**Deep Skilling Week 5 Assignment**

**Kafka Integration with C#:**

**Producer.cs**

using Confluent.Kafka;

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

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

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

string? input;

while ((input = Console.ReadLine()) != "exit")

{

producer.Produce("chat-topic", new Message<Null, string> { Value =

input });

producer.Flush(TimeSpan.FromSeconds(2));

}

**Consumer.cs**

using Confluent.Kafka;

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("Listening for messages...");

while (true)

{

var cr = consumer.Consume();

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

}

****Form1.cs** - Basic chat sender**

private async void btnSend\_Click(object sender, EventArgs e)

{

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

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

var msg = txtMessage.Text;

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

listBoxMessages.Items.Add("You: " + msg);

txtMessage.Clear();

}

**Message Listener**

private async void StartConsumer()

{

await Task.Run(() =>

{

var config = new ConsumerConfig

{

BootstrapServers = "localhost:9092",

GroupId = "chat-winforms-group",

AutoOffsetReset = AutoOffsetReset.Earliest

};

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

consumer.Subscribe("chat-topic");

while (true)

{

var msg = consumer.Consume();

Invoke((MethodInvoker)(() => {

listBoxMessages.Items.Add("Friend: " + msg.Message.Value);

}));

}

});

}

**Output:**

Friend: Hello!

Friend: How are you?

**Authentication and Authorization in ASP.NET Core Web API Microservices.**

**Question 1: Implement JWT Authentication**

**Code:**

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

"Jwt": {

"Key": "ThisIsASecretKeyForJwtToken",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

}

**Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

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"]))

};

});

builder.Services.AddAuthorization();

builder.Services.AddControllers();

var app = builder.Build();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

#### ****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 (model.Username == "admin" && model.Password == "admin123")

{

var token = GenerateJwtToken(model.Username);

return Ok(new { Token = token });

}

return Unauthorized("Invalid credentials.");

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username)

};

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);

}

}

**Question 2: Secure an API Endpoint Using JWT**

**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.");

}

}

**Question 3: Add Role-Based Authorization**

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, "Admin")

};

**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.");

}

}

**Question 4: Validate Token Expiry and Handle Unauthorized Access**

**Program.cs**

builder.Services.AddAuthentication("Bearer")

.AddJwtBearer("Bearer", options =>

{

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

if (context.Exception.GetType() ==

typeof(SecurityTokenExpiredException))

{

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

}

return Task.CompletedTask;

}

};

});