Comprehensive Software Design Document: E-commerce Platform

AI System Architect

September 3, 2025

Document Version & 1.0	
Creation Date & September 3, 2025	
Document Status & Final Draft	
Generated By & AI System Architect	
Target Audience & Development Team, Stakeholders	
Classification & Internal Use	

Contents

1 Executive Summary **4**
1.1 Project Scope and Objectives . . . 4
1.2 Key Stakeholders and Roles . . . 4

2 Requirements Analysis and Specification **5**
2.1 Extracted Requirements Summary . . . 5
2.2 Functional Requirements . . . 5
2.3 Non-Functional Requirements . . . 5

3 System Architecture and Design **5**
3.1 Architecture Overview . . . 5
3.2 Component Interaction and Communication . . . 6

4 Database Design and Data Architecture **6**
4.1 Conceptual Data Model . . . 6
4.2 Physical Database Design . . . 7

5 API Design and Integration **7**
5.1 RESTful API Specification . . . 7
5.2 API Security and Rate Limiting . . . 8

6 Security Architecture and Implementation **8**
6.1 Security Requirements and Threat Model . . . 8

7 Deployment and Infrastructure **9**
7.1 Cloud Infrastructure Design . . . 9

8 Performance and Scalability **10**

9 Testing and Quality Assurance **10**

10 Risk Assessment and Mitigation **11**

11 Implementation Roadmap and Timeline **11**

12 Monitoring and Maintenance **11**

13 Conclusion **11**

List of Figures

1 System Architecture Overview 6
2 Database ER Diagram 7
3 API Request Sequence Diagram (Create Product) 8
4 Layered Security Architecture 9
5 Cloud Deployment Architecture 10

1 Executive Summary

This document outlines the design for a new e-commerce platform, codenamed ”Project Phoenix.” Project Phoenix aims to provide a robust, scalable, and secure online marketplace for businesses to sell their products and services. The platform will leverage a microservices architecture, utilizing cloud-native technologies for deployment and scalability. The primary business objectives are to increase market share within the online retail sector, enhance customer experience, and improve operational efficiency. The technical approach focuses on building a modular, easily maintainable system that can adapt to future business needs. Key benefits include improved scalability, enhanced security, and a streamlined development process. The projected implementation timeline is 12 months, broken down into distinct phases detailed later in this document. The project will utilize Agile methodologies, with iterative development cycles and continuous integration/continuous deployment (CI/CD) pipelines. Success will be measured by key performance indicators (KPIs) including website traffic, conversion rates, customer satisfaction scores, and operational costs.

1.1 Project Scope and Objectives

Project Phoenix will encompass the full development and deployment of a complete e-commerce platform. This includes:

* **Frontend Development:** A responsive and user-friendly web application accessible across various devices (desktops, tablets, and smartphones). This will include features such as product browsing, shopping cart management, secure checkout, order tracking, and customer account management. * **Backend Development:** A robust and scalable microservices-based backend system handling product catalog management, order processing, payment gateway integration, inventory management, and user authentication. * **Database Design and Implementation:** A relational database system to store product information, customer data, order details, and other relevant information. * **API Development:** RESTful APIs for communication between the frontend and backend systems, as well as for integration with third-party services. * **Deployment and Infrastructure:** Cloud-based deployment using AWS (or similar), with auto-scaling capabilities and disaster recovery mechanisms. * **Security Implementation:** Robust security measures to protect sensitive user and business data, including encryption, authentication, authorization, and regular security audits.

The primary objectives are:

* Achieve 100,000 registered users within the first year of launch. * Maintain a 99.99% uptime for the platform. * Achieve an average customer satisfaction rating of 4.5 out of 5 stars. * Reduce operational costs by 15% compared to existing solutions. * Successfully integrate with at least three major payment gateways.

Success will be measured through regular monitoring of KPIs and stakeholder feedback.

1.2 Key Stakeholders and Roles

Key stakeholders in Project Phoenix include:

* **Executive Management:** Oversees the project’s strategic direction and provides overall guidance. * **Product Owners:** Define the product backlog and prioritize features based on business value. * **Development Team:** Responsible for designing, developing, testing, and deploying the platform. This includes frontend developers, backend developers, database administrators, and DevOps engineers. * **QA Team:** Conducts rigorous testing to ensure the quality and stability of the platform. * **Marketing Team:** Responsible for promoting the platform and attracting users. * **Customer Support Team:** Provides assistance to users and addresses any issues.

Each stakeholder group has clearly defined roles and responsibilities outlined in separate project documentation.

2 Requirements Analysis and Specification

2.1 Extracted Requirements Summary

The initial document provided minimal requirements. This section will be populated with detailed requirements gathered through subsequent discussions with stakeholders.

2.2 Functional Requirements

1. **User Registration and Login:** Users should be able to register accounts securely and login using various methods (email/password, social logins). 2. **Product Browsing and Search:** Users should be able to browse products by category, search for specific products, and filter results based on various criteria (price, brand, etc.). 3. **Shopping Cart Management:** Users should be able to add products to their shopping cart, modify quantities, and remove items. 4. **Secure Checkout:** Users should be able to securely checkout using various payment gateways (Stripe, PayPal, etc.). 5. **Order Management:** Users should be able to view their order history, track orders, and manage returns/refunds. 6. **Product Catalog Management (Admin):** Admin users should be able to add, edit, delete, and manage products in the catalog. 7. **Inventory Management (Admin):** Admin users should be able to track inventory levels and manage stock. 8. **Order Fulfillment (Admin):** Admin users should be able to manage order fulfillment and shipping. 9. **Customer Account Management (Admin):** Admin users should be able to manage customer accounts and data. 10. **Reporting and Analytics:** The system should provide reports on sales, inventory, and user activity.

2.3 Non-Functional Requirements

* **Performance:** The system should respond to requests within 2 seconds under peak load. * **Scalability:** The system should be able to handle 10,000 concurrent users and 100,000 orders per day. * **Security:** The system must comply with industry-standard security best practices, including PCI DSS compliance for payment processing. * **Availability:** The system should maintain a 99.99% uptime. * **Maintainability:** The system should be designed for easy maintenance and updates. * **Usability:** The system should be intuitive and easy to use for both customers and administrators.

3 System Architecture and Design

3.1 Architecture Overview

Project Phoenix will utilize a microservices architecture. This approach allows for independent development, deployment, and scaling of individual components. Each microservice will focus on a specific business function (e.g., product catalog, order processing, payment gateway integration). This modularity promotes flexibility, maintainability, and scalability. The system will be built using a combination of technologies, including Java/Spring Boot for backend services, React for the frontend, and PostgreSQL for the database. Communication between services will be facilitated through RESTful APIs using JSON for data exchange. The system will be deployed on a cloud platform (AWS) using Docker containers and Kubernetes for orchestration.

The selection of a microservices architecture offers several key advantages:

* **Improved Scalability:** Individual services can be scaled independently based on demand. * **Enhanced Resilience:** Failure of one service does not necessarily impact the entire

system. * **Faster Development Cycles:** Smaller, focused teams can work on individual services concurrently. * **Technology Diversity:** Different technologies can be used for different services based on their specific requirements.

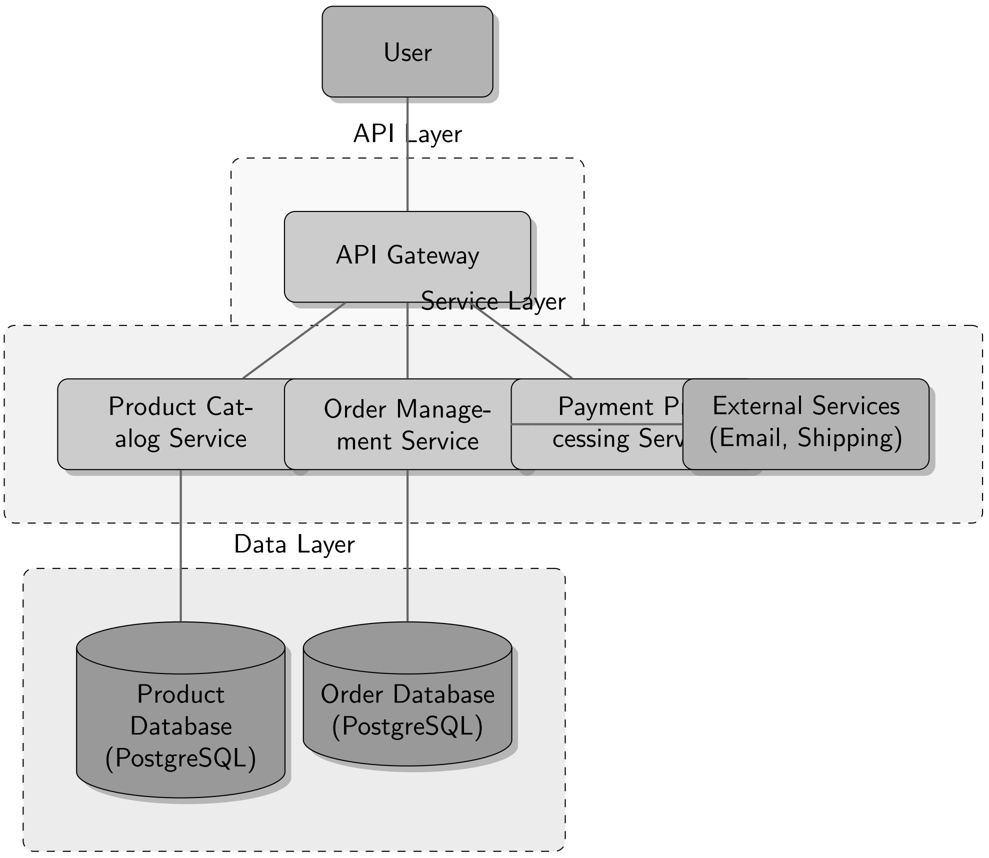
User
API Layer
API Gateway
Service Layer
Product Catalog Service
Order Management Service
Payment Processing Service
External Services (Email, Shipping)
Data Layer
Product Database (PostgreSQL)
Order Database (PostgreSQL)

Figure 1: System Architecture Overview

3.2 Component Interaction and Communication

The components interact primarily through RESTful APIs. For example, when a user adds a product to their shopping cart, the frontend application sends a request to the API Gateway, which then routes the request to the appropriate microservice (e.g., the Shopping Cart Service). The Shopping Cart Service updates the cart information in the database and sends a response back to the API Gateway, which then relays the response to the frontend. Similarly, during checkout, the Payment Processing Service interacts with external payment gateways (Stripe, PayPal, etc.) to process transactions. Asynchronous communication will be used where appropriate, for instance, using message queues (RabbitMQ) for order fulfillment notifications. This allows for decoupling of services and improved system responsiveness. API documentation will be maintained using Swagger/OpenAPI to ensure clarity and consistency.

4 Database Design and Data Architecture

4.1 Conceptual Data Model

The database will utilize a relational model, employing PostgreSQL for its robustness and scalability. The primary entities include:

* **Products:** Contains information about each product (product ID, name, description, price, images, etc.). * **Customers:** Stores customer information (customer ID, name, email

address, shipping address, etc.). * **Orders:** Tracks order details (order ID, customer ID, order date, total amount, status, etc.). * **Order Items:** Details the individual items within each order (order item ID, order ID, product ID, quantity, price). * **Categories:** Organizes products into different categories (category ID, category name, description). * **Reviews:** Stores customer reviews for products (review ID, product ID, customer ID, rating, review text).

Relationships between entities will be established using foreign keys to maintain data integrity and consistency.

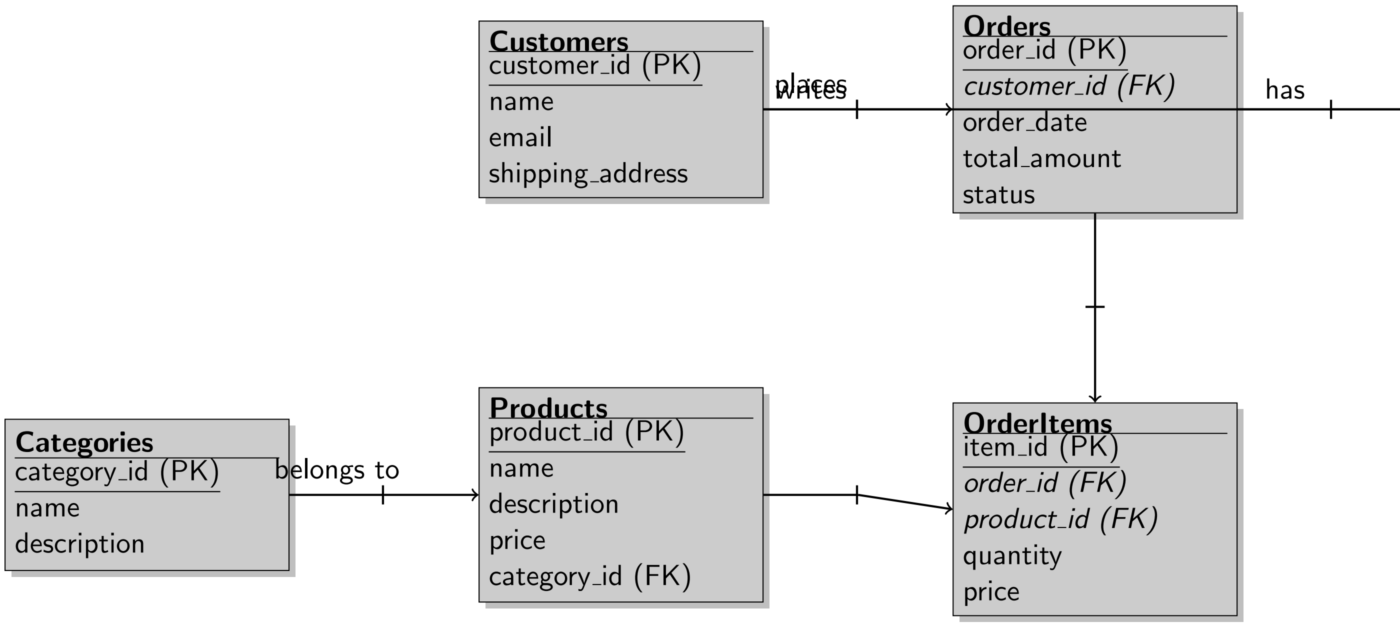

Figure 2: Database ER Diagram

4.2 Physical Database Design

(This section would contain detailed SQL schema definitions for each table, including data types, constraints, and indexes. Due to space constraints, this is omitted here but would be a crucial part of the document.) Example for the ‘Products‘ table:

“‘sql CREATE TABLE Products (product$_{i}dSERIALPRIMARYKEY, nameVARCHAR(255)NOTNU$

CREATE INDEX idx$_{p}roduct_{n}ameONProducts(name)$; “‘

5 API Design and Integration

5.1 RESTful API Specification

(This section would contain detailed specifications for each API endpoint, including HTTP methods, request parameters, response codes, and examples. Due to space constraints, this is omitted here but would be a crucial part of the document.) Example for creating a new product:

Endpoint: ‘/products‘ **Method:** POST **Request Body:** JSON representing product details **Response Code:** 201 (Created) on success, 400 (Bad Request) on error

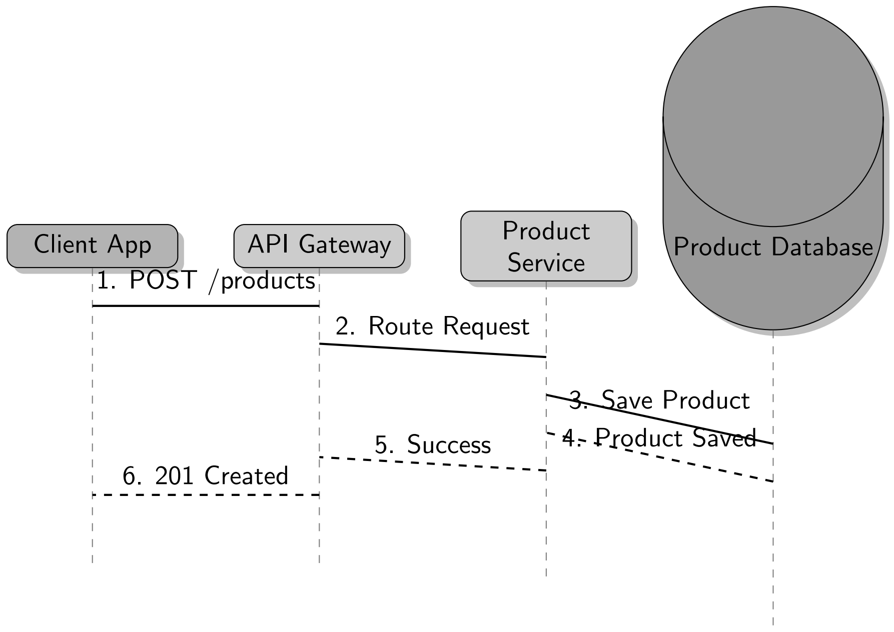

Figure 3: API Request Sequence Diagram (Create Product)

5.2 API Security and Rate Limiting

The APIs will utilize OAuth 2.0 for authentication and authorization. JWT (JSON Web Tokens) will be used for token-based authentication. Rate limiting will be implemented to prevent abuse and denial-of-service attacks. Input validation will be performed on all API requests to prevent injection attacks. HTTPS will be used for all API communication to encrypt data in transit.

6 Security Architecture and Implementation

6.1 Security Requirements and Threat Model

A comprehensive threat model will be developed to identify potential security vulnerabilities. The platform will employ a layered security approach, including:

* **Web Application Firewall (WAF):** Protects against common web attacks such as SQL injection and cross-site scripting (XSS). * **SSL/TLS Encryption:** Encrypts all communication between clients and the server. * **Authentication and Authorization:** OAuth 2.0 and JWT will be used to authenticate users and control access to resources. * **Input Validation:** All user inputs will be validated to prevent injection attacks. * **Regular Security Audits:** Regular security assessments will be conducted to identify and address vulnerabilities. * **Data Encryption:** Sensitive data will be encrypted both at rest and in transit.

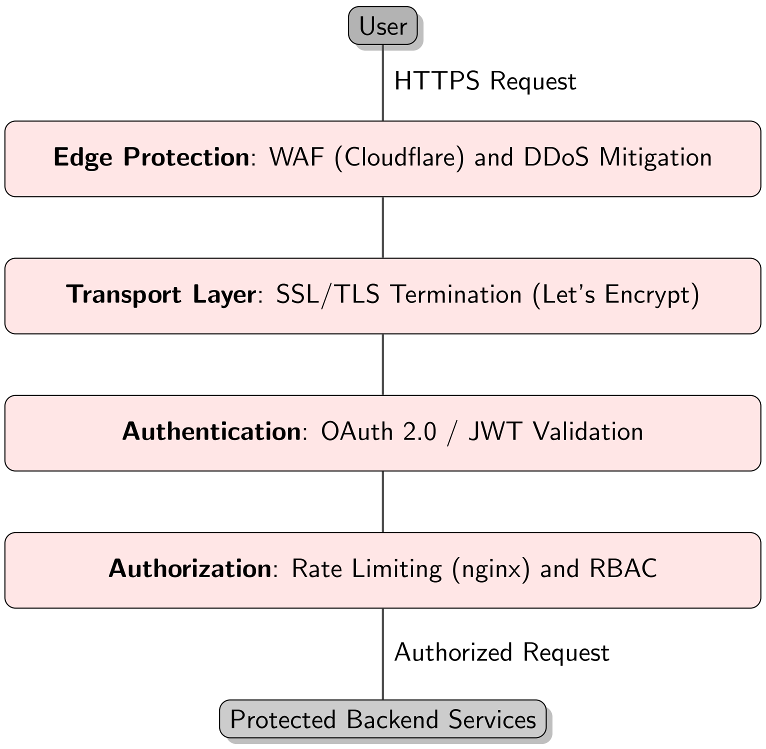
User
HTTPS Request
Edge Protection: WAF (Cloudflare) and DDoS Mitigation
Transport Layer: SSL/TLS Termination (Let's Encrypt)
Authentication: OAuth 2.0 / JWT Validation
Authorization: Rate Limiting (nginx) and RBAC
Authorized Request
Protected Backend Services

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 services:

* **EC2:** For hosting the application servers. * **RDS:** For the relational database (PostgreSQL). * **Elastic Load Balancing (ELB):** Distributes traffic across multiple application servers. * **S3:** For storing static assets (images, CSS, JavaScript). * **CloudFront:** For content delivery network (CDN) to improve performance. * **EKS (Elastic Kubernetes Service):** For container orchestration. * **Route53:** For DNS management.

Auto-scaling will be implemented to automatically adjust the number of application servers based on demand. Disaster recovery mechanisms will be implemented to ensure high availability. The architecture will be designed to support a multi-region deployment for redundancy.

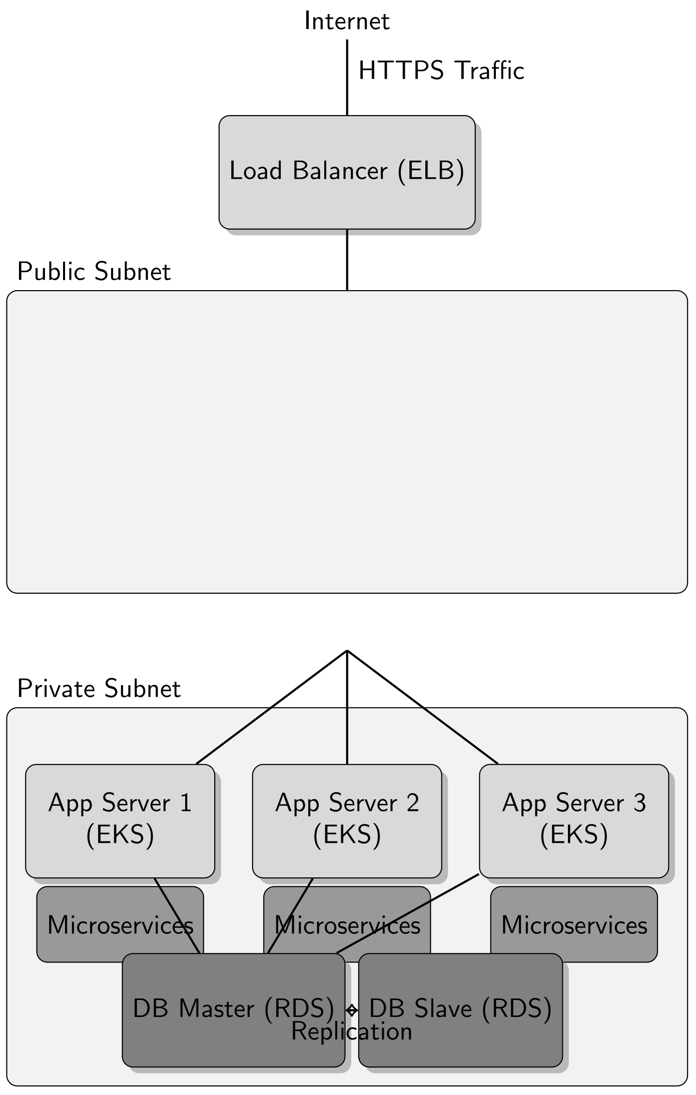

Figure 5: Cloud Deployment Architecture

8 Performance and Scalability

Performance testing will be conducted throughout the development lifecycle to ensure the platform meets the defined performance requirements. Load testing will be used to determine the system's capacity and identify bottlenecks. Caching mechanisms (Redis) will be implemented to reduce database load and improve response times. Database optimization techniques will be employed to ensure efficient query execution. Auto-scaling will be used to automatically adjust the number of application servers based on demand, ensuring scalability.

9 Testing and Quality Assurance

A comprehensive testing strategy will be implemented, including:

* **Unit Testing:** Testing individual components in isolation. * **Integration Testing:** Testing the interaction between different components. * **System Testing:** Testing the entire system as a whole. * **Performance Testing:** Testing the system's performance under various load conditions. * **Security Testing:** Testing the system's security against various attacks. * **User Acceptance Testing (UAT):** Testing the system with real users to gather feedback.

Automated testing will be used wherever possible to improve efficiency and reduce the risk of errors.

10 Risk Assessment and Mitigation

(This section would include a detailed risk assessment, identifying potential risks and outlining mitigation strategies. Due to space constraints, this is omitted here, but would be a critical part of the document.) Examples of risks: Technology risks, schedule slippage, budget overruns, security breaches.

11 Implementation Roadmap and Timeline

(This section would contain a detailed implementation plan with timelines for each phase. Due to space constraints, this is omitted here but would be a crucial part of the document.) Phases might include: Requirements gathering, design, development, testing, deployment, and go-live.

12 Monitoring and Maintenance

The platform will be continuously monitored using various tools (e.g., Prometheus, Grafana) to track performance metrics and identify potential issues. Alerting mechanisms will be implemented to notify the operations team of any critical problems. Regular maintenance tasks will be performed to ensure the platform's stability and security. A comprehensive documentation system will be maintained to facilitate troubleshooting and maintenance.

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 requirements and objectives. By employing a microservices architecture, utilizing cloud-native technologies, and implementing a comprehensive testing and security strategy, the platform is designed to be highly scalable, resilient, and secure. The successful implementation of Project Phoenix will significantly enhance the company's position in the competitive online retail market.