**Superset ID:6373322**

**COGNIZANT DIGITAL NURTURE 4.0 JAVA FSE**

**Week- 5 : Microservices with API gateway**

**Creating Microservices for account and loan**

In this hands on exercises, we will create two microservices for a bank. One microservice for handing accounts and one for handling loans. Each microservice will be a specific independent Spring RESTful Webservice maven project having it's own pom.xml. The only difference is that, instead of having both account and loan as a single application, it is split into two different applications. These webservices will be a simple service without any backend connectivity.

Follow steps below to implement the two microservices:

**Account Microservice**

• Create folder with employee id in D: drive

• Create folder named 'microservices' in the new folder created in previous step. This folder will contain all the sample projects that we will create for learning microservices. • Open https://start.spring.io/ in browser

• Enter form field values as specified below:

o Group: com.cognizant

o Artifact: account

• Select the following modules o Developer Tools > Spring Boot DevTools o Web > Spring Web

• Click generate and download the zip file

• Extract 'account' folder from the zip and place this folder in the 'microservices' folder created earlier

• Open command prompt in account folder and build using mvn clean package command

• Import this project in Eclipse and implement a controller method for getting account details based on account number. Refer specification below:

o Method: GET

o Endpoint: /accounts/{number}

o Sample Response. Just a dummy response without any backend connectivity.

{ number: "00987987973432", type: "savings", balance: 234343 }

• Launch by running the application class and test the service in browser

Loan Microservice

• Follow similar steps specified for Account Microservice and implement a service API to get loan account details

o Method: GET

o Endpoint: /loans/{number}

o Sample Response. Just a dummy response without any backend connectivity.

{ number: "H00987987972342", type: "car", loan: 400000, emi: 3258, tenure: 18 }

• Launching this application by having account service already running

• This launch will fail with error that the bind address is already in use

• The reason is that each one of the service is launched with default port number as 8080. Account service is already using this port and it is not available for loan service.

• Include "server.port" property with value 8081 and try launching the application

• Test the service with 8081 port

Now we have two microservices running on different ports.

NOTE: The console window of Eclipse will have both the service console running. To switch between different consoles use the monitor icon within the console view.

**CODE:**

**AccountController.java**

package com.cognizant.account.controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

import java.util.Map;

*@RestController*

public class AccountController {

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

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

return Map.*of*(

"number", number,

"type", "savings",

"balance", 234343

);

}

}

**AccountApplication.java**

package com.cognizant.account;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class AccountApplication {

public static void main(String[] args) {

SpringApplication.run(AccountApplication.class, args);

