1. **Sample Microservices**

**Exercise 1: Implementing Centralized Authentication with OAuth 2.1/OIDC**

**Application.yml**

spring:

security:

oauth2:

client:

registration:

google:

client-id: 505765374759-rdnq3epm9q31fsdsleqd1t22jvads19f.apps.googleusercontent.com

client-secret: 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

scope: openid, profile, email

authorization-grant-type: authorization\_code

redirect-uri: "<http://localhost:8080/login/oauth2/code/google>"

provider:

google:

authorization-uri: <https://accounts.google.com/o/oauth2/auth>

token-uri: <https://oauth2.googleapis.com/token>

user-info-uri: <https://openidconnect.googleapis.com/v1/userinfo>

user-name-attribute: sub

**Pom.yml**

<dependency>

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

<artifactId>spring-boot-starter-oauth2-client</artifactId>

</dependency>

<dependency>

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

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

</dependency>

**SecurityConfig.java**

**package** com.microservices.oauth2\_login\_demo;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.security.web.SecurityFilterChain;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

@Configuration

**public** **class** SecurityConfig {

@Bean

**public** SecurityFilterChain securityFilterChain(HttpSecurity http) **throws** Exception {

http

.authorizeHttpRequests(auth -> auth

.anyRequest().authenticated()

)

.~~oauth2Login~~();

**return** http.build();

}

}

**UserController.java**

**package** com.microservices.oauth2\_login\_demo.controller;

**import** java.security.Principal;

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

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

@RestController

**public** **class** UserController {

@GetMapping("/user")

**public** Principal user(Principal principal) {

**return** principal;

}

}

**Output:**



**Exercise 2: Configuring Authorization Servers and Resource Servers**

**SecureController.java**

**package** com.microservices.oauth2\_login\_demo.controller;

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

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

@RestController

**public** **class** SecureController {

@GetMapping("/secure")

**public** String securedData() {

**return** "Protected Data";

}

}

**ResourceServerConfig.java**

**package** com.microservices.oauth2\_login\_demo;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.web.SecurityFilterChain;

@Configuration

**public** **class** ResourceServerConfig {

@Bean

**public** SecurityFilterChain resourceFilterChain(HttpSecurity http) **throws** Exception {

http

.authorizeHttpRequests(auth -> auth

.anyRequest().authenticated()

)

.oauth2ResourceServer(oauth2 -> oauth2

.jwt()

);

**return** http.build();

}

}

**Application.yml**

redirect-uri: "http://localhost:8080/login/oauth2/code/google"

**Exercise 3: Using JSON Web Tokens (JWT) for Secure Communication**

**Appication.yml**

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<dependency>

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

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

</dependency>

**JwtTokenProvider.java**

**package** com.microservices.oauth2\_login\_demo.security;

**import** java.util.Date;

**import** java.util.List;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.stereotype.Component;

**import** com.microservices.oauth2\_login\_demo.config.JwtConfig;

**import** io.jsonwebtoken.Claims;

**import** io.jsonwebtoken.Jwts;

**import** io.jsonwebtoken.SignatureAlgorithm;

@Component

**public** **class** JwtTokenProvider {

@Autowired

**private** JwtConfig jwtConfig;

**public** String createToken(String username) {

Claims claims = Jwts.*claims*().setSubject(username);

Date now = **new** Date();

Date expiry = **new** Date(now.getTime() + 3600000); // 1 hour

**return** Jwts.*builder*()

.setClaims(claims)

.setIssuedAt(now)

.setExpiration(expiry)

.~~signWith~~(SignatureAlgorithm.***HS256***, jwtConfig.getSecret())

.compact();

}

**public** **boolean** validateToken(String token) {

**try** {

Jwts.~~parser~~().~~setSigningKey~~(jwtConfig.getSecret()).parseClaimsJws(token);

**return** **true**;

} **catch** (Exception e) {

**return** **false**;

}

}

**public** Authentication getAuthentication(String token) {

String username = Jwts.~~parser~~()

.~~setSigningKey~~(jwtConfig.getSecret())

.parseClaimsJws(token)

.getBody()

.getSubject();

**return** **new** UsernamePasswordAuthenticationToken(username, "", List.*of*());

}

}

**JwtTokenFilter.java**

**package** com.microservices.oauth2\_login\_demo.security;

**import** jakarta.servlet.FilterChain;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.stereotype.Component;

**import** org.springframework.web.filter.OncePerRequestFilter;

**import** java.io.IOException;

@Component

**public** **class** JwtTokenFilter **extends** OncePerRequestFilter {

@Autowired

**private** JwtTokenProvider jwtTokenProvider;

@Override

**protected** **void** doFilterInternal(HttpServletRequest request,

HttpServletResponse response,

FilterChain filterChain) **throws** ServletException, IOException {

String token = resolveToken(request);

**if** (token != **null** && jwtTokenProvider.validateToken(token)) {

Authentication auth = jwtTokenProvider.getAuthentication(token);

SecurityContextHolder.*getContext*().setAuthentication(auth);

}

filterChain.doFilter(request, response);

}

**private** String resolveToken(HttpServletRequest request) {

String bearer = request.getHeader("Authorization");

**if** (bearer != **null** && bearer.startsWith("Bearer ")) {

**return** bearer.substring(7);

}

**return** **null**;

}

}

**JwtConfig.java**

**package** com.microservices.oauth2\_login\_demo.config;

**import** org.springframework.beans.factory.annotation.Value;

**import** org.springframework.context.annotation.Configuration;

@Configuration

**public** **class** JwtConfig {

@Value("${spring.security.jwt.secret}")

**private** String secret;

**public** String getSecret() {

**return** secret;

}

}

**AuthController.java**

**package** com.microservices.oauth2\_login\_demo.controller;

**import** java.util.Map;

**import** org.springframework.beans.factory.annotation.Autowired;

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

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

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

**import** com.microservices.oauth2\_login\_demo.security.JwtTokenProvider;

@RestController

**public** **class** AuthController {

@Autowired

**private** JwtTokenProvider jwtTokenProvider;

@PostMapping("/login")

**public** Map<String, String> login(@RequestParam String username) {

String token = jwtTokenProvider.createToken(username);

**return** Map.*of*("token", token);

}

}

**SecureController.java**

@RestController

**public** **class** SecureController {

@GetMapping("/secure")

**public** String secure() {

**return** "You are authenticated with a locally generated JWT";

}

}

**SecurityConfig.java**

@Configuration

**public** **class** SecurityConfig {

@Autowired

**private** JwtTokenFilter jwtTokenFilter;

@Bean

**public** SecurityFilterChain securityFilterChain(HttpSecurity http) **throws** Exception {

http

.authorizeHttpRequests(auth -> auth

.anyRequest().authenticated()

)

.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.**class**);

**return** http.build();

}

}

**ResourseServerConfig.java**

**package** com.microservices.oauth2\_login\_demo;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.security.config.annotation.web.builders.HttpSecurity;

**import** org.springframework.security.web.SecurityFilterChain;

@Configuration

**public** **class** ResourceServerConfig {

@Bean

**public** SecurityFilterChain resourceFilterChain(HttpSecurity http) **throws** Exception {

http

.authorizeHttpRequests(auth -> auth

.anyRequest().authenticated()

)

.oauth2ResourceServer(oauth2 -> oauth2

.~~jwt~~()

);

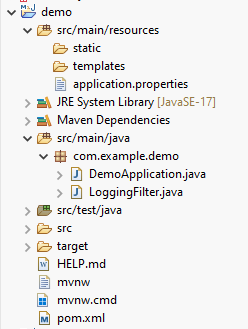
**return** http.build();

}

}

**Docs 2 Edge\_Services\_API\_Gateway\_Exercises**

**Exercise 1: Implementing Edge Services for Routing and Filtering**



**Application.properties**

spring.application.name=demo

spring.cloud.gateway.routes[0].id=example\_route

spring.cloud.gateway.routes[0].uri=http://example.org

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

spring.main.web-application-type=reactive

**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>3.1.9</version>

<relativePath/>

</parent>

<groupId>com.example</groupId>

<artifactId>demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>demo</name>

<description>Demo project for Spring Boot Gateway</description>

<properties>

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

<spring-cloud.version>2022.0.5</spring-cloud.version>

</properties>

<dependencies>

<!-- Spring Boot Web (optional for fallback or error handling endpoints)

<dependency>

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

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

</dependency>

-->

<!-- Spring Cloud Gateway -->

<dependency>

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

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

<!-- Developer Tools -->

<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>

</plugin>

</plugins>

</build>

</project>

**LoggingFilter.java**

**package** com.example.demo;

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

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

**import** org.springframework.core.annotation.Order;

**import** org.springframework.stereotype.Component;

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

**import** reactor.core.publisher.Mono;

@Component

@Order(1)

**public** **class** LoggingFilter **implements** GlobalFilter {

@Override

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

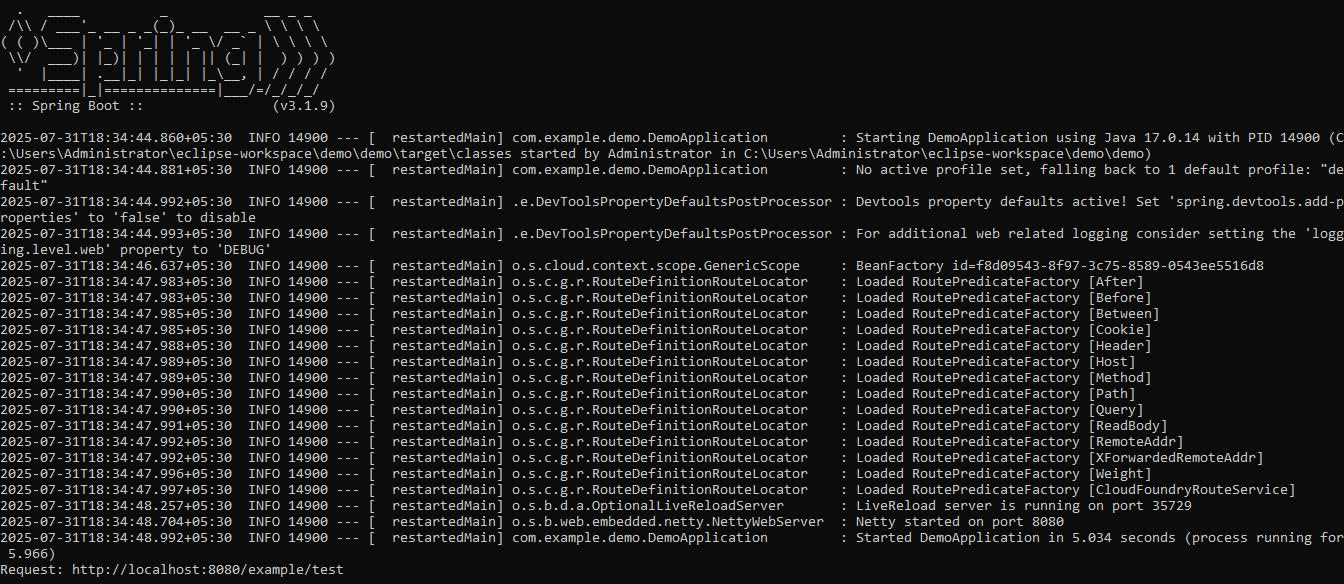
System.***out***.println("Request: " + exchange.getRequest().getURI());

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

}

}

**Output:**



**Exercise 2: Load Balancing in an API Gateway**

**LoadBalanceConfiguration.java**

**package** com.example.demo;

**import** org.springframework.cloud.client.loadbalancer.LoadBalancerClientFactory;

**import** org.springframework.cloud.client.loadbalancer.ServiceInstanceListSupplier;

**import** org.springframework.cloud.client.loadbalancer.reactive.ReactorServiceInstanceLoadBalancer;

**import** org.springframework.cloud.loadbalancer.core.RandomLoadBalancer;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** org.springframework.core.env.Environment;

@Configuration

**public** **class** LoadBalancerConfiguration {

@Bean

**public** ReactorServiceInstanceLoadBalancer randomLoadBalancer(Environment environment,

LoadBalancerClientFactory clientFactory) {

String serviceId = environment.getProperty(LoadBalancerClientFactory.PROPERTY\_NAME);

**return** **new** RandomLoadBalancer(

clientFactory.getLazyProvider(serviceId, ServiceInstanceListSupplier.**class**),

serviceId

);

}

}

**MockServiceController.java**

**package** com.example.demo;

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

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

@RestController

**public** **class** MockServiceController {

@GetMapping("/loadbalanced/instance1")

**public** String instance1() {

**return** "Response from Mock Instance 1";

}

@GetMapping("/loadbalanced/instance2")

**public** String instance2() {

**return** "Response from Mock Instance 2";

}

}

**Application.yml**

server:

port: 8080

spring:

cloud:

gateway:

routes:

- id: my-service

uri: lb://my-service

predicates:

- Path=/api/\*\*

**Pom.xml**

<dependencies>

<!-- Spring Cloud Gateway -->

<dependency>

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

<artifactId>spring-cloud-starter-gateway</artifactId>

</dependency>

<!-- Spring Cloud LoadBalancer -->

<dependency>

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

<artifactId>spring-cloud-starter-loadbalancer</artifactId>

</dependency>

<!-- DevTools (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>

<!-- Spring Cloud BOM Import -->

<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>

**StaticServiceInstanceSupplier.java**

**package** com.example.demo;

**import** org.springframework.cloud.client.DefaultServiceInstance;

**import** org.springframework.cloud.client.ServiceInstance;

**import** org.springframework.cloud.client.loadbalancer.ServiceInstanceListSupplier;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.Configuration;

**import** reactor.core.publisher.Flux;

**import** java.util.List;

@Configuration

**public** **class** StaticServiceInstanceSupplier {

@Bean

**public** ServiceInstanceListSupplier staticListSupplier() {

**return** **new** ServiceInstanceListSupplier() {

@Override

**public** String getServiceId() {

**return** "example-service";

}

@Override

**public** Flux<List<ServiceInstance>> get() {

**return** Flux.*just*(List.*of*(

**new** DefaultServiceInstance("instance1", "example-service", "localhost", 8081, **false**),

**new** DefaultServiceInstance("instance2", "example-service", "localhost", 8082, **false**)

));

}

};

}

}

**Exercise 3: Resilience Patterns in an API Gateway**

**Pom.xml**

<dependency>

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

<artifactId>spring-cloud-starter-circuitbreaker-reactor-resilience4j</artifactId>

</dependency>

**Application.yml**

server:

port: 8080

spring:

cloud:

gateway:

routes:

- id: my-service

uri: <http://localhost:8081>

predicates:

- Path=/api/\*\*

filters:

- name: CircuitBreaker

args:

name: myCircuitBreaker

fallbackUri: forward:/fallback

**FallbackController.java**

**package** com.example.demo;

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

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

@RestController

**public** **class** FallbackController {

@GetMapping("/fallback")

**public** String fallback() {

**return** "Fallback response: service is temporarily unavailable.";

}

}

**Docs 3 Microservices with Springboot 3.0**

3.0 3.0

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

**Order.java**

**package** com.order\_service.model;

**import** jakarta.persistence.\*;

**import** lombok.\*;

@Entity

@Table(name = "orders") // 👈 this fixes the SQL syntax error

@Data

@NoArgsConstructor

@AllArgsConstructor

**public** **class** Order {

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** Long id;

**private** String product;

**private** Double amount;

**private** Long userId;

}

**OrderService.java**

**package** com.order\_service.service;

**import** org.springframework.stereotype.Service;

**import** com.order\_service.feign.UserClient;

**import** com.order\_service.model.Order;

**import** com.order\_service.model.OrderResponse;

**import** com.order\_service.model.User;

**import** com.order\_service.repository.OrderRepository;

**import** lombok.RequiredArgsConstructor;

@Service

@RequiredArgsConstructor

**public** **class** OrderService {

**private** **final** OrderService orderService;

**private** **final** OrderRepository orderRepository;

**private** **final** UserClient userClient;

**public** Order saveOrder(Order order) {

**return** orderRepository.save(order);

}

**public** OrderResponse getOrderWithUser(Long orderId) {

Order order = orderRepository.findById(orderId).orElseThrow();

User user = userClient.getUserById(order.getUserId());

**return** **new** OrderResponse(order, user);

}

}

**OrderResponse.java**

**package** com.order\_service.model;

**import** lombok.\*;

@Data

@AllArgsConstructor

**public** **class** OrderResponse {

**private** Order order;

**private** User user;

}

**OrderController.java**

**package** com.order\_service.controller;

**import** com.order\_service.model.Order;

**import** com.order\_service.model.OrderResponse;

**import** com.order\_service.service.OrderService;

**import** lombok.RequiredArgsConstructor;

**import** org.springframework.http.\*;

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

@RestController

@RequestMapping("/api/orders")

@RequiredArgsConstructor

**public** **class** OrderController {

**private** **final** OrderService orderService;

@PostMapping

**public** ResponseEntity<Order> create(@RequestBody Order order) {

**return** **new** ResponseEntity<>(orderService.saveOrder(order), HttpStatus.***CREATED***);

}

@GetMapping("/{id}")

**public** ResponseEntity<OrderResponse> getOrder(@PathVariable Long id) {

**return** ResponseEntity.*ok*(orderService.getOrderWithUser(id));

}

}

**OrderServiceApplication.java**

**package** com.order\_service;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** OrderServiceApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(OrderServiceApplication.**class**, args);

}

}

**UserClient.java**

**package** com.order\_service.feign;

**import** com.order\_service.model.User;

**import** org.springframework.cloud.openfeign.FeignClient;

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

@FeignClient(name = "user-service", url = "<http://localhost:8080>")

**public** **interface** UserClient {

@GetMapping("/api/users/{id}")

User getUserById(@PathVariable("id") Long id);

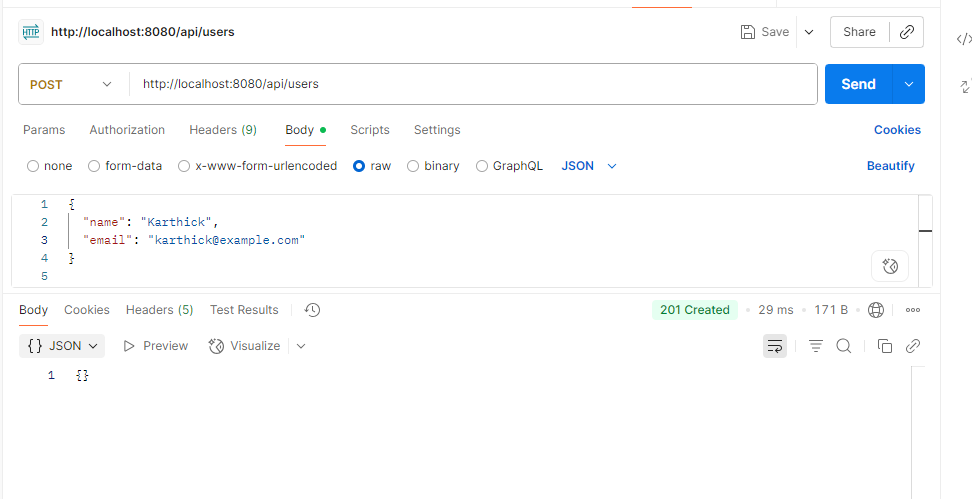
}

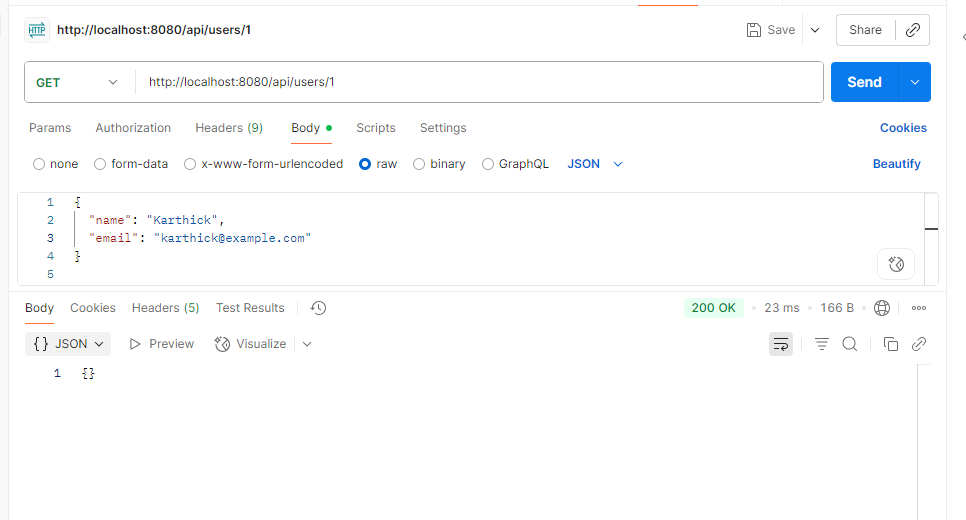
**Sql**

CREATE DATABASE userdb;

CREATE DATABASE orderdb;

**Output:**





**2. Inventory Management System with Service Discovery**

**DiscoveryServerApplication.java**

**package** com.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** DiscoveryServerApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(DiscoveryServerApplication.**class**, args);

}

}

**Application.properties**

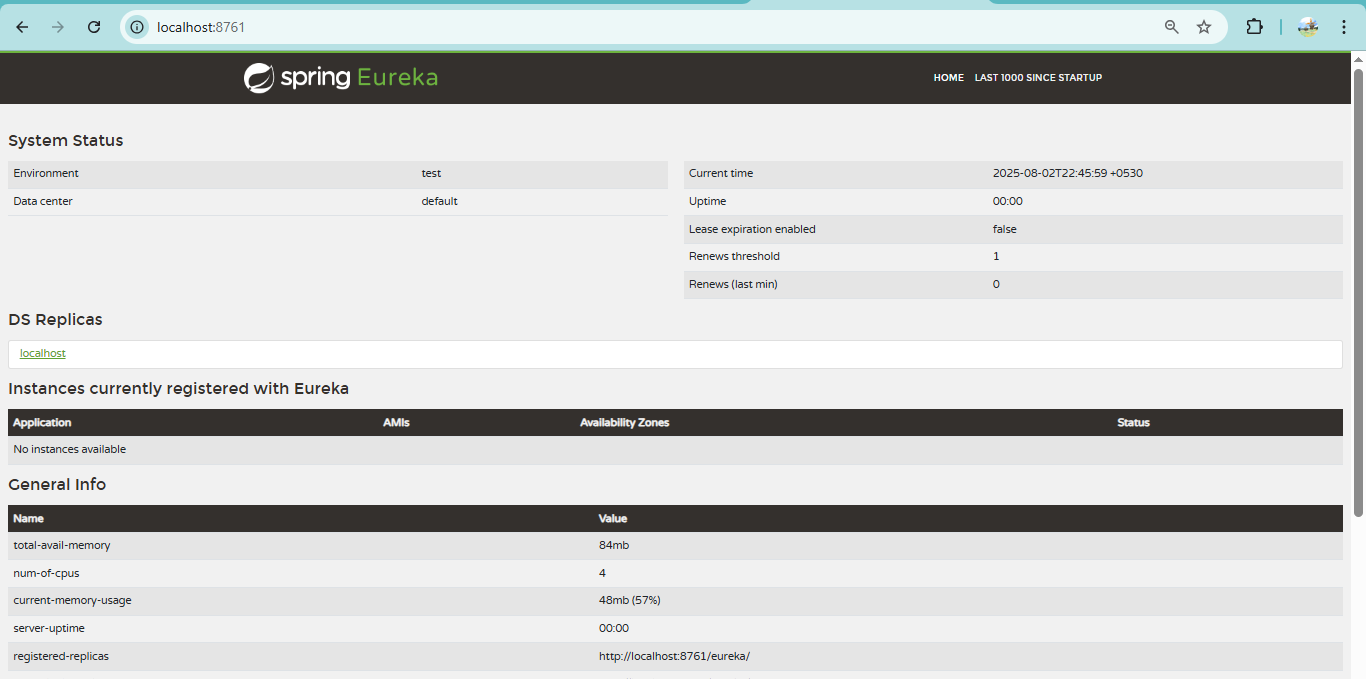
spring.application.name=Discovery Server

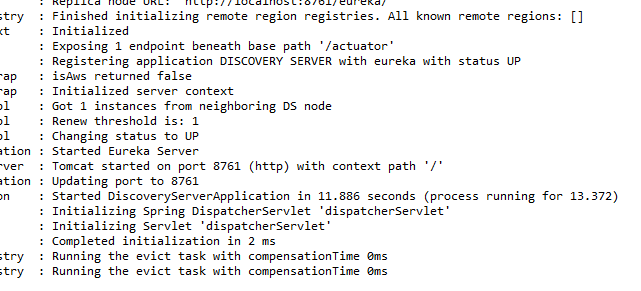
server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

**Output:**





**3. Implement an API Gateway**

**Api gateway application.properties**

spring.application.name=api-gateway

server.port=8080

# Eureka Discovery

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

# Route definitions

spring.cloud.gateway.routes[0].id=customer-service

spring.cloud.gateway.routes[0].uri=lb://CUSTOMER-SERVICE

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

spring.cloud.gateway.routes[0].filters[0]=RewritePath=/api/customer/(?<segment>.\*), /${segment}

spring.cloud.gateway.routes[0].filters[1]=RequestRateLimiter=redis-rate-limiter.replenishRate=10, redis-rate-limiter.burstCapacity=20

spring.cloud.gateway.routes[1].id=billing-service

spring.cloud.gateway.routes[1].uri=lb://BILLING-SERVICE

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

spring.cloud.gateway.routes[1].filters[0]=RewritePath=/api/billing/(?<segment>.\*), /${segment}

**BillingController.java**

**package** com.billing.service.billing\_service.controller;

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

@RestController

@RequestMapping("/summary")

**public** **class** BillingController {

@GetMapping

**public** String getBillingSummary() {

**return** "Total Due: ₹3,250.00, Last Payment: ₹1,000.00";

}

}

**Application.properties**

spring.application.name=billing-service

server.port=8083

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

**CustomerController.java**

**package** com.customer.service.customer\_service.controller;

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

**import** java.util.List;

@RestController

@RequestMapping("/all")

**public** **class** CustomerController {

@GetMapping

**public** List<String> getAllCustomers() {

**return** List.*of*("Lakshmi", "Arun", "Meena");

}

}

**Application.properties**

spring.application.name=customer-service

server.port=8082

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

**Eureka server application.properties**

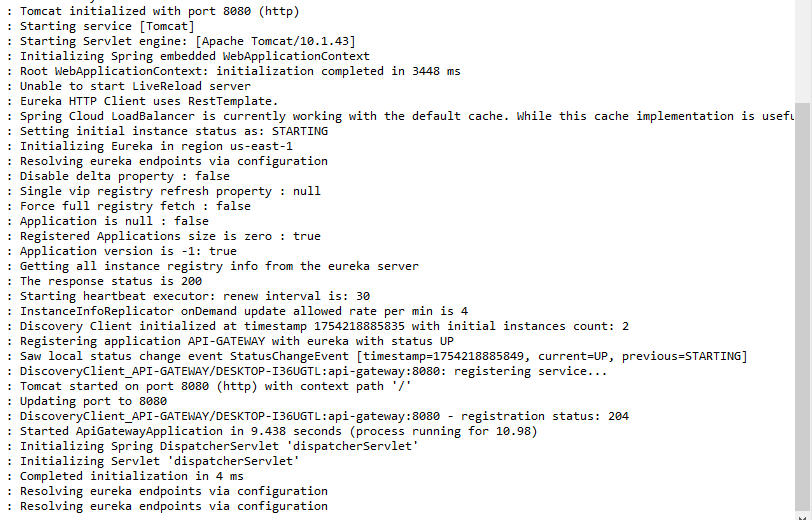
spring.application.name=Discovery Server

server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

**Output:**



**4. Resilient Microservices with Circuit Breaker**

**Api gateway application.properties**

spring.cloud.gateway.routes[0].id=customer-service

spring.cloud.gateway.routes[0].uri=lb://CUSTOMER-SERVICE

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

spring.cloud.gateway.routes[0].filters[0]=RewritePath=/api/customer/(?<segment>.\*), /${segment}

spring.cloud.gateway.routes[0].filters[1]=CircuitBreaker=name=customerCB, fallbackUri=forward:/fallback/customer

**FallbackController.java**

**package** com.gateway.api\_gateway.controller;

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

**import** reactor.core.publisher.Mono;

@RestController

@RequestMapping("/fallback")

**public** **class** FallbackController {

@GetMapping("/customer")

**public** Mono<String> customerFallback() {

**return** Mono.*just*("Customer Service is currently unavailable. Please try again later.");

}

@GetMapping("/billing")

