**AMAZON E-COMM APPLICATION**

**Document Version:** 1.0

**Date:** 25/02/2024

**Prepared By:** ABHINAY KUMAR. NALBAND

TABLE OF CONTENTS

1. **Introduction**

* + Why this HLD?
  + Scope
  + Definitions
  + Overview
  + Purpose of this Document

2. **Features , Assumptions &** **Prerequisites**

3. **Architecture Overview**

* + Technology Stack
  + Design Principles
  + Scalability Considerations
  + Activity Diagram

**4. Non Functional Requirements**

**5. Component Details**

* Search for products
* Recommendations on User Homepage
* Place Order
* Check Order Status
* Write / View Product Review

**6. Application Modules**

**7. Security Considerations**

* Data Encryption
* Authentication Mechanisms
* Authorization Policies
* Secure Payment Processing

**8. Deployment Strategy**

* Containerization with Docker
* Orchestration with Kubernetes
* CI/CD Pipeline

**9. Conclusion**

**10. System design**

1. **Introduction**

* Why this HLD document?

This High-Level Design (HLD) document is created to provide a structured plan and overview of the architecture, features, and key components of a application. The HLD document plays a crucial role in guiding the development process, aligning stakeholders, mitigating risks, ensuring compliance and security, and ultimately, delivering a successful application that meets the needs and expectations of its users.

The HLD will:

* Present all of the design aspects and define them in detail
* Describe the user interface being implemented
* Describe the hardware and software interfaces
* Describe the performance requirements
* Include design features and the architecture of the project
* List and describe the non-functional attributes like:

1. Scalability
2. Reliability
3. Security
4. Performance
5. Usability
6. Compliance
7. Maintainability
8. Monitoring and Logging

* Scope

This HLD documentation is comprehensive, covering various aspects essential for the successful design, development, and deployment of a application. It provides a holistic view of the application's architecture, features, data management, security, and deployment strategy, serving as a guide for the development team and stakeholders involved in the project.

* Definitions
* High Level Design (HLD) – A document that provides an overview of the architecture, features and key components of a software system.
* Microservices architecture – An architectural style that structures an application as a collection of loosely coupled services, each encapsulating a specific business function.
* Containerization – Process of encapsulating an application into containers which can be deployed across different environments.
* Continuous Integration/ Continuous Deployment (CI/CD) – These pipelines automate the build, test and deployment processes, enabling the rapid and reliable software delivery.
* Relational Databases – Organizes data into tables (rows and columns) with pre-defined relation between them.
* NoSQL Database - A non-relational database that provides flexible data models and scalability for handling large volumes of unstructured or semi-structured data.
* Authentication – The process of verifying the identity of users or systems accessing a software application.
* Authorization – The process of granting or denying access to specific resources or functionalities based on authenticated user’s permissions.
* Payment Gateway Integration – The integration of third-party payment processing services into a software application to facilitate secure and convenient payment transactions.
* Overview of Application:-
* Amazon is an advanced e-commerce platform offering a seamless shopping experience across diverse product categories. With a user-friendly interface, it provides easy navigation, personalized recommendations, and secure payment options. Amazon prioritizes security, partnering with trusted payment gateways for safe transactions. From fashion to electronics, users can explore a wide range of products from reputable brands. Amazon aims to revolutionize online retail by delivering convenience, reliability, and quality service to modern consumers.
* Purpose of this Document
* This High-Level Design (HLD) document specifies the implementation, including inter-component dependencies, and provides sufficient design detail that any product based on this HLD will satisfy the product requirements.

1. **Features:**

* Search for products.
* Recommendations on UserHomepage.
* Place Order.
* Check Order Status.
* Write / View Product Review.

**Assumptions:-**

1. User Profile Creation is available.
2. Product onboard is Provided.
3. Payment Service is Available.

**Prerequisite**

i. Internet Connectivity: It is assumed that users will have reliable internet access to engage with Amazon seamlessly across various devices.

ii. Device Compatibility: Amazon is designed to be compatible with smartphones, tablets, and computers, ensuring accessibility for a wide range of users.

iii. Product Availability: The assumption is made that products showcased on Amazon are readily available for purchase, maintaining user trust and satisfaction.

iv. Security Measures: Robust security measures will be implemented to safeguard user data and ensure secure transactions, maintaining the integrity of Amazon's platform.

v. Payment Gateway Integration: Amazon's functionality relies on seamless integration with trusted payment gateways for efficient transaction processing and user convenience.

vi. Server Infrastructure: Availability of stable server infrastructure is a prerequisite to support platform operations and ensure consistent performance.

vii. Database Management: Amazon depends on a reliable database management system for efficient data storage, retrieval, and management, ensuring seamless user experiences.

viii. Third-party Integrations: Integration with third-party services such as shipping providers and analytics platforms enhances Amazon's functionality and user experience.

These assumptions and foundational elements will be regularly reviewed and assessed

throughout the design process to ensure their validity and alignment with Amazon's objectives.

**3. Architectural Overview**

The Amazon application will be built on a robust microservices architecture, leveraging modern technologies and design principles to ensure scalability, reliability, and performance. The architecture will comprise various backend services responsible for different aspects of the application, each encapsulating specific functionality and interacting with others through well-defined APIs.

**i. Technology Stack:**

The technology stack refers to the combination of programming languages, frameworks, libraries, databases, and other tools used to build the Amazon like application. Here's how each component of the technology stack might be utilized:

**i. Backend Technologies:**

o Programming Language: Java

o Framework: Spring Boot (for Java) which provides a robust foundation for building RESTful APIs and handling business logic.

o Database: A combination of relational ( MySQL) and NoSQL (MongoDB) databases for storing different types of data efficiently.

**ii. Frontend Technologies:**

o JavaScript Framework: Angular for building interactive and responsive user interfaces.

**iii. Containerization and Orchestration:**

* Containerization Platform: Docker for packaging applications and their dependencies into containers, ensuring consistency across development, testing, and production environments.
* Orchestration Platform: Kubernetes for automating deployment, scaling, and management of containerized applications, providing resilience and scalability.

**ii. Design Principles:** The architectural design of the Amazon application adheres to several key principles to ensure modularity, scalability, and maintainability:

**i. Microservices Architecture:** Decomposes the application into smaller, loosely coupled services, each responsible for specific business functions. This promotes modularity, agility, and independent deployment of services.

**ii. Service-Oriented Design:** Services are designed around specific business capabilities and expose well-defined APIs for interaction with other services and clients. This fosters reusability, flexibility, and separation of concerns.

**iii. API-First Approach**: Design APIs before implementing functionality, ensuring consistency, interoperability, and ease of integration with frontend and third-party systems.

**iv. Fault Tolerance and Resilience**: Services are designed to handle failures gracefully, with built-in redundancy, retry mechanisms, and circuit breakers to maintain system stability and availability.

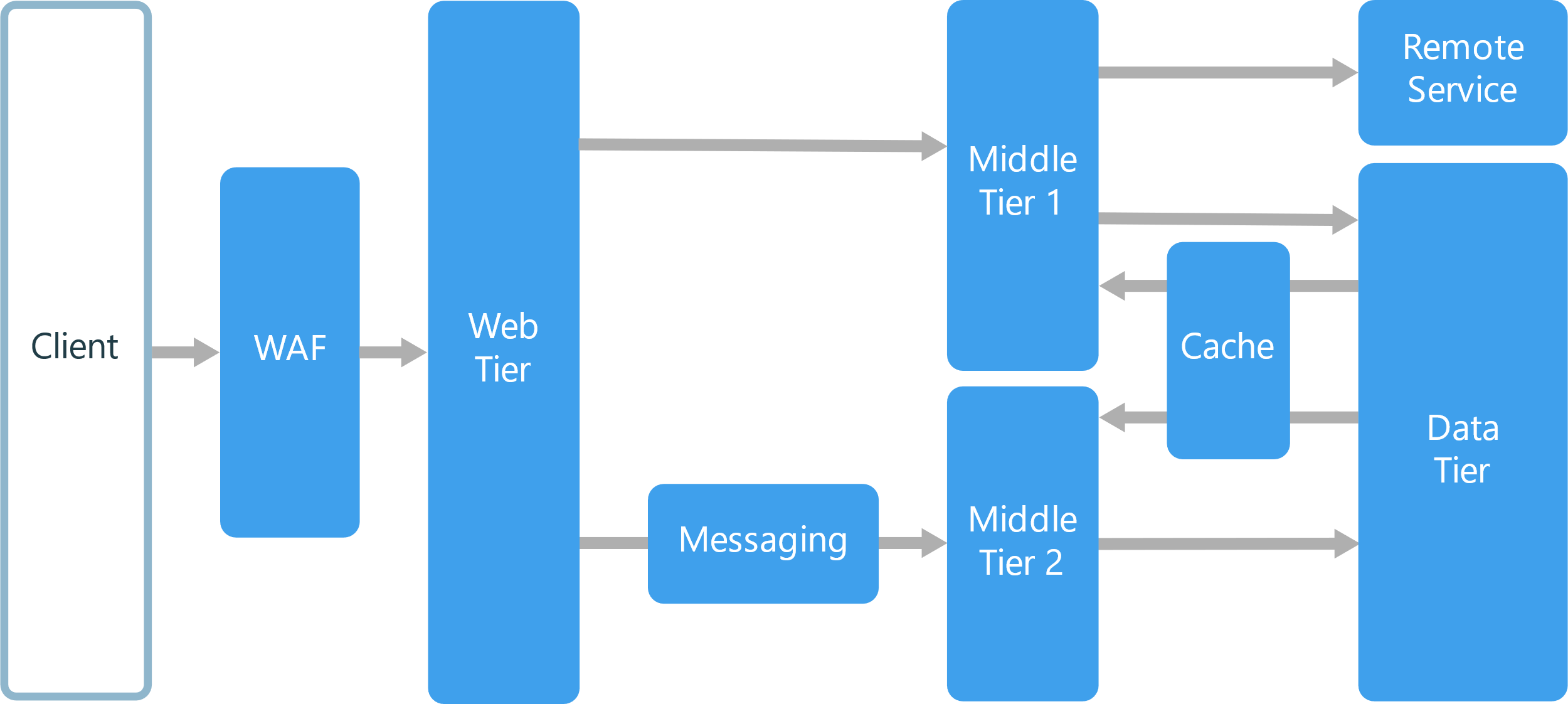
**iii. Scalability Considerations:**

Scalability is a critical aspect of the Amazon application's architecture, enabling it to handle increasing user demand and workload efficiently. Here's how scalability is addressed:

**i. Horizontal Scaling:** Services are designed to scale horizontally by adding more instances or replicas to distribute load and handle increased traffic effectively. This allows the application to scale out across multiple servers or containers.

**ii. Load Balancing:** Load balancers are employed to evenly distribute incoming requests across multiple instances or replicas of services, ensuring optimal resource utilization and preventing overload on individual components.

**iii. Caching Mechanisms:** Caching is used to reduce latency and improve performance by storing frequently accessed data in memory or distributed caches (e.g., Redis). This helps offload the backend systems and handle spikes in traffic more efficiently.



1. **Non Functional Requirements:**

* Low latency(Recommendation & Search)
* High Consistency(Placing Order, order Status, and Payments)

1. Component Details:-
2. **Search for Products:**

**Components**:

Frontend (Mobile/Web Interface)

Backend Server

Database (Product Listings)

**Flow**:

Users initiate product searches through the frontend interface.

Frontend sends search queries to the backend server.

Backend processes search queries, retrieves relevant product information from the database, and sends the results back to the frontend for display.

1. **Recommendations on User Homepage:**

**Components**:

Frontend (User Homepage Interface)

Backend Server

Database (User Preferences, Product Data)

**Flow**:

Backend analyzes user preferences and behavior.

Based on the analysis, backend generates personalized recommendations.

Recommendations are sent to the frontend and displayed on the user's homepage interface.

1. **Place Order:**

**Components**:

Frontend (Shopping Cart Interface)

Backend Server

Database (Orders, Product Inventory)

**Flow:**

Users confirm their orders through the shopping cart interface on the frontend.

Frontend transmits order details to the backend server.

Backend validates the order, updates the product inventory, and stores comprehensive order information securely in the database.

1. **Check Order Status:**

**Components**:

Frontend (Order Status Interface)

Backend Server

Database (Orders)

**Flow**:

Users access the order status interface through the frontend.

Frontend sends requests for order status information to the backend server.

Backend retrieves order status from the database and sends it back to the frontend for display.

1. **Write / View Product Review**:

**Components**:

Frontend (Product Page Interface)

Backend Server

Database (Product Reviews)

**Flow**:

Users write or view product reviews on the product page interface on the frontend.

Frontend sends review submissions or requests to view reviews to the backend server.

Backend processes review submissions, stores them securely in the database, and retrieves review data to display on the frontend.

Each of these features follows a similar structure with frontend interfaces interacting with backend servers, which in turn interact with databases to fulfill user requests and provide necessary functionalities.

1. **Application Modules**

* Core Components

Amazon's core components are developed using Java, chosen for its robustness and scalability. These components are designed as microservices, allowing for independent development, deployment, and scalability. Each microservice is implemented using the Spring Boot framework, facilitating rapid development and deployment of standalone services.

* User Interfaces and Layers

Amazon offers intuitive and responsive user interfaces developed using HTML, CSS, and JavaScript for web interfaces, and frameworks like React.js for dynamic and interactive user experiences. These interfaces interact with the backend microservices through RESTful APIs, ensuring seamless communication between the frontend and backend layers.

* Database Layer

The database layer of Amazon utilizes a combination of relational and NoSQL databases. A relational database management system (RDBMS) such as MySQL or PostgreSQL is used for structured data storage, while a NoSQL database like MongoDB is employed for handling unstructured or semi-structured data. These databases are accessed through Spring Data JPA or Hibernate for efficient data manipulation.

* Business Logic Modules

Amazon's business logic modules are implemented within the backend microservices using Java. These modules handle various tasks such as user authentication, product management, cart management, order processing, and payment validation. Business logic is encapsulated within service classes and interacts with the database layer using JPA repositories.

* Licensed Components

Amazon may incorporate licensed components or third-party libraries to enhance its functionality. These components may include encryption libraries such as Bouncy Castle for secure data transmission, payment gateway APIs like PayPal or Stripe for handling payment transactions, and logging frameworks such as Log4j for effective logging and monitoring.

* APIs and External Connections

Amazon integrates with external systems and services through APIs to extend its capabilities. Payment processing is facilitated through integration with payment gateway APIs such as PayPal API or Stripe API. Additionally, integration with shipping provider APIs like FedEx or UPS enables seamless order fulfillment. Analytics APIs such as Google Analytics may also be integrated for tracking user behavior and performance metrics.

1. **Security Considerations**

Security will be paramount in the design and implementation of the Amazon-like application, with measures in place to protect user data, authenticate users securely, and ensure secure payment processing. Encryption, authentication

compliance with industry standards such as PCI DSS will be implemented to safeguard user information and maintain the integrity of the platform.

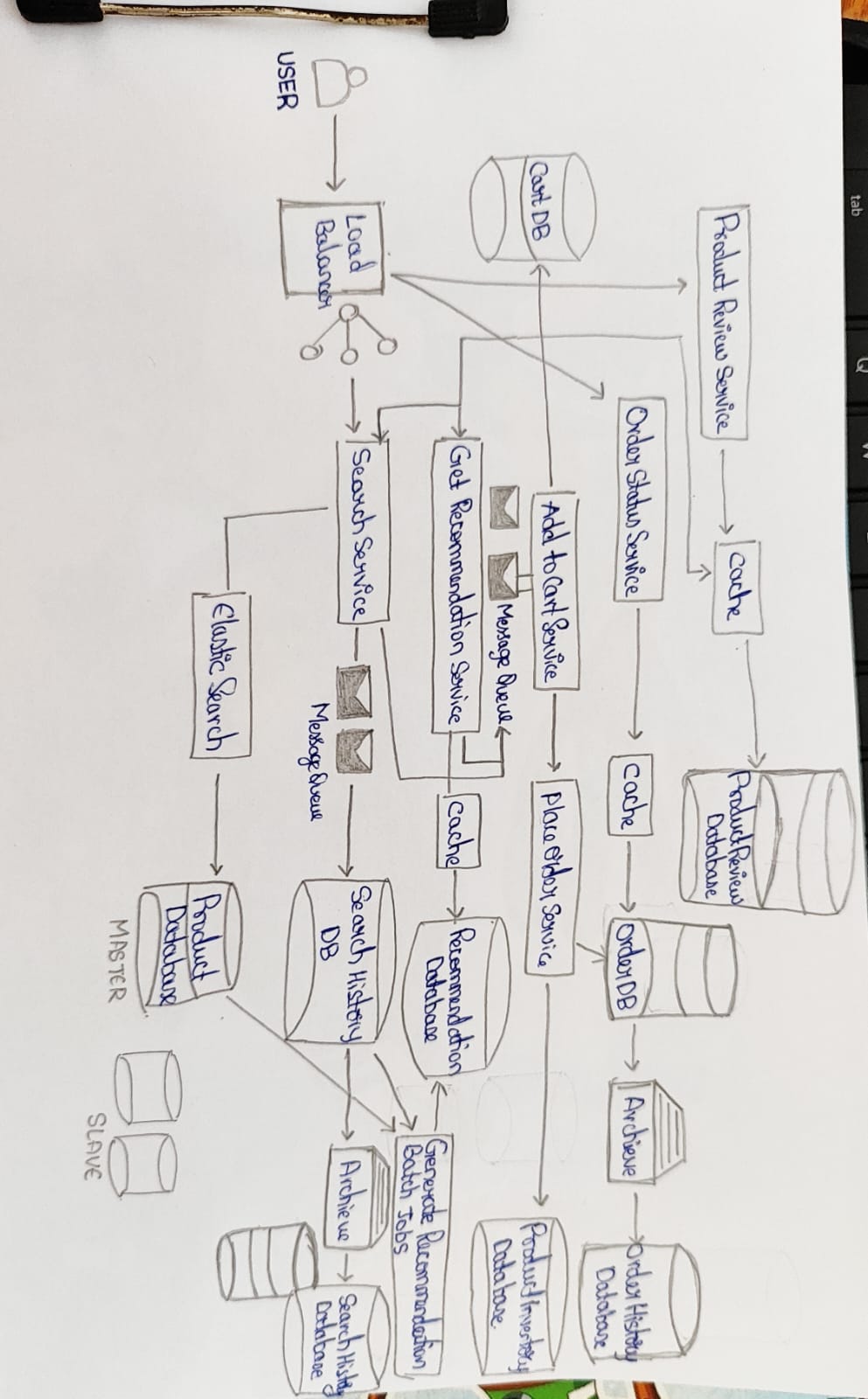
1. **Deployment Strategy**

The application will be deployed using containerization with Docker for packaging and Kubernetes for orchestration, enabling seamless scaling, deployment, and management of containerized microservices. Continuous Integration/Continuous Deployment (CI/CD) pipelines will be implemented to automate the deployment process, ensuring rapid delivery of updates and enhancements to the platform.

1. **Conclusion**

The Amazon-like application represents a groundbreaking solution to meet the evolving needs of modern consumers in the digital product delivery landscape. By offering a comprehensive platform with user-friendly features, robust architecture, and stringent security measures, the application aims to redefine the dining experience and become the go-to destination for users seeking culinary delights at their fingertips.

**System Design:-**

****