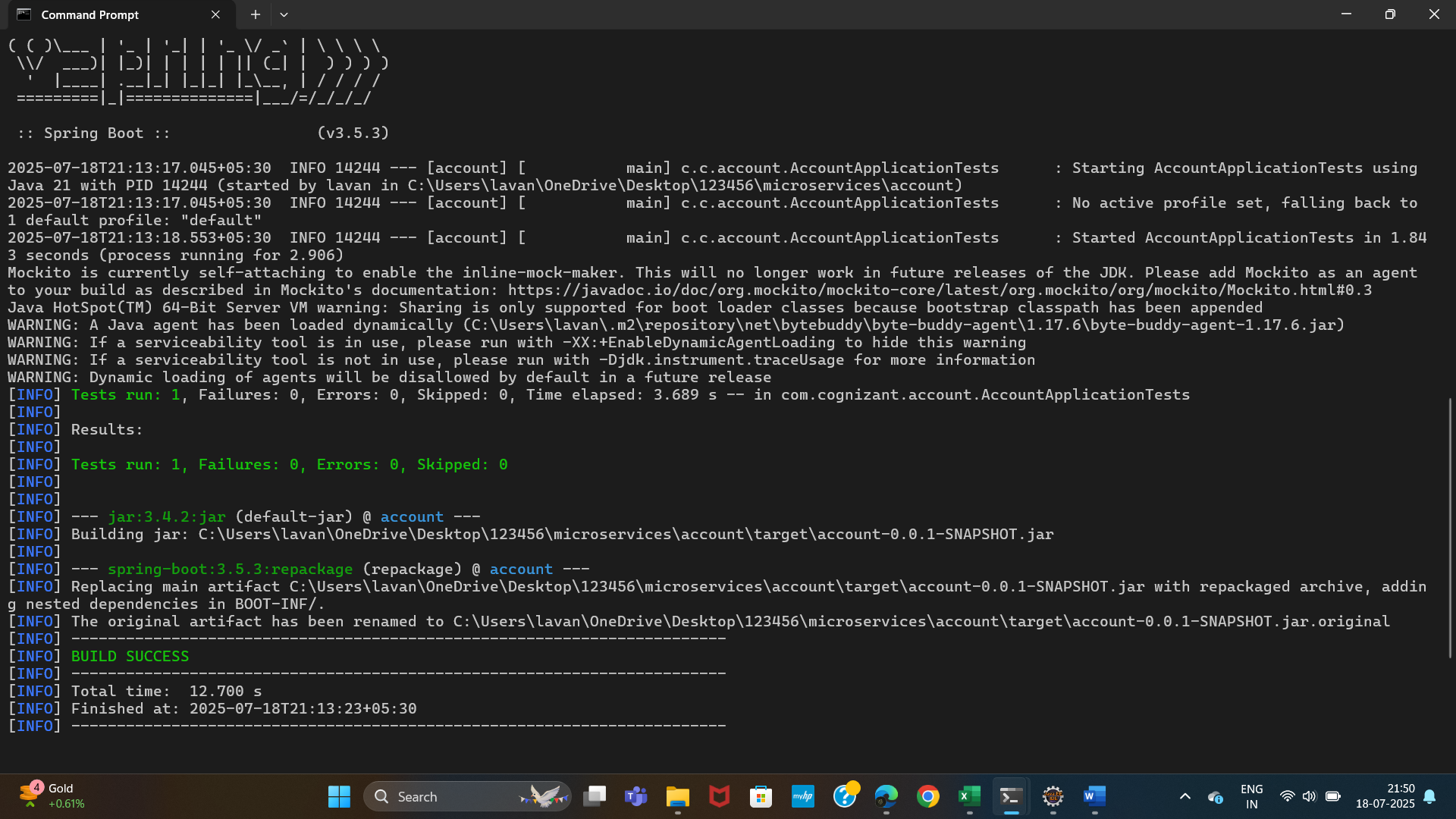
}

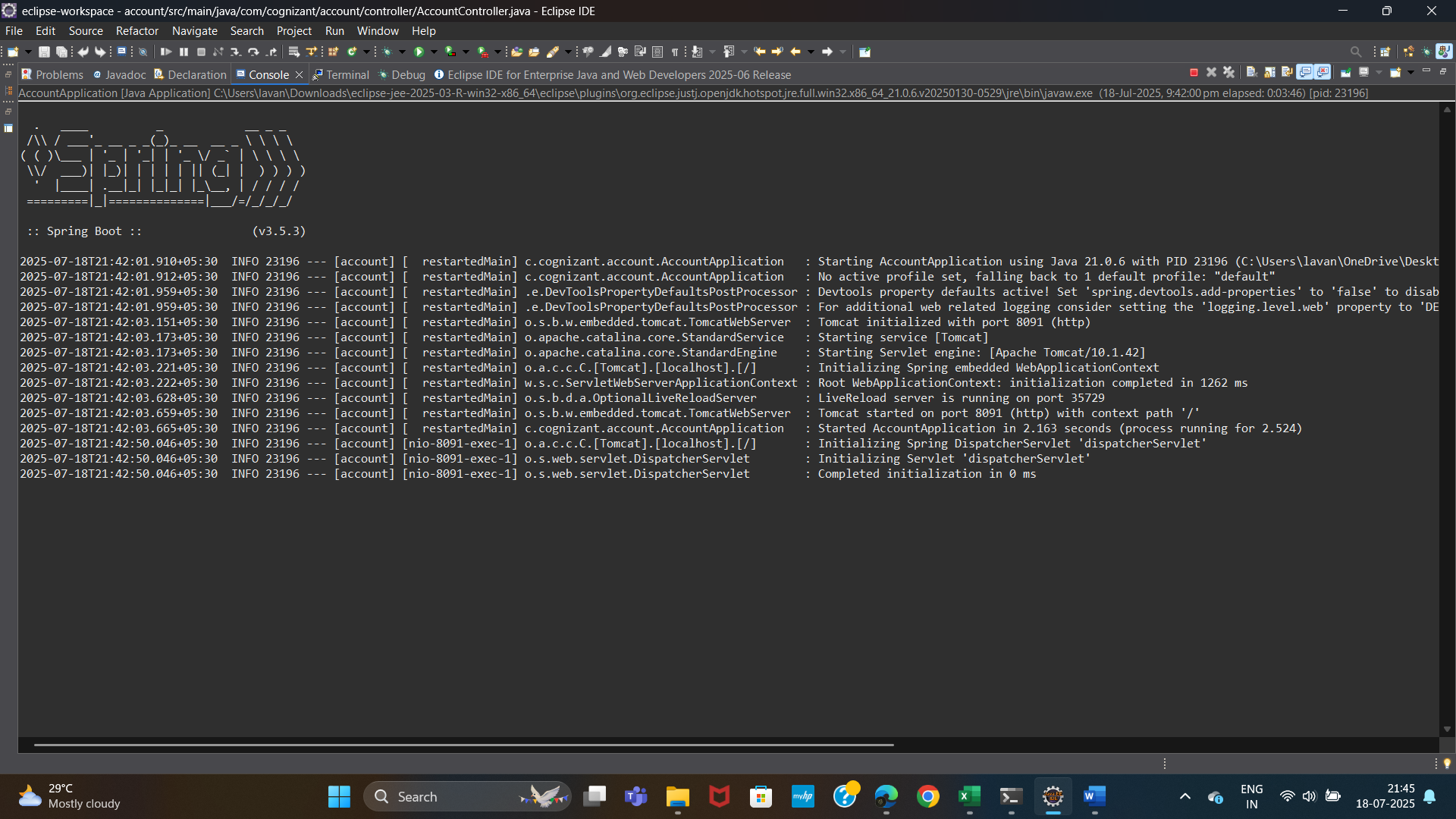
}

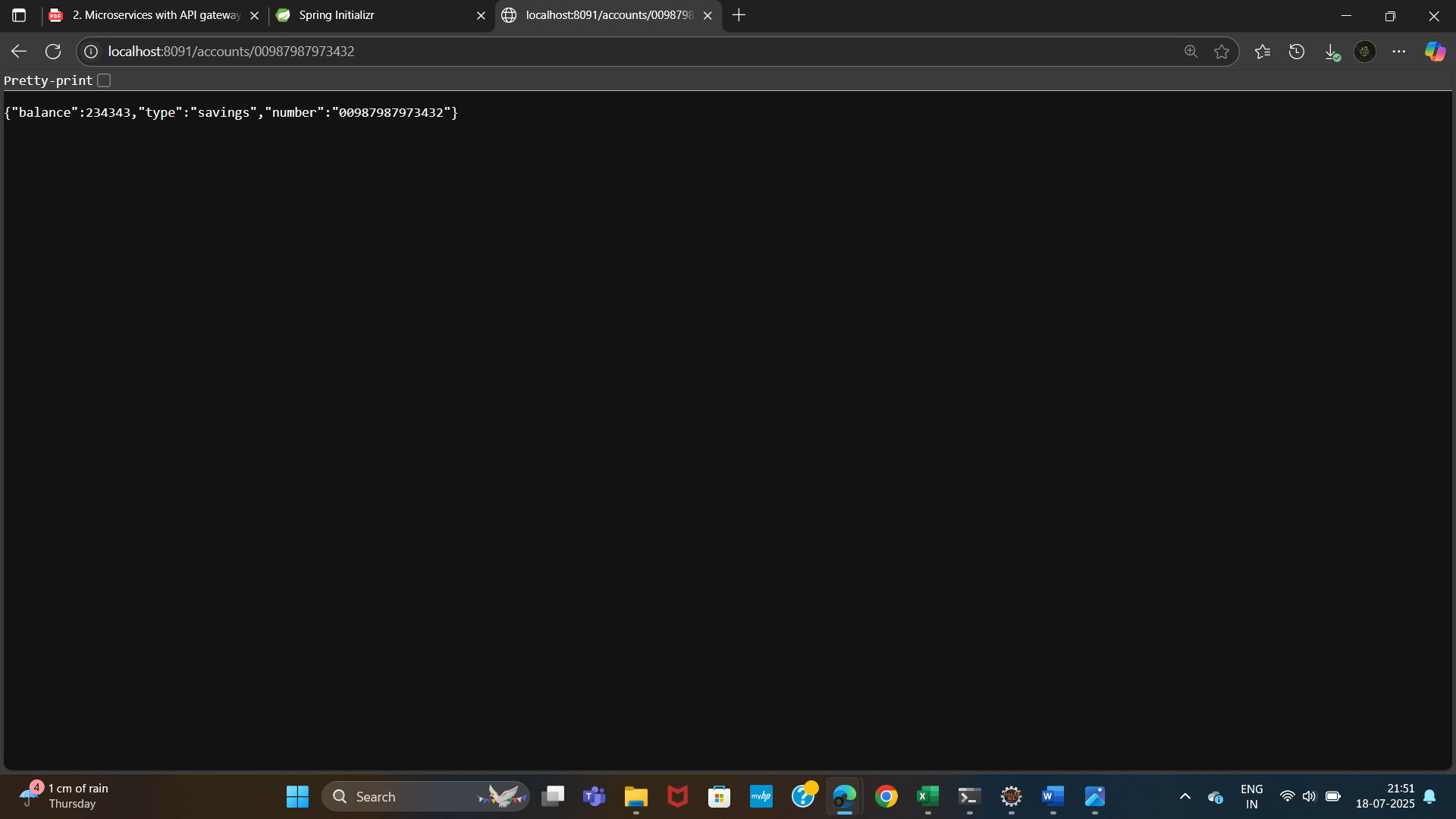
**application.properties**

spring.application.name=account

server.port=8091 //8081 is already in use

**OUTPUT:** 





**LoanController.java**

package com.cognizant.loan.controller;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

import java.util.Map;

*@RestController*

public class LoanController {

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

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

return Map.*of*(

"number", number,

"type", "car",

"loan", 400000,

"emi", 3258,

"tenure", 18

);

}

}

**LoanApplication.java**

package com.cognizant.loan;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LoanApplication {

public static void main(String[] args) {

SpringApplication.run(LoanApplication.class, args);

