**WEEK-5**

**MICROSERVICES**

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

pom.xml:

<dependency>

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

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

</dependency>

application.properties:

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/

Custom Filter: LoggingFilter.java

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

import org.springframework.core.annotation.Order;

import org.springframework.stereotype.Component;

import org.springframework.web.server.ServerWebExchange;

import reactor.core.publisher.Mono;

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

@Component

@Order(1)

public class LoggingFilter implements GlobalFilter {

@Override

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

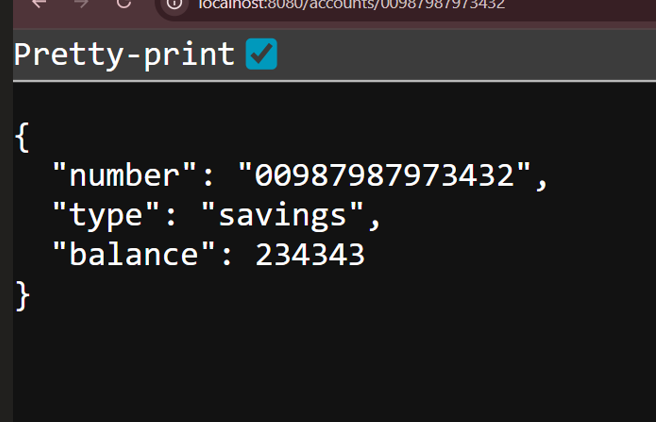
System.out.println("Request: " + exchange.getRequest().getURI());

return chain.filter(exchange);

}

}

OUTPUT



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

pom.xml:

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

application.properties:

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

spring.cloud.gateway.routes[0].uri=lb://example-service

spring.cloud.gateway.routes[0].predicates[0]=Path=/loadbalanced/

LoadBalancerConfiguration.java:

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

import org.springframework.cloud.client.ServiceInstance;

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

import org.springframework.cloud.client.loadbalancer.ReactorLoadBalancer;

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

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 ReactorLoadBalancer<ServiceInstance> randomLoadBalancer(

Environment environment,

LoadBalancerClientFactory loadBalancerClientFactory) {

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

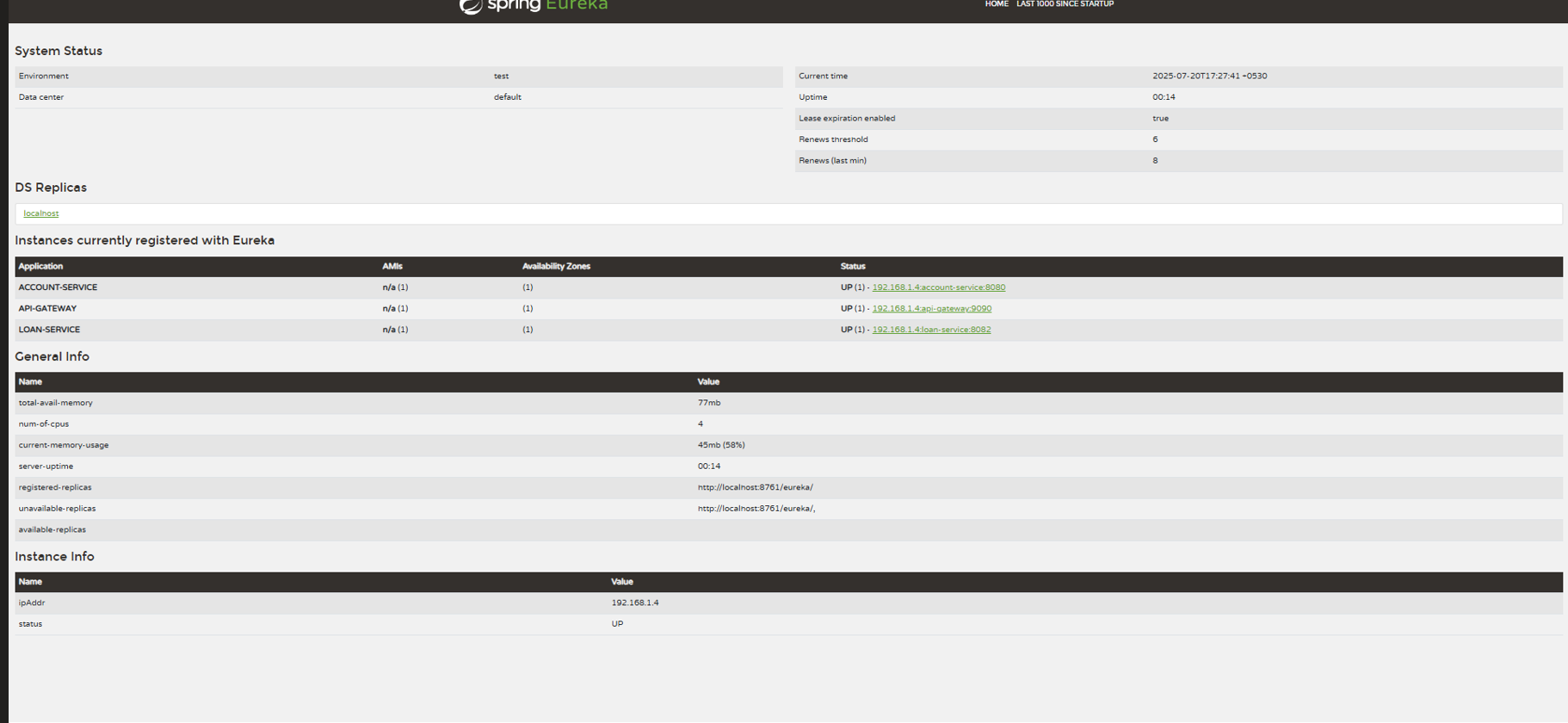
return new RandomLoadBalancer(

loadBalancerClientFactory.getLazyProvider(name, ServiceInstanceListSupplier.class), name);

}

}

OUTPUT:



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

pom.xml:

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot2</artifactId>

</dependency>

application.properties:

resilience4j.circuitbreaker.instances.exampleCircuitBreaker.registerHealthIndicator=true

resilience4j.circuitbreaker.instances.exampleCircuitBreaker.slidingWindowSize=10

resilience4j.circuitbreaker.instances.exampleCircuitBreaker.failureRateThreshold=50

ResilienceConfiguration.java:

import io.github.resilience4j.circuitbreaker.CircuitBreakerConfig;

import io.github.resilience4j.timelimiter.TimeLimiterConfig;

import org.springframework.cloud.circuitbreaker.resilience4j.ReactiveResilience4JCircuitBreakerFactory;

import org.springframework.cloud.circuitbreaker.resilience4j.Resilience4JConfigBuilder;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.cloud.circuitbreaker.resilience4j.Customizer;

@Configuration

public class ResilienceConfiguration {

@Bean

public Customizer<ReactiveResilience4JCircuitBreakerFactory> defaultCustomizer() {

return factory -> factory.configureDefault(id ->

new Resilience4JConfigBuilder(id)

.circuitBreakerConfig(CircuitBreakerConfig.ofDefaults())

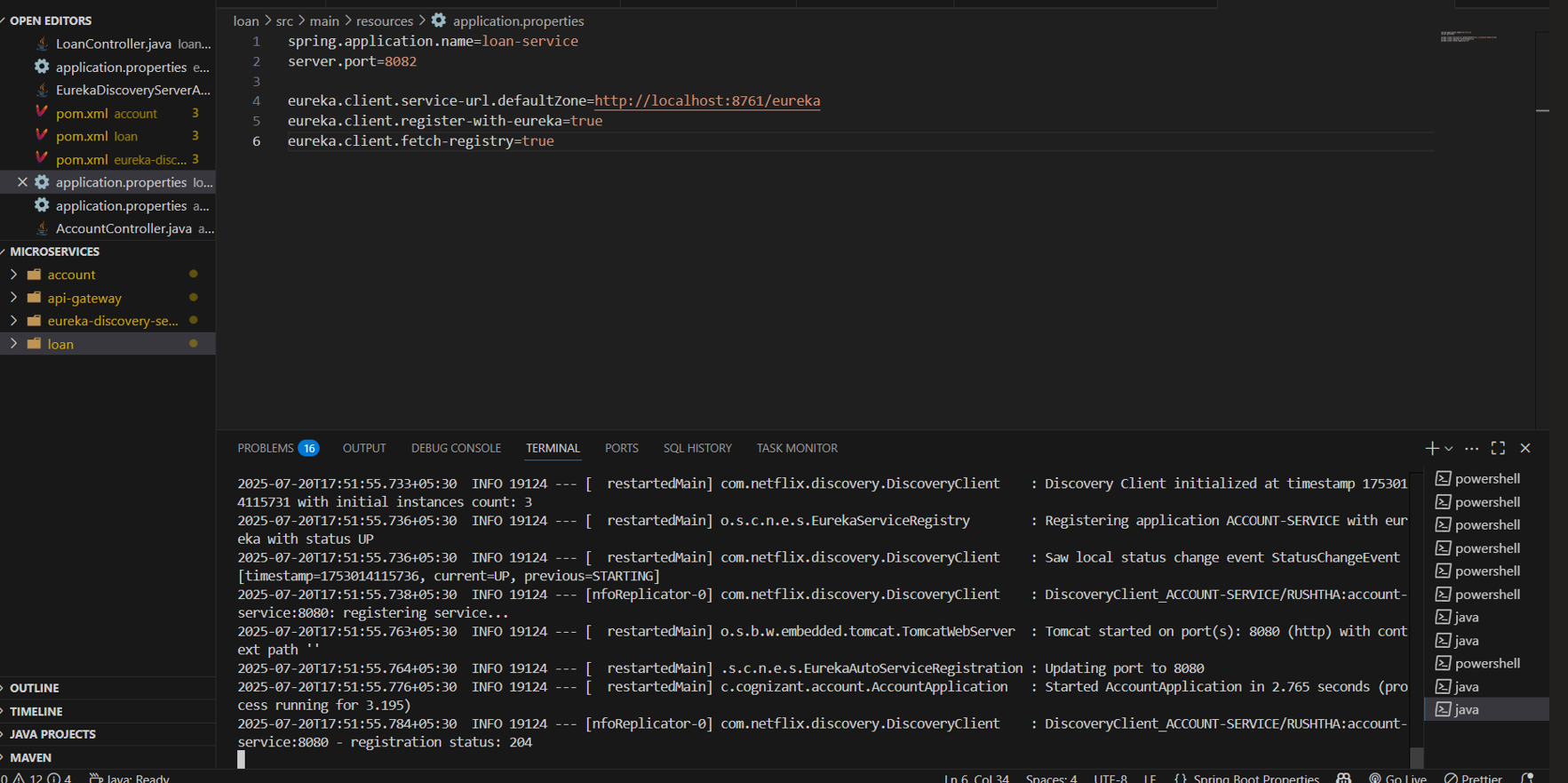
.timeLimiterConfig(TimeLimiterConfig.ofDefaults())

.build());

}

}

OUTPUT:



**Sample Microservices Exercises**

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

pom.xml Dependencies:

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

application.yml:

spring:

security:

oauth2:

client:

registration:

my-client:

client-id: YOUR\_CLIENT\_ID

client-secret: YOUR\_CLIENT\_SECRET

scope: openid, profile, email

authorization-grant-type: authorization\_code

redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"

provider:

my-provider:

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

SecurityConfig.java:

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2Login();

}

}

UserController.java:

@RestController

public class UserController {

@GetMapping("/user")

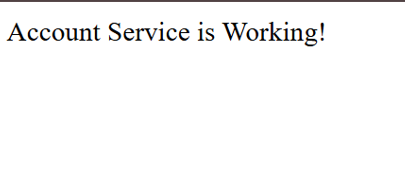
public Principal user(Principal principal) {

return principal;

}

}

OUTPUT:



**Exercise 2: Authorization Server + Resource Server Setup**

pom.xml Dependencies

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-oauth2-resource-server</artifactId>

</dependency>

application.yml:

spring:

security:

oauth2:

resourceserver:

jwt:

issuer-uri:

ResourceServerConfig.java:

@EnableWebSecurity

public class ResourceServerConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

.and()

.oauth2ResourceServer()

.jwt();

}

}

SecureController.java:

@RestController

public class SecureController {

@GetMapping("/secure")

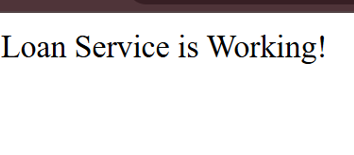
public String secure() {

return "This is a secure endpoint";

