

WEEK 3 – ASSIGNMENT

Superset ID: 6390124

Microservices with Spring Boot 3 and Spring Cloud Exercises:-

Exercise 1: Creating Microservices for account and loan

In this hands-on exercises, we will create two microservices for a bank. One microservice for handling accounts and one for handling loans.

Each microservice will be a specific independent Spring RESTful Webservice maven project having its 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.

src/main/java/com/cognizant/account/controller/AccountController.java

```
package com.cognizant.account.controller;

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

import java.util.Map;

@RestController
@RequestMapping("/accounts")
public class AccountController {

    @GetMapping("/{number}")
    public Map<String, Object> getAccountByNumber(@PathVariable String
number) {
        return Map.of(
            "number", number,
            "type", "savings",
            "balance", 234343);
    }
}
```

src/main/java/com/cognizant/account/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);
    }

}
```

src/main/resources/application.properties

```
spring.application.name=account
server.port=8080
```

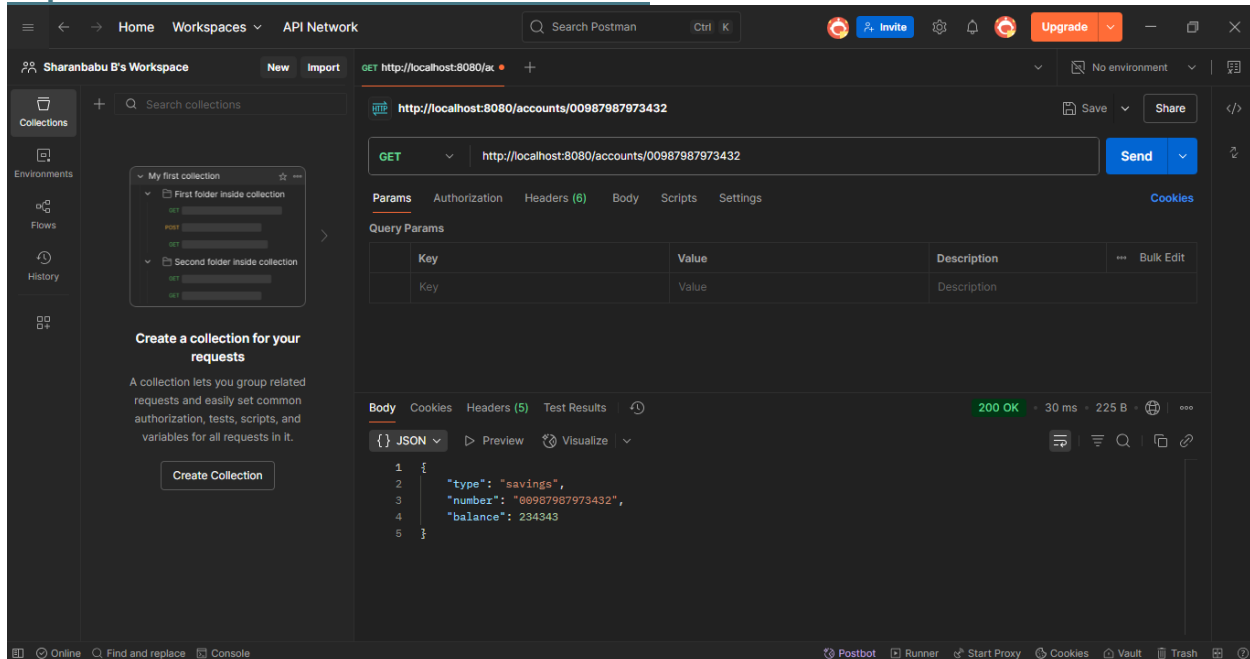
Output:



The screenshot shows a terminal window with the following output:

```
t 8080 (http) with context path '/'
2025-07-19T13:37:14.401+05:30 INFO 6008 --- [account] [ restartedMain] c.cognizant.account.AccountApplication : Started AccountApplic
ation in 4.526 seconds (process running for 5.683)
2025-07-19T13:37:58.540+05:30 INFO 6008 --- [account] [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring D
ispatcherServlet 'dispatcherServlet'
2025-07-19T13:37:58.540+05:30 INFO 6008 --- [account] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet
'dispatcherServlet'
2025-07-19T13:37:58.545+05:30 INFO 6008 --- [account] [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initializat
ion in 2 ms
```

<http://localhost:8080/accounts/00987987973432>



src/main/java/com/cognizant/loan/controller/LoanController.java

```
package com.cognizant.loan.controller;
```

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

```
import java.util.Map;
```

```
@RestController
```

```
@RequestMapping("/loans")
```

```
public class LoanController {
```

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

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

```
        return Map.of(
            "number", number,
            "type", "car",
            "loan", 400000,
            "emi", 3258,
            "tenure", 18);
    }
```

```
}
```

src/main/java/com/cognizant/loan/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);
    }
}
```

src/main/resources/application.properties

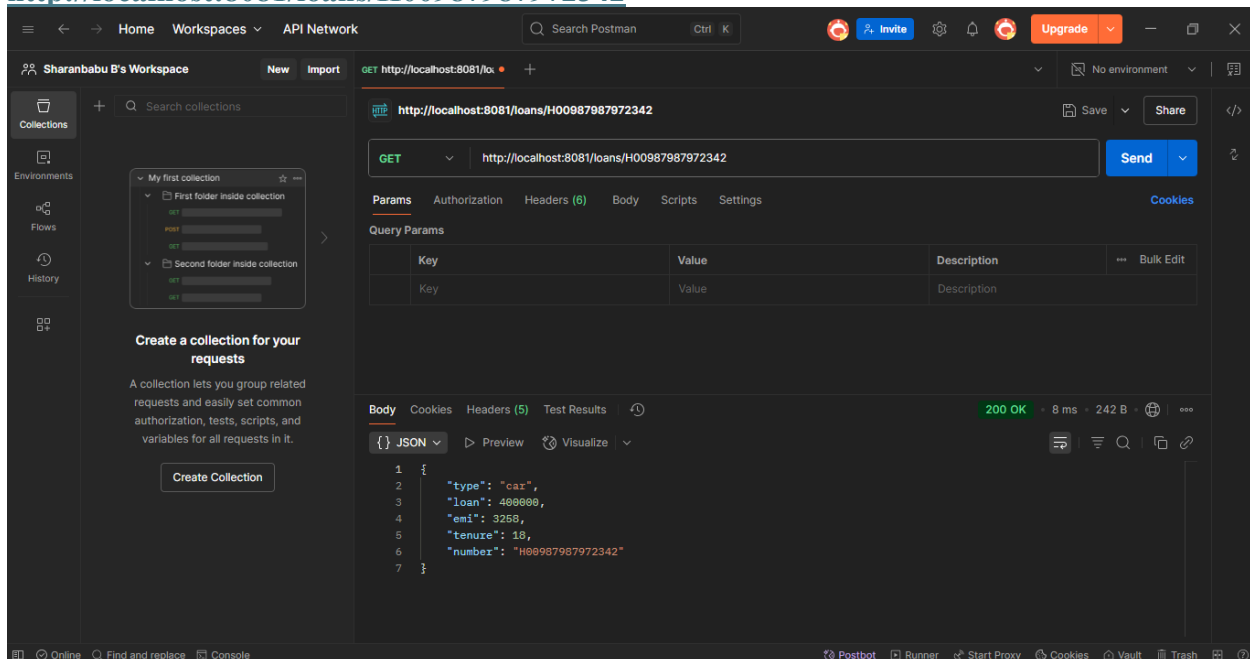
```
spring.application.name=loan
server.port=8081
```

Output:



```
8081 (http) with context path '/'
2025-07-19T14:11:24.698+05:30 INFO 19324 --- [loan] [ restartedMain] com.cognizant.loan.LoanApplication : Started LoanApplication
in 4.284 seconds (process running for 4.998)
2025-07-19T14:11:25.279+05:30 INFO 19324 --- [loan] [nio-8081-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring Dis
patcherServlet 'dispatcherServlet'
2025-07-19T14:11:25.280+05:30 INFO 19324 --- [loan] [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'd
ispatcherServlet'
2025-07-19T14:11:25.281+05:30 INFO 19324 --- [loan] [nio-8081-exec-1] o.s.web.servlet.DispatcherServlet : Completed initializatio
n in 1 ms
```

http://localhost:8081/loans/H00987987972342



GET http://localhost:8081/loans/H00987987972342

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value	Description
Key	Value	Description

Body Cookies Headers (5) Test Results

200 OK · 8 ms · 242 B

```
{
  "type": "car",
  "loan": 400000,
  "emi": 3250,
  "tenure": 18,
  "numbez": "H00987987972342"
}
```

Exercise 2: 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`:
`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`
- The above configuration runs the discovery service in port 8761
- The eureka properties prohibits direct registration of services, instead discovery server will find available services and register them.
- Launch the service by running the application class
- The discovery service can be view by launching `http://localhost:8761` in the browser.
- This will display the discover server details
- Look into the section "Instances currently registered with Eureka", which will have an empty list
- Follow steps below to add account and loan service to this discovery server.

