

# **Software Requirements Specification (SRS)**

## **Internal Inventory Management System**

**Prepared by:** Dhanajalie Wijerathne

**Company:** Innovior (Pvt) Ltd

## Table of Contents

<b>1. Introduction.....</b>	5
<b>1.1 Purpose.....</b>	5
<b>1.2 Scope.....</b>	5
<b>1.3 Definitions and Abbreviations .....</b>	5
<b>1.4 References .....</b>	6
<b>2. Overall Description.....</b>	6
<b>2.1 Product Perspective .....</b>	6
<b>2.2 Product Functions.....</b>	6
<b>2.3 User Classes and Characteristics.....</b>	7
<b>2.4 Operating Environment .....</b>	7
<b>2.5 Design and Implementation Constraints .....</b>	8
<b>3. System Features .....</b>	8
<b>3.1 Product and Catalog Management.....</b>	8
<b>3.2 Warehouses and Inventory Levels .....</b>	8
<b>3.3 Stock Movements .....</b>	9
<b>3.4 Suppliers and Purchase Orders .....</b>	9
<b>3.5 Reorder Points and Alerts.....</b>	10
<b>3.6 Reporting .....</b>	10
<b>4. External Interface Requirements .....</b>	11
<b>4.1 User Interfaces .....</b>	11
<b>4.2 Hardware Interfaces.....</b>	11

<b>4.3 Software Interfaces .....</b>	11
<b>5. Non-Functional Requirements.....</b>	12
<b>  5.1 Performance Requirements .....</b>	12
<b>  5.2 Security Requirements .....</b>	12
<b>  5.3 Reliability Requirements.....</b>	12
<b>  5.4 Usability Requirements .....</b>	12
<b>  5.5 Scalability Requirements .....</b>	12
<b>6. System Architecture.....</b>	13
<b>  6.1 Architectural Overview .....</b>	13
<b>  6.2 Technology Stack .....</b>	13
<b>  6.3 System Components.....</b>	14
<b>7. Data Requirements .....</b>	14
<b>  7.1 Database Schema .....</b>	14
<b>  7.2 Data Validation Rules.....</b>	18
<b>  7.3 Data Backup and Recovery .....</b>	18
<b>8. Implementation Plan .....</b>	19
<b>  8.1 Development Phases.....</b>	19
<b>  8.2 Implementation Priority Matrix.....</b>	20
<b>  8.3 Risk Mitigation Strategies.....</b>	21
<b>9. Quality Assurance.....</b>	21
<b>  9.1 Testing Strategy .....</b>	21
<b>  9.2 Code Quality Standards .....</b>	21
<b>  9.3 Security Testing.....</b>	21

<b>10. Appendices.....</b>	22
<b>Appendix A: File Structure Overview .....</b>	22
<b>Appendix B: API Endpoint Documentation.....</b>	22
<b>Appendix C: Database Indexing Strategy .....</b>	23
<b>Appendix D: Deployment Configuration.....</b>	23

# 1. Introduction

## 1.1 Purpose

This Software Requirements Specification (SRS) document outlines the functional and non-functional requirements for the Internal Inventory Management System. The system is designed to manage inventory across multiple warehouses, track stock movements, handle purchase orders, and provide reporting for internal staff use only, accessible via company network or SSO/VPN.

## 1.2 Scope

The system includes:

- **Inventory Management:** Real-time tracking of stock levels, movements, and valuation.
- **Purchase Order Management:** Creation, approval, and receiving of purchase orders.
- **User Management:** Role-based access for Admin, Inventory Manager, Clerk, and Viewer.
- **Reporting:** Stock on hand, movement history, aging, and valuation reports.
- **Alerts:** Low-stock notifications and reorder point management.

**Out of Scope (v1):** Lot/batch tracking, serial number tracking, barcode integration, advanced costing beyond FIFO/moving average, and external supplier EDI integration.

## 1.3 Definitions and Abbreviations

SKU - Stock Keeping Unit - unique product identifier

PO - Purchase Order

Qty\_on\_hand - Physical quantity available in a warehouse

RBAC - Role-Based Access Control

FIFO - First-In, First-Out inventory valuation method

MVP - Minimum Viable Product

SSO - Single Sign-On

VPN - Virtual Private Network

## 1.4 References

- Laravel 11 Documentation
- MySQL 8 Documentation
- Redis Documentation
- Spatie Permission Package Documentation
- Laravel Breeze Documentation

## 2. Overall Description

### 2.1 Product Perspective

The Internal Inventory Management System is a web-based application for internal staff to manage inventory across multiple warehouses, ensuring accurate stock tracking, movement logging, and purchase order processing, with secure access via SSO/VPN or IP allowlist.

### 2.2 Product Functions

- **Catalog Management:** Manage products, categories, and units of measure.
- **Inventory Control:** Track stock levels and movements (receipts, issues, transfers, adjustments).
- **Purchase Order Processing:** Create, approve, and receive purchase orders.
- **Reorder Alerts:** Notify managers of low stock based on reorder points.
- **Reporting:** Generate stock, movement, aging, and valuation reports.
- **User Management:** Role-based access control for internal users.

## 2.3 User Classes and Characteristics

User Type	Access Level	Primary Functions
Admin	Internal	Manage users, roles, system settings
Inventory Manager	Internal	Manage products, warehouses, approve adjustments/transfers, view reports
Clerk	Internal	Create POs, record receipts/issues, request transfers
Viewer	Internal	Read-only access to stock and reports

## 2.4 Operating Environment

- **Backend:** PHP 8.2+, Laravel 11, MySQL 8, Redis
- **Frontend:** Blade templates or Livewire components
- **Infrastructure:** Company network with SSO/VPN or IP allowlist, Redis for queues/cache, SMTP for alerts
- **Development:** GitHub Actions for CI/CD

## **2.5 Design and Implementation Constraints**

- Internal-only access via SSO/VPN or IP allowlist.
- No negative inventory allowed (prevent overselling).
- Use Laravel Breeze for authentication and Spatie Permission for RBAC.
- Asynchronous tasks (alerts, reports) handled via Laravel queues.
- Development timeline: 4 weeks for MVP by one intern.

# **3. System Features**

## **3.1 Product and Catalog Management**

### **3.1.1 Description**

Manage product master data, categories, and units of measure with import/export capabilities.

### **3.1.2 Functional Requirements**

#### **FR-PROD-01: Product Creation and Management**

- System shall allow managers to create, edit, and deactivate products with SKU, name, category, unit, cost price, and sale price.
- SKU must be unique; inactive products cannot be used in transactions.

#### **FR-PROD-02: Product Search and Listing**

- Clerks can search products by SKU or name and view stock levels by warehouse.
- Product lists support pagination, filters (category, active), and sorting.

#### **FR-PROD-03: Import/Export**

- Managers can import products via CSV with validation for required fields.
- System supports CSV export of product lists.

## **3.2 Warehouses and Inventory Levels**

### **3.2.1 Description**

Track stock levels per product per warehouse with reorder point management.

### **3.2.2 Functional Requirements**

### **FR-WH-01: Warehouse Management**

- Managers can create/edit warehouses with unique code, name, and address.

### **FR-WH-02: Inventory Tracking**

- System tracks qty\_on\_hand and reorder\_point per product per warehouse.
- Available stock displayed across warehouses.

### **FR-WH-03: Reorder Points**

- Managers can set reorder points per product per warehouse.
- System generates reorder reports for items below thresholds.

## **3.3 Stock Movements**

### **3.3.1 Description**

Record and track stock receipts, issues, transfers, and adjustments with audit trails.

### **3.3.2 Functional Requirements**

#### **FR-MOVE-01: Stock Transactions**

- Clerks can record receipts, issues, and transfers; managers can approve adjustments.
- Movements capture user, timestamp, type, reference, and optional unit cost.

#### **FR-MOVE-02: Inventory Invariants**

- System enforces non-negative inventory (qty\_on\_hand  $\geq 0$ ).
- Transfers are atomic, creating paired ISSUE and RECEIPT movements.

#### **FR-MOVE-03: Audit Trail**

- All movements logged immutably with user, timestamp, and reason.
- Adjustments require reason and optional manager approval (configurable).

## **3.4 Suppliers and Purchase Orders**

### **3.4.1 Description**

Manage suppliers and purchase order lifecycle, including partial receiving.

### **3.4.2 Functional Requirements**

### **FR-PO-01: Supplier Management**

- Managers can create/edit suppliers with name, email, phone, and address.

### **FR-PO-02: Purchase Order Lifecycle**

- Clerks can create POs with supplier, product lines, quantities, costs, and expected dates.
- Managers can approve, send, or cancel POs.
- PO status: DRAFT → SENT → RECEIVED or CANCELLED.

### **FR-PO-03: Receiving**

- Clerks can record full or partial receipts against POs, updating stock and qty\_received.
- POs close automatically when fully received.

## **3.5 Reorder Points and Alerts**

### **3.5.1 Description**

Monitor stock levels and notify managers of low stock via email or logs.

### **3.5.2 Functional Requirements**

#### **FR-ALERT-01: Low Stock Alerts**

- Nightly job identifies items below reorder points and emails managers.
- Dashboard widget displays low-stock items with filters.

## **3.6 Reporting**

### **3.6.1 Description**

Provide reports on stock, movements, aging, and valuation with CSV export.

### **3.6.2 Functional Requirements**

#### **FR-REP-01: Stock Reports**

- Stock on hand by warehouse/product.
- Movement history by date range, type, and reference.

#### **FR-REP-02: Valuation Reports**

- Stock aging and valuation using FIFO or moving average.

## **FR-REP-03: Data Export**

- All reports exportable to CSV.

# **4. External Interface Requirements**

## **4.1 User Interfaces**

### **4.1.1 Administrative Interface**

- Role-based dashboard with widgets for stock, POs, and alerts.
- Data tables with sorting, filtering, and pagination.
- Clear form validation and error messages.
- Keyboard-friendly navigation with modal dialogs for actions.

## **4.2 Hardware Interfaces**

- Standard web server hardware with sufficient storage and memory.
- Redis server for caching and queue management.
- MySQL server for data storage.

## **4.3 Software Interfaces**

### **4.3.1 Database Integration**

- MySQL 8: Primary data storage with relational schema.
- Redis: Caching and queue processing.

### **4.3.2 Email Integration**

- SMTP: Configurable for low-stock alerts and notifications.
- Queue-based asynchronous email delivery.

## 5. Non-Functional Requirements

### 5.1 Performance Requirements

- Page load times: < 1.5 seconds for lists and forms.
- Report generation: Asynchronous for heavy reports, < 5 seconds for simple reports.
- Concurrent users: Support at least 50 concurrent sessions.

### 5.2 Security Requirements

- **Authentication:** Laravel Breeze with secure password hashing (bcrypt).
- **Authorization:** RBAC via Spatie Permission package.
- **Input Validation:** Form Requests for server-side validation and sanitization.
- **CSRF Protection:** Enabled for all web forms.
- **Rate Limiting:** Applied to write endpoints.
- **Secrets Management:** Stored in .env, never committed.

### 5.3 Reliability Requirements

- **Uptime:** 99.9% during business hours.
- **Transaction Integrity:** ACID compliance for stock updates using DB transactions.
- **Error Handling:** User-friendly error messages with graceful degradation.

### 5.4 Usability Requirements

- Intuitive forms with clear validation feedback.
- Maximum 3 clicks to access key functions.
- Browser support: Latest Chrome, Firefox, Safari, Edge.

### 5.5 Scalability Requirements

- Horizontal scaling via load balancing.
- Redis caching to reduce database load.
- Queue management for asynchronous tasks.

## 6. System Architecture

### 6.1 Architectural Overview

The system follows the Model-View-Controller (MVC) pattern with a service-oriented approach for business logic.

### 6.2 Technology Stack

Layer	Technology	Purpose
Frontend	Blade/Livewire	User interface rendering
Backend	PHP 8.2, Laravel 11	Business logic and API
Database	MySQL 8	Relational data storage
Cache/Queue	Redis	Caching and asynchronous tasks
Email	SMTP/Laravel Mail	Notification delivery

## 6.3 System Components

### 6.3.1 Presentation Layer

- Blade or Livewire for admin UI with role-based dashboards.
- RESTful API for data interaction.

### 6.3.2 Business Logic Layer

- **Controllers:** Handle HTTP requests.
- **Services:** Encapsulate business logic (e.g., StockMovementService).
- **Form Requests:** Input validation.
- **Policies:** RBAC enforcement.

### 6.3.3 Data Layer

- **Models:** Eloquent ORM for MySQL.
- **Migrations:** Schema versioning.
- **Seeders/Factories:** Test and demo data.

## 7. Data Requirements

### 7.1 Database Schema

#### 7.1.1 Core Tables

##### Products

{

id: Integer,

sku: String (unique),

name: String,

category\_id: Integer,

unit\_id: Integer,

```
    description: Text,  
    cost_price: Decimal,  
    sale_price: Decimal,  
    is_active: Boolean,  
    created_at: DateTime,  
    updated_at: DateTime  
}
```

## **Warehouses**

```
{  
    id: Integer,  
    code: String (unique),  
    name: String,  
    address: Text,  
    created_at: DateTime,  
    updated_at: DateTime  
}
```

## **Inventory Levels**

```
{  
    id: Integer,  
    product_id: Integer,  
    warehouse_id: Integer,  
    qty_on_hand: Integer,  
    reorder_point: Integer,
```

```
    created_at: DateTime,  
    updated_at: DateTime  
}
```

## Stock Movements

```
{  
    id: Integer,  
    product_id: Integer,  
    warehouse_id_from: Integer (nullable),  
    warehouse_id_to: Integer (nullable),  
    type: Enum (RECEIPT, ISSUE, TRANSFER, ADJUSTMENT),  
    qty: Integer,  
    unit_cost: Decimal (nullable),  
    ref_no: String,  
    user_id: Integer,  
    moved_at: DateTime,  
    created_at: DateTime,  
    updated_at: DateTime  
}
```

## Suppliers

```
{  
    id: Integer,  
    name: String,  
    email: String,
```

```
    phone: String,  
    address: Text,  
    created_at: DateTime,  
    updated_at: DateTime  
}
```

## Purchase Orders

```
{  
    id: Integer,  
    po_number: String (unique),  
    supplier_id: Integer,  
    status: Enum (DRAFT, SENT, RECEIVED, CANCELLED),  
    expected_date: Date,  
    created_by: Integer,  
    created_at: DateTime,  
    updated_at: DateTime  
}
```

## PO Lines

```
{  
    id: Integer,  
    purchase_order_id: Integer,  
    product_id: Integer,  
    qty_ordered: Integer,  
    unit_cost: Decimal,
```

```
    qty_received: Integer,  
  
    created_at: DateTime,  
  
    updated_at: DateTime  
  
}
```

## Categories

```
{  
  
    id: Integer,  
  
    name: String  
  
}
```

## Units

```
{  
  
    id: Integer,  
  
    code: String,  
  
    name: String  
  
}
```

## 7.2 Data Validation Rules

- Monetary values: Decimal with 2 decimal places.
- SKU: Alphanumeric, 3-20 characters, unique.
- Email: RFC 5322 compliant.
- Timestamps: UTC timezone.
- qty\_on\_hand, qty\_received: Non-negative integers.

## 7.3 Data Backup and Recovery

- **Backup Frequency:** Daily at 2 AM UTC.
- **Retention:** 30 days for daily backups, 12 months for monthly.
- **Recovery Time Objective:** 4 hours maximum downtime.
- **Recovery Point Objective:** Maximum 1 hour data loss.

## 8. Implementation Plan

### 8.1 Development Phases

#### Phase 1: Foundation Setup (Week 1)

- **Objectives:** Set up environment, database, and authentication.
- **Deliverables:** Migrations, seeders, RBAC, basic UI, product/warehouse CRUD.
- **Key Files:** database/migrations/, app/Models/, routes/web.php, routes/api.php.

#### Phase 2: Core Business Logic (Week 2)

- **Objectives:** Implement stock movements and PO lifecycle.
- **Deliverables:** StockMovementService, receipt/issue/transfer logic, PO creation/approval/receiving.
- **Key Files:** app/Services/StockMovementService.php, app/Http/Controllers/PurchaseOrderController.php.

#### Phase 3: Reports and Alerts (Week 3)

- **Objectives:** Develop reports, alerts, and polish UI.
- **Deliverables:** Stock/movement/valuation reports, low-stock alerts, activity logging.
- **Key Files:** app/Http/Controllers/ReportController.php, app/Jobs/ReorderAlertJob.php.

#### Phase 4: Quality and Deployment (Week 4)

- **Objectives:** Testing, documentation, and deployment preparation.
- **Deliverables:** Feature tests, README, demo data, deployment script, demo video.
- **Key Files:** tests/Feature/, README.md, deploy.sh.

## 8.2 Implementation Priority Matrix

Priority	Component	Business Impact	Technical Risk	Implementation Days
1	Foundation & Auth	High	High	7
2	Product/Warehouse use	Critical	Medium	4
3	Stock Movements	Critical	High	5
4	Purchase Orders	High	Medium	4
5	Reports/Alerts	High	Medium	6

## 8.3 Risk Mitigation Strategies

- **Concurrency:** Use database transactions and row-level locking.
- **Data Integrity:** Validate inputs via Form Requests; enforce invariants in services.
- **Performance:** Cache reports with Redis; use queues for alerts.

# 9. Quality Assurance

## 9.1 Testing Strategy

### 9.1.1 Unit Testing

- **Framework:** Pest/PHPUnit.
- **Coverage:** 80% minimum for services and logic.
- **Focus:** StockMovementService invariants, reorder calculations.

### 9.1.2 Integration Testing

- **Focus:** PO receiving updates stock correctly, no overselling, concurrent transfer safety.
- **Tools:** Laravel testing suite with database transactions.

### 9.1.3 End-to-End Testing

- **Workflows:** Product creation to PO receiving, stock movement logging.
- **Scenarios:** Invalid inputs, concurrent updates, low-stock alerts.

## 9.2 Code Quality Standards

- **Code Style:** PSR-12 via Laravel Pint.
- **Static Analysis:** PHPStan level 6+.
- **Documentation:** Inline comments, README with setup/test instructions.

## 9.3 Security Testing

- **Vulnerability Scanning:** Automated tools in CI pipeline.
- **Input Validation:** Form Requests and sanitization.
- **CSRF/SQL Injection:** Laravel middleware and Eloquent ORM.

# 10. Appendices

## Appendix A: File Structure Overview

```
└── app/
    ├── Http/Controllers/ (Product, Warehouse, PurchaseOrder, Report)
    ├── Models/ (Product, Warehouse, InventoryLevel, StockMovement, etc.)
    ├── Services/ (StockMovementService, ReportService)
    ├── Http/Requests/ (Form validation)
    └── Jobs/ (ReorderAlertJob)

    └── resources/views/ (Blade templates for admin UI)

    └── database/migrations/ (MySQL schema definitions)

    └── database/seeders/ (Demo data)

    └── routes/ (web.php, api.php)

    └── tests/ (Unit and Feature tests)
```

## Appendix B: API Endpoint Documentation

### Product Management

- GET /api/products: List products with filters.
- POST /api/products: Create product (Admin/Manager).
- PUT /api/products/{id}: Update product.
- DELETE /api/products/{id}: Deactivate product.

### Inventory Management

- GET /api/inventory: Stock levels by warehouse/product.
- POST /api/movements/receipt|issue|transfer|adjustment: Record stock movements.

### Purchase Orders

- GET /api/purchase-orders: List POs.
- POST /api/purchase-orders: Create PO.

- POST /api/purchase-orders/{id}/receive: Record receipt.

## Reports

- GET /api/reports/stock-on-hand: Stock levels.
- GET /api/reports/movements: Movement history.

## Appendix C: Database Indexing Strategy

- **Products:** Unique index on sku; index on category\_id, is\_active.
- **Warehouses:** Unique index on code.
- **Inventory Levels:** Compound index on product\_id, warehouse\_id.
- **Stock Movements:** Index on product\_id, moved\_at.
- **Purchase Orders:** Unique index on po\_number; index on supplier\_id, status.

## Appendix D: Deployment Configuration

### Production Requirements

- **Server:** 2 CPU cores, 4GB RAM, 50GB SSD.
- **Network:** Internal network with SSO/VPN or IP allowlist.
- **SSL:** Optional for internal access.

### Sample .env Configuration

```
APP_ENV=production  
  
DB_CONNECTION=mysql  
  
DB_HOST=mysql  
  
DB_DATABASE=inventory  
  
REDIS_HOST=redis  
  
MAIL_DRIVER=smtp
```