}

}

OUTPUT:



**Exercise 3: JWT for Secure Communication**

pom.xml Dependencies

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

application.yml

yaml

spring:

security:

jwt:

secret: YOUR\_SECRET\_KEY

JwtConfig.java

java

CopyEdit

@Configuration

public class JwtConfig {

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

private String secret;

public String getSecret() {

return secret;

}

}

JwtTokenProvider.java

java

CopyEdit

@Component

public class JwtTokenProvider {

@Autowired

private JwtConfig jwtConfig;

public String createToken(String username) {

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

Date now = new Date();

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

return Jwts.builder()

.setClaims(claims)

.setIssuedAt(now)

.setExpiration(validity)

.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, "", new ArrayList<>());

}

}

JwtTokenFilter.java

java

CopyEdit

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 bearerToken = request.getHeader("Authorization");

return (bearerToken != null && bearerToken.startsWith("Bearer ")) ? bearerToken.substring(7) : null;

}

}

SecurityConfig.java

java

CopyEdit

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtTokenFilter jwtTokenFilter;

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests()

.anyRequest().authenticated()

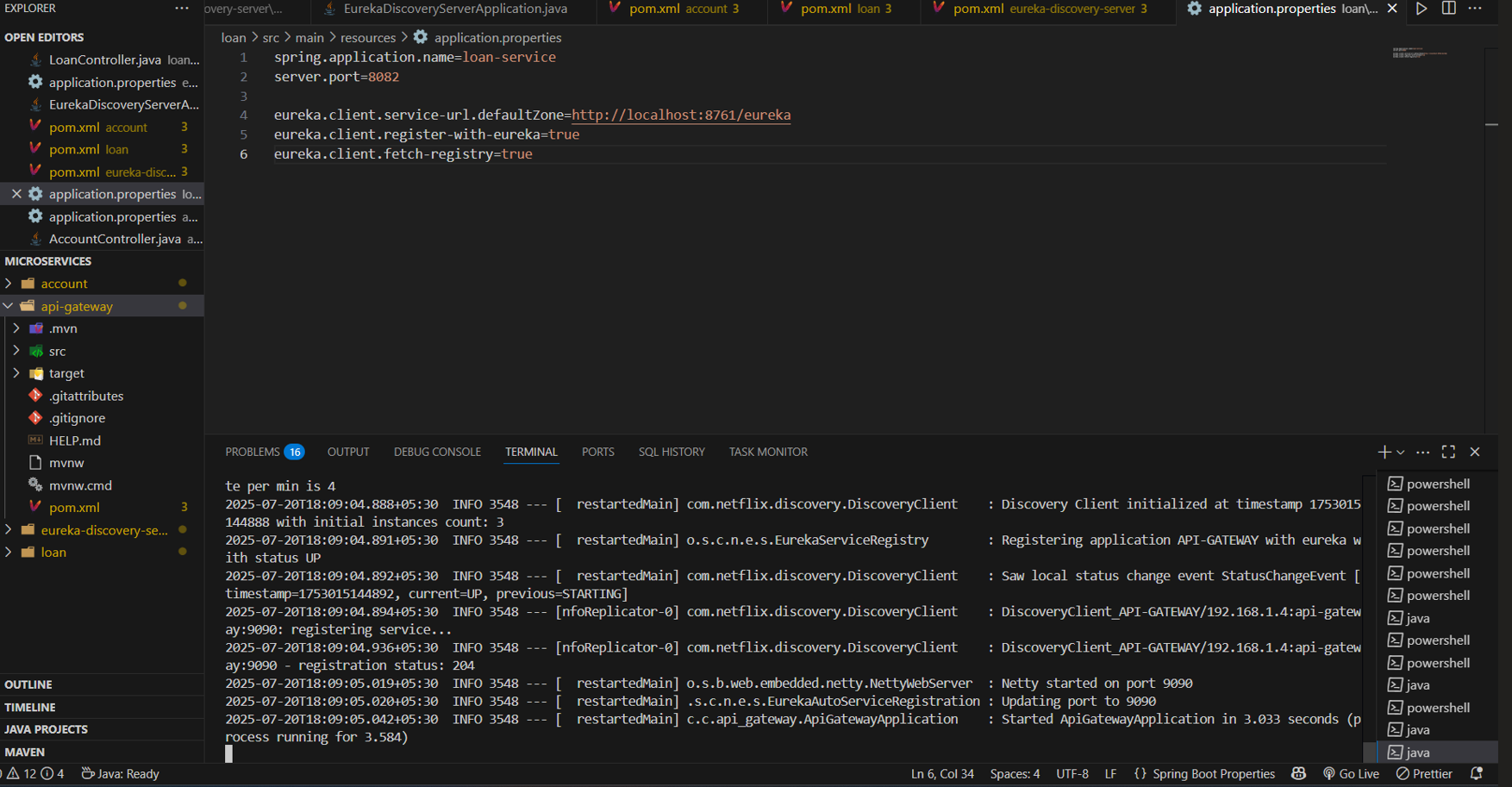
.and()

.addFilterBefore(jwtTokenFilter, UsernamePasswordAuthenticationFilter.class);

}

}

OUTPUT:



**Exercise 1: User and Order Management System**

UserService.java (REST Controller)

java

CopyEdit

@RestController

@RequestMapping("/users")

@RequiredArgsConstructor

public class UserController {

private final UserService userService;

@PostMapping

public ResponseEntity<User> createUser(@RequestBody User user) {

return ResponseEntity.ok(userService.createUser(user));

}

@GetMapping("/{id}")

public ResponseEntity<User> getUser(@PathVariable Long id) {

return ResponseEntity.ok(userService.getUserById(id));

}

}

OrderService.java (Calling User Service via WebClient)

java

CopyEdit

@RestController

@RequestMapping("/orders")

@RequiredArgsConstructor

public class OrderController {

private final WebClient.Builder webClientBuilder;

private final OrderService orderService;

@PostMapping

public ResponseEntity<Order> createOrder(@RequestBody Order order) {

// Optionally call user-service to validate user

User user = webClientBuilder.build()

.get()

.uri("http://USER-SERVICE/users/" + order.getUserId())

.retrieve()

.bodyToMono(User.class)

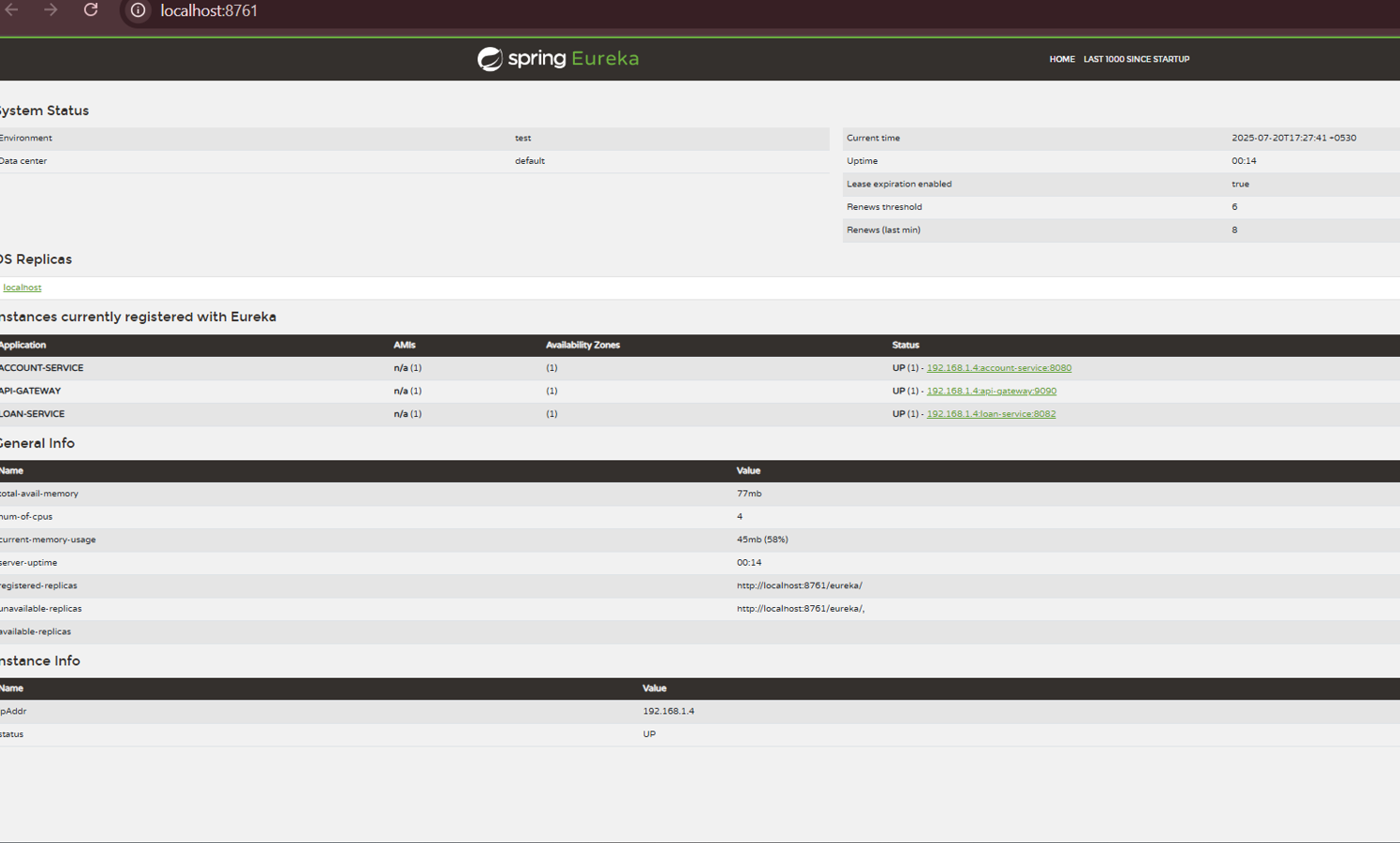
.block();

return ResponseEntity.ok(orderService.createOrder(order));

}

}

OUTPUT:



**Exercise 2: Inventory Management with Eureka + Config Server**

**pom.xml Dependencies**

xml

CopyEdit

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

**application.yml**

yaml

CopyEdit

spring:

application:

name: product-service

config:

import: optional:configserver:http://localhost:8888

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

**ProductController.java**

java

CopyEdit

@RestController

@RequestMapping("/products")

@RequiredArgsConstructor

public class ProductController {

private final ProductService productService;

@GetMapping("/{id}")

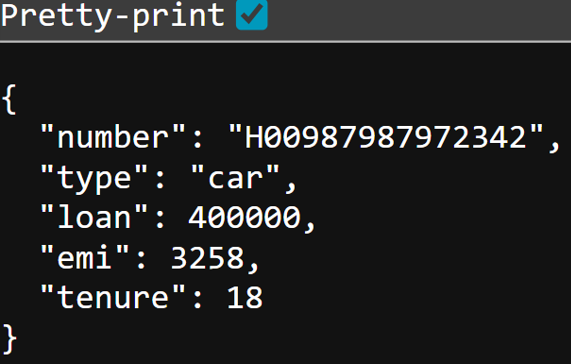
public ResponseEntity<Product> getProduct(@PathVariable Long id) {

return ResponseEntity.ok(productService.getProductById(id));

}

}

OUTPUT:



**Exercise 3: API Gateway**

**pom.xml Dependencies**

xml

CopyEdit

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

**application.yml**

yaml

CopyEdit

spring:

cloud:

gateway:

routes:

- id: customer-service

uri: lb://CUSTOMER-SERVICE

predicates:

- Path=/customers/\*\*

filters:

- RewritePath=/customers/(?<segment>.\*), /${segment}

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 10

redis-rate-limiter.burstCapacity: 20

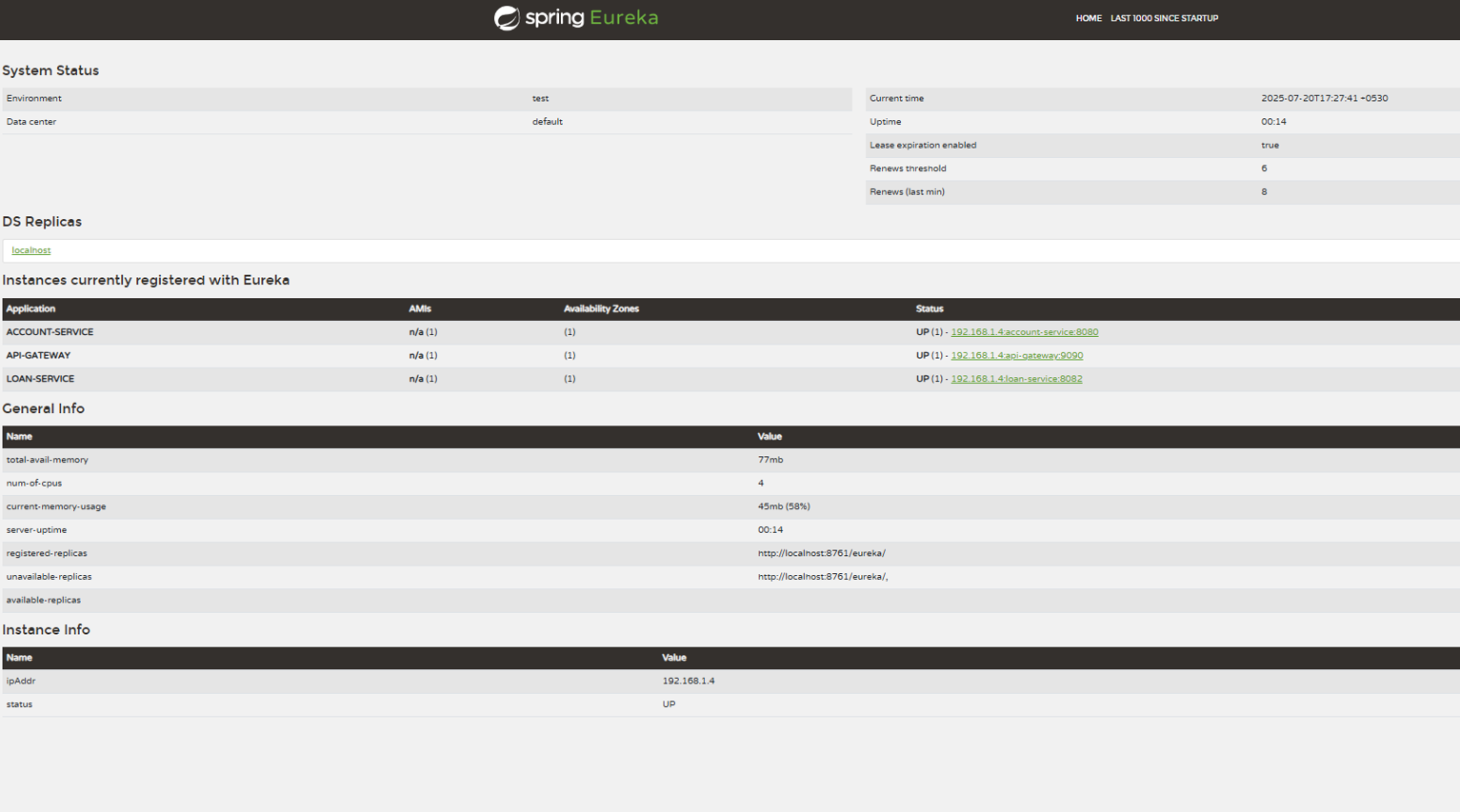
- id: billing-service

uri: lb://BILLING-SERVICE

predicates:

- Path=/billing/

OUTPUT:



**Exercise 4: Resilient Microservices with Circuit Breaker**

**pom.xml**

xml

CopyEdit

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot3</artifactId>

</dependency>

**application.yml**

yaml

CopyEdit

resilience4j:

circuitbreaker:

instances:

paymentService:

slidingWindowSize: 5

failureRateThreshold: 50

waitDurationInOpenState: 10000

**PaymentService.java**

java

CopyEdit

@Service

@RequiredArgsConstructor

public class PaymentService {

private final WebClient.Builder webClientBuilder;

@CircuitBreaker(name = "paymentService", fallbackMethod = "fallbackPayment")

public String callThirdPartyAPI() {

return webClientBuilder.build()

.get()

.uri("http://third-party-service/api/payment")

.retrieve()

.bodyToMono(String.class)

.block();

}

public String fallbackPayment(Throwable t) {

return "Payment service is currently unavailable. Please try again later.";

}

}

OUTPUT:

