Microservices & WebAPI Hands-On Solutions

# Submitted by: Utsav Kumar

## Roll Number: 22051121

# Part 1: Implementing JWT Authentication in ASP.NET Core Web API

In this exercise, we build a simple ASP.NET Core Web API microservice that uses JWT (JSON Web Token) for authentication. This allows us to securely log in users and protect endpoints from unauthorized access.

## Scenario

We are creating a microservice that requires secure login. Users will send their credentials to a login endpoint, and if valid, they will receive a JWT token. This token must then be used to access protected API endpoints.

## Step-by-Step Solution

1. Create a New ASP.NET Core Web API Project

Command:  
dotnet new webapi -n JwtAuthDemo  
This sets up a basic ASP.NET Core Web API project.

2. Install Required NuGet Packages

We need the JWT Bearer authentication package. Run:  
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

3. Configure JWT Settings in appsettings.json

{  
 "Jwt": {  
 "Key": "ThisIsASecretKeyForJwtToken",  
 "Issuer": "MyAuthServer",  
 "Audience": "MyApiUsers",  
 "DurationInMinutes": 60  
 }  
}

4. Configure Authentication in Program.cs

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();

5. Create a Login Model and Auth Controller

public class LoginModel  
{  
 public string Username { get; set; }  
 public string Password { get; set; }  
}  
  
[ApiController]  
[Route("api/[controller]")]  
public class AuthController : ControllerBase  
{  
 [HttpPost("login")]  
 public IActionResult Login([FromBody] LoginModel model)  
 {  
 // For demo purposes, accept only Utsav Kumar  
 if (model.Username == "Utsav Kumar" && model.Password == "22051121")  
 {  
 var token = GenerateJwtToken(model.Username);  
 return Ok(new { Token = token });  
 }  
 return Unauthorized();  
 }  
  
 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);  
 }  
}

6. Protect an API Endpoint with Authorize

[ApiController]  
[Route("api/[controller]")]  
public class SecureController : ControllerBase  
{  
 [HttpGet("data")]  
 [Authorize]  
 public IActionResult GetSecureData()  
 {  
 return Ok("Welcome Utsav Kumar, you have accessed protected data");  
 }  
}

## Sample Output

Request:  
POST /api/auth/login  
{ "username": "Utsav Kumar", "password": "22051121" }  
  
Response:  
{ "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..." }  
  
Request with token:  
GET /api/secure/data  
Authorization: Bearer <token>  
  
Response:  
"Welcome Utsav Kumar, you have accessed protected data"

# Part 2: Kafka Integration with C#

We build a Kafka-based chat system that can send and receive messages in real-time.

## Scenario

We want to build two Kafka-based chat applications:  
1. A simple console-based chat publisher and consumer.  
2. A Windows Forms app that can send and receive chat messages across multiple clients.

## Step-by-Step Solution

1. Install and Run Kafka and Zookeeper

Start Zookeeper:  
Zookeeper-server-start.bat ../../config/zookeeper.properties  
  
Start Kafka Server:  
Kafka-server-start.bat ../../config/server.properties  
  
Create a topic:  
Kafka-topics.bat --create --topic chat-topic --bootstrap-server localhost:9092

2. Console-Based Kafka Chat Producer

using Confluent.Kafka;  
  
class Program  
{  
 static async Task Main()  
 {  
 var config = new ProducerConfig { BootstrapServers = "localhost:9092" };  
  
 using var producer = new ProducerBuilder<Null, string>(config).Build();  
  
 Console.WriteLine("Type a message (or 'exit' to quit):");  
 while (true)  
 {  
 var msg = Console.ReadLine();  
 if (msg == "exit") break;  
  
 await producer.ProduceAsync("chat-topic", new Message<Null, string> { Value = "Utsav Kumar: " + msg });  
 Console.WriteLine("Message sent successfully");  
 }  
 }  
}

3. Kafka Consumer to Receive Messages

var config = new ConsumerConfig  
{  
 GroupId = "chat-group",  
 BootstrapServers = "localhost:9092",  
 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($"Received: {cr.Message.Value}");  
}

## Sample Output

Producer Output:  
Type a message (or 'exit' to quit):  
Hello from Utsav Kumar  
Message sent successfully  
  
Consumer Output:  
Received: Utsav Kumar: Hello from Utsav Kumar

## References

https://www.c-sharpcorner.com/article/apache-kafka-net-application/

https://www.c-sharpcorner.com/article/step-by-step-installation-and-configuration-guide-of-apache-kafka-on-windows-ope/