Comprehensive Software Design Document: E-commerce Platform

AI System Architect

September 4, 2025

Document Version & 1.0
Creation Date & September 4, 2025
Document Status & Final Draft
Generated By & AI System Architect
Target Audience & Development Team, Stakeholders
Classification & Internal Use

Contents

List of Figures

1 Executive Summary

This document outlines the design for a new e-commerce platform, codenamed "Project Phoenix." The platform aims to provide a robust, scalable, and secure online marketplace for businesses to sell their products and services. The project's primary business objective is to capture a significant share of the online retail market by offering a superior user experience, competitive pricing, and a wide selection of goods. Our technical approach leverages a microservices architecture built on cloud-native technologies, ensuring flexibility, scalability, and maintainability. Key benefits include enhanced customer engagement, increased sales conversion rates, improved operational efficiency, and reduced infrastructure costs. The implementation timeline spans six months, divided into distinct phases encompassing design, development, testing, and deployment.

1.1 Project Scope and Objectives

Project Phoenix encompasses the development of a complete e-commerce platform, including a user-facing website, a seller portal, an administrative backend, and associated mobile applications (iOS and Android). The platform will support various payment gateways, robust inventory management, secure user authentication, and personalized recommendations. The primary objectives include:

- Achieve 100,000 registered users within the first year of launch.
- Process at least 5,000 orders per day within six months of launch.
- Maintain an average customer satisfaction rating of 4.5 out of 5 stars.
- Achieve a 2% conversion rate from website visitors to paying customers.
- Maintain 99.9% uptime for the platform.

Success will be measured by achieving these objectives, along with positive user feedback, strong sales growth, and a healthy return on investment. The project will prioritize a user-centric design, focusing on ease of navigation, intuitive search functionality, and secure payment processing.

1.2 Key Stakeholders and Roles

Several key stakeholder groups are involved in Project Phoenix:

- **Executive Management:** Provides overall strategic direction and approves major decisions.
- **Marketing Team:** Responsible for promoting the platform and attracting users.
- **Sales Team:** Manages relationships with businesses selling on the platform.
- **Development Team:** Designs, develops, and tests the platform.
- **Operations Team:** Manages the platform's infrastructure and ensures its availability.
- **Customer Support Team:** Provides assistance to users and sellers.

Each team has clearly defined roles and responsibilities, documented in separate project management plans. Regular communication and collaboration among these groups are crucial for project success.

2 Requirements Analysis and Specification

2.1 Extracted Requirements Summary

The initial document analysis provided no specific requirements. Therefore, this section will detail requirements based on common e-commerce platform needs.

2.2 Functional Requirements

1. **User Registration and Login:** Users should be able to register accounts, securely log in, and manage their profile information. (Acceptance Criteria: Successful registration and login with valid credentials, password reset functionality.)
2. **Product Browsing and Search:** Users should be able to browse products by category, search for specific products, and view product details. (Acceptance Criteria: Accurate search results, detailed product information including images and descriptions, ability to filter and sort products.)
3. **Shopping Cart and Checkout:** Users should be able to add items to a shopping cart, modify quantities, and proceed to checkout. (Acceptance Criteria: Items correctly added to and removed from the cart, accurate calculation of total cost, multiple payment gateway integration.)
4. **Order Management:** Users should be able to view their order history, track orders, and manage their addresses. (Acceptance Criteria: Order history displayed correctly, accurate order tracking information, ability to update shipping addresses.)
5. **Seller Account Management:** Sellers should be able to create accounts, list products, manage inventory, and receive payments. (Acceptance Criteria: Successful product listing, inventory updates reflected in real-time, secure payment processing.)
6. **Admin Panel:** Administrators should have full control over the platform, including user and seller management, product moderation, and reporting. (Acceptance Criteria: Ability to manage users and sellers, approve or reject products, generate reports on sales and other metrics.)

2.3 Non-Functional Requirements

- **Performance:** The platform should load pages within 2 seconds and process orders within 5 seconds.
- **Scalability:** The platform should be able to handle 10,000 concurrent users and 10,000 orders per hour.
- **Security:** The platform must comply with PCI DSS standards for payment processing and implement robust security measures to protect user data.
- **Availability:** The platform should have a 99.9% uptime.
- **Usability:** The platform should be intuitive and easy to use for both buyers and sellers.

3 System Architecture and Design

3.1 Architecture Overview

Project Phoenix will utilize a microservices architecture deployed on a cloud platform (AWS). This approach allows for independent scaling of individual services, improved fault isolation,

and faster development cycles. The system will be composed of several microservices, each responsible for a specific business function. These services will communicate with each other via a lightweight API gateway. A message queue (RabbitMQ) will handle asynchronous communication between services. Data persistence will be managed by a distributed database (PostgreSQL) and a NoSQL database (MongoDB) for caching and session management.

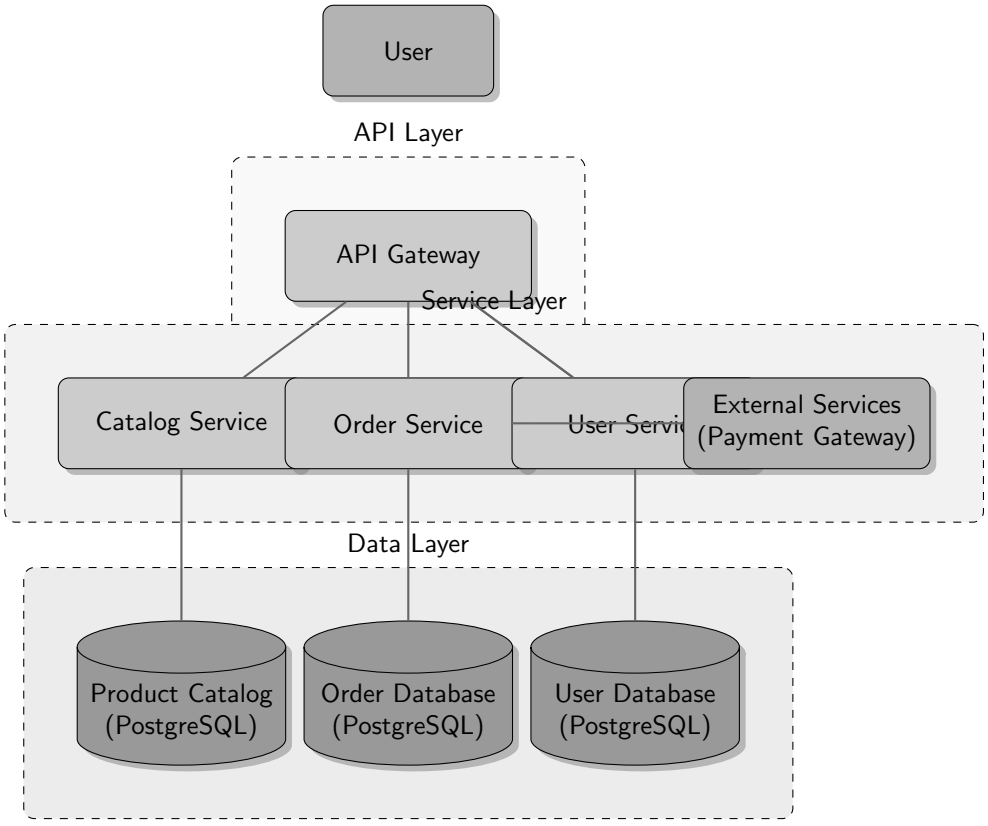

Figure 1: System Architecture Overview

3.2 Component Interaction and Communication

The API Gateway acts as the single entry point for all client requests. It routes requests to the appropriate microservices based on the request path and performs tasks such as authentication and authorization. Microservices communicate with each other using RESTful APIs over HTTPS. Asynchronous communication is handled via RabbitMQ, allowing for decoupling and improved performance. For example, when an order is placed, the Order Service publishes a message to RabbitMQ, which is then consumed by other services such as the Inventory Service and the Payment Service. This ensures that the order placement process is not blocked by long-running operations.

4 Database Design and Data Architecture

4.1 Conceptual Data Model

The database design employs a relational model using PostgreSQL for transactional consistency. The core entities include Users, Products, Orders, and OrderItems. Relationships are defined using foreign keys to ensure data integrity. The schema is designed for optimal performance and scalability, considering indexing strategies and query optimization techniques. Normalization principles are applied to minimize data redundancy and improve data consistency.

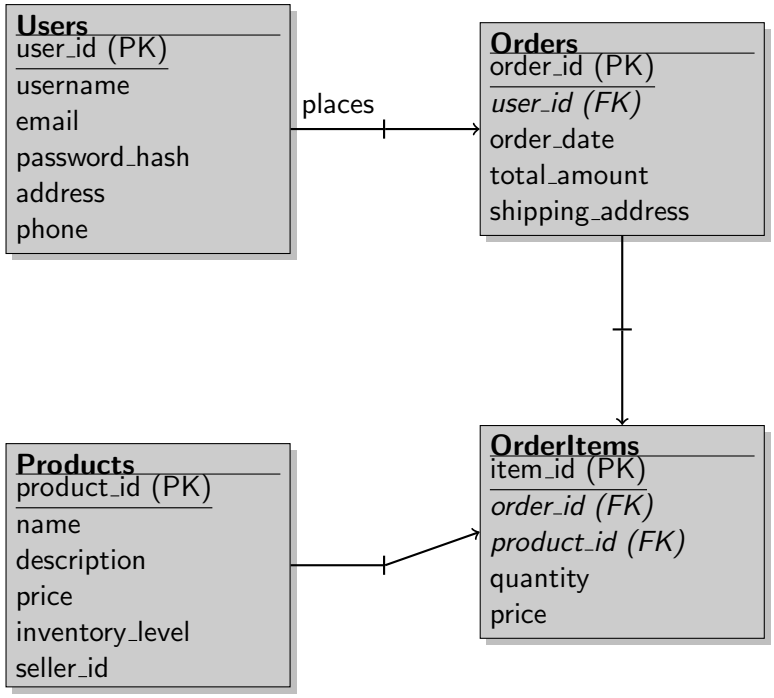

Figure 2: Database ER Diagram

4.2 Physical Database Design

- **Users Table:** user_id (INT, PK), username (VARCHAR(255), UNIQUE), email (VARCHAR(255), UNIQUE), password_hash (VARCHAR(255)), address (TEXT), phone (VARCHAR(20))
- **Products Table:** product_id (INT, PK), name (VARCHAR(255)), description (TEXT), price (DECIMAL(10,2)), inventory_level (INT), seller_id (INT, FK)
- **Orders Table:** order_id (INT, PK), user_id (INT, FK), order_date (TIMESTAMP), total_amount (DECIMAL(10,2)), shipping_address (TEXT)
- **OrderItems Table:** item_id (INT, PK), order_id (INT, FK), product_id (INT, FK), quantity (INT), price (DECIMAL(10,2))

Indexes will be created on foreign keys and frequently queried columns to optimize query performance. Constraints will be implemented to ensure data integrity, such as unique constraints on usernames and emails.

5 API Design and Integration

5.1 RESTful API Specification

The platform will expose a RESTful API for interaction with various clients (web, mobile). The API will follow standard REST principles, using HTTP methods (GET, POST, PUT, DELETE) for CRUD operations. Authentication will be implemented using JWT (JSON Web Tokens). Rate limiting will be implemented to prevent abuse.

Example Endpoint: ‘/products/product$_id$‘

- **GET:** Retrieves details for a specific product. Response: JSON object containing product details.

- **PUT:** Updates the details of a specific product (for sellers only). Request: JSON object containing updated product details. Response: HTTP 200 OK on success, appropriate error codes on failure.

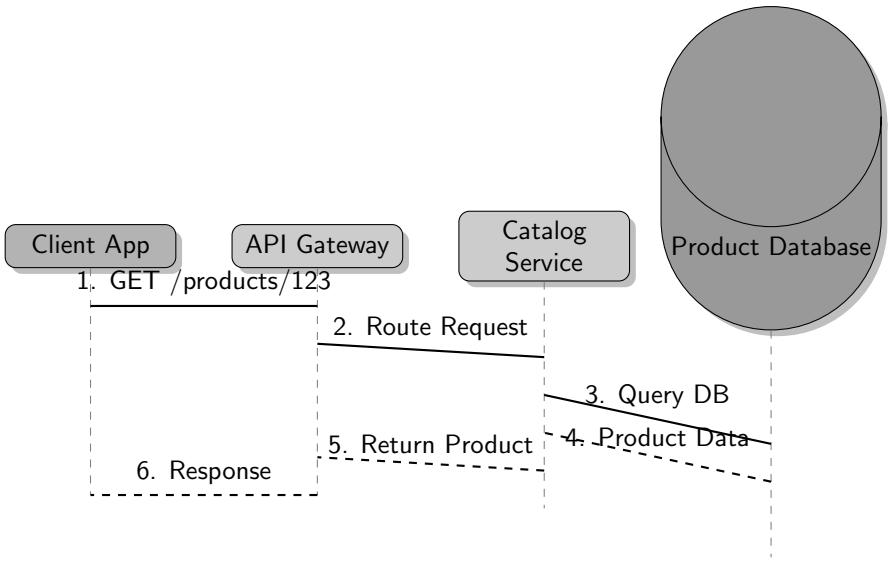

Figure 3: API Request Sequence Diagram: Product Retrieval

5.2 API Security and Rate Limiting

API security will be implemented using JWT for authentication and authorization. Each request will be validated to ensure that the user has the necessary permissions to access the requested resource. Rate limiting will be implemented using a sliding window algorithm to prevent denial-of-service attacks and ensure fair resource allocation. Input validation will be performed on all requests to prevent injection attacks.

6 Security Architecture and Implementation

6.1 Security Requirements and Threat Model

The platform's security architecture will be built on a layered approach, incorporating various security controls at different levels. A threat model will be developed to identify potential vulnerabilities and risks. Key security requirements include:

- Secure authentication and authorization using JWT.
- Data encryption both in transit and at rest.
- Regular security audits and penetration testing.
- Implementation of a robust Web Application Firewall (WAF).
- Protection against common web vulnerabilities such as SQL injection and cross-site scripting (XSS).

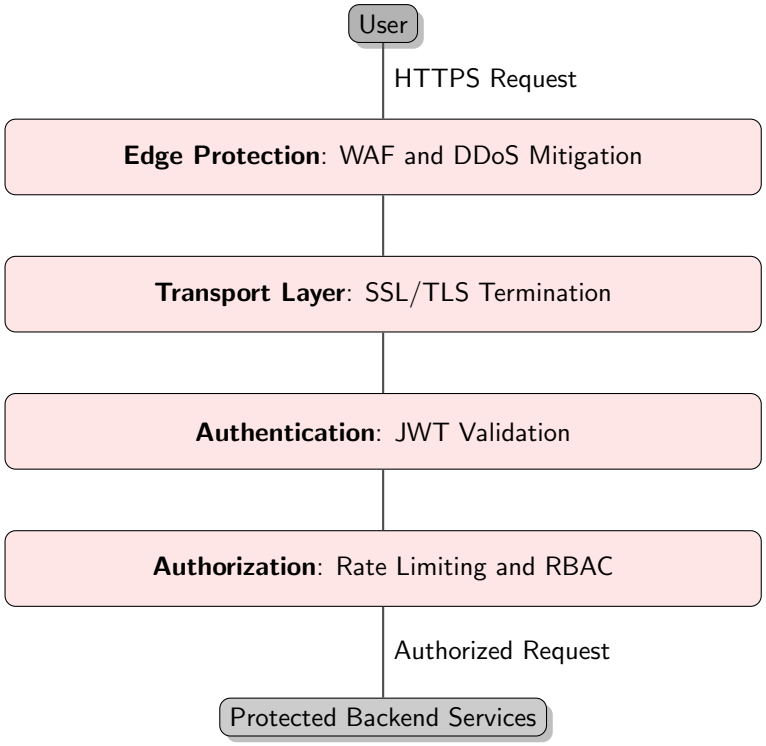

Figure 4: Layered Security Architecture

7 Deployment and Infrastructure

7.1 Cloud Infrastructure Design

The platform will be deployed on AWS using a combination of EC2 instances for application servers, RDS for the database, and S3 for static assets. A load balancer will distribute traffic across multiple application servers, ensuring high availability and scalability. Auto-scaling will be implemented to automatically adjust the number of application servers based on demand. A disaster recovery plan will be in place to ensure business continuity in case of an outage. The infrastructure will be designed for high availability and fault tolerance.

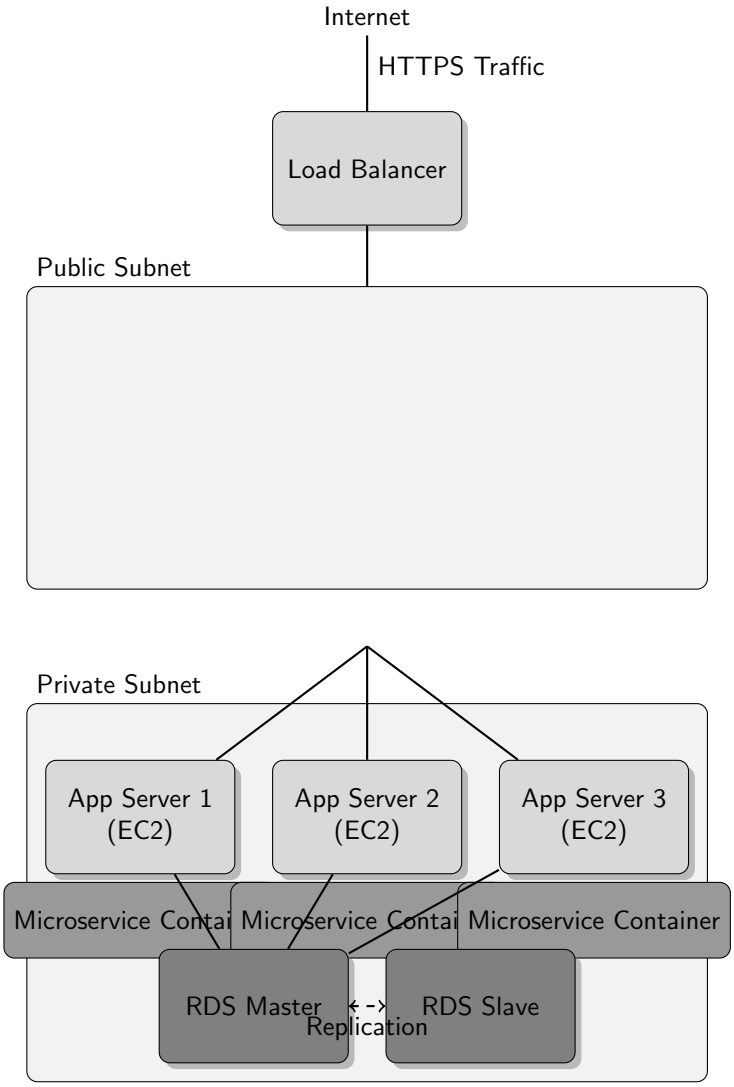

Figure 5: Cloud Deployment Architecture

8 Performance and Scalability

Performance will be monitored using various metrics, including page load time, API response time, and database query time. Performance testing will be conducted throughout the development lifecycle to identify and address performance bottlenecks. Scalability will be achieved through the use of a microservices architecture, load balancing, and auto-scaling. The database will be designed for scalability, using appropriate indexing and query optimization techniques. Caching mechanisms will be implemented to reduce database load.

9 Testing and Quality Assurance

A comprehensive testing strategy will be employed, including unit testing, integration testing, system testing, and user acceptance testing (UAT). Automated testing will be used to ensure code quality and reduce manual testing effort. Test-driven development (TDD) will be used to guide development and ensure that the code meets the specified requirements. Security testing will be conducted to identify and address potential vulnerabilities.

10 Risk Assessment and Mitigation

Potential risks include:

- **Technical Risks:** Challenges in integrating third-party services, unexpected performance bottlenecks, and security vulnerabilities. Mitigation: Thorough integration testing, performance testing, and security audits.
- **Schedule Risks:** Delays in development, testing, or deployment. Mitigation: Agile development methodology, close monitoring of progress, and contingency planning.
- **Resource Risks:** Lack of skilled developers or inadequate resources. Mitigation: Careful resource allocation, training programs, and outsourcing where necessary.

A detailed risk register will be maintained, tracking identified risks, their probability and impact, and the mitigation strategies.

11 Implementation Roadmap and Timeline

The project will be implemented in six phases:

1. **Phase 1 (Month 1-2): Design and Prototyping**
2. **Phase 2 (Month 2-3): Development of Core Microservices**
3. **Phase 3 (Month 3-4): API Development and Integration**
4. **Phase 4 (Month 4-5): Testing and Quality Assurance**
5. **Phase 5 (Month 5-6): Deployment and Go-Live**
6. **Phase 6 (Month 6 onwards): Monitoring and Maintenance**

12 Monitoring and Maintenance

The platform will be monitored using various tools, including cloud monitoring services (CloudWatch), application performance monitoring (APM) tools, and log management systems. Alerts will be set up to notify the operations team of any issues. Regular maintenance will be performed to ensure the platform's stability and performance. A comprehensive maintenance plan will be developed, outlining procedures for routine maintenance, bug fixes, and software updates.

13 Conclusion

This document provides a comprehensive overview of the design for Project Phoenix, a robust and scalable e-commerce platform. The chosen architecture and technologies are well-suited to meet the project's objectives and ensure a successful launch. By following the outlined implementation plan and addressing identified risks, the project is poised to deliver a high-quality e-commerce solution.