SOFTWARE DESIGN SPECIFICATION

Inventory Management System

G2J

1.0 Introduction

Grocery stores need to process many concurrent customer purchases while ensuring their inventory stays stocked with products in high demand. This software design specification outlines an Inventory Management program that will process the items that are scanned and auto-order to assist with inventory management.

1.1 Goals and objectives

Inventory shall be accurately updated and require a prominent level of reliability and efficiency. The program will use inventory input files to adjust inventory count and order replacement inventory, when required.

1.2 Statement of scope

Major inputs will be the products scanned corresponding to their barcode. The barcodes will be transferred to a text file and the program will read the input text file, add to a counter and auto-order new products once the count meets pre-set conditions. It will also process when new inventory is received and added to the existing stock.

An input file will be created to initially set up desired barcodes, units per case, and threshold for when to initiate an order to replace stock. Once the threshold is met, a re-order list will be generated. Bulk orders, by case, will be written to an output file, to restock product inventory. A report can be generated listing current inventory and order history.

1.3 Software context

The program will have a user interface, which allows products, associated barcodes, units per case and desired stocking requirements to be entered by stakeholders/software customers before implementation. This will be used as the program's input file. The software customer will also need to be able to update inventory when new stock is received and added to current stock.

Data storage will be needed using an in-memory data structure to track products, by case and individual units. The program will need to connect to a database to make the system persistent across sessions and take inputs from different register inputs.

The program will need to calculate when additional cases are needed based on the number of units sold and current stock inventory. The inventory will need to be updated and as product stock is updated.

Output will need to display updates on current inventory and pending orders. It will also need to generate reports, as needed of current inventory, weekly account of units sold (by case) and orders that are pending for delivery.

1.4 Major constraints

We will only be using a small sample of specific products and barcodes as initial inputs to ensure program functionality. There are 100,000+ possible products that the software customer will need, and we will be unable to include that large of a volume during this time limit.

Bulk lists of products and barcodes can be purchased online, but we are not provided with a budget to obtain this information.

Due to time constraints, we will be focused on developing the code for the portion of the program that reads the text input file and updates current inventory to determine when products need to be ordered. It will be assumed that other teams have developed the user interface to create the text file and generate the output report.

2.0 Data design

The software will manage and manipulate the data related to the products, inventory, transactions, and orders to ensure accurate stock tracking and automated re-ordering.

2.1 Data structures

Data objects will be stored in an in-memory data structure for real-time processing and will be continuously used across sessions in the database.

<u>Product Function:</u> Used in program that will read text file to update Stock, create input text file and output file for replacement inventory.

- ProductID: Barcode/UPC
- Name: Product Name
- UnitsPerCase: Number of individual units in a case
- DesiredStockLevel: Target stock level in units or cases
- ReorderThreshold: Minimum stock level that triggers reorder.

<u>Inventory Function:</u> Used in program that will read text file to update Stock.

- InventoryID: Unique identifier used for the inventory entry.
- ProductID: Links to Product

- CurrentStockUnits: Current number of individual units in stock
- CurrentStockCases: Current number of cases in stock
- LastUpdated: Timestamp of the last inventory update

Transaction Function: Used in program that will create text file based on cash register transactions.

- TransactionID: Unique identifier that is used for each sale.
- ProductID: Links to Product
- UnitsSold: Number of units sold in the transaction.
- Timestamp: Date and time of the transaction
- RegisterID: Identifier of the cash register used.

Order Attributes: Used in the program that will generate report of replacement stock inventory.

- OrderID: Unique identifier that will be used for each order.
- ProductID: Linked to the Product
- CasesOrdered: Number of cases that were ordered.
- OrderDate: Date the order was placed.
- Status: Represented as, "Pending" or "Delivered"
- ExpectedDeliveryDate: Tracks the anticipated delivery date for product.

2.2 Database description

A relational database will be used for long-term storage, supporting concurrent usage during peak hours. Relationships among data objects are described using an Entity-Relationship Diagram (ERD) like form.

Product to Inventory: (1:1) One-to-One
Each product will have a single corresponding inventory entry that will track its stock levels.

Product to Transaction: (1:M) One-to-Many
A single product can have or be involved in multiple transactions as it is sold over time.

Product to Order: (1:M) One-to-Many
A single product can have multiple orders placed when the stock falls below the reorder threshold.

Inventory to Transaction: (1:M) One-to-Many
Each inventory entry is updated by multiple transactions as the products are sold.

Inventory to Order: (1:M) One-to-Many
Each inventory entry can trigger multiple orders when the stock levels require a refill.

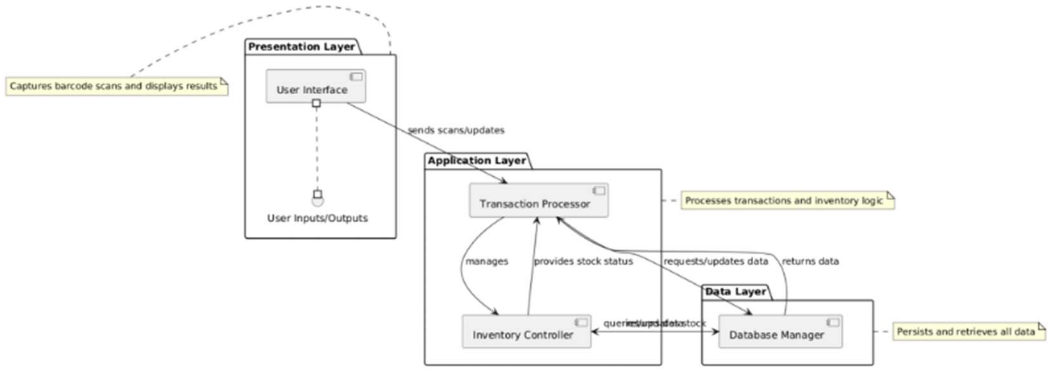
3.0 Architectural and component-level design

The goal of the G2J software project is to deliver a cash register and inventory management system for grocery stores. It is designed to process customer purchases, track inventory in real time, and automate reordering based on stock levels all simultaneously. The software architecture is structured as a modular, three-tier/layer system to meet these goals, ensuring scalability, reliability, and maintainability. It consists of a presentation layer for user interaction, an application layer for business logic, and a data layer for persistent storage which are all interconnected to support the system's core functions: barcode-driven transaction processing, inventory updates, and order generation.

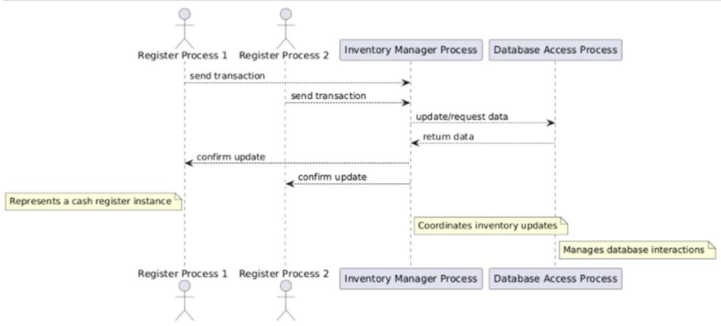
3.1 Architecture diagrams

Various views (logical, process, physical, development) of architecture are presented with descriptions.

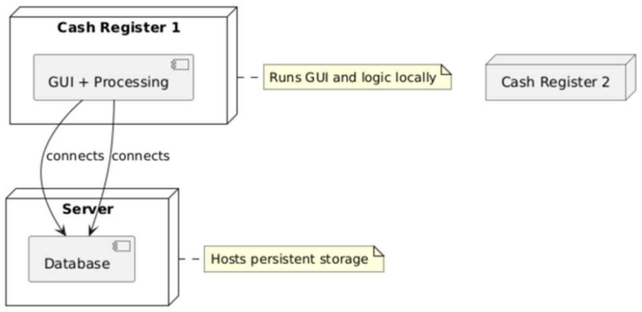
Logical view → This shows the functional components as well as their relationships with each other. The functional components are GUI layer, processing layer, and data layer. In the diagram, the directed arrows indicate interaction flow from GUI to processing to data layers.



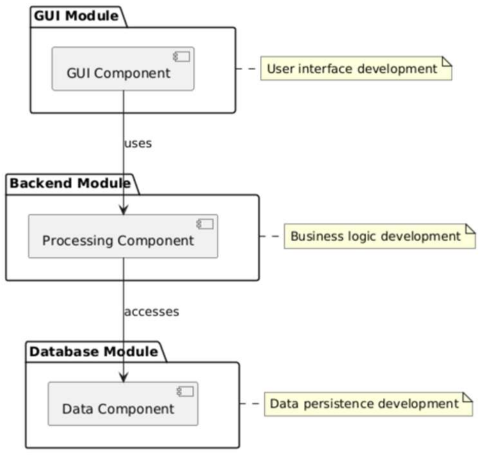
Process view → This shows the interacting processes at runtime. In this case, it will be multiple register processes interacting with a shared inventory manager and database access process. In the diagram, a sequence is diagram is shown with register processes that are simultaneously working to send transactions to a central inventory manager that interacts with the database process.



Physical view → This shows how the hardware and software are distributed in which it maps the software to the corresponding hardware with GUI and processing working together when it comes to cash registers and the database on a central server. In the diagram, two cash register nodes are shown, and they are connected to a server node that is hosting the database.



Development view → This shows how the software is broken down into components for development (the code structure). In this case, the components will be broken into GUI, Backend, and Database. In the diagram, components are grouped into modules with arrows indicating their dependencies.



3.2 Description for Components

For the G2J project plan, the architecture will have three major software components: User Interface (GUI), Inventory Processor, and Data Manager. Each is described below with detailed sub-sections.

3.2.1 Component 1: User Interface

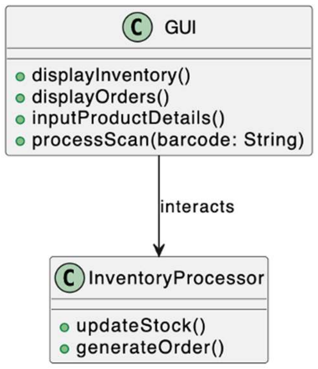
3.2.1.1 Interface description

Input → The system accepts inputs such as barcode scans (through scanner hardware or manual entry), product details (ex: ProductID, Name, UnitsPerCase) that are entered by the stakeholders, and inventory updates (ex: new stock received).

Output → With the input, the system generates outputs including real-time inventory levels (CurrentStockUnits, CurrentStockCases), pending order statuses (OrderID, Status), and reports (ex: weekly sales by case, current inventory).

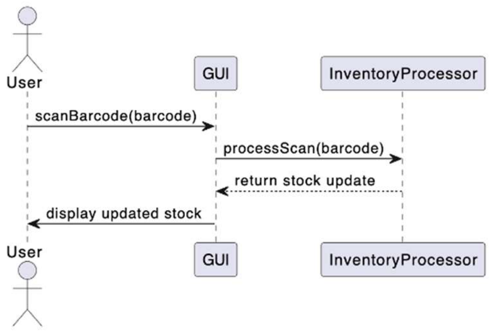
Exceptions → Exceptions will be implemented to handle any system issues such as when an invalid barcode (unknown ProductID) is scanned or when a stock update fails (ex: negative stock after a transaction).

3.2.3.2 Static models



In this diagram, GUI interacts with the InventoryProcessor to relay user inputs and display results.

3.2.3.3 Dynamic models



3.2.1 Component 2: Inventory processing

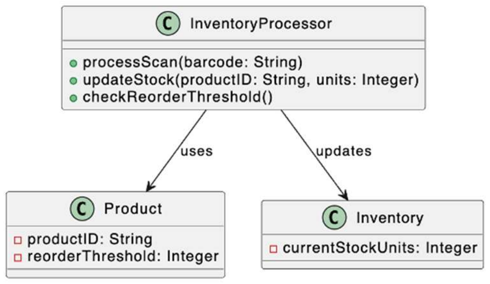
3.2.1.1 Interface description

Input → The system processes inputs like scanned barcodes (ProductID) from the (User Interface aka GUI) and stock updates from either the GUI or the Data Manager.

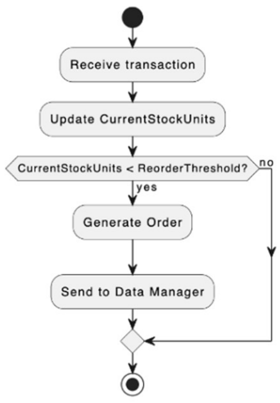
Output → With the input, the system produces outputs such as updated inventory counts (CurrentStockUnits, CurrentStockCases) and generated orders (OrderID, CasesOrdered) when stock drops below the ReorderThreshold.

Exceptions → Exceptions will be implemented to handle any system issues such as when there is stock underflow (attempting to sell more units than available) and order generation failure (ex: due to an unavailable supplier).

3.2.3.2 Static models


InventoryProcessor
processScan(barcode: String)
updateStock(productID: String, units: Integer)
checkReorderThreshold()
uses
updates
Product
productID: String
reorderThreshold: Integer
Inventory
currentStockUnits: Integer

3.2.3.3 Dynamic models



3.2.1 Component 3: Data Manager

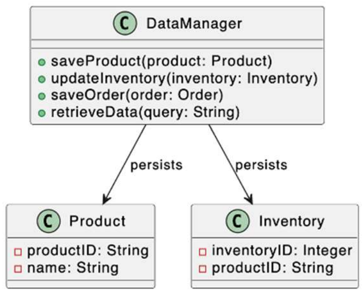
3.2.1.1 Interface description

Input → The system takes inputs such as inventory updates and order details from the Inventory Processor, along with product data entered through or from the GUI.

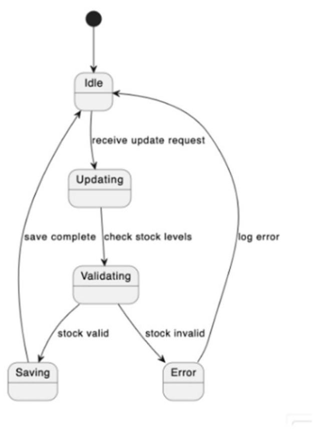
Output → After receiving the input, the system produces outputs like persistent storage of Product, Inventory, Transaction, and Order data, as well as retrieved data for GUI displays or reports.

Exceptions → Exceptions will be implemented to handle any system issues such as when there's a database connection failure or a data integrity violation, such as a duplicate ProductID.

3.2.3.2 Static models



3.2.3.3 Dynamic models



3.3 External Interface Description

The software's interface(s) to the outside world (other software or hardware systems) are described.

The Supplier Ordering System could be a potential future software interface. Although, it is not fully implemented in the initial scope due to project constraints, the point of the design is to send order details (OrderID, ProductID, CasesOrdered) to an external supplier system when the stock levels trigger reordering, in which it would log these as pending orders for manual processing. If it is developed, it could utilize a REST API or file export (e.g., CSV) as its protocol.

The Database Server is another software interface that connects to a relational database (e.g., MySQL, SQLite) on a central server to persistently store the Product, Inventory, Transaction, and Order data, using JDBC/ODBC or a native database driver with SQL queries for CRUD operations.

4.0 User interface design

A description of the user interface design of the software is presented.

4.1 Description of the User Interface

The user interface (UI) for the G2J inventory management software is designed to be simple and intuitive. The GUI will serve two major roles: (1) initial setup and configuration by the user (e.g., store managers) and (2) ongoing inventory updates and report viewing.

4.1.1 Textual description of the key interface components:

Main Dashboard:

Elements:

A search bar to search for a product by UPC/Product Code, or product name. From there the GUI will navigate to the configuration screen for the desired product.

A button to navigate to the report screen.

Behavior:

Correctly supports product look up and accurately navigates to the configuration and report screens.

Configuration Screen:

Elements:

Modifiable fields to update the products information such as unit price, case price, case size, order frequency and desired stock level may be updated. These updates will be passed to the database for persistent storage.

A button to return to the main dashboard.

A button to save changes.

Behavior:

Correctly updates product information and updates the database when the save button is hit, does not update the database until the updates are saved. Correctly navigate back to the main dashboard when the corresponding button is clicked.

Reports Screen:

Elements:

Displays a list of recent reports. Each report is selectable for viewing.

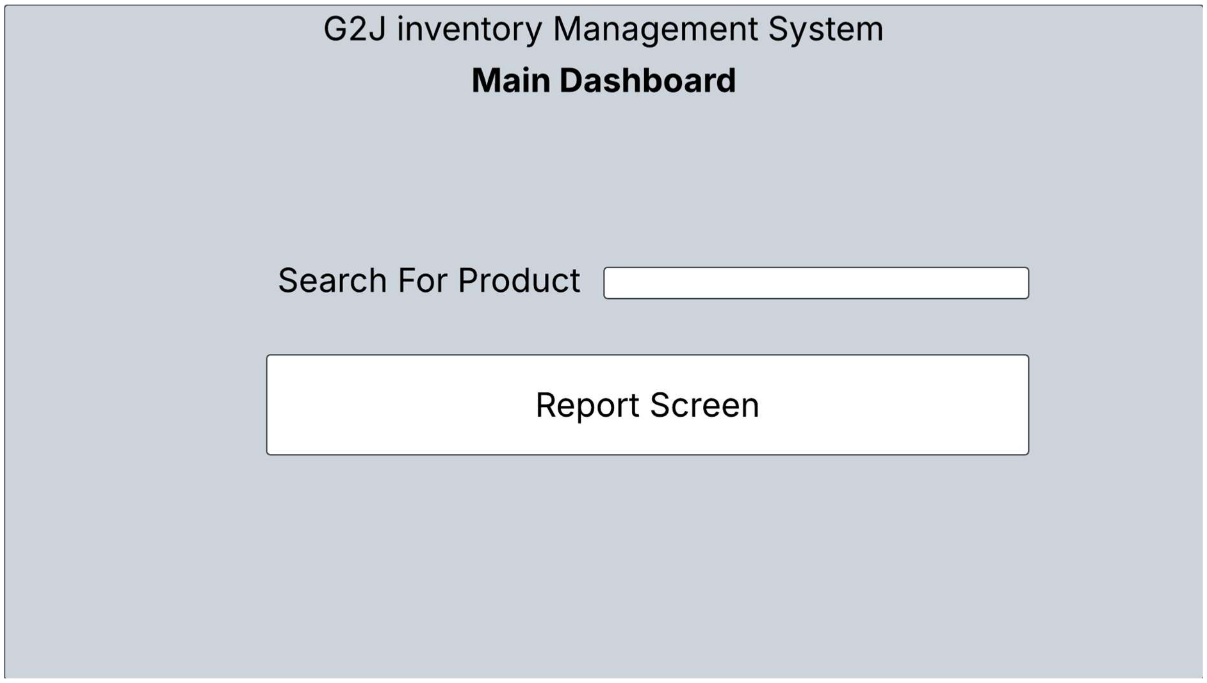
Behavior:

When a report is selected it is opened for viewing.

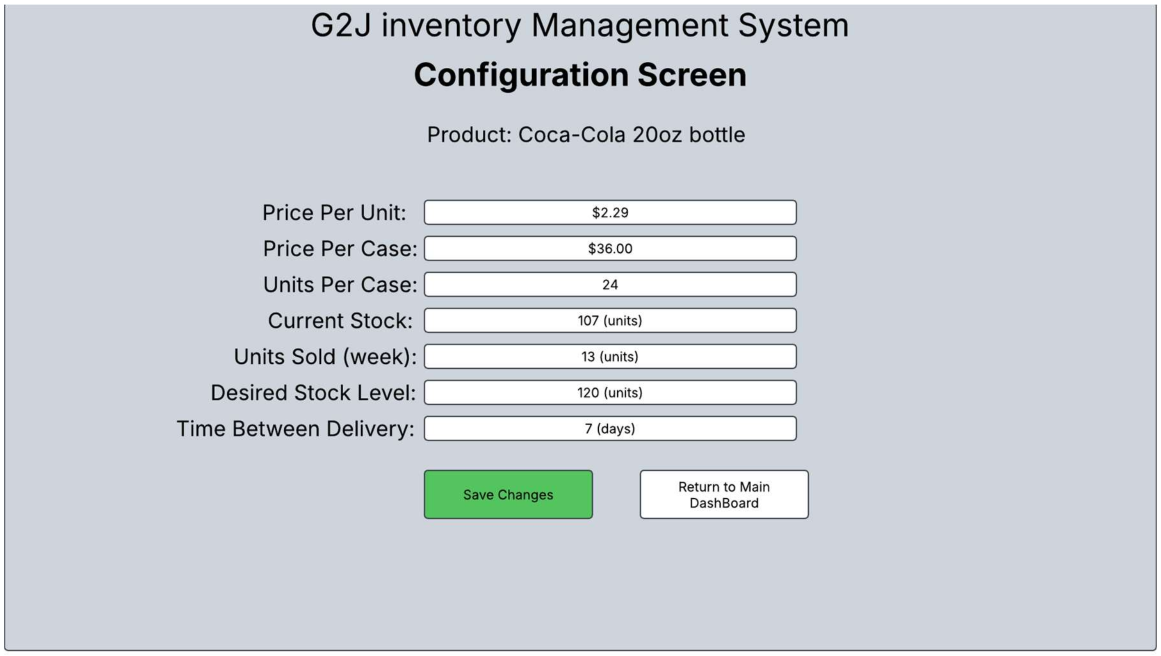
4.1.2 Prototype/ Mock Design

Textless, white box: modifiable field

White box with text: button


G2J inventory Management System

Main Dashboard


G2J inventory Management System

Configuration Screen

Product: Coca-Cola 20oz bottle

Price Per Unit:	$2.29
Price Per Case:	$36.00
Units Per Case:	24
Current Stock:	107 (units)
Units Sold (week):	13 (units)
Desired Stock Level:	120 (units)
Time Between Delivery:	7 (days)

4.2 Interface design rules

Conventions and standards used for designing/implementing the user interface are stated.

The UI design adheres to the following conventions and standards to ensure usability and consistency:

Simplicity and Clarity: The interface prioritizes minimalism, using a clean layout with limited color schemes.

Consistency: All screens use a uniform font button style, and navigation structure to ensure high usability.

Feedback Mechanisms: Confirmation dialogs (e.g., “Are you sure you want to save?”) provide immediate user feedback.

Accessibility: Text is legible (minimum 12pt font size), and controls are keyboard-navigable to support diverse users, aligning with basic WCAG 2.1 guidelines

Responsive Design: The GUI adapts to different screen sizes (e.g., desktop monitors at cash registers or manager workstations), using a modular layout that scales without loss of functionality.

Error Prevention: Input fields include validation (e.g., barcode format checks, numeric-only fields for quantities) to minimize user errors, with tooltips or placeholder text guiding correct usage.

5.0 Restrictions, limitations, and constraints

Special design issues which impact the design or implementation of the software are noted here.

The design and implementation of the software, including the user interface, are subject to the following restrictions, limitations, and constraints derived from the project plan:

Limited Product Scope: The initial UI will only support a small sample of products and barcodes due to time and resource constraints (Section 1.4). This limits the configuration screen’s capacity to handle the full 100,000+ products required by the customer, possibly requiring future scalability enhancements.

Budget Constraint: Lack of funding to purchase bulk products and barcode lists (Section 1.4) restricts the UI to manually entered or sampled data, potentially impacting the realism of testing and configuration workflows.

Risk of Storage Underperformance: If the persistent database fails (Section 2.1), the UI’s inventory updates and reports may display inaccurate data, requiring robust error-handling mechanisms (e.g., offline mode or alerts) in the design.

Scalability Challenge: The initial system size may be underestimated (Section 2.1), potentially slowing UI responsiveness as product volume grows. The design incorporates a scalable architecture, but resource allocation must be dynamically adjusted.

6.0 Appendices

6.1 Requirements traceability matrix

Requirement ID	Requirement Description	Component	Data Structure	Traceability
REQ-001	The system must maintain an accurate inventory count.	Inventory Management Module	Inventory List (Array, Hash Map)	The module updates and tracks inventory levels.
REQ-002	The system should update inventory after every transaction.	Transaction Handler	Transaction Log (List, Queue)	Tracks changes to inventory based on transactions.
REQ-003	The system should reorder items when inventory falls below a predefined threshold.	Order Management Module	Inventory List, Reorder List	Inventory is checked, and reorder decisions are made.
REQ-004	The system must track and manage suppliers for replacement inventory.	Supplier Management Module	Supplier List (Array, Hash Map)	Tracks supplier details and orders from them when necessary.
REQ-005	The system must provide error handling and notifications if inventory cannot be updated.	Error Handling and Notification	Error Log (Array, Queue)	Logs errors and sends notifications if issues arise.

REQ-006	The system should ensure data is persistent across restarts.	Data Persistence Module	Database (Relational DB, File System)	Ensures inventory data is saved and loaded between program runs.
REQ-007	The program should be able to handle large volumes of inventory data efficiently.	Inventory Manageme nt Module	Optimized Inventory List (Hash Map)	Optimized data structures ensure efficient querying and updating of inventory.
REQ-008	The program must have audit logs for all changes made to the inventory	Audit Log Module	Audit Log (Database, File)	All updates to the inventory are logged with timestamps

6.2 Implementation issues

6.2.1 Data Integrity and Consistency

Maintaining accurate and consistent data across multiple components (e.g., inventory count, transactions, orders) can be challenging, especially in cases of failure (e.g., system crashes, power outages).

6.2.2 Concurrency Conditions

When multiple users or systems interact with the inventory (e.g., multiple employees updating stock), conditions can arise, causing inconsistent data (e.g., two users simultaneously updating the same inventory count).

6.2.3 Performance and Scalability

As the volume of inventory data grows, performance might degrade if the program cannot handle large datasets efficiently. Slow lookups or updates could affect the user experience.

6.2.4 Handling Large Inventory Data

Inventory systems may have large volumes of data, and managing large files or database tables can lead to slower performance, especially when the data needs to be updated or accessed frequently.

6.2.4 Handling Data Persistence and Recovery

Ensuring that data is not lost in case of a system crash or unexpected shutdown is crucial for inventory systems.

6.2.5 Error Handling and Validation

User or system errors (e.g., invalid data, network failures) can lead to incorrect inventory updates or failed transactions.

6.2.6 Reordering Logic

Managing reorder logic when inventory falls below a threshold can be complex, particularly if multiple suppliers or ordering strategies are involved.

6.2.7 Integration with Other Systems (e.g., Supplier Systems, Sales Systems)

Integrating with other systems (e.g., supplier databases, sales platforms, or ERP systems) can introduce compatibility and data consistency issues, especially if these external systems change or are unreliable.

6.2.7 Security Concerns

Inventory data is sensitive and must be protected from unauthorized access or tampering. Security breaches could lead to loss of data, manipulation of inventory counts, or theft.

6.2.8 User Interface and Experience

If the inventory management system is user-facing, the user interface (UI) needs to be intuitive and efficient, especially when dealing with large datasets and complex tasks.

6.2.9 Handling Inventory Returns and Adjustments

Managing inventory adjustments, especially for returns or damage adjustments, can be tricky. These transactions may require multiple updates to the inventory and the creation of records for auditing purposes.

6.2.10 Testing and Quality Assurance

Testing an inventory system can be difficult due to the complexity of different workflows (e.g., sales, returns, reordering) and the need for extensive data validation.

6.2.11 Version Control and Change Management

Managing updates to the software and ensuring that changes do not introduce bugs or break existing functionality is an ongoing challenge.