**public** Mono<String> billingFallback() {

**return** Mono.*just*("Billing Service is currently down. Please try again soon.");

}

}

**Pom.xml**

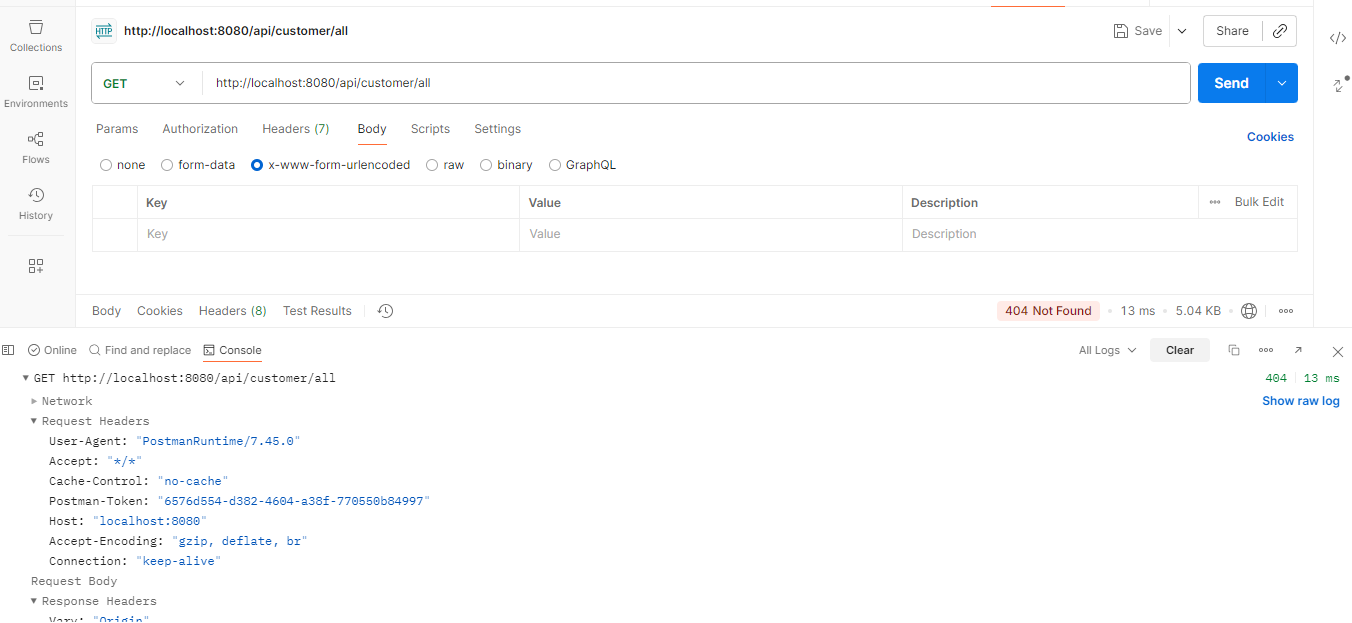
<dependency>

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

<artifactId>spring-cloud-starter-circuitbreaker-reactor-resilience4j</artifactId>

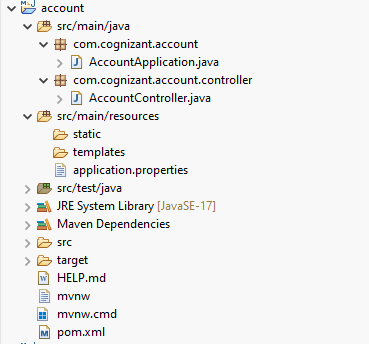
</dependency>

**Output:**



**Docs 4 05-01-microservices-handson**

**Creating Microservices for account and loan**



**AccountController.java**

**package** com.cognizant.account.controller;

**import** java.util.Map;

**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

@RequestMapping("/accounts")

**public** **class** AccountController {

@GetMapping("/{number}")

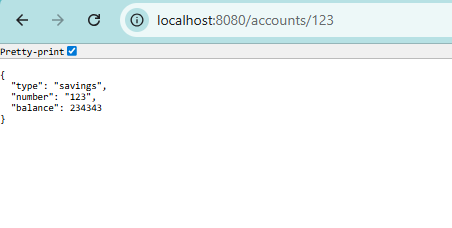
**public** Map<String, Object> getAccount(@PathVariable String number) {

**return** Map.*of*("number", number, "type", "savings", "balance", 234343);

}

}

**Output:**



**LoanController.java**

**package** com.cognizant.loan.controller;

**import** java.util.Map;

**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

@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

);

}

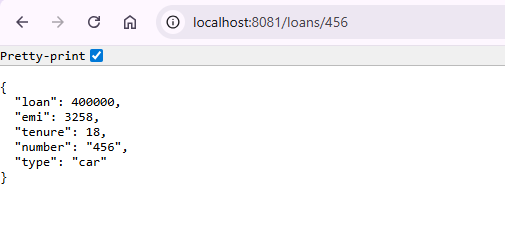
}

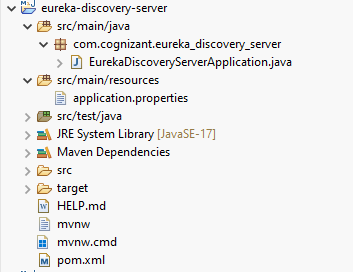
**Application.properties**

spring.application.name=loan

server.port=8081

**Output:**





**Application.properties**

spring.application.name=eureka-discovery-server

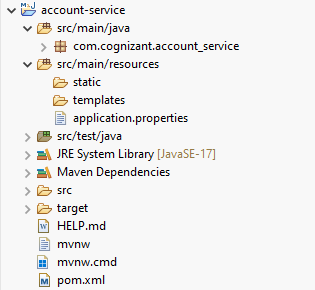
server.port=8761

eureka.client.register-with-eureka=false

eureka.client.fetch-registry=false

logging.level.com.netflix.eureka=OFF

logging.level.com.netflix.discovery=OFF



**Application.properties**

spring.application.name=account-service

server.port=8080

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

**AccountController.java**

**package** com.cognizant.account\_service.controller;

**import** java.util.Map;

**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

@RequestMapping("/accounts")

**public** **class** AccountController {

@GetMapping("/{number}")