}

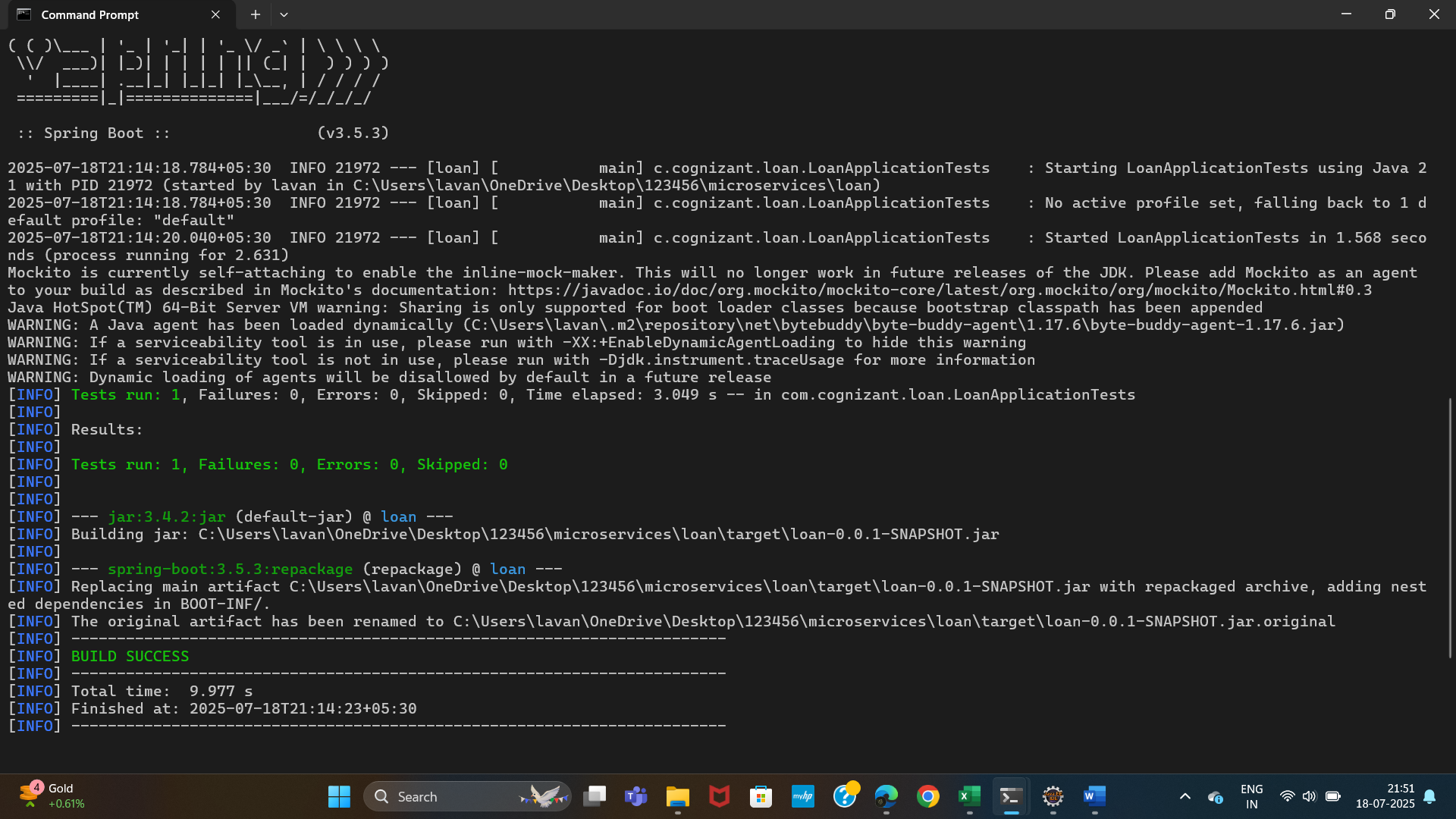
}

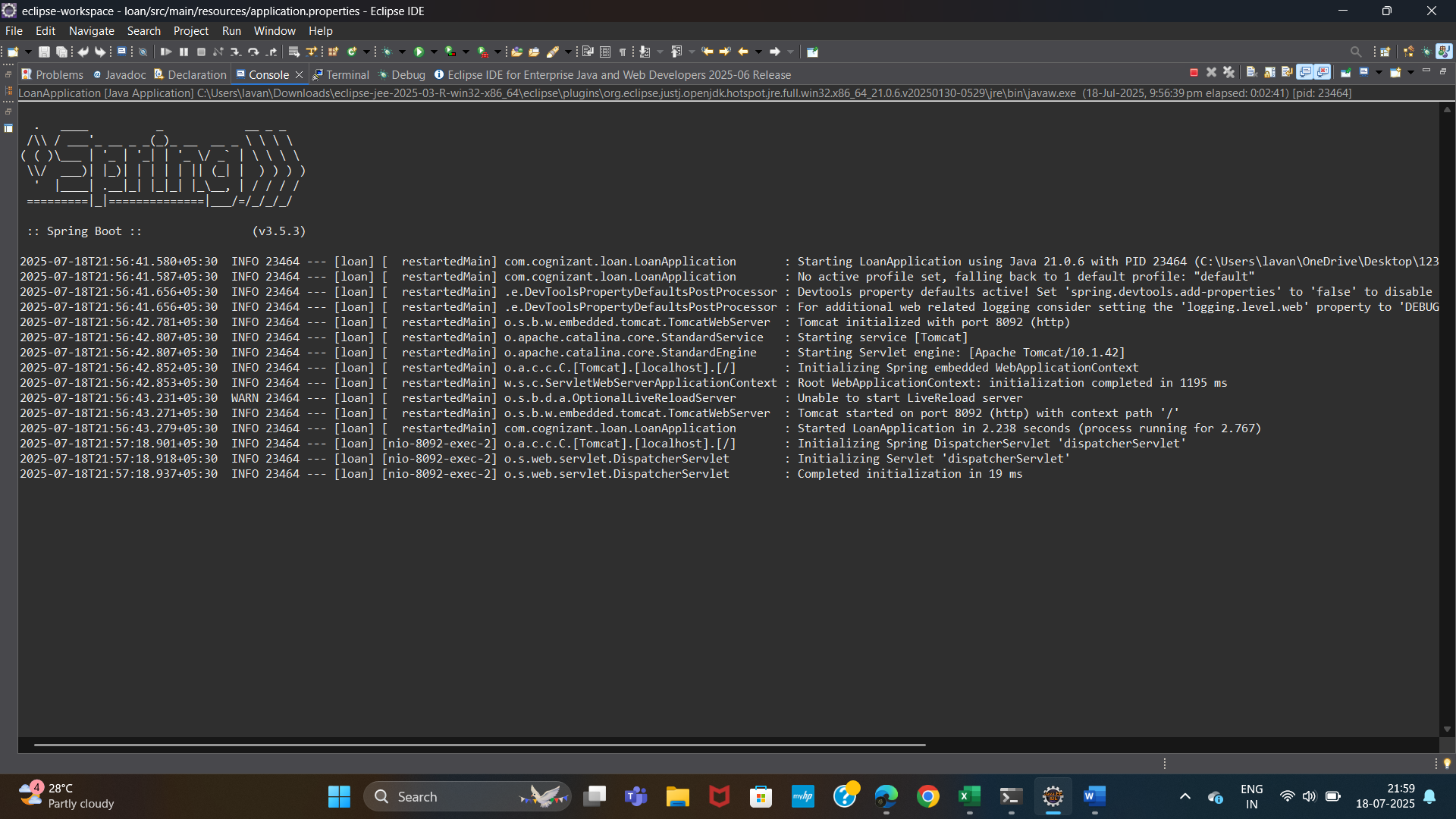
**application.properties**

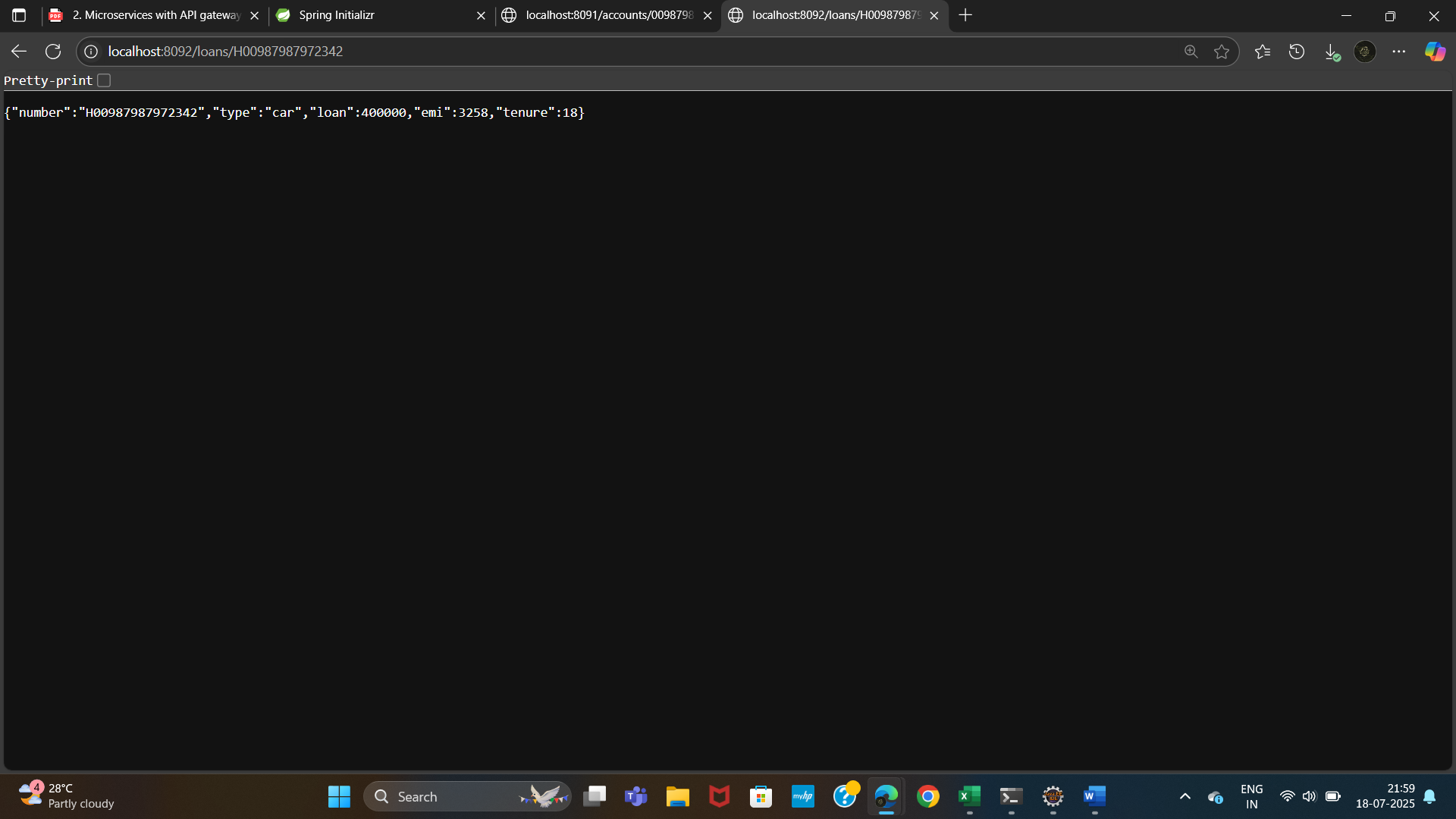
spring.application.name=loan

server.port=8092

**OUTPUT:**







**Create Eureka Discovery Server and register microservices**

Eureka Discovery Server holds a registry of all the services that are available for immediate consumption. Anybody whom wants to consume a RESTful Web Service can come to the discovery server and find out what is available and ready for consumption. Eureka Discovery Server is part of spring cloud module.

**CODE:**

**application.properties**

spring.application.name=eureka-discovery-server

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

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>eureka-discovery-server</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>eureka-discovery-server</name>

<description>Eureka Discovery Server for Microservices</description>

<properties>

<java.version>21</java.version>

<spring-cloud.version>2025.0.0</spring-cloud.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

<!-- Optional: DevTools for hot reload -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<!-- Test dependencies -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

</project>

**EurekaDiscoveryServerApplication.java**

package com.cognizant.eureka\_discovery\_server;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class EurekaDiscoveryServerApplication {

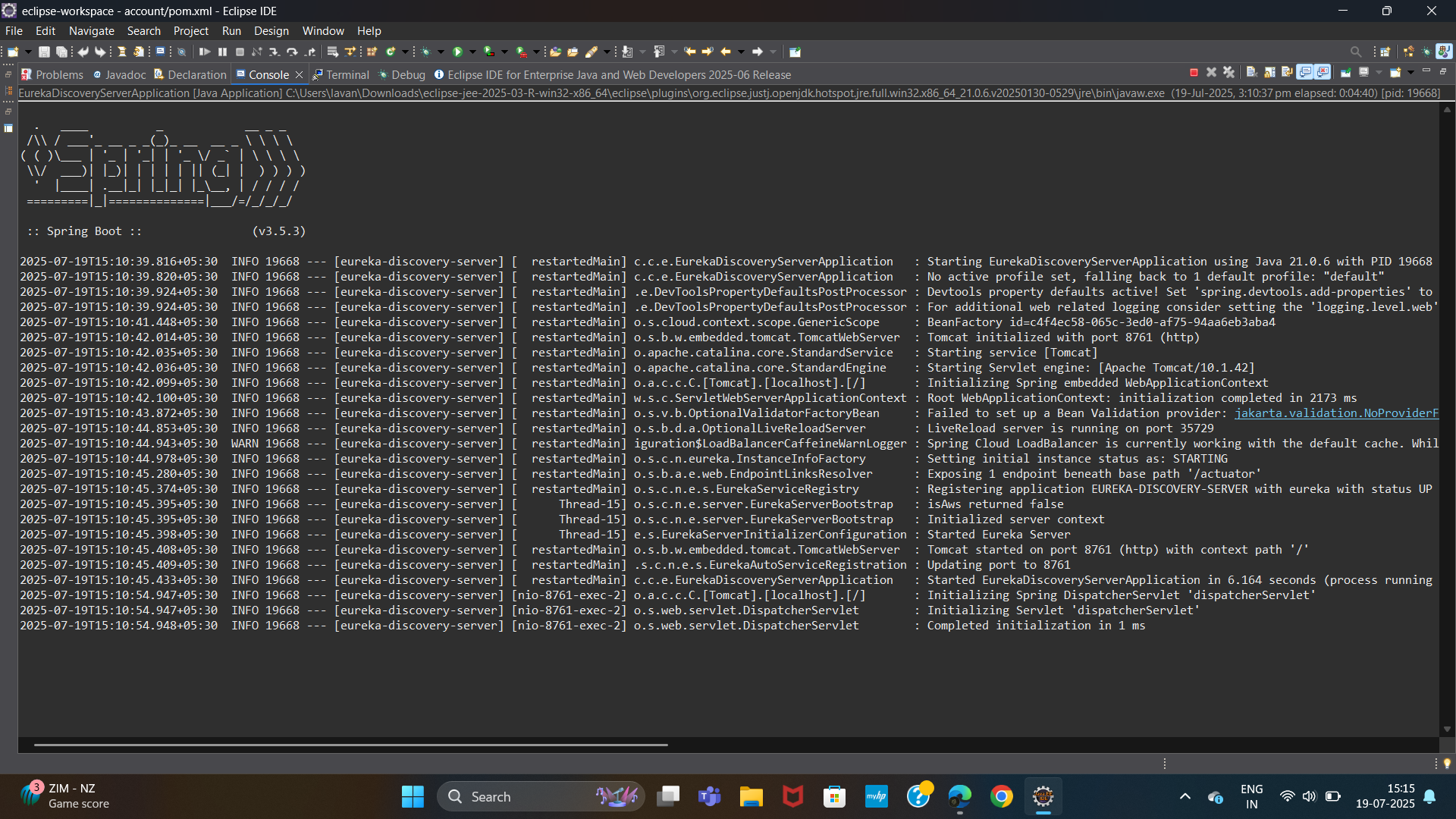
public static void main(String[] args) {

