# Project Proposal: Transition to Microservices for E-Commerce Platform

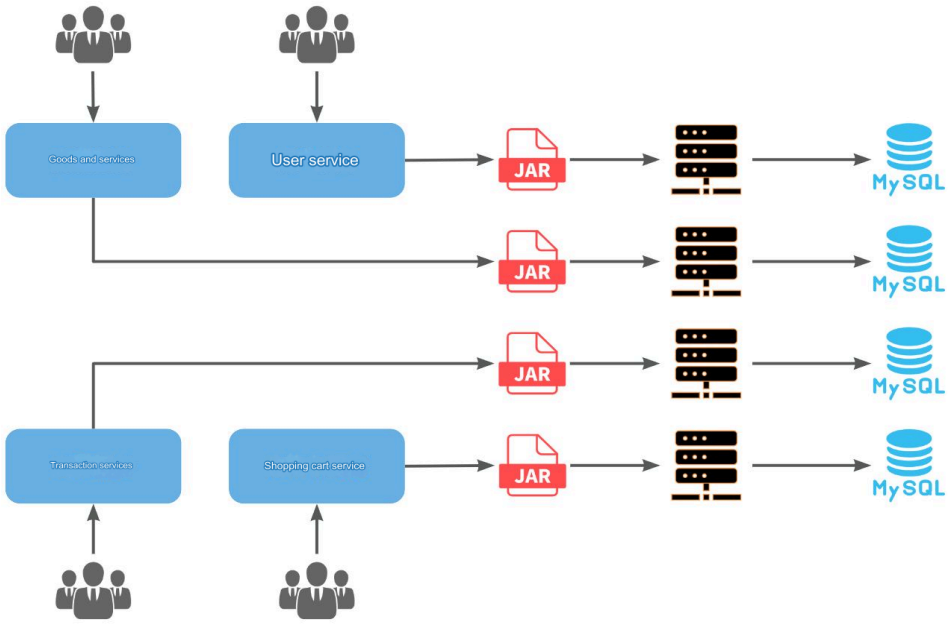
## Executive Summary

This proposal outlines the strategic plan to transition our current monolithic e-commerce platform into a more scalable, resilient, and maintainable microservices architecture. By leveraging Spring Cloud, Nacos, and OpenFeign, we aim to improve our platform's scalability, fault tolerance, and deployment speeds, ensuring our ability to meet the evolving needs of our customers and business.

## Application Functionality Overview

Our e-commerce platform facilitates user registration and management, product browsing and search, order placement and management, and secure payment processing. The platform is designed to provide a seamless and efficient shopping experience for users, alongside an easy-to-use management interface for administrators and sellers.

## High-Level System Architecture and Component Design




### Architecture Overview

- Frontend: React-based SPA (Single Page Application) for both customer-facing and admin interfaces.
- API Gateway: Nginx or Spring Cloud Gateway routing external requests to appropriate microservices.
- Microservices:
    - User Service: Handles user registration, authentication, and profile management.
    - Product Service: Manages product catalog, including product listings, descriptions, and inventory.
    - Order Service: Manages the lifecycle of customer orders from placement through fulfillment.
    - Payment Service: Handles payment processing, including integration with external payment gateways.
- Service Discovery and Configuration: Nacos for dynamic service discovery and centralized configuration management.
- Database (MySQL): Separate databases for each service to ensure loose coupling and autonomy.

### Component Interaction

- Services register with and query Nacos for discovery.
- The API Gateway routes incoming requests to the appropriate services.
- Services communicate with each other using OpenFeign for synchronous HTTP calls.

## Distributed Computing Algorithms and Techniques

- Service Discovery: Leveraging Nacos for dynamic discovery and load balancing.
- Stateless Design: Ensuring scalability and resilience by designing stateless services wherever possible.
- Database Sharding and Replication: Implementing database sharding for scalability and replication for availability.
- Circuit Breaker: Using Spring Cloud Circuit Breaker to prevent failures in one service from cascading.

## Test Strategy

- Unit Testing: To validate individual components for correctness.
- Integration Testing: To ensure that microservices interact with each other and external components correctly.
- Performance Testing: To assess the system's behavior under load using tools like JMeter.
- End-to-End Testing: To validate the entire application flow from the user interface through to the backend.
- Security Testing: To identify and mitigate vulnerabilities.

# Delivery Plan

## Core Capabilities

- Phase 1: Setup of basic microservices (User and Product Services) and integration with Nacos for service discovery.
- Phase 2: Addition of Order and Payment Services, introduction of API Gateway, and initial deployment to a staging environment.
- Phase 3: Optimization for performance, scalability, and security. Full integration testing and preparation for production deployment.

## Incremental Enhancements

- Phase 4: Incorporate feedback, refine service interfaces, and expand product and order management features.
- Phase 5: Further optimization and scaling. Introduce advanced features like personalized recommendations and dynamic pricing.

# Project Split

## System Design

- Define microservices boundaries and responsibilities.
- Design data models and APIs for each service.
- Architect the deployment and runtime environment.

## Engineering Activities

- Implement microservices using Spring Boot and integrate with Spring Cloud components.
- Develop frontend applications and integrate with backend services.
- Setup CI/CD pipelines for automated testing and deployment.

## Test Activities

- Develop and execute unit and integration tests.
- Conduct performance and security assessments.
- Organize user acceptance testing with stakeholders.