Software Architecture and Design Specification (SAD)

Project: Perishables Management System (Retail Stores)

Version: 1.0

Authors: Rithvik Matta, Rishi Sheth, Rishil Abhijit, Rohan Jogi

Date: 09-10-2025

Status: Draft

1. Introduction

1.1 Purpose

This document specifies the architecture and design of the Perishables Management System (PMS). It translates the requirements defined in the SRS into a structured architecture and design model to guide implementation and ensure traceability between requirements and the system design.

1.2 Scope

The PMS supports inventory management, expiry tracking and alerts, automatic discounting of near-expiry items, supplier and purchase management, reporting & analytics, and secure role-based access control. It integrates with POS systems and barcode/QR scanners to streamline perishable goods management in retail stores.

1.3 Audience

This document is intended for Developers, QA Engineers, Retail Store Managers & Staff, System Administrators, and Instructors.

1.4 Definitions

POS – Point of Sale
SKU – Stock Keeping Unit
FIFO – First In, First Out
PMS – Perishables Management System

2. Document Overview

This document provides architectural deliverables including UML diagrams, architecture decisions, threat models, API design, deployment views, and traceability to requirements.

3. Architecture

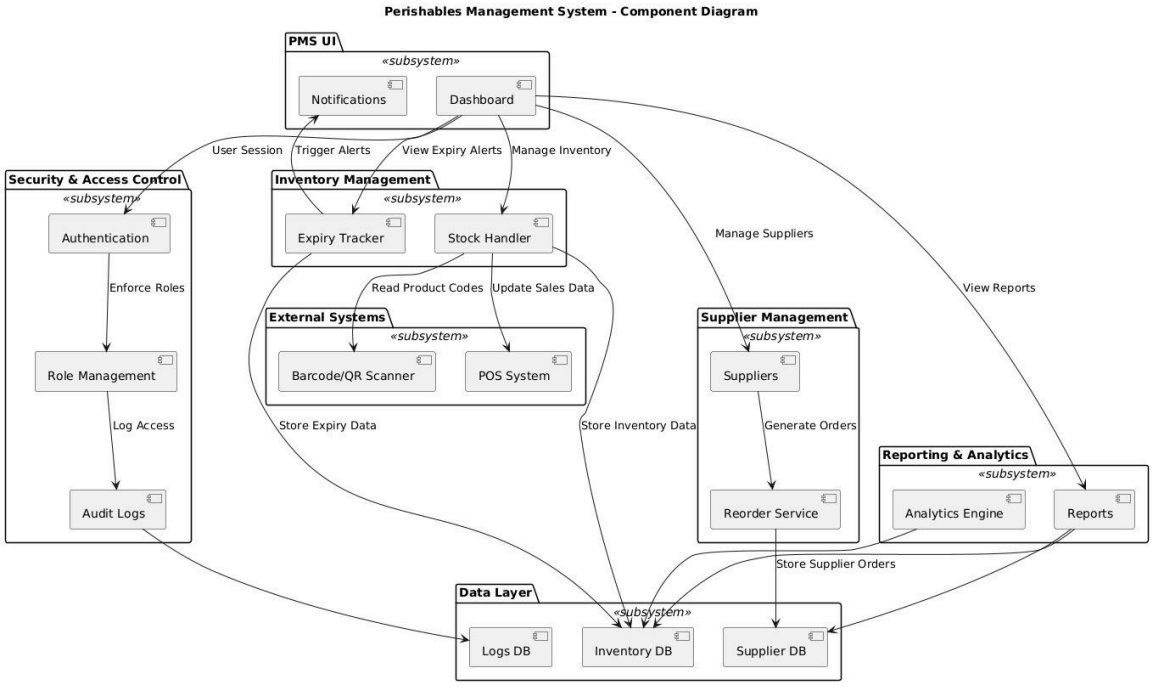
3.1 Goals & Constraints

- Goals: Secure, reliable, scalable, easy to use, real-time inventory tracking, regulatory compliance.
- Constraints: Real-time data synchronization, multi-store support, food safety log compliance.

3.2 Stakeholders & Concerns

Store Managers: Accuracy, real-time alerts, reporting
Staff: Usability, efficiency
Suppliers: Order accuracy, timely notifications
Developers: Modularity, maintainability
Administrators: Security, availability

3.3 Component (UML) Diagram



3.4 Component Descriptions

- PMS UI: Provides web and mobile interfaces for staff and managers.
- Inventory Service: Handles stock addition, expiry date management, sales integration.
- Expiry & Alerts Service: Tracks expiry dates, sends notifications, applies automatic discounts.
- Supplier Management Service: Stores supplier details, manages purchase requests.
- Reporting & Analytics Service: Generates wastage, sales, shelf life, and performance reports.
- Auth & Access Control Service: Provides authentication, authorization, role-based access.

- POS Integration API: Connects to POS systems for sales and stock updates.
- Database: Stores inventory, supplier, transaction, and audit log data.

3.5 Chosen Architecture Pattern and Rationale

A layered architecture is chosen for clear separation of concerns, maintainability, and scalability. The layers include Presentation, Business Logic, Integration, and Data Persistence.

3.6 Quality Attribute Scenarios

- Performance: 95% of inventory updates processed in under 2 seconds.
- Availability: System uptime of 99.9%.
- Security: All sensitive data encrypted using AES-256, TLS 1.2+ for communication.
- Scalability: Support 10,000+ SKUs across multiple stores.
- Usability: ≥85% positive feedback in user testing.

3.7 Technology Stack & Data Stores

Frontend: React.js
Backend: Node.js
Database: Firebase / MongoDB
Integration: REST APIs (POS, supplier)
Security: TLS 1.2+, AES-256 encryption
Cloud Hosting: AWS

3.8 Security & Compliance Considerations

- TLS 1.2+ for all communication
- AES-256 encryption at rest
- Role-based access control
- Tamper-proof audit logs
- Food safety regulation compliance

3.9 Architecture Decisions (ADR)

ADR-001: Chose layered architecture for clarity and maintainability.
ADR-002: TLS 1.2+ for secure communication.
ADR-003: AES-256 for database encryption at rest.

3.10 Risks & Mitigations

Risk: Network downtime → Mitigation: Redundant connectivity
Risk: Data inconsistency in multi-store sync → Mitigation: Conflict resolution policies
Risk: Unauthorized access → Mitigation: Role-based access, regular audits

3.11 Traceability to Requirements

PMS-F-001 (Add stock with expiry) → Inventory Service
PMS-F-004 (Expiry alerts) → Expiry & Alerts Service
PMS-F-007 (Reorder suggestions) → Supplier Management Service

PMS-F-010 (Wastage reports) → Reporting & Analytics Service
PMS-F-013 (Role-based access) → Auth & Access Control Service
PMS-NF-001 (Response time ≤2s) → Backend Services
PMS-SR-001 (TLS enforced) → Security Layer

3.12 Security Architecture

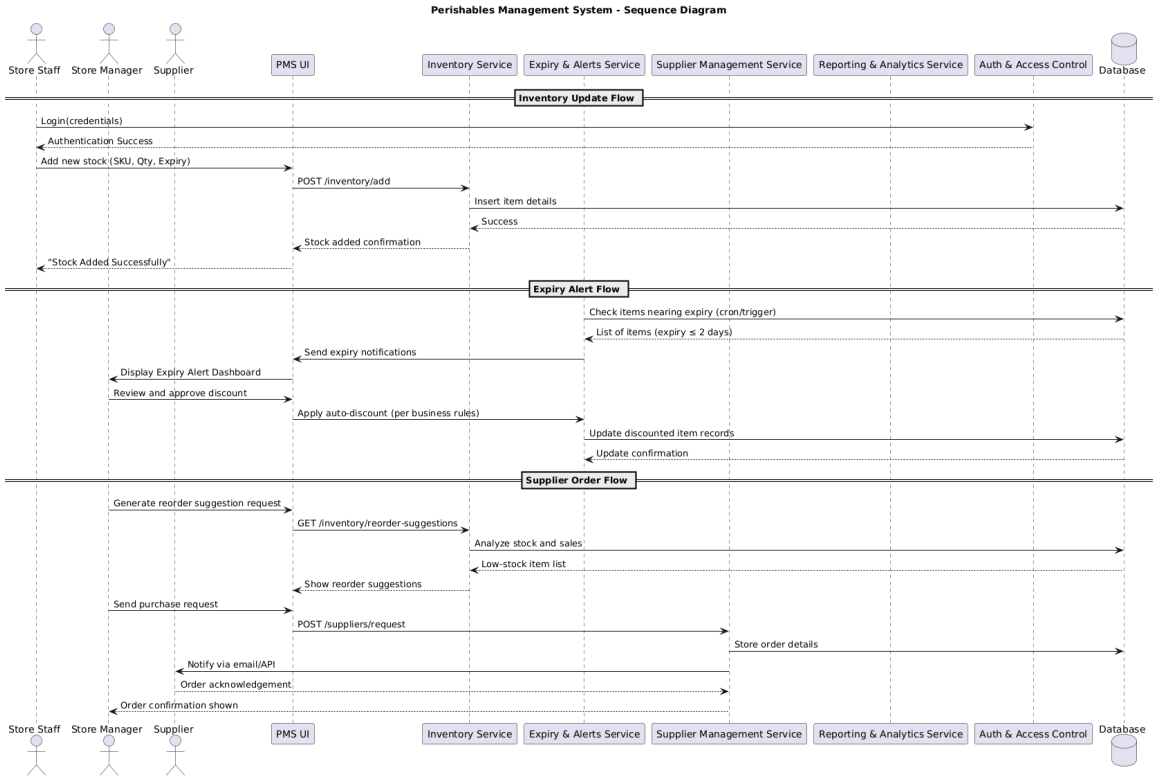
Threat Modeling (STRIDE):
- Spoofing → Strong authentication
- Tampering → Database encryption, secure APIs
- Repudiation → Immutable audit logs
- Information Disclosure → TLS & AES encryption
- Denial of Service → Rate limiting, scaling policies
- Elevation of Privilege → Role-based access control

4. Design

4.1 Design Overview

PMS is designed with layered modules to separate concerns across presentation, business logic, integration, and data persistence.

4.2 UML Sequence Diagrams



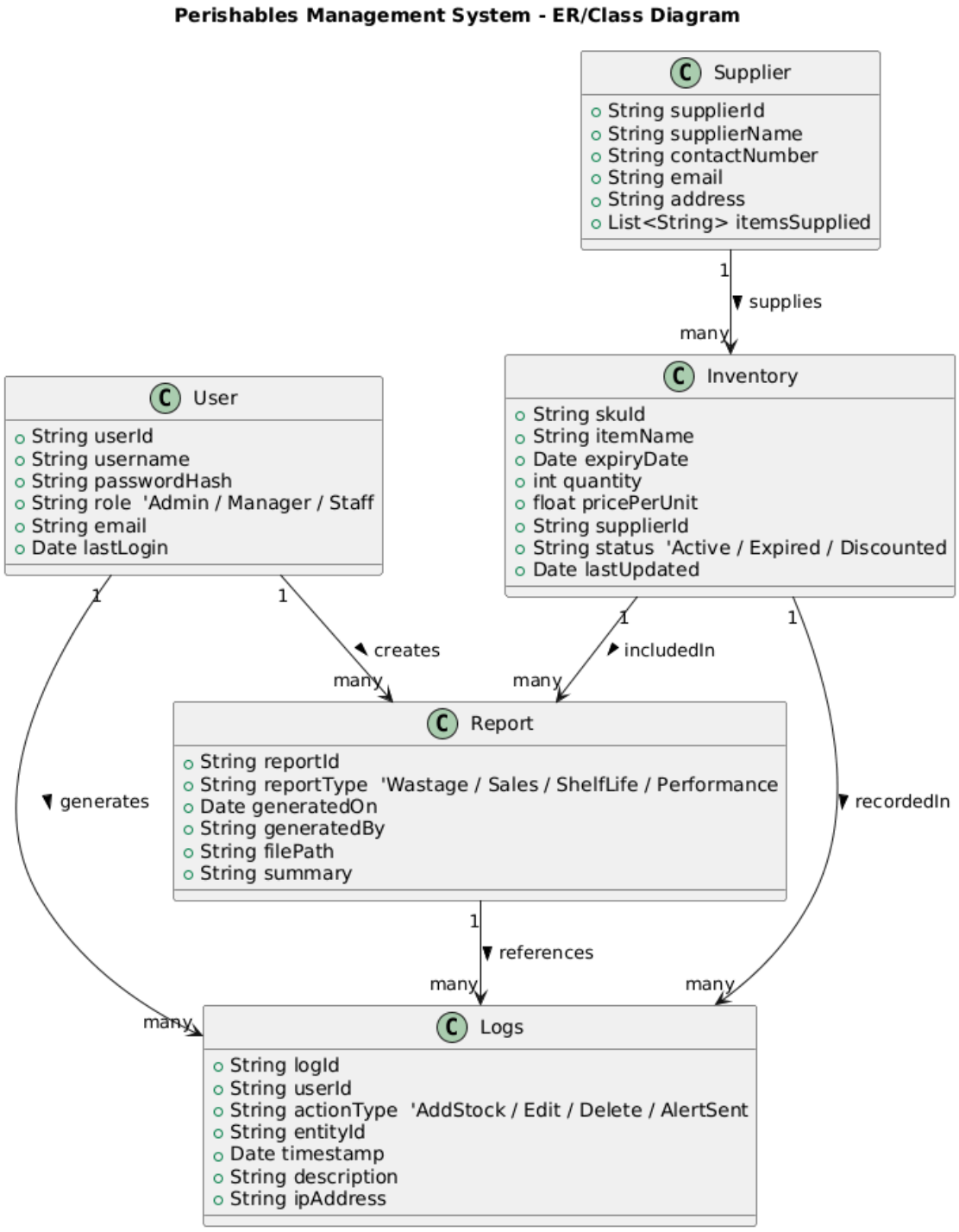
4.3 API Design

Endpoint: /inventory/add
Method: POST
Request: {skuId, name, expiryDate, quantity}
Response: {status, message}

Endpoint: /alerts/notify
Method: GET
Response: {skuId, expiryDate, daysRemaining, alertStatus}

Endpoint: /suppliers/request
Method: POST
Request: {supplierId, items[]}
Response: {status, orderId}

4.4 Data Model & Class Diagrams



4.5 Error Handling, Logging & Monitoring

- Consistent error codes across APIs
- No sensitive data in logs

- Monitoring of expiry alerts, stock updates, and uptime

4.6 UX Design

Web dashboard with accessible fonts, intuitive layout, expiry notifications, and analytics dashboards.

4.7 Non-functional Design Considerations

- Caching for faster dashboard queries
- Event-driven updates for inventory sync
- Concurrency management for stock operations

4.8 Testability & Hooks

- Unit test hooks for Inventory and Alerts services
- Integration testing with POS APIs
- Simulation of expiry alerts

4.9 Open Issues & Next Steps

Future enhancements: AI-based demand forecasting, image recognition for damaged items, IoT sensor integration.

5. Appendices

5.1 Glossary: PMS, SKU, POS, FIFO
5.2 References: IEEE 42010, OWASP, NIST SP 800-160
5.3 Tools: PlantUML, draw.io, Swagger, Firebase Console