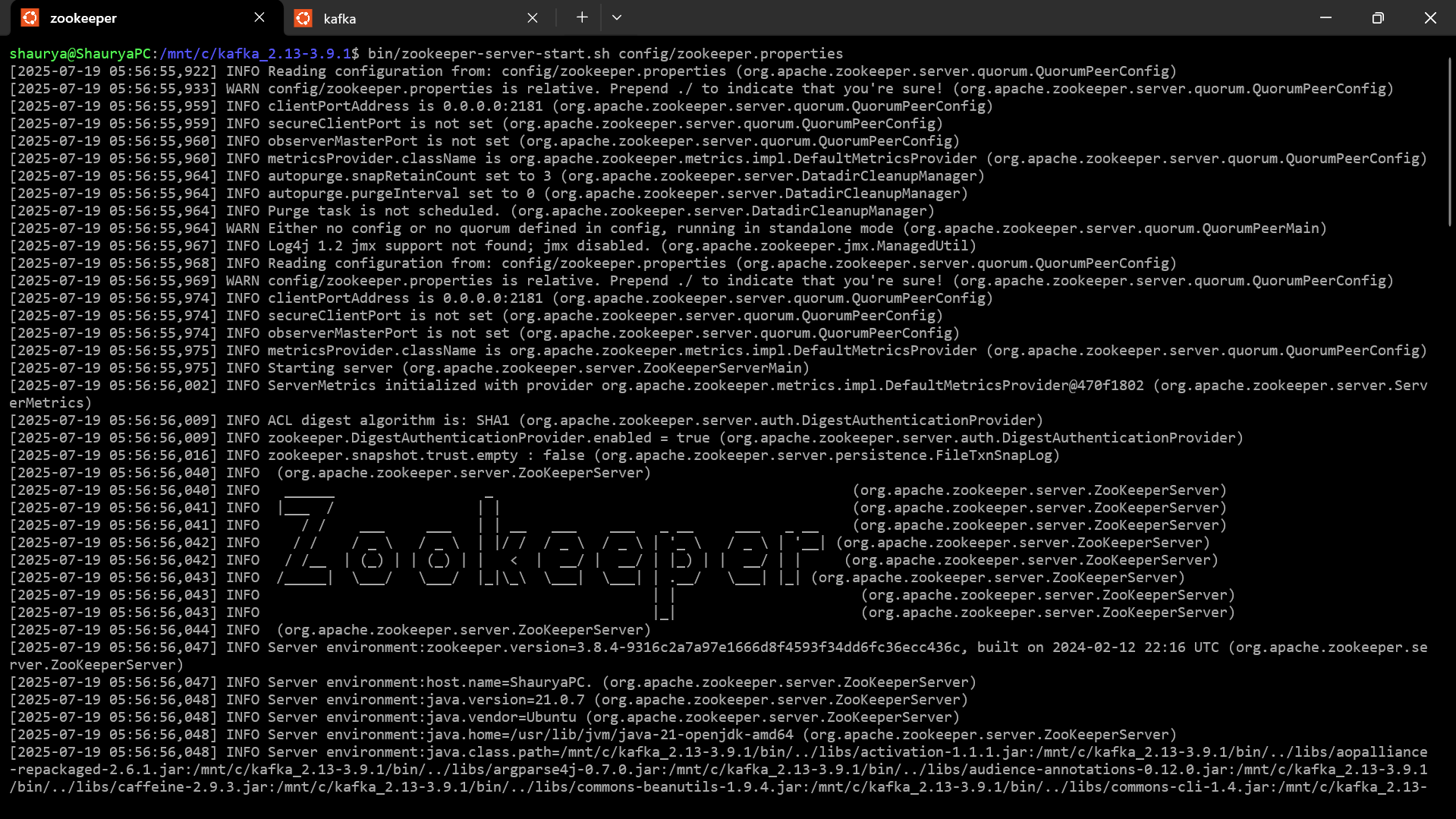
