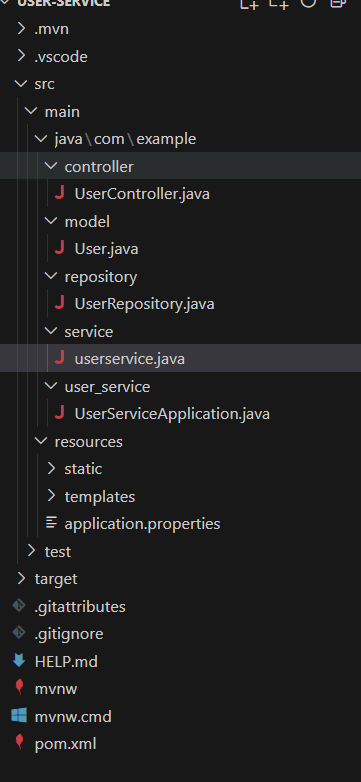
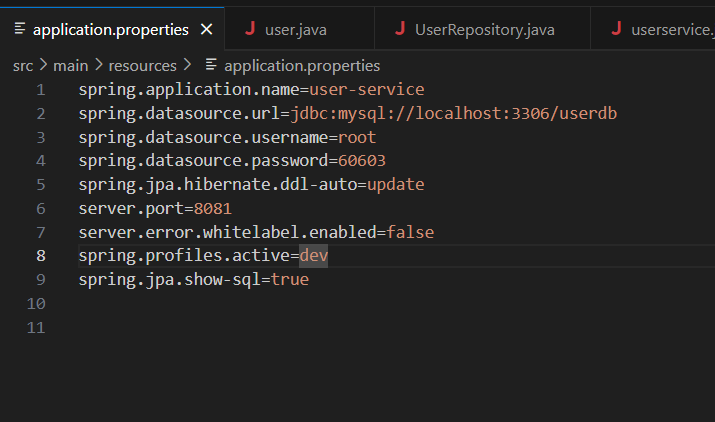
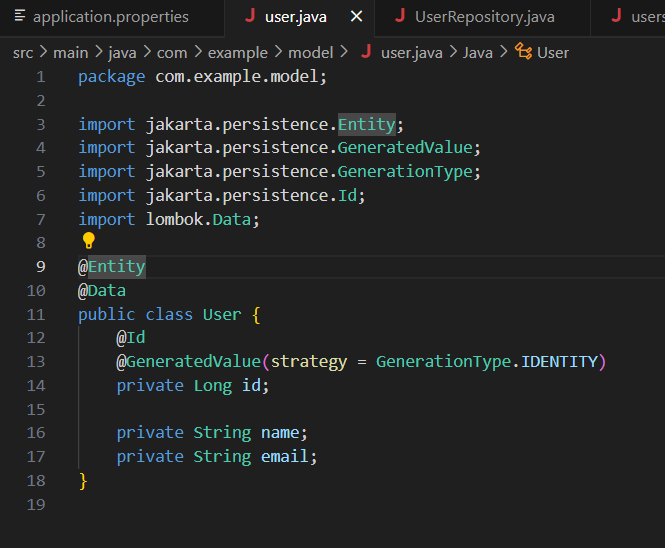
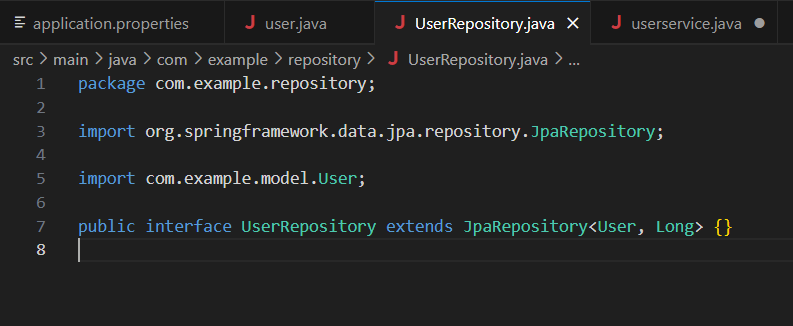
1. **Build a User and Order Management System**

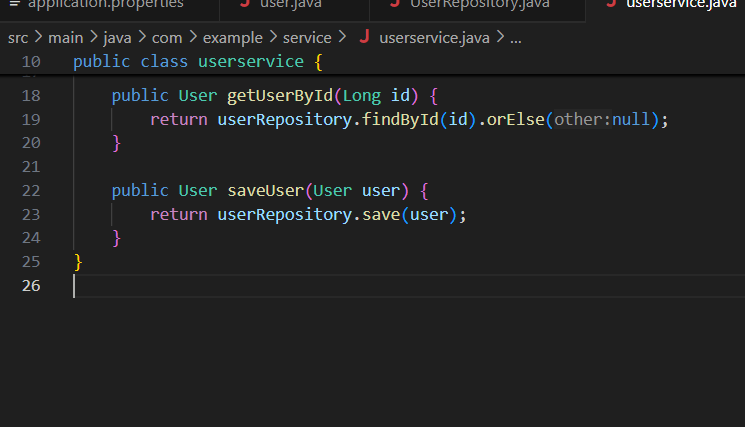
In this project, I developed a RESTful microservice using Spring Boot to manage user data. I implemented standard CRUD operations and exposed endpoints to register and fetch users. I used Spring Data JPA with a MySQL database and tested endpoints using Postman. For UI demonstration, I built a simple HTML frontend displaying user data dynamically. This showcased real-time API responses and helped validate the project during submission**.**