Register Account REST API to eureka discovery

- Go to <https://start.spring.io> and provide the following configuration:
 - o Group: com.cognizant
 - o Artifact: account
 - o Modules:
 1. Spring Boot DevTools
 2. Eureka Discovery Client
 3. Spring Web
- Click "Explore", which will open pom.xml
- Use copy option in the opened window to copy the pom.xml and overwrite the pom.xml in account project
- Build the project using maven in console
- Include `@EnableDiscoveryClient` annotation to application class of account project
- Include application name for account application as specified below in `application.properties`. This is the name that will be displayed in the eureka discovery registry.

`spring.application.name=account-service`

- Stop all services (account, loan, eureka-discovery-server) using the console window of Eclipse. Use the monitor icon in console view to switch between applications and use the Terminate button to stop the server.
- First start eureka-discovery-server and wait till the application starts completely. Then open `http://localhost:8761` in browser. The service list should be empty.
- Then start account application and wait till the application starts.
- Refresh the eureka-discovery-server web page in browser, the account service will be listed in the registry
- Perform similar steps for loan application and have it registered with eureka-discovery-server.

**Create a Spring Cloud API Gateway and call one microservice thru the API gateway.
Configure a global filter to log each request targeting the microservice using Spring Cloud API Gateway.**

Steps.

1. Create Simple Microservice greet-service that returns “Hello World” using Spring Initializer.
2. Select the latest version of Spring Boot and the dependencies as shown in the image below.
3. Configure the application name in “application.properties” as shown below
4. Create a controller as shown below.
5. Run the microservice and make sure that it is working fine as shown below
6. Create a second microservice that is acting as the Discovery Server using Spring Cloud Eureka Server.
7. Select the latest version of Spring Boot and the required dependencies as shown below
8. Include the following configurations in the “application.properties” of the Eureka server
9. Annotate the main class in Eureka Server with `@EnableEurekaServer` as shown below.
10. Run the Eureka Server and ensure that the server is functioning properly by entering the below URL at any of the browser.
`http://localhost:8761`
11. Open the pom.xml of greet-service microservice and add the eureka client dependencies as shown in the image below.
12. Copy the spring cloud version from pom.xml of eureka-server and paste it at the appropriate location in the pom.xml of greet-service as shown below.
13. Copy the “dependency-management” session in the pom.xml of the eureka-server and paste it immediately after the dependencies session of the pom.xml of the greet-service as shown below.
14. Restart greet-service and refresh the `http://localhost:8761` to see whether the name of the greet-service figuring in the eureka-server console as shown below.
15. Create another microservice “api-gateway” using spring initializer.
16. Select the latest version of Spring Boot and add the required dependencies as shown below.
17. Make the necessary configurations in the “application.properties” file as shown below.
18. Run the api-gateway service and check if it is getting registered with the eureka-server.
19. Try to access the following URL from any of the browser and see if you are able to access the greet-service thru the api-gateway.
`http://localhost:9090/GREET-SERVICE/greet`
20. Include the following configuration in the “ application.properties” of the api-gateway to specify the service name in lower case.

21. Implement a global filter which logs all incoming requests.
 - a. Create a LogFilter class as shown below.
22. Try to access the url <http://localhost:9090/greet-service/greet>. You will get the output as shown below.
23. Check the console of the api-gateway service and you should be getting the log shown below.

**src/main/java/com/cognizant/eurekadiscoveryserver/
EurekaDiscoveryServerApplication.java**

```
package com.cognizant.eurekadiscoveryserver;

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);
    }

}
```

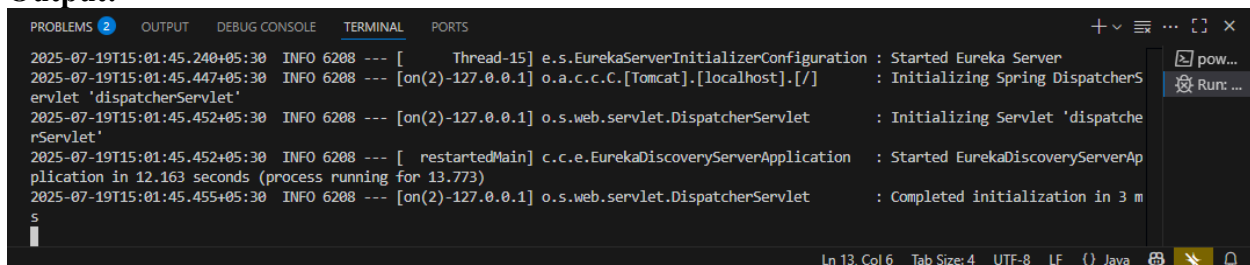
src/main/resources/application.properties

```
server.port=8761

eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false

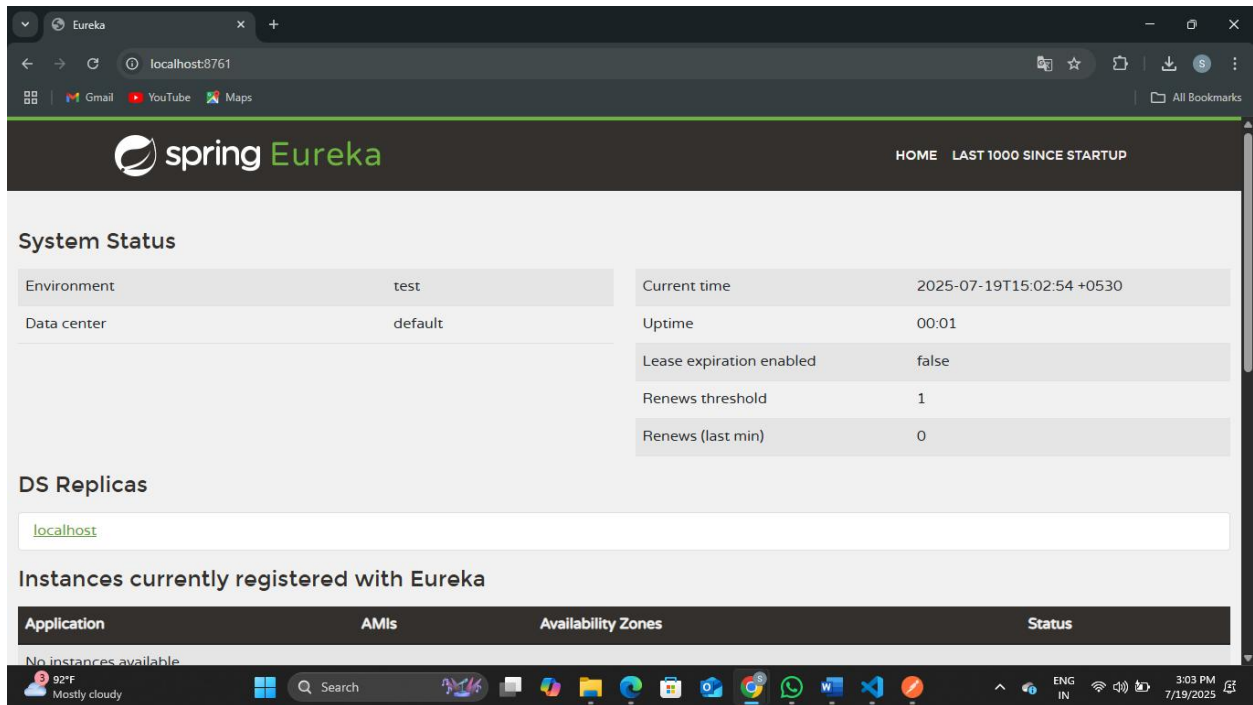
logging.level.com.netflix.eureka=OFF
logging.level.com.netflix.discovery=OFF
```

Output:



```
2025-07-19T15:01:45.240+05:30 INFO 6208 --- [ Thread-15] e.s.EurekaServerInitializerConfiguration : Started Eureka Server
2025-07-19T15:01:45.447+05:30 INFO 6208 --- [on(2)-127.0.0.1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-07-19T15:01:45.452+05:30 INFO 6208 --- [on(2)-127.0.0.1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-07-19T15:01:45.452+05:30 INFO 6208 --- [ restartedMain] c.c.e.EurekaDiscoveryServerApplication : Started EurekaDiscoveryServerApplication in 12.163 seconds (process running for 13.773)
2025-07-19T15:01:45.455+05:30 INFO 6208 --- [on(2)-127.0.0.1] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms
```

