6.WebApi_Handson

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

Program.cs

```
using Confluent.Kafka;
using System;
using System.Threading.Tasks;

0 references
class Program
{
    0 references
    static async Task Main(string[] args)
    {
        var config = new ProducerConfig
        {
            BootstrapServers = "localhost:9092"
        };

        Console.WriteLine("Enter your name:");
        string user = Console.ReadLine();

        using (var producer = new ProducerBuilder<Null, string>(config).Build())
        {
            Console.WriteLine("Start chatting (type 'exit' to quit):");

            while (true)
            {
                string message = Console.ReadLine();
                if (message.ToLower() == "exit") break;

```

```
                string fullMessage = $"{user}: {message}";

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

                Console.WriteLine($"[Sent] → {fullMessage}");
            }
        }
    }
}
```

OUTPUT:

consumes the chat messages in the command prompt.

```
D:\KafkaChatProducer\bin\De
Enter your name:
THEJA
Start chatting (type 'exit' to quit):
HI
[Sent] ? THEJA: HI
HOW ARE YOU
[Sent] ? THEJA: HOW ARE YOU
```

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

ProducerApp

Program.cs

```
ProducerApp                                   Program
using System;
using Confluent.Kafka;

class Program
{
    static async Task Main(string[] args)
    {
        Console.WriteLine("⚡ Producer App Starting...");
        Console.Write("Enter your name: ");
        string name = Console.ReadLine();

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

        using var producer = new ProducerBuilder<Null, string>(config).Build();
        Console.WriteLine("☑ Connected to Kafka");

        while (true)
        {
            Console.Write("Start chatting (type 'exit' to quit): ");
            string message = Console.ReadLine();

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

            string finalMessage = $"{name}: {message}";
            await producer.ProduceAsync("chat-topic", new Message<Null, string> { Value = finalMessage });
            Console.WriteLine($"[Sent] → {finalMessage}");
        }
    }
}
```

ConsumerApp

Program.cs

```
ConsumerApp                                                    Program
using System;
using Confluent.Kafka;

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("🟠 Consumer App Starting...");
        var config = new ConsumerConfig
        {
            BootstrapServers = "localhost:9092",
            GroupId = Guid.NewGuid().ToString(),
            AutoOffsetReset = AutoOffsetReset.Earliest
        };

        using var consumer = new ConsumerBuilder<Ignore, string>(config).Build();
        consumer.Subscribe("chat-topic");

        Console.WriteLine("✅ Subscribed to topic 'chat-topic'. Listening for messages...\n");

        while (true)
        {
            var cr = consumer.Consume();
            Console.WriteLine(cr.Message.Value);
        }
    }
}
```

OUTPUT:

```
D:\KafkaChatProducer\bin\De
Enter your name:
THEJA
Start chatting (type 'exit' to quit):
HI
[Sent] ? THEJA: HI
HOW ARE YOU
[Sent] ? THEJA: HOW ARE YOU
```

```
PS D:\KafkaChat\ConsumerApp\bin\Debug\net9.0> .\ConsumerApp.exe
?? Consumer App Starting...
? Subscribed to topic 'chat-topic'. Listening for messages...

THEJA: HI
THEJA: HOW ARE YOU
```

Authentication and Authorization in ASP.NET Core Web API Microservices Hands-On Exercises

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

Program.cs

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        var config = builder.Configuration;
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            ValidIssuer = config["Jwt:Issuer"],
            ValidAudience = config["Jwt:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(config["Jwt:Key"]))
        };
    });

builder.Services.AddAuthorization();

```

```
var app = builder.Build();

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

app.UseAuthentication();
app.UseAuthorization();

var summaries = new[]
{
    "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
};

app.MapGet("/weatherforecast", () =>
{
    var forecast = Enumerable.Range(1, 5).Select(index =>
        new WeatherForecast
        (
            DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
            Random.Shared.Next(-20, 55),
            summaries[Random.Shared.Next(summaries.Length)]
        ))
```

```
54            ))
55            .ToArray();
56        return forecast;
57    })
58    .WithName("GetWeatherForecast")
59    .WithOpenApi();
60
61    app.MapGet("/secret", [Microsoft.AspNetCore.Authorization.Authorize] () =>
62    {
63        return Results.Ok("🎉 You accessed a protected endpoint!");
64    })
65    .WithName("SecretEndpoint");
66
67    app.Run();
68
      1 reference
69    internal record WeatherForecast(DateOnly Date, int TemperatureC, string? Summary)
70    {
          0 references
71        public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);
72    }
```

AuthController.cs

```
1     using JwtAuth.Models;
2     using Microsoft.AspNetCore.Mvc;
3     using Microsoft.IdentityModel.Tokens;
4     using System.IdentityModel.Tokens.Jwt;
5     using System.Security.Claims;
6     using System.Text;
7
8     namespace JwtAuth.Controllers
9     {
10        [ApiController]
11        [Route("api/[controller]")]
          1 reference
12        public class AuthController : ControllerBase
13        {
14            private readonly IConfiguration _config;
15
              0 references
16            public AuthController(IConfiguration config)
17            {
18                _config = config;
19            }
20
21            [HttpPost("login")]
              0 references
22            public IActionResult Login([FromBody] LoginModel model)
23            {
24                if (IsValidUser(model))
25                {
26                    var token = GenerateJwtToken(model.Username);
27                    return Ok(new { Token = token });
28                }
```

```
30                  return Unauthorized();
31              }
32
                1 reference
33              private bool IsValidUser(LoginModel model)
34              {
35                  return model.Username == "admin" && model.Password == "admin123";
36              }
37
                1 reference
38              private string GenerateJwtToken(string username)
39              {
40                  var claims = new[]
41                  {
42                      new Claim(ClaimTypes.Name, username)
43                  };
44
45                  var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
46                  var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
47
48                  var token = new JwtSecurityToken(
49                      issuer: _config["Jwt:Issuer"],
50                      audience: _config["Jwt:Audience"],
51                      claims: claims,
52                      expires: DateTime.Now.AddMinutes(Convert.ToDouble(_config["Jwt:DurationInMinutes"])),
53                      signingCredentials: creds
54                  );
55
56                  return new JwtSecurityTokenHandler().WriteToken(token);
57              }
58          }
59      }
60
```

appsettings.json

```
1   {
2     "Logging": {
3       "LogLevel": {
4         "Default": "Information",
5         "Microsoft.AspNetCore": "Warning"
6       }
7     },
8     "AllowedHosts": "*",
9     "Jwt": {
10      "Key": "ThisIsASecretKeyForJwtToken",
11      "Issuer": "MyAuthServer",
12      "Audience": "MyApiUsers",
13      "DurationInMinutes": 60
14    }
15
16  }
```

LoginModel.cs

```
1  namespace JwtAuth.Models
2  {
       2 references
3      public class LoginModel
4      {
           2 references
5          public string Username { get; set; }
           1 reference
6          public string Password { get; set; }
7      }
8  }
```

OUTPUT:

http://localhost:5085/api/auth/login

Save | Share

POST http://localhost:5085/api/auth/login Send

Params | Authorization | Headers (9) | Body | Scripts | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON | Beautify

```
1 {
2   "username": "admin",
3   "password": "password"
4 }
5
```

Body | Cookies | Headers (4) | Test Results | 200 OK | 4.06 s | 505 B

{} JSON | Preview | Visualize


```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
      eyJodHRwOi8vc2NoZW1hcy54bWxzb2FwLm9yZy93cy8yMDA1LzA1L2lkZW50aXR5L2NsYWltcy9uYW1lIjoiYWRtaW4iLCJodHRwOi8vc2NoZW1hcy5taWNyb3NvZnQuY29tL3dzLzIwMDgvMDYvaWRlbnRpdHkvY2xhaW1zL3Jvb
      GUiOiJBZG1pbiIsImV4cCI6MTc1MjkSMTIyNCwiaXNzIjoiTXlBdXRoU2VydmVyIiwiYXVkIjoiTXlBcGlVc2VycyJ9.V07t_ZORpVpYnkpTP-iGsj__txsa7dKIt0sFway19xc"
3 }
```

TASK 2: Secure an API Endpoint Using JWT

Scenario:

You want to restrict access to a sensitive endpoint using JWT authentication.

Steps:

1. Add `[Authorize]` to a controller.

2. Test access with and without a valid token.

Program.cs

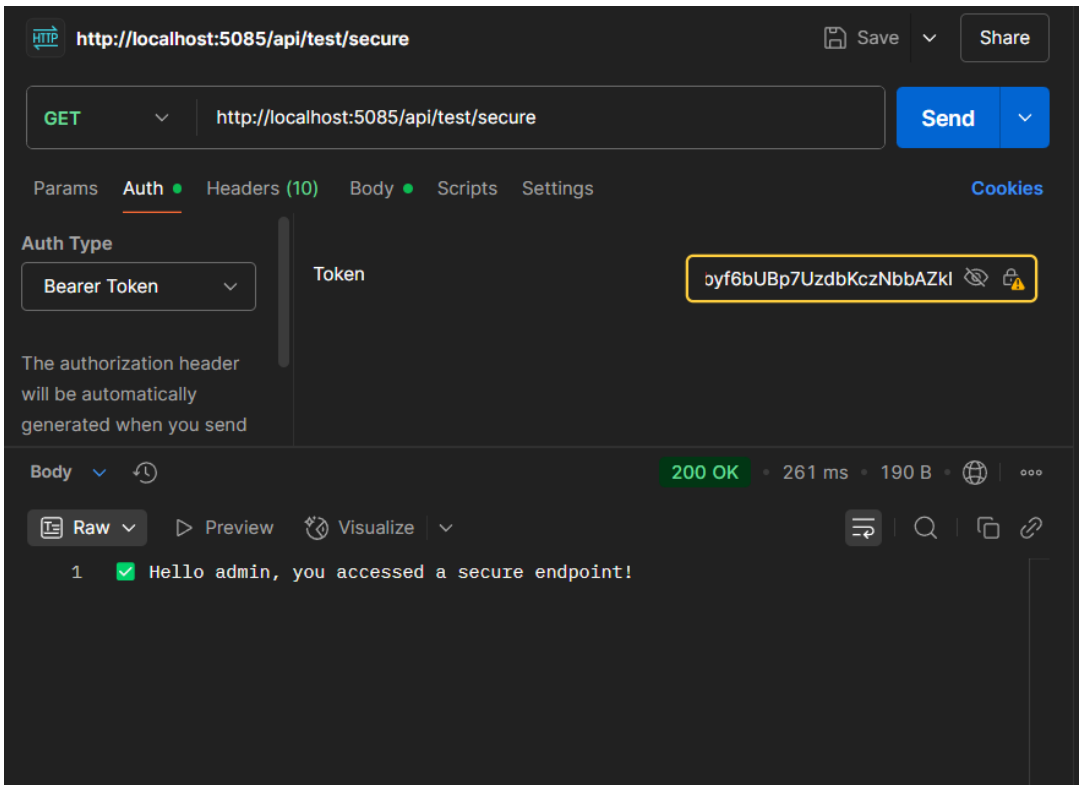
```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using JwtAuth.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;

namespace JwtAuth.Controllers
{
    [ApiController]
    [Route("api/[controller]")]
    1 reference
    public class AuthController : ControllerBase
    {
        private readonly IConfiguration _config;

        private readonly List<LoginModel> _users = new()
        {
            new LoginModel { Username = "admin", Password = "password", Role = "Admin" },
        };

        0 references
        public AuthController(IConfiguration config)
        {
            _config = config;
        }

        [HttpPost("login")]
        0 references
        public IActionResult Login([FromBody] LoginModel model)
```

```
29          {
30              var user = _users.FirstOrDefault(u =>
31                  u.Username == model.Username &&
32                  u.Password == model.Password);
33
34              if (user == null)
35                  return Unauthorized("Invalid username or password");
36
37              var token = GenerateJwtToken(user);
38              return Ok(new { token });
39          }
40          [Route("api/test")]
41          [ApiController]
            0 references
42          public class TestController : ControllerBase
43          {
44              [HttpGet("secure")]
45              [Authorize]
                0 references
46              public IActionResult GetSecureData()
47              {
48                  return Ok("This is a protected endpoint. You are authorized!");
49              }
50          }
51
            1 reference
52          private string GenerateJwtToken(LoginModel user)
53          {
54              var claims = new[]
55              {
```

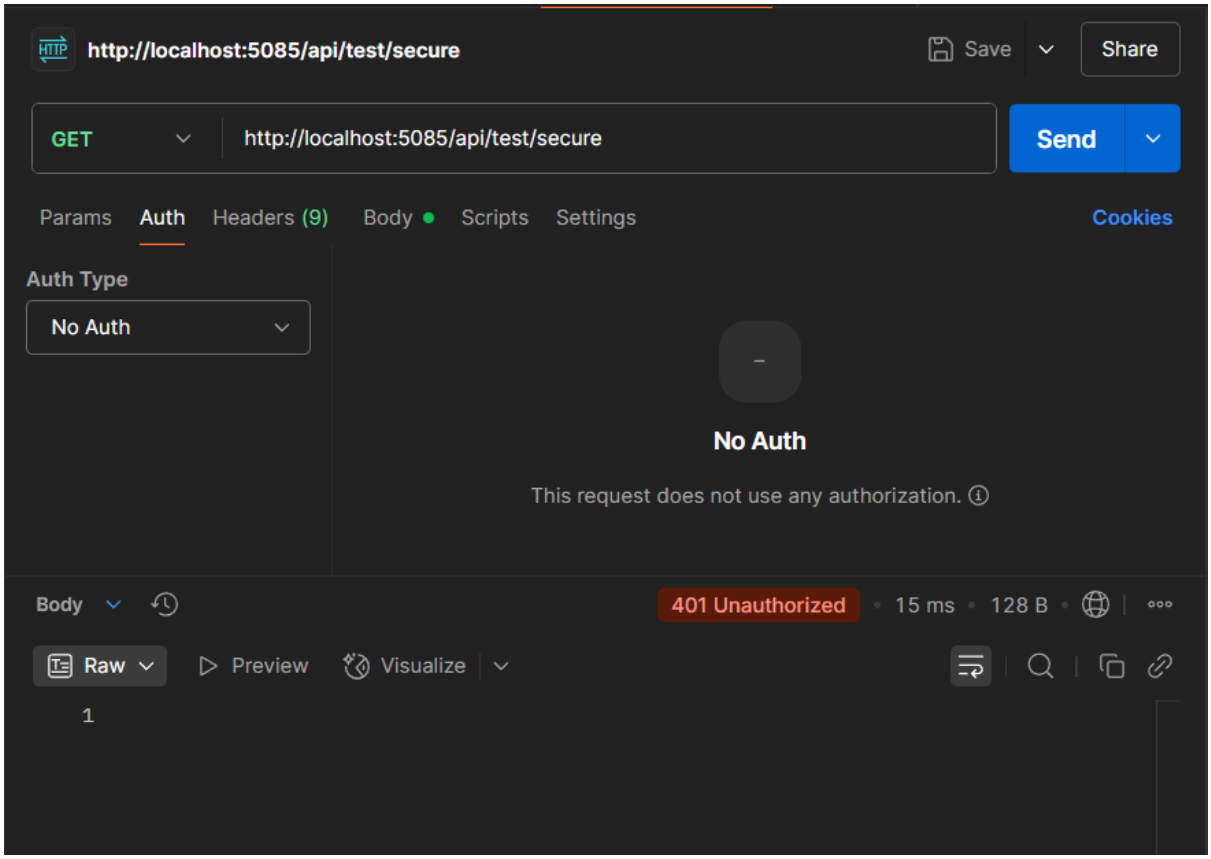
```
56                  new Claim(ClaimTypes.Name, user.Username),
57                  new Claim(ClaimTypes.Role, user.Role)
58              };
59
60              var key = _config["Jwt:Key"];
61              var issuer = _config["Jwt:Issuer"];
62              var audience = _config["Jwt:Audience"];
63
64              var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(key));
65              var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
66
67              var token = new JwtSecurityToken(
68                  issuer: issuer,
69                  audience: audience,
70                  claims: claims,
71                  expires: DateTime.UtcNow.AddHours(1),
72                  signingCredentials: credentials
73              );
74
75              return new JwtSecurityTokenHandler().WriteToken(token);
76          }
77
78      }
79  }
```

OUTPUT:

With Header



Without header



TASK 3: Add Role-Based Authorization

Scenario:

You want to allow only users with the "Admin" role to access certain endpoints.

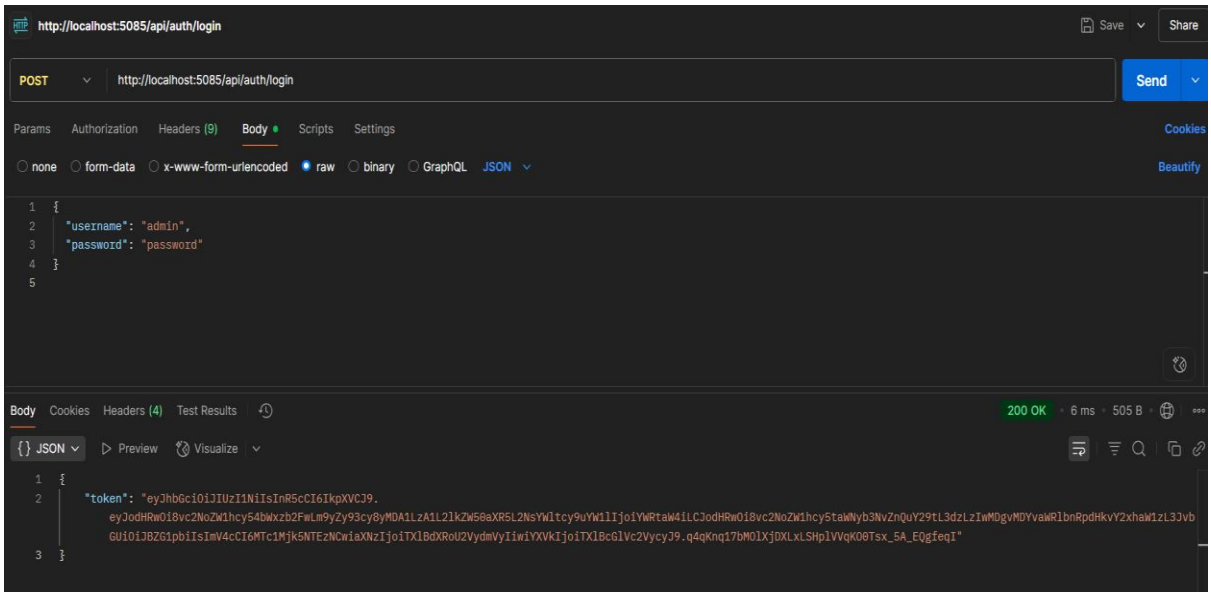
Steps:

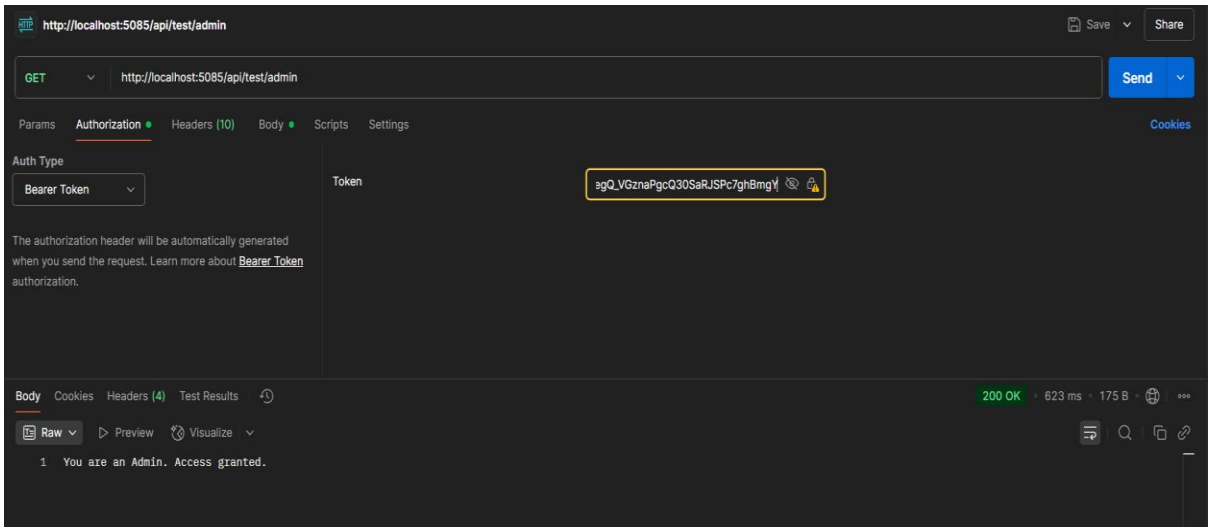
1. Add roles to JWT claims.

2. Use `[Authorize(Roles = "Admin")]`.

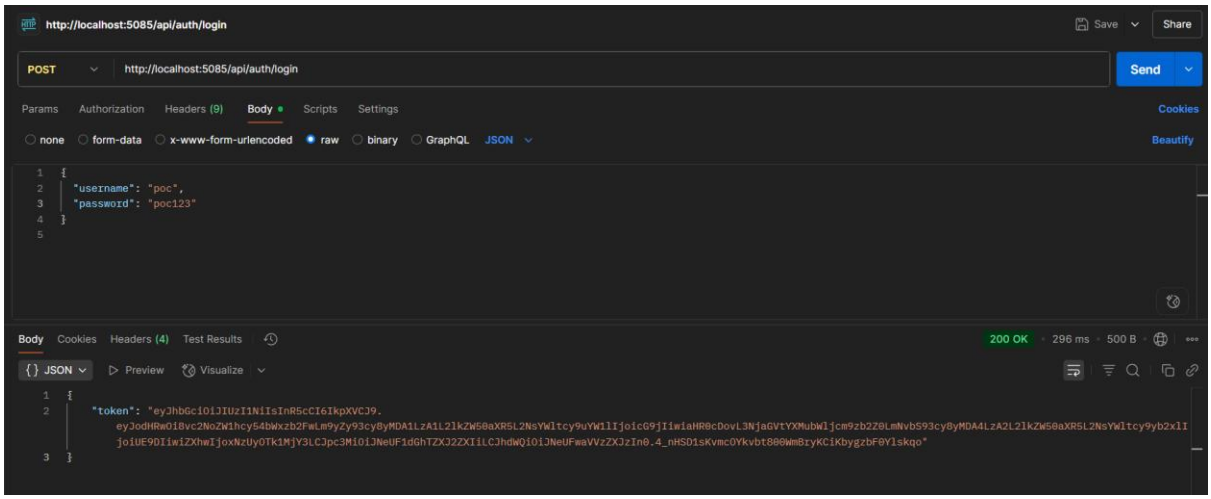
OUTPUT:

i)Admin

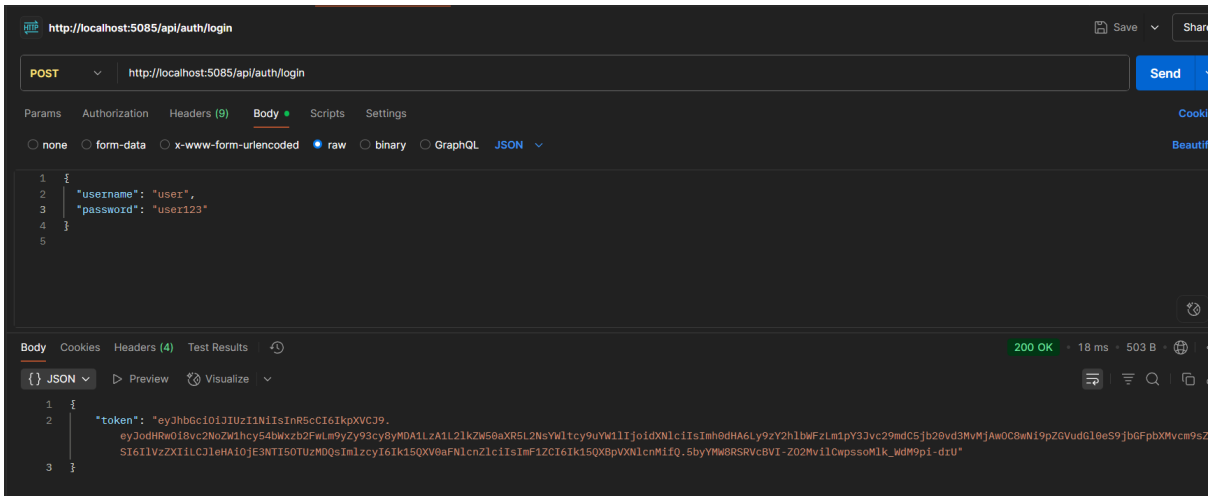




ii)poc



iii)user



TASK 4: Validate JWT Token Expiry and Handle Unauthorized Access Scenario: You want to handle expired or invalid tokens gracefully.

Steps:

1. Configure JWT bearer events.

2. Return custom messages for unauthorized access.

Program.cs

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using Microsoft.AspNetCore.Http;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
})
.AddJwtBearer(options =>
{
    var jwtSettings = builder.Configuration.GetSection("Jwt");
    var key = Encoding.UTF8.GetBytes(jwtSettings["Key"]);

    options.TokenValidationParameters = new TokenValidationParameters
    {
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        ValidIssuer = jwtSettings["Issuer"],
        ValidAudience = jwtSettings["Audience"],
        IssuerSigningKey = new SymmetricSecurityKey(key)
    };

    options.Events = new JwtBearerEvents
    {
        OnAuthenticationFailed = context =>
```

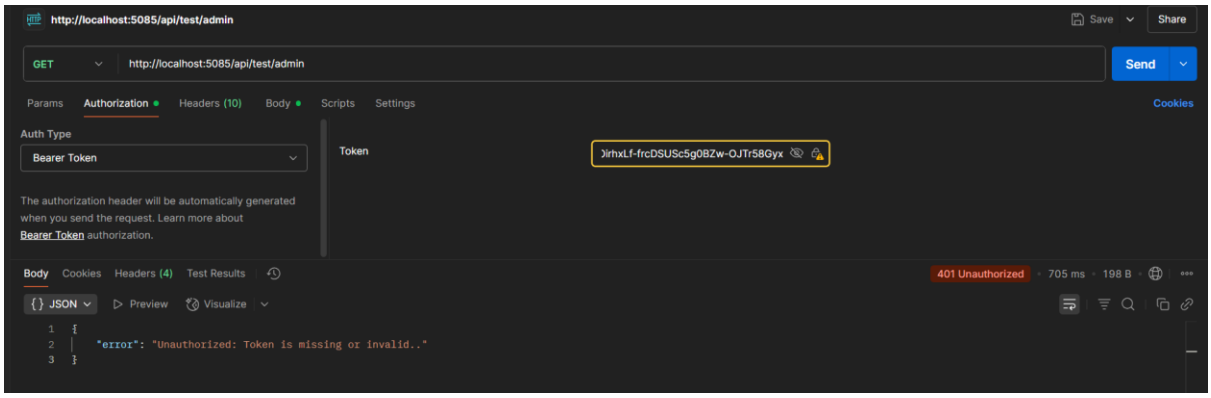
```
32                {
33                    if (context.Exception.GetType() == typeof(SecurityTokenExpiredException))
34                    {
35                        context.Response.Headers.Add("Token-Expired", "true");
36                    }
37                    return Task.CompletedTask;
38                },
39
40                OnChallenge = context =>
41                {
42                    context.HandleResponse();
43                    context.Response.StatusCode = 401;
44                    context.Response.ContentType = "application/json";
45
46                    var result = System.Text.Json.JsonSerializer.Serialize(new
47                    {
48                        error = "Unauthorized: Token is missing, invalid, or expired."
49                    });
50
51                    return context.Response.WriteAsync(result);
52                }
53            };
54        });
55
56        builder.Services.AddAuthorization();
57        builder.Services.AddControllers();
58
59        var app = builder.Build();
60
61        app.UseRouting();
62        app.UseAuthentication();
63        app.UseAuthorization();
64        app.MapControllers();
65
66        app.Run();
```

appsettings.json

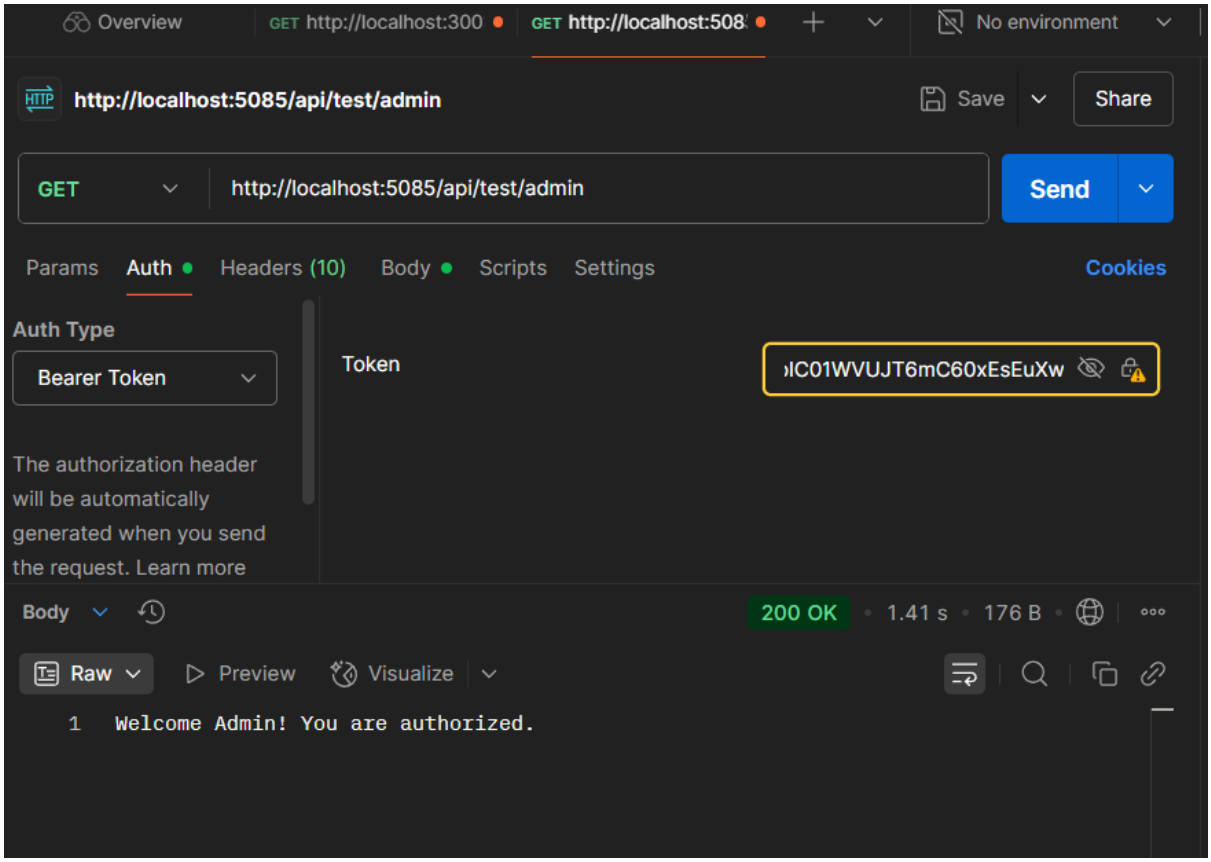
```
1  {
2    "Jwt": {
3      "Key": "ThisIsASecretKeyForJwtTokenWith32Chars!!",
4      "Issuer": "MyAuthServer",
5      "Audience": "MyApiUsers",
6      "DurationInMinutes": 2
7    },
8    "Logging": {
9      "LogLevel": {
10       "Default": "Information",
11       "Microsoft.AspNetCore": "Warning"
12     }
13   },
14   "AllowedHosts": "*"
15 }
```

OUTPUT:

For Unauthorized User login:



Token Before expiring:



Token after expiring: