Sample Hands-on Exercises on Edge Services and API Gateway with Spring Boot 3 and Spring Cloud

Exercise 1: Implementing Edge Services for Routing and Filtering

Task: Implement an edge service for routing and filtering requests in a microservices architecture using Spring Boot 3 and Spring Cloud.

Step-by-Step Explanation:

1. Create a new Spring Boot project with the necessary dependencies for Spring Cloud Gateway.

2. Configure the application properties for routing.

3. Implement a custom filter for logging requests.

4. Test the routing and filtering functionality.

Solution Code:

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:

```
@Component
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);
  }
}
```

Exercise 2: Load Balancing in an API Gateway

Task: Implement load balancing in an API Gateway using Spring Boot 3 and Spring Cloud.

Step-by-Step Explanation:

1. Create a new Spring Boot project with the necessary dependencies for Spring Cloud Gateway and Spring Cloud LoadBalancer.

2. Configure the application properties for load balancing.

3. Implement a custom load balancing configuration.

4. Test the load balancing functionality.

Solution Code:

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/**
```

Load Balancer Configuration:

```
@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);
  }
}
```

Exercise 3: Resilience Patterns in an API Gateway

Task: Implement resilience patterns in an API Gateway using Spring Boot 3 and Spring Cloud.

Step-by-Step Explanation:

1. Create a new Spring Boot project with the necessary dependencies for Spring Cloud Gateway and Resilience4j.

2. Configure the application properties for resilience patterns.

3. Implement a custom resilience configuration.

4. Test the resilience functionality.

Solution Code:

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
```

Resilience Configuration:

```
@Configuration
public class ResilienceConfiguration {
  @Bean
  public Customizer<ReactiveResilience4JCircuitBreakerFactory> defaultCustomizer() {
    return factory -> factory.configureDefault(id -> new Resilience4JConfigBuilder(id)
      .circuitBreakerConfig(CircuitBreakerConfig.ofDefaults())
      .timeLimiterConfig(TimeLimiterConfig.ofDefaults())
      .build());
  }
}
```