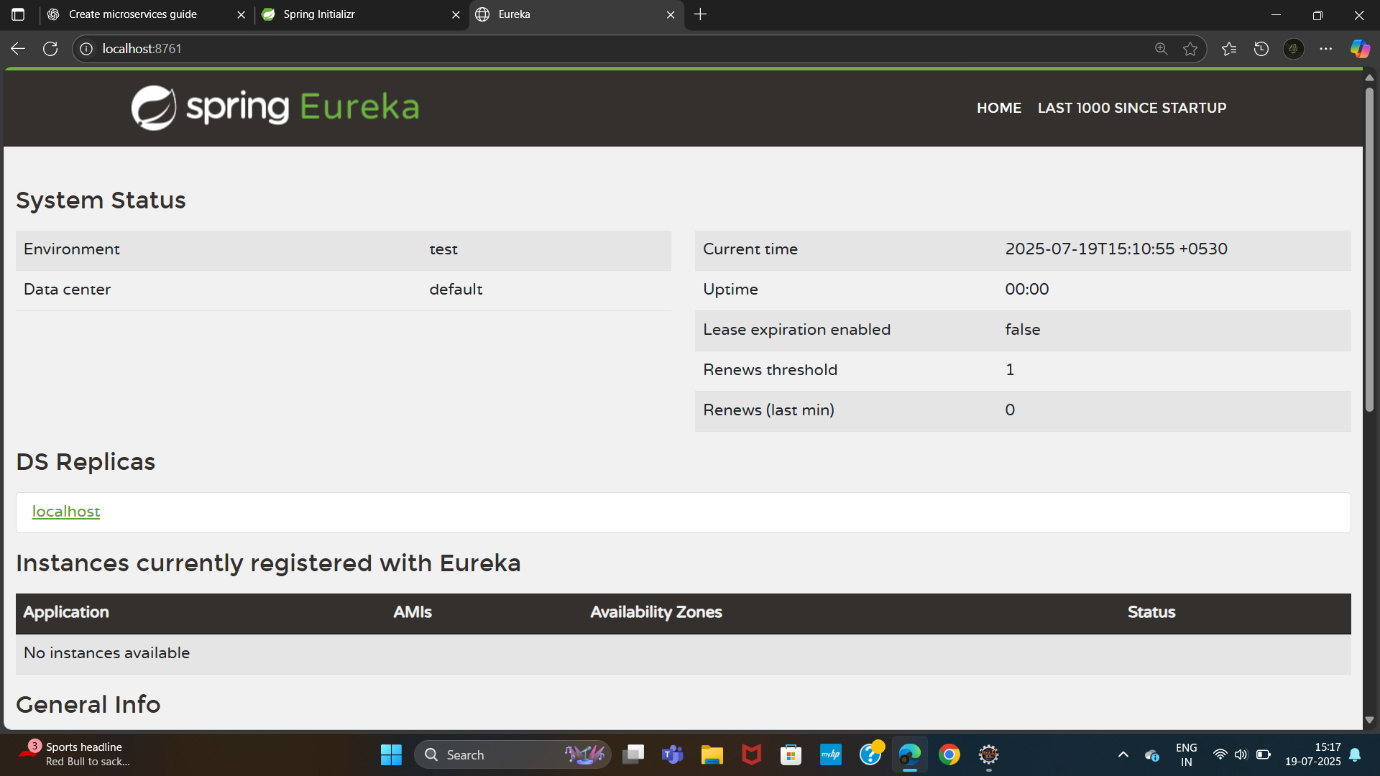
SpringApplication.run(EurekaDiscoveryServerApplication.class, args);

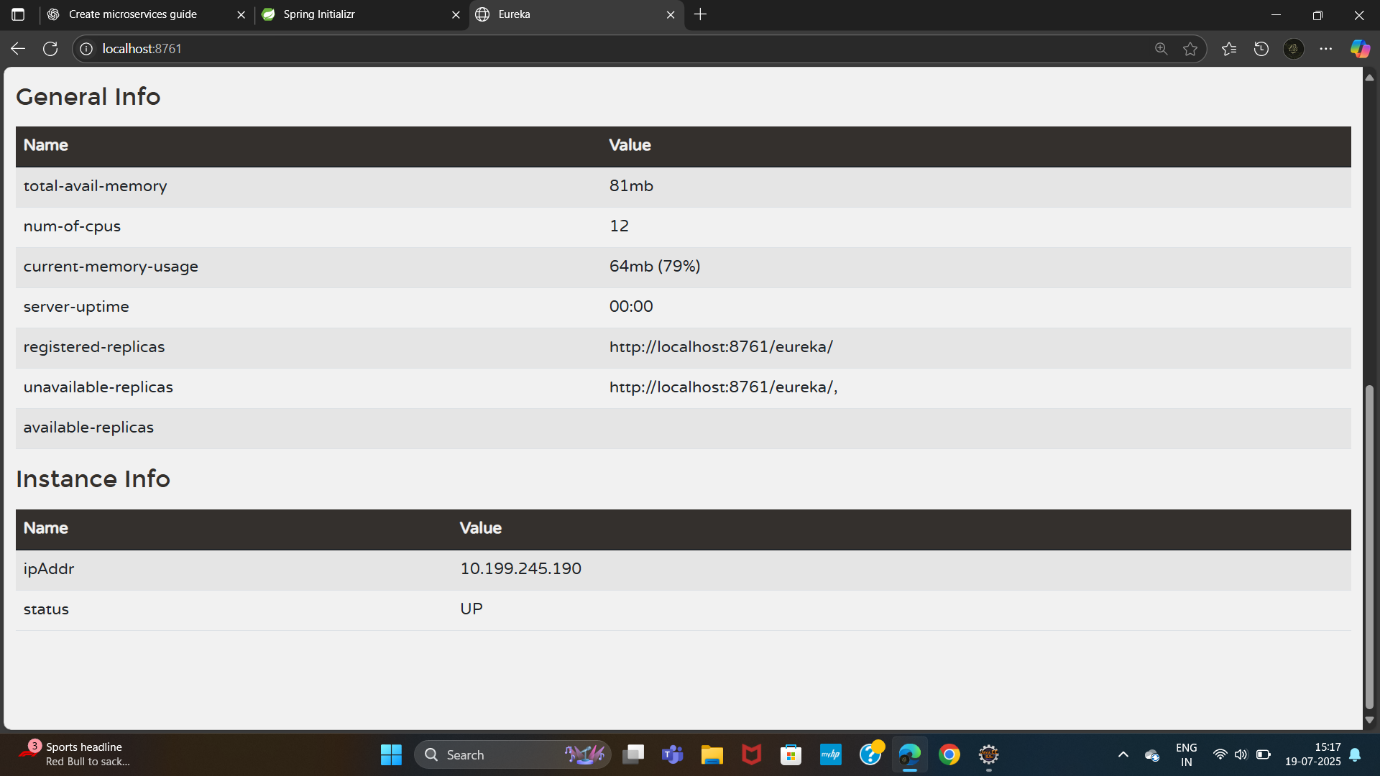
}

}

**OUTPUT:**







**Account -application.properties**

server.port=8083

spring.application.name=ACCOUNT-SERVICE

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

eureka.instance.prefer-ip-address=true

Loan-application.properties

server.port=8082

spring.application.name=LOAN-SERVICE

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

eureka.instance.prefer-ip-address=true

**ApiGatewayApplication.java**

package com.cognizant.api\_gateway;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

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

@SpringBootApplication

@EnableDiscoveryClient

public class ApiGatewayApplication {

public static void main(String[] args) {

SpringApplication.run(ApiGatewayApplication.class, args);

}

}

**Application.properties**

spring.application.name=API-GATEWAY

server.port=8080

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

eureka.client.register-with-eureka=true

eureka.client.fetch-registry=true

# Spring Cloud Gateway Routes

spring.cloud.gateway.routes[0].id=ACCOUNT-SERVICE

spring.cloud.gateway.routes[0].uri=http://localhost:8083

spring.cloud.gateway.routes[0].predicates[0]=Path=/account/\*\*

spring.cloud.gateway.routes[1].id=LOAN-SERVICE

spring.cloud.gateway.routes[1].uri=http://localhost:8082

spring.cloud.gateway.routes[1].predicates[0]=Path=/loan/\*\*

# Ensure it uses reactive web app type

spring.main.web-application-type=reactive

**OUTPUT:**