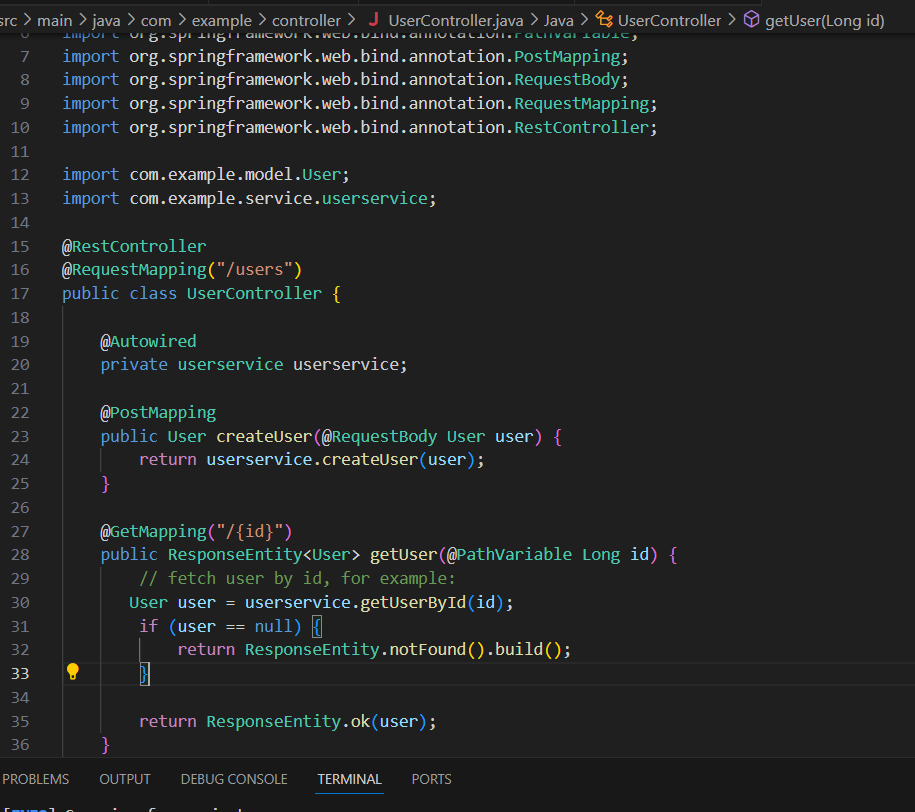


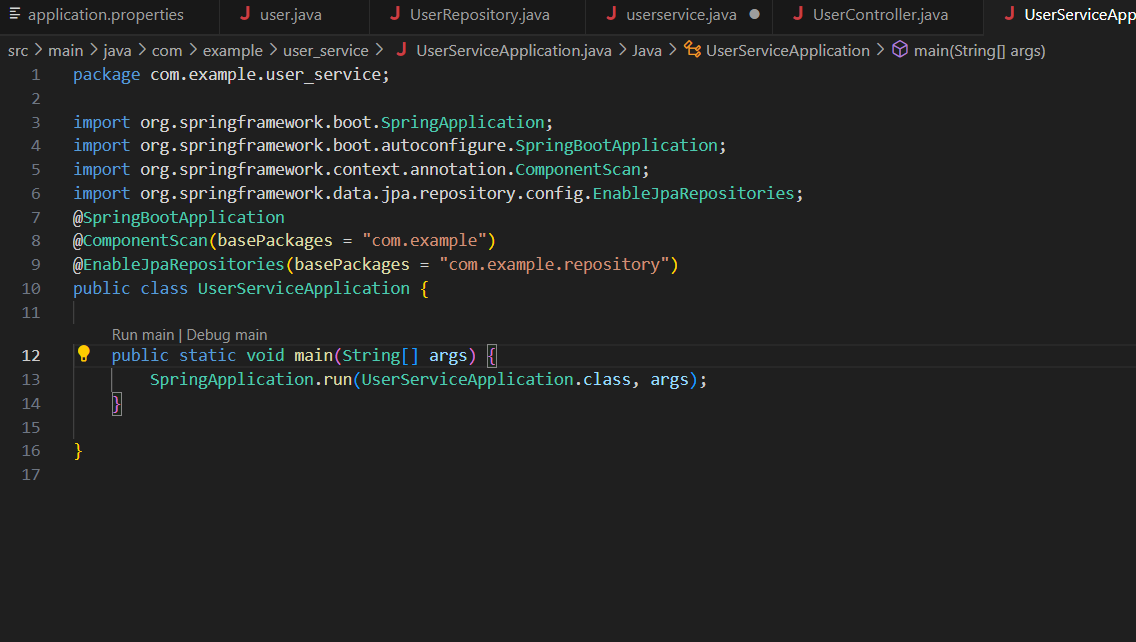


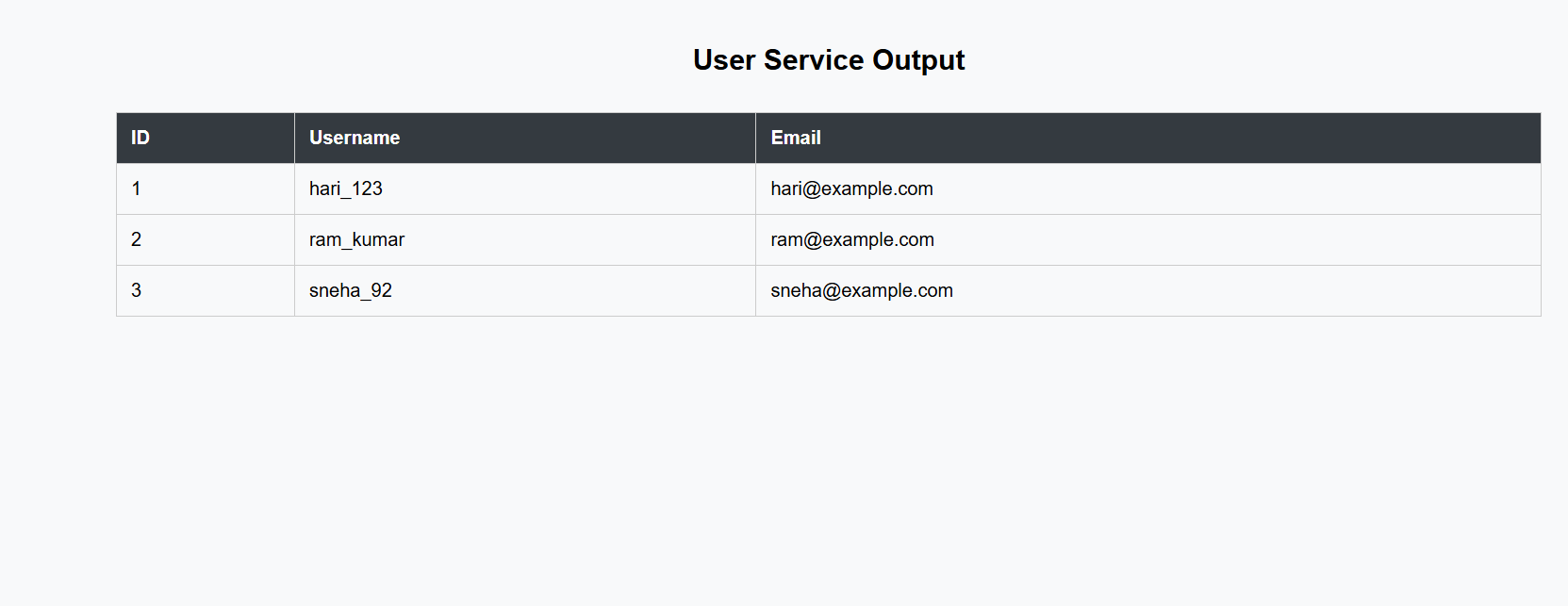




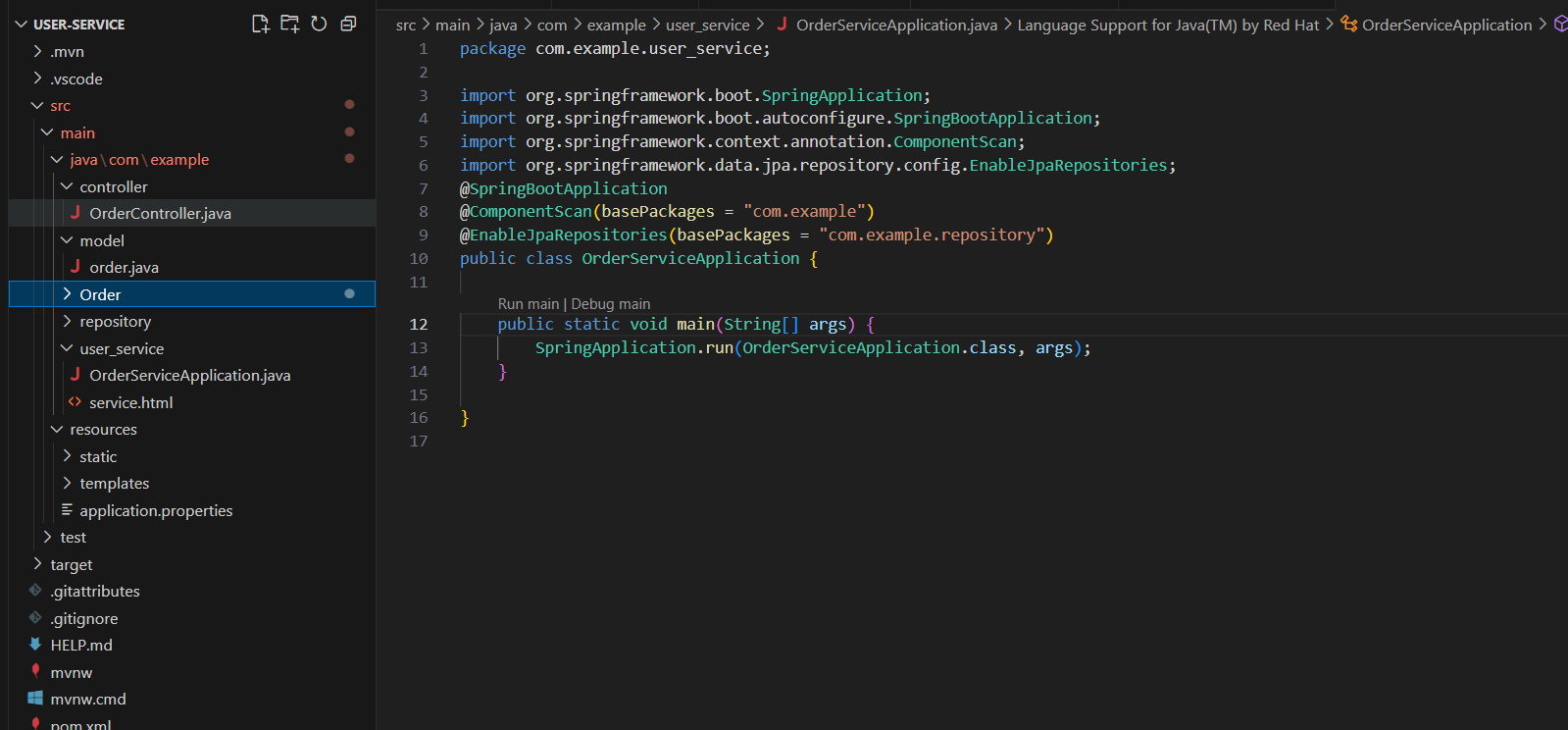


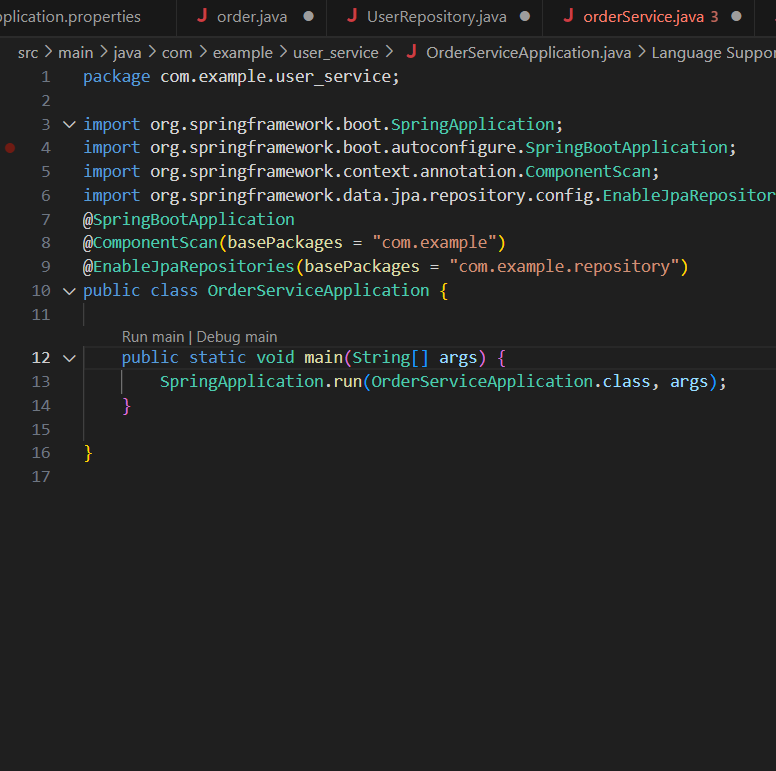


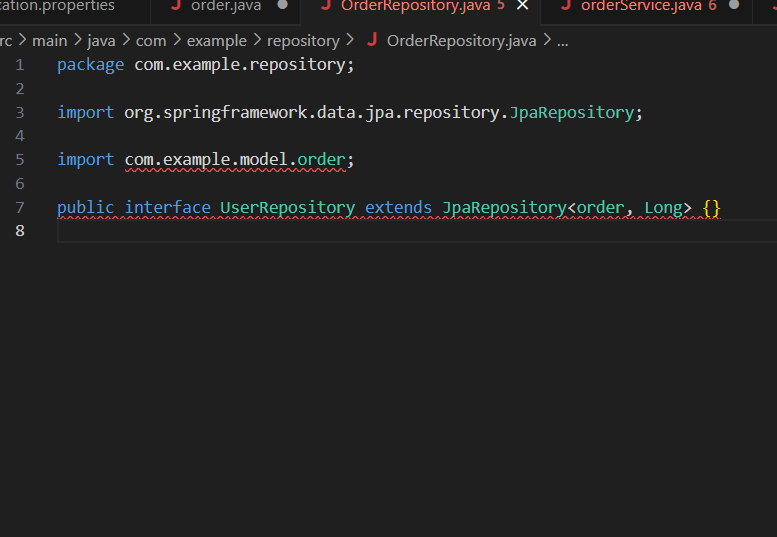


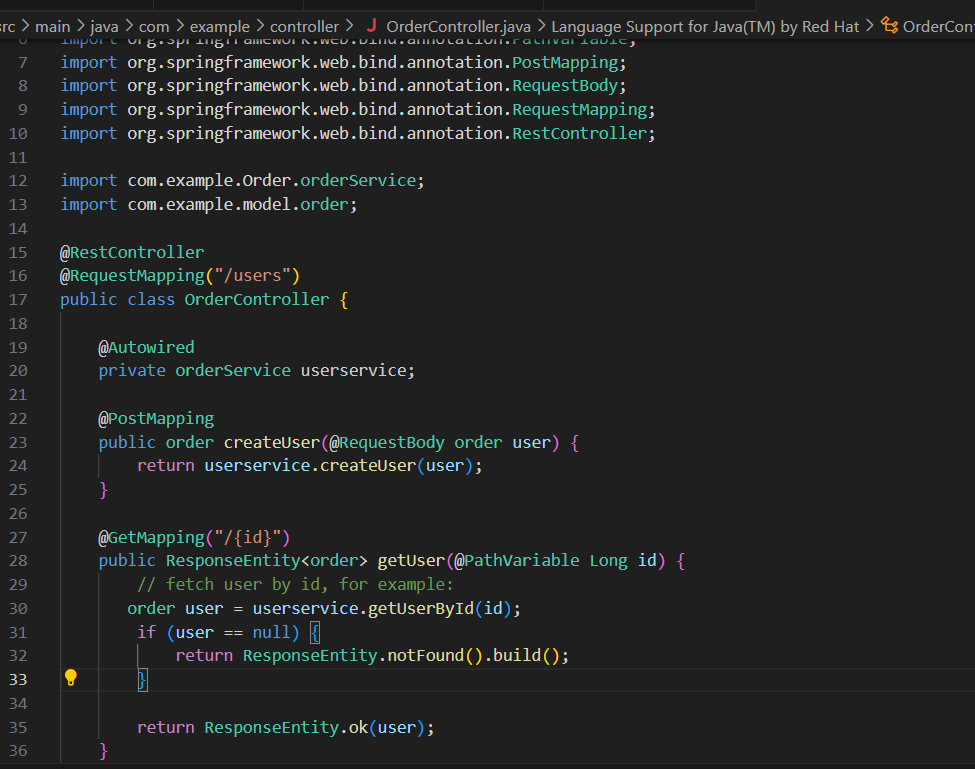
