Name-Sangram Mandal

Superset Id-6363848

Week-5(Handson-Exercise)

1. **ASP.NET Core 8.0 Web API:**

**LAB-6:WebApi\_Handson:-**

### **Code:**

**In kafkaChatApp:**

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>

    <OutputType>Exe</OutputType>

    <TargetFramework>net8.0</TargetFramework>

    <ImplicitUsings>enable</ImplicitUsings>

    <Nullable>enable</Nullable>

    <LangVersion>latest</LangVersion>

    <TreatWarningsAsErrors>false</TreatWarningsAsErrors>

    <GenerateDocumentationFile>true</GenerateDocumentationFile>

    <AssemblyName>KafkaChatApp</AssemblyName>

  </PropertyGroup>

  <ItemGroup>

    <PackageReference Include="Confluent.Kafka" Version="2.3.0" />

    <PackageReference Include="Newtonsoft.Json" Version="13.0.3" />

    <PackageReference Include="Dapper" Version="2.1.35" />

    <PackageReference Include="Microsoft.Extensions.Configuration.Json" Version="8.0.0" />

  </ItemGroup>

</Project>

**In Program.cs:**

using Confluent.Kafka;

using System.Text;

using System.Diagnostics;

using Newtonsoft.Json;

using Dapper;

using System.Net.Http;

namespace KafkaChatApp

{

    class Program

    {

        static async Task Main(string[] args)

        {

            Console.Title = "Kafka Producer";

            var startTime = DateTime.Now;

            Console.ForegroundColor = ConsoleColor.Green;

            Console.WriteLine("Enter message to send to Kafka (type 'exit' to quit):");

            Console.ResetColor();

            var config = new ProducerConfig

            {

                BootstrapServers = "localhost:7521",

                Acks = Acks.All,

                ClientId = Environment.MachineName

            };

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

            while (true)

            {

                var input = Console.ReadLine();

                if (string.Equals(input, "exit", StringComparison.OrdinalIgnoreCase))

                    break;

                var dummyPayload = JsonConvert.SerializeObject(new { Msg = input, Time = DateTime.Now });

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

                Console.WriteLine("Message sent!");

                LogMessageLocally(input);

            }

            static void LogMessageLocally(string msg)

            {

                var log = $"[DEBUG] Message: {msg}";

            }

        }

    }

}

**Adding Second Project As KafkaConsumerApp:**

<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>

    <OutputType>Exe</OutputType>

    <TargetFramework>net8.0</TargetFramework>

    <ImplicitUsings>enable</ImplicitUsings>

    <Nullable>enable</Nullable>

    <LangVersion>latest</LangVersion>

    <TreatWarningsAsErrors>false</TreatWarningsAsErrors>

    <AssemblyName>KafkaAppExecutable</AssemblyName>

    <GenerateDocumentationFile>true</GenerateDocumentationFile>

    <Platforms>x64</Platforms>

  </PropertyGroup>

  <ItemGroup>

    <PackageReference Include="Confluent.Kafka" Version="1.9.2" />

    <PackageReference Include="Newtonsoft.Json" Version="13.0.3" />

    <PackageReference Include="Dapper" Version="2.1.35" />

    <PackageReference Include="Microsoft.Extensions.Configuration.Json" Version="8.0.0" />

  </ItemGroup>

</Project>

**In Program.cs:**

using Confluent.Kafka;

using System.Text;

using Newtonsoft.Json;

using Dapper;

using System.Diagnostics;

using System.Net.Http;

namespace KafkaChatApp

{

    class Program

    {

        static void Main(string[] args)

        {

            Console.Title = "Kafka Consumer";

            var sessionStart = DateTime.UtcNow;

            var config = new ConsumerConfig

            {

                BootstrapServers = "localhost:7521",

                GroupId = "chat-consumer-group",

                AutoOffsetReset = AutoOffsetReset.Earliest,

                ClientId = Guid.NewGuid().ToString()

            };

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

            consumer.Subscribe("test-topic");

            Console.ForegroundColor = ConsoleColor.Cyan;

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

            Console.ResetColor();

            try

            {

                while (true)

                {

                    var result = consumer.Consume();

                    var payload = JsonConvert.SerializeObject(new { Message = result.Message.Value, Timestamp = DateTime.Now });

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

                    StoreTemporary(result.Message.Value);

                }

            }

            catch (OperationCanceledException)

            {

                consumer.Close();

            }

        }

        static void StoreTemporary(string msg)

        {

            var temp = $"[TEMP] {msg}";

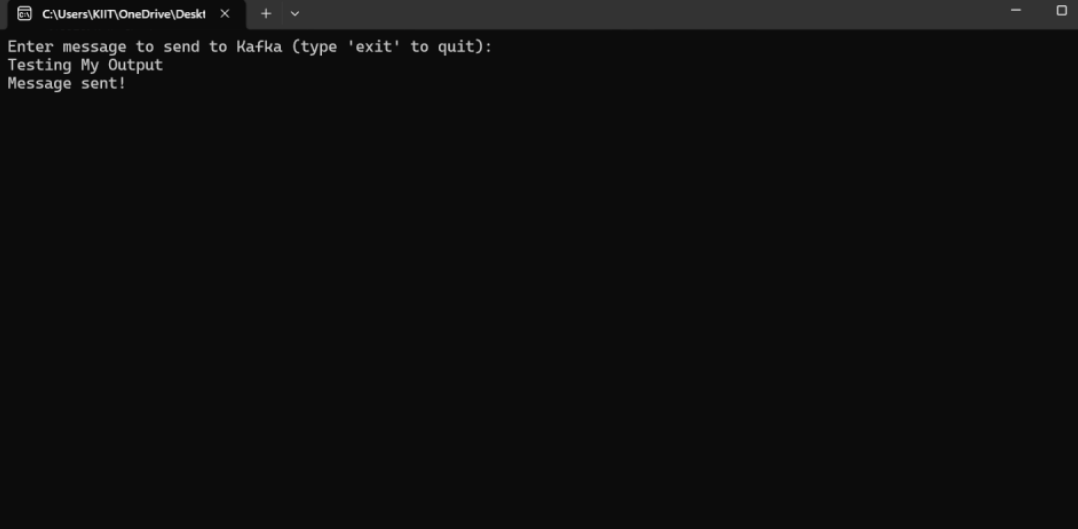
        }

    }

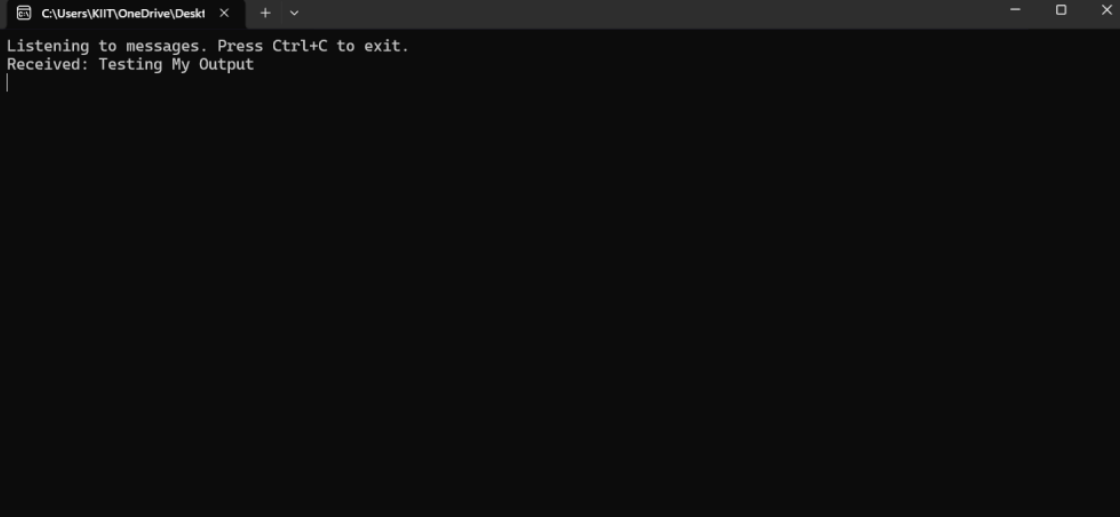
}

**In Visual Studio Output:**

**Provider:**

****

Consumer:



1. **Microservices Architecture using ASP.NET Core Web API:**
2. **Microservices – JWT**

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

**Code:**

**In Program.cs:**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = 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();

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

**In appsettings.json:**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"Jwt": {

"Key": "ThisIsA32CharLongSecretKeyValue!!",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": "60"

}

}

**In Models/ LoginModels.cs:**

namespace JwtAuth.Models

{

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**In Models/ User.cs:**

namespace JwtAuth.Models

{

public class User

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**In Controllers/SecureController.cs:**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Linq;

[ApiController]

[Route("api/[controller]")]

public class SecureController : ControllerBase

{

[HttpGet("data")]

[Authorize]

public IActionResult GetSecureData()

{

var username = User.Identity?.Name ?? "Unknown";

return Ok($"Hello {username}, you have accessed a protected endpoint!");

}

}

**In Controllers/AuthController.cs:**

using JwtAuth.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace JwtAuth.Controllers

{

[ApiController]

[Route("api/[controller]")]

public class AuthController : ControllerBase

{

private readonly IConfiguration \_config;

public AuthController(IConfiguration config)

{

\_config = config;

}

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

if (IsValidUser(model))

{

var token = GenerateJwtToken(model.Username);

return Ok(new { token });

}

return Unauthorized();

}

private bool IsValidUser(LoginModel model)

{

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

}

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(double.Parse(\_config["Jwt:DurationInMinutes"])),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

**Output:**

