**DN 4.0 Dotnet FSE**

**Name: Sparsh Guha**

**Superset ID:6361106**

**Week 5**

**QUESTION - 1 (API-6)  
Kafka Integration with C#:**

1. **Create a Chat Application which uses Kafka as a streaming platform and consume the chat messages in the command prompt.**
2. **Create a Chat Application using C# Windows Application using Kafka and consume the message in different client applications.**

**Introduction to Kafka**  
Apache Kafka is a distributed event streaming platform for high-throughput, low-latency data pipelines and streaming apps  
It enables real-time publish/subscribe messaging, storage, and processing of streams.

**Kafka Architecture**  
Producer: Publishes messages to Kafka topics.

Consumer: Subscribes and reads messages from topics.

Broker: Kafka server that stores messages and handles requests.

Multiple brokers form a cluster for scalability and fault tolerance

Topic: Logical channel for organizing messages. Producers write to topics, consumers read from them

Partition: Each topic is split into partitions for parallelism and scalability. Each partition is an ordered, immutable sequence of messages

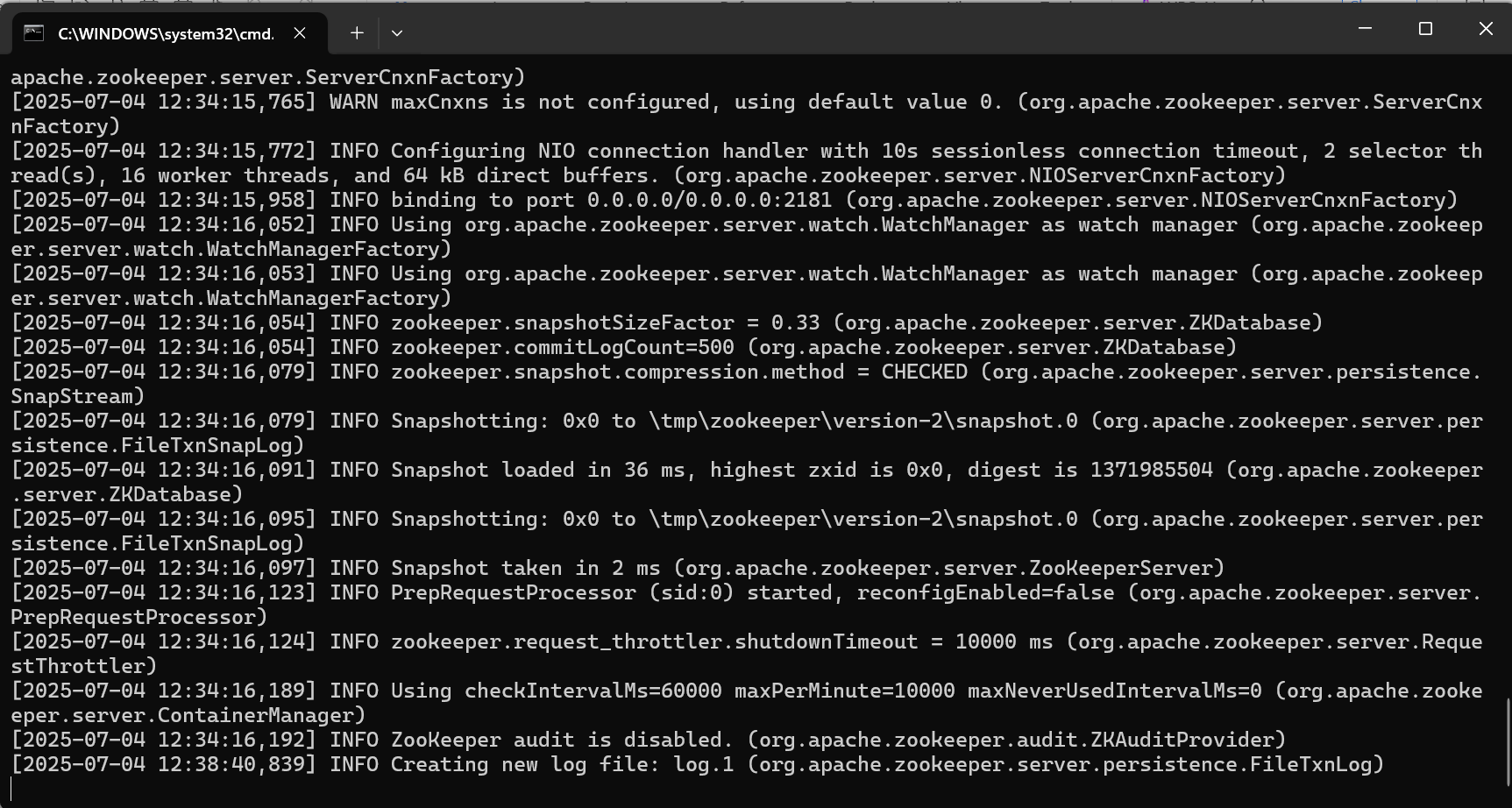
ZooKeeper: Coordinates brokers and manages cluster state (required for older Kafka versions)