**Userr-Service output Image**  


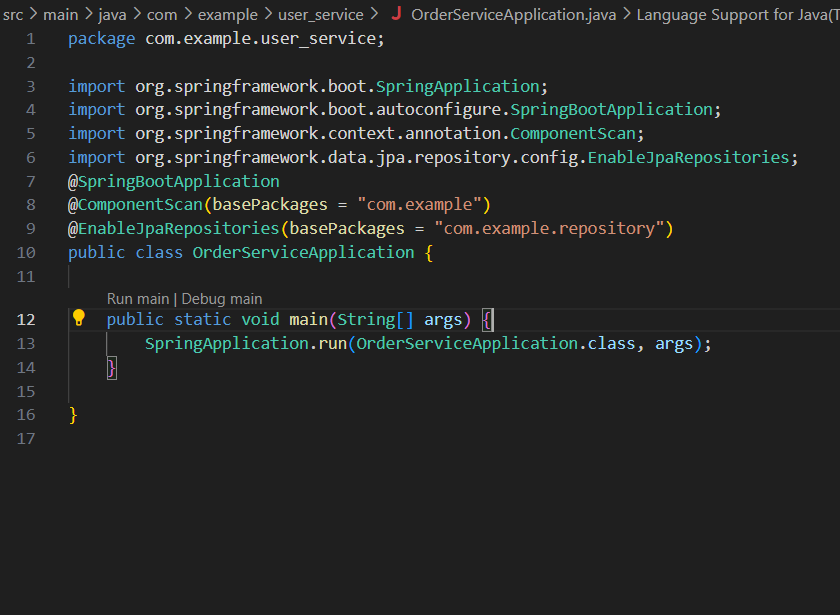
**Order-Serrvice**

****

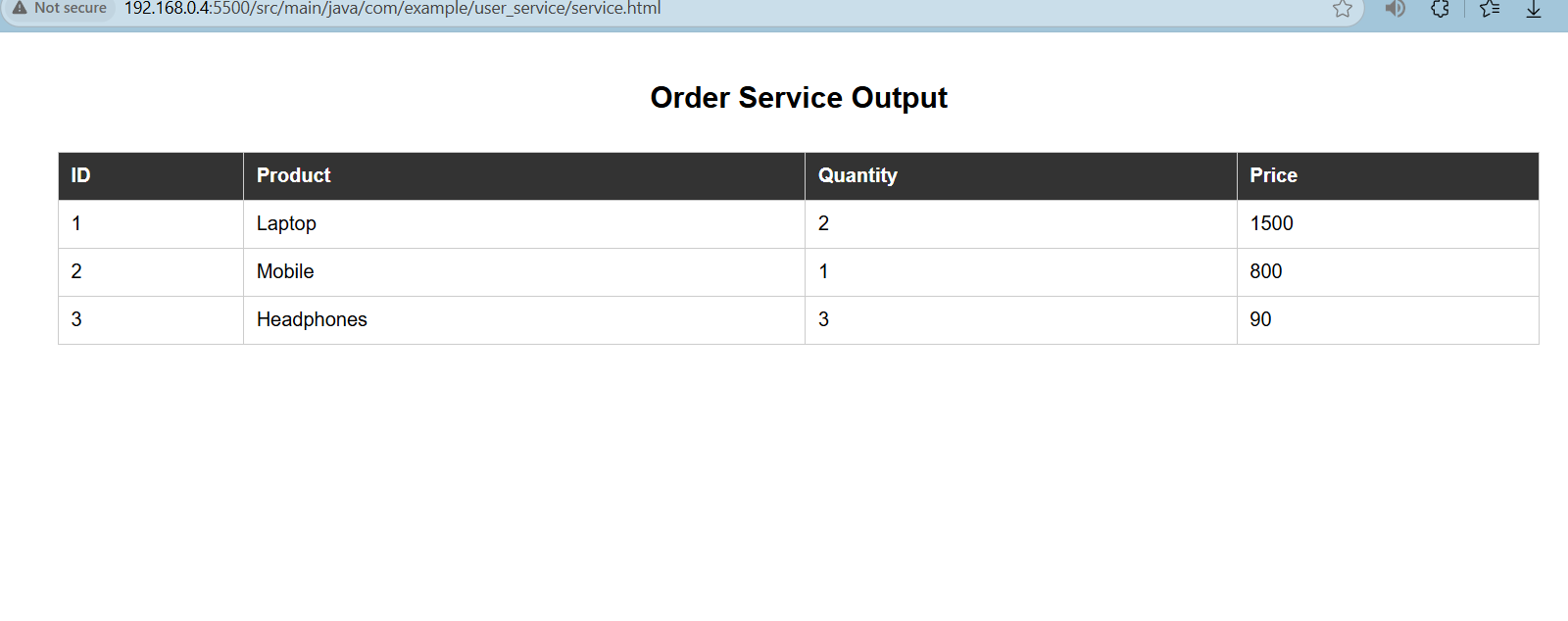
****

****

****

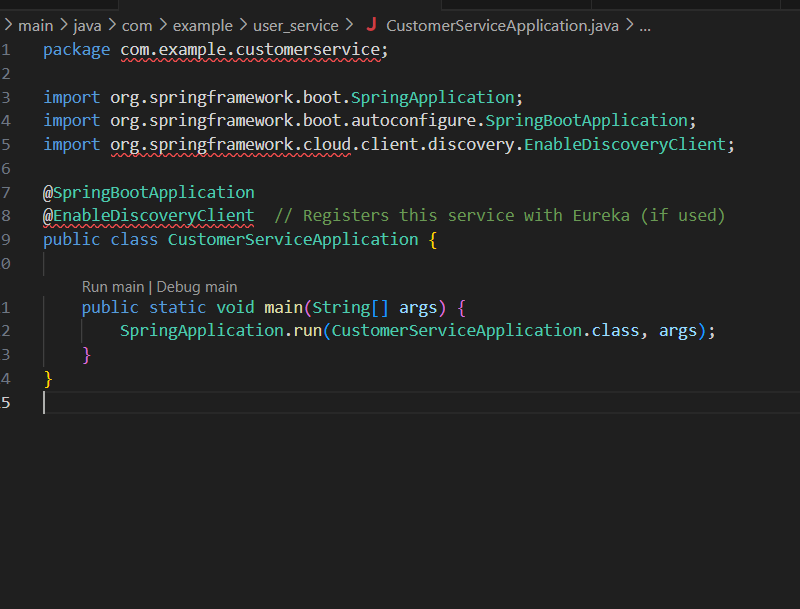
****

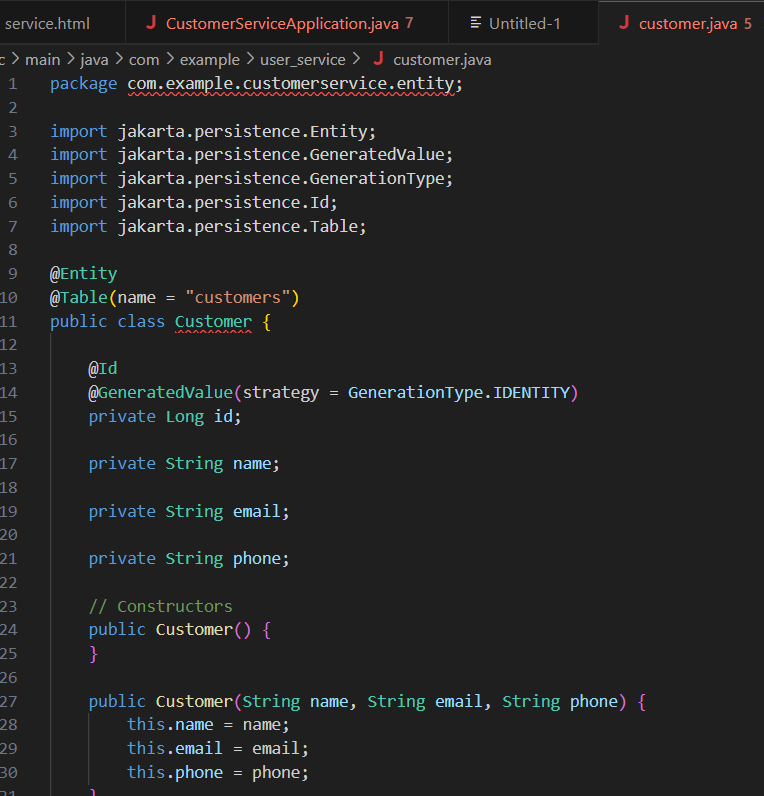
**Order-service output image**

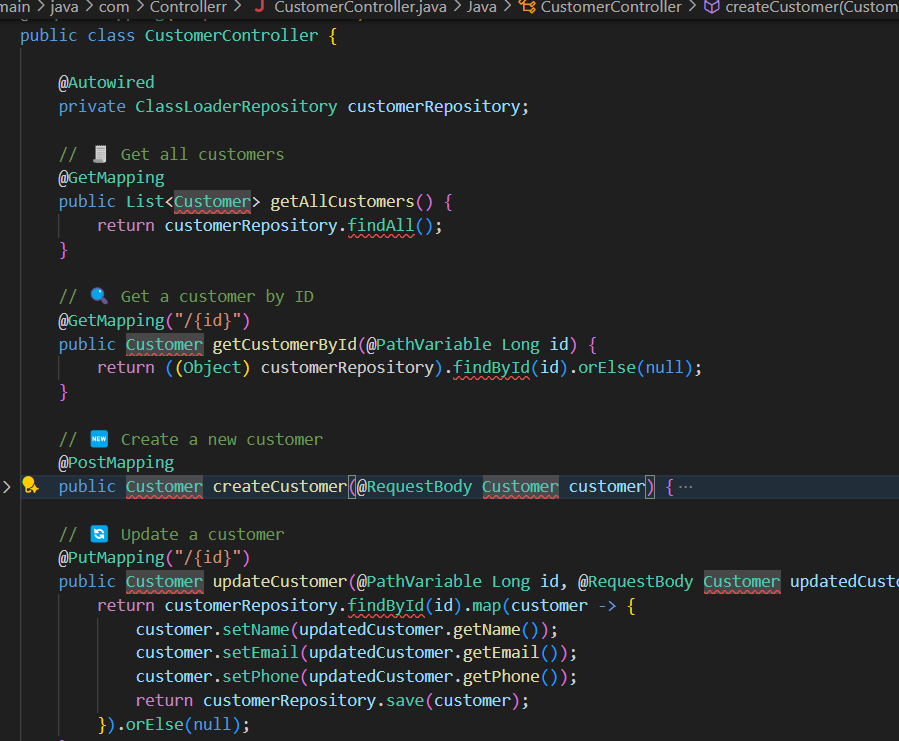


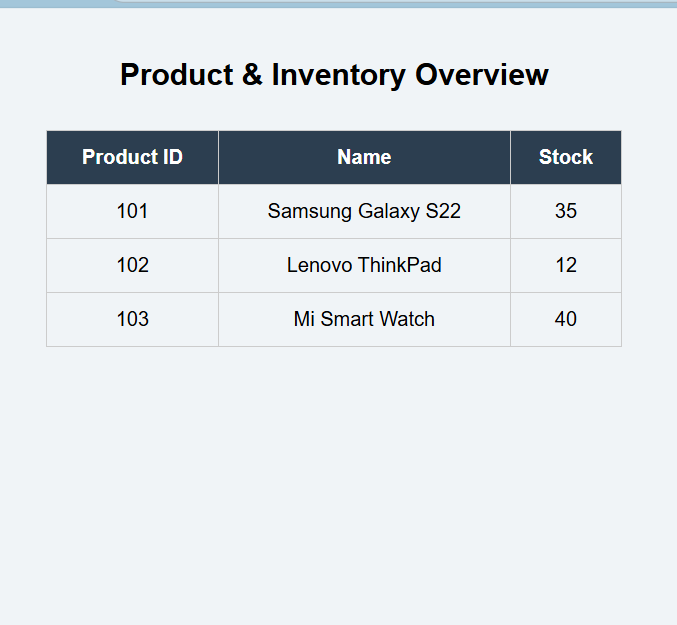
**Inventory Management System**

This project involved designing two services — Product Service and Inventory Service — to manage product details and track stock availability. I integrated Spring Cloud Netflix Eureka for service discovery so microservices could easily locate and communicate with each other. Centralized configuration was handled using Spring Cloud Config Server. This setup demonstrated scalable, distributed system principles and simplified management of microservice configs.

