Inventory Management System - Complete Guide

🎯 Overview

Comprehensive inventory management system with automatic stock deduction, ingredient tracking per menu item, low-stock alerts, and detailed reporting.

🔑 Key Features

1. Inventory Item Management

- Create/update/delete inventory items
- Track stock levels in real-time
- Support for multiple units (KG, L, PCS, etc.)
- Category-based organization
- Supplier information tracking
- Expiry date monitoring
- Storage location tracking

2. Automatic Stock Deduction

- **Automatic deduction when orders are confirmed**
- Ingredient tracking per menu item
- Quantity validation before order confirmation
- Transaction audit trail
- Support for optional ingredients

3. Low Stock Alerts

- Automatic alert generation
- Alert types: LOW_STOCK, OUT_OF_STOCK, EXPIRING_SOON
- Alert acknowledgment workflow
- Alert resolution tracking
- Real-time alert count

4. Inventory Reporting

- Total inventory value calculation
- Category-wise summaries
- Top value items
- Recently updated items
- Purchase cost analysis
- Transaction history

📊 System Architecture

Entity Relationships

```
InventoryItem (1) ←→ (N) MenuItemInventory ←→ (1) MenuItem
      ↓
      ├── (N) StockTransaction
      └── (N) LowStockAlert
```

Core Entities

InventoryItem

```java
- id, itemCode, name, description
- category (VEGETABLES, MEAT, DAIRY, etc.)
- unit (KG, L, PCS, etc.)
- currentQuantity, minimumQuantity, maximumQuantity
- costPerUnit, supplierName, supplierContact
- status (IN_STOCK, LOW_STOCK, OUT_OF_STOCK)
- expiryDate, storageLocation
```

MenuItemInventory

```java
- menuItemId, inventoryItemId
- quantityRequired (per 1 menu item)
- isOptional (can be made without this ingredient)
```

StockTransaction

```java
- transactionType (PURCHASE, ORDER_DEDUCTION, etc.)
- quantity, quantityBefore, quantityAfter
- costPerUnit, totalCost
- orderId (if deduction for order)
- performedBy, referenceNumber, supplier
```

LowStockAlert

```java
- alertType (LOW_STOCK, OUT_OF_STOCK, EXPIRING_SOON)
- currentQuantity, minimumQuantity
- status (ACTIVE, ACKNOWLEDGED, RESOLVED)
- acknowledgedBy, acknowledgedAt, resolvedAt
```

🔄 Automatic Stock Deduction Flow

Step 1: Link Ingredients to Menu Items

```http
POST /api/inventory/{inventoryItemId}/link-menu-item
{
  "menuItemId": 1,
  "quantityRequired": 0.5,  // 0.5 KG per menu item
  "isOptional": false,
  "notes": "Fresh tomatoes"
}
```

Example: Pizza Margherita

- Tomatoes: 0.5 KG (required)
- Cheese: 0.2 KG (required)
- Basil: 0.05 KG (optional)

Step 2: Customer Places Order

```http
POST /api/orders
{
  "orderType": "DINE_IN",
  "tableNumber": "T5",
  "items": [
    {
      "menuItemId": 1,  // Pizza Margherita
      "quantity": 2,
      "modifierIds": []
    }
  ]
}
```

System automatically checks stock availability:

- Pizza requires: 0.5 KG tomatoes × 2 = 1.0 KG
- Available: 50 KG ✅
- Order proceeds

Step 3: Restaurant Confirms Order

```http
PUT /api/orders/{orderId}/status
{
  "status": "CONFIRMED"
}
```

Automatic stock deduction occurs:

```java
// In OrderService.updateOrderStatus
if (newStatus == Order.OrderStatus.CONFIRMED) {
    inventoryService.deductStockForOrder(order, currentUser);
}
```

What happens:

1. For each order item (2 pizzas)
2. Find all linked ingredients
3. Calculate required quantity: 0.5 KG × 2 = 1.0 KG
4. Check availability
5. Deduct stock: 50 KG → 49 KG
6. Create transaction record
7. Check if alert should be created

Step 4: Transaction Record Created

```json
{
  "transactionType": "ORDER_DEDUCTION",
  "quantity": -1.0,  // Negative for deduction
  "quantityBefore": 50.0,
  "quantityAfter": 49.0,
  "orderId": 123,
  "orderNumber": "ORD20240120001",
  "notes": "Order #ORD123 - Pizza Margherita x2"
}
```

🚨 Low Stock Alert System

Alert Triggers

1. Stock Falls Below Minimum:

```
Current: 8 KG
Minimum: 10 KG
→ Creates LOW_STOCK alert
```

2. Stock Reaches Zero:

```
Current: 0 KG
→ Creates OUT_OF_STOCK alert
```

3. Approaching Expiry:

```
Expiry Date: 2024-01-25
Current Date: 2024-01-23
→ Creates EXPIRING_SOON alert
```

Alert Workflow

```
1. ACTIVE → Alert created, visible to staff
   ↓
2. ACKNOWLEDGED → Staff has seen the alert
   ↓
3. RESOLVED → Stock replenished or issue addressed
```

Get Active Alerts:

```http
GET /api/inventory/alerts/active
```

Response:

```json
{
  "data": [
    {
      "inventoryItemName": "Fresh Tomatoes",
      "itemCode": "TOMATO-001",
      "alertType": "LOW_STOCK",
      "currentQuantity": 8.0,
      "minimumQuantity": 10.0,
      "unit": "KG",
      "status": "ACTIVE"
    }
  ]
}
```

Acknowledge Alert:

```http
POST /api/inventory/alerts/{alertId}/acknowledge
```

Resolve Alert:

```http
POST /api/inventory/alerts/{alertId}/resolve
```

📈 Inventory Reporting

1. Dashboard Summary

```http
GET /api/inventory/reports/summary
```

Response:

```json
{
  "totalItems": 150,
  "inStockItems": 120,
  "lowStockItems": 25,
  "outOfStockItems": 5,
  "totalInventoryValue": 45000.00,
  "categorySummaries": [
    {
      "category": "VEGETABLES",
      "itemCount": 30,
      "totalValue": 5000.00
    }
  ]
}
```

---

2. Stock Transactions

```http
GET /api/inventory/{itemId}/transactions
```

Transaction Types:

- `PURCHASE` - Received from supplier
- `ORDER_DEDUCTION` - Used for order
- `WASTAGE` - Spoiled/damaged
- `MANUAL_ADDITION` - Manual increase
- `MANUAL_DEDUCTION` - Manual decrease
- `ADJUSTMENT` - Stock count adjustment
- `TRANSFER_IN/OUT` - Branch transfers
- `RETURN_TO_SUPPLIER` - Returned items

---

3. Purchase Cost Analysis

```http
GET /api/inventory/reports/purchase-cost?startDate=2024-01-01&endDate=2024-01-31
```

Calculates total cost of purchases in date range

🔧 API Endpoints

Inventory Management

```
POST   /api/inventory                    # Create item
GET    /api/inventory                    # List items
GET    /api/inventory/{id}               # Get item
PUT    /api/inventory/{id}               # Update item
POST   /api/inventory/{id}/add-stock     # Add stock
POST   /api/inventory/{id}/deduct-stock  # Deduct stock
GET    /api/inventory/{id}/transactions  # View transactions
```

Menu Item Linking

```
POST   /api/inventory/{id}/link-menu-item          # Link ingredient
GET    /api/inventory/menu-item/{id}/availability  # Check availability
```

Alerts & Reports

```
GET    /api/inventory/low-stock              # Low stock items
GET    /api/inventory/alerts                 # All alerts
GET    /api/inventory/alerts/active          # Active alerts
GET    /api/inventory/alerts/count           # Alert count
POST   /api/inventory/alerts/{id}/acknowledge # Acknowledge alert
POST   /api/inventory/alerts/{id}/resolve     # Resolve alert
```

💡 Usage Examples

Example 1: Setting Up Inventory for Pizza

Step 1: Create inventory items

```http
POST /api/inventory
{
  "itemCode": "TOMATO-001",
  "name": "Fresh Tomatoes",
  "category": "VEGETABLES",
  "unit": "KG",
  "initialQuantity": 50.0,
  "minimumQuantity": 10.0,
  "costPerUnit": 2.50,
  "supplierName": "Fresh Farms Co."
}
```

Step 2: Link to menu item

```http
POST /api/inventory/1/link-menu-item
{
  "menuItemId": 10,  // Pizza Margherita
  "quantityRequired": 0.5,
  "isOptional": false
}
```

Step 3: Order is placed and confirmed

- Stock automatically deducted: 50 KG → 49.5 KG
- Transaction recorded
- Status updated if needed

Example 2: Restocking Inventory

```http
POST /api/inventory/1/add-stock
{
  "quantity": 30.0,
  "transactionType": "PURCHASE",
  "costPerUnit": 2.50,
  "referenceNumber": "PO-2024-001",
  "supplier": "Fresh Farms Co.",
  "notes": "Weekly delivery"
}
```

Result:

- Stock: 8 KG → 38 KG
- Transaction created
- LOW_STOCK alert auto-resolved

Example 3: Handling Wastage

```http
POST /api/inventory/1/deduct-stock
{
  "quantity": 5.0,
  "transactionType": "WASTAGE",
  "notes": "Spoiled due to storage temperature issue"
}
```

✅ Best Practices

1. Initial Setup

- Create all inventory items before linking to menu items
- Set appropriate minimum quantities based on usage
- Configure reorder quantities for easy restocking

2. Regular Maintenance

- Review low stock alerts daily
- Conduct weekly inventory audits
- Update expiry dates regularly
- Keep supplier information current

3. Menu Item Configuration

- Link all required ingredients
- Mark truly optional ingredients as optional
- Set accurate quantity requirements
- Test availability checks before launch

4. Stock Management

- Always use transaction types correctly
- Include reference numbers for purchases
- Add notes for context
- Review transaction history for anomalies

🔐 Permissions

Role	Create	View	Add Stock	Deduct	Link Menu	View Alerts
RESTAURANT_ADMIN	✅	✅	✅	✅	✅	✅
CHEF	❌	✅	❌	❌	✅	✅
ADMIN	✅	✅	✅	✅	✅	✅

📊 Database Views

inventory_value_summary

Category-wise inventory value and stock status

menu_item_availability

Real-time availability status for all menu items

🚀 Production Features

✅ Automatic stock deduction on order confirmation
✅ Real-time low stock alerts
✅ Complete transaction audit trail
✅ Multi-category support
✅ Multiple unit types
✅ Expiry date tracking
✅ Supplier management
✅ Branch-level inventory
✅ Optional ingredient support
✅ Comprehensive reporting
✅ Database triggers for automation
✅ Multi-tenant safe

🎯 Next Steps

1. **Set up inventory items** for your restaurant
2. **Link ingredients** to menu items
3. **Configure minimum quantities** for alerts
4. **Test order flow** to verify automatic deduction
5. **Monitor alerts** and adjust thresholds
6. **Review reports** to optimize stock levels

Ready for production! 🚀