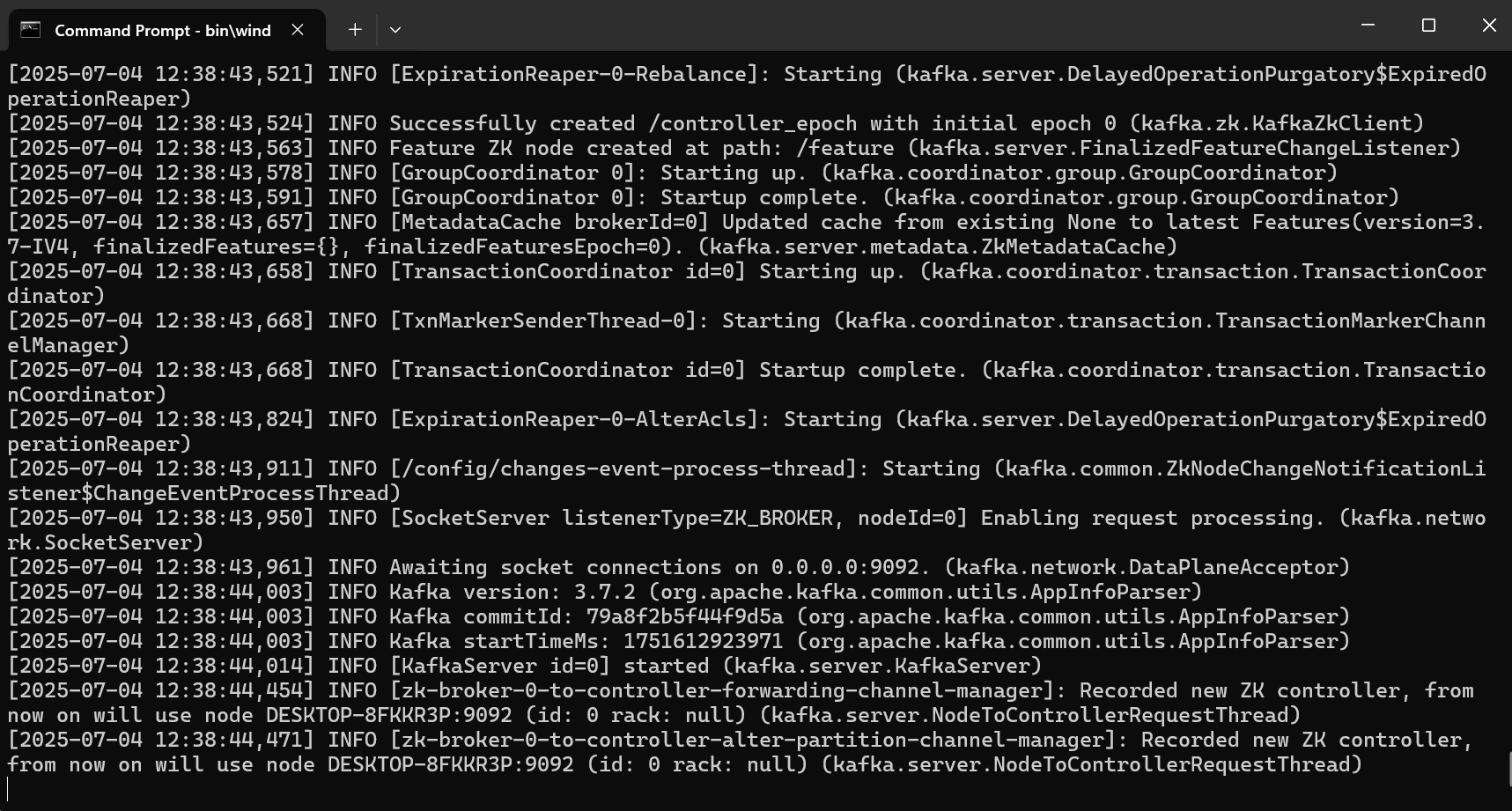
Replication: Partitions are replicated across brokers for fault tolerance.

