**ADDITIONAL EXERCISES HANDSON SOLUTIONS -WEEK-05**

***-AAISHA SULTANA GUDURU-6384619***

**EXERCISE-01 MICROSERVICES**

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. Follow steps below to implement:

Create and Launch Eureka Discovery Server

• Using https://start.spring.io generate a project with following configuration:

o Group: com.cognizant

o Artifact: eureka-discovery-server

o Module: Spring Cloud Discovery > Eureka Server

• Download the project, build it using maven in command line

• Import the project in Eclipse

• Include @EnableEurekaServer in class EurekaDiscoveryServerApplication

• Include the following configurations in application.properties:

**IMPLEMENTATION:**

The objective of this exercise is to stand up a Spring Cloud Netflix Eureka Discovery Server and register two independent REST microservices,account-service and loan-service,so that both appear in the Eureka registry and are discoverable for downstream consumers. The stack used is JDK 17, Spring Boot 3.5.4, and Spring Cloud 2025.0.0 (imported via the spring-cloud-dependencies BOM), built with Maven and run from Eclipse/STS.

First, the Eureka Discovery Server project is created with group com.cognizant and artifact eureka-discovery-server, including the dependency spring-cloud-starter-netflix-eureka-server.

The main application class is annotated with @EnableEurekaServer and @SpringBootApplication to activate the Eureka dashboard and server endpoints.

In application.properties the server is configured to listen on port 8761 and to avoid acting as a client by setting server.port=8761, eureka.client.register-with-eureka=false, and eureka.client.fetch-registry=false.

Optional log level lines can be added to quiet Eureka’s own logs during development.

With these settings in place, the application is built and started; navigating to <http://localhost:8761> displays the Eureka dashboard, which initially shows an empty list of registered instances.

Next, the account-service microservice is created with group com.cognizant and artifact account, adding Spring Web for REST and spring-cloud-starter-netflix-eureka-client for service registration.

To ensure version compatibility, the same Spring Cloud BOM and spring-cloud.version used in the server POM are also declared in the account POM.

The main class is annotated with @EnableDiscoveryClient and @SpringBootApplication.

The application is named in application.properties so it appears correctly in the registry by setting spring.application.name=account-service and is configured to run on port 8080.

The Eureka server address is provided with eureka.client.service-url.defaultZone=<http://localhost:8761/eureka/>.

Similarly, the loan-service microservice is created with the same dependencies and annotations and with the same BOM alignment as account-service.

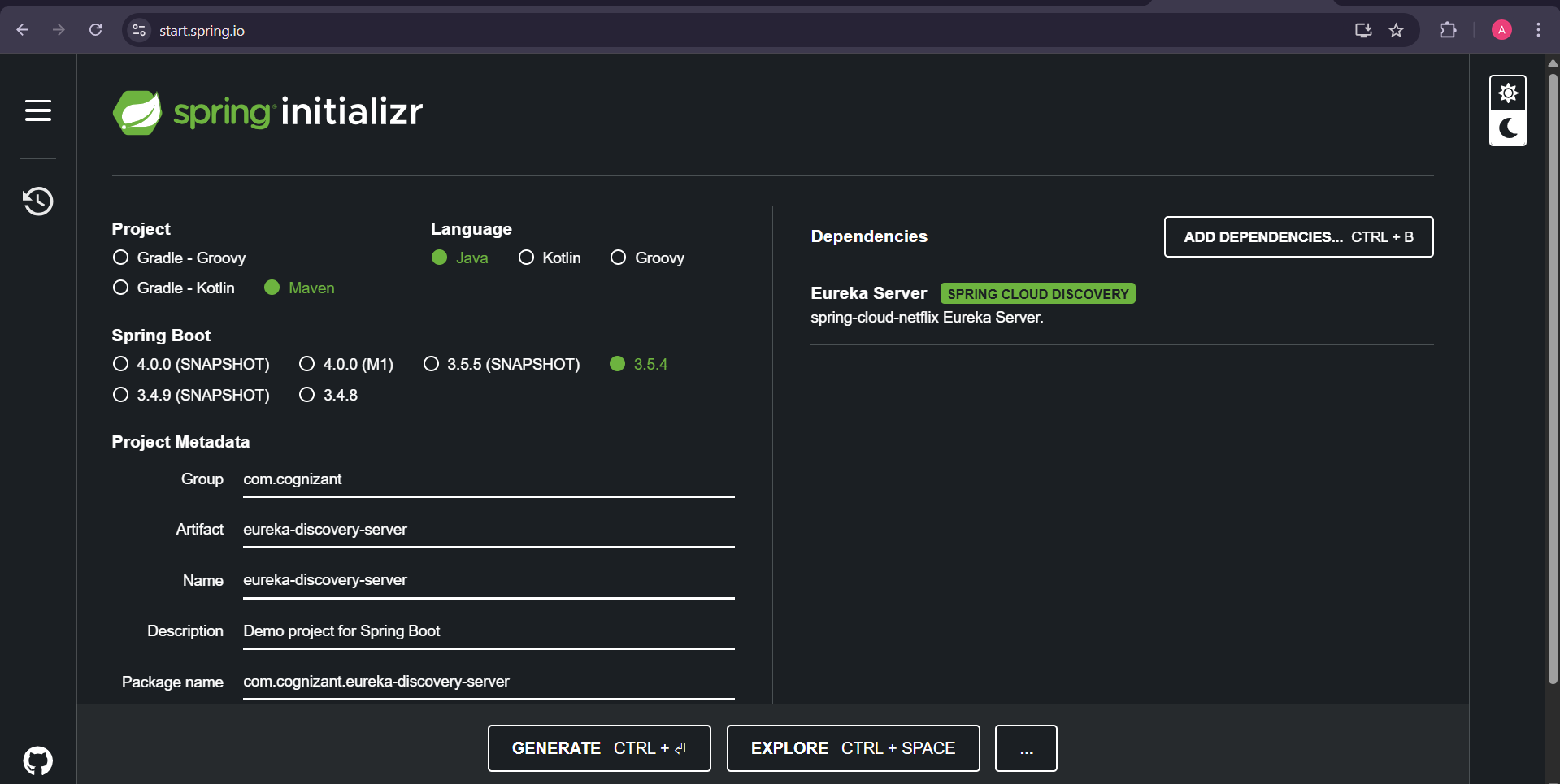
In application.properties it is named spring.application.name=loan-service, configured to use a non-conflicting port such as 8081,

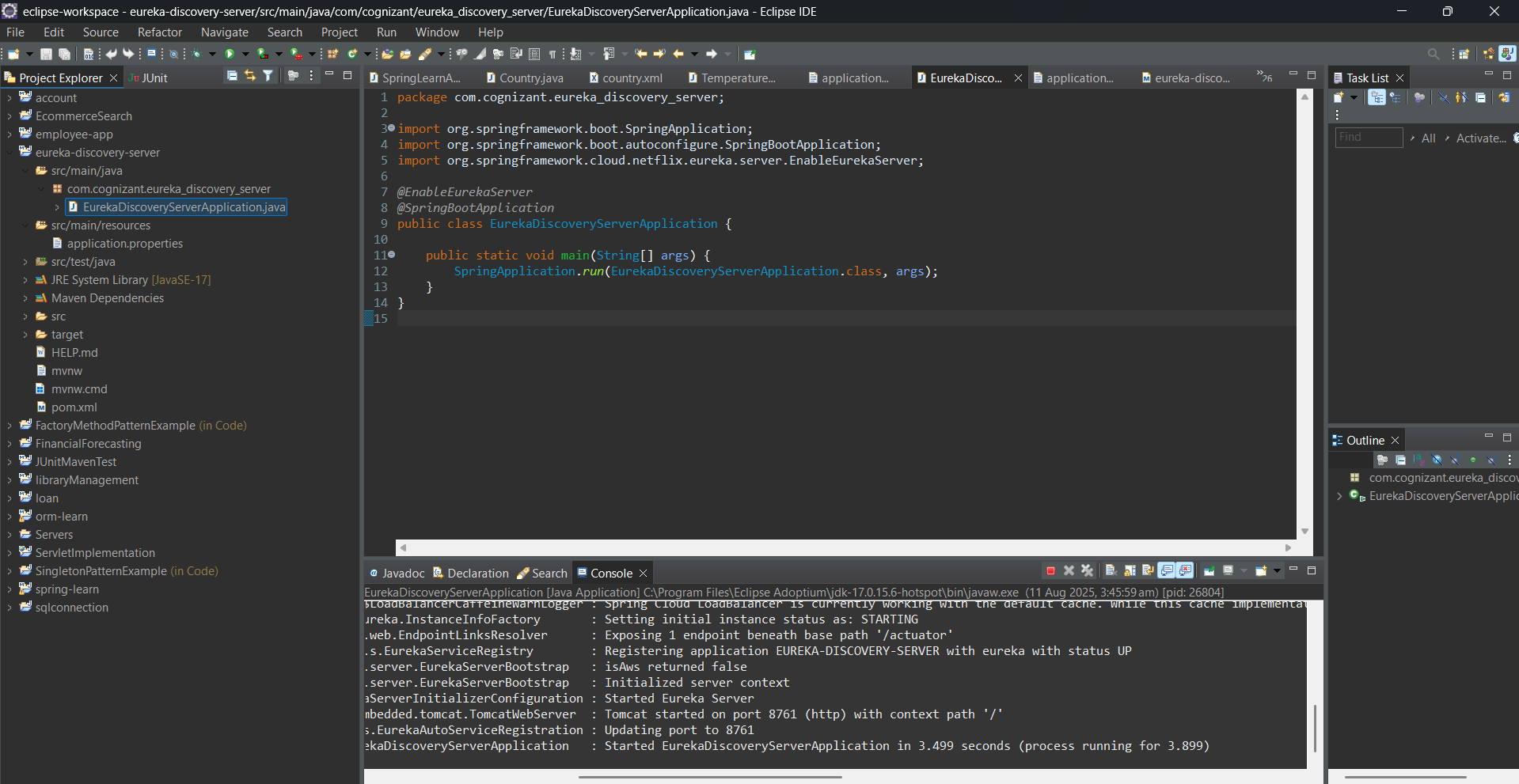
and pointed to the same Eureka server with eureka.client.service-url.defaultZone=<http://localhost:8761/eureka/>.

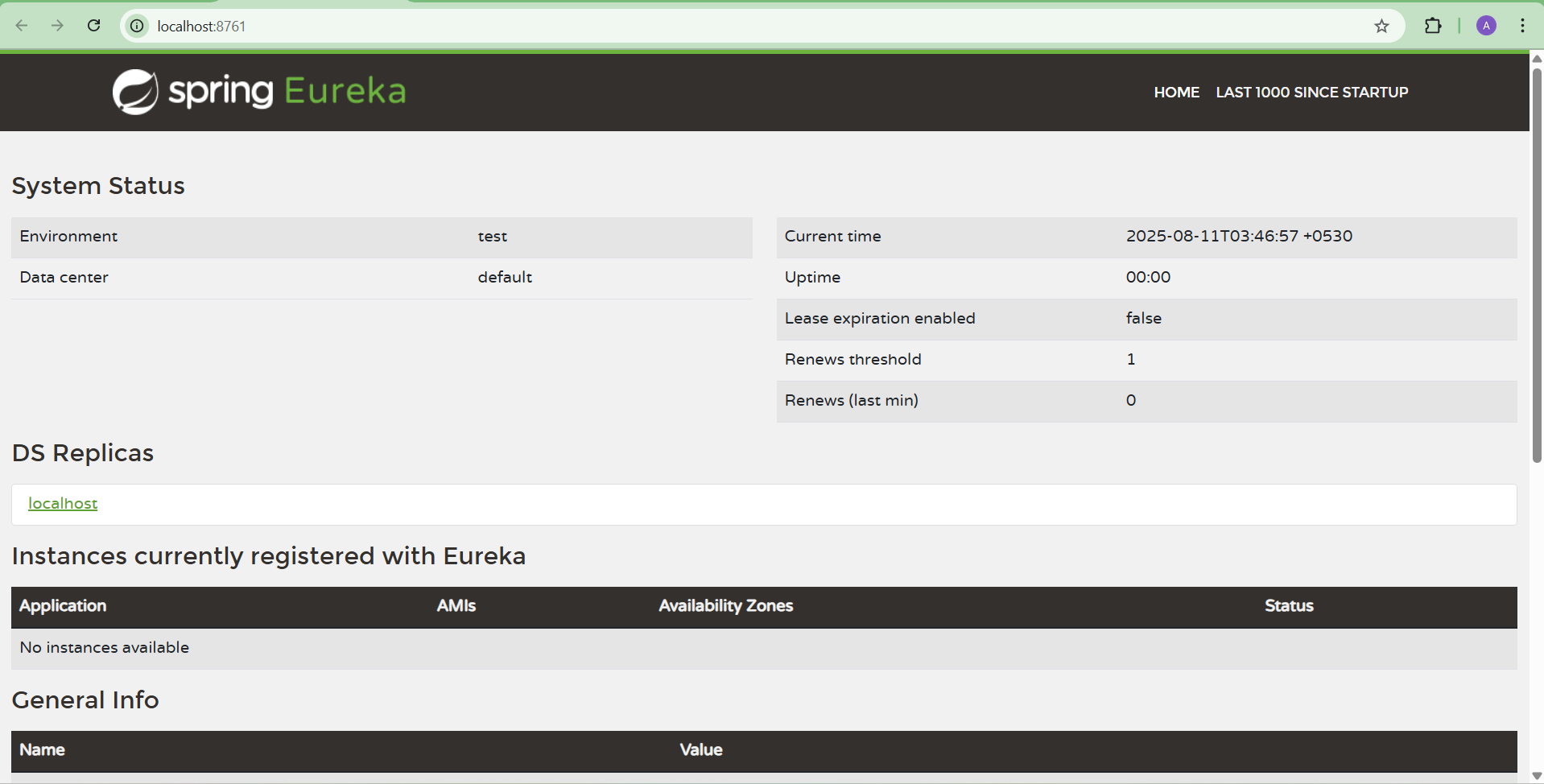
A simple GET endpoint is added to confirm the service responded to HTTP requests.

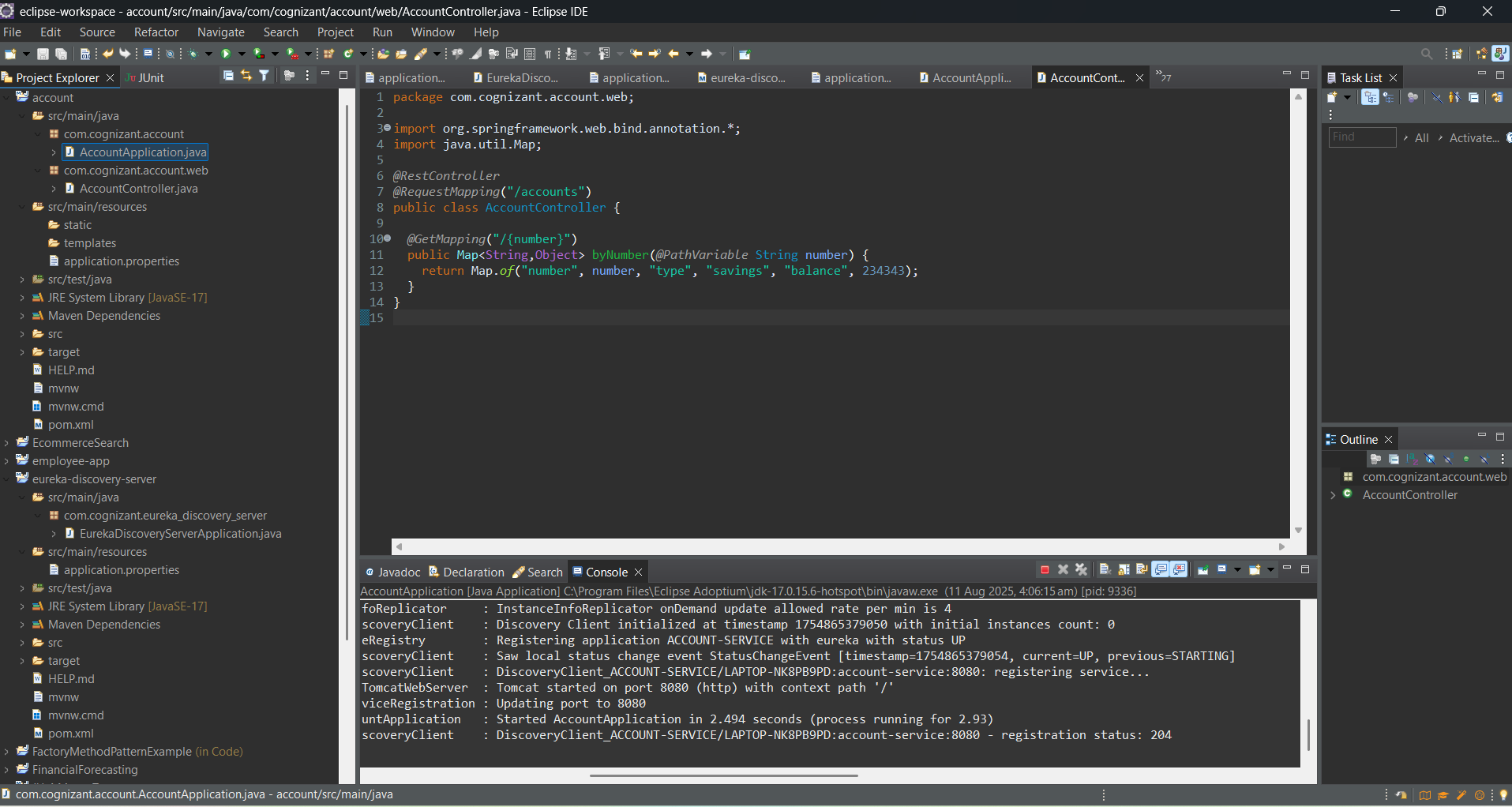
The run order started the Eureka server first and confirmed that the dashboard is reachable at <http://localhost:8761> with no instances listed. Then started the account-service and waited for it to complete initialization; refreshing the Eureka dashboard now showed ACCOUNT-SERVICE under the list of registered instances. Finally, started loan-service and refreshed the dashboard again to see LOAN-SERVICE appear alongside ACCOUNT-SERVICE. I directly tested the sample REST endpoints at [http://localhost:8080/accounts/{id}](http://localhost:8080/accounts/%7Bid%7D) and [http://localhost:8081/loans/{id}](http://localhost:8081/loans/%7Bid%7D) to confirm that both microservices are running while Eureka reflects their presence.

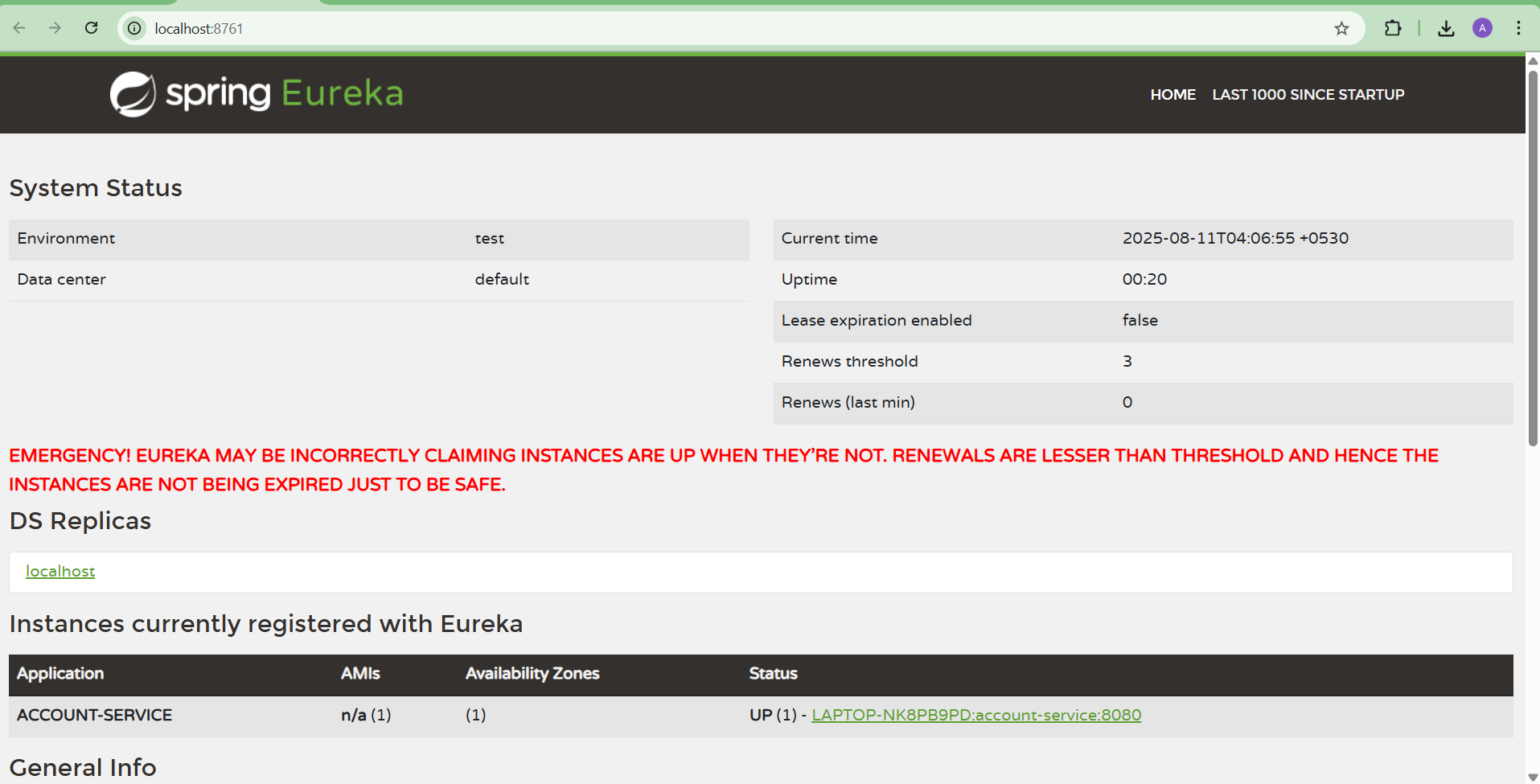
Below snippets provide a clear and visual understanding of the entire implementation and show the expected outputs successfully.



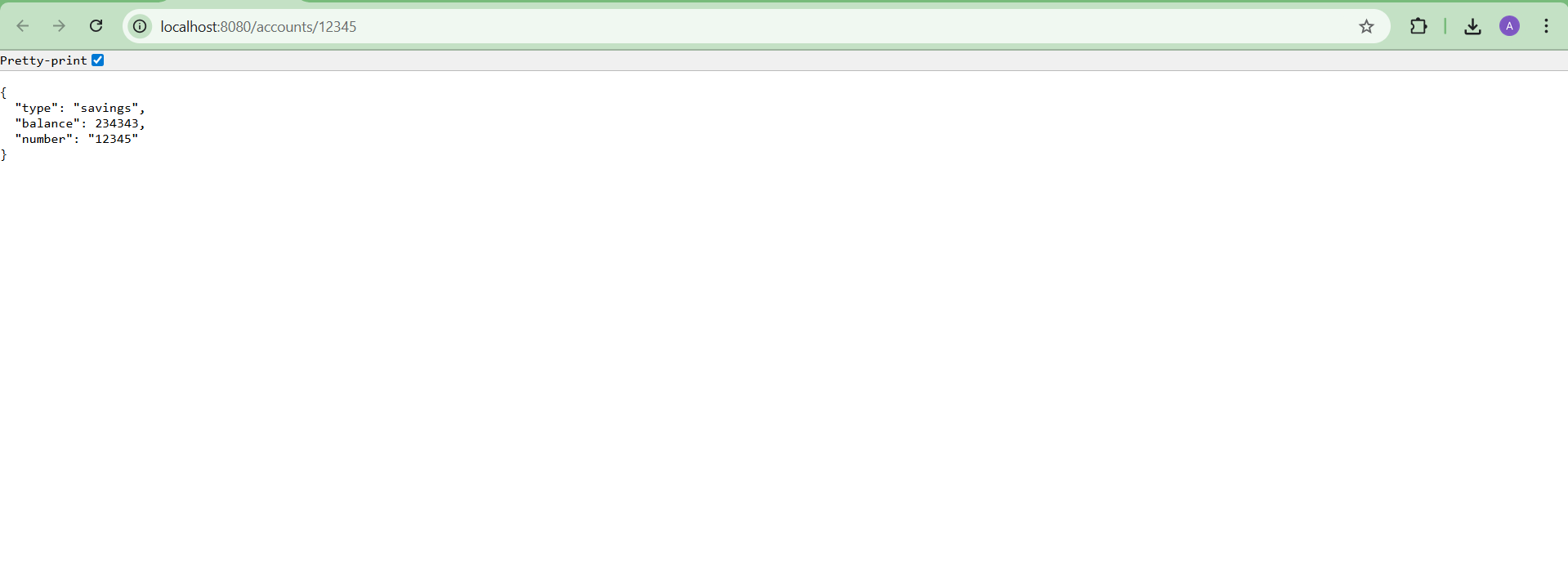


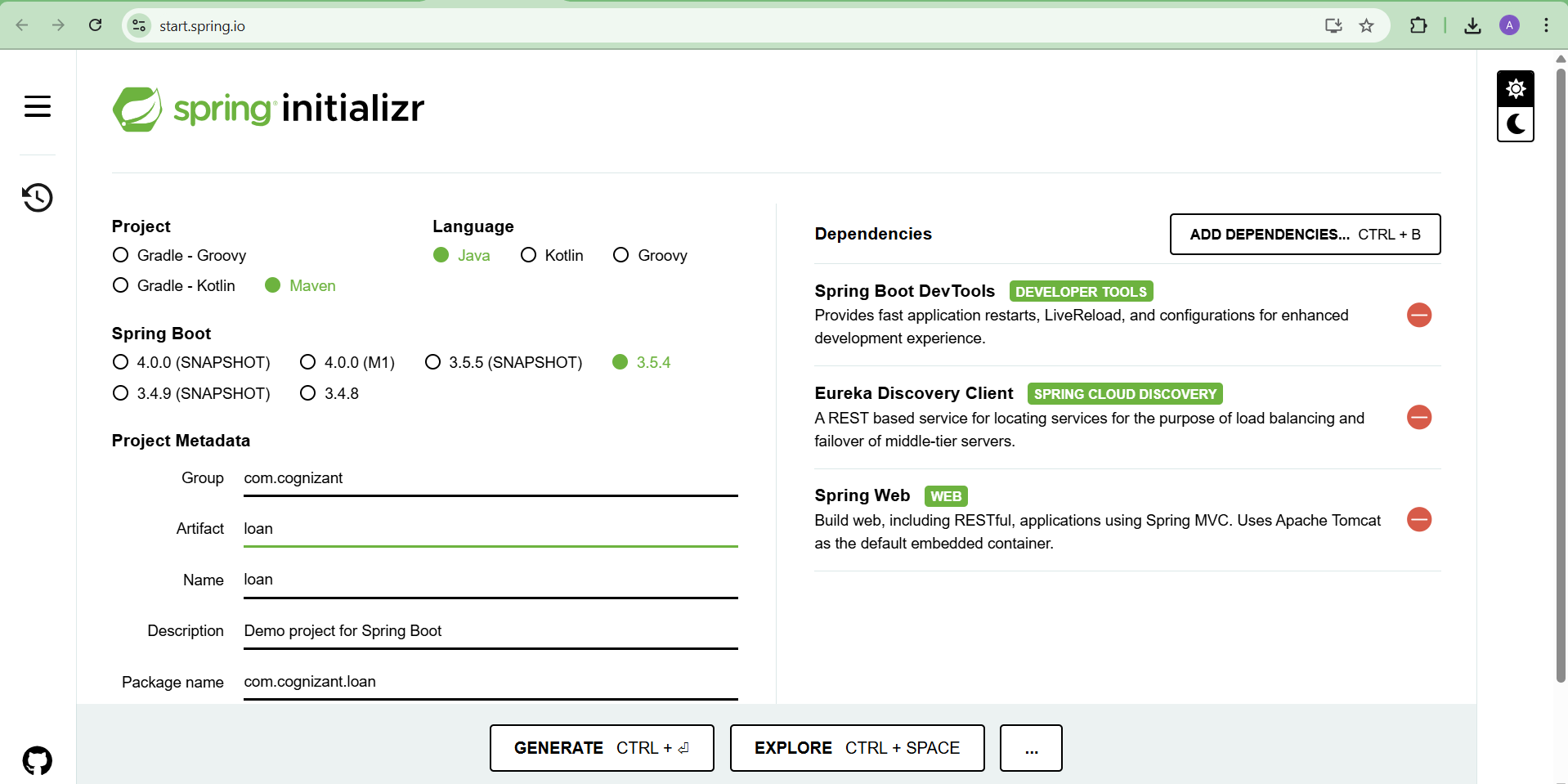


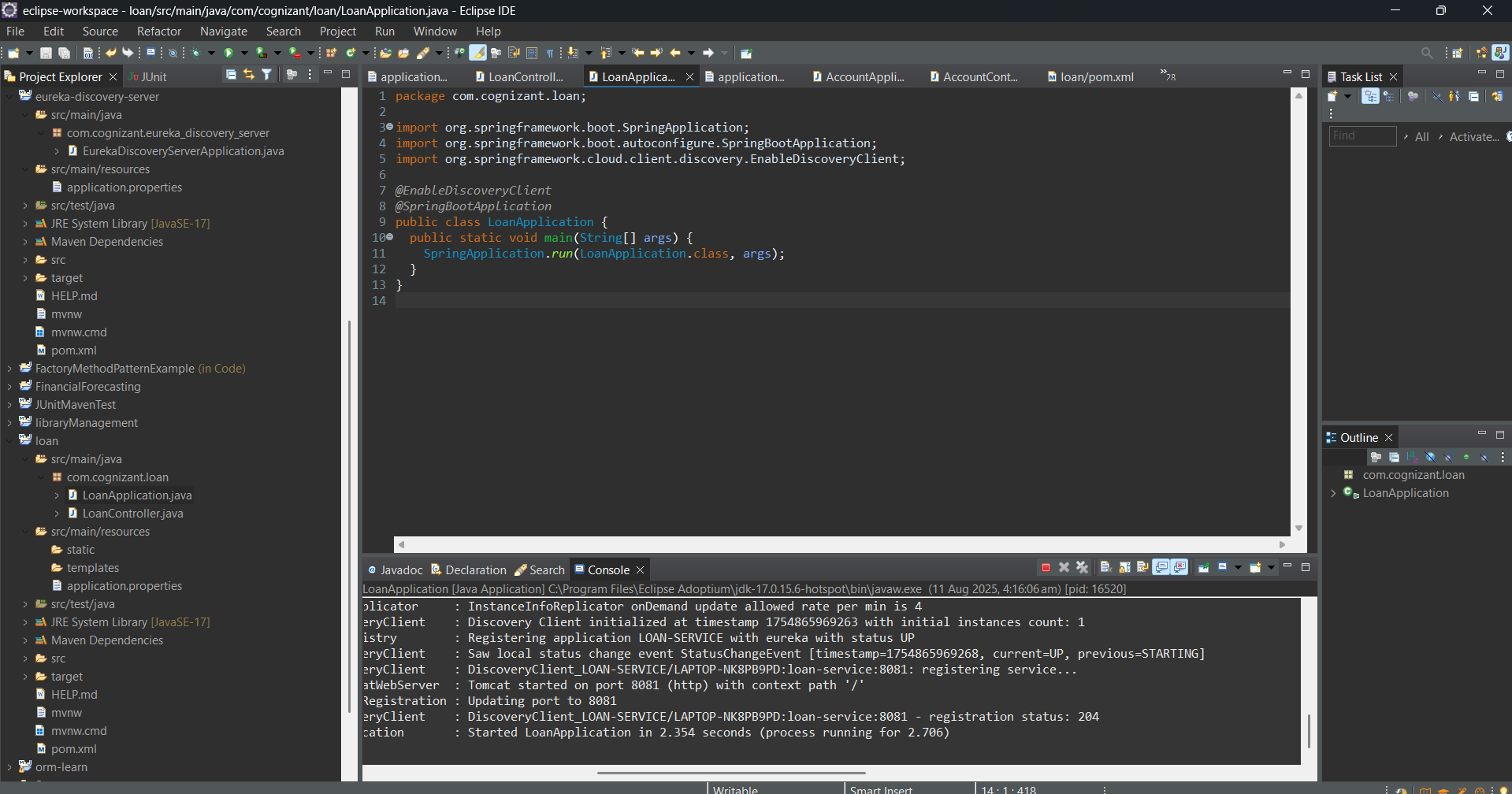




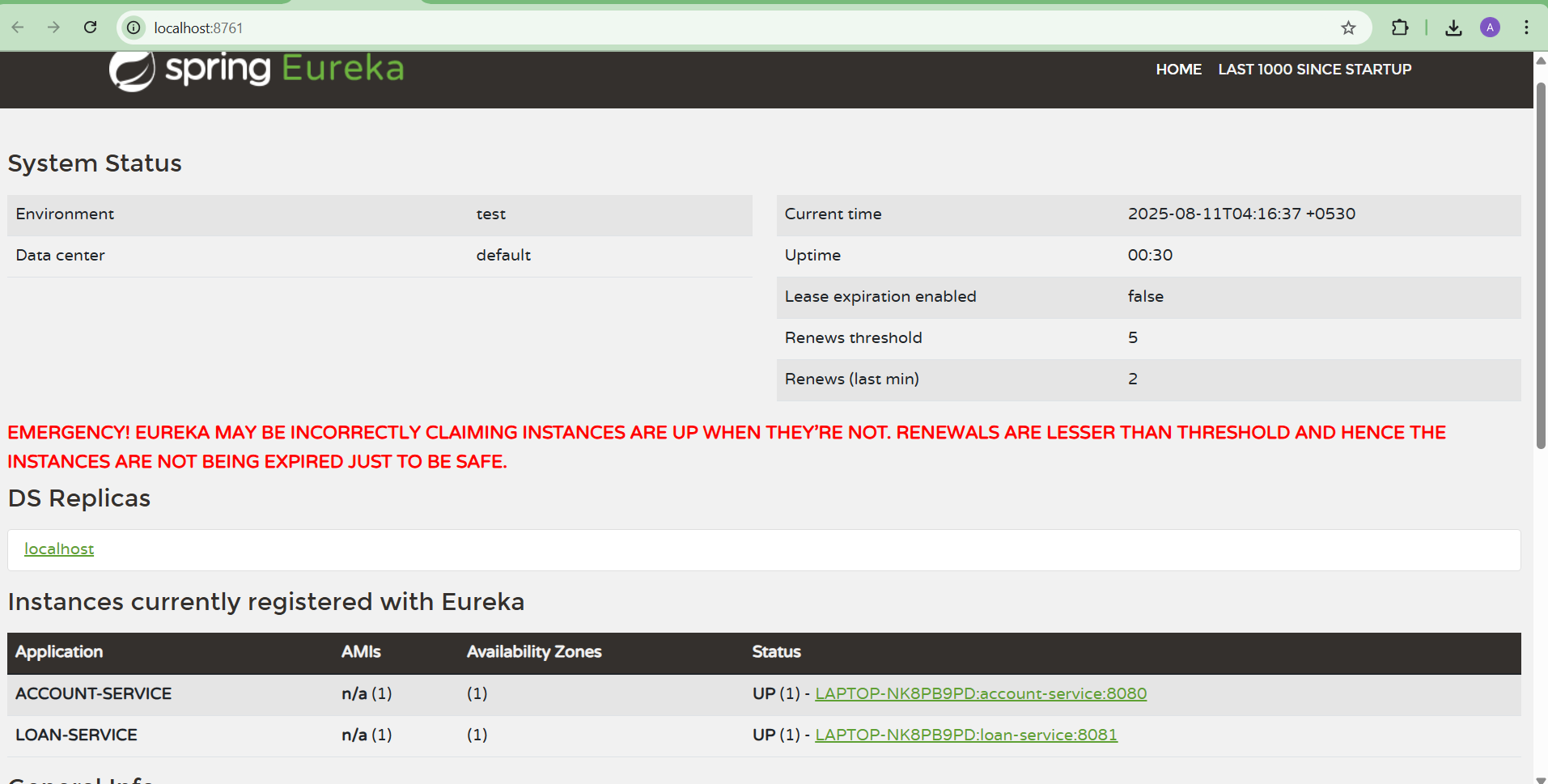


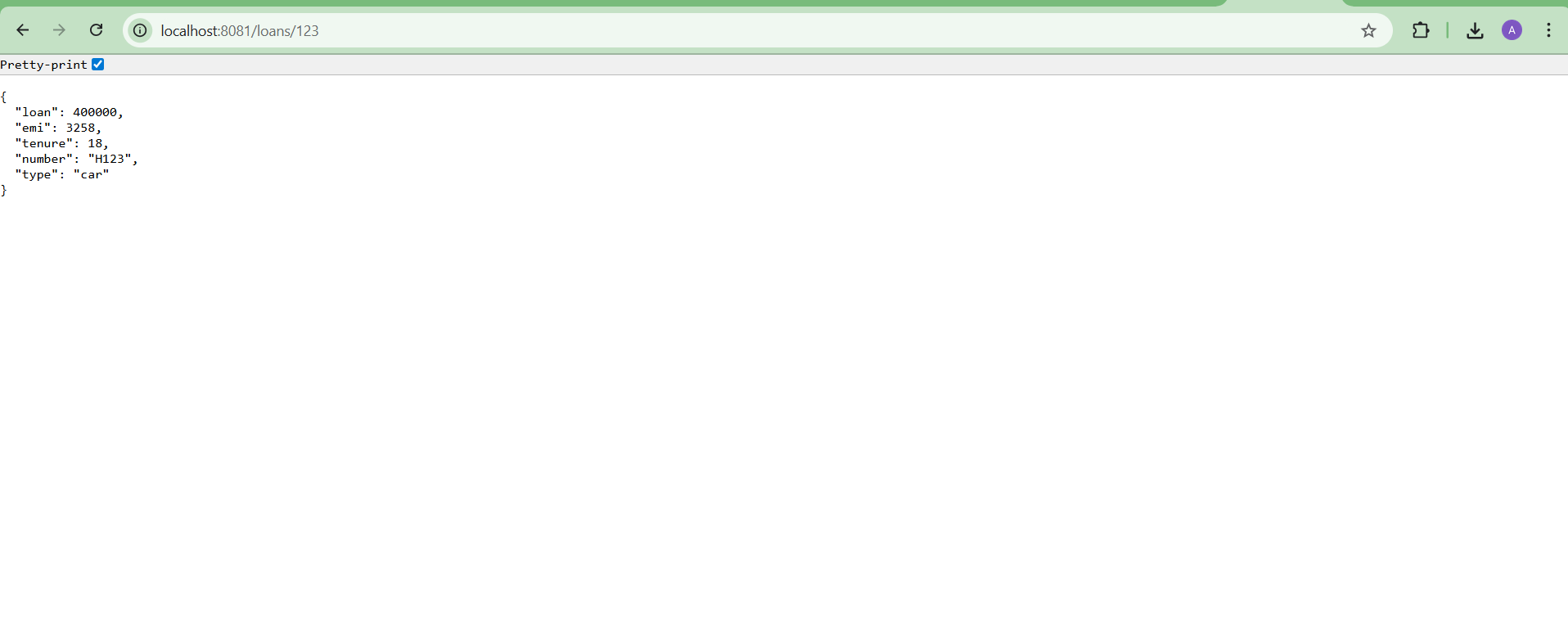






**OUTPUT:**

****

****