# Lab 6 - Kafka Chat GUI Application with Avalonia and C#

## Objective

In this extended part of Lab 6, I developed a cross-platform GUI-based chat application using Avalonia UI and C#. The app integrates Apache Kafka for real-time message streaming, allowing users to send and receive chat messages across multiple running instances.

## Prerequisites

• .NET 6 or newer installed

• Avalonia UI templates installed via: dotnet new install Avalonia.Templates

• Apache Kafka and Zookeeper running on localhost

• Kafka topic 'chat-topic' created

• Visual Studio Code (or terminal-based editor)

## Setup Steps

1. Start Zookeeper:

.\zookeeper-server-start.bat ..\..\config\zookeeper.properties

2. Start Kafka server:

.\kafka-server-start.bat ..\..\config\server.properties

3. Create topic (if not already created):

.\kafka-topics.bat --create --topic chat-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

## Avalonia GUI Application Structure

KafkaAvaloniaChatApp/

├── MainWindow.axaml

├── MainWindow.axaml.cs

├── Program.cs

└── Dependencies (Confluent.Kafka, Avalonia UI packages)

## MainWindow.axaml

Defines the user interface layout with a ListBox to display messages, a TextBox to type new messages, and a Send Button.

<Window xmlns="https://github.com/avaloniaui"

        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

        x:Class="KafkaAvaloniaChatApp.MainWindow"

        Title="Kafka Chat" Width="400" Height="500">

    <StackPanel Margin="10">

        <ListBox x:Name="ChatList" Height="350"/>

        <TextBox x:Name="InputBox" Watermark="Type your message..." />

        <Button Content="Send" Click="Send\_Click" Margin="0,10,0,0"/>

    </StackPanel>

</Window>

## MainWindow.axaml.cs

Handles Kafka producer and consumer logic. It sends messages to the Kafka topic when the Send button is clicked, and receives messages from the topic to update the chat window.

using Avalonia.Controls;

using Avalonia.Threading;

using Confluent.Kafka;

using System;

using System.Collections.ObjectModel;

using System.Threading.Tasks;

namespace KafkaAvaloniaChatApp;

public partial class MainWindow : Window

{

    private string bootstrapServers = "localhost:9092";

    private string topic = "chat-topic";

    private ObservableCollection<string> messages = new ObservableCollection<string>();

    public MainWindow()

    {

        InitializeComponent();

        ChatList.ItemsSource = messages;

        StartConsumer();

    }

    private async void Send\_Click(object? sender, Avalonia.Interactivity.RoutedEventArgs e)

    {

        string message = InputBox.Text.Trim();

        if (!string.IsNullOrWhiteSpace(message))

        {

            try

            {

                var config = new ProducerConfig { BootstrapServers = bootstrapServers };

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

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

                await Dispatcher.UIThread.InvokeAsync(() =>

                {

                    messages.Add("Me: " + message);

                    InputBox.Text = "";

                });

                Console.WriteLine("Sent: " + message);

            }

            catch (Exception ex)

            {

                Console.WriteLine("Producer error: " + ex.Message);

            }

        }

    }

    private void StartConsumer()

    {

        Task.Run(() =>

        {

            var config = new ConsumerConfig

            {

                BootstrapServers = bootstrapServers,

                GroupId = $"chat-client-{Guid.NewGuid()}",

                AutoOffsetReset = AutoOffsetReset.Earliest

            };

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

            consumer.Subscribe(topic);

            Console.WriteLine("Consumer started...");

            try

            {

                while (true)

                {

                    var cr = consumer.Consume();

                    Console.WriteLine("Received: " + cr.Message.Value);

                    Dispatcher.UIThread.InvokeAsync(() =>

                    {

                        messages.Add("Friend: " + cr.Message.Value);

                    });

                }

            }

            catch (Exception ex)

            {

                Console.WriteLine("Consumer error: " + ex.Message);

            }

        });

    }

}

## Program.cs

Standard Avalonia application startup file.

using Avalonia;

using System;

namespace KafkaAvaloniaChatApp;

class Program

{

    // Initialization code. Don't use any Avalonia, third-party APIs or any

    // SynchronizationContext-reliant code before AppMain is called: things aren't initialized

    // yet and stuff might break.

    [STAThread]

    public static void Main(string[] args) => BuildAvaloniaApp()

        .StartWithClassicDesktopLifetime(args);

    // Avalonia configuration, don't remove; also used by visual designer.

    public static AppBuilder BuildAvaloniaApp()

        => AppBuilder.Configure<App>()

            .UsePlatformDetect()

            .WithInterFont()

            .LogToTrace();

}

## Running the Application

Use the following command in terminal:

dotnet run

Open two terminal windows and run the app in each to simulate two clients.

## Conclusion

This lab extension taught me how to create cross-platform desktop applications with Avalonia UI and integrate Kafka messaging to create a real-time chat system. By combining GUI development with message streaming, I gained valuable experience in asynchronous programming, UI threading, and distributed communication.

## Screenshots



