# E-Commerce Microservices Architecture Design Document

## 1. Overview

This document outlines the design and architecture for an e-commerce platform built using microservices. The system supports modularity, scalability, maintainability, and asynchronous processing. It includes services for user management, product catalog, order processing, and card validation.

## 2. Technology Stack

* **Language:** Java
* **Framework:** Spring Boot
* **Build Tool:** Maven
* **Database:** H2 (Dev), MySQL/PostgreSQL (Prod)
* **Inter-Service Communication:** REST (via RestTemplate/FeignClient)
* **Logging:** SLF4J with Logback
* **Testing:** JUnit & Mockito
* **Containerization:** Docker
* **Deployment:** Kubernetes-ready YAMLs

## 3. Microservices

### 3.1 User Service

Handles user registration, login, and profile management. - CRUD operations on users - Data validation - Connected to its own user table

### 3.2 Product Catalog Service

Manages product listing and availability. - Fetch product info (GET) - Maintain inventory (future scope) - Connected to its own product table

### 3.3 Order Service (with Multithreading)

Processes orders and asynchronously verifies payment status. - Receives order requests - Starts main thread for order handling - Spawns a child thread to check payment and notify users - Logs structured events

### 3.4 Card Validation Service

Validates payment card numbers. - Simple REST GET API - Uses regex to validate 16-digit card number - Provides response: Valid or Invalid

## 4. Architecture

* Services are stateless and independent
* RESTful APIs for communication
* Each service may have its own database (Database per Service)
* Common libraries (e.g., logging) shared across services

## 5. Exception Handling

* @ControllerAdvice based global handler
* Handles general and specific exceptions (like IllegalArgumentException)
* Returns meaningful HTTP status codes

## 6. Logging

* Uses SLF4J with Logback
* Logs are structured with service context and operation metadata
* Captures success/failure of threads, input validation, etc.

## 7. Testing Strategy

* JUnit for unit tests
* Mockito for mocking services
* Coverage goal: 80%+
* Tests written for controllers, services, and DAOs

## 8. Multithreading Logic (Order Service)

* Main thread: Processes the order
* Child thread: Validates payment and sends notifications
* Thread.join() ensures proper lifecycle handling
* Exceptions handled gracefully within threads

## 9. Deployment

* Dockerfiles for each service
* Kubernetes deployment YAMLs per service
* Environment configs via application.properties

## 10. Future Enhancements

* Implement distributed tracing (e.g., with Zipkin)
* Circuit breakers with Resilience4j
* API Gateway and service discovery with Spring Cloud Netflix

Prepared by: *ChatGPT — July 2025*