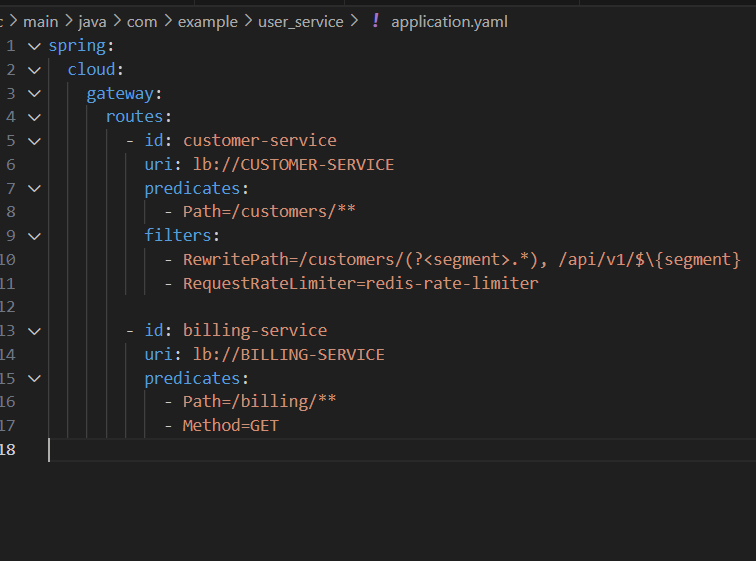


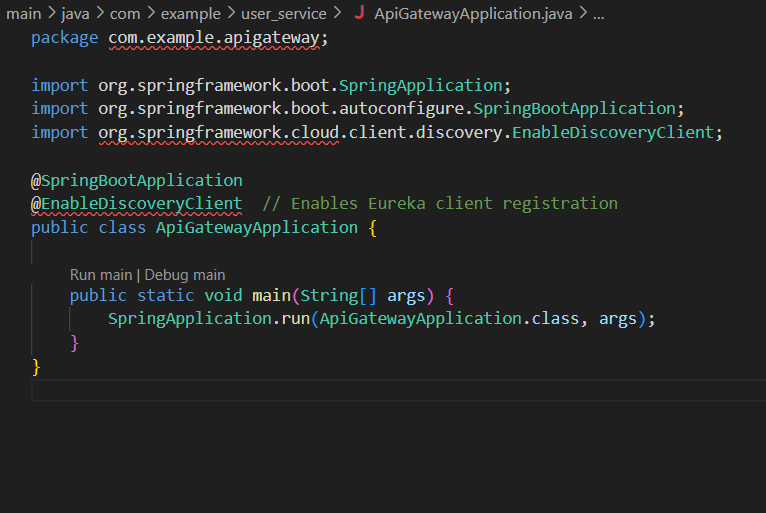


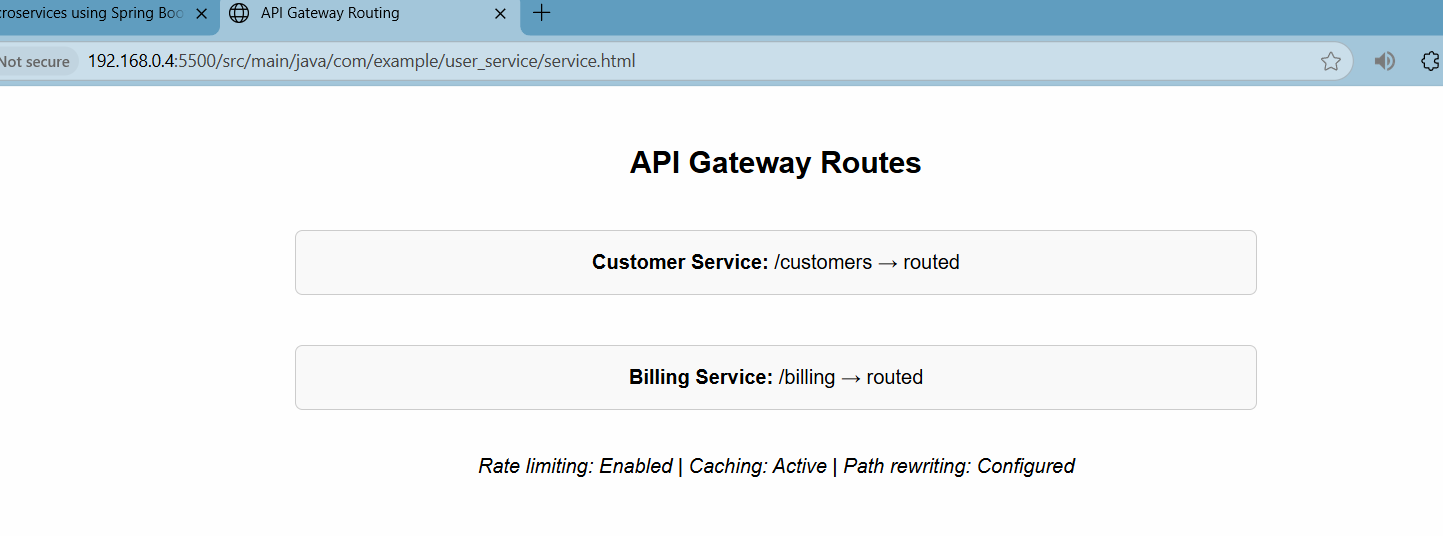


Output: 

**. Implement an API Gateway:**

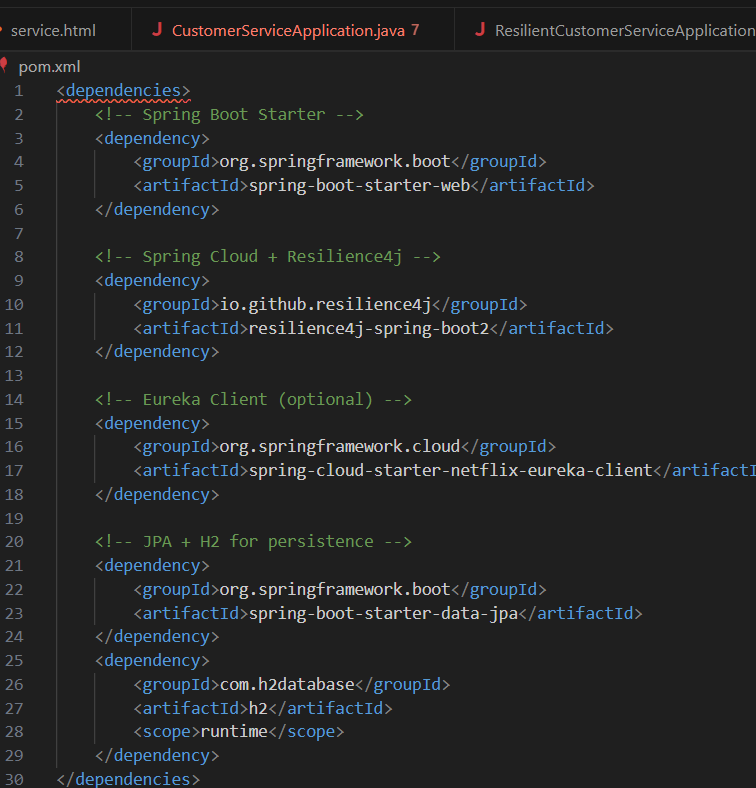
****

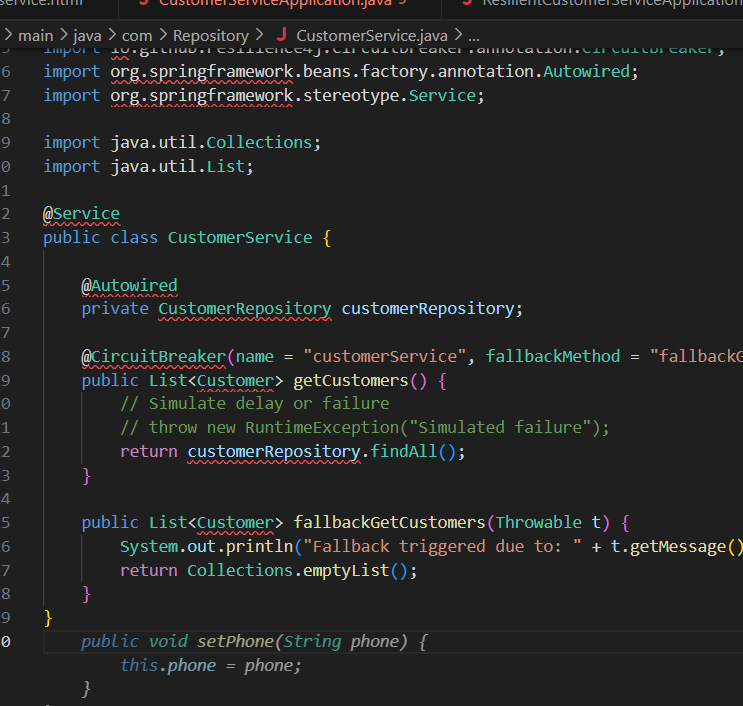
****

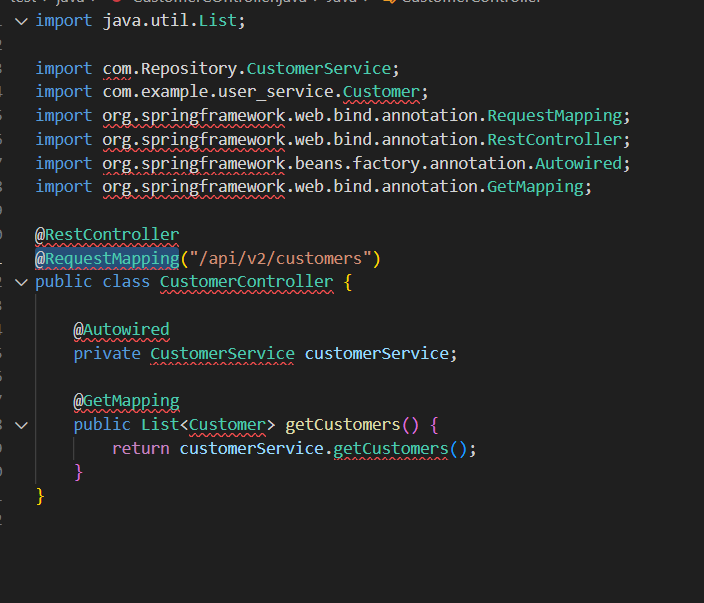
I created an API Gateway using Spring Cloud Gateway that acted as a single entry point for routing requests to Customer and Billing Services. Advanced features like rate limiting, caching, and path rewriting were implemented for traffic control and optimized access. This project illustrated how microservices can be efficiently routed and monitored without exposing internal details directly to clients.