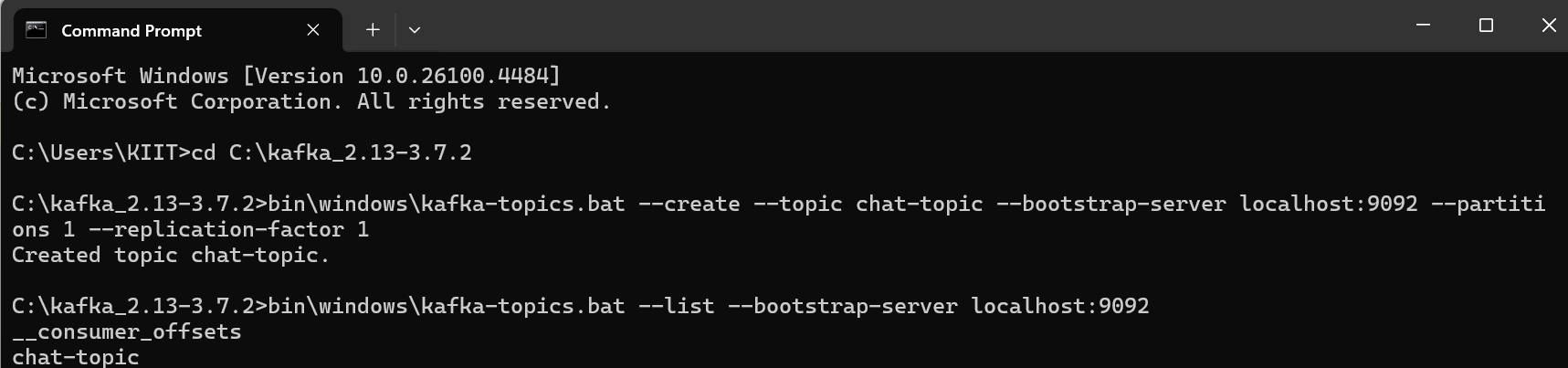
Consumer Group: Multiple consumers sharing the workload for a topic.

