Report : Inventory Management System

Institute of Engineering and Management

Salt Lake, Kolkata

Submitted By:

- Kaustav Saha
    - Enrolment No.: 12023052017063
    - Class Roll No.: 63
- Aritra Chowdhury
    - Enrolment No.: 12023052017061
    - Class Roll No.: 61

Course Details:

- Subject: Advanced Programming (OOPs) Laboratory
- Stream: CSE(IoTCSBT)
- Semester: 4th

Software Requirements Specification

Version: 1.0
Date: March 18, 2025

Table of Contents

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to define the requirements for the Inventory Management System (IMS). The IMS is designed to assist businesses in efficiently tracking inventory, maintaining optimal stock levels, and providing timely alerts when stock falls below predefined thresholds.

1.2 Scope

The Inventory Management System will be a desktop application targeted at small to medium-sized businesses. It will enable users to add, modify, delete, and view products in their inventory, while also generating alerts for low stock conditions. The application will be implemented using the Java Swing framework following the Model-View-Controller (MVC) architectural pattern.

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
IMS	Inventory Management System
GUI	Graphical User Interface
MVC	Model-View-Controller
SRS	Software Requirements Specification
Product	An item in the inventory that is tracked for stock levels
Reorder Level	The minimum quantity threshold at which a product reorder should be initiated
Stock Alert	A notification indicating that a product's stock level has fallen below its reorder level

1.4 References

- IEEE Standard 830-1998: IEEE Recommended Practice for Software Requirements Specifications
- Java Swing Framework Documentation
- MVC Design Pattern Guidelines

1.5 Overview

The remainder of this document provides a general description of the system, including its major functions, user characteristics, constraints, and dependencies. Detailed requirements are specified in Section 3, followed by external interface requirements, non-functional requirements, and supporting information in the appendices.

2. Overall Description

2.1 Product Perspective

The Inventory Management System will be a standalone desktop application with the following major components:

- **Product Class**: Represents individual inventory items with properties such as name, category, quantity, price, and reorder level.
- **Inventory Class**: Manages the collection of products and provides methods for adding, updating, and removing items.
- **Alert System**: Monitors stock levels and triggers notifications when quantities fall below reorder levels.
- **User Interface**: Provides a graphical interface for users to interact with the system.
- **Main Application**: Coordinates system functions using the MVC pattern.

2.2 Product Functions

The primary functions of the IMS include:

- Managing product information (add, edit, delete)
- Tracking inventory levels
- Generating low-stock alerts
- Sorting and filtering product inventory
- User authentication and access control
- Displaying product information in a user-friendly format

2.3 User Characteristics

The system is designed for the following user types:

- **Business Owners**: Individuals who manage inventory and need tools to track stock efficiently.
- **Inventory Managers**: Personnel responsible for monitoring stock levels and making reordering decisions.
- **Administrative Users**: Staff authorized to add, update, or remove product entries.

Users are expected to have basic computer skills and familiarity with desktop applications and graphical interfaces.

2.4 Constraints

- The application will be developed using Java and the Swing framework.
- The system must be compatible with Windows, macOS, and Linux operating systems.
- Initial implementation will store data in memory using Java collections.
- The system must operate on computers with minimum specifications as defined in section 4.2.

2.5 Assumptions and Dependencies

- Users will have Java 8 or later installed on their computers.
- The system assumes users have basic computer literacy.
- Initial implementation will not include database integration.
- The system is designed for single-user operation, with potential for multi-user functionality in future versions.

3. Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

- **Main Window**: Displays the product inventory table with columns for product details.
- **Product Entry Form**: Dialog for adding and editing product information.
- **Alert Notifications**: Visual indicators for products with low stock.
- **Menu Bar**: Contains options for file operations, product management, and help.
- **Toolbar**: Provides quick access to common functions (add, edit, delete).
- **Status Bar**: Displays system status and notifications.

3.1.2 Hardware Interfaces

The system requires:

- Computer with minimum 2GB RAM

- Intel i3 processor or equivalent
- Monitor with minimum resolution of 1024x768
- Standard input devices (keyboard and mouse)

3.1.3 Software Interfaces

- Java Runtime Environment (JRE) 8 or higher
- Compatible operating system (Windows 7+, macOS 10.10+, or Linux)

3.1.4 Communications Interfaces

- Not applicable for the initial version of the standalone application.

3.2 Functional Requirements

3.2.1 Product Management

- **FR1.1**: The system shall provide functionality to add new products with the following attributes:
    - Product Name (mandatory)
    - Category (mandatory)
    - Quantity (mandatory, non-negative integer)
    - Price (mandatory, positive number)
    - Reorder Level (mandatory, non-negative integer)
- **FR1.2**: The system shall allow users to edit existing product details.
- **FR1.3**: The system shall provide functionality to delete products from the inventory.
- **FR1.4**: The system shall validate all product data to ensure it meets specified criteria before saving.

3.2.2 Inventory Tracking

- **FR2.1**: The system shall maintain an up-to-date list of all products in the inventory.
- **FR2.2**: The system shall display current stock levels for all products.
- **FR2.3**: The system shall automatically update stock quantities when products are added, removed, or modified.
- **FR2.4**: The system shall calculate and display the total value of inventory (price × quantity for all products).

3.2.3 Low-Stock Alerts

- **FR3.1**: The system shall continuously monitor product stock levels.
- **FR3.2**: The system shall generate alerts when a product's quantity falls below its reorder level.
- **FR3.3**: Low-stock alerts shall display:

    - Product name
    - Current quantity
    - Reorder level
    - Shortfall amount (reorder level minus current quantity)
- **FR3.4**: The system shall provide a function to view all products currently below their reorder level.

3.2.4 Product Sorting and Filtering

- **FR4.1**: The system shall allow users to sort the product list by:
    - Product name (alphabetically)
    - Category (alphabetically)
    - Quantity (ascending or descending)
    - Price (ascending or descending)
    - Stock status (whether below reorder level)
- **FR4.2**: The system shall provide filtering capabilities by:
    - Category
    - Stock status (all, in stock, low stock)
    - Price range

3.2.5 User Authentication

- **FR5.1**: The system shall require users to log in before accessing the application features.
- **FR5.2**: The system shall support two user roles:
    - Administrator (full access to all features)
    - Standard User (view-only access to inventory, no add/edit/delete capabilities)
- **FR5.3**: The system shall validate user credentials against stored username and password combinations.

3.2.6 User Interface Requirements

- **FR6.1**: The system shall display a table showing all products with columns for ID, name, category, quantity, price, and reorder level.
- **FR6.2**: The system shall provide clearly labeled buttons for adding, editing, and deleting products.
- **FR6.3**: The system shall visually highlight products with low stock in the inventory table.
- **FR6.4**: The system shall provide confirmation dialogs before executing irreversible actions (e.g., deleting products).
- **FR6.5**: The system shall automatically refresh the display when inventory data changes.

3.3 Performance Requirements

- **PR1**: The system shall support management of up to 1,000 products without performance degradation.
- **PR2**: The system shall respond to user actions within 2 seconds under normal operating conditions.
- **PR3**: The system shall perform inventory updates within 1 second of data entry.
- **PR4**: The system shall generate low-stock alerts within 3 seconds of a product falling below its reorder level.

3.4 Security Requirements

- **SR1**: The system shall authenticate users before granting access to application features.
- **SR2**: The system shall restrict product modification functions to users with administrative privileges.
- **SR3**: The system shall encrypt user passwords when stored in the system.
- **SR4**: The system shall automatically log out inactive users after 15 minutes.

3.5 Software Quality Attributes

3.5.1 Reliability

- **QR1.1**: The system shall operate continuously without failure during business hours.
- **QR1.2**: The system shall maintain data integrity during all operations.

3.5.2 Usability

- **QR2.1**: New users shall be able to use core system functions after no more than 30 minutes of training.
- **QR2.2**: The system shall provide tooltips and help text for all major features.
- **QR2.3**: The user interface shall follow standard desktop application conventions.

3.5.3 Maintainability

- **QR3.1**: The system shall be developed following the MVC architectural pattern.
- **QR3.2**: The system shall use clear, consistent naming conventions and comprehensive documentation.
- **QR3.3**: The system shall be designed with modules that can be modified independently.

3.5.4 Portability

- **QR4.1**: The system shall operate on Windows, macOS, and Linux platforms with Java support.
- **QR4.2**: The system shall not require platform-specific libraries or dependencies.

4. External Interface Requirements

4.1 User Interfaces

The application will use Java Swing to create a graphical user interface with the following components:

- **Main Product Table**: Displays all inventory items with columns for:
    - Product ID
    - Product Name
    - Category
    - Quantity
    - Price
    - Reorder Level
    - Stock Status (visual indicator)
- **Control Panel**: Contains buttons for:
    - Add Product
    - Edit Product
    - Delete Product
    - Check Low Stock
    - Generate Reports (future enhancement)
- **Product Entry/Edit Dialog**:
    - Text fields for product details
    - Validation for numeric fields
    - Save and Cancel buttons
- **Low Stock Alert Display**:
    - List of products below reorder level
    - Option to sort by shortage severity
- **Search and Filter Panel**:
    - Text field for search by name
    - Dropdown for category filter
    - Radio buttons for stock status filter
- **Menu Bar**:

- File menu (Exit)
- Products menu (Add, Edit, Delete)
- Reports menu (Low Stock, Inventory Value)
- Help menu (About, User Guide)

4.2 Hardware Interfaces

The system will run on standard desktop or laptop computers with:

- Minimum 2GB RAM (4GB recommended)
- Intel i3 processor or equivalent
- 100MB free disk space
- Screen resolution of 1024x768 or higher
- Standard keyboard and mouse

4.3 Software Interfaces

The system requires:

- Java Runtime Environment (JRE) version 8 or higher
- Operating System: Windows 7 or later, macOS 10.10 or later, or Linux with graphical desktop environment
- No database system is required for the initial implementation as data will be stored in memory using Java collections

4.4 Communications Interfaces

- Not applicable for the current version of the application.

5. Other Requirements

5.1 Data Storage

- **DR1**: The system shall store product data in memory using Java collections (ArrayList).
- **DR2**: Future versions may incorporate persistent storage using file system or database technology.

5.2 Legal and Regulatory Requirements

- **LR1**: The system shall comply with data protection regulations if user data is stored.
- **LR2**: The system shall maintain accurate inventory records as required for business accounting purposes.

5.3 Localization

- **LOC1**: The initial version will support English language only.

- **LOC2**: The system shall be designed to facilitate future localization efforts.

5.4 Documentation Requirements

- **DOC1**: The system shall include a user manual describing all features and functions.
- **DOC2**: The system shall provide context-sensitive help within the application.
- **DOC3**: The system code shall be documented with clear comments explaining the purpose and functionality of major components.

6. Appendices

6.1 Glossary

Term	Definition
Product	An item tracked in the inventory system
Reorder Level	The minimum stock threshold that triggers a reorder alert
Stock Alert	A notification indicating low stock for a specific product
Category	A classification grouping for similar products
Inventory	The complete collection of products being tracked
Stock Status	Indicator of whether a product is adequately stocked or below reorder level

6.2 Analysis Models

6.2.1 Class Diagram (Conceptual)

- Product Class
    - Attributes: id, name, category, quantity, price, reorderLevel
    - Methods: getters/setters, isLowStock(), getValue()
- Inventory Class
    - Attributes: products (collection)
    - Methods: addProduct(), updateProduct(), deleteProduct(), getAllProducts(), getLowStockProducts()
- AlertSystem Class
    - Methods: checkLowStock(), generateAlert()
- UserInterface Class
    - Methods: displayProducts(), showAddDialog(), showEditDialog(), showAlerts()

6.2.2 Use Case Diagram (Main Functions)

- Actor: Administrator
    - Use Cases: Login, Add Product, Edit Product, Delete Product, View Inventory, Check Low Stock
- Actor: Standard User
    - Use Cases: Login, View Inventory, Check Low Stock

6.3 Issues List

- Future consideration for persistent data storage
- Potential integration with barcode scanner hardware
- Multi-user concurrent access requirements for networked deployment

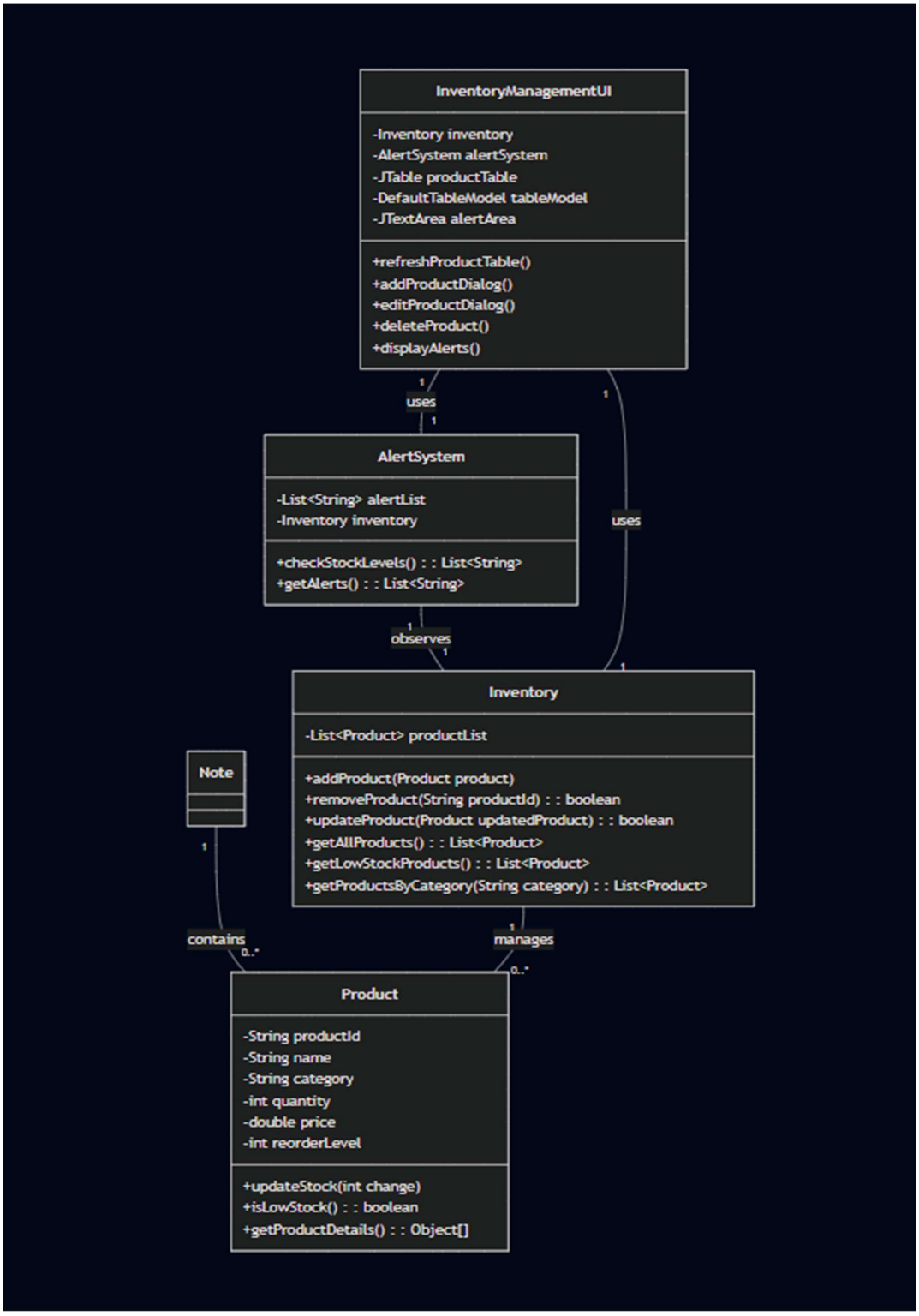
6.4 User Interface Mockups

Placeholder for UI mockups that would illustrate:

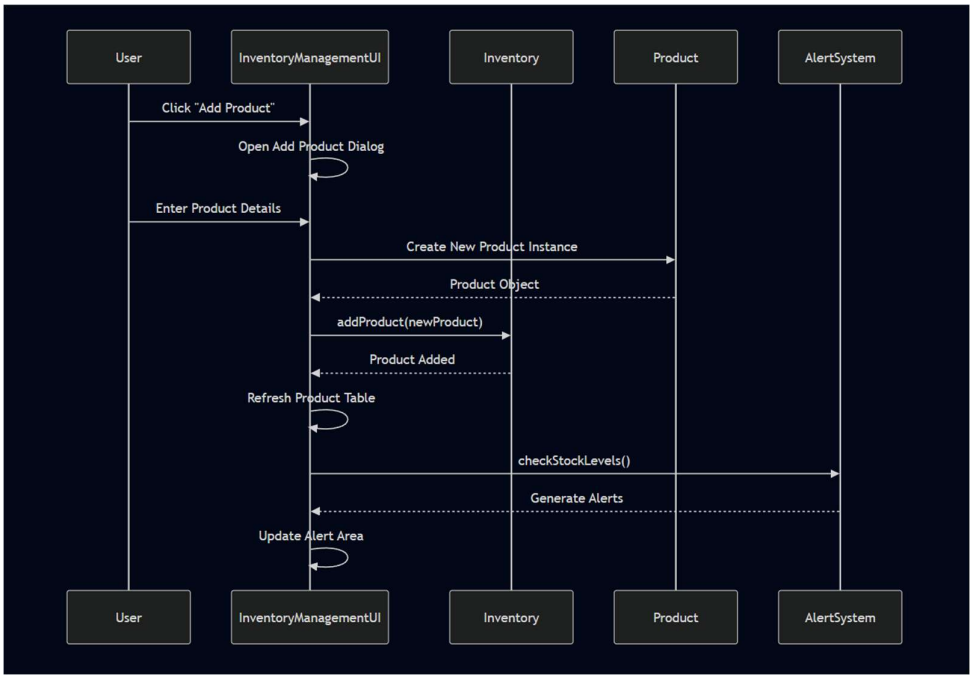
- Main inventory table layout
- Product entry/edit form
- Low stock alert display
- Search and filter panel

UML diagrams:

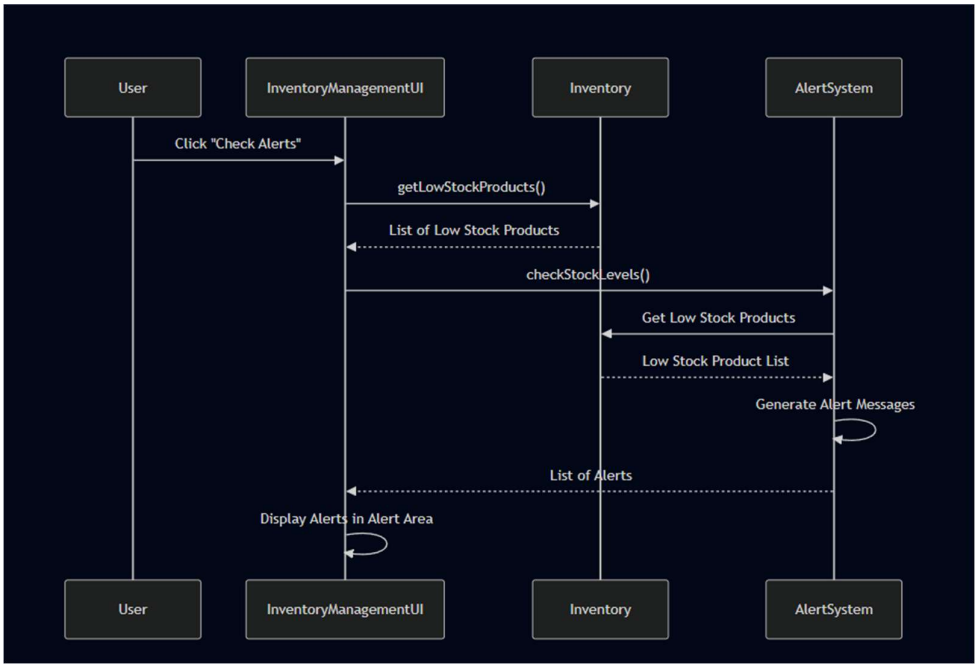
1.Class diagram



2. Product Addition Sequence diagram



3. Low Stock Alert Sequence diagram



Implementation details:

1. Core Class Structure

1.1 Product Class

The Product class serves as the fundamental data model for inventory items:

```
class Product {
   private String productId;      // Unique identifier
   private String name;           // Product name
   private String category;       // Product category
   private int quantity;          // Current stock quantity
   private double price;          // Product price
   private int reorderLevel;      // Threshold for low stock
}
```

Key Methods:

- updateStock(int change): Modifies product quantity
- isLowStock(): Checks if current quantity is below reorder level
- Encapsulation through private attributes and public getters/setters

1.2 Inventory Management

The Inventory class manages product collection using ArrayList:

```
class Inventory {
   private List<Product> productList;  // Dynamic product storage

   public void addProduct(Product product) { ... }
   public boolean removeProduct(String productId) { ... }
   public List<Product> getLowStockProducts() {
      return productList.stream()
         .filter(Product::isLowStock)
         .collect(Collectors.toList());
   }
}
```

Implementation Highlights:

- Uses Java Stream API for efficient filtering
- Supports dynamic product management
- Provides methods for adding, removing, and filtering products

1.3 Alert System

The AlertSystem implements a monitoring mechanism:

```
class AlertSystem {
    private List<String> alertList;
    private Inventory inventory;

    public List<String> checkStockLevels() {
        List<Product> lowStockProducts = inventory.getLowStockProducts();

        return lowStockProducts.stream()
            .map(product -> String.format(
                "LOW STOCK ALERT: %s (ID: %s) - Current: %d, Reorder: %d",
                product.getName(),
                product.getProductId(),
                product.getQuantity(),
                product.getReorderLevel()
            ))
            .collect(Collectors.toList());
    }
}
```

Key Features:

- Generates descriptive low-stock alerts
- Integrates with Inventory to monitor stock levels
- Uses Stream API for alert generation

2. User Interface Design

2.1 GUI Components

- JTable for displaying product list
- JButtons for product management actions
- Dialog-based input for adding/editing products
- JTextArea for displaying alerts

2.2 Event Handling

Implemented action listeners for:

- Adding new products
- Editing existing products
- Deleting products
- Checking low-stock alerts

3. Data Validation and Error Handling

```
private void addProductDialog() {
    try {
        Product newProduct = new Product(
            idField.getText(),
            nameField.getText(),
            categoryField.getText(),
            Integer.parseInt(quantityField.getText()),
            Double.parseDouble(priceField.getText()),
            Integer.parseInt(reorderLevelField.getText())
        );
        inventory.addProduct(newProduct);
    } catch (NumberFormatException ex) {
        JOptionPane.showMessageDialog(
            this,
            "Invalid input. Please check your numbers.",
            "Error",
            JOptionPane.ERROR_MESSAGE
```

```
                );
            }
        }
```

Validation Strategies:

- Number parsing with exception handling
- Input validation through dialog confirmations
- Error messages for invalid inputs