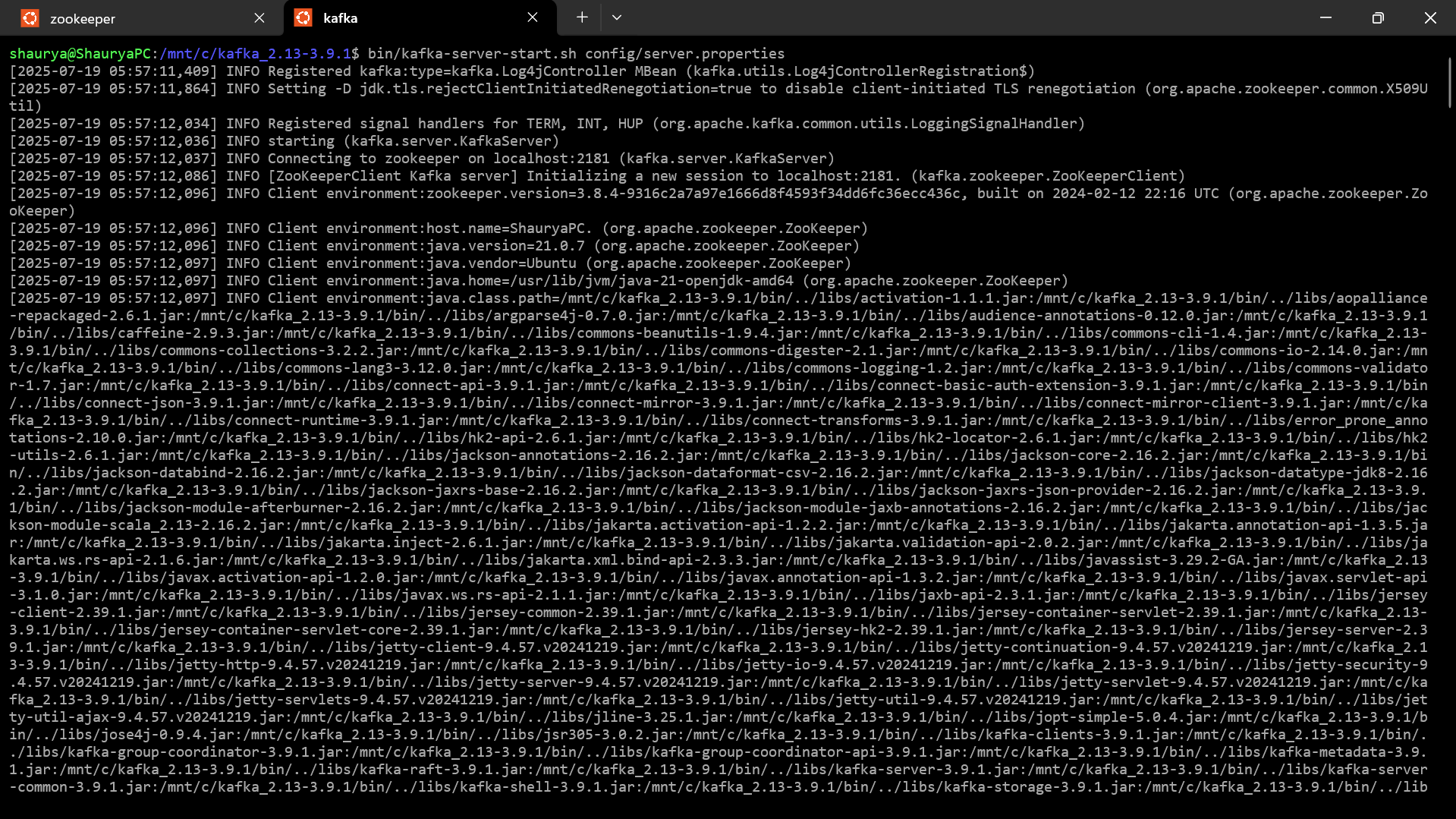
# Week 5 Microservices HandsOn

## Kafka Integration with C#

### Started the Zookeeper and Kafka in the background



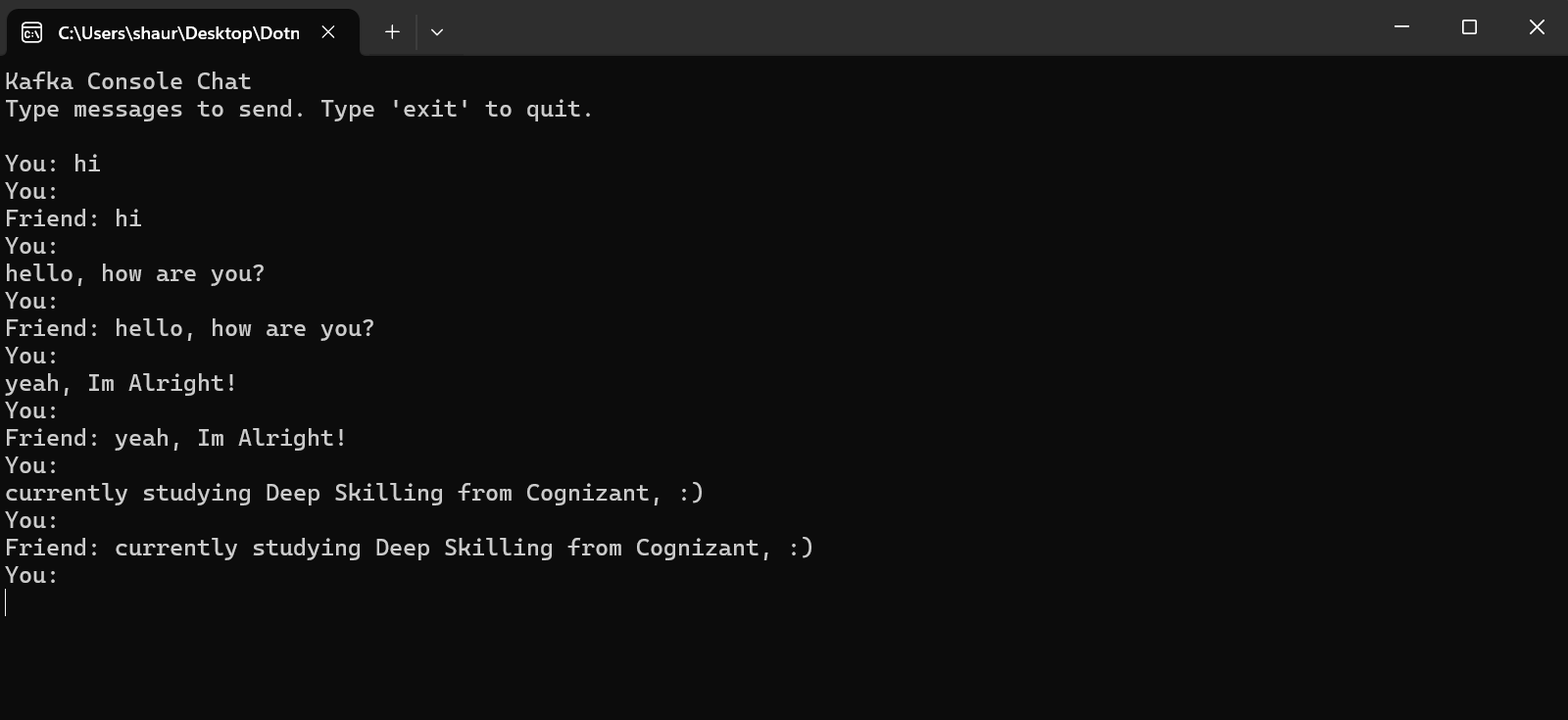


### Kafka Console Chat-App:

*Code: (Program.cs)*

|  |
| --- |
| using Confluent.Kafka;  using System;  using System.Threading;  using System.Threading.Tasks;    class Program  {  private static readonly string bootstrapServers = "192.168.187.8:9092";  private static readonly string topic = "chat-app";  private static CancellationTokenSource cts = new();    static async Task Main(string[] args)  {  Console.WriteLine("Kafka Console Chat");  Console.WriteLine("Type messages to send. Type 'exit' to quit.\n");    Task consumerTask = Task.Run(() => StartConsumer(cts.Token));    var producerConfig = new ProducerConfig  {  BootstrapServers = bootstrapServers  };    using var producer = new ProducerBuilder<Null, string>(producerConfig).Build();    while (true)  {  Console.Write("You: ");  var message = Console.ReadLine();    if (message?.ToLower() == "exit")  {  break;  }    await producer.ProduceAsync(topic, new Message<Null, string> { Value = message });  }    cts.Cancel();  Console.WriteLine("Exited chat.");  }    private static void StartConsumer(CancellationToken token)  {  var consumerConfig = new ConsumerConfig  {  BootstrapServers = bootstrapServers,  GroupId = Guid.NewGuid().ToString(),  AutoOffsetReset = AutoOffsetReset.Earliest  };    using var consumer = new ConsumerBuilder<Ignore, string>(consumerConfig).Build();  consumer.Subscribe(topic);    try  {  while (!token.IsCancellationRequested)  {  var cr = consumer.Consume(token);  Console.WriteLine($"\nFriend: {cr.Message.Value}\nYou: ");  }  }  catch (OperationCanceledException)  {  consumer.Close();  }  }  } |

*Output:*

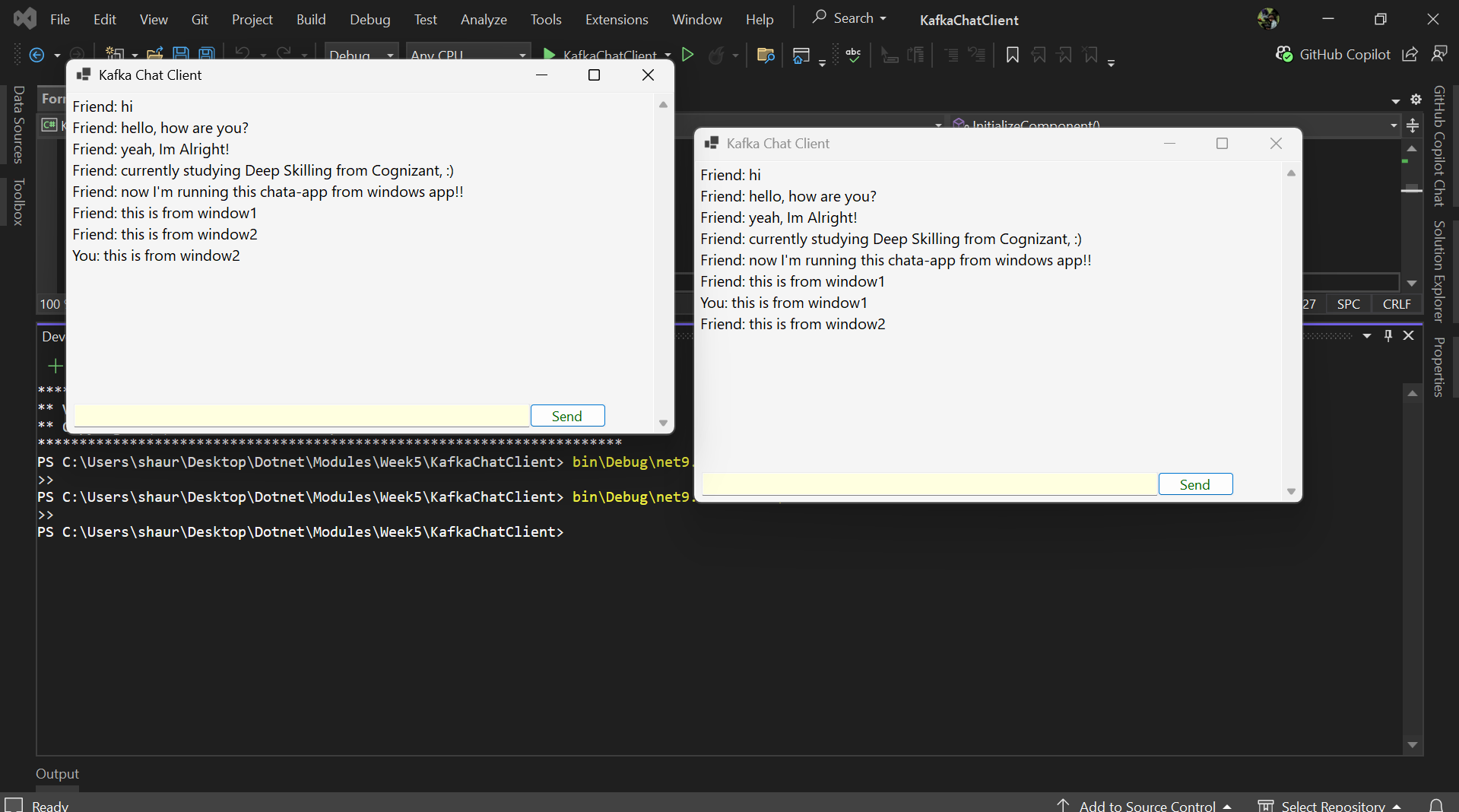


### Kafka Windows Application Chat-App:

*Code: (Form1.cs)*

|  |
| --- |
| using Confluent.Kafka;  using System;  using System.Threading;  using System.Threading.Tasks;  using System.Windows.Forms;    namespace KafkaChatClient  {  public partial class Form1 : Form  {  private CancellationTokenSource \_cts = new();    public Form1()  {  InitializeComponent();  StartKafkaConsumer();  }    private async void btnSend\_Click(object sender, EventArgs e)  {  var message = txtInput.Text.Trim();    if (string.IsNullOrEmpty(message))  return;    var config = new ProducerConfig  {  BootstrapServers = "192.168.187.8:9092"  };    using var producer = new ProducerBuilder<Null, string>(config).Build();    try  {  var result = await producer.ProduceAsync("chat-app", new Message<Null, string> { Value = message });  txtMessages.AppendText($"You: {message}{Environment.NewLine}");  txtInput.Clear();  }  catch (Exception ex)  {  MessageBox.Show($"Error sending message: {ex.Message}");  }  }    private async void StartKafkaConsumer()  {  var config = new ConsumerConfig  {  BootstrapServers = "192.168.187.8:9092",  GroupId = Guid.NewGuid().ToString(),  AutoOffsetReset = AutoOffsetReset.Earliest  };    using var consumer = new ConsumerBuilder<Ignore, string>(config).Build();  consumer.Subscribe("chat-app");    await Task.Run(() =>  {  try  {  while (!\_cts.Token.IsCancellationRequested)  {  var cr = consumer.Consume(\_cts.Token);  AppendTextToChat(cr.Message.Value);  }  }  catch (OperationCanceledException)  {  consumer.Close();  }  });  }    private void AppendTextToChat(string message)  {  if (txtMessages.InvokeRequired)  {  txtMessages.Invoke(() =>  txtMessages.AppendText($"Friend: {message}{Environment.NewLine}")  );  }  else  {  txtMessages.AppendText($"Friend: {message}{Environment.NewLine}");  }  }    protected override void OnFormClosing(FormClosingEventArgs e)  {  \_cts.Cancel();  base.OnFormClosing(e);  }  }  } |

*Output:*



## Implementing JWT Authentication in ASP.NET Core Web API

*Code:*

*LoginModel.cs*

|  |
| --- |
| namespace JwtAuthMicroservice.Models  {  public class LoginModel  {  public string username { get; set; }  public string password { get; set; }  }  } |

*AuthController.cs*

|  |
| --- |
| using JwtAuthMicroservice.Models;  using Microsoft.AspNetCore.Mvc;  using Microsoft.IdentityModel.Tokens;  using System.IdentityModel.Tokens.Jwt;  using System.Security.Claims;  using System.Text;    namespace JwtAuthMicroservice.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();  }    private bool IsValidUser(LoginModel model)  {  return model.username == "user" && model.password == "password";  }    private string GenerateJwtToken(string username)  {  var claims = new[]  {  new Claim(ClaimTypes.Name, username)  };    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]));  var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);    var token = new JwtSecurityToken(  issuer: \_configuration["Jwt:Issuer"],  audience: \_configuration["Jwt:Audience"],  claims: claims,  expires: DateTime.Now.AddMinutes(60),  signingCredentials: creds  );    return new JwtSecurityTokenHandler().WriteToken(token);  }  }  } |

*Program.cs*

|  |
| --- |
| using Microsoft.AspNetCore.Authentication.JwtBearer;  using Microsoft.IdentityModel.Tokens;  using System.Text;    var builder = WebApplication.CreateBuilder(args);    builder.Services.AddControllers();  builder.Services.AddOpenApi();    var jwtSettings = builder.Configuration.GetSection("Jwt");    builder.Services.AddAuthentication("Bearer")  .AddJwtBearer("Bearer", options =>  {  options.TokenValidationParameters = new TokenValidationParameters  {  ValidateIssuer = true,  ValidateAudience = true,  ValidateLifetime = true,  ValidateIssuerSigningKey = true,  ValidIssuer = jwtSettings["Issuer"],  ValidAudience = jwtSettings["Audience"],  IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings["Key"]))  };  });    builder.Services.AddAuthorization();    var app = builder.Build();    if (app.Environment.IsDevelopment())  {  app.MapOpenApi();  }    app.UseHttpsRedirection();    app.UseAuthorization();  app.UseAuthentication();    app.MapControllers();    app.Run(); |

*Output:*