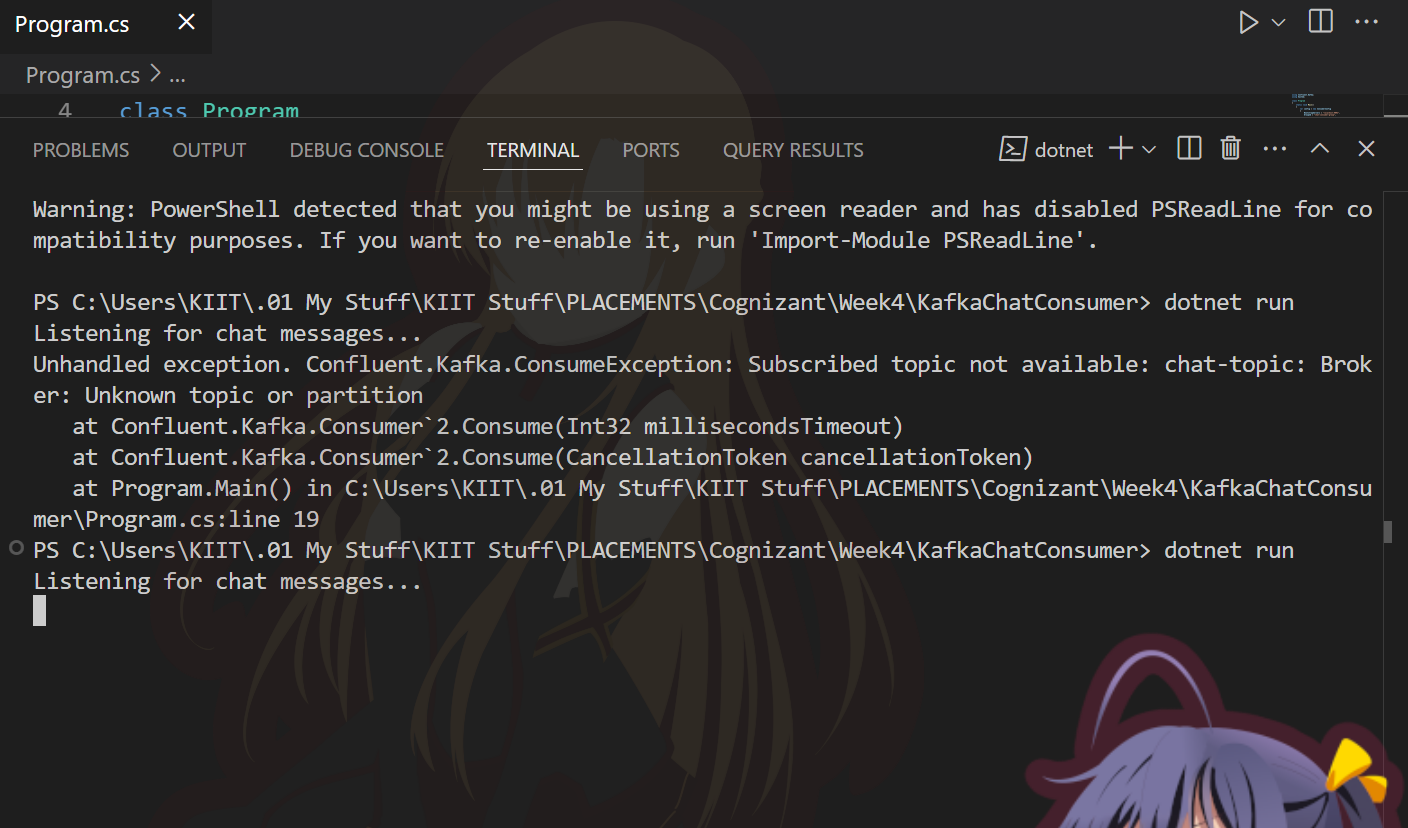
**Input:**  
**Kafka Chat Application: Step-by-Step Commands**  
  
**1.  
Start ZooKeeper**  
cd C:\kafka\_2.13-3.7.2  
bin\windows\zookeeper-server-start.bat config\zookeeper.properties  
  
**| 2 | Start Kafka Broker |**  
cd C:\kafka\_2.13-3.7.2  
bin\windows\kafka-server-start.bat config\server.properties  
  
**| 3 | Create Kafka Topic (if not already created) |**  
cd C:\kafka\_2.13-3.7.2  
bin\windows\kafka-topics.bat --create --topic chat-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1  
 **| 4 | Run Kafka Consumer Console App |**  
cd path\_to\_KafkaChatConsumer\_project  
dotnet run  
 **| 5 | Run Kafka Producer Console App |**  
cd path\_to\_KafkaChatProducer\_project  
dotnet run  
  
**| 6 | Send and Receive Messages |**  
Type chat messages in the producer console and see them appear in the consumer console.

