**Week 4:**

**WEB API AND MICROSERVICES:**

**Web API Hands\_ON:**

**Program .cs:**

using System;

using Confluent.Kafka;

using System.Threading.Tasks;

class Producer

{

public static async Task Main(string[] args)

{

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

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

Console.WriteLine("Enter messages, type 'exit' to quit:");

while (true)

{

Console.Write("> ");

var input = Console.ReadLine();

if (input == "exit") break;

var result = await producer.ProduceAsync(

"chat-topic",

new Message<Null, string> { Value = input });

Console.WriteLine($"Sent to {result.TopicPartitionOffset}");

}

}

}

Kafka Consumer:

using System;

using Confluent.Kafka;

class Consumer

{

public 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...");

while (true)

{

var result = consumer.Consume();

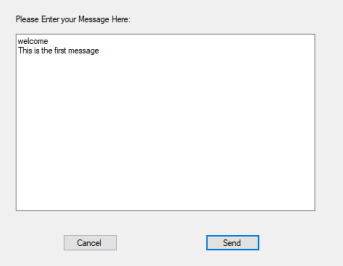
Console.WriteLine($"Received: {result.Message.Value}");

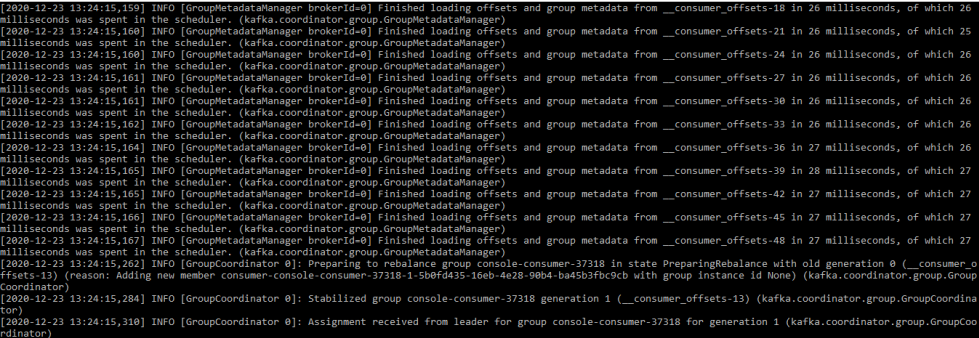
}

}

}

OUTPUT:





**Microservices:**

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

[**Program.cs**](http://program.cs)

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

builder.Services.AddAuthentication("Bearer")

.AddJwtBearer("Bearer", 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"]))

};

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

if (context.Exception.GetType() == typeof(SecurityTokenExpiredException))

{

context.Response.Headers.Add("Token-Expired", "true");

}

return Task.CompletedTask;

}

};

});

builder.Services.AddAuthorization();

**User and login Model:**

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

[**AuthController.cs**](http://authcontroller.cs)

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

[ApiController]

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

public class AuthController : ControllerBase

{

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

if (IsValidUser(model))

{

var token = GenerateJwtToken(model.Username, "Admin"); // Add "Admin" for testing role-based

return Ok(new { Token = token });

}

return Unauthorized();

}

private bool IsValidUser(LoginModel model)

{

// Replace this with your actual user validation logic

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

}

private string GenerateJwtToken(string username, string role)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, role)

};

var key = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes("ThisIsASecretKeyForJwtToken"));

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

var token = new JwtSecurityToken(

issuer: "MyAuthServer",

audience: "MyApiUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(60),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

[**SecureController.cs**](http://securecontroller.cs)

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

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

public class SecureController : ControllerBase

{

[HttpGet("data")]

[Authorize]

public IActionResult GetSecureData()

{

return Ok("This is protected data.");

}

}

[**AdminController.cs**](http://admincontroller.cs)

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

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

public class AdminController : ControllerBase

{

[HttpGet("dashboard")]

[Authorize(Roles = "Admin")]

public IActionResult GetAdminDashboard()

{

return Ok("Welcome to the admin dashboard.");

}

}