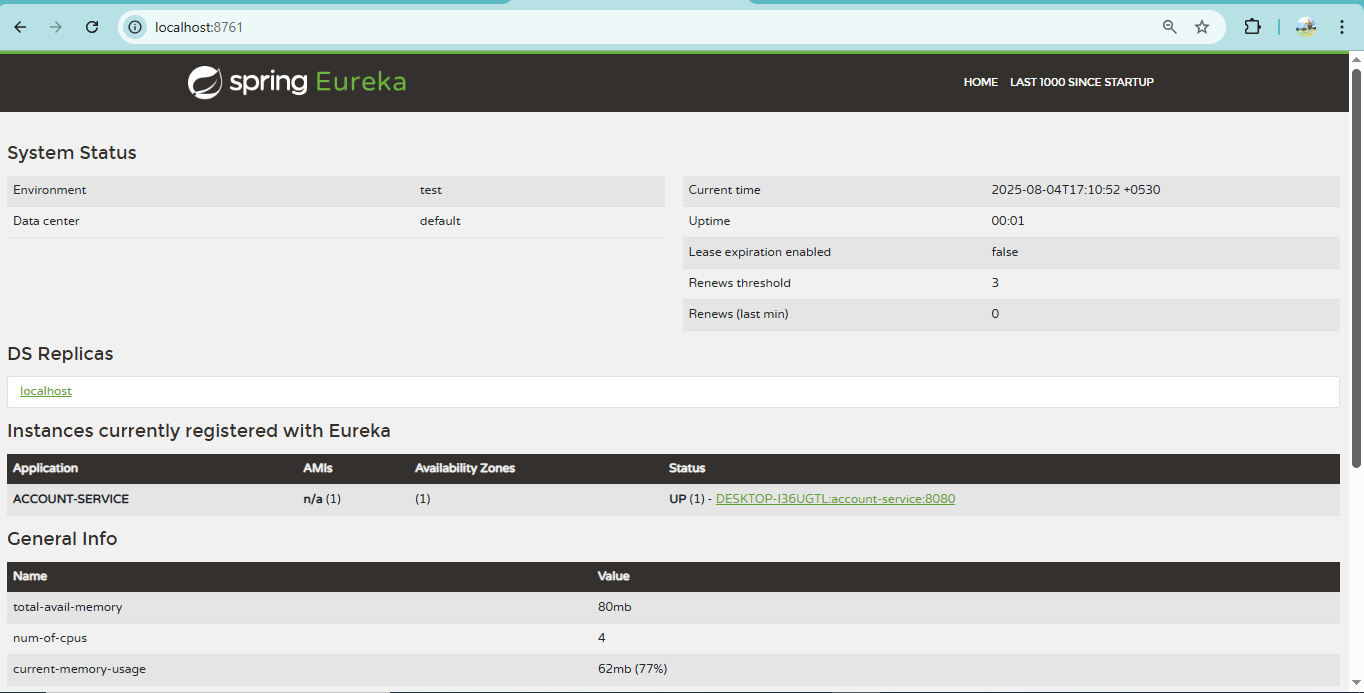
**public** Map<String, Object> getAccount(@PathVariable String number) {

**return** Map.*of*("number", number, "type", "savings", "balance", 234343);

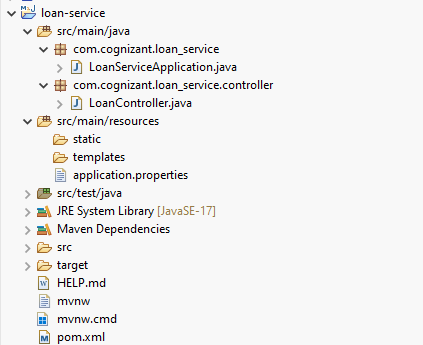
}

}

**Output:**



**Loan-service**



**Application.properties**

spring.application.name=loan-service

server.port=8081

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

**LoanController.java**

**package** com.cognizant.loan\_service.controller;

**import** java.util.Map;

**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

@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,

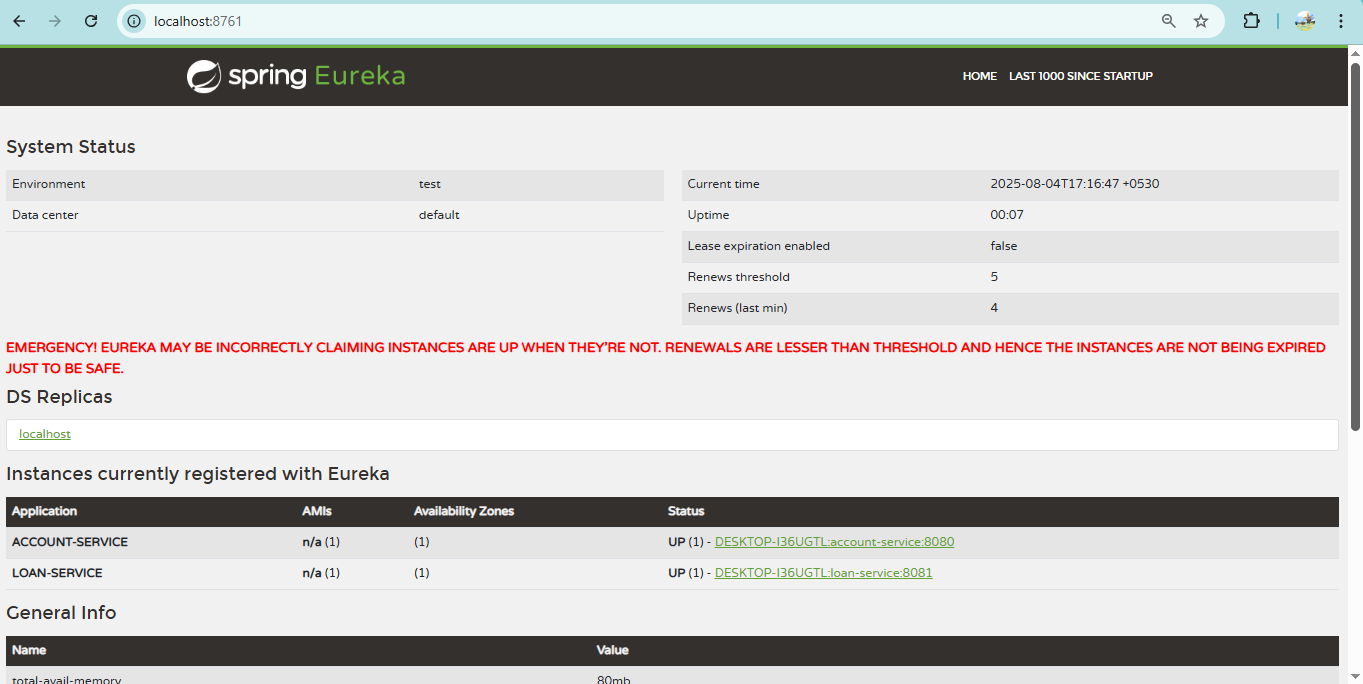
"tenure", 18

);

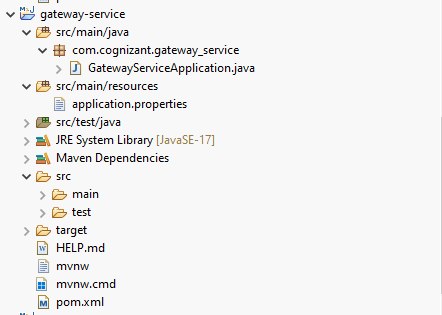
}

}

**Output**



**Gateway-service**



**Application.properties**

server.port=9090

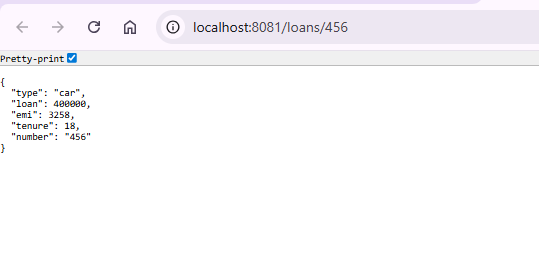
spring.application.name=gateway-service

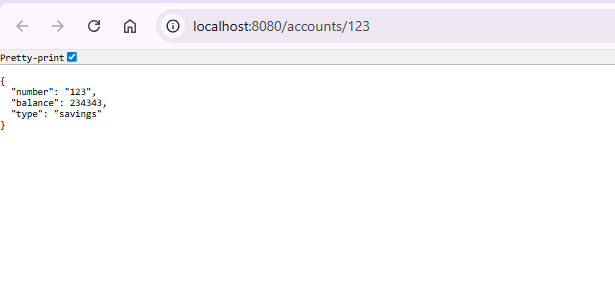
eureka.client.service-url.defaultZone=http://localhost:8761/eureka/

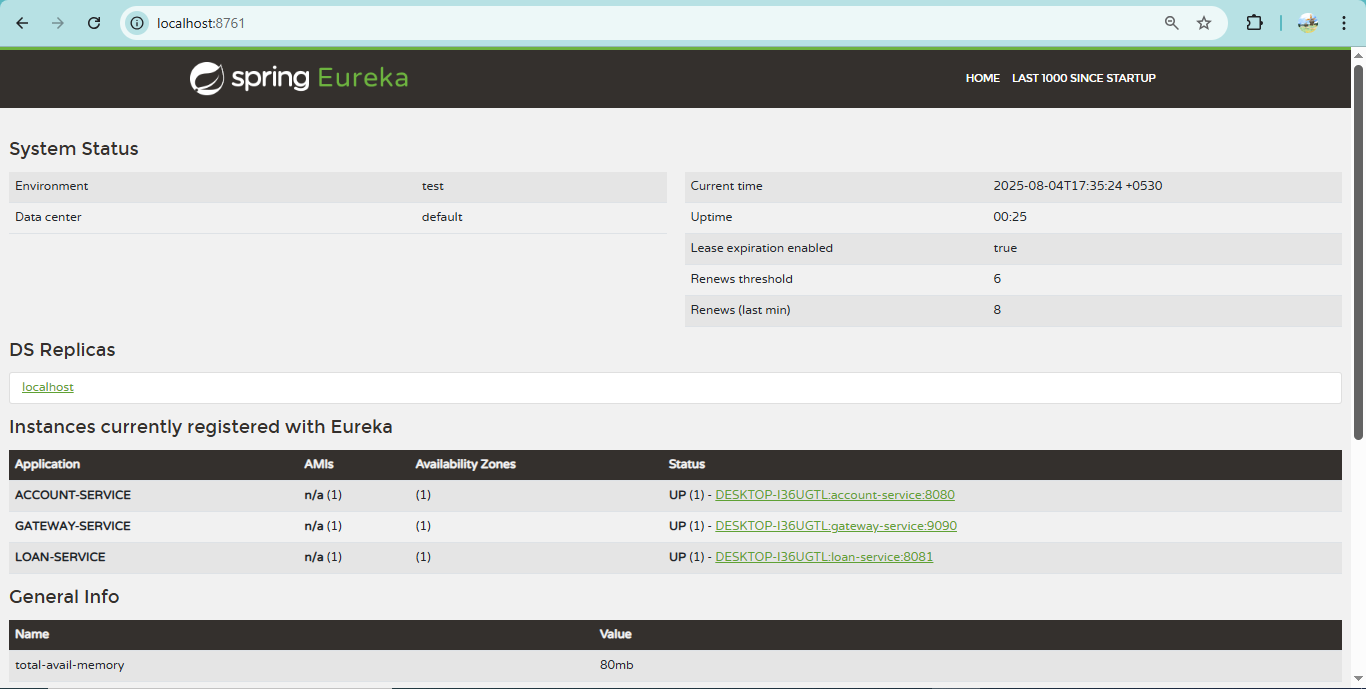
spring.cloud.gateway.discovery.locator.enabled=true

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

**Output:**

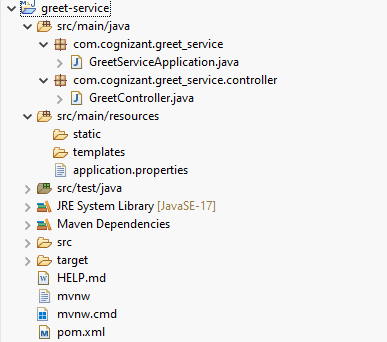






**Create Eureka Discovery Server and register microservices**

**Greet-service**



**GreetController.java**

**package** com.cognizant.greet\_service.controller;

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

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

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

@RestController

@RequestMapping("/greet")

**public** **class** GreetController {

@GetMapping

**public** String greet() {

**return** "Hello from greet-service!";

}

}

**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>3.2.4</version>

<relativePath/>

</parent>

<groupId>com.cognizant</groupId>

<artifactId>greet-service</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>greet-service</name>

<description>greet-service project for Spring Boot</description>

<properties>

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

<spring-cloud.version>2023.0.1</spring-cloud.version>

</properties>

<dependencies>

<!-- REST API -->

<dependency>

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

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

</dependency>

<!-- Eureka Client -->

<dependency>

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

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

</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>

</plugin>

</plugins>

</build>

</project>

**Output:**

