**MICROSERVICES WITH SPRING BOOT 3 AND SPRING CLOUD**

**Task: Creating Microservices for account and loan**

In this hands on exercise 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**

1. Create folder with employee id in D: drive
2. 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.
3. Open https://start.spring.io/ in browser

Enter form field values as specified below:

**Group**: com.cognizant

**Artifact**: account

Select the following modules

**Developer Tools > Spring Boot DevTools**

**Web > Spring Web**

1. Click generate and download the zip file
2. Extract 'account' folder from the zip and place this folder in the 'microservices' folder created earlier
3. Open command prompt in account folder and build using mvn clean

package command

1. Import this project in Eclipse and implement a controller method for getting account details based on account number.

Refer specification below:

* Method: GET
* Endpoint: /accounts/{number}
* Sample Response. Just a dummy response without any backend connectivity.   
  { number: "00987987973432", type: "savings", balance: 234343 }

**Loan Microservice**

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

* Method: GET
* Endpoint: /loans/{number}
* Sample Response. Just a dummy response without any backendconnectivity.
* { number: "H00987987972342", type: "car", loan: 400000, emi: 3258, tenure: 18 }

1. Launching this application by having account service already running
2. This launch will fail with error that the bind address is already in use
3. 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.
4. Include "server.port" property with value 8081 and try launching the application
5. Test the service with 8081 port

Now we have two microservices running on different ports.

Now we create an eureka server and register these two microservices with that server in order integrate various services.

**Code snippets for Account MicroService:**

**Model.java:**

package com.cognizant.account.model;public class Account {  
 private String number;  
 private String type;  
 private double balance;   
 public Account() {  
 }public Account(String number, String type, double balance) {  
 this.number = number;  
 this.type = type;  
 this.balance = balance;  
 }  
 public String getNumber() {  
 return number;  
 }  
 public void setNumber(String number) {  
 this.number = number;  
 }  
 public String getType() {  
 return type;  
 }  
 public void setType(String type) {  
 this.type = type;  
 }  
  
 public double getBalance() {  
 return balance;  
 }  
 public void setBalance(double balance) {  
 this.balance = balance;  
 }  
 @Override  
 public String toString() {  
 return "Account{" +  
 "number='" + number + '\'' +  
 ", type='" + type + '\'' +  
 ", balance=" + balance +  
 '}';  
 }  
}

**AccountRepository.java:**

package com.cognizant.account.repository;  
  
import com.cognizant.account.model.Account;  
import org.springframework.stereotype.Repository;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Optional;

@Repository  
public class AccountRepository {  
   
 private final Map<String, Account> accounts = new HashMap<>();  
  
 public AccountRepository() {  
 accounts.put("00987987973432", new Account("00987987973432", "savings", 234343.00));  
 accounts.put("12345678901234", new Account("12345678901234", "current", 50000.50));  
 accounts.put("98765432109876", new Account("98765432109876", "savings", 15000.75));  
 }  
public Optional<Account> findByNumber(String accountNumber) {  
 return Optional.*ofNullable*(accounts.get(accountNumber));  
 }  
}

**AccountController.java:**

package com.cognizant.account.controller;  
  
import com.cognizant.account.model.Account;  
import com.cognizant.account.repository.AccountRepository;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
@RestController // Marks this class as a REST controller  
@RequestMapping("/accounts") // Base path for all endpoints in this controller

public class AccountController {  
  
 private final AccountRepository accountRepository;public AccountController(AccountRepository accountRepository) {  
 this.accountRepository = accountRepository;  
 }  
@GetMapping("/{number}") // Maps GET requests to /accounts/{number}  
 public ResponseEntity<Account> getAccountDetails(@PathVariable("number") String number) {  
 // Use the repository to find the account by number  
 return accountRepository.findByNumber(number)  
 .map(ResponseEntity::*ok*) // If account is present, return it with 200 OK status  
 .orElse(ResponseEntity.*notFound*().build()); // If not found, return 404 Not Found status  
 }  
}

**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 // Combines @Configuration, @EnableAutoConfiguration, and @ComponentScan  
@EnableDiscoveryClient  
public class AccountApplication {  
 public static void main(String[] args) {  
 // Runs the Spring Boot application  
 SpringApplication.*run*(AccountApplication.class, args);  
 }  
}

**application.properties:**

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

**Code Snippets for Loan Service:**

**LoanModel.java**

package com.cognizant.loan.model;  
public class Loan {  
 private String number;  
 private String type;  
 private double loanAmount;  
 private double emi;  
 private int tenure;  
  
 public Loan() {  
 }  
  
 public Loan(String number, String type, double loanAmount, double emi, int tenure) {  
 this.number = number;  
 this.type = type;  
 this.loanAmount = loanAmount;  
 this.emi = emi;  
 this.tenure = tenure;  
 }  
  
 public String getNumber() {  
 return number;  
 }  
  
 public void setNumber(String number) {  
 this.number = number;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public void setType(String type) {  
 this.type = type;  
 }  
  
 public double getLoanAmount() {  
 return loanAmount;  
 }  
  
 public void setLoanAmount(double loanAmount) {  
 this.loanAmount = loanAmount;  
 }  
  
 public double getEmi() {  
 return emi;  
 }  
  
 public void setEmi(double emi) {  
 this.emi = emi;  
 }  
  
 public int getTenure() {  
 return tenure;  
 }  
  
 public void setTenure(int tenure) {  
 this.tenure = tenure;  
 }  
  
 @Override  
 public String toString() {  
 return "Loan{" +  
 "number='" + number + '\'' +  
 ", type='" + type + '\'' +  
 ", loanAmount=" + loanAmount +  
 ", emi=" + emi +  
 ", tenure=" + tenure +  
 '}';  
 }  
}

**LoanRepository.java**

package com.cognizant.loan.repository;  
  
import com.cognizant.loan.model.Loan;  
import org.springframework.stereotype.Repository;  
  
import java.util.HashMap;  
import java.util.Map;  
import java.util.Optional;  
@Repository  
public class LoanRepository {  
 // Using a HashMap to simulate a database table where loan number is the key  
 private final Map<String, Loan> loans = new HashMap<>();  
  
 public LoanRepository() {  
 // Initialize with some dummy loan data  
 loans.put("H00987987972342", new Loan("H00987987972342", "car", 400000.00, 3258.00, 18));  
 loans.put("H11223344556677", new Loan("H11223344556677", "home", 5000000.00, 45000.00, 180));  
 loans.put("H99887766554433", new Loan("H99887766554433", "personal", 100000.00, 9000.00, 12));  
 }  
public Optional<Loan> findByNumber(String loanNumber) {  
 return Optional.*ofNullable*(loans.get(loanNumber));  
 }  
}

**LoanController.java:**

package com.cognizant.loan.controller;  
  
import com.cognizant.loan.model.Loan;  
import com.cognizant.loan.repository.LoanRepository;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.PathVariable;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
@RestController // Marks this class as a REST controller  
@RequestMapping("/loans") // Base path for all endpoints in this controller  
public class LoanController {  
  
 private final LoanRepository loanRepository;  
public LoanController(LoanRepository loanRepository) {  
 this.loanRepository = loanRepository;  
 }  
@GetMapping("/{number}") // Maps GET requests to /loans/{number}  
 public ResponseEntity<Loan> getLoanDetails(@PathVariable("number") String number) {  
 // Use the repository to find the loan by number  
 return loanRepository.findByNumber(number)  
 .map(ResponseEntity::*ok*) // If loan is present, return it with 200 OK status  
 .orElse(ResponseEntity.*notFound*().build()); // If not found, return 404 Not Found status  
 }  
}

**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);  
 }  
  
}

**application.properties:**

spring.application.name=loan  
server.port=8081  
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka

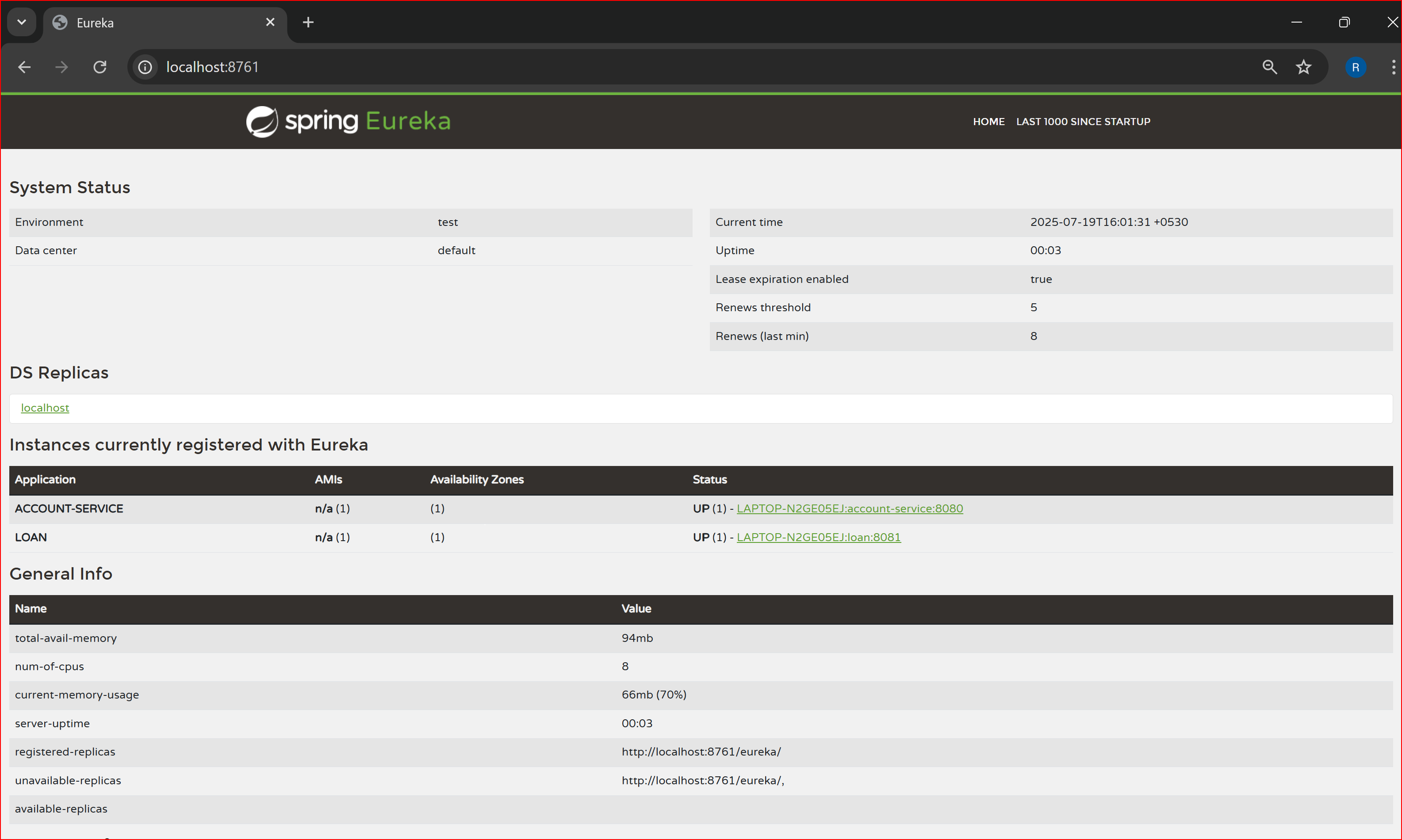
**Code snippets for eurekaserver:**

**EurekaDiscoveryServerApplication.java:**

package com.cognizant.eurekaserver;  
  
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);  
 }  
  
}

**application.properties:**

spring.application.name=eureka-dicovery-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

**output:**