**Output:**









**QUESTION - 2 (JWT -1)**

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

**INPUT:**

**CODE:**

**Program.cs**  
using Microsoft.AspNetCore.Authentication.JwtBearer;  
using Microsoft.IdentityModel.Tokens;  
using Microsoft.OpenApi.Models;  
using System.Text;  
  
var builder = WebApplication.CreateBuilder(args);  
  
  
builder.Services.AddControllers();  
  
  
builder.Services.AddAuthentication(options =>  
{  
options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;  
options.DefaultChallengeScheme = 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();  
  
  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen(c =>  
{  
c.SwaggerDoc("v1", new OpenApiInfo { Title = "JWTToken\_Auth\_API", Version = "v1" });  
  
c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme  
{  
Name = "Authorization",  
Type = SecuritySchemeType.ApiKey,  
Scheme = "Bearer",  
BearerFormat = "JWT",  
In = ParameterLocation.Header,  
Description = "JWT Authorization header using the Bearer scheme. \r\n\r\nEnter 'Bearer' [space] and then your token in the text input below.\r\n\r\nExample: \"Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...\""  
});  
  
c.AddSecurityRequirement(new OpenApiSecurityRequirement  
{  
{  
new OpenApiSecurityScheme  
{  
Reference = new OpenApiReference  
{  
Type = ReferenceType.SecurityScheme,  
Id = "Bearer"  
}  
},  
new string[] {}  
}  
});  
});  
  
var app = builder.Build();  
  
  
app.UseSwagger();  
app.UseSwaggerUI();  
  
app.UseHttpsRedirection();  
  
app.UseAuthentication();  
app.UseAuthorization();  
  
app.MapControllers();  
  
app.Run();  
  
Steps to Document and Follow  
Create a new ASP.NET Core Web API project.  
Add JWT settings in**appsettings.json:**  
  
   
json  
{  
"Jwt": {  
"Key": "ThisIsASecretKeyForJwtToken1234567890",  
"Issuer": "MyAuthServer",  
"Audience": "MyApiUsers",  
"DurationInMinutes": 60  
}  
}  
  
Replace Program.cs with the above compiled code.

Create a LoginModel class with Username and Password properties.

**LoginModel.cs**  
public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

Create an **AuthController.cs** with a login endpoint that validates user and generates a JWT token.

**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);

return Ok(new { Token = token });

}

return Unauthorized();

}

private bool IsValidUser(LoginModel model)

{

// Demo: hardcoded user. Replace with DB check in real apps.

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

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username)

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("ThisIsASecretKeyForJwtToken1234567890"));

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);

}

}

Create a**SecureController** with an [Authorize] attribute to protect endpoints.

**SecureController.cs**

**using Microsoft.AspNetCore.Authorization;**

**using Microsoft.AspNetCore.Mvc;**

**[ApiController]**

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

**public class SecureController : ControllerBase**

**{**

**[HttpGet]**

**[Authorize]**

**public IActionResult GetSecret()**

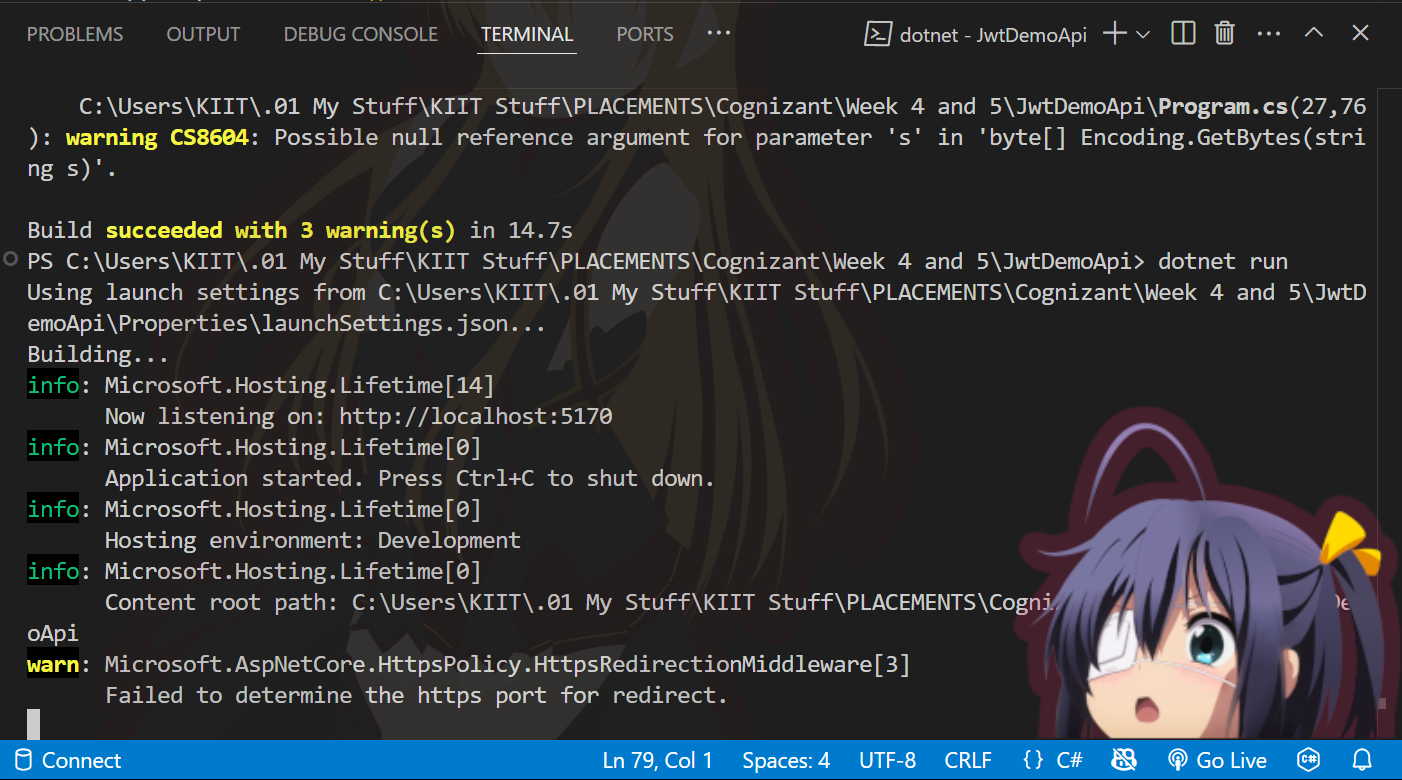
