Web-Based Inventory Management System

CT/2021/057 - THENNAKOON B.C.B.M.

SWST 31032 - APPLIED INFORMATION SYSTEMS

Abstract:

This project is the design and implementation of a **Web-Based Inventory Management System** developed using Python Flask as the web framework. The primary motivation of this project was to replace inefficient manual or spreadsheet-based stock tracking methods with a centralized, real-time, and scalable digital solution.

The system provides administrators with comprehensive control over all inventory processes. Key features include full **CRUD (Create, Read, Update, Delete)** operations for defining and managing core inventory attributes, such as **Item Types, Categories, and Storage Locations**. The system ensures accurate, real-time stock level visibility and allows the admin to facilitate essential business operations, specifically the **issuance of items to buyers**. Also, provide an alert system for low stock levels.

Furthermore, the application is equipped with a robust reporting module that allows the admin to **check and download detailed Sales**. Built on a lightweight Flask backend and utilizing a relational database, the system offers a secure and responsive administrative interface. The successful deployment of this solution validates the ability of Python-based micro-frameworks to deliver powