Software Project Final Report: Inventory Management System

G2J: Gloria Nzoh-Ndem, Jamison Brennan, John Kubas

4/20/25

Contents

List of Figures

List of Tables

1. 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. Purpose and Scope

1.1.1 Purpose

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

1.1.2 Scope

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

An input file was 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 is generated. Bulk orders, by case, are written to an output file, to restock product inventory. A report can be generated listing current inventory and order history.

1.2. Product Overview

The program has 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 is used as the program's input file. The software customer will be able to update inventory when new stock is received and added to current stock.

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

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

Program output displays updates on current inventory and pending orders. It also generates reports, as needed of current inventory, weekly accounts of units sold (by case) and orders that are pending for delivery.

1.3. Structure of the Document

This document outlines the development of the Inventory Management Program developed by G2J. It begins with the project's organization and discusses project risk analysis.

Stakeholders are defined along with use cases for requirements specifications, including non-functional requirements.

System architecture and system design are also outlined, including interface, component, and database design. This is followed by a list of test cases and results developed for the program.

The final section discusses the project outcomes and lessons learned during the development of the Inventory Management System.

1.4. Terms, Acronyms, and Abbreviations

CRUD: Create, Read, Update, and Delete
CSV: Comma Separated Values
ERD: Entity-Relationship-Diagram
GUI: Graphical User Interface
ICS: Inventory Control Specialist
JDBC: Java Database Connectivity
MySQL: My Structured Query Language
ODBC: Open Database Connectivity
REST API: Representational State Transfer Application Programming Interface
RM3: Risk Mitigation, Monitoring, Management
SQL: Structured Query Language
SSL: Secure Sockets Layer
UI: User Interface
UPC: Universal Product Code
WCAG: An international standard, developed by World Wide Web Consortium (W3C) Web Accessibility Initiative (WAI)

2. Project Management Plan

2.1. Project Organization

2.1.1 Project Task Set

We followed the Incremental Process Model, allowing us to develop the system in well-defined stages. This model provides flexibility, enabling us to test and validate each functional increment before moving forward.

Framework Activities:

1. Communication: we determined user requirements; John has worked in a grocery store for over 7 years with experience ordering products. He has the experience required to justly represent how the end user would expect the software to behave. He represents our user for this project.
2. Planning: We developed a comprehensive plan for distributing our resources, as well as determining potential risks and errors and how we would mitigate them
3. Modeling: We modeled the input/output handling for the backend including the way UPCs are read and processed, and the database schema.
4. Construction: Handled incrementally in the series of steps defined as follows:
    1. Develop and test the backend data processing.
    2. Develop and test the database.
    3. Integrate the backend data processing and the database to complete the backend.
    4. Develop and test the GUI.
    5. Integrate the GUI and the backend.

Task Set:

1. Requirement Analysis (Weeks 1-6)
2. UPC processing logic (Develop and Test) (Week 7-8)
3. Design Database schema and create the database (Week 8-9)
4. Integrate the UPC processing logic software with the database to form the complete backend (Week 9-10)
5. GUI (Development and Testing) (Week 10-11)
6. Integration of Backend and GUI (Develop and Test) (Week 11-12)

2.1.2 Functional Decomposition

1. Input Handling
2. Read and process UPC codes from input files.
3. Validate input data.
4. Inventory Processing Logic

5. Track item counts against case sizes.
6. Generate re-order lists based on thresholds.
7. Output Handling
8. Write the required orders to an output file.
9. Graphical User Interface (GUI)
10. Build a simple and intuitive GUI for file selection, processing, and result display.
11. System Integration
12. Connect backend logic with the GUI.
13. Testing and Validation
14. Unit testing for backend
15. GUI testing
16. System testing (end-to-end)

2.2. Task Network

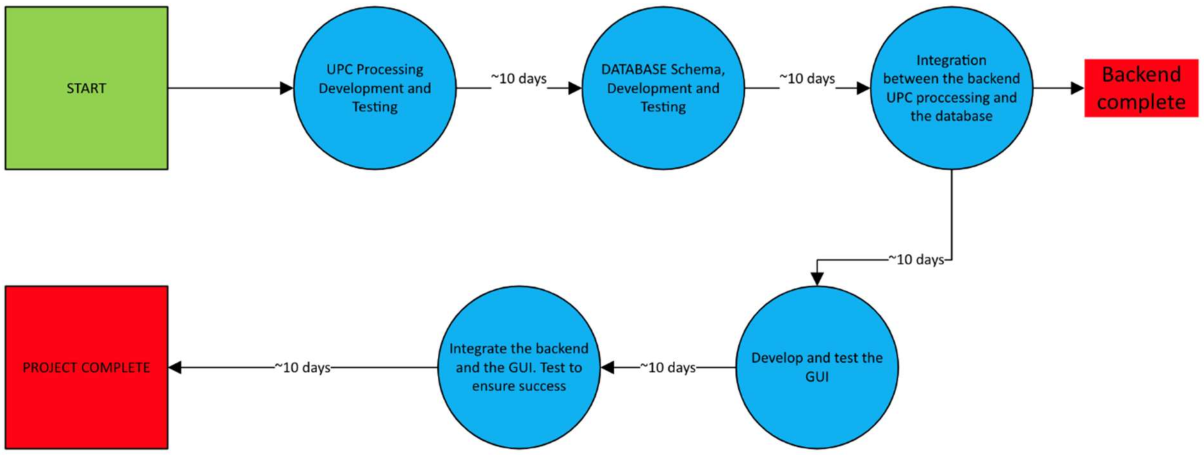

START
UPC Processing Development and Testing
~10 days
DATABASE Schema, Development and Testing
~10 days
Integration between the backend UPC proccessing and the database
Backend complete
~10 days
Develop and test the GUI
~10 days
Integrate the backend and the GUI. Test to ensure success
~10 days
PROJECT COMPLETE

Figure 2-1 Task Network

2.3. Risk Analysis

2.3.1 Project Risks

Software tool underperformance: Product inventory is not tracked accurately it will lead to incorrect ordering of new stock or not ordering enough stock to meet customer needs.

Storage underestimate: Memory is not properly allocated to meet the needs of the number of products and inventory needed by the customer.

Storage underperformance: Persistent storage of inputs to the system fails and inventory is not correctly tracked between sessions.

Size underestimate: The size of the system has been underestimated, slowing down processing of information and program execution.

Table 2-1: Risk Table

Risk	Probability	Impact	RM3 Pointer
Software Tool underperformance	Moderate	Catastrophic	Implement rigorous testing (this includes unit, integration, and end-to-end), real-time monitoring of transactions, and continuous user feedback to refine the functionality.
Storage underestimate	Low	Serious	Optimize the memory allocation, conduct performance stress tests, and ensure efficient data handling to avoid storage limitations.
Storage underperformance	Moderate	Catastrophic	Implement automated backup systems, redundancy mechanisms, and database integrity checks to prevent data loss.
Size underestimate	Low	Moderate	Design a scalable system architecture, run the performance benchmarks, and allocate the resources dynamically based on usage demands.

2.3.2 Overview of RM3

To effectively manage the project risks, we implemented a structured approach that focused on mitigation, monitoring, and management (RM3). Our RM3 strategy includes identification of risks, assessment, and implementation of plans to minimize their impact.

2.3.2.1 Mitigation Strategies

Software Tool Underperformance: Implement detailed testing procedures that include unit and integration testing to validate the accuracy of inventory tracking and order generation. User feedback is required to be gathered to refine functionality regularly.

Storage Underestimate: Memory allocation and data structures require optimization to ensure efficient storage use. Performance testing with increasing data loads to assess system scalability are required to be conducted.

Storage Underperformance: Automated backups and redundancy mechanisms are implemented to ensure constant storage. The database system requires rigorous testing to prevent data loss across the sessions.

Size Underestimate: Scalable architecture is utilized to accommodate potential system growth while the performance benchmarks are set to evaluate the system's efficiency under varying loads.

2.3.2.2 Monitoring Approaches

Performance assessments have been conducted regularly to track the system's efficiency. The logging and error reporting mechanisms are integrated to detect and diagnose potential failures.

Constant testing and validation were performed at each development stage to ensure the system's reliability.

2.3.2.2 Management Plans

The risk assessments and mitigation updates were overseen throughout the development cycle. If the critical risks materialize, fallback solutions such as manual inventory adjustments or temporary overrides can be implemented as they resolve the system's failures.

The stakeholders require updates on risk status and mitigation efforts regularly to ensure alignment with the project's objectives. This structured RM3 approach helped maintain the system's reliability, efficiency, and scalability, while minimizing disruptions in inventory management and transaction processing.

2.4. Hardware and Software Resource Requirements

A description of each major software function and software interface is presented.

2.4.1 Description of Major Functions

Each requirement is uniquely identified based on the major software functions outlined in the project plan.

2.4.1.1 Input Handling

The software reads and process Universal Product Codes (UPCs) from a text file generated by scanned inputs at cash registers. It validates input data to ensure accurate product identification and prevent processing errors.

Inputs are product UPCs scanned via barcode during customer transactions. The system processes these inputs to update inventory counts.

2.4.1.2 Inventory Processing Logic

The software tracks individual item counts against predefined case sizes and generate re-order lists when inventory levels fall below specified thresholds.

The system calculates when additional cases are needed based on units sold and current stock, ensuring automatic re-ordering to maintain stock levels. This will be accomplished via the modulus function.

2.4.1.3 Output Handling

The software will generate a text file to be sent out as an order. It will be capable of displaying updates on current inventory, pending orders, and reports including weekly units sold (by case) and pending delivery orders.

2.4.1.4 Data Storage and Persistence

The software maintains an in-memory data structure to track products (by case and individual units) and connect to a persistent database to store inventory data across sessions.

Ensures data consistency and availability across multiple cash register inputs and sessions.

2.4.1.5 User Interface for Configuration

The software provides a graphical user interface (GUI) allowing stakeholders to enter products, associated barcodes, units per case, and desired stocking requirements prior to implementation, as well as update inventory counts and desired levels when new circumstances arise.

2.4.2 Software Interface Description

The software interface to the outside world is described based on the project plan’s scope and functionality.

2.4.2.1 External Machine Interfaces

Cash register hardware receives barcode scan inputs from multiple registers concurrently, which is added to a text file. The software interfaces with this text file to update the inventory system.

The system must manage large volumes of real-time input during peak hours, ensuring compatibility with standard cash register barcode scanners.

2.4.2.2 External System Interfaces

The software interfaces with a persistent database system to store and retrieve inventory data across sessions.

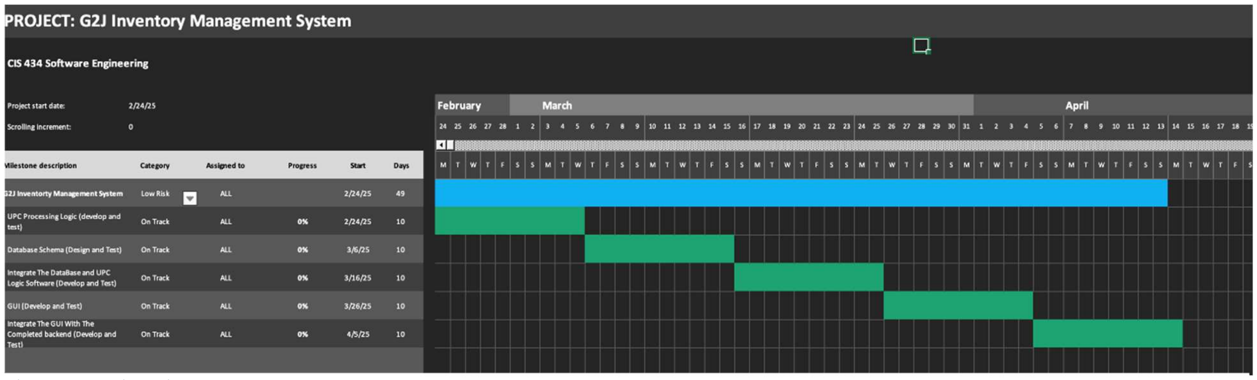
The database ensures data persistence and supports integration with backend logic for inventory tracking and re-ordering.

2.4.2.3 Human Interface

The software provides a graphical user interface (GUI) for stakeholders to configure product details (barcodes, units per case, stocking requirements), updates inventory, and view reports.

The GUI will be simple, designed for file selection, data entry, transaction processing, and result display (e.g., current inventory, pending orders). It supports real-time monitoring and manual inventory updates.

2.5.Deliverables and schedule



PROJECT: G2J Inventory Management System

CIS 434 Software Engineering

Project start date: 2/24/25

Scrolling increment: 0

Milestone description	Category	Assigned to	Progress	Start	Days
G2J Inventorty Management System	Low Risk	ALL		2/24/25	49
UPC Processing Logic (develop and test)	On Track	ALL	0%	2/24/25	10
Database Schema (Design and Test)	On Track	ALL	0%	3/6/25	10
Integrate The DataBase and UPC Logic Software (Develop and Test)	On Track	ALL	0%	3/16/25	10
GUI (Develop and Test)	On Track	ALL	0%	3/26/25	10
Integrate The GUI With The Completed backend (Develop and Test)	On Track	ALL	0%	4/5/25	10

Figure 2-2 Timeline Chart

3. Requirements Specifications

3.1.Stakeholders for the system

Inventory Control Specialist: Mid-level employee who will be updating the program when new inventory is received and responsible for overall inventory management. May have little programming experience.

Cashier: Entry-level employee who will be scanning barcodes to update product purchases. May have no programming or computer experience.

Store Manager: Higher level employee who will have program oversight and will be reviewing inventory updates and product orders. They will need to run reports for updates on store inventory. Managers will require training on the program's functionality and have more experience with inventory management.

3.2.Use Cases

Use case examples are listed below for store employees who will be responsible for using and updating the inventory management system.

3.2.1. Graphic Use Case Model

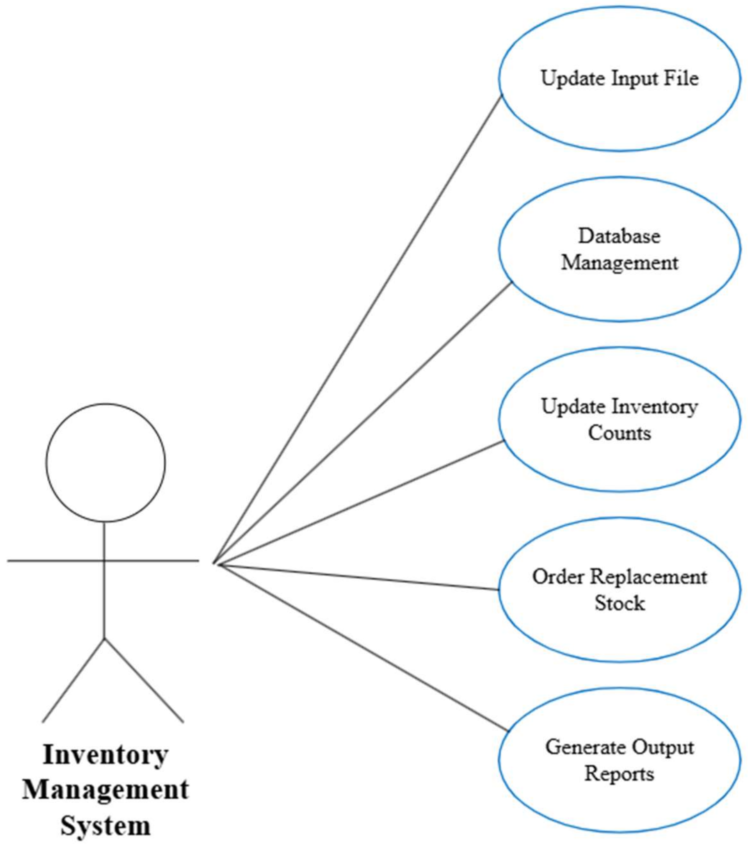


Figure 3-1: Use Case Model

3.2.2. Textual Description for each use case

Table 3-1: ICS Use Case Description

Inventory Management System: Inventory Processing Use Case	
Actors	ICS, Store Manager, System Input File, Inventory Management System
Description	ICS may update the input file with desired product barcodes, units per case and threshold of when new product orders need to be initiated. ICS will transfer input file data to Inventory Management System and update as new Inventory is received. ICS can run reports listing current inventory and expected replacement inventory.
Data	Current Inventory count, Units per Case, new Inventory Count
Stimulus	User command issued by ICS
Response	Confirmation that the Inventory has been updated.
Comments	The ICS must have appropriate security permissions to create the input file and update the Inventory Management System. The store manager should also have the same security permissions as backup to ICS and for proper oversight.

Table 3-2: Cashier Use Case Description

Inventory Management System: Process Barcodes Use Case	
Actors	Cashier, Cash Register, Inventory Management System
Description	Customers will arrive at the checkout with their products for purchase. Cashiers will scan or enter barcodes in the cash register system. The barcodes will be added to a text file which will be used to update the count in the Inventory Management System.

Data	Product barcodes
Stimulus	Cashier scans or enters barcode
Response	Confirmation that barcode is valid and transfer data to text file.
Comments	Cashiers will have access permissions for the system used to scan or enter barcode (cash register).

Table 3-3: Store Manager Use Case Description

Inventory Management System: Report Generation Use Case	
Actors	Store Manager, Inventory Management System
Description	The store manager can generate reports on current inventory, product inventory trends and expected inventory for delivery.
Data	Current Inventory count, Units per Case, new Inventory Count, Report
Stimulus	User command issued by Store manager.
Response	Output file created with contents of report.
Comments	The Manager must have appropriate security permissions to generate reports.

3.3.Rationale for your use case model

The program requires a simple user interface that does not require extensive computer programming knowledge to understand and use correctly. Product inventory will be updated by grocery store employees who may have no experience with computers or programming.

3.4.Non-functional Requirements

3.4.1 Security

The system must ensure secure connections to the MySQL database using SSL or other security protocols. Only authorized users (e.g., ICS, Store Manager) can initiate inventory updates or view reports. Sensitive data (e.g., database credentials, report files) must be encrypted or stored securely.

3.4.2 Performance

The program shall process the barcode file and update inventory within 1 second for up to 100 entries. Reports shall be generated under 5 seconds, even for large inventories.

3.4.3 Reliability

The system must handle errors gracefully, such as missing barcode files or database connection failures. The program must retry or log and failed operations. The inventory updates shall be transactional, meaning either all updates succeed or none do.

3.4.4 Maintainability

The codebase shall be modular and well-documented to support easy updates and bug fixes. Configuration should be externalized.

3.4.5 Testability

The system shall be structured to allow unit, component, and system testing, especially for barcode parsing, database updates and report generation.

3.4.6 Scalability

The program shall be able to handle increasing inventory sizes and multiple barcode files without significant performance degradation.

3.4.7 Availability

The application shall be able to run continuously, or on a schedule and support automatic recovery or manual restart without data loss.

4. Architecture

4.1.Architectural Style

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.

4.2. Architectural model (includes components and their interactions)

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

4.2.1 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.

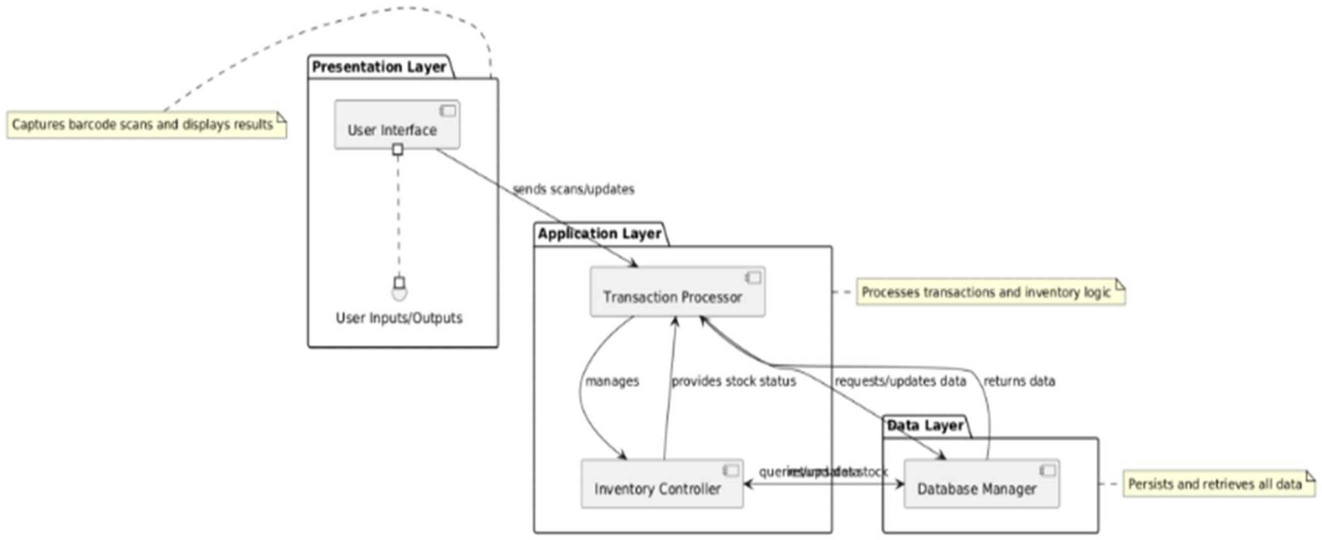


Figure 4-1: Logical View

4.2.2 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.

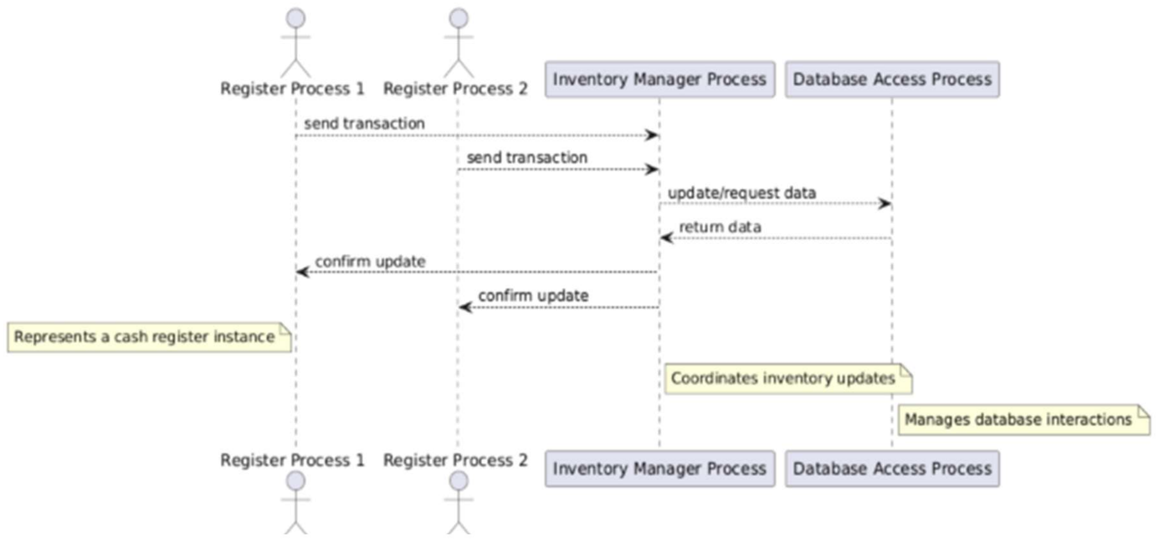


Figure 4-2 Process View

4.2.3 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.

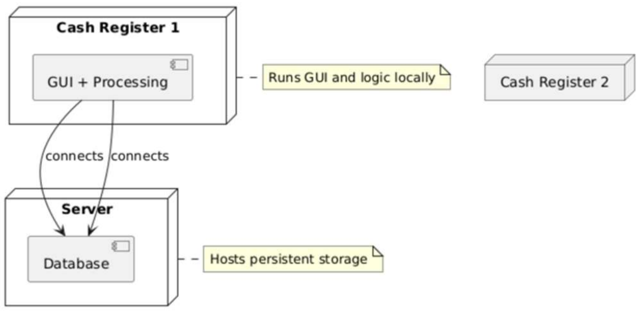


Figure 4-3 Physical View

4.2.4 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.

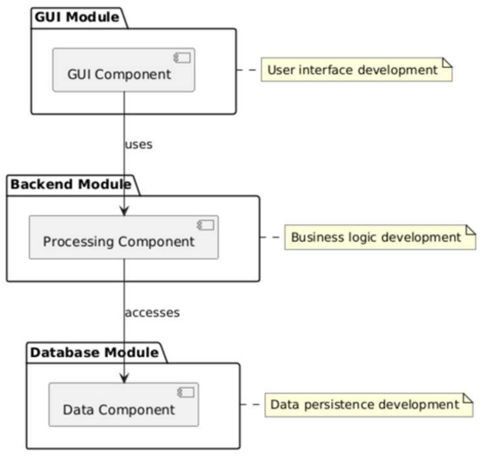


Figure 4-4 Developmental View

4.3. Technology, Software, and Hardware Used

The program code has been written using Python language and MySQL to compile the database. The program's current state runs on personal computer with installations of MySQL and Python support.

4.4. Rationale for your Architectural Style and Model

4.4.1 Scalability

The application layer can scale independently (e.g., multiple processes parsing barcodes), and presentation layer can support more future interfaces without changing the underlying logic or database.

Additionally, multiple instances can be deployed in the application layer to process barcode files in parallel and a data layer can be optimized without affecting business logic or user interface.

4.4.2 Reliability

The layers can be isolated, ensuring if a presentation layer crashes, it does not affect the core processing and the back-end logic still functions. Also, if the data layer is slow or temporarily unavailable, the application layer can queue transactions or retry.

Centralizing the logic in the application layer allows for consistent handling of errors (e.g., invalid barcodes, stock miscounts). Failures in one layer can be logged and reattempted without affecting others, making recovery more manageable.

4.4.3 Maintainability

The presentation layer can be updated without touching the database or core logic. Business rules can also be modified in the application layer without rewriting how data is stored.

Unit testing becomes easier when layers are cleanly separated, and the user interface can be tested independently.

5. Design

5.1. User Interface Design

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

5.1.1 Description of 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.

5.1.2 Textual description of the key interface components

Table 5-1: 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.
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.
Report 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.

5.1.3 Prototype/Mock Design

Textless, white box: modifiable field

White box with text: button

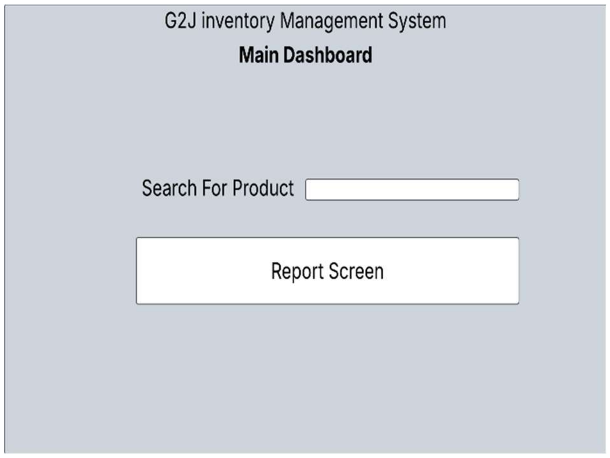
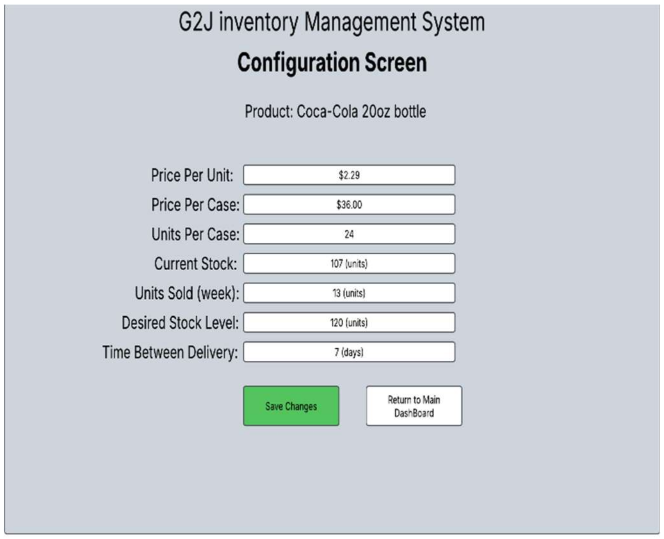


Figure 5-1: User Interface Prototype


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)

Figure 5-2: Interface Configuration Screen

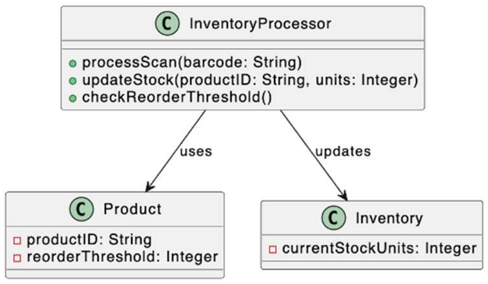
5.2.Components Design (static and dynamic models of each component)

5.2.1 Inventory Processing

Input: The system processes input file of scanned barcodes (product_name) from the user interface (GUI) and stock updates from either the GUI or the Data Manager.

Output: With the input, the system produces outputs (product_data) such as updated inventory counts (current_quantity), case size (case_size) and unit price (unit_price). It generates orders when stock drops below the Reorder Threshold.

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).



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

Figure 5-3: Inventory Processing Static Model

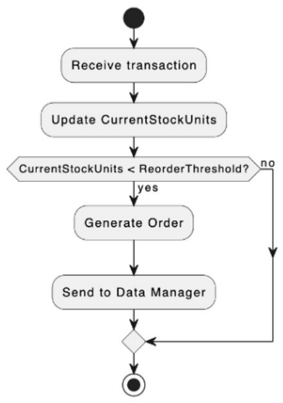


Figure 5-4 Inventory Processing Dynamic Model

5.2.2 Data Manager

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 product_name.

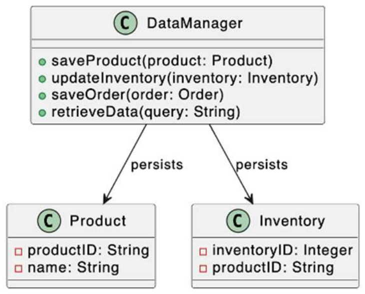


Figure 5-5: Data Manager Static Model

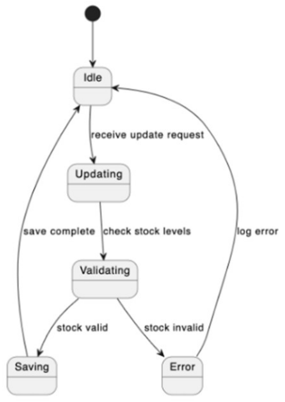


Figure 5-6: Data Manager Dynamic Model

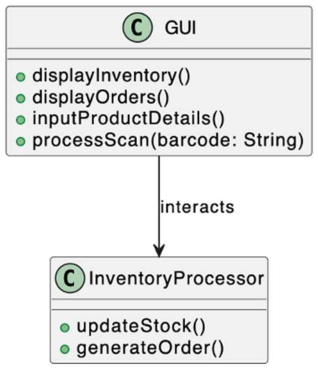
5.2.3 User Interface

Input: The system accepts inputs such as barcode scans (through scanner hardware or manual entry), product details (ex: product_name, description, category, case_size) 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 (current_quantity), pending order statuses, 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 product_name) is scanned or when a stock update fails (ex: negative stock after a transaction).

In Figure 5-7, GUI interacts with the Inventory Processor to relay user inputs and display results.



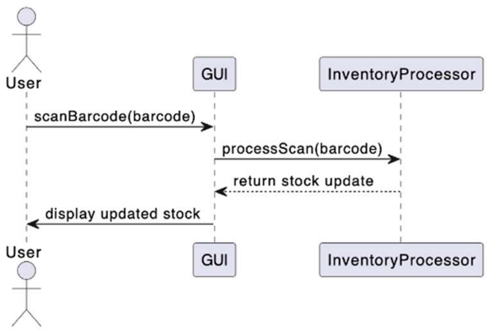


Figure 5-8: GUI Dynamic Model

Figure 5-7: GUI Static Model

5.3.Database Design

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.

<u>Product to Inventory:</u> (1:1) One-to-One

Each product will have a single corresponding inventory entry that will track its stock levels.

<u>Product to Transaction:</u> (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.

5.4.Rationale for your detailed design models

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.

5.5. Traceability from requirements to detailed design models.

Table 5-2: 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	Data Persistence Module	Database (Relational	Ensures inventory data is saved and

	persistent across restarts.		DB, File System)	loaded between program runs.
REQ-007	The program should be able to handle large volumes of inventory data efficiently.	Inventory Management Module	Optimized Inventory List (Hash Map)	Optimized data structures ensure efficient querying and updating of inventory.

6. Test Management

6.1. A complete list of system test cases

6.1.1 Successful Query of an existing item by name:

Table 6-1: Successful Query of an existing item by name

ID	1
Test Input	Scan barcode "123456789012" (CurrentStockUnits = 50)
Expected Output	CurrentStockUnits = 49, transaction recorded, UI updates quantity
Description	Tests Cashier barcode scan and inventory update.

6.1.2 Searching an invalid item by UPC, handled properly:

Table 6-2: Searching an invalid item

ID	**2**
Test Input	Enter UPC= "1234567890123" in the search bar on the dashboard, click "Search"
Expected Output	"No products found matching the term" message appears, "Recent Products" list remains unchanged
Description	Tests searching for a non-existent product by UPC and verifies error handling

6.1.3 Searching for a valid item by UPC:

Table 6-3: Searching for valid item by UPC

ID	**3**
Test Input	From dashboard, select "Milk (1 gallon)" (UPC ="123456789012") from "Recent Products" list
Expected Output	Product configuration screen loads with ProductID = 1, UPC= "123456789012", Product Name="Milk (1 gallon)", Description

	"Whole milk, 1 gallon", Category "Dairy", Current Quantity is 50, Case Size 12, Unit Price $3.99
Description	Tests navigation to product configuration screen and verifies product details

6.1.4 Adding an item with invalid fields, handled properly:

Table 6-4: Adding item with invalid fields

ID	**4**
Test Input	Click "Enter New Item", input UPC "" (empty), Product Name ="Pepsi 20oz bottle", Description "single bottle for cold case by register", Category "Soda", Current Quantity is 24, Case Size 24, Unit Price $2.29, click "Save"
Expected Output	UPC field highlights with "This is an invalid field", clicking "Save" shows error: "Data row too long for column 'UPC' at row 1", product not added, "Recent Products" list unchanged
Description	Tests error handling for invalid UPC when adding a new product

6.1.5 Adding an item with valid fields:

Table 6-5: Adding item with valid fields

ID	**5**
Test Input	Click "Enter New Item", input UPC "41525", Product Name "Pepsi 20oz bottle", Description "single bottle for cold case by register", Category "Soda", Current Quantity 24, Case Size 24, Unit Price $2.29, click "Save"
Expected Output	"New product added successfully" message appears, dashboard "Recent Products" list updates with UPC "41525", Product Name "Pepsi 20oz bottle", Quantity 24
Description	Tests adding a new product via the "Enter New Item" screen

6.2.Traceability of test cases to use cases

Table 6-6: Test Case Traceability

Test Case ID	Use Case Name	Description of Use Case	Traceability Justification
TC01	Process Sale via Barcode Scan	Cashier scans an item, triggering a transaction and inventory update	Verifies that scanning a product updates inventory (CurrentStockUnits -1) and logs the sale
TC02	Search Product by UPC	User searches for a product using a barcode or UPC number	Ensures system gracefully handles invalid UPC searches with proper error messaging
TC03	View Product Details	Staff selects a product to view full configuration and details	Confirms that selecting a product via UPC loads complete data into the product config screen

TC04	Add New Product	Inventory staff adds a new item to the catalog	Validates that input validation works properly; test ensures invalid UPC is rejected
TC05	Add New Product	Inventory staff adds a new item to the catalog	Validates that a correctly filled-out form results in successful product addition

6.3. Techniques used for test case generation

6.3.1 Test Environment:

Development Workstation: A MacBook Pro (e.g., 16GB RAM, 2.6 GHz multi-core CPU, macOS) used for code creation, initial testing, and development of the Python-based application and MySQL database interactions. This workstation serves as the primary development and unit testing environment.

Client Terminals: Multiple personal computers (PCs) simulating cash registers, each with a minimum configuration of 8GB RAM, 2.0 GHz CPU, and running Windows, Linux, or macOS. These imitate the cash register hardware described in the external machine interface.

Database Server: A single server with 16GB RAM and a 4-core CPU to host the MySQL database, ensuring robust data persistence and concurrent access.

6.3.2 Tools:

Unit Testing: unittest (Python’s built-in testing framework) is used to test individual components, such as UPC processing logic and inventory updates, executable on the MacBook Pro.

Integration Testing: pytest is used with plugins for testing interactions between the GUI, Inventory Processor, and Data Manager, runnable on the MacBook Pro and client terminals.

Performance Testing: Apache JMeter is used to simulate concurrent transactions from multiple cash registers, testing the system’s ability to process barcode files within 1 second for up to 5-10 entries.

Database Testing: MySQL Workbench and custom SQL scripts are used which makes it accessible from the MacBook Pro, to validate data integrity, persistence, and transactional updates.

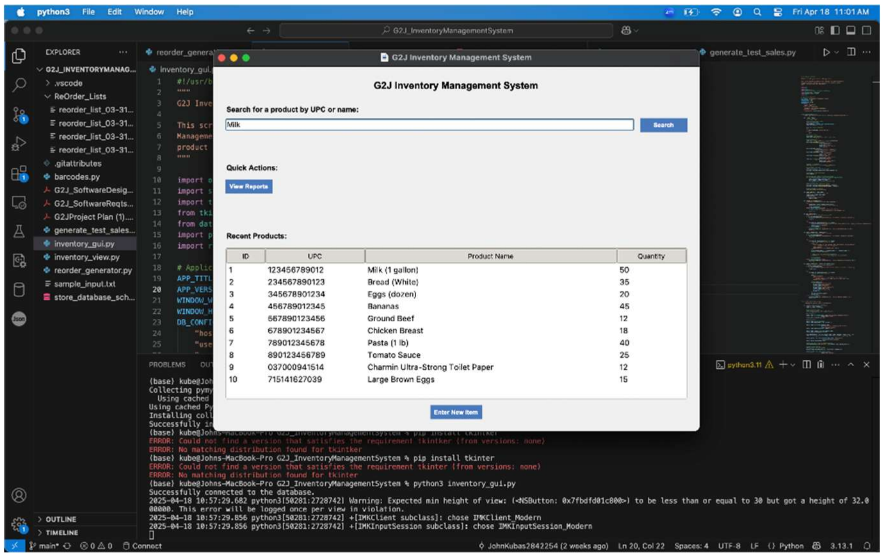
GUI Testing: PyQtTest or Selenium (if a web-based GUI is implemented) is used to automate user interface interactions, such as product entry and report generation, executable on the MacBook Pro.

Simulators: These are used to custom Python scripts, and they are developed and executed on the MacBook Pro, to emulate barcode scanner inputs, generating text files with UPC codes to mimic cash register outputs.

Test Files: These are in the format of CSV files containing sample product data and transaction logs, created and managed on the MacBook Pro.

6.4. Test Results and Assessments

6.4.1 Successful Query of an existing item by name:



G2J Inventory Management System

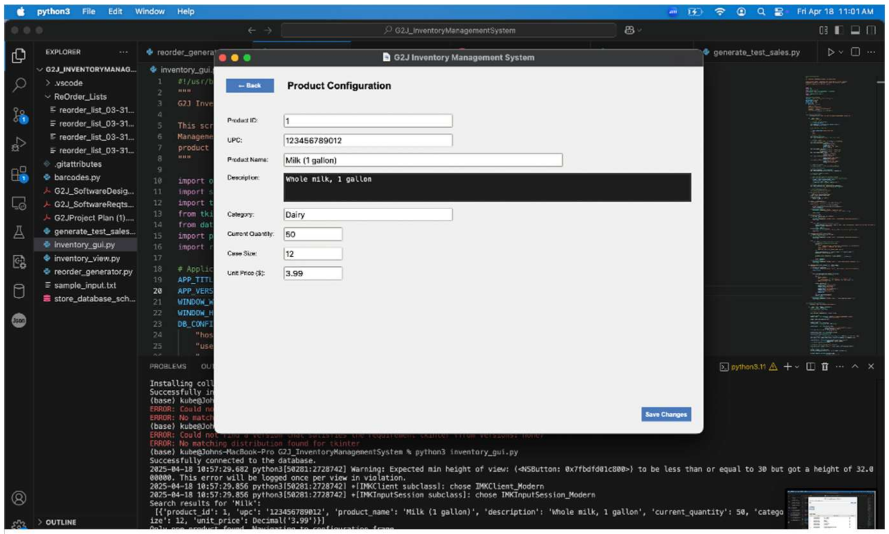
Search for a product by UPC or name: Milk

Recent Products:

ID	UPC	Product Name	Quantity
1	123456789012	Milk (1 gallon)	50
2	234567890123	Bread (White)	35
3	345678901234	Eggs (dozen)	20
4	456789012345	Bananas	45
5	567890123456	Ground Beef	12
6	678901234567	Chicken Breast	18
7	789012345678	Pasta (1 lb)	40
8	890123456789	Tomato Sauce	25
9	037000941514	Charmin Ultra-Strong Toilet Paper	12
10	715141627039	Large Brown Eggs	15

```
(base) kube@Johns-MacBook-Pro G2J_InventoryManagementSystem % pip install tkinter
ERROR: Could not find a version that satisfies the requirement tkinter (from versions: none)
ERROR: No matching distribution found for tkinter
(base) kube@Johns-MacBook-Pro G2J_InventoryManagementSystem % python3 inventory_gui.py
Successfully connected to the database.
2025-04-18 10:57:29.682 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdfd01c800>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
2025-04-18 10:57:29.856 python3[50281:2728742] +[IMKClient subclass]: chose IMKClient_Modern
2025-04-18 10:57:29.856 python3[50281:2728742] +[IMKInputSession subclass]: chose IMKInputSession_Modern
```

Figure 6-1: Successful Query of Existing Item by Name



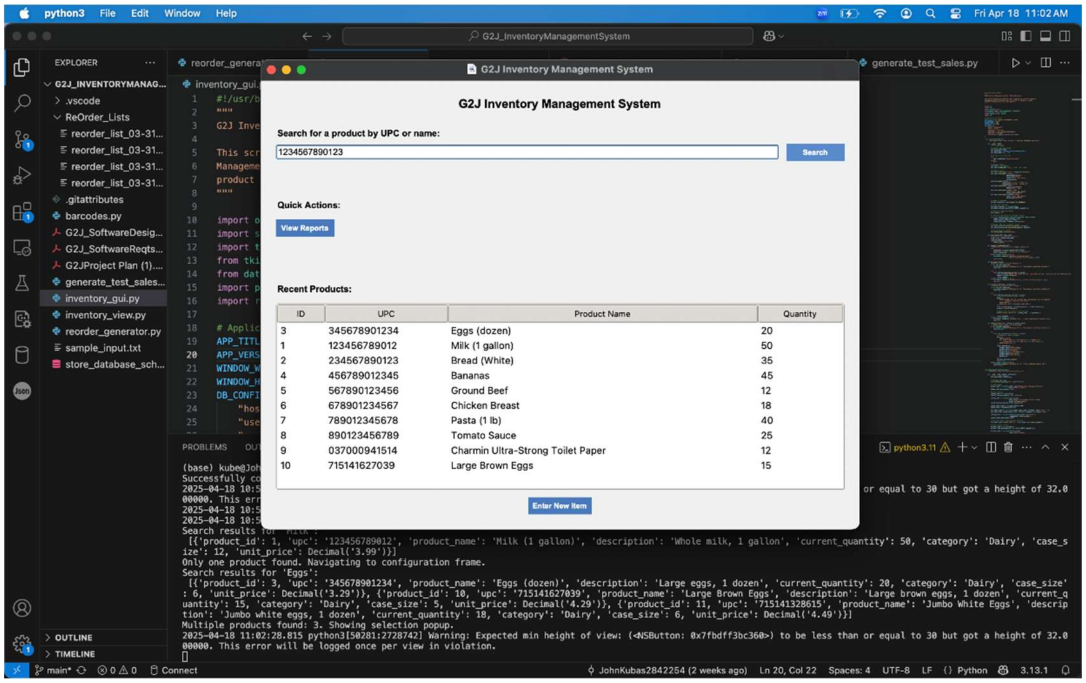
Product Configuration

Product ID: 1
UPC: 123456789012
Product Name: Milk (1 gallon)
Description: Whole milk, 1 gallon
Category: Dairy
Current Quantity: 50
Case Size: 12
Unit Price ($): 3.99

```
(base) kube@Johns-MacBook-Pro G2J_InventoryManagementSystem % python3 inventory_gui.py
Successfully connected to the database.
2025-04-18 10:57:29.682 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdfd01c800>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
2025-04-18 10:57:29.856 python3[50281:2728742] +[IMKClient subclass]: chose IMKClient_Modern
2025-04-18 10:57:29.856 python3[50281:2728742] +[IMKInputSession subclass]: chose IMKInputSession_Modern
Search results for 'Milk':
 [{'product_id': 1, 'upc': '123456789012', 'product_name': 'Milk (1 gallon)', 'description': 'Whole milk, 1 gallon', 'current_quantity': 50, 'catego
ize': 12, 'unit_price': Decimal('3.99')}]
```

Figure 6-2 Successful Query of Existing Item by Name Result

6.4.2 Searching an invalid item by UPC, handled properly:



G2J Inventory Management System

Search for a product by UPC or name:

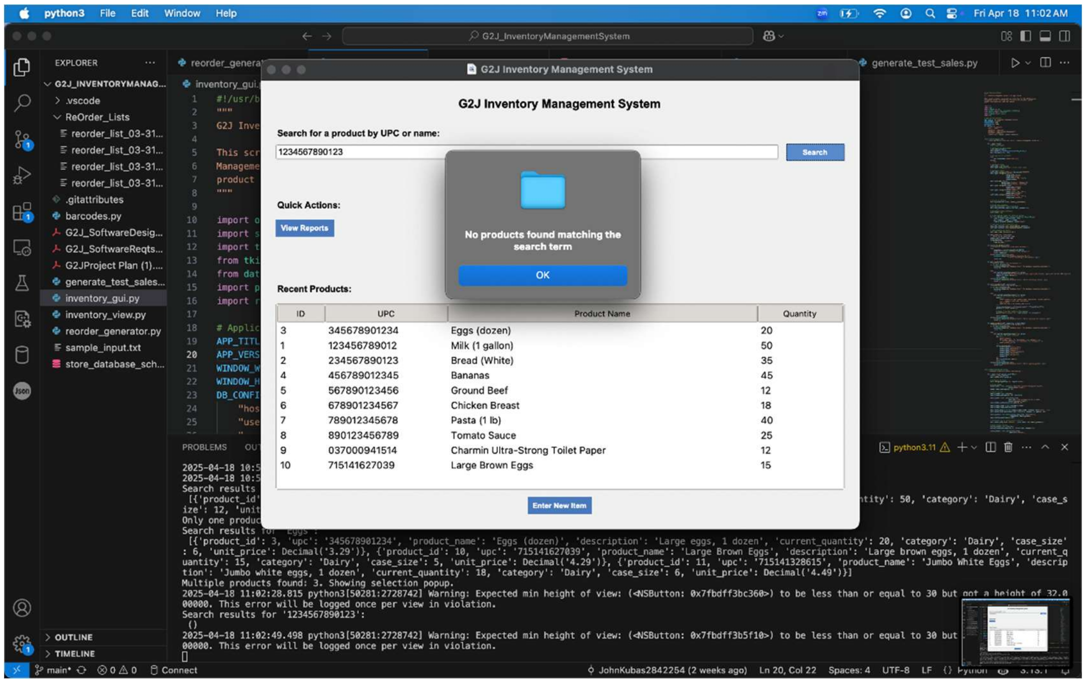
1234567890123

Recent Products:

ID	UPC	Product Name	Quantity
3	345678901234	Eggs (dozen)	20
1	123456789012	Milk (1 gallon)	50
2	234567890123	Bread (White)	35
4	456789012345	Bananas	45
5	567890123456	Ground Beef	12
6	678901234567	Chicken Breast	18
7	789012345678	Pasta (1 lb)	40
8	890123456789	Tomato Sauce	25
9	037000941514	Charmin Ultra-Strong Toilet Paper	12
10	715141627039	Large Brown Eggs	15

```
 [{'product_id': 1, 'upc': '123456789012', 'product_name': 'Milk (1 gallon)', 'description': 'Whole milk, 1 gallon', 'current_quantity': 50, 'category': 'Dairy', 'case_s
ize': 12, 'unit_price': Decimal('3.99')}]
Only one product found. Navigating to configuration frame.
Search results for 'Eggs':
 [{'product_id': 3, 'upc': '345678901234', 'product_name': 'Eggs (dozen)', 'description': 'Large eggs, 1 dozen', 'current_quantity': 20, 'category': 'Dairy', 'case_size'
: 6, 'unit_price': Decimal('3.29')}, {'product_id': 10, 'upc': '715141627039', 'product_name': 'Large Brown Eggs', 'description': 'Large brown eggs, 1 dozen', 'current_q
uantity': 15, 'category': 'Dairy', 'case_size': 5, 'unit_price': Decimal('4.29')}, {'product_id': 11, 'upc': '715141328615', 'product_name': 'Jumbo White Eggs', 'descrip
tion': 'Jumbo white eggs, 1 dozen', 'current_quantity': 18, 'category': 'Dairy', 'case_size': 6, 'unit_price': Decimal('4.49')}]
Multiple products found: 3. Showing selection popup.
2025-04-18 11:02:28.815 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3bc360>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
```

Figure 6-3 Searching an Invalid item by UPC

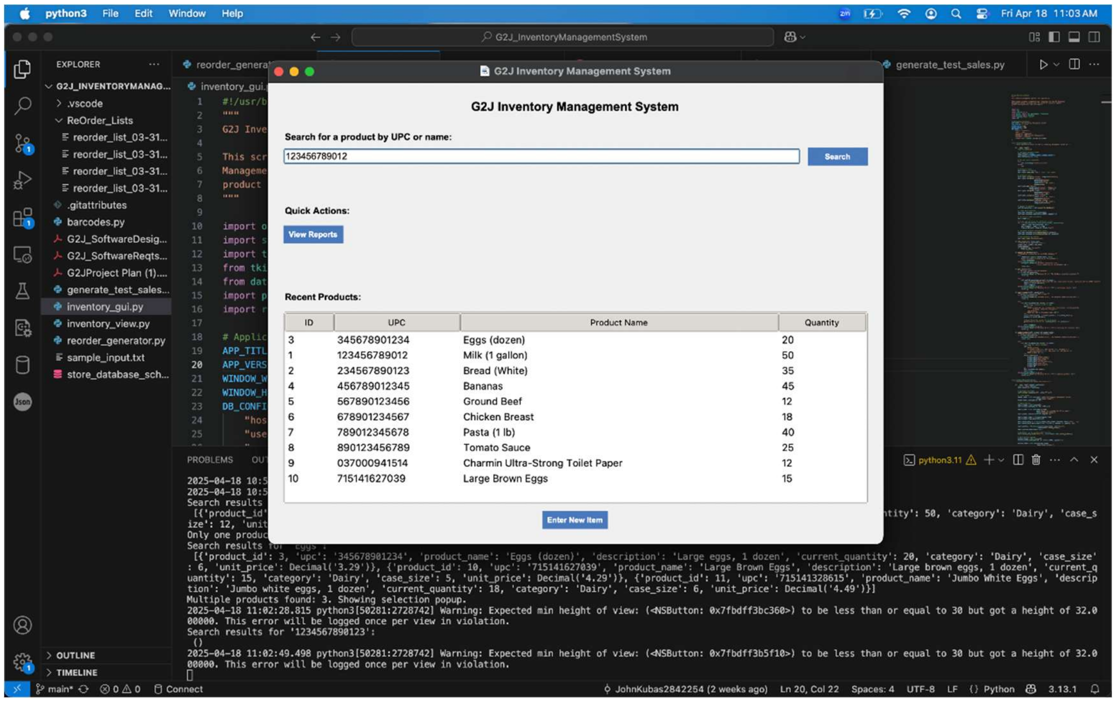


No products found matching the search term

```
Search results for '1234567890123':
 ()
2025-04-18 11:02:49.498 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3b5f10>) to be less than or equal to 30 but
00000. This error will be logged once per view in violation.
```

Figure 6-4: Searching an Invalid item by UPC Result

6.4.3 Searching for a valid item by UPC:



G2J Inventory Management System

Search for a product by UPC or name: 123456789012

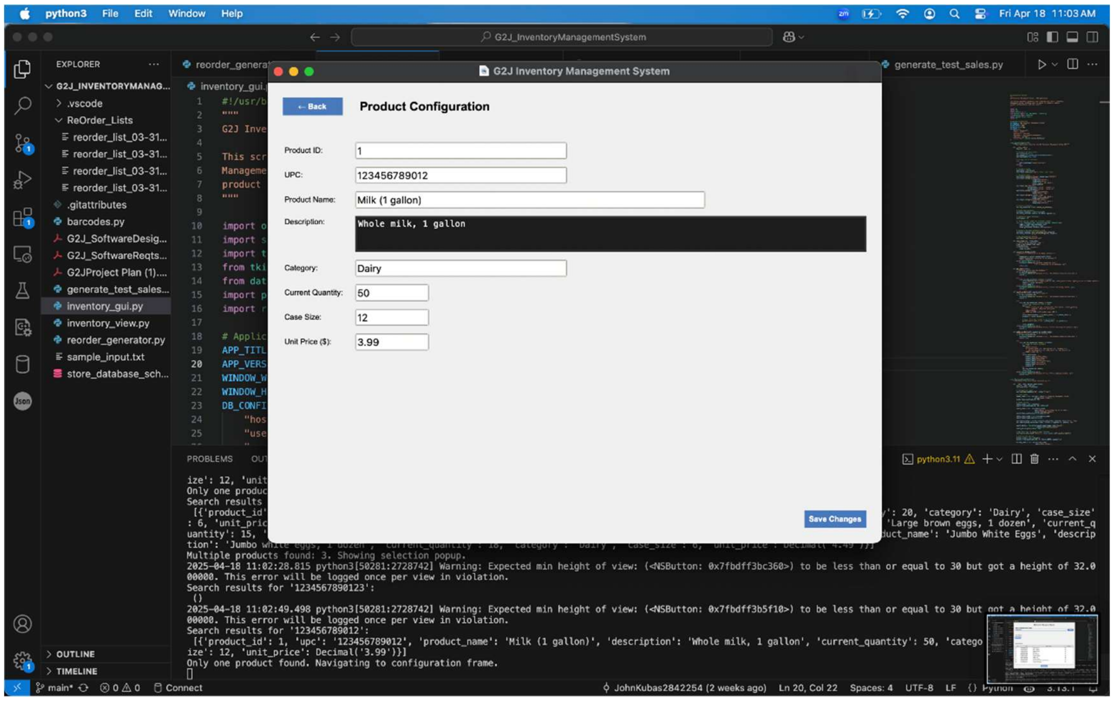
Quick Actions:

Recent Products:

ID	UPC	Product Name	Quantity
3	345678901234	Eggs (dozen)	20
1	123456789012	Milk (1 gallon)	50
2	234567890123	Bread (White)	35
4	456789012345	Bananas	45
5	567890123456	Ground Beef	12
6	678901234567	Chicken Breast	18
7	789012345678	Pasta (1 lb)	40
8	890123456789	Tomato Sauce	25
9	037000941514	Charmin Ultra-Strong Toilet Paper	12
10	715141627039	Large Brown Eggs	15

```
 [{'product_id': 3, 'upc': '345678901234', 'product_name': 'Eggs (dozen)', 'description': 'Large eggs, 1 dozen', 'current_quantity': 20, 'category': 'Dairy', 'case_size'
: 6, 'unit_price': Decimal('3.29')}, {'product_id': 10, 'upc': '715141627039', 'product_name': 'Large Brown Eggs', 'description': 'Large brown eggs, 1 dozen', 'current_q
uantity': 15, 'category': 'Dairy', 'case_size': 5, 'unit_price': Decimal('4.29')}, {'product_id': 11, 'upc': '715141328615', 'product_name': 'Jumbo White Eggs', 'descrip
tion': 'Jumbo white eggs, 1 dozen', 'current_quantity': 18, 'category': 'Dairy', 'case_size': 6, 'unit_price': Decimal('4.49')}]
Multiple products found: 3. Showing selection popup.
2025-04-18 11:02:28.815 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3bc360>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
Search results for '1234567890123':
 ()
2025-04-18 11:02:49.498 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3b5f10>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
```

Figure 6-5: Searching for valid item by UPC



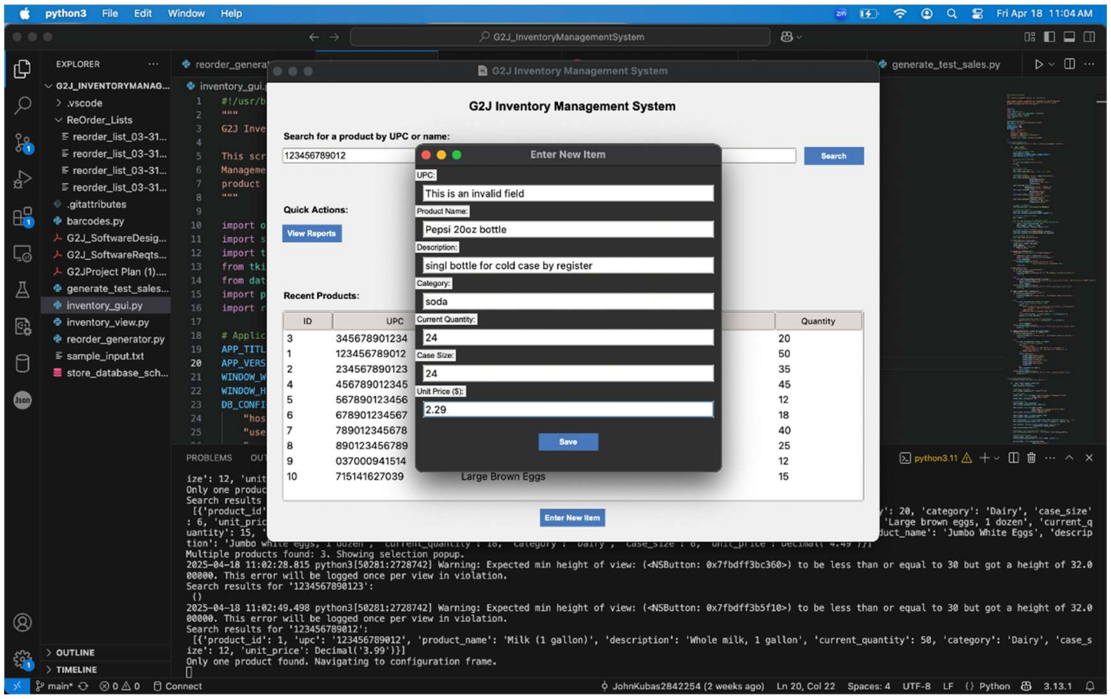
Product Configuration

Product ID:	1
UPC:	123456789012
Product Name:	Milk (1 gallon)
Description:	Whole milk, 1 gallon
Category:	Dairy
Current Quantity:	50
Case Size:	12
Unit Price ($):	3.99

```
Multiple products found: 3. Showing selection popup.
2025-04-18 11:02:28.815 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3bc360>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
Search results for '1234567890123':
 ()
2025-04-18 11:02:49.498 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3b5f10>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
Search results for '123456789012':
 [{'product_id': 1, 'upc': '123456789012', 'product_name': 'Milk (1 gallon)', 'description': 'Whole milk, 1 gallon', 'current_quantity': 50, 'catego
ize': 12, 'unit_price': Decimal('3.99')}]
Only one product found. Navigating to configuration frame.
```

Figure 6-6 Searching for valid item by UPC Result

6.4.4 Adding an item with invalid fields, handled properly:



G2J Inventory Management System

G2J Inventory Management System

Search for a product by UPC or name: 123456789012 Search

Quick Actions: View Reports

Recent Products:

Enter New Item

UPC:	This is an invalid field
Product Name:	Pepsi 20oz bottle
Description:	singl bottle for cold case by register
Category:	soda
Current Quantity:	24
Case Size:	24
Unit Price ($):	2.29

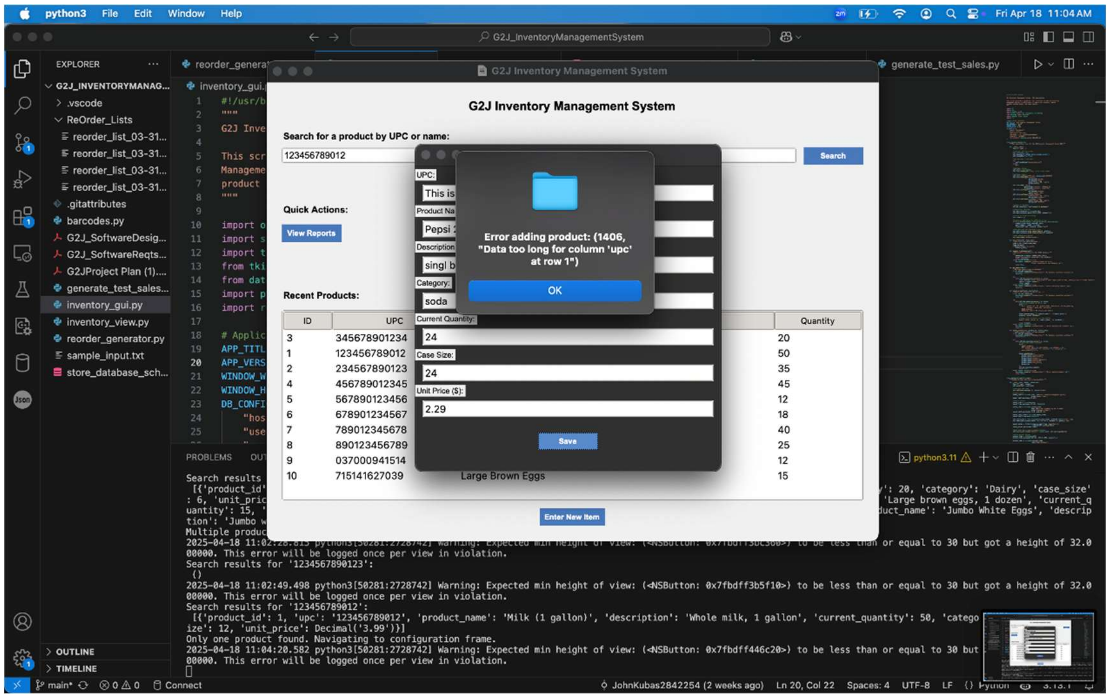
Save

ID	UPC		Quantity
3	345678901234		20
1	123456789012		50
2	234567890123		35
4	456789012345		45
5	567890123456		12
6	678901234567		18
7	789012345678		40
8	890123456789		25
9	037000941514		12
10	715141627039	Large Brown Eggs	15

Enter New Item

```
ize': 12, 'unit
Only one produc
Search results
 [{'product_id'                                                         ': 20, 'category': 'Dairy', 'case_size'
: 6, 'unit_pric                                                         'Large brown eggs, 1 dozen', 'current_q
uantity': 15, '                                                         duct_name': 'Jumbo White Eggs', 'descrip
tion': 'Jumbo white eggs, 1 dozen', 'current_quantity': 18, 'category': 'Dairy', 'case_size': 6, 'unit_price': Decimal('4.49')}]
Multiple products found: 3. Showing selection popup.
2025-04-18 11:02:28.815 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3bc360>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
Search results for '1234567890123':
 ()
2025-04-18 11:02:49.498 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3b5f10>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
Search results for '123456789012':
 [{'product_id': 1, 'upc': '123456789012', 'product_name': 'Milk (1 gallon)', 'description': 'Whole milk, 1 gallon', 'current_quantity': 50, 'category': 'Dairy', 'case_s
ize': 12, 'unit_price': Decimal('3.99')}]
Only one product found. Navigating to configuration frame.
```

Figure 6-7: Adding Item with Invalid Fields


python3 File Edit Window Help
Fri Apr 18 11:04 AM
G2J_InventoryManagementSystem
EXPLORER
G2J_INVENTORYMANAG...
OUTLINE
TIMELINE
generate_test_sales.py
PROBLEMS
python3.11
main*
Connect
JohnKubas2842254 (2 weeks ago) Ln 20, Col 22 Spaces: 4 UTF-8 LF

G2J Inventory Management System

G2J Inventory Management System

Search for a product by UPC or name: 123456789012 Search

Quick Actions: View Reports

Recent Products:

Error adding product: (1406, "Data too long for column 'upc' at row 1")

OK

UPC:	This is
Product Na	Pepsi 2
Description	singl b
Category:	soda
Current Quantity:	24
Case Size:	24
Unit Price ($):	2.29

Save

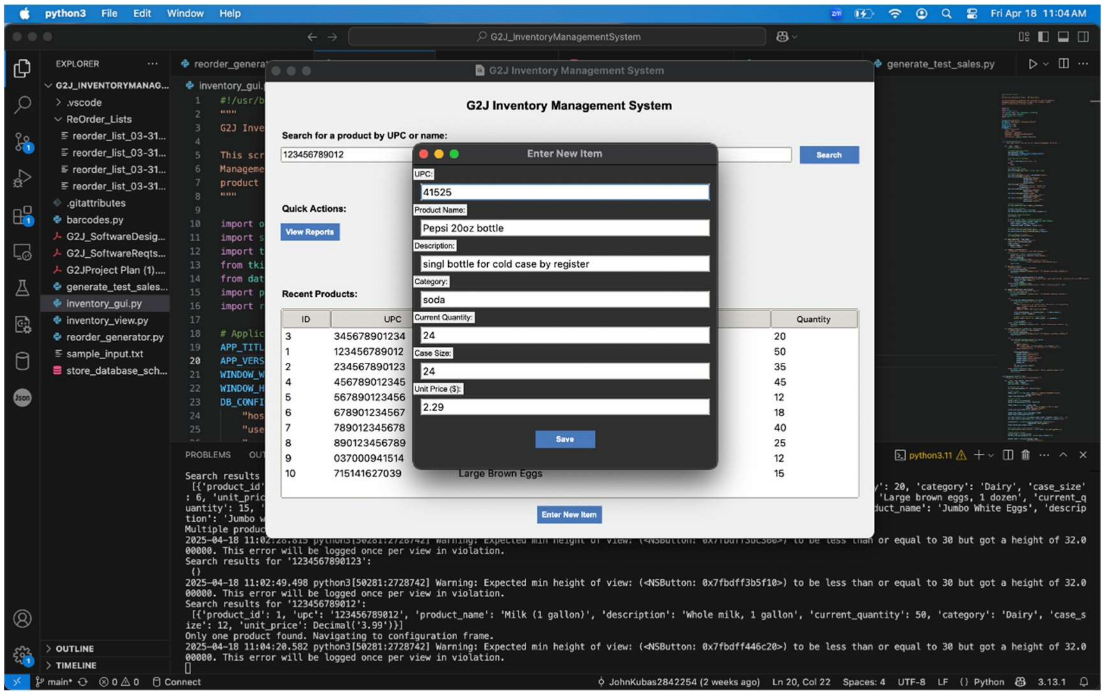
ID	UPC		Quantity
3	345678901234		20
1	123456789012		50
2	234567890123		35
4	456789012345		45
5	567890123456		12
6	678901234567		18
7	789012345678		40
8	890123456789		25
9	037000941514		12
10	715141627039	Large Brown Eggs	15

Enter New Item

```
Search results
 [{'product_id'                                                         ': 20, 'category': 'Dairy', 'case_size'
: 6, 'unit_pric                                                         'Large brown eggs, 1 dozen', 'current_q
uantity': 15, '                                                         duct_name': 'Jumbo White Eggs', 'descrip
tion': 'Jumbo w
Multiple produc
2025-04-18 11:02:28.815 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3bc360>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
Search results for '1234567890123':
 ()
2025-04-18 11:02:49.498 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3b5f10>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
Search results for '123456789012':
 [{'product_id': 1, 'upc': '123456789012', 'product_name': 'Milk (1 gallon)', 'description': 'Whole milk, 1 gallon', 'current_quantity': 50, 'catego
ize': 12, 'unit_price': Decimal('3.99')}]
Only one product found. Navigating to configuration frame.
2025-04-18 11:04:20.582 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff446c20>) to be less than or equal to 30 but
00000. This error will be logged once per view in violation.
```

Figure 6-8: Adding Item with Invalid Fields Result

6.4.5 Adding an item with valid fields:



G2J Inventory Management System

Search for a product by UPC or name:

123456789012

Enter New Item

UPC: 41525

Product Name: Pepsi 20oz bottle

Description: singl bottle for cold case by register

Category: soda

Current Quantity: 24

Case Size: 24

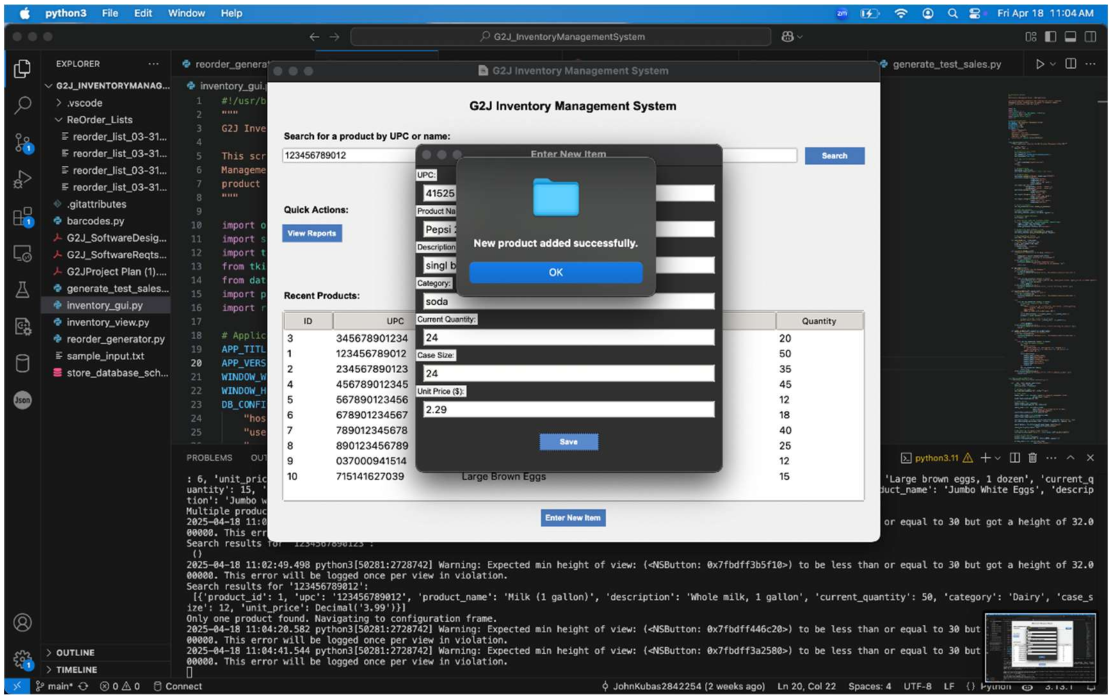
Unit Price ($): 2.29

Recent Products:

ID	UPC		Quantity
3	345678901234		20
1	123456789012		50
2	234567890123		35
4	456789012345		45
5	567890123456		12
6	678901234567		18
7	789012345678		40
8	890123456789		25
9	037000941514		12
10	715141627039	Large Brown Eggs	15

```
2025-04-18 11:02:49.498 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3b5f10>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
Search results for '123456789012':
 [{'product_id': 1, 'upc': '123456789012', 'product_name': 'Milk (1 gallon)', 'description': 'Whole milk, 1 gallon', 'current_quantity': 50, 'category': 'Dairy', 'case_s
ize': 12, 'unit_price': Decimal('3.99')}]
Only one product found. Navigating to configuration frame.
2025-04-18 11:04:20.582 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff446c20>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
```

Figure 6-9: Adding Item with Valid Fields



New product added successfully.

```
2025-04-18 11:02:49.498 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3b5f10>) to be less than or equal to 30 but got a height of 32.0
00000. This error will be logged once per view in violation.
Search results for '123456789012':
 [{'product_id': 1, 'upc': '123456789012', 'product_name': 'Milk (1 gallon)', 'description': 'Whole milk, 1 gallon', 'current_quantity': 50, 'category': 'Dairy', 'case_s
ize': 12, 'unit_price': Decimal('3.99')}]
Only one product found. Navigating to configuration frame.
2025-04-18 11:04:20.582 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff446c20>) to be less than or equal to 30 but
00000. This error will be logged once per view in violation.
2025-04-18 11:04:41.544 python3[50281:2728742] Warning: Expected min height of view: (<NSButton: 0x7fbdff3a2580>) to be less than or equal to 30 but
00000. This error will be logged once per view in violation.
```

Figure 6-10: Adding Item with Valid Fields Result

7. Conclusions

7.1.Outcomes of the project

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. This limits the configuration screen's capacity to handle the full 100,000+ products required by the customer, requiring future scalability enhancements.

Budget Constraint: Lack of funding to purchase bulk products and barcode lists restrict 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, 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, potentially slowing UI responsiveness as product volume grows. The design incorporates a scalable architecture, but resource allocation must be dynamically adjusted.

7.2.Lessons Learned

The most important lesson learned is proper communication and coordination between project team members. Our team initially had conflicting understandings about the scope and overall design of the project, which led to us revising the documentation throughout the project since we all explained aspects of the project differently.

Additionally, having a stronger outline as a basis for our project before we started the code and documentation would have led to less rework during the project.

7.3.Future Development

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 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.

References:

Oracle Corporation. (n.d.). *MySQL* (Version 8.0) [Computer software]. https://www.mysql.com/

Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson Education.

We used ChatGPT, an AI language model developed by OpenAI (ChatGPT, personal communication, April 16, 2025), to generate suggestions for our software documentation.

GitHub Link: Inventory Management System Code