**{**

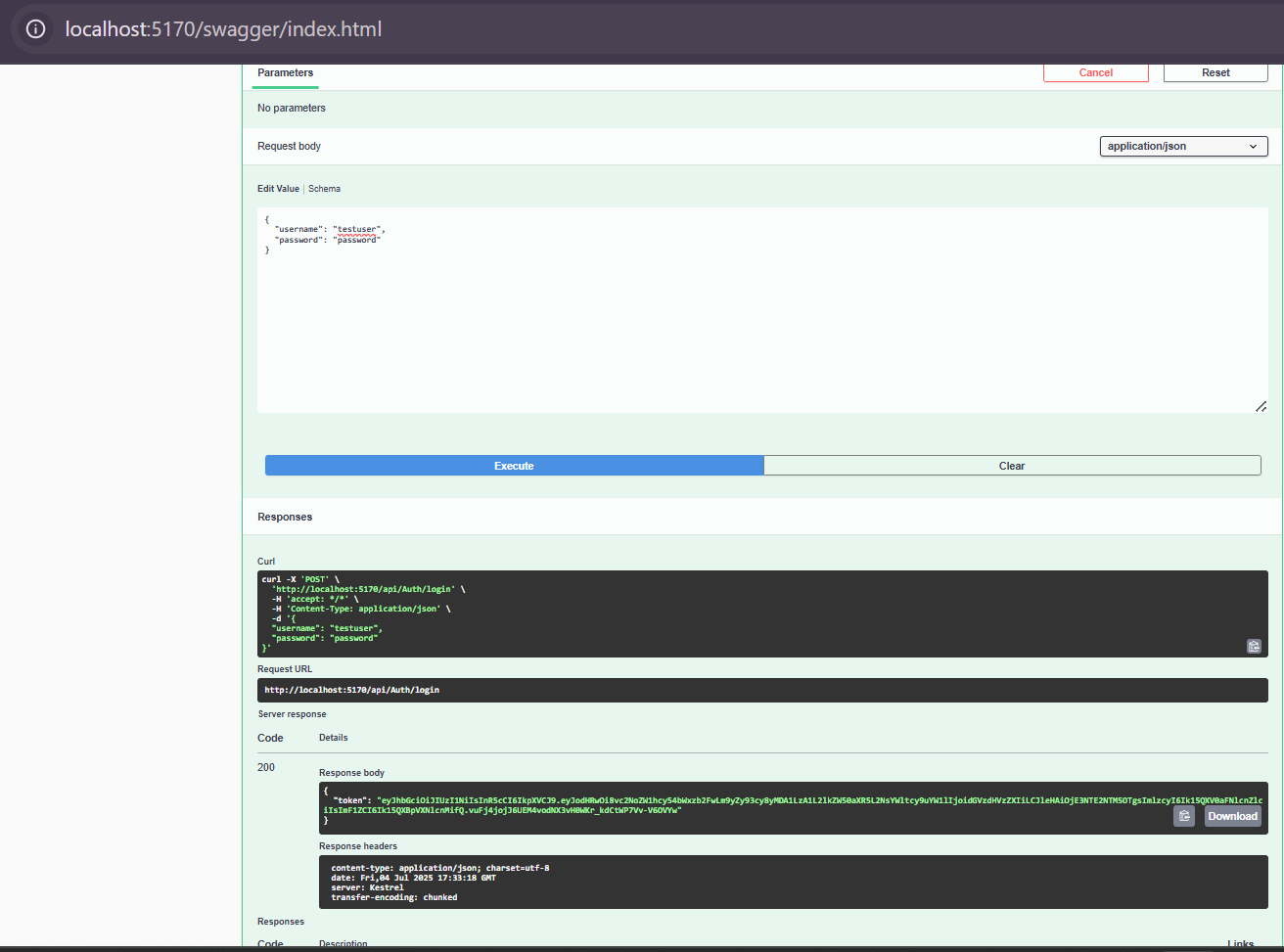
**return Ok("This is a protected endpoint!");**

**}**

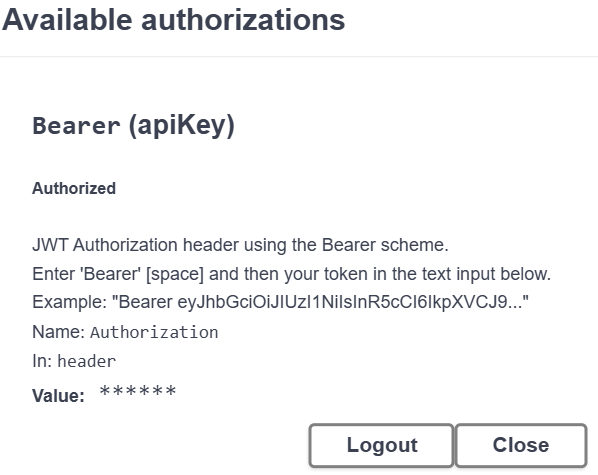
**}**

**OUTPUT:**  
Run the project and use Swagger UI at /swagger to test login and protected endpoints.





Use the Authorize button in Swagger UI to input the JWT token for authenticated requests.

****