**Exercises on Microservices with Spring Boot 3.0**

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

### 1. ****User Service****

#### User.java

@Entitypublic class User {

@Id @GeneratedValue

private Long id;

private String name;

private String email;

}

#### UserRepository.java

public interface UserRepository extends JpaRepository<User, Long> {}

#### UserService.java

@Servicepublic class UserService {

@Autowired

private UserRepository repo;

public User createUser(User user) {

return repo.save(user);

}

public User getUser(Long id) {

return repo.findById(id).orElseThrow();

}

}

#### UserController.java

@RestController@RequestMapping("/users")public class UserController {

@Autowired

private UserService service;

@PostMapping

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

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

}

@GetMapping("/{id}")

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

return ResponseEntity.ok(service.getUser(id));

}

}

#### application.properties

spring.datasource.url=jdbc:mysql://localhost:3306/userdb

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.jpa.hibernate.ddl-auto=update

2. **Order Service**

#### Order.java

@Entitypublic class Order {

@Id @GeneratedValue

private Long id;

private Long userId;

private String product;

private int quantity;

}

#### OrderRepository.java

public interface OrderRepository extends JpaRepository<Order, Long> {}

#### OrderService.java

java

CopyEdit

@Servicepublic class OrderService {

@Autowired

private OrderRepository repo;

@Autowired

private UserClient userClient;

public Order placeOrder(Order order) {

// Validate user exists

userClient.getUser(order.getUserId());

return repo.save(order);

}

}

Communication (Feign Client Example)

#### UserClient.java

@FeignClient(name = "user-service", url = "http://localhost:8081")public interface UserClient {

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

User getUser(@PathVariable Long id);

}

#### OrderController.java

@RestController@RequestMapping("/orders")public class OrderController {

@Autowired

private OrderService service;

@PostMapping

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

return ResponseEntity.ok(service.placeOrder(order));

}

}

application.properties

spring.datasource.url=jdbc:mysql://localhost:3306/orderdb

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.jpa.hibernate.ddl-auto=update

# Feign

feign.client.config.default.connect-timeout=5000

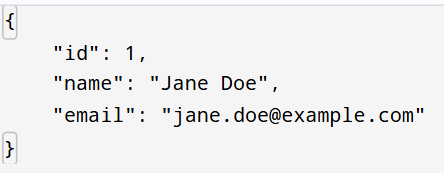
feign.client.config.default.read-timeout=5000

**OUTPUT:**

Output for Creating a New User:

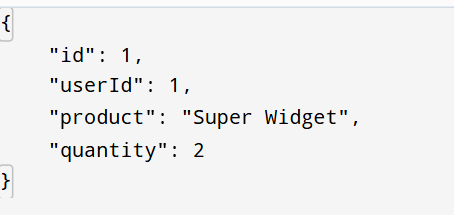


Output for Getting a User by ID:

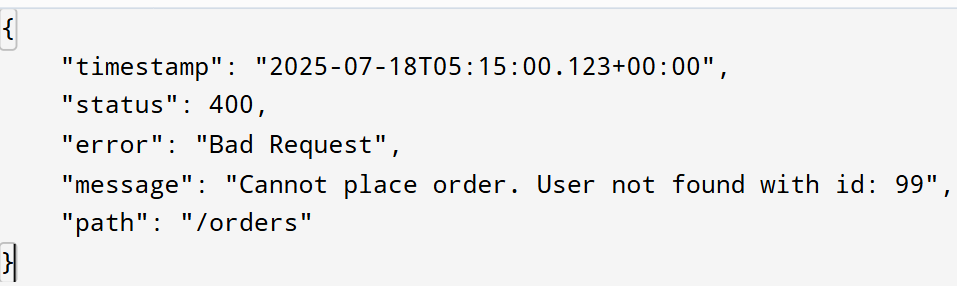


### **Order Service Output**

**1. Output for Placing a Successful Order:**



**2. Output for Placing a Failed Order (User Not Found):**



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

### Main Application

@EnableConfigServer@SpringBootApplicationpublic class ConfigServerApplication {

public static void main(String[] args) {

SpringApplication.run(ConfigServerApplication.class, args);

}

}

application.yml

server:

port: 8888

spring:

cloud:

config:

server:

git:

uri: https://github.com/your-git-repo/config-repo

Create a Git repo config-repo with files like product-service.yml, inventory-service.yml.

2. EUREKA SERVER

### Setup (Standalone Project)

**Dependencies**:

Spring Cloud Netflix Eureka Server

Spring Web

### Main Application

@EnableEurekaServer@SpringBootApplicationpublic class EurekaServerApplication {

public static void main(String[] args) {

SpringApplication.run(EurekaServerApplication.class, args);

}

}

### application.yml

server:

port: 8761

eureka:

client:

register-with-eureka: false

fetch-registry: false

Access Eureka at http://localhost:8761

3. PRODUCT SERVICE

### Dependencies:

Spring Web

Spring Data JPA

Eureka Client

Config Client

Lombok

MySQL or H2

Spring Boot Actuator

application.yml (in Git config repo)

product-service.yml

spring:

application:

name: product-service

datasource:

url: jdbc:mysql://localhost:3306/productdb

username: root

password: root

jpa:

hibernate:

ddl-auto: update

show-sql: true

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

Product Entity

java

CopyEdit

@Entity@Datapublic class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String description;

}

Repository & Service

java

CopyEdit

public interface ProductRepository extends JpaRepository<Product, Long> {}

@Servicepublic class ProductService {

@Autowired

private ProductRepository productRepository;

public Product saveProduct(Product product) {

return productRepository.save(product);

}

public List<Product> getAll() {

return productRepository.findAll();

}

}

Controller

@RestController@RequestMapping("/products")public class ProductController {

@Autowired

private ProductService productService;

@PostMapping

public Product save(@RequestBody Product product) {

return productService.saveProduct(product);

}

@GetMapping

public List<Product> getAll() {

return productService.getAll();

}

}

4. INVENTORY SERVICE

### application.yml (in Git config repo)

inventory-service.yml

spring:

application:

name: inventory-service

datasource:

url: jdbc:mysql://localhost:3306/inventorydb

username: root

password: root

jpa:

hibernate:

ddl-auto: update

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka

Stock Entity

java

CopyEdit

@Entity@Datapublic class Stock {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private Long productId;

private Integer quantity;

}

Repository & Service

public interface StockRepository extends JpaRepository<Stock, Long> {}

@Servicepublic class StockService {

@Autowired

private StockRepository stockRepository;

public Stock updateStock(Stock stock) {

return stockRepository.save(stock);

}

public Stock getByProductId(Long productId) {

return stockRepository.findAll()

.stream()

.filter(s -> s.getProductId().equals(productId))

.findFirst().orElse(null);

}

}

Controller

java

CopyEdit

@RestController@RequestMapping("/stocks")public class StockController {

@Autowired

private StockService stockService;

@PostMapping

public Stock update(@RequestBody Stock stock) {

return stockService.updateStock(stock);

}

@GetMapping("/{productId}")

public Stock getByProduct(@PathVariable Long productId) {

return stockService.getByProductId(productId);

}

}

**OUTPUT:**

**3. Implement an API Gateway**

## 1. CUSTOMER SERVICE

### Sample Controller

@RestController@RequestMapping("/customer")public class CustomerController {

@GetMapping("/info")

public String getCustomerInfo() {

return "Customer Information";

}

}

2. BILLING SERVICE

### Sample Controller

@RestController@RequestMapping("/billing")public class BillingController {

@GetMapping("/info")

public String getBillingInfo() {

return "Billing Information";

}

}

3. API GATEWAY (Spring Cloud Gateway)

### Dependencies (pom.xml)

<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>

<dependency>

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

<artifactId>spring-boot-starter-data-redis</artifactId></dependency>

<dependency>

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

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

application.yml (Gateway Configuration)

server:

port: 8080

spring:

redis:

host: localhost

port: 6379

cloud:

gateway:

routes:

- id: customer-service

uri: http://localhost:8081

predicates:

- Path=/api/customer/\*\*

filters:

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

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 5

redis-rate-limiter.burstCapacity: 10

- id: billing-service

uri: http://localhost:8082

predicates:

- Path=/api/billing/\*\*

filters:

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

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 3

redis-rate-limiter.burstCapacity: 5

default-filters:

- name: RequestRateLimiter

args:

redis-rate-limiter.replenishRate: 10

redis-rate-limiter.burstCapacity: 20

PATH REWRITING

**Example:**  
GET /api/customer/info → rewrites and forwards to → http://localhost:8081/customer/info

RATE LIMITING

Redis-backed token bucket algorithm using Spring Gateway’s built-in RequestRateLimiter

Limits:

Customer Service: 5 requests/sec (burst up to 10)

Billing Service: 3 requests/sec (burst up to 5)

CACHING (OPTIONAL)

You can use Redis cache inside Customer or Billing services:

@Servicepublic class CustomerService {

@Autowired

private RedisTemplate<String, String> redisTemplate;

public String getCustomerInfo() {

String cacheKey = "customerInfo";

String cached = redisTemplate.opsForValue().get(cacheKey);

if (cached != null) return cached;

String response = "Customer Information";

redisTemplate.opsForValue().set(cacheKey, response, Duration.ofMinutes(10));

return response;

}

}

Or use Spring Cache abstraction + @Cacheable annotation.

**OUTPUT:**

**4. Resilient Microservices with Circuit Breaker**

### Add Dependencies (pom.xml)

<dependency>

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

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

<dependency>

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

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

<dependency>

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

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

<dependency>

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

<artifactId>spring-boot-starter-aop</artifactId></dependency>

application.yml

server:

port: 8080

resilience4j:

circuitbreaker:

instances:

thirdPartyApi:

registerHealthIndicator: true

slidingWindowSize: 5

failureRateThreshold: 50

waitDurationInOpenState: 10s

permittedNumberOfCallsInHalfOpenState: 2

minimumNumberOfCalls: 5

management:

endpoints:

web:

exposure:

include: '\*'

2. IMPLEMENTATION

### Simulate a Slow Third-party API Call

@Servicepublic class ThirdPartyService {

public String callSlowApi() {

try {

// Simulating delay

Thread.sleep(4000); // 4 seconds

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

return "Third-party API Response";

}

}

Payment Service with Circuit Breaker

@Servicepublic class PaymentService {

@Autowired

private ThirdPartyService thirdPartyService;

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

public String makePayment() {

return thirdPartyService.callSlowApi();

}

public String fallbackPayment(Throwable t) {

System.out.println("⚠️ Fallback triggered: " + t.getMessage());

return "Fallback: Payment service is temporarily unavailable.";

}

}

Payment Controller

@RestController@RequestMapping("/payment")public class PaymentController {

@Autowired

private PaymentService paymentService;

@GetMapping("/process")

public String processPayment() {

return paymentService.makePayment();

}

}

**OUTPUT:**