Solutions to Microservices

# 1. User and Order Management System

Create two microservices:  
  
- User Service: Exposes REST APIs for creating and retrieving users. Stores user details in a MySQL/PostgreSQL database.  
- Order Service: Exposes REST APIs for placing and fetching orders. It communicates with User Service using WebClient or OpenFeign.  
  
Technologies: Spring Boot 3, Spring Data JPA, Spring WebFlux or OpenFeign, MySQL/PostgreSQL  
  
Example:   
- User Service URL: `/users/{id}`  
- Order Service calls `/users/{id}` using WebClient or Feign to verify the user before placing an order.

# 2. Inventory Management System with Service Discovery

Create two services:  
  
- Product Service: Manages product information.  
- Inventory Service: Manages stock levels and queries the Product Service.  
  
Use Spring Cloud Netflix Eureka for service registration and discovery.  
- Add `@EnableEurekaClient` to both services.  
- Use Eureka server for discovery.  
- Use Spring Cloud Config Server for centralized configuration.  
  
Example properties for discovery:  
  
eureka.client.service-url.defaultZone=http://localhost:8761/eureka  
spring.application.name=product-service

# 3. Implement an API Gateway

Create a Spring Cloud Gateway application.  
  
Routes requests to:  
- Customer Service: `/customer/`  
- Billing Service: `/billing/`  
  
Features to implement:  
- Rate Limiting using RedisRateLimiter  
- Caching using a cache layer like Caffeine or external Redis  
- Path Rewriting via gateway filters  
  
Example application.yml snippet:  
spring.cloud.gateway.routes:  
 - id: customer-service  
 uri: lb://CUSTOMER-SERVICE  
 predicates:  
 - Path=/customer/\*\*  
 filters:  
 - RewritePath=/customer/(?<segment>.\*), /$\{segment}  
 - RequestRateLimiter

# 4. Resilient Microservices with Circuit Breaker

Use Resilience4j for implementing Circuit Breaker for Payment Service.  
Steps:  
- Add dependency: `resilience4j-spring-boot3`  
- Use `@CircuitBreaker` annotation on the method calling third-party API.  
- Provide fallback method to return dummy/default response.  
  
Example:  
@CircuitBreaker(name = "paymentService", fallbackMethod = "fallbackPayment")  
public String callPaymentAPI() {  
 return webClient.get().uri("http://slow-api").retrieve().bodyToMono(String.class).block();  
}  
  
public String fallbackPayment(Throwable t) {  
 log.warn("Fallback triggered", t);  
 return "Payment service is currently unavailable";  
}  
Also configure monitoring/logging using Micrometer and Resilience4j dashboard.