4. Performance Considerations

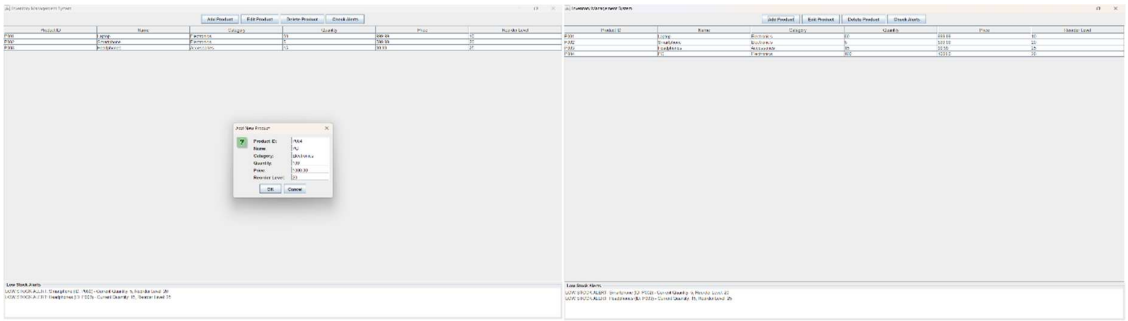
- Lightweight data structures (ArrayList)
- Efficient Stream API operations
- Minimal computational overhead
- Responsive UI design

5. Design Patterns Applied

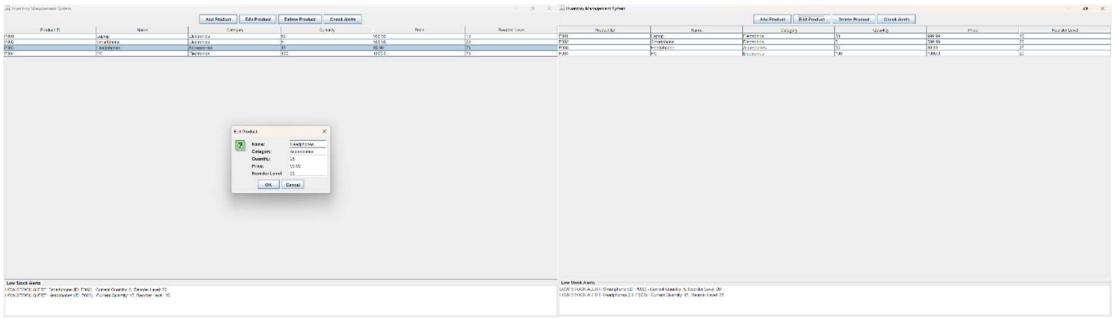
- Encapsulation in Product class
- Observer pattern in AlertSystem
- Model-View-Controller (MVC) architecture

Testing and Sample outputs:

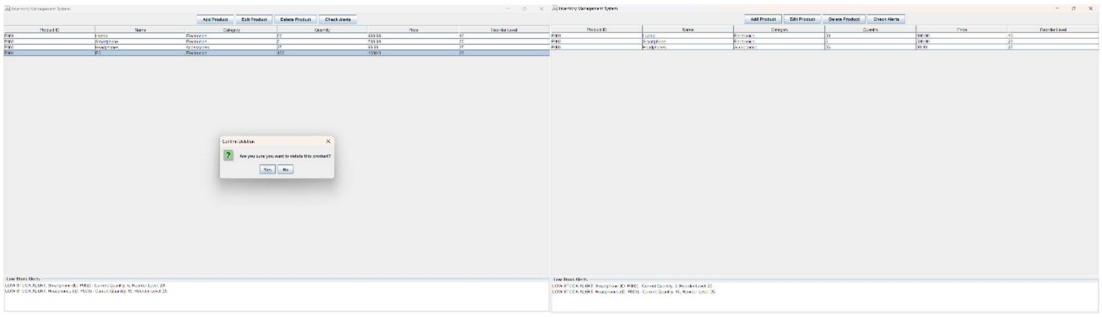
1.Add product:



2.Edit product:



3.Delete product:



4.Low stock alert: