Software Requirements Specification

Version 0.1

September 24, 2025

**Vending Machine Control System**

Bruno Gil

Aspire Systems

Latin America

## **Vending Machine Control System**

### **Document Information**

* *Document Version:* 0.1
* *Creation Date:* 24-09-2025
* *Last Modified:* 26-09-2025
* *Project Duration:* 17 working days
* *Total Effort:* 102 hours (9 hours/day)
* *Prepared By:* Bruno Gil
* *Reviewed By:* [TO BE FILLED]
* *Approved By:* [TO BE FILLED]

[Vending Machine Control System 1](#_Toc301423411)

[1. Introduction 3](#_Toc48648671)

[2. Overall Description 4](#_Toc1158728046)

[3. System Features 6](#_Toc111994165)

[4. External Interface Requirements 14](#_Toc632637094)

[5. System Architecture 16](#_Toc498819574)

[6. Non-Functional Requirements 18](#_Toc1777368375)

[7. Database Design 20](#_Toc796832075)

[8. Security Requirements 23](#_Toc183221579)

[9. Quality Assurance 25](#_Toc2059457655)

[10. Design Patterns & Principles 27](#_Toc380442701)

[11. System Design Diagrams 32](#_Toc1160097261)

[12. API Design Standards 33](#_Toc1099073381)

[13. Error Handling, Logging & Monitoring 38](#_Toc569769961)

[14. Requirements Traceability Matrix 43](#_Toc1862065830)

[15. Database Design Enhancements 45](#_Toc427214591)

[16. Security Enhancements 49](#_Toc1193618327)

[17. Deployment Considerations 50](#_Toc684131227)

## **1. Introduction**

### **1.1 Purpose**

This Software Requirements Specification (SRS) document describes the functional and non-functional requirements for a simplified microservices-based vending machine control system. The system is designed as a learning project to practice microservices architecture using Java 17, Spring Boot, and related technologies.

### **1.2 Scope**

The vending machine control system provides:

* *Inventory Management*: Real-time stock tracking and management
* *Transaction Processing*: Purchase orchestration and payment handling
* *Item Dispensing*: Simulated mechanical dispensing operations
* *Administrative Functions*: User management and system monitoring
* *Event-Driven Architecture*: Kafka-based inter-service communication

### **1.3 Intended Audience**

* *Development Team*: Bruno Gil only
* *System Administrators*: Bruno Gil only
* *Stakeholders*: Suresh Rajendran
* *Future Maintainers*: Bruno Gil

### **1.4 Product Overview**

A microservices-based system consisting of 8 core services communicating through Apache Kafka, with centralized configuration management, service discovery, and API gateway for security and routing.

## **2. Overall Description**

### **2.1 Product Perspective**

The system operates as a distributed microservices application in a local development environment, simulating a real-world vending machine control system without actual hardware integration.

### **2.2 Product Functions**

*Primary Functions:*

* Real-time inventory management with automatic stock updates
* Multi-payment method support (cash and card simulation)
* Event-driven transaction orchestration
* Administrative user management with role-based access
* System-wide notification and alert handling
* Comprehensive monitoring and observability

### **2.3 User Characteristics**

*Administrator Users:*

* *SUPER\_ADMIN*: Full system access and user management
* *ADMIN*: Operational management capabilities
* Technical knowledge: Intermediate to advanced
* Primary interface: REST APIs via Postman or similar tools

*End Customers:*

* Interact through public APIs (simulated customer interface)
* No authentication required for purchase operations
* Primary operations: Product selection and purchase

### **2.4 Operating Environment**

*Development Environment:*

* *Operating System*: Cross-platform (Windows, macOS, Linux)
* *Java Runtime*: Java 17 or higher
* *Database*: MySQL 8.0+
* *Message Broker*: Apache Kafka with Zookeeper
* *Memory Requirements*: 6GB RAM minimum
* *Storage*: 2GB free disk space

### **2.5 Design and Implementation Constraints**

* *Local Development Only*: No production deployment considerations
* *Simulated Hardware*: No real vending machine integration
* *Single Instance Deployment*: No clustering or high availability
* *Manual Database Schema*: No automated migration tools
* *HTTP Communication*: No HTTPS/SSL requirements for local development

### **2.6 Assumptions and Dependencies**

*Assumptions:*

* Stable local development environment
* MySQL and Kafka services available and running
* Network connectivity between services via localhost
* Sufficient system resources for concurrent service operation

*Dependencies:*

* Java 17 SDK
* Maven 3.8+ for build management
* MySQL server for data persistence
* Apache Kafka and Zookeeper for messaging

## **3. System Features**

### **3.1 Inventory Management Service**

#### **3.1.1 Description**

Manages product catalog and stock levels with real-time updates through event-driven architecture.

#### **3.1.2 Functional Requirements**

*REQ-INV-001*: Product Management

* *Priority*: High
* *Description*: System shall provide CRUD operations for product management
* *Inputs*: Product details (name, description, price)
* *Outputs*: Product information and confirmation
* *Acceptance Criteria*: Products can be created, read, updated, and deleted via REST APIs

*REQ-INV-002*: Stock Tracking

* *Priority*: High
* *Description*: System shall track real-time stock quantities for all products
* *Inputs*: Stock updates from dispensing operations
* *Outputs*: Current stock levels and availability status
* *Acceptance Criteria*: Stock levels accurate within 1 second of any transaction

*REQ-INV-003*: Low Stock Alerts

* *Priority*: Medium
* *Description*: System shall generate alerts when stock falls below configurable threshold
* *Inputs*: Current stock level and minimum threshold
* *Outputs*: Notification event to notification service
* *Acceptance Criteria*: Alerts triggered when quantity ≤ minimum threshold

*REQ-INV-004*: Event-Driven Updates

* *Priority*: High
* *Description*: System shall publish inventory events to Kafka topics
* *Inputs*: Stock level changes and product modifications
* *Outputs*: Kafka events for downstream consumption
* *Acceptance Criteria*: All inventory changes published as events within 100ms

#### **3.1.3 API Endpoints**

GET /api/inventory/products - Retrieve all products (Public) GET /api/inventory/availability/{productId} - Check product availability (Public) POST /api/admin/inventory/products - Create new product (Admin) PUT /api/admin/inventory/stock/{productId} - Update stock level (Admin) GET /api/admin/inventory/reports - Generate inventory reports (Admin)

### **3.2 Payment Processing Service**

#### **3.2.1 Description**

Handles payment processing simulation for cash and card transactions with comprehensive logging.

#### **3.2.2 Functional Requirements**

*REQ-PAY-001*: Payment Method Support

* *Priority*: High
* *Description*: System shall support cash and card payment simulation
* *Inputs*: Payment amount and method selection
* *Outputs*: Payment confirmation or failure notification
* *Acceptance Criteria*: Support for CASH, CREDIT\_CARD, and DEBIT\_CARD methods

*REQ-PAY-002*: Payment Simulation

* *Priority*: High
* *Description*: System shall simulate payment processing with configurable success rates
* *Inputs*: Payment transaction details
* *Outputs*: Payment status (PENDING, COMPLETED, FAILED)
* *Acceptance Criteria*: Configurable success/failure rates for testing scenarios

*REQ-PAY-003*: Transaction Logging

* *Priority*: Medium
* *Description*: System shall log all payment transactions for audit purposes
* *Inputs*: Payment transaction data
* *Outputs*: Persistent transaction records
* *Acceptance Criteria*: Complete audit trail maintained in database

*REQ-PAY-004*: Event Publishing

* *Priority*: High
* *Description*: System shall publish payment events to Kafka topics
* *Inputs*: Payment status changes
* *Outputs*: Payment events for transaction coordination
* *Acceptance Criteria*: Payment events published within 100ms of status change

#### **3.2.3 API Endpoints**

POST /api/payment/process - Process payment transaction (Public) GET /api/admin/payment/transactions - Retrieve payment history (Admin) GET /api/admin/payment/statistics - Payment processing statistics (Admin)

### **3.3 Transaction Orchestration Service**

#### **3.3.1 Description**

Coordinates the complete purchase flow from inventory check through payment processing to item dispensing.