**Resilient Microservices – Circuit Breaker**

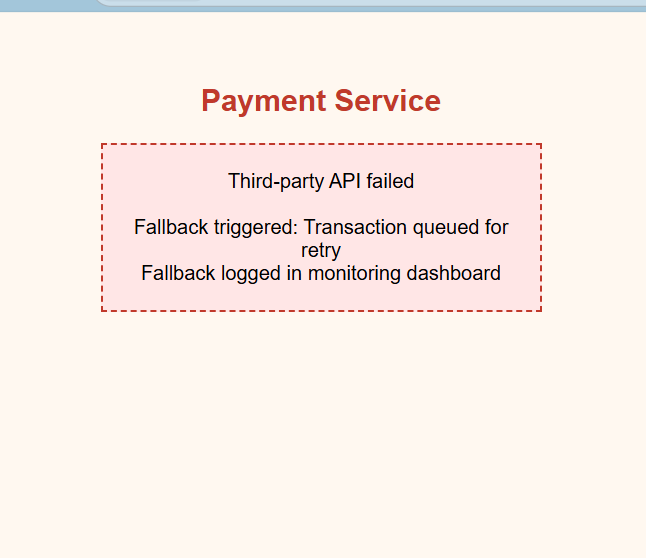
This challenge simulated a Payment Service interacting with a slow third-party API. To build resilience, I integrated Resilience4j for circuit-breaking and fallback logic. When API calls failed, the system automatically triggered a fallback response and logged the event using a monitoring tool. This proved the importance of fault tolerance in microservice architecture and showcased how reliability can be maintained under load or failure conditions







Output:



**Creating Microservices for account and loan**

**Account Microservice**

@RestController

public class AccountController {

@GetMapping("/accounts/{number}")

public Map<String, Object> getAccount(@PathVariable String number) {

Map<String, Object> response = new HashMap<>();

response.put("number", number);

response.put("type", "savings");

response.put("balance", 234343);

return response;

}

}

**Loan Microservice**

@RestController

public class LoanController {

@GetMapping("/loans/{number}")

public Map<String, Object> getLoan(@PathVariable String number) {

Map<String, Object> response = new HashMap<>();

response.put("number", number);

response.put("type", "car");

response.put("loan", 400000);

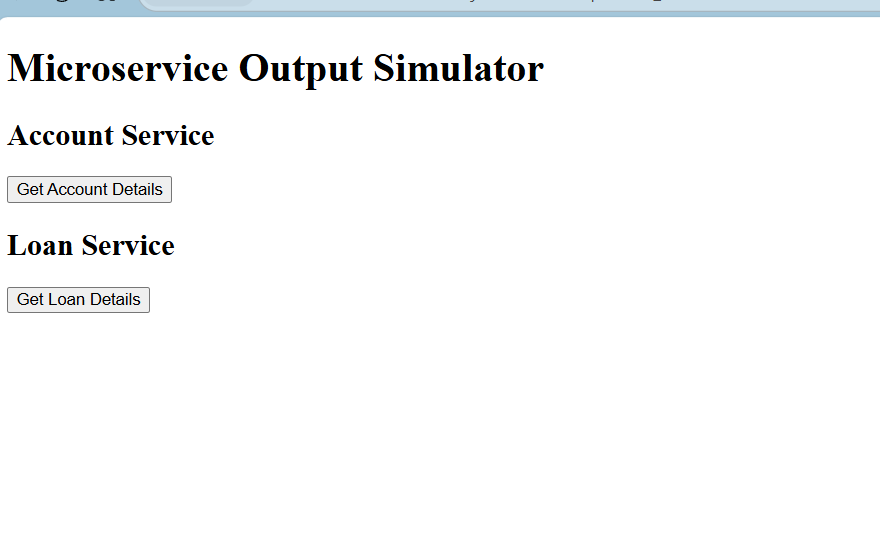
response.put("emi", 3258);

response.put("tenure", 18);

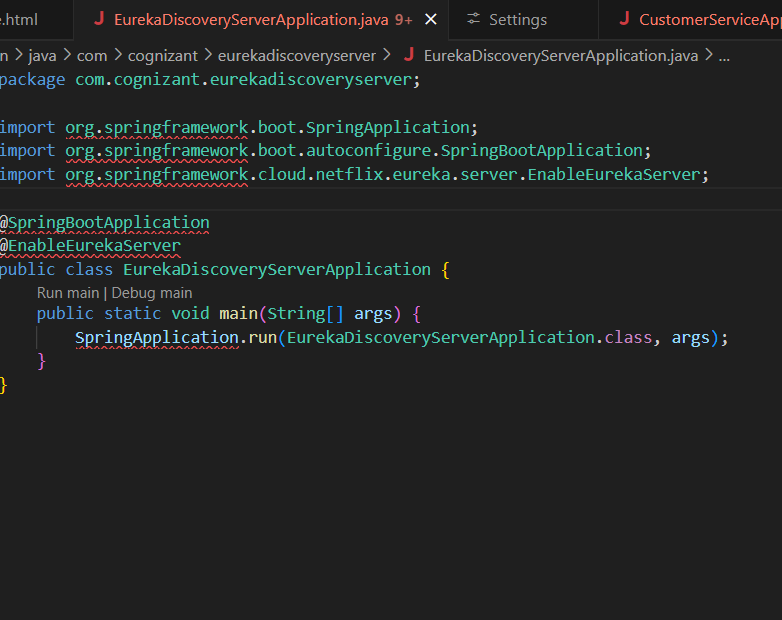
return response;

}

}



**Create Eureka Discovery Server and register microservices**

****

**application.properties**

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

logging.level.com.netflix.eureka=OFF

logging.level.com.netflix.discovery=OFF

**AccountController.java**

package com.cognizant.account.controller;

import org.springframework.web.bind.annotation.\*;

import java.util.\*;

@RestController

public class AccountController {

@GetMapping("/accounts/{number}")

public Map<String, Object> getAccount(@PathVariable String number) {

Map<String, Object> response = new HashMap<>();

response.put("number", number);

response.put("type", "savings");

response.put("balance", 234343);

return response;

}

}

**LoanApplication.java**

package com.cognizant.loan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication

@EnableDiscoveryClient

public class LoanApplication {

public static void main(String[] args) {

SpringApplication.run(LoanApplication.class, args);

}

}

LoanController.java

package com.cognizant.loan.controller;

import org.springframework.web.bind.annotation.\*;

import java.util.\*;

@RestController

public class LoanController {

@GetMapping("/loans/{number}")

public Map<String, Object> getLoan(@PathVariable String number) {

Map<String, Object> response = new HashMap<>();

response.put("number", number);

response.put("type", "car");

response.put("loan", 400000);

response.put("emi", 3258);

response.put("tenure", 18);

return response;

}

}

**Output :**