<http://localhost:8761>



src/main/java/com/cognizant/account/AccountApplication.java

```
package com.cognizant.account;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class AccountApplication {
    public static void main(String[] args) {
        SpringApplication.run(AccountApplication.class, args);
    }
}
```

src/main/java/com/cognizant/account/controller/AccountController.java

```
package com.cognizant.account.controller;

import org.springframework.web.bind.annotation.*;
import java.util.Map;

@RestController
@RequestMapping("/accounts")
public class AccountController {

    @GetMapping("/{number}")
    public Map<String, Object> getAccount(@PathVariable String number) {
        return Map.of(
            "number", number,
            "type", "savings",
            "balance", 234343);
    }
}
```



```
}
```

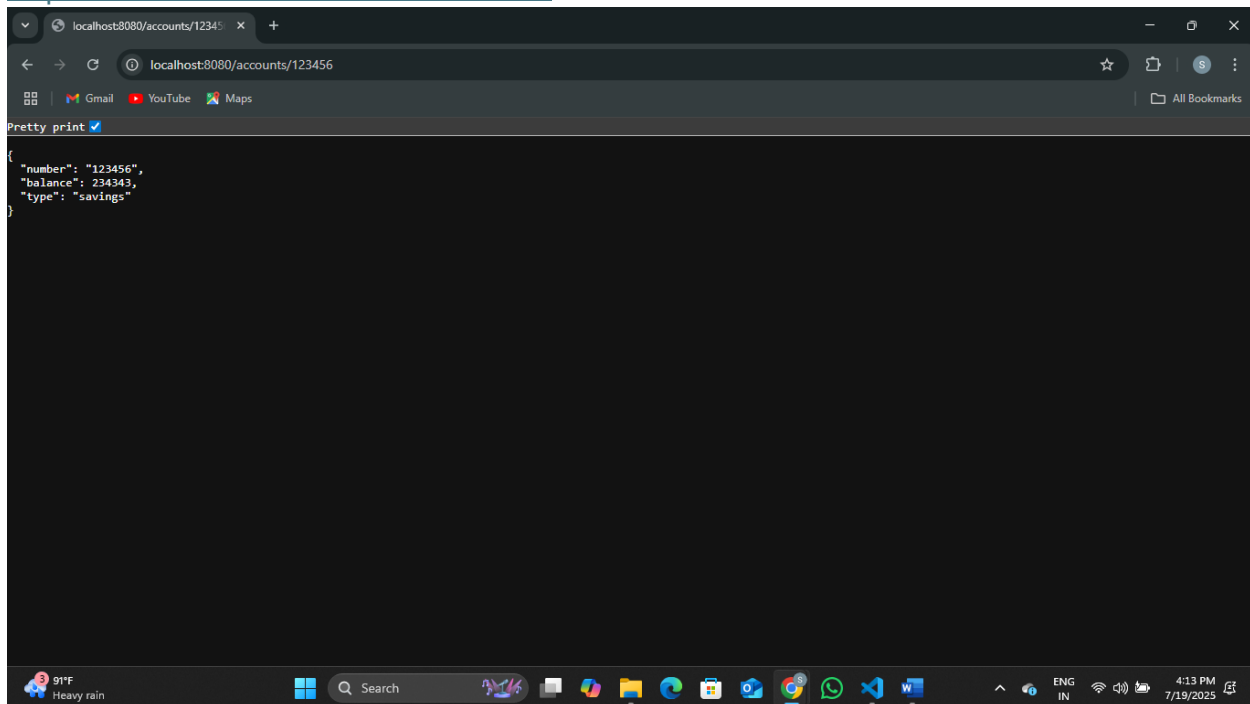
src/main/resources/application.properties

```
server.port=8080
spring.application.name=account-service
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

Output:

A screenshot of an IDE terminal window. The terminal shows logs for a Spring Boot application starting on port 8080. It includes timestamps, log levels (INFO), and messages about the application starting, Spring initializing, and the DispatcherServlet initializing and completing initialization. The status bar at the bottom indicates 'Ln 11, Col 47', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Java'.

<http://localhost:8080/accounts/123456>

A screenshot of a web browser window. The address bar shows 'localhost:8080/accounts/123456'. The page content displays a JSON object: {"number": "123456", "balance": 234343, "type": "savings"}. The browser's status bar at the bottom shows '91°F Heavy rain', 'Search', and the date '7/19/2025'.

src/main/java/com/cognizant/loan/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);
    }
}
```

```
}  
}
```

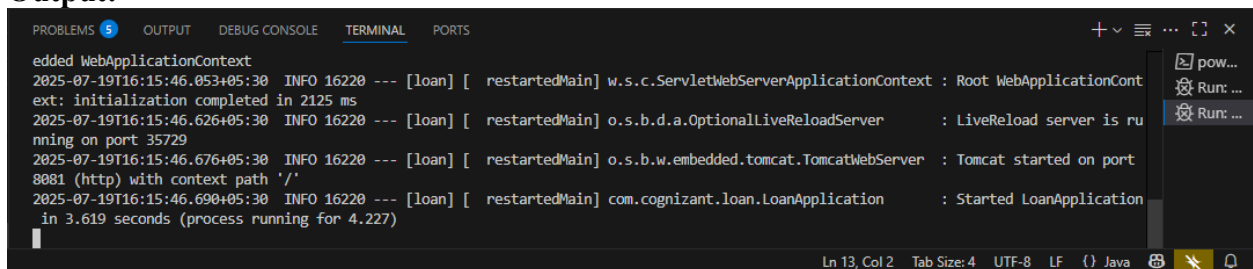
src/main/java/com/cognizant/loan/controller/LoanController.java

```
package com.cognizant.loan.controller;  
  
import org.springframework.web.bind.annotation.*;  
import java.util.Map;  
  
@RestController  
@RequestMapping("/loans")  
public class LoanController {  
  
    @GetMapping("/{number}")  
    public Map<String, Object> getLoan(@PathVariable String number) {  
        return Map.of(  
            "number", number,  
            "type", "car",  
            "loan", 400000,  
            "emi", 3258,  
            "tenure", 18);  
    }  
}
```

src/main/resources/application.properties

```
server.port=8081  
spring.application.name=loan-service  
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

Output:

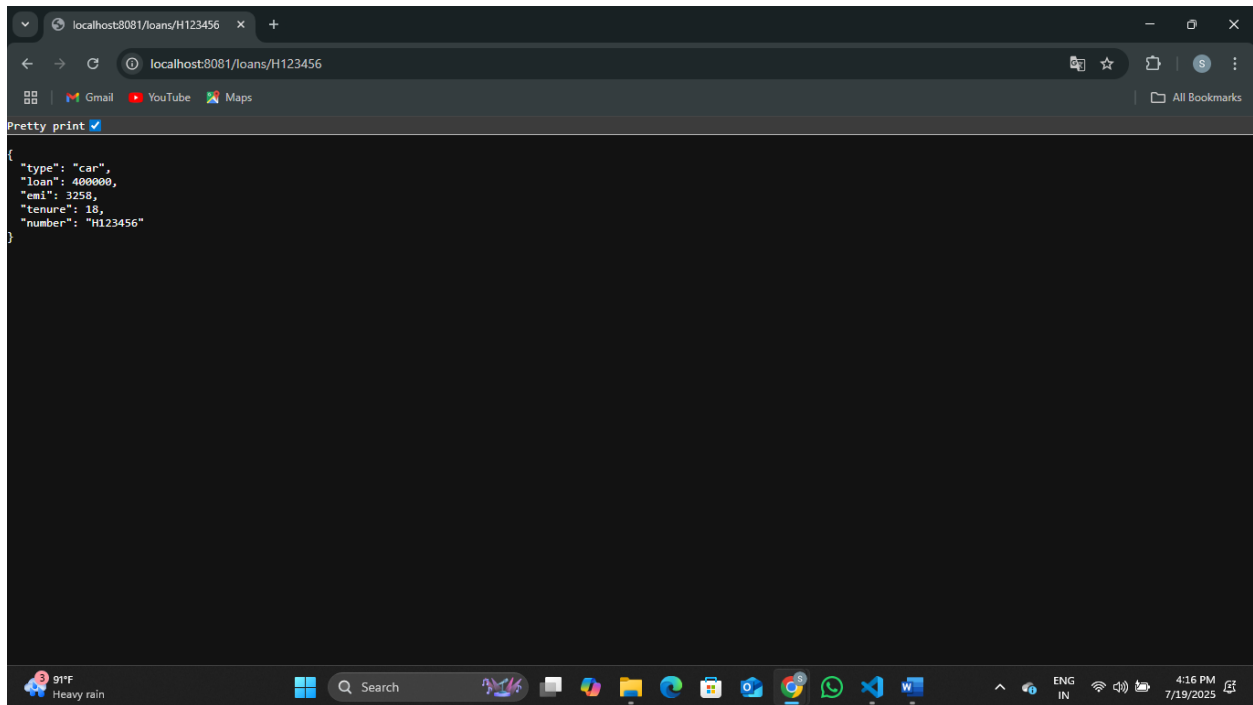


The screenshot shows an IDE terminal window with the following logs:

```
edded WebApplicationContext  
2025-07-19T16:15:46.053+05:30 INFO 16220 --- [loan] [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationCont  
ext: initialization completed in 2125 ms  
2025-07-19T16:15:46.626+05:30 INFO 16220 --- [loan] [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is ru  
nning on port 35729  
2025-07-19T16:15:46.676+05:30 INFO 16220 --- [loan] [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port  
8081 (http) with context path '/'  
2025-07-19T16:15:46.690+05:30 INFO 16220 --- [loan] [ restartedMain] com.cognizant.loan.LoanApplication : Started LoanApplication  
in 3.619 seconds (process running for 4.227)
```

The terminal window has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The status bar at the bottom indicates 'Ln 13, Col 2', 'Tab Size: 4', 'UTF-8', 'LF', and '() Java'.

<http://localhost:8081/loans/H123456>



src/main/java/com/cognizant/greet/GreetServiceApplication.java

```
package com.cognizant.greet;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class GreetServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(GreetServiceApplication.class, args);
    }
}
```

src/main/java/com/cognizant/greet/controller/GreetController.java

```
package com.cognizant.greet.controller;

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

@RestController
public class GreetController {

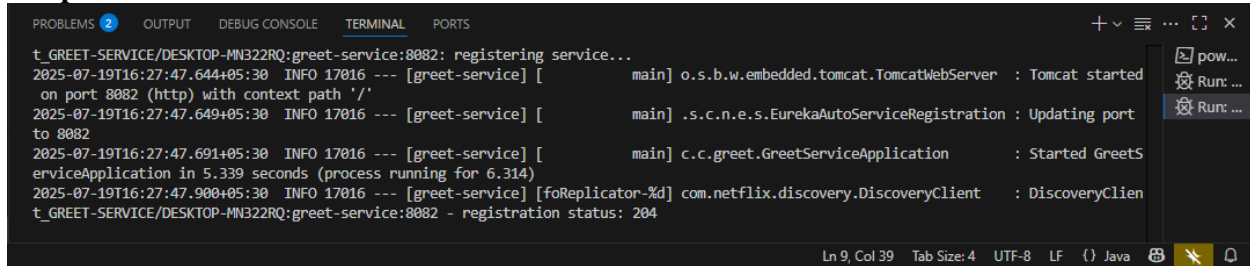
    @GetMapping("/greet")
    public String greet() {
        return "Hello World";
    }
}
```

src/main/resources/application.properties

```
server.port=8082
spring.application.name=greet-service
```

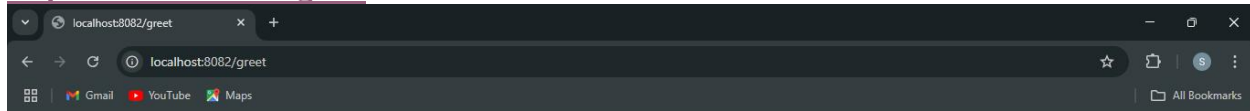
```
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
```

Output:



```
t_GREET-SERVICE/DESKTOP-MN322RQ:greet-service:8082: registering service...
2025-07-19T16:27:47.644+05:30 INFO 17016 --- [greet-service] [      main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started
on port 8082 (http) with context path '/'
2025-07-19T16:27:47.649+05:30 INFO 17016 --- [greet-service] [      main] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port
to 8082
2025-07-19T16:27:47.691+05:30 INFO 17016 --- [greet-service] [      main] c.c.greet.GreetServiceApplication : Started GreetS
erviceApplication in 5.339 seconds (process running for 6.314)
2025-07-19T16:27:47.900+05:30 INFO 17016 --- [greet-service] [foReplicator-%d] com.netflix.discovery.DiscoveryClient : DiscoveryClien
t_GREET-SERVICE/DESKTOP-MN322RQ:greet-service:8082 - registration status: 204
```

<http://localhost:8082/greet>



Hello World



src/main/java/com/cognizant/apigateway/ApiGatewayApplication.java

```
package com.cognizant.apigateway;

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);
    }
}
```

src/main/resources/application.properties

```
server.port=9090
```

```
spring.application.name=api-gateway
```

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

```
spring.cloud.gateway.discovery.locator.enabled=true
```

```
spring.cloud.gateway.discovery.locator.lower-case-service-id=true
```

src/main/java/com/cognizant/apigateway/filter/LogFilter.java

```
package com.cognizant.apigateway.filter;
```

```
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
```

```
import org.springframework.cloud.gateway.filter.GlobalFilter;
```

```
import org.springframework.cloud.gateway.filter.GatewayFilterChain;
```

```
import org.springframework.stereotype.Component;
```

```
import org.springframework.web.server.ServerWebExchange;
```

```
import reactor.core.publisher.Mono;
```

```
@Component
```

```
public class LogFilter implements GlobalFilter {
```

```
    private static final Logger logger =  
    LoggerFactory.getLogger(LogFilter.class);
```

```
    @Override
```

```
    public Mono<Void> filter(ServerWebExchange exchange, GatewayFilterChain  
chain) {
```

```
        logger.info("Incoming request path: {}",  
exchange.getRequest().getPath());
```

```
        return chain.filter(exchange);
```

```
    }
```

```
}
```

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
```

```
http://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>3.2.12</version> <!-- Latest patch -->
```

```
    </parent>
```

```
    <properties>
```

```
        <java.version>17</java.version>
```

```
        <spring-cloud.version>2023.0.1</spring-cloud.version> <!-- Compatible
```

```
-->
```

```
    </properties>
```

```
    <groupId>com.cognizant</groupId>
```

```
    <artifactId>apigateway</artifactId>
```

```
    <version>0.0.1-SNAPSHOT</version>
```

```

<name>api-gateway</name>
<description>API Gateway using Spring Cloud Gateway</description>

<dependencies>
  <!-- Spring Cloud Gateway -->
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
  </dependency>

  <!-- Eureka Client for Service Discovery -->
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-
client</artifactId>
  </dependency>

  <!-- DevTools for live reload (optional) -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>

  <!-- Testing -->
  <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>

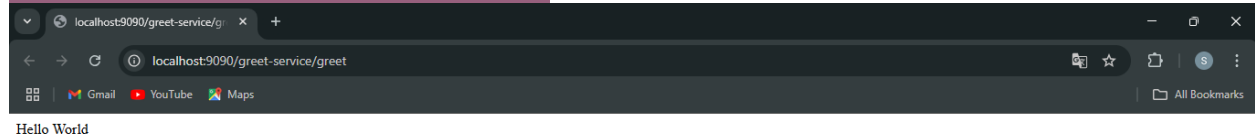
<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
      <version>3.2.4</version>
    </plugin>
  </plugins>
</build>
</project>

```

Output:

```
API-GATEWAY/DESKTOP-MN322RQ:api-gateway:9090: registering service...
2025-07-19T16:29:38.883+05:30 INFO 20136 --- [api-gateway] [infoReplicator-0] com.netflix.discovery.DiscoveryClient : DiscoveryClient_
API-GATEWAY/DESKTOP-MN322RQ:api-gateway:9090 - registration status: 204
2025-07-19T16:29:39.155+05:30 INFO 20136 --- [api-gateway] [ restartedMain] o.s.b.web.embedded.netty.NettyWebServer : Netty started on
port 9090
2025-07-19T16:29:39.158+05:30 INFO 20136 --- [api-gateway] [ restartedMain] .s.c.n.e.s.EurekaAutoServiceRegistration : Updating port to
9090
2025-07-19T16:29:39.191+05:30 INFO 20136 --- [api-gateway] [ restartedMain] c.c.apigateway.ApiGatewayApplication : Started ApiGatew
ayApplication in 7.424 seconds (process running for 8.504)
```

<http://localhost:9090/greet-service/greet>




<http://localhost:8761/>

Eureka

localhost:8761

Gmail YouTube Maps

All Bookmarks

spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2025-07-19T16:32:05 +0530
Data center	default	Uptime	00:05
		Lease expiration enabled	true
		Renews threshold	5
		Renews (last min)	8

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - DESKTOP-MN322RQ:api-gateway:9090

Eureka

localhost:8761

Gmail YouTube Maps

All Bookmarks

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - DESKTOP-MN322RQ:api-gateway:9090
GREET-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-MN322RQ:greet-service:8082

General Info

Name	Value
total-avail-memory	86mb
num-of-cpus	4
current-memory-usage	56mb (65%)
server-uptime	00:05
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/ ,
available-replicas	

Instance Info