#### **3.3.2 Functional Requirements**

*REQ-TXN-001*: Purchase Orchestration

* *Priority*: High
* *Description*: System shall orchestrate complete purchase transactions
* *Inputs*: Product selection, payment method, customer session
* *Outputs*: Transaction status and completion confirmation
* *Acceptance Criteria*: Complete purchase flow coordination within 5 seconds

*REQ-TXN-002*: Inventory Validation

* *Priority*: High
* *Description*: System shall validate product availability before processing payment
* *Inputs*: Product ID and requested quantity
* *Outputs*: Availability confirmation or rejection
* *Acceptance Criteria*: Real-time inventory check with accurate availability status

*REQ-TXN-003*: Event-Driven Coordination

* *Priority*: High
* *Description*: System shall coordinate services through Kafka event consumption
* *Inputs*: Payment and dispensing events
* *Outputs*: Transaction status updates
* *Acceptance Criteria*: Proper transaction state management based on service events

*REQ-TXN-004*: Failure Handling

* *Priority*: Low
* *Description*: System shall implement basic compensation for failed transactions
* *Inputs*: Failure notifications from payment or dispensing services
* *Outputs*: Transaction rollback and user notification
* *Acceptance Criteria*: Failed transactions properly handled with user feedback

#### **3.3.3 API Endpoints**

POST /api/transaction/purchase - Initiate purchase transaction (Public) GET /api/admin/transaction/history - Retrieve transaction history (Admin) GET /api/admin/transaction/statistics - Transaction processing statistics (Admin)

### **3.4 Dispensing Simulation Service**

#### **3.4.1 Description**

Simulates mechanical dispensing operations with configurable success/failure scenarios for testing.

#### **3.4.2 Functional Requirements**

*REQ-DISP-001*: Dispensing Simulation

* *Priority*: High
* *Description*: System shall simulate item dispensing with configurable parameters
* *Inputs*: Dispensing request from transaction service
* *Outputs*: Dispensing success or failure notification
* *Acceptance Criteria*: Configurable success/failure rates for different scenarios

*REQ-DISP-002*: Hardware Status Simulation

* *Priority*: Medium
* *Description*: System shall simulate hardware operational status and failures
* *Inputs*: Dispensing operation requests
* *Outputs*: Hardware status reports and failure notifications
* *Acceptance Criteria*: Realistic hardware failure simulation for testing

*REQ-DISP-003*: Event Processing

* *Priority*: High
* *Description*: System shall consume dispensing requests and publish results via Kafka
* *Inputs*: Kafka events from transaction service
* *Outputs*: Dispensing result events and inventory updates
* *Acceptance Criteria*: Event processing within 100ms with proper result publication

#### **3.4.3 API Endpoints**

GET /api/admin/dispensing/status - Hardware status information (Admin) GET /api/admin/dispensing/history - Dispensing operation history (Admin) PUT /api/admin/dispensing/configuration - Update simulation parameters (Admin)

### **3.5 Notification Management Service**

#### **3.5.1 Description**

Centralizes system notifications and alerts from all services for comprehensive monitoring.

#### **3.5.2 Functional Requirements**

*REQ-NOT-001*: Event Aggregation

* *Priority*: Medium
* *Description*: System shall consume notification events from all services
* *Inputs*: Kafka events from inventory, payment, transaction, and dispensing services
* *Outputs*: Stored notifications for administrative review
* *Acceptance Criteria*: All notification events properly consumed and stored

*REQ-NOT-002*: Alert Classification

* *Priority*: Medium
* *Description*: System shall classify notifications by type and severity
* *Inputs*: Notification event data
* *Outputs*: Categorized notification records
* *Acceptance Criteria*: Proper notification categorization (LOW\_STOCK, TRANSACTION\_FAILED, etc.)

*REQ-NOT-003*: Administrative Interface

* *Priority*: Medium
* *Description*: System shall provide administrative access to notification history
* *Inputs*: Administrative requests for notification data
* *Outputs*: Filtered notification lists and statistics
* *Acceptance Criteria*: Comprehensive notification management interface

#### **3.5.3 API Endpoints**

GET /api/admin/notifications - Retrieve notification history (Admin) GET /api/admin/notifications/statistics - Notification statistics (Admin) PUT /api/admin/notifications/{id}/acknowledge - Acknowledge notification (Admin)

### **3.6 Authentication and Authorization**

#### **3.6.1 Description**

Provides JWT-based authentication and role-based authorization at the API Gateway level.

#### **3.6.2 Functional Requirements**

*REQ-AUTH-001*: JWT Authentication

* *Priority*: High
* *Description*: System shall provide JWT-based authentication for administrative operations
* *Inputs*: Username and password credentials
* *Outputs*: JWT token with 8-hour expiry
* *Acceptance Criteria*: Secure token generation and validation

*REQ-AUTH-002*: Role-Based Authorization

* *Priority*: High
* *Description*: System shall enforce role-based access control
* *Inputs*: JWT token with role information
* *Outputs*: Access granted or denied based on role permissions
* *Acceptance Criteria*: SUPER\_ADMIN and ADMIN roles properly enforced

*REQ-AUTH-003*: User Management

* *Priority*: Medium
* *Description*: System shall provide user management capabilities
* *Inputs*: User creation/modification requests
* *Outputs*: User account status and confirmation
* *Acceptance Criteria*: Complete user lifecycle management

#### **3.6.3 API Endpoints**

POST /api/auth/login - User authentication (Public) POST /api/admin/users - Create new user (Super Admin) GET /api/admin/users - List all users (Admin) PUT /api/admin/users/{id} - Update user information (Super Admin) DELETE /api/admin/users/{id} - Delete user (Super Admin)

## **4. External Interface Requirements**

### **4.1 User Interfaces**

*Administrative Interface:*

* RESTful API endpoints accessible via HTTP clients (Postman, curl)
* JSON-based request/response format
* Standard HTTP status codes for operation results
* Correlation ID headers for request tracing

*Customer Interface:*

* Public API endpoints for product browsing and purchase initiation
* No authentication required for basic operations
* Simplified request/response format for ease of use

### **4.2 Hardware Interfaces**

*Simulated Hardware:*

* No actual hardware interfaces required
* Dispensing operations simulated through software
* Hardware status simulation for testing failure scenarios
* Configurable parameters for different hardware behaviors

### **4.3 Software Interfaces**

*Database Interfaces:*

* MySQL database connections via JDBC
* HikariCP connection pooling
* Service-specific database schemas
* Standard SQL operations for data persistence

*Message Broker Interfaces:*

* Apache Kafka for event publishing and consumption
* JSON-serialized message format
* Topic-based message routing
* At-least-once delivery semantics

*External Service Interfaces:*

* Service discovery via Eureka Server
* Configuration retrieval from Config Server
* Health check endpoints via Spring Boot Actuator
* RESTful inter-service communication where needed

### **4.4 Communication Interfaces**

*Network Protocols:*

* HTTP/1.1 for REST API communication
* TCP for Kafka message transport
* Standard localhost networking for local development
* Port-based service identification

*Data Formats:*

* JSON for REST API payloads
* JSON for Kafka message content
* UTF-8 character encoding throughout
* ISO 8601 date/time format standardization

## **5. System Architecture**

### **5.1 Architectural Overview**

The system follows a microservices architecture pattern with the following key components:

*Infrastructure Layer:*

* Config Server (Port 8888): Centralized configuration management
* Eureka Server (Port 8761): Service discovery and registration
* API Gateway (Port 8080): Authentication and routing
* Apache Kafka (Port 9092): Event streaming platform
* MySQL Database (Port 3306): Data persistence layer

*Business Services Layer:*

* Inventory Service (Port 8081): Product and stock management
* Payment Service (Port 8082): Payment processing simulation
* Transaction Service (Port 8083): Purchase orchestration
* Dispensing Service (Port 8084): Item dispensing simulation
* Notification Service (Port 8085): System alert management

### **5.2 Service Interaction Patterns**

#### **5.2.1 Synchronous Communication**

Client → API Gateway → Business Services

* JWT authentication at gateway level
* Service discovery via Eureka
* Direct HTTP calls for immediate responses
* Timeout and retry mechanisms

#### **5.2.2 Asynchronous Communication**

Service A → Kafka Topic → Service B

* Event-driven architecture for loose coupling
* Publish-subscribe messaging pattern
* Eventual consistency through event propagation
* Idempotent event processing

### **5.3 Kafka Topic Architecture**

#### **5.3.1 Topic Configuration**

transaction-events: partitions: 3 replication-factor: 1 events: [transaction.created, transaction.completed, transaction.failed]

payment-events: partitions: 3 replication-factor: 1 events: [payment.initiated, payment.completed, payment.failed]

inventory-events: partitions: 3 replication-factor: 1 events: [stock.updated, stock.low, product.added]

dispensing-events: partitions: 3 replication-factor: 1 events: [dispensing.requested, dispensing.completed, dispensing.failed]

notification-events: partitions: 3 replication-factor: 1 events: [notification.created, notification.sent]

#### **5.3.2 Event Schema Standards**

json { "eventId": "uuid", "eventType": "string", "timestamp": "ISO 8601 datetime", "source": "service name", "correlationId": "uuid", "payload": "event-specific data" }

### **5.4 Data Flow Architecture**

#### **5.4.1 Purchase Transaction Flow**

1. Customer Request → API Gateway → Transaction Service
2. Transaction Service → Inventory Service (availability check)
3. Transaction Service → Payment Service (payment processing)
4. Payment Service → Kafka (payment.completed event)
5. Transaction Service → Dispensing Service (dispensing request)
6. Dispensing Service → Kafka (dispensing.completed event)
7. Inventory Service → Kafka (stock.updated event)
8. Transaction Service → Customer Response

#### **5.4.2 Administrative Operation Flow**

1. Admin Request → API Gateway (JWT validation)
2. API Gateway → Target Service (with user context)
3. Service → Database (data operation)
4. Service → Kafka (event publication if applicable)
5. Service → Admin Response

## **6. Non-Functional Requirements**

### **6.1 Performance Requirements**

#### **6.1.1 Response Time Requirements**

* *API Response Time*: All REST API endpoints shall respond within 2000ms under normal load
* *Event Processing Time*: Kafka events shall be processed within 100ms of receipt
* *Database Query Time*: Database operations shall complete within 500ms
* *Service Startup Time*: All services shall start and be ready within 120 seconds

#### **6.1.2 Throughput Requirements**

* *Concurrent Users*: System shall support 10-50 concurrent transactions
* *Event Throughput*: Kafka shall handle 1000+ events per minute
* *Database Connections*: Connection pool shall support 20+ concurrent connections per service
* *API Request Rate*: System shall handle 100+ API requests per minute

#### **6.1.3 Resource Requirements**

* *Memory Usage*: Total system memory consumption shall not exceed 6GB
* *CPU Utilization*: Normal operations shall not exceed 70% CPU utilization
* *Disk Space*: System shall require maximum 2GB disk space
* *Network Bandwidth*: Local network communication sufficient for localhost deployment

### **6.2 Scalability Requirements**

* *Service Instances*: Single instance deployment for local development
* *Database Scaling*: Single MySQL server with service-specific databases
* *Kafka Partitions*: Configurable partition counts for topic scaling
* *Connection Pooling*: Scalable connection pool configurations

### **6.3 Reliability Requirements**

* *Service Availability*: Services shall maintain 95%+ uptime during development
* *Data Consistency*: Eventually consistent data through event-driven updates
* *Error Recovery*: Automatic retry mechanisms for transient failures
* *Event Delivery*: At-least-once delivery guarantee for Kafka events

### **6.4 Usability Requirements**

* *API Design*: RESTful APIs following standard conventions
* *Error Messages*: Clear, descriptive error messages with correlation IDs
* *Documentation*: Comprehensive API documentation with examples
* *Testing Tools*: Complete Postman collection for all endpoints

### **6.5 Maintainability Requirements**

* *Code Structure*: Clean architecture with separation of concerns
* *Configuration Management*: Externalized configuration via Config Server
* *Logging Standards*: Structured logging with correlation ID tracking
* *Monitoring Capabilities*: Health checks and metrics via Actuator endpoints

## **7. Database Design**

### **7.1 Database Architecture**

* *Database Server*: Single MySQL 8.0+ instance
* *Schema Strategy*: Separate database per service (database-per-service pattern)
* *Connection Management*: HikariCP connection pooling for each service
* *Transaction Management*: Service-level transaction boundaries

### **7.2 Service Database Schemas**

#### **7.2.1 Authentication Service (vending\_auth)**

sql -- Admin users table CREATE TABLE admin\_users ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, username VARCHAR(255) UNIQUE NOT NULL, password\_hash VARCHAR(255) NOT NULL, role ENUM('SUPER\_ADMIN', 'ADMIN') NOT NULL, active BOOLEAN DEFAULT TRUE, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP );

-- Create indexes CREATE INDEX idx\_username ON admin\_users(username); CREATE INDEX idx\_active ON admin\_users(active);

#### **7.2.2 Inventory Service (vending\_inventory)**

sql -- Products table CREATE TABLE products ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, name VARCHAR(255) NOT NULL, description TEXT, price DECIMAL(10,2) NOT NULL, active BOOLEAN DEFAULT TRUE, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP );

-- Stock table CREATE TABLE stock ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, product\_id BIGINT NOT NULL, quantity INT NOT NULL DEFAULT 0, min\_threshold INT NOT NULL DEFAULT 5, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP, FOREIGN KEY (product\_id) REFERENCES products(id), UNIQUE KEY unique\_product\_stock (product\_id) );

-- Create indexes CREATE INDEX idx\_product\_active ON products(active); CREATE INDEX idx\_stock\_quantity ON stock(quantity);

#### **7.2.3 Payment Service (vending\_payment)**

sql -- Payment transactions table CREATE TABLE payment\_transactions ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, transaction\_id VARCHAR(255) UNIQUE NOT NULL, amount DECIMAL(10,2) NOT NULL, payment\_method ENUM('CASH', 'CREDIT\_CARD', 'DEBIT\_CARD') NOT NULL, status ENUM('PENDING', 'COMPLETED', 'FAILED') NOT NULL, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP );

-- Create indexes CREATE INDEX idx\_transaction\_id ON payment\_transactions(transaction\_id); CREATE INDEX idx\_payment\_status ON payment\_transactions(status); CREATE INDEX idx\_payment\_created ON payment\_transactions(created\_at);

#### **7.2.4 Transaction Service (vending\_transaction)**

sql -- Transactions table CREATE TABLE transactions ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, customer\_session VARCHAR(255), total\_amount DECIMAL(10,2) NOT NULL, status ENUM('CREATED', 'PAYMENT\_PROCESSING', 'DISPENSING', 'COMPLETED', 'FAILED') NOT NULL, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP );

-- Transaction items table CREATE TABLE transaction\_items ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, transaction\_id BIGINT NOT NULL, product\_id BIGINT NOT NULL, quantity INT NOT NULL, unit\_price DECIMAL(10,2) NOT NULL, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, FOREIGN KEY (transaction\_id) REFERENCES transactions(id) );

-- Create indexes CREATE INDEX idx\_transaction\_status ON transactions(status); CREATE INDEX idx\_transaction\_created ON transactions(created\_at); CREATE INDEX idx\_transaction\_items\_txn ON transaction\_items(transaction\_id);

#### **7.2.5 Dispensing Service (vending\_dispensing)**

sql -- Dispensing operations table CREATE TABLE dispensing\_operations ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, transaction\_id BIGINT NOT NULL, product\_id BIGINT NOT NULL, quantity INT NOT NULL, status ENUM('REQUESTED', 'IN\_PROGRESS', 'COMPLETED', 'FAILED') NOT NULL, failure\_reason VARCHAR(500), created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP );

-- Hardware status table CREATE TABLE hardware\_status ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, component\_name VARCHAR(255) NOT NULL, status ENUM('OPERATIONAL', 'MAINTENANCE', 'FAILED') NOT NULL, last\_check TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP );

-- Create indexes CREATE INDEX idx\_dispensing\_status ON dispensing\_operations(status); CREATE INDEX idx\_dispensing\_transaction ON dispensing\_operations(transaction\_id);

#### **7.2.6 Notification Service (vending\_notification)**

sql -- Notifications table CREATE TABLE notifications ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, type ENUM('LOW\_STOCK', 'TRANSACTION\_FAILED', 'HARDWARE\_FAILURE', 'PAYMENT\_FAILED') NOT NULL, title VARCHAR(255) NOT NULL, message TEXT NOT NULL, severity ENUM('INFO', 'WARNING', 'ERROR', 'CRITICAL') NOT NULL, source\_service VARCHAR(255) NOT NULL, acknowledged BOOLEAN DEFAULT FALSE, created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, acknowledged\_at TIMESTAMP NULL );

-- Create indexes CREATE INDEX idx\_notification\_type ON notifications(type); CREATE INDEX idx\_notification\_severity ON notifications(severity); CREATE INDEX idx\_notification\_acknowledged ON notifications(acknowledged); CREATE INDEX idx\_notification\_created ON notifications(created\_at);

### **7.3 Data Integrity Constraints**

* *Primary Keys*: Auto-increment BIGINT for all tables
* *Foreign Keys*: Proper referential integrity where applicable
* *Unique Constraints*: Enforce data uniqueness where required
* *Check Constraints*: Validate enum values and business rules
* *Index Strategy*: Optimize for common query patterns

### **7.4 Database Configuration**

yaml Database Connection Settings:

* Connection Pool Size: 10-20 connections per service
* Connection Timeout: 30 seconds
* Idle Timeout: 600 seconds
* Max Lifetime: 1800 seconds
* Auto Commit: false (explicit transaction management)

## **8. Security Requirements**

### **8.1 Authentication Requirements**

#### **8.1.1 JWT Authentication**

* *Token Format*: JSON Web Token (JWT) with HS256 signing
* *Token Expiry*: 8 hours for administrative sessions
* *Token Storage*: Client-side storage responsibility
* *Token Validation*: At API Gateway level for all admin endpoints
* *Secret Management*: Configurable JWT signing secret

#### **8.1.2 User Credential Management**

* *Password Hashing*: BCrypt with configurable rounds
* *Username Requirements*: Unique, alphanumeric characters
* *Password Policy*: [TO BE DEFINED BASED ON REQUIREMENTS]
* *Account Lockout*: [TO BE DEFINED BASED ON REQUIREMENTS]

### **8.2 Authorization Requirements**

#### **8.2.1 Role-Based Access Control**

yaml SUPER\_ADMIN:

* All system operations
* User management (create, update, delete users)
* System configuration management
* Access to all administrative endpoints

ADMIN:

* Inventory management operations
* Transaction and payment monitoring
* Notification management
* System status monitoring
* Limited user information access

#### **8.2.2 Endpoint Authorization**

yaml Public Endpoints (No Authentication):

* GET /api/inventory/products
* GET /api/inventory/availability/{productId}
* POST /api/transaction/purchase
* POST /api/payment/process
* POST /api/auth/login

Admin Endpoints (Authentication Required):

* All /api/admin/\* endpoints
* User management operations
* Administrative reporting
* System configuration

### **8.3 Data Security Requirements**

#### **8.3.1 Data Protection**

* *Sensitive Data*: User passwords hashed with BCrypt
* *Data Transmission*: HTTP for local development (HTTPS for production)
* *Database Security*: Standard MySQL user authentication
* *Audit Logging*: Complete logging of administrative operations

#### **8.3.2 Input Validation**

* *SQL Injection Prevention*: Prepared statements and parameterized queries
* *Input Sanitization*: Validation and sanitization of all user inputs
* *Request Size Limits*: Configurable limits on request payload sizes
* *Rate Limiting*: Basic request rate limiting at gateway level

### **8.4 API Security**

#### **8.4.1 Request Security**

* *CORS Configuration*: Configurable cross-origin request policies
* *Security Headers*: Standard security headers (X-Frame-Options, etc.)
* *Request Validation*: Comprehensive input validation at API boundaries
* *Error Handling*: Secure error messages without sensitive information exposure

#### **8.4.2 Session Management**

* *Stateless Design*: JWT-based stateless authentication
* *Token Refresh*: [TO BE DEFINED - OUT OF CURRENT SCOPE]
* *Session Timeout*: Token expiry-based session termination
* *Concurrent Sessions*: [TO BE DEFINED BASED ON REQUIREMENTS]

## **9. Quality Assurance**

### **9.1 Testing Requirements**

#### **9.1.1 Unit Testing**

* *Coverage Target*: 80%+ coverage for business logic
* *Testing Framework*: JUnit 5 with Mockito for mocking
* *Test Categories*: Service layer, repository layer, utility classes
* *Continuous Testing*: Automated test execution on build

#### **9.1.2 Integration Testing**

* *Database Integration*: Tests with embedded or containerized databases
* *Kafka Integration*: Tests with embedded Kafka for event processing
* *Service Integration*: Tests for inter-service communication
* *API Integration*: Complete endpoint testing with authentication

#### **9.1.3 System Testing**

* *End-to-End Testing*: Complete transaction flow validation
* *Load Testing*: Performance testing under expected load
* *Failure Testing*: Error handling and recovery testing
* *Security Testing*: Authentication and authorization validation

### **9.2 Code Quality Standards**

#### **9.2.1 Coding Standards**

* *Java Standards*: Oracle Java coding conventions
* *Spring Standards*: Spring Boot best practices and patterns
* *Documentation*: Comprehensive JavaDoc for public APIs
* *Code Review*: [TO BE DEFINED BASED ON TEAM PROCESS]

#### **9.2.2 Static Analysis**

* *Code Analysis Tools*: [TO BE SELECTED - SonarQube, SpotBugs, etc.]
* *Quality Gates*: Automated quality checks on build
* *Dependency Scanning*: Security vulnerability scanning
* *License Compliance*: [TO BE DEFINED]

## **10. Design Patterns & Principles**

### **10.1 Architectural Patterns**

#### **10.1.1 Microservices Architecture**

**Pattern Description:** The system follows microservices architecture with service decomposition based on business capabilities. Each service owns its data and communicates through well-defined interfaces.

**Implementation:**

* **Service Boundaries:** Each service represents a bounded context (Inventory, Payment, Transaction, Dispensing, Notification)
* **Data Ownership:** Database-per-service pattern ensures loose coupling
* **Communication:** Synchronous (REST) for queries, asynchronous (Kafka) for state changes
* **Service Discovery:** Eureka Server enables dynamic service location

**Benefits:**

* Independent deployment and scaling
* Technology flexibility per service
* Fault isolation
* Team autonomy (single developer in this case)

#### **10.1.2 Event-Driven Architecture (EDA)**

**Pattern Description:** Services communicate through domain events published to Kafka topics, enabling loose coupling and eventual consistency.

**Implementation:**

* **Event Types:** Domain Events, Integration Events, Notification Events
* **Event Schema:** Standardized JSON format with correlation ID
* **Event Flow:** Producer → Kafka Topic → Consumer(s)
* **Idempotency:** Event handlers designed for at-least-once delivery

**Benefits:**

* Loose coupling between services
* Asynchronous processing for better performance
* Event replay capability for debugging
* Scalable message processing

#### **10.1.3 SAGA Pattern for Distributed Transactions**

**Pattern Description:** Orchestration-based SAGA pattern manages distributed transactions across services without requiring distributed locks.

**Implementation:**

* **Orchestrator:** Transaction Service coordinates the purchase flow
* **Saga Steps:**
  + Validate inventory availability
  + Process payment
  + Request dispensing
  + Update inventory
  + Complete transaction
* **Compensation Logic:** Basic rollback for payment failures (simplified for learning)
* **State Management:** Transaction status tracking (CREATED → PAYMENT\_PROCESSING → DISPENSING → COMPLETED/FAILED)

**Limitations (Acknowledged for Learning Project):**

* Simplified compensation logic
* No advanced retry mechanisms
* Manual rollback procedures for complex failures

#### **10.1.4 API Gateway Pattern**

**Pattern Description:** Single entry point for all client requests, providing authentication, routing, and cross-cutting concerns.

**Implementation:**

* **Authentication:** JWT validation before routing
* **Routing:** Dynamic routing based on service discovery
* **Request Enhancement:** Inject user context headers
* **Security:** CORS, rate limiting, security headers
* **Load Balancing:** Client-side load balancing via Eureka

### **10.2 Design Principles**

#### **10.2.1 SOLID Principles**

**Single Responsibility Principle (SRP):**

* Each service has a single business capability
* Classes and methods have focused responsibilities
* Example: PaymentService only handles payment processing, not inventory

**Open/Closed Principle (OCP):**

* Services extensible through configuration
* Payment methods extendable without modifying core logic
* Strategy pattern for dispensing simulation configurations

**Liskov Substitution Principle (LSP):**

* Interface-based service contracts
* Repository pattern allows database implementation swapping
* Event handlers implement common interfaces

**Interface Segregation Principle (ISP):**

* Focused service interfaces
* Clients depend only on required methods
* REST endpoints grouped by functionality

**Dependency Inversion Principle (DIP):**

* Services depend on abstractions (interfaces)
* Spring dependency injection for loose coupling
* Repository pattern abstracts data access

#### **10.2.2 Domain-Driven Design (DDD) Concepts**

**Bounded Contexts:**

* **Inventory Context:** Products, Stock, Availability
* **Payment Context:** Transactions, Payment Methods, Processing
* **Transaction Context:** Orders, Purchase Flow, Orchestration
* **Dispensing Context:** Hardware Operations, Dispensing Status
* **Notification Context:** Alerts, Notifications, Event Aggregation

**[DETAILED DDD IMPLEMENTATION SPECIFICATIONS: TO BE DEFINED]**

* Aggregate root definitions and boundaries
* Value objects identification
* Repository patterns per aggregate
* Domain service responsibilities

#### **10.2.3 Microservices Design Principles**

**Service Independence:**

* Each service can be developed, deployed, and scaled independently
* No direct database sharing between services
* Asynchronous communication for non-blocking operations

**Smart Endpoints, Dumb Pipes:**

* Business logic in services, not in middleware
* Kafka used as simple message transport
* No complex ESB-style transformations

**Decentralized Data Management:**

* Each service owns its data
* No distributed transactions
* Eventual consistency through events

**Design for Failure:**

* Graceful degradation when services unavailable
* Timeout configurations for external calls
* **[CIRCUIT BREAKER IMPLEMENTATION: TO BE DEFINED]**
* Comprehensive error logging with correlation IDs

**Infrastructure Automation:**

* Spring Boot Actuator for health checks
* Automated service registration with Eureka
* Centralized configuration management
* Container-ready design (future Docker deployment)

### **10.3 Design Patterns Implementation**

#### **10.3.1 Creational Patterns**

**[DETAILED PATTERN IMPLEMENTATIONS: TO BE DEFINED]**

* Factory patterns for event creation
* Builder patterns for complex objects
* Singleton patterns managed by Spring

#### **10.3.2 Structural Patterns**

**Repository Pattern:**

* Data access abstraction layer
* Spring Data JPA repositories
* Separation of domain logic from persistence

**[ADDITIONAL STRUCTURAL PATTERNS: TO BE DEFINED]**

* Adapter patterns
* Facade patterns
* Proxy patterns

#### **10.3.3 Behavioral Patterns**

**Observer Pattern:**

* Kafka event pub/sub mechanism
* Service event listeners
* Notification aggregation from multiple sources

**[ADDITIONAL BEHAVIORAL PATTERNS: TO BE DEFINED]**

* Strategy pattern implementations
* Template method patterns
* Chain of responsibility patterns

### **10.4 Anti-Patterns Avoided**

**Distributed Monolith:**

* Services avoid tight coupling through shared databases
* No synchronous chains of service calls
* Event-driven architecture prevents cascading dependencies

**[ADDITIONAL ANTI-PATTERNS ANALYSIS: TO BE DEFINED]**

* Chatty services mitigation strategies
* God service prevention
* Shared database anti-pattern avoidance

## **11. System Design Diagrams**

### **11.1 System Context Diagram**

[PLACE HERE]

### **11.2 Service Interaction Sequence Diagram**

#### **11.2.1 Complete Purchase Flow**

[PLACE DIAGRAM HERE]

**[ADDITIONAL SEQUENCE DIAGRAMS: TO BE DEFINED]**

* Failed payment flow
* Failed dispensing flow
* Low stock alert flow
* Admin operations flow

### **11.3 Kafka Topic Flow Diagram**

[PLACE DIAGRAM HERE]

### **11.4 Component Diagram**

[PLACE DIAGRAM HERE]

### **11.5 Database Entity Relationship Diagram**

[PLACE DIAGRAM HERE]

### **11.6 Deployment Diagram (Local Development)**

[PLACE DIAGRAM HERE]

**[ADDITIONAL DEPLOYMENT DIAGRAMS: TO BE DEFINED]**

* Production deployment architecture
* Cloud deployment topology
* Container orchestration diagram
* Network security zones

## **12. API Design Standards**

### **12.1 RESTful API Conventions**

#### **12.1.1 Resource Naming**

**Guidelines:**

* Use plural nouns for collections: /api/products, /api/transactions
* Use hierarchical structure for relationships: /api/inventory/products/{id}/stock
* Use hyphens for multi-word resources (if needed)
* Keep URLs lowercase
* Avoid verbs in URLs (use HTTP methods instead)

**Examples from SRS:**

✓ GET /api/inventory/products  
✓ GET /api/inventory/availability/{productId}  
✓ POST /api/admin/inventory/products  
✓ PUT /api/admin/inventory/stock/{productId}  
✓ POST /api/transaction/purchase  
✓ POST /api/payment/process

#### **12.1.2 HTTP Method Usage**

**Standard HTTP Methods:**

* **GET:** Retrieve resources (read-only operations)
* **POST:** Create new resources
* **PUT:** Update/Replace entire resource
* **DELETE:** Remove resources

**[DETAILED HTTP METHOD SPECIFICATIONS: TO BE DEFINED]**

* PATCH method usage policy
* Idempotency requirements
* Safe method guidelines

#### **12.1.3 HTTP Status Code Standards**

**Success Codes:**

* 200 OK - Successful GET, PUT, DELETE
* 201 Created - Successful POST (resource created)

**Client Error Codes:**

* 400 Bad Request - Invalid request syntax or parameters
* 401 Unauthorized - Missing or invalid authentication
* 403 Forbidden - Authenticated but not authorized
* 404 Not Found - Resource does not exist

**Server Error Codes:**

* 500 Internal Server Error - Unexpected server error
* 503 Service Unavailable - Service temporarily unavailable

**[DETAILED STATUS CODE USAGE MATRIX: TO BE DEFINED]**

* Complete status code mappings per endpoint
* Error scenario to status code matrix
* Custom status code policies

### **12.2 Request/Response Format Standards**

#### **12.2.1 Standard Response Format**

**[STANDARD RESPONSE WRAPPER STRUCTURE: TO BE DEFINED]**

* JSON response format specification
* Success response schema
* Error response schema
* Metadata inclusion policy

**Known Requirements:**

* All REST API endpoints shall respond within 2000ms under normal load
* JSON-based request/response format
* Standard HTTP status codes for operation results
* Correlation ID headers for request tracing

#### **12.2.2 Error Response Standards**

**[ERROR RESPONSE FORMAT: TO BE DEFINED]**

* Error code taxonomy
* Error message structure
* Validation error format
* Business logic error format
* System error format

**Known Requirements:**

* Clear, descriptive error messages with correlation IDs
* Standard HTTP status codes

### **12.3 API Versioning Strategy**

**[API VERSIONING POLICY: TO BE DEFINED]**

* Versioning approach (URL path, header, query parameter)
* Version deprecation policy
* Backward compatibility requirements
* Version migration strategy

### **12.4 Query Parameters Standards**

**[QUERY PARAMETER SPECIFICATIONS: TO BE DEFINED]**

* Filtering syntax
* Sorting syntax
* Pagination parameters
* Field selection policy

### **12.5 Security Headers**

#### **12.5.1 Request Headers**

**Required for Admin Endpoints:**

* Authorization: Bearer {JWT\_TOKEN} - JWT authentication token (8-hour expiry)
* Content-Type: application/json

**[ADDITIONAL HEADER SPECIFICATIONS: TO BE DEFINED]**

* Optional request headers
* Custom header definitions
* Header validation rules

#### **12.5.2 Response Headers**

**Standard Response Headers:**

* Content-Type: application/json
* Correlation ID header for request tracing

**[SECURITY HEADER SPECIFICATIONS: TO BE DEFINED]**

* CORS headers configuration
* Security headers (X-Content-Type-Options, X-Frame-Options, etc.)
* Rate limiting headers
* Response timing headers

### **12.6 Rate Limiting**

**Known Requirements:**

* Basic request rate limiting at gateway level

**[RATE LIMITING SPECIFICATIONS: TO BE DEFINED]**

* Rate limit thresholds per endpoint type
* Rate limit exceeded response format
* Rate limit headers
* Rate limiting algorithm

### **12.7 API Documentation Standards**

**[API DOCUMENTATION REQUIREMENTS: TO BE DEFINED]**

* Documentation format (Swagger/OpenAPI, etc.)
* Endpoint documentation template
* Example request/response requirements
* Testing collection requirements

**Known Requirements:**

* Comprehensive API documentation with examples
* Complete Postman collection for all endpoints

## **13. Error Handling, Logging & Monitoring**

### **13.1 Error Handling Strategy**

#### **13.1.1 Error Classification**

**Known Error Categories:**

* Transient errors (network timeouts, service unavailability)
* Validation errors (input validation failures)
* Business logic errors (insufficient stock, payment failures)
* Authorization errors (access denied)
* System errors (unexpected failures)

**[DETAILED ERROR CLASSIFICATION: TO BE DEFINED]**

* Complete error taxonomy
* Error severity levels
* Error handling procedures per category
* Recovery strategies

#### **13.1.2 Exception Handling Hierarchy**

**[EXCEPTION HIERARCHY DESIGN: TO BE DEFINED]**

* Base exception classes
* Service-specific exceptions
* HTTP status code mappings
* Exception propagation rules

#### **13.1.3 Global Exception Handler**

**[EXCEPTION HANDLER IMPLEMENTATION: TO BE DEFINED]**

* Global exception handler pattern
* Exception to HTTP status mapping
* Error response generation
* Logging integration

#### **13.1.4 Retry Mechanisms**

**Known Requirements:**

* Automatic retry mechanisms for transient failures
* Timeout and retry mechanisms for service communication

**[RETRY CONFIGURATION: TO BE DEFINED]**

* Maximum retry attempts
* Backoff strategy (exponential, linear, etc.)
* Retry-eligible scenarios
* Circuit breaker integration

### **13.2 Correlation ID Tracking**

#### **13.2.1 Correlation ID Strategy**

**Known Requirements:**

* Correlation ID headers for request tracing
* Correlation IDs in error messages
* Request tracing across services

**[CORRELATION ID IMPLEMENTATION: TO BE DEFINED]**

* ID generation strategy (UUID, sequential, etc.)
* Header name convention
* Propagation mechanism across services
* Storage in database records
* Integration with Kafka events

#### **13.2.2 MDC (Mapped Diagnostic Context) Usage**

**[MDC IMPLEMENTATION: TO BE DEFINED]**

* MDC key definitions
* Thread-local context management
* Filter/interceptor implementation
* Cleanup procedures

### **13.3 Logging Standards**

#### **13.3.1 Log Levels**

**[LOG LEVEL USAGE POLICY: TO BE DEFINED]**

* TRACE level usage
* DEBUG level usage
* INFO level usage
* WARN level usage
* ERROR level usage
* FATAL/CRITICAL level usage

#### **13.3.2 Structured Logging Format**

**Known Requirements:**

* Structured logging with correlation ID tracking
* Comprehensive logging for all operations

**[LOGGING FORMAT SPECIFICATION: TO BE DEFINED]**

* Log message format/pattern
* Structured logging schema (JSON, key-value)
* Required fields per log entry
* Optional contextual fields

#### **13.3.3 Logging Best Practices**

**[LOGGING GUIDELINES: TO BE DEFINED]**

* What to log (events, errors, business operations)
* What NOT to log (sensitive data, PII)
* Log message construction guidelines
* Performance considerations
* Data masking requirements

### **13.4 Monitoring & Observability**

#### **13.4.1 Health Checks**

**Known Implementation:**

* Health check endpoints via Spring Boot Actuator
* Service availability monitoring

**[HEALTH CHECK SPECIFICATIONS: TO BE DEFINED]**

* Health check endpoint definitions
* Health check response format
* Component health check details (database, Kafka)
* Health check intervals
* Failure thresholds

#### **13.4.2 Custom Metrics**

**Known Requirements:**

* Health checks and metrics via Actuator endpoints

**[METRICS DEFINITION: TO BE DEFINED]**

* Business metrics (transactions, payments, inventory)
* Technical metrics (response times, error rates)
* Infrastructure metrics (JVM, database, Kafka)
* Metric naming conventions
* Metric collection frequency

#### **13.4.3 Alerting Strategy**

**Known Requirements from SRS:**

* Low stock alerts when quantity ≤ minimum threshold
* Notification events from all services

**[ALERTING SPECIFICATIONS: TO BE DEFINED]**

* Alert levels and severity
* Alert conditions and thresholds
* Alert response time requirements
* Notification delivery methods
* Alert escalation procedures

#### **13.4.4 Performance Monitoring**

**Known Requirements from SRS Section 6.1:**

* API Response Time: All REST API endpoints shall respond within 2000ms
* Event Processing Time: Kafka events shall be processed within 100ms
* Database Query Time: Database operations shall complete within 500ms
* Service Startup Time: All services shall start within 120 seconds
* Concurrent Users: System shall support 10-50 concurrent transactions
* Event Throughput: Kafka shall handle 1000+ events per minute

**[PERFORMANCE MONITORING IMPLEMENTATION: TO BE DEFINED]**

* Performance metric collection methods
* Performance threshold alerting
* Slow query detection
* Performance dashboards
* Load testing procedures

### **13.5 Distributed Tracing**

**[DISTRIBUTED TRACING STRATEGY: TO BE DEFINED - FUTURE ENHANCEMENT]**

* Tracing framework selection (Sleuth, Zipkin, Jaeger)
* Trace context propagation
* Span creation and management
* Trace visualization

## **14. Requirements Traceability Matrix**

### **14.1 Functional Requirements to Services Mapping**

|  |  |  |
| --- | --- | --- |
| **Requirement ID** | **Requirement Name** | **Implemented In** |
| REQ-INV-001 | Product Management - CRUD operations | Inventory Service |
| REQ-INV-002 | Stock Tracking - Real-time updates | Inventory Service |
| REQ-INV-003 | Low Stock Alerts | Inventory Service, Notification Service |
| REQ-INV-004 | Event-Driven Updates - Publish within 100ms | Inventory Service |
| REQ-PAY-001 | Payment Method Support (Cash, Card) | Payment Service |
| REQ-PAY-002 | Payment Simulation | Payment Service |
| REQ-PAY-003 | Transaction Logging | Payment Service |
| REQ-PAY-004 | Event Publishing within 100ms | Payment Service |
| REQ-TXN-001 | Purchase Orchestration within 5 seconds | Transaction Service |
| REQ-TXN-002 | Inventory Validation | Transaction Service, Inventory Service |
| REQ-TXN-003 | Event-Driven Coordination | Transaction Service |
| REQ-TXN-004 | Basic Failure Handling | Transaction Service |
| REQ-DISP-001 | Dispensing Simulation | Dispensing Service |
| REQ-DISP-002 | Hardware Status Simulation | Dispensing Service |
| REQ-DISP-003 | Event Processing within 100ms | Dispensing Service |
| REQ-NOT-001 | Event Aggregation from all services | Notification Service |
| REQ-NOT-002 | Alert Classification by type and severity | Notification Service |
| REQ-NOT-003 | Administrative Interface | Notification Service |
| REQ-AUTH-001 | JWT Authentication with 8-hour expiry | API Gateway |
| REQ-AUTH-002 | Role-Based Authorization (SUPER\_ADMIN, ADMIN) | API Gateway |
| REQ-AUTH-003 | User Management (create/update/delete) | API Gateway, Auth Service |

### **14.2 Non-Functional Requirements to Design Decisions**

|  |  |  |  |
| --- | --- | --- | --- |
| **NFR ID** | **Requirement** | **Design Decision** | **Validation Method** |
| NFR-PERF-001 | API Response Time <2000ms | Asynchronous processing, connection pooling | Performance testing |
| NFR-PERF-002 | Event Processing <100ms | Kafka configuration optimization | Load testing |
| NFR-PERF-003 | Database Query <500ms | Indexed columns, connection pooling (HikariCP) | Database profiling |
| NFR-PERF-004 | Service Startup <120s | Lazy initialization, minimal dependencies | Startup monitoring |
| NFR-SCALE-001 | 50+ Concurrent Users | Connection pooling, async processing | Load testing |
| NFR-SCALE-002 | 1000+ Events/Minute | Kafka partitioning (3 partitions per topic) | Performance testing |
| NFR-SCALE-003 | Database Connections | Connection pool: 20+ concurrent per service | [TO BE DEFINED] |
| NFR-REL-001 | 95%+ Service Uptime | Health checks, retry mechanisms | Uptime monitoring |
| NFR-REL-002 | Eventual Consistency | Event-driven architecture via Kafka | Integration testing |
| NFR-REL-003 | At-Least-Once Delivery | Kafka configuration | Integration testing |
| NFR-USE-001 | RESTful API Design | Standard HTTP methods and status codes | API documentation review |
| NFR-USE-002 | Clear Error Messages | Structured error responses with correlation IDs | Error scenario testing |
| NFR-USE-003 | Comprehensive Documentation | API documentation with examples | Documentation review |
| NFR-USE-004 | Testing Tools | Complete Postman collection | [TO BE DEFINED] |
| NFR-MAINT-001 | Clean Architecture | Separation of concerns, SOLID principles | Code review |
| NFR-MAINT-002 | Externalized Configuration | Config Server (file-based) | Configuration testing |
| NFR-MAINT-003 | Structured Logging | Correlation IDs, structured format | Log analysis |
| NFR-SEC-001 | JWT Authentication | Token-based stateless auth, 8-hour expiry | Security testing |
| NFR-SEC-002 | Password Hashing | BCrypt with configurable rounds | Security audit |
| NFR-SEC-003 | Input Validation | Validation and sanitization | Validation testing |
| NFR-SEC-004 | Audit Logging | Admin operation logging | [TO BE DEFINED] |

### **14.3 Gap Analysis**

**Implemented Features (from SRS and Plan):** ✓ All core functional requirements (21 requirements) ✓ Microservices architecture with 8 services ✓ Event-driven communication via Kafka ✓ JWT-based authentication and RBAC ✓ Database-per-service pattern ✓ Service discovery (Eureka) and centralized configuration ✓ Basic health checks via Actuator ✓ Correlation ID tracking capability

**Acknowledged Limitations (from SRS Section 2.5):**

* Local Development Only: No production deployment considerations
* Simulated Hardware: No real vending machine integration
* Single Instance Deployment: No clustering or high availability
* Manual Database Schema: No automated migration tools
* HTTP Communication: No HTTPS/SSL for local development

**Future Enhancements (Out of Scope - from Plan):**

* Real payment gateway integration
* Advanced caching with Redis
* Container orchestration (Docker/Kubernetes)
* Advanced monitoring (Prometheus/Grafana)
* Message persistence and replay capabilities
* Advanced security (OAuth2, refresh tokens)
* Multi-channel notification delivery
* Real hardware integration
* Advanced analytics and reporting

**[DETAILED GAP ANALYSIS: TO BE COMPLETED]**

* Missing features identification
* Priority ranking of gaps
* Implementation roadmap
* Risk assessment

## **15. Database Design Enhancements**

### **15.1 Data Integrity Constraints**

#### **15.1.1 Referential Integrity**

**Foreign Key Constraints (from SRS Section 7.2):**

-- Inventory Service  
ALTER TABLE stock  
ADD CONSTRAINT fk\_stock\_product  
FOREIGN KEY (product\_id) REFERENCES products(id);  
  
-- Transaction Service  
ALTER TABLE transaction\_items  
ADD CONSTRAINT fk\_transaction\_items\_transaction  
FOREIGN KEY (transaction\_id) REFERENCES transactions(id);

**[ADDITIONAL CONSTRAINT SPECIFICATIONS: TO BE DEFINED]**

* ON DELETE policies (CASCADE, RESTRICT, SET NULL)
* ON UPDATE policies
* Constraint naming conventions
* Deferred constraint evaluation

#### **15.1.2 Check Constraints**

**[CHECK CONSTRAINT SPECIFICATIONS: TO BE DEFINED]**

* Price validation (must be positive)
* Quantity validation (non-negative)
* Enum value validation
* Business rule constraints

#### **15.1.3 Unique Constraints**

**Known Unique Constraints (from SRS Section 7.2):**

* admin\_users.username - UNIQUE
* payment\_transactions.transaction\_id - UNIQUE
* stock.product\_id - UNIQUE (one stock record per product)

**[ADDITIONAL UNIQUE CONSTRAINT SPECIFICATIONS: TO BE DEFINED]**

* Complete unique constraint list
* Composite unique keys
* Unique index strategy

### **15.2 Index Strategy**

**Existing Indexes (from SRS Section 7.2):**

-- Authentication Service  
CREATE INDEX idx\_username ON admin\_users(username);  
CREATE INDEX idx\_active ON admin\_users(active);  
  
-- Inventory Service  
CREATE INDEX idx\_product\_active ON products(active);  
CREATE INDEX idx\_stock\_quantity ON stock(quantity);  
  
-- Payment Service  
CREATE INDEX idx\_transaction\_id ON payment\_transactions(transaction\_id);  
CREATE INDEX idx\_payment\_status ON payment\_transactions(status);  
CREATE INDEX idx\_payment\_created ON payment\_transactions(created\_at);  
  
-- Transaction Service  
CREATE INDEX idx\_transaction\_status ON transactions(status);  
CREATE INDEX idx\_transaction\_created ON transactions(created\_at);  
CREATE INDEX idx\_transaction\_items\_txn ON transaction\_items(transaction\_id);  
  
-- Dispensing Service  
CREATE INDEX idx\_dispensing\_status ON dispensing\_operations(status);  
CREATE INDEX idx\_dispensing\_transaction ON dispensing\_operations(transaction\_id);  
  
-- Notification Service  
CREATE INDEX idx\_notification\_type ON notifications(type);  
CREATE INDEX idx\_notification\_severity ON notifications(severity);  
CREATE INDEX idx\_notification\_acknowledged ON notifications(acknowledged);  
CREATE INDEX idx\_notification\_created ON notifications(created\_at);

**[ADDITIONAL INDEX SPECIFICATIONS: TO BE DEFINED]**

* Composite index strategy
* Index maintenance procedures
* Index usage monitoring
* Index rebuild schedule

### **15.3 Data Archiving Strategy**

**[ARCHIVING POLICY: TO BE DEFINED]**

* Data retention periods per table
* Archive table structures
* Archiving procedures
* Archive access methods
* Purge policies

### **15.4 Database Performance Optimization**

#### **15.4.1 Query Optimization Guidelines**

**[QUERY OPTIMIZATION STANDARDS: TO BE DEFINED]**

* Query best practices
* Query anti-patterns to avoid
* Execution plan analysis
* Query tuning procedures

#### **15.4.2 Connection Pooling Configuration**

**Known Configuration (from SRS Section 7.4):**

* Connection Pool Size: 10-20 connections per service
* Connection Timeout: 30 seconds
* Idle Timeout: 600 seconds
* Max Lifetime: 1800 seconds
* Auto Commit: false (explicit transaction management)

**[DETAILED CONNECTION POOL SPECIFICATIONS: TO BE DEFINED]**

* HikariCP complete configuration
* Pool sizing strategy
* Connection validation
* Leak detection configuration

#### **15.4.3 Database Caching Strategy**

**[CACHING STRATEGY: TO BE DEFINED]**

* Query result caching approach
* Cache invalidation strategy
* Cache configuration
* Cache monitoring

### **15.5 Database Backup and Recovery**

**[BACKUP AND RECOVERY PROCEDURES: TO BE DEFINED]**

* Backup schedule (full, incremental, differential)
* Backup retention policy
* Backup verification procedures
* Point-in-time recovery procedures
* Disaster recovery plan

## **16. Security Enhancements**

### **16.1 Security Testing Requirements**

**[SECURITY TEST SPECIFICATIONS: TO BE DEFINED]**

* Authentication testing scenarios
* Authorization testing scenarios
* Input validation testing
* Session management testing
* Security vulnerability scanning

### **16.2 Secrets Management**

#### **16.2.1 Environment-Specific Configuration**

**Known Requirements (from SRS Section 8.1):**

* JWT signing secret: Configurable
* Password hashing: BCrypt with configurable rounds
* User credentials: MySQL authentication

**[SECRETS MANAGEMENT STRATEGY: TO BE DEFINED]**

* Secret storage mechanism (environment variables, vault, etc.)
* Secret rotation policy
* Secret distribution to services
* Development vs. production secret handling

#### **16.2.2 Secrets Rotation Policy**

**[ROTATION PROCEDURES: TO BE DEFINED]**

* Rotation schedule
* Rotation procedures
* Service restart strategy
* Token/credential invalidation

### **16.3 Security Monitoring and Audit**

#### **16.3.1 Audit Logging**

**[AUDIT LOGGING SPECIFICATIONS: TO BE DEFINED]**

* Auditable events list
* Audit log structure
* Audit log retention
* Audit log analysis procedures

**Known Requirements:**

* Complete logging of administrative operations (from SRS Section 8.3)

#### **16.3.2 Security Event Monitoring**

**[SECURITY MONITORING SPECIFICATIONS: TO BE DEFINED]**

* Monitored security events
* Alert triggers and thresholds
* Security incident response procedures
* Brute force protection mechanisms

## **17. Deployment Considerations**

### **17.1 Local Development Deployment**

#### **17.1.1 Service Startup Order**

**Recommended Startup Sequence (from Development Plan):**

**Phase 1 - Infrastructure:**

1. MySQL Server (Port 3306)
2. Zookeeper (Port 2181)
3. Kafka Broker (Port 9092)

**Phase 2 - Configuration:** 4. Config Server (Port 8888)

**Phase 3 - Service Discovery:** 5. Eureka Server (Port 8761)

**Phase 4 - Business Services (parallel start):** 6. Inventory Service (Port 8081) 7. Payment Service (Port 8082) 8. Transaction Service (Port 8083) 9. Dispensing Service (Port 8084) 10. Notification Service (Port 8085)

**Phase 5 - Gateway:** 11. API Gateway (Port 8080)

**[STARTUP AUTOMATION: TO BE DEFINED]**

* Startup scripts for each platform
* Health check wait procedures
* Dependency verification
* Startup troubleshooting guide

#### **17.1.2 Environment Configuration**

**Development Environment Requirements (from SRS Section 2.4):**

* Operating System: Cross-platform (Windows, macOS, Linux)
* Java Runtime: Java 17 or higher
* Database: MySQL 8.0+
* Message Broker: Apache Kafka with Zookeeper
* Memory Requirements: 6GB RAM minimum
* Storage: 2GB free disk space

**[ENVIRONMENT CONFIGURATION DETAILS: TO BE DEFINED]**

* Environment variable definitions
* Configuration file templates
* Service-specific configurations
* Local development setup guide

### **17.2 Future Deployment Strategies**

#### **17.2.1 Containerization (Docker)**

**[CONTAINERIZATION STRATEGY: TO BE DEFINED - FUTURE ENHANCEMENT]**

* Docker image definitions
* Docker Compose configuration
* Container networking
* Volume management
* Container orchestration

**Note:** Container-ready design mentioned in SRS Section 2.6

#### **17.2.2 Cloud Deployment Considerations**

**[CLOUD DEPLOYMENT ARCHITECTURE: TO BE DEFINED - FUTURE ENHANCEMENT]**

* Cloud provider selection
* Service deployment topology
* Managed services integration
* Scalability configuration
* High availability design