**Week -5**

6. WebApi\_Handson

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.

Producer.cs

using Confluent.Kafka;

class Program

{

    static async Task Main(string[] args)

    {

        var config = new ProducerConfig

        {

            BootstrapServers = "localhost:9092"

        };

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

        while (true)

        {

            Console.Write("Enter message (or 'exit'): ");

            var message = Console.ReadLine();

            if (message == "exit")

                break;

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

            Console.WriteLine($"Sent to partition {result.Partition} at offset {result.Offset}");

        }

    }

}

**consumer.cs**

using Confluent.Kafka;

class Program

{

    static void Main(string[] args)

    {

        var config = new ConsumerConfig

        {

            BootstrapServers = "localhost:9092",

            GroupId = "chat-consumer-group",

            AutoOffsetReset = AutoOffsetReset.Earliest

        };

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

        consumer.Subscribe("chat-topic");

        Console.WriteLine("Listening for messages... Press Ctrl+C to exit.");

        try

        {

            while (true)

            {

                var result = consumer.Consume();

                Console.WriteLine($"Received: {result.Message.Value}");

            }

        }

        catch (OperationCanceledException) { }

        finally

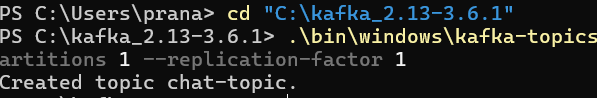
        {

            consumer.Close();

        }

    }

}



Week-5

Hands-On Exercises: Authentication and Authorization in ASP.NET Core Web API Microservices

Question 1: Implement JWT Authentication in ASP.NET Core Web API

Authcontroller.cs

using JwtAuthDemo.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace JwtAuthDemo.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class AuthController : ControllerBase

    {

        [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)

        {

            // 🔐 Replace with real user validation logic

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

        }

        private readonly IConfiguration \_config;

public AuthController(IConfiguration config)

{

    \_config = config;

}

        private string GenerateJwtToken(string username)

{

    var claims = new[]

    {

        new Claim(ClaimTypes.Name, username)

    };

    var key = new SymmetricSecurityKey(

        Encoding.UTF8.GetBytes(\_config["Jwt:Key"]));

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

    var token = new JwtSecurityToken(

        issuer: \_config["Jwt:Issuer"],

        audience: \_config["Jwt:Audience"],

        claims: claims,

        expires: DateTime.Now.AddMinutes(60),

        signingCredentials: creds);

    return new JwtSecurityTokenHandler().WriteToken(token);

}

    }

}

WeatherForcast.cs

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class ValuesController : ControllerBase

    {

        [HttpGet]

        [Authorize]

        public IActionResult Get()

        {

            var username = User.Identity?.Name;

            return Ok($"Hello {username}, this is a protected endpoint!");

        }

    }

}

Models

namespace JwtAuthDemo.Models

{

    public class LoginModel

    {

        public string Username { get; set; }

        public string Password { get; set; }

    }

}

Program.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// 🔐 Add Authentication

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();

// Configure the HTTP request pipeline.

app.UseAuthentication(); // 🔐 Must come before UseAuthorization

app.UseAuthorization();

app.MapControllers();

app.Run();

