Architecture Design Document for Inventory Management System (IMS)

1. Introduction

1.1 Purpose

The purpose of this document is to provide a detailed architectural framework for the Inventory Management System (IMS) being developed for Kirana stores. This document outlines the system's architecture, covering key components, deployment configurations, and critical non-functional requirements such as performance, security, and scalability. It serves as a guide for developers, architects, and stakeholders, ensuring alignment with project goals and overall system design.

1.2 Scope

The scope of this document includes the high-level design of the IMS, focusing on core system architecture, interactions between components, deployment configurations, and the system's ability to meet the needs of Kirana store owners and staff. This document is based on Agile development principles, emphasizing iterative development and continuous feedback.

1.3 References

- Software Requirements Specification (SRS) for IMS
- Agile Development Principles
- SQL Database (MySQL) Design Patterns
- Object-Oriented Design Patterns

---

2. Architectural Representation

The architecture of the IMS is represented through multiple views to provide a comprehensive overview of its structure and behavior:

- **Use-Case View**: Focuses on the main use cases driving the system’s design.
- **Logical View**: Describes the system’s structure in terms of modules and their interactions.
- **Process View**: Explains how processes interact within the system.
- **Deployment View**: Describes how the system is deployed and the interaction between different components.

---

3. Architectural Goals and Constraints

- **Scalability**: The system must handle increasing numbers of products, transactions, and users (store owners) without a decline in performance.
- **Security**: Secure user authentication and role-based access control are essential for protecting sensitive inventory and sales data.
- **Usability**: The system must be user-friendly and accessible across devices (desktop and mobile).
- **Reliability**: The system should ensure high availability (99.9% uptime) and robust error-handling mechanisms.

---

4. Use-Case View

4.1 Architecturally Significant Use Cases

1. **Product Management**:
    - Store owners and managers can add, update, and delete products, track stock levels, set reorder points, and view expiry dates.
2. **Sales Management**:
    - Cashiers enter sales transactions, manually input payment amounts, and update inventory.
3. **Stock Monitoring**:
    - The system tracks stock levels, automatically generates low-stock alerts, and updates the product's status.
4. **Supplier Management**:
    - Automates purchase order generation and tracks supplier information.
5. **Reporting and Analytics**:
    - Generates reports on sales, stock levels, and supplier orders, providing insights to store owners and managers.

---

5. Logical View

5.1 Architecture Overview – Modules and Subsystem Layering

The system is structured into the following key modules:

1. **Product Management Module**:
    - Manages CRUD operations for products, including stock levels and expiration dates.
2. **Sales Module**:

    - Records sales transactions, updates stock, and stores payment details (manually entered by the cashier).
3. **Stock Monitoring Module**:
    - Tracks stock levels and triggers alerts when stock falls below the reorder threshold.
4. **Supplier Management Module**:
    - Automates and manages the process of generating and sending purchase orders to suppliers.
5. **Reporting and Analytics Module**:
    - Provides data insights into sales trends, inventory turnover, and product performance.

5.2 Process View

5.2.1 Processes

- **User Interaction Process**: Manages the user interface (UI) for product management, sales, and reporting.
- **Inventory Management Process**: Automates stock updates and generates notifications for low stock or expiring products.
- **Supplier Order Process**: Automates purchase orders based on stock levels and sends notifications to suppliers.

5.2.2 Process to Design Elements

- **Product Management Process** interacts with the **Product Management Module** for CRUD operations on products.
- **Sales Process** interacts with both the **Sales Module** and the **Product Management Module** to update stock and store sales transactions.
- **Stock Monitoring Process** is tied to the **Stock Monitoring Module** and the **Supplier Management Module**.

5.2.3 Model Dependencies

- **Sales Module** depends on the **Product Management Module** to update inventory after each transaction.
- **Stock Monitoring Module** depends on the **Product Management Module** for real-time stock level data.
- **Supplier Management Module** is linked to the **Stock Monitoring Module** for automated purchase orders.

---

6. Deployment View

6.1 User Devices

- **Desktop/Mobile Devices**: Used by store owners, managers, and cashiers to access the IMS. The front-end is built using **React** for an intuitive interface, which interacts with the back-end through REST APIs.

6.2 Backend Server

- **Purpose**: Hosts the core application logic, including stock monitoring, sales processing, and reporting.
- **Components**:
    - **Product Management Service**
    - **Sales Management Service**
    - **Stock Monitoring Service**
    - **Supplier Management Service**

6.3 Database Server

- **Purpose**: Stores all product, sales, and supplier data in a **MySQL database**.
- **Components**:
    - **Product Table**
    - **Sales Table**
    - **Supplier Table**
    - **StockTransaction Table**

6.4 Authentication Server

- **Purpose**: Manages user authentication using **Auth0** or another authentication service.
- **Components**:
    - **User Authentication Module**
    - **Role-Based Access Control**

---

7. Performance

- The system is optimized for **real-time inventory updates** with low-latency SQL queries.
- **Load balancing** and **database indexing** strategies will be applied to maintain performance during peak usage.
- Performance testing will ensure the system can handle multiple cashiers recording sales transactions simultaneously.

---

8. Quality

- **Code Reviews**: Regular peer reviews will ensure code quality and adherence to best practices.

- **Automated Testing**: Tests will be conducted to maintain code reliability, using tools such **Gantt and Whiteboard and Sticky Notes.**