Exercise 1: User and Order Management System

Technologies:

Spring Boot 3.0, Spring Data JPA, OpenFeign/WebClient, MySQL or PostgreSQL

user-service

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
</dependency>
```

User.java

```
@Entity
public class User {
    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private String email;
}
```

UserRepository.java

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

UserController.java

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

```java
    private UserRepository repo;

    @PostMapping
    public User createUser(@RequestBody User user) {
        return repo.save(user);
    }

    @GetMapping("/{id}")
    public User getUser(@PathVariable Long id) {
        return repo.findById(id).orElseThrow();
    }
}
```

order-service

pom.xml

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-openfeign</artifactId>
</dependency>
```

Order.java

```java
@Entity
public class Order {
    @Id
    @GeneratedValue
    private Long id;
    private Long userId;
    private String product;
}
```

OrderRepository.java

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

UserClient.java

```java
@FeignClient(name = "user-service", url = "http://localhost:8081")
```

```
public interface UserClient {
    @GetMapping("/users/{id}")
    User getUserById(@PathVariable Long id);
}
```

OrderController.java

```
@RestController
@RequestMapping("/orders")
public class OrderController {
    @Autowired
    private OrderRepository repo;
    @Autowired
    private UserClient userClient;

    @PostMapping
    public Order createOrder(@RequestBody Order order) {
        userClient.getUserById(order.getUserId());
        return repo.save(order);
    }
}
```

Exercise 2: Inventory Management System with Service Discovery

Technologies:

Spring Boot 3.0, Eureka Discovery, Spring Cloud Config, MySQL

eureka-server

pom.xml

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

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

EurekaServerApplication.java

```java
@EnableEurekaServer
@SpringBootApplication
public class EurekaServerApplication {
    public static void main(String[] args) {
        SpringApplication.run(EurekaServerApplication.class, args);
    }
}
```

application.yml

```yaml
server.port: 8761
eureka:
  client:
    register-with-eureka: false
    fetch-registry: false
```

config-server

pom.xml

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-config-server</artifactId>
</dependency>
```

ConfigServerApplication.java

```java
@EnableConfigServer
@SpringBootApplication
public class ConfigServerApplication {}
```

application.yml

```yaml
server.port: 8888
spring:
  cloud:
    config:
      server:
        git:
          uri: https://github.com/your-config-repo
```

product-service

Product.java

```
@Entity
public class Product {
    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private int stock;
}
```

ProductRepository.java

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

application.yml

```
spring:
  application:
    name: product-service
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka
```

inventory-service

ProductClient.java

```java
@FeignClient("product-service")
public interface ProductClient {
    @GetMapping("/products/{id}")
    Product getProduct(@PathVariable Long id);
}
```

InventoryController.java

```java
@RestController
@RequestMapping("/inventory")
public class InventoryController {
    @Autowired
    private ProductClient productClient;

    @GetMapping("/check/{productId}")
    public boolean checkStock(@PathVariable Long productId) {
        Product p = productClient.getProduct(productId);
        return p.getStock() > 0;
    }
}
```

Exercise 3: API Gateway

Technologies:

Spring Boot 3.0, Spring Cloud Gateway, Redis (optional)

gateway-service

pom.xml

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
```

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

application.yml

```
spring:
  cloud:
    gateway:
      routes:
        - id: customer-service
          uri: lb://CUSTOMER-SERVICE
          predicates:
            - Path=/customer/**
          filters:
            - RewritePath=/customer/(?<path>.*), /${path}
            - RequestRateLimiter=redis-rate-limiter[replenishRate=5,
burstCapacity=10]
        - id: billing-service
          uri: lb://BILLING-SERVICE
          predicates:
            - Path=/billing/**
          filters:
            - RewritePath=/billing/(?<path>.*), /${path}
```

Exercise 4: Resilient Microservices with Circuit Breaker

Technologies:

Spring Boot 3.0, Resilience4j, Spring Boot Actuator

payment-service

pom.xml

```
<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>
```

PaymentService.java

```
@Service
public class PaymentService {

    @CircuitBreaker(name = "paymentAPI", fallbackMethod =
"fallback")
    public String callPaymentAPI() {
        return new
RestTemplate().getForObject("http://slow-api.com/pay",
String.class);
    }

    public String fallback(Throwable t) {
        return "Payment service down. Please try later.";
    }
}
```

application.yml

```
resilience4j:
  circuitbreaker:
    instances:
      paymentAPI:
        registerHealthIndicator: true
        failureRateThreshold: 50
        waitDurationInOpenState: 10s
        slidingWindowSize: 10

management:
  endpoints:
    web:
      exposure:
        include: "*"
```