

Console.WriteLine($"Chat Consumer Started. Listening to topic: {\_topicName}");

Console.WriteLine("Press Ctrl+C to stop...");

var cts = new CancellationTokenSource();

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

{

e.Cancel = true;

cts.Cancel();

};

try

{

while (!cts.Token.IsCancellationRequested)

{

try

{

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

if (result != null)

{

Console.WriteLine($"[{result.Message.Key}] {result.Message.Value}");

}

}

catch (ConsumeException ex)

{

Console.WriteLine($"✗ Consume error: {ex.Error.Reason}");

}

}

}

catch (OperationCanceledException)

{

Console.WriteLine("Shutting down consumer...");

}

finally

{

consumer.Close();

}

}

}

}  
}

**Key Components Explained**

* **ConsumerConfig:** Consumer settings
* **ConsumerBuilder:** Creates typed consumer
* **Subscribe:** Start listening for messages on topic
* **Consume:** Poll for messages

**File 3: Program.cs**

* **Purpose:** User interface and application entry point.
* **Key Concepts:** App flow, user input, dependency coordination.

using System;

using System.Threading.Tasks;

namespace KafkaChatApp

{

class Program

{

private const string BootstrapServers = "localhost:9092";

private const string TopicName = "chat-topic";

static async Task Main(string[] args)

{

Console.WriteLine("Kafka Chat Application");

Console.WriteLine("1. Producer (Send messages)");

Console.WriteLine("2. Consumer (Receive messages)");

Console.Write("Choose option (1 or 2): ");

var choice = Console.ReadLine();

switch (choice)

{

case "1":

await RunProducer();

break;

case "2":

RunConsumer();

break;

default:

Console.WriteLine("Invalid choice. Exiting...");

break;

}

}

private static async Task RunProducer()

{

var producer = new ChatProducer(BootstrapServers, TopicName);

await producer.StartProducing();

}

private static void RunConsumer()

{

var consumer = new ChatConsumer(BootstrapServers, TopicName, "chat-group");

consumer.StartConsuming();

}

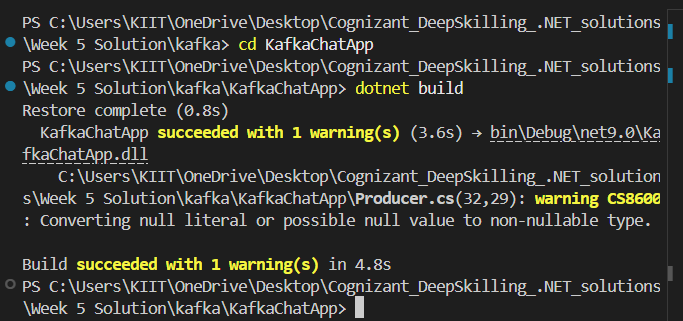
}

}

**Key Components Explained**

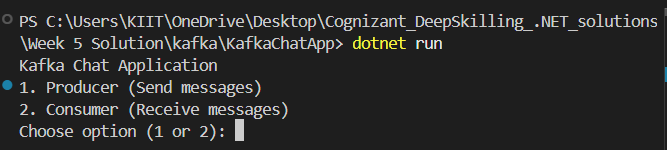
* **Main:** Entry point uses async/await
* **Switch:** User chooses producer/consumer mode
* **Method Separation:** Clear mode control

**Build and Compile**

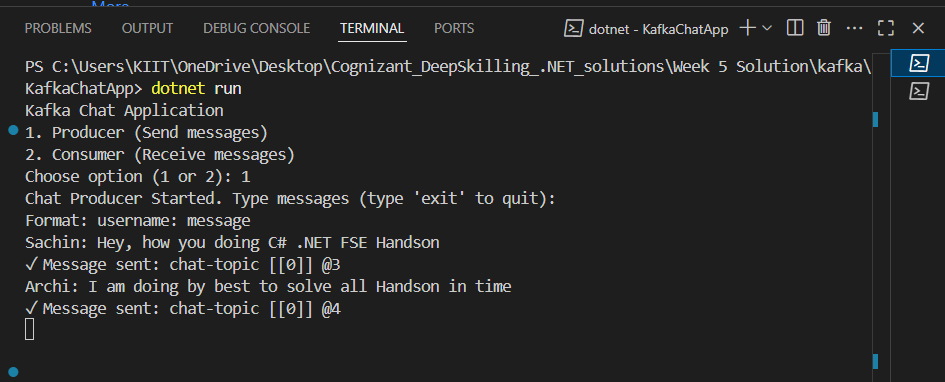


**Running the Application**

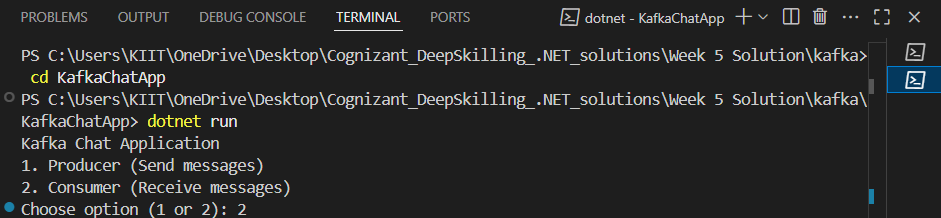
* **Terminal 1: Starting Producer**



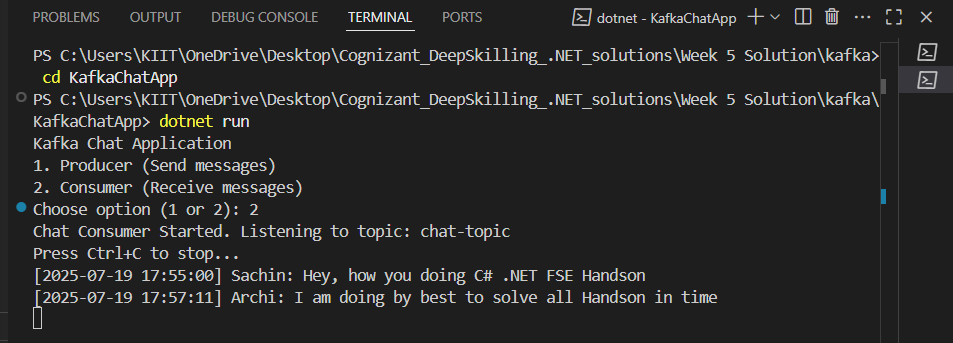
**Output**



* **Terminal 2: Starting Consumer**



**Output**



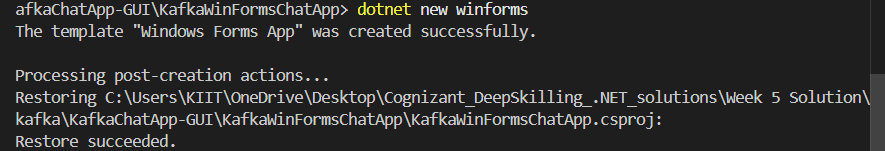
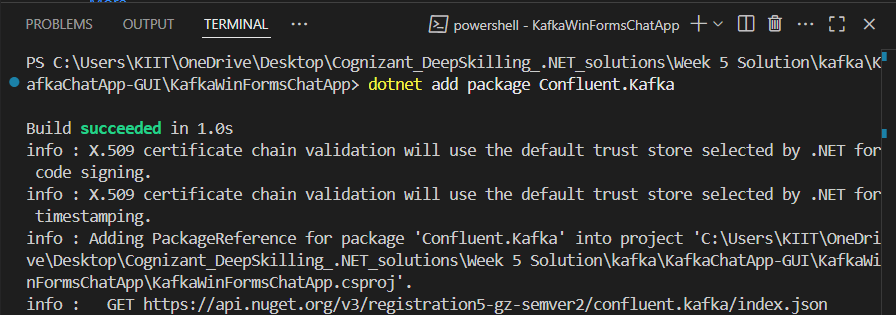
**Note:   
These commands must be running on in the separate Powershell Cmd Prompt**

* **PS C:\kafka\_2.13-3.9.1> .\bin\windows\zookeeper-server-start.bat config\zookeeper.properties**
* **PS C:\kafka\_2.13-3.9.1> .\bin\windows\kafka-server-start.bat .\config\server.properties**

**HANDSON-2:   
  
Kafka Windows Forms Chat Application GUI Based**

**1. Creating a New Project**

Opening a terminal and creating a new Windows Forms application separate from any Console apps:

# Navigating to development folder  
C:\Users\KIIT\OneDrive\Desktop\Cognizant\_DeepSkilling\_.NET\_solutions\Week 5 Solution\kafka\KafkaChatApp-GUI  
  
# Creating project directory  
mkdir KafkaWinFormsChatApp  
cd KafkaWinFormsChatApp  
  
# Initializing Windows Forms project  
dotnet new winforms  
  
  
# Adding Kafka client library  
dotnet add package Confluent.Kafka  


**2. Replacing Program.cs**

Updating Program.cs so the project launches the main chat form:

using System;  
using System.Windows.Forms;  
  
namespace KafkaWinFormsChatApp  
{  
 internal static class Program  
 {  
 [STAThread]  
 static void Main()  
 {  
 Application.EnableVisualStyles();  
 Application.SetCompatibleTextRenderingDefault(false);  
 Application.Run(new MainForm());  
 }  
 }  
}

**3. Adding MainForm.cs**

This file contains the chat interface, Kafka connection, and all messaging logic. Saving the following as MainForm.cs:

using System;

using System.Drawing;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using Confluent.Kafka;

namespace KafkaWinFormsChatApp

{

    public partial class MainForm : Form

    {

        private const string BootstrapServers = "localhost:9092";

        private const string TopicName = "chat-topic";

        // UI Controls

        private TextBox txtUsername;

        private TextBox txtMessage;

        private RichTextBox rtbChatHistory;

        private Button btnSend;

        private Button btnConnect;

        private Button btnDisconnect;

        private Label lblStatus;

        private Label lblUsername;

        private Label lblInstructions;

        // Kafka components

        private IProducer<string, string> \_producer;

        private IConsumer<string, string> \_consumer;

        private CancellationTokenSource \_cancellationTokenSource;

        private Task \_consumerTask;

        private bool \_isConnected = false;

        private string \_groupId;

        public MainForm()

        {

            InitializeComponent();

            \_groupId = $"chat-group-{Environment.MachineName}-{DateTime.Now.Ticks}";

        }

        private void InitializeComponent()

        {

            this.SuspendLayout();

            // Form properties

            this.Text = "Kafka Chat Application - Windows Forms";

            this.Size = new Size(700, 550);

            this.StartPosition = FormStartPosition.CenterScreen;

            this.MinimumSize = new Size(600, 400);

            // Instructions label

            lblInstructions = new Label

            {

                Text = "Enter your username and click Connect to join the chat",

                Location = new Point(10, 10),

                Size = new Size(400, 20),

                Font = new Font("Segoe UI", 9, FontStyle.Italic),

                ForeColor = Color.DarkBlue

            };

            this.Controls.Add(lblInstructions);

            // Username label

            lblUsername = new Label

            {

                Text = "Username:",

                Location = new Point(10, 40),

                Size = new Size(80, 23),

                Font = new Font("Segoe UI", 9, FontStyle.Bold)

            };

            this.Controls.Add(lblUsername);

            // Username textbox

            txtUsername = new TextBox

            {

                Location = new Point(100, 37),

                Size = new Size(150, 23),

                Text = Environment.UserName,

                Font = new Font("Segoe UI", 9)

            };

            this.Controls.Add(txtUsername);

            // Connect button

            btnConnect = new Button

            {

                Text = "Connect",

                Location = new Point(260, 36),

                Size = new Size(80, 25),

                Font = new Font("Segoe UI", 9),

                BackColor = Color.LightGreen

            };

            btnConnect.Click += BtnConnect\_Click;

            this.Controls.Add(btnConnect);

            // Disconnect button

            btnDisconnect = new Button

            {

                Text = "Disconnect",

                Location = new Point(350, 36),

                Size = new Size(80, 25),

                Enabled = false,

                Font = new Font("Segoe UI", 9),

                BackColor = Color.LightCoral

            };

            btnDisconnect.Click += BtnDisconnect\_Click;

            this.Controls.Add(btnDisconnect);

            // Status label

            lblStatus = new Label

            {

                Text = "Status: Disconnected",

                Location = new Point(440, 40),

                Size = new Size(200, 23),

                ForeColor = Color.Red,

                Font = new Font("Segoe UI", 9, FontStyle.Bold)

            };

            this.Controls.Add(lblStatus);

            // Chat history

            rtbChatHistory = new RichTextBox

            {

                Location = new Point(10, 70),

                Size = new Size(660, 350),

                ReadOnly = true,

                BackColor = Color.White,

                ScrollBars = RichTextBoxScrollBars.Vertical,

                Font = new Font("Consolas", 9),

                BorderStyle = BorderStyle.Fixed3D

            };

            this.Controls.Add(rtbChatHistory);

            // Message input label

            var lblMessage = new Label

            {

                Text = "Message:",

                Location = new Point(10, 435),

                Size = new Size(60, 23),

                Font = new Font("Segoe UI", 9, FontStyle.Bold)

            };

            this.Controls.Add(lblMessage);

            // Message input textbox

            txtMessage = new TextBox

            {

                Location = new Point(80, 432),

                Size = new Size(500, 23),

                Enabled = false,

                Font = new Font("Segoe UI", 9)

            };

            txtMessage.KeyPress += TxtMessage\_KeyPress;

            this.Controls.Add(txtMessage);

            // Send button

            btnSend = new Button

            {

                Text = "Send",

                Location = new Point(590, 431),

                Size = new Size(80, 25),

                Enabled = false,

                Font = new Font("Segoe UI", 9),

                BackColor = Color.LightBlue

            };

            btnSend.Click += BtnSend\_Click;

            this.Controls.Add(btnSend);

            // Form event handlers

            this.FormClosing += MainForm\_FormClosing;

            this.Load += MainForm\_Load;

            this.ResumeLayout();

        }

        private void MainForm\_Load(object sender, EventArgs e)

        {

            AppendToChatHistory("System", "Welcome to Kafka Chat! Enter your username and click Connect.", Color.Blue);

            txtUsername.Focus();

        }

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

        {

            if (string.IsNullOrWhiteSpace(txtUsername.Text))

            {

                MessageBox.Show("Please enter a username", "Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);

                txtUsername.Focus();

                return;

            }

            try

            {

                btnConnect.Enabled = false;

                btnConnect.Text = "Connecting...";

                await ConnectToKafka();

                UpdateUIConnectionState(true);

                AppendToChatHistory("System", $"Connected to chat as '{txtUsername.Text}'", Color.Green);

                txtMessage.Focus();

            }

            catch (Exception ex)

            {

                MessageBox.Show($"Failed to connect to Kafka:\n{ex.Message}", "Connection Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

                btnConnect.Enabled = true;

                btnConnect.Text = "Connect";

            }

        }

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

        {

            btnDisconnect.Enabled = false;

            btnDisconnect.Text = "Disconnecting...";

            await DisconnectFromKafka();

            UpdateUIConnectionState(false);

            AppendToChatHistory("System", "Disconnected from chat", Color.Red);

            btnDisconnect.Text = "Disconnect";

        }

        private async Task ConnectToKafka()

        {

            // Initialize producer

            var producerConfig = new ProducerConfig

            {

                BootstrapServers = BootstrapServers,

                ClientId = $"chat-producer-{txtUsername.Text}",

                MessageTimeoutMs = 10000

            };

            \_producer = new ProducerBuilder<string, string>(producerConfig).Build();

            // Initialize consumer

            var consumerConfig = new ConsumerConfig

            {

                BootstrapServers = BootstrapServers,

                GroupId = \_groupId,

                AutoOffsetReset = AutoOffsetReset.Latest,

                EnableAutoCommit = true,

                ClientId = $"chat-consumer-{txtUsername.Text}",

                SessionTimeoutMs = 10000

            };

            \_consumer = new ConsumerBuilder<string, string>(consumerConfig).Build();

            \_consumer.Subscribe(TopicName);

            // Start consuming messages

            \_cancellationTokenSource = new CancellationTokenSource();

            \_consumerTask = Task.Run(() => ConsumeMessages(\_cancellationTokenSource.Token));

            \_isConnected = true;

        }

        private async Task DisconnectFromKafka()

        {

            \_isConnected = false;

            if (\_cancellationTokenSource != null)

            {

                \_cancellationTokenSource.Cancel();

                if (\_consumerTask != null)

                {

                    try

                    {

                        await \_consumerTask;

                    }

                    catch (OperationCanceledException)

                    {

                        // Expected

                    }

                }

            }

            \_consumer?.Close();

            \_consumer?.Dispose();

            \_producer?.Dispose();

        }

        private void ConsumeMessages(CancellationToken cancellationToken)

        {

            try

            {

                while (!cancellationToken.IsCancellationRequested)

                {

                    try

                    {

                        var result = \_consumer.Consume(TimeSpan.FromMilliseconds(1000));

                        if (result != null && !result.IsPartitionEOF)

                        {

                            var parts = result.Message.Value.Split(new[] { ": " }, 2, StringSplitOptions.None);

                            if (parts.Length == 2)

                            {

                                var username = parts[0];

                                var message = parts[1];

                                // Don't display our own messages (they're already shown when sent)

                                if (username != txtUsername.Text)

                                {

                                    this.Invoke(new Action(() =>

                                    {

                                        AppendToChatHistory(username, message, Color.Blue);

                                    }));

                                }

                            }

                        }

                    }

                    catch (ConsumeException ex)

                    {

                        if (!cancellationToken.IsCancellationRequested)

                        {

                            this.Invoke(new Action(() =>

                            {

                                AppendToChatHistory("System", $"Error receiving message: {ex.Error.Reason}", Color.Red);

                            }));

                        }

                    }

                }

            }

            catch (OperationCanceledException)

            {

                // Expected when cancellation is requested

            }

        }

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

        {

            await SendMessage();

        }

        private void TxtMessage\_KeyPress(object sender, KeyPressEventArgs e)

        {

            if (e.KeyChar == (char)Keys.Enter)

            {

                e.Handled = true;

                Task.Run(async () => await SendMessage());

            }

        }

        private async Task SendMessage()

        {

            if (string.IsNullOrWhiteSpace(txtMessage.Text) || !\_isConnected)

                return;

            var messageText = txtMessage.Text;

            try

            {

                var kafkaMessage = new Message<string, string>

                {

                    Key = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"),

                    Value = $"{txtUsername.Text}: {messageText}"

                };

                await \_producer.ProduceAsync(TopicName, kafkaMessage);

                // Display our own message immediately

                this.Invoke(new Action(() =>

                {

                    AppendToChatHistory(txtUsername.Text, messageText, Color.DarkGreen);

                    txtMessage.Clear();

                }));

            }

            catch (ProduceException<string, string> ex)

            {

                this.Invoke(new Action(() =>

                {

                    AppendToChatHistory("System", $"Failed to send message: {ex.Error.Reason}", Color.Red);

                }));

            }

        }

        private void AppendToChatHistory(string username, string message, Color color)

        {

            var timestamp = DateTime.Now.ToString("HH:mm:ss");

            // Move to end

            rtbChatHistory.SelectionStart = rtbChatHistory.TextLength;

            rtbChatHistory.SelectionLength = 0;

            // Add timestamp

            rtbChatHistory.SelectionColor = Color.Gray;

            rtbChatHistory.AppendText($"[{timestamp}] ");

            // Add username

            rtbChatHistory.SelectionColor = color;

            rtbChatHistory.SelectionFont = new Font(rtbChatHistory.Font, FontStyle.Bold);

            rtbChatHistory.AppendText($"{username}: ");

            // Add message

            rtbChatHistory.SelectionColor = Color.Black;

            rtbChatHistory.SelectionFont = new Font(rtbChatHistory.Font, FontStyle.Regular);

            rtbChatHistory.AppendText($"{message}\n");

            // Reset formatting

            rtbChatHistory.SelectionColor = rtbChatHistory.ForeColor;

            rtbChatHistory.SelectionFont = rtbChatHistory.Font;

            // Auto-scroll

            rtbChatHistory.ScrollToCaret();

        }

        private void UpdateUIConnectionState(bool connected)

        {

            \_isConnected = connected;

            btnConnect.Enabled = !connected;

            btnConnect.Text = "Connect";

            btnDisconnect.Enabled = connected;

            txtMessage.Enabled = connected;

            btnSend.Enabled = connected;

            txtUsername.Enabled = !connected;

            lblStatus.Text = connected ? "Status: Connected" : "Status: Disconnected";

            lblStatus.ForeColor = connected ? Color.Green : Color.Red;

            if (connected)

            {

                lblInstructions.Text = $"Connected as '{txtUsername.Text}' - Type messages below";

                lblInstructions.ForeColor = Color.DarkGreen;

            }

            else

            {

                lblInstructions.Text = "Enter your username and click Connect to join the chat";

                lblInstructions.ForeColor = Color.DarkBlue;

            }

        }

        private async void MainForm\_FormClosing(object sender, FormClosingEventArgs e)

        {

            if (\_isConnected)

            {

                await DisconnectFromKafka();

            }

        }

    }

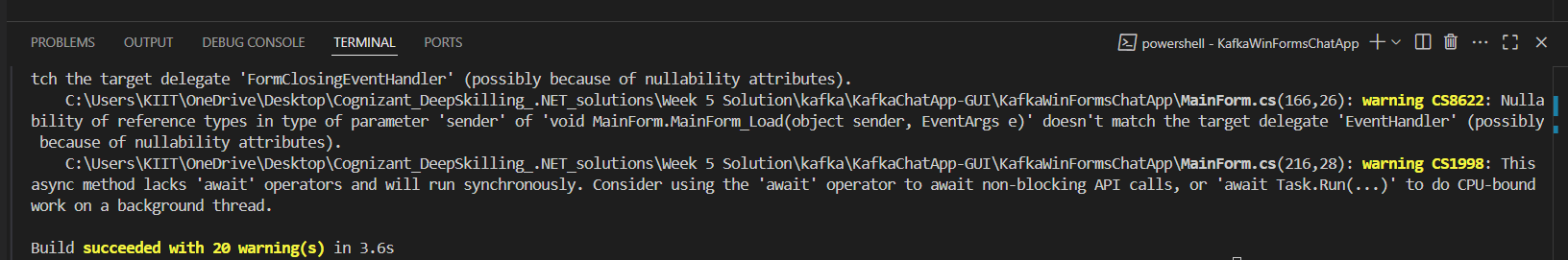
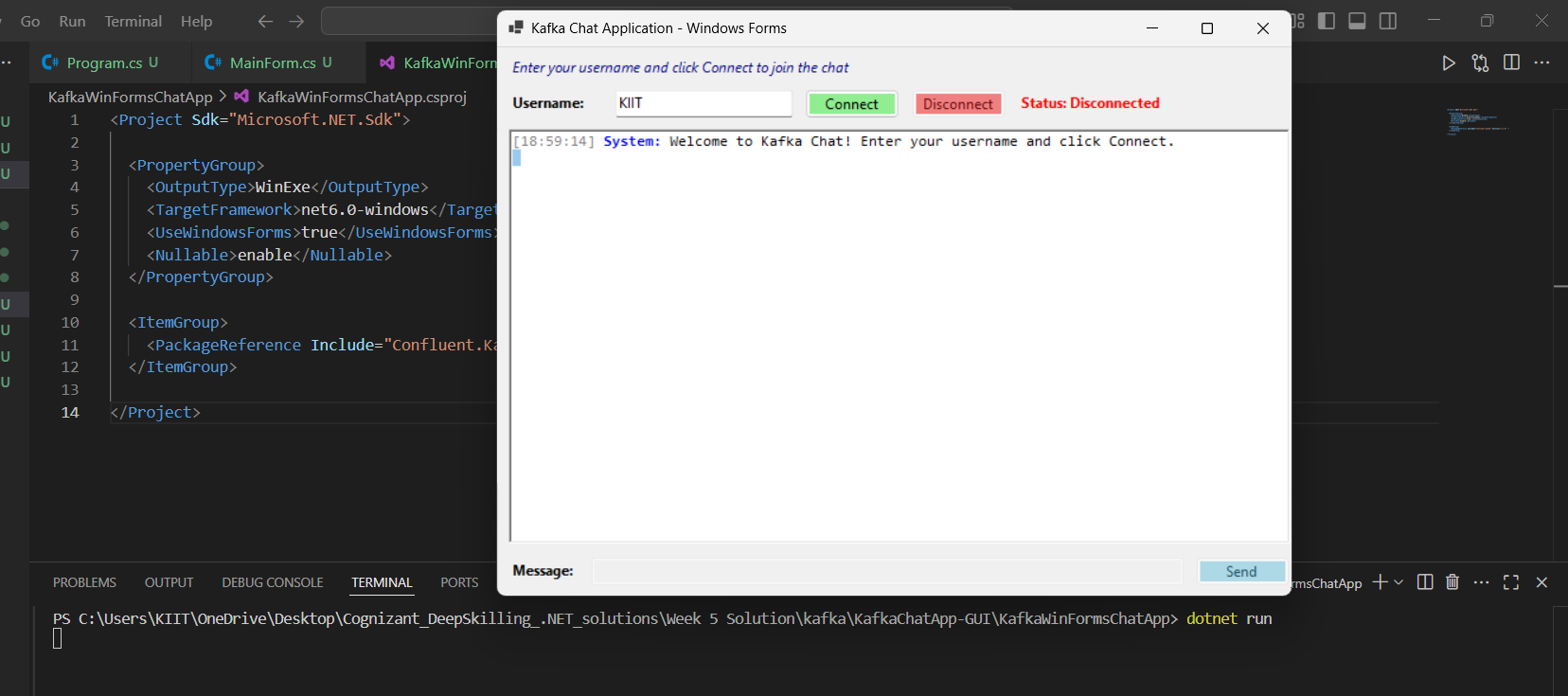
}

**4. Updating Project File**

Replacing KafkaWinFormsChatApp.csproj with:

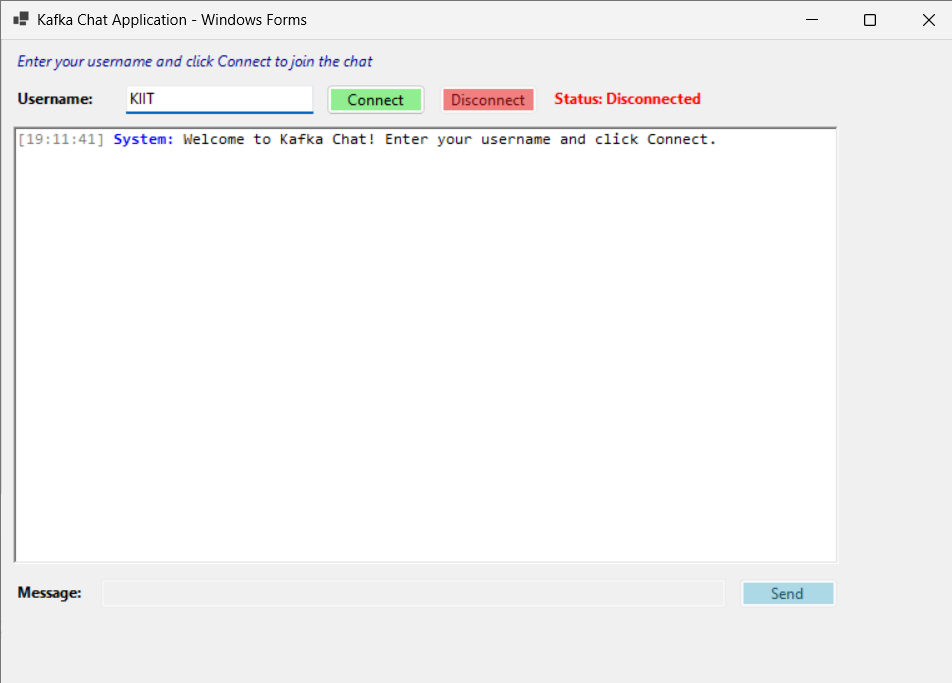
<Project Sdk="Microsoft.NET.Sdk">  
 <PropertyGroup>  
 <OutputType>WinExe</OutputType>  
 <TargetFramework>net6.0-windows</TargetFramework>  
 <UseWindowsForms>true</UseWindowsForms>  
 <Nullable>enable</Nullable>  
 </PropertyGroup>  
  
 <ItemGroup>  
 <PackageReference Include="Confluent.Kafka" Version="2.3.0" />  
 </ItemGroup>  
</Project>

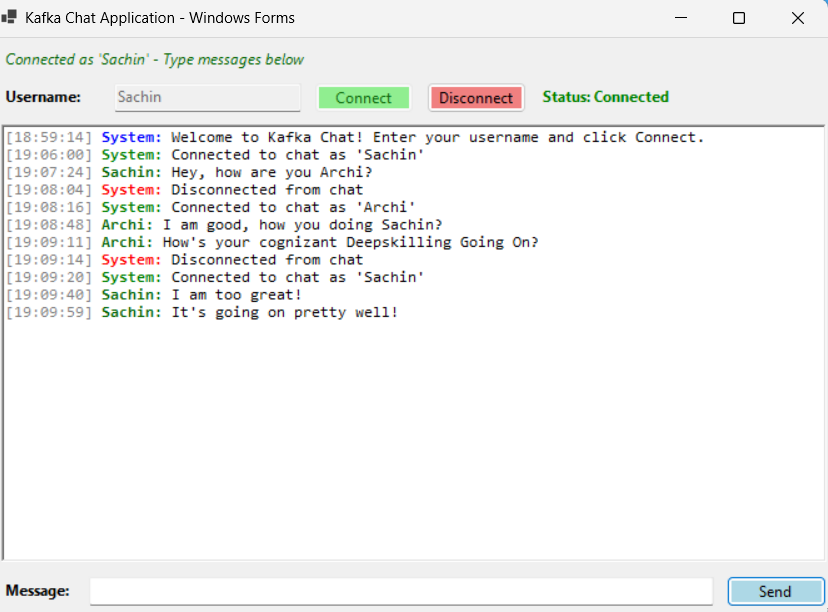
**5. Building and Running the Application**

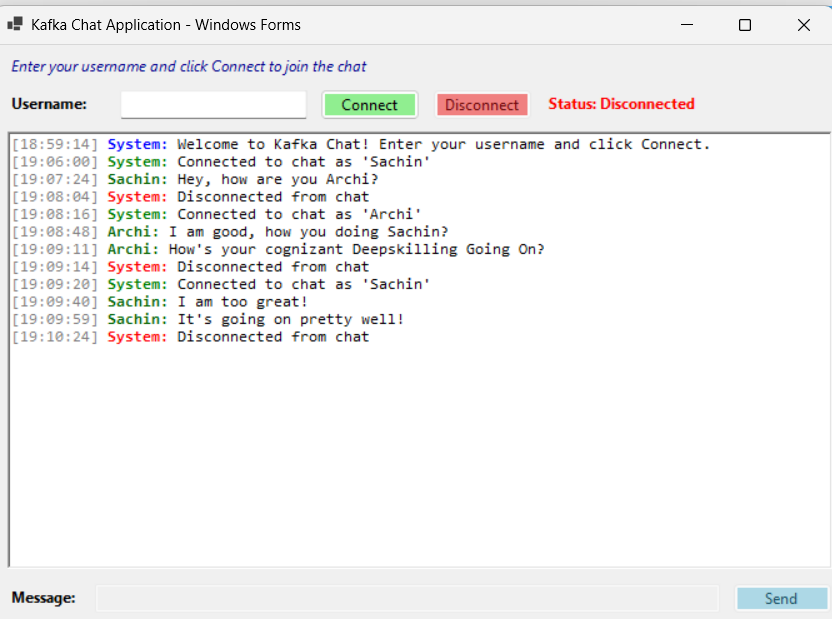
dotnet build  
  
dotnet run  


Output:

**Application Starting:**



**When Connected and Chatting:**

  
  
Chatting using two different Terminal (**Multi-Client Real-Time Chat**):   
