**Author - Utsav Saxena**  **Superset Id - 6361856**

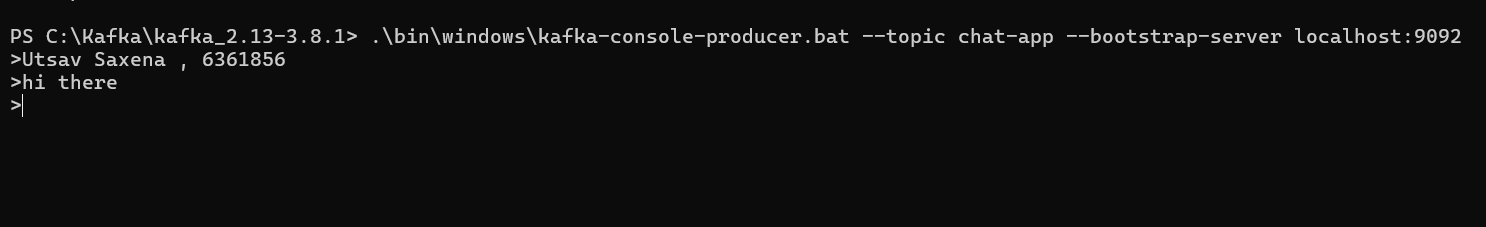
**WEEK 5 - HANDS-ON**

**Module - 6.WebApi HandsOn**

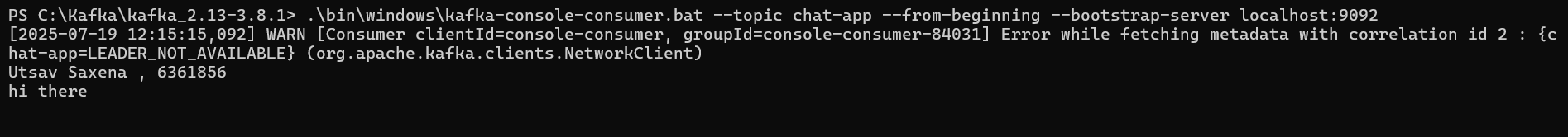
**Question 1:Create a Chat Application which uses Kafka as a streaming platform and consume the chat messages in the command prompt.**

**OUTPUT**

**SENT**

****

**RECEIVED**

****

**Question 2:Create a Chat Application using C# Windows Application using Kafka and consume the message in different client applications.**

**CODE**

**/Producer.cs**

using Confluent.Kafka;

using System;

using System.Threading.Tasks;

class ProducerApp

{

public static async Task Run()

{

var config = new ProducerConfig { BootstrapServers = "localhost:9092" };

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

Console.WriteLine("Send messages (User: Utsav Saxena, 6361856):");

while (true)

{

string input = Console.ReadLine();

if (input.ToLower() == "exit") break;

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

Console.WriteLine("Sent.");

}

}

}

**/Consumer.cs**

using Confluent.Kafka;

using System;

using System.Threading;

class ConsumerApp

{

public static void Run()

{

var config = new ConsumerConfig

{

BootstrapServers = "localhost:9092",

GroupId = "chat-group",

AutoOffsetReset = AutoOffsetReset.Earliest

};

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

consumer.Subscribe("chat-topic");

Console.WriteLine("Kafka Chat reciever Listening...");

while (true)

{

var msg = consumer.Consume(CancellationToken.None);

Console.WriteLine("Friend: " + msg.Message.Value);

}

}

}

**/Program.cs**

class Program

{

static async Task Main(string[] args)

{

Console.WriteLine("Enter mode (send / receive): ");

var mode = Console.ReadLine();

if (mode == "send")

await ProducerApp.Run();

else if (mode == "receive")

ConsumerApp.Run();

else

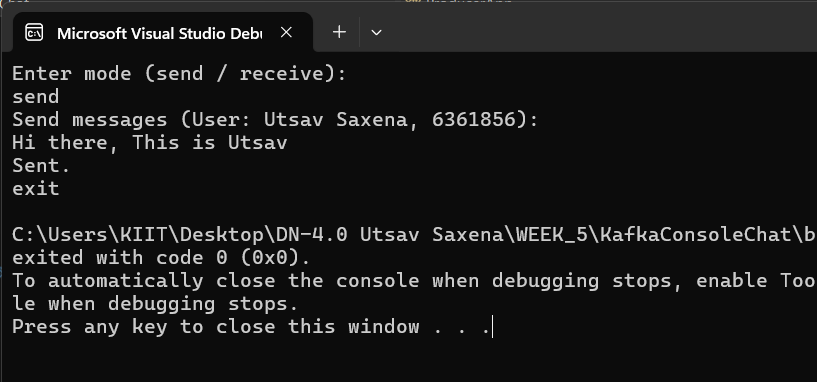
Console.WriteLine("Unknown mode.");

}

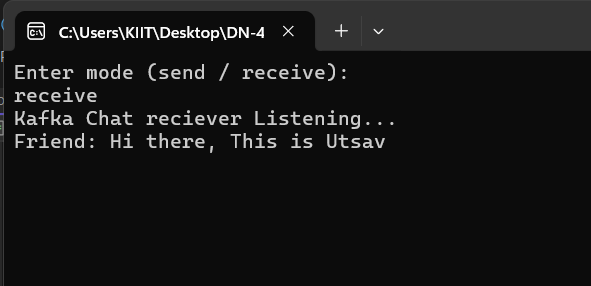
}

**OUTPUT**

SENT

****

RECEIVED



**Module - 1.MicroServices - JWT**

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

**/Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

namespace JwtAuthDemo

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new() { Title = "JwtAuthDemo", Version = "v1" });

c.AddSecurityDefinition("Bearer", new Microsoft.OpenApi.Models.OpenApiSecurityScheme

{

Description = "JWT Authorization 'Bearer {token}'",

Name = "Authorization",

In = Microsoft.OpenApi.Models.ParameterLocation.Header,

Type = Microsoft.OpenApi.Models.SecuritySchemeType.ApiKey,

Scheme = "Bearer"

});

c.AddSecurityRequirement(new Microsoft.OpenApi.Models.OpenApiSecurityRequirement

{

{

new Microsoft.OpenApi.Models.OpenApiSecurityScheme

{

Reference = new Microsoft.OpenApi.Models.OpenApiReference

{

Type = Microsoft.OpenApi.Models.ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] {}

}

});

});

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();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

}

}

}

**/WeatherForecast.cs**

namespace JwtAuthDemo

{

public class WeatherForecast

{

public DateOnly Date { get; set; }

public int TemperatureC { get; set; }

public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

public string? Summary { get; set; }

}

}

**/Controllers/AuthController.cs**

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

{

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("Invalid username or password");

}

private bool IsValidUser(LoginModel model)

{

return model.Username == "Utsav" && model.Password == "6361856";

}

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);

}

}

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**/Controllers/UserController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

[ApiController]

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

public class UserController : ControllerBase

{

[HttpGet("getdata")]

[Authorize]

public IActionResult GetData()

{

var username = User.Identity?.Name;

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

}

}

}

**/Controllers/UserController.cs**

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

[ApiController]

[Route("[controller]")]

public class WeatherForecastController : ControllerBase

{

private static readonly string[] Summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

};

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(ILogger<WeatherForecastController> logger)

{

\_logger = logger;

}

[HttpGet(Name = "GetWeatherForecast")]

public IEnumerable<WeatherForecast> Get()

{

return Enumerable.Range(1, 5).Select(index => new WeatherForecast

{

Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),

TemperatureC = Random.Shared.Next(-20, 55),

Summary = Summaries[Random.Shared.Next(Summaries.Length)]

})

.ToArray();

}

}

}

**/Controllers/WeatherForecastController.cs**

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

[ApiController]

[Route("[controller]")]

public class WeatherForecastController : ControllerBase

{

private static readonly string[] Summaries = new[]

{

"Utsav", "Bracing", "Chilly", "Cool", "Saxena", "Warm", "Balmy", "Hot", "6361856", "Scorching"

};

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(ILogger<WeatherForecastController> logger)

{

\_logger = logger;

}

[HttpGet(Name = "GetWeatherForecast")]

public IEnumerable<WeatherForecast> Get()

{

return Enumerable.Range(1, 5).Select(index => new WeatherForecast

{

Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),

TemperatureC = Random.Shared.Next(-20, 55),

Summary = Summaries[Random.Shared.Next(Summaries.Length)]

})

.ToArray();

}

}

}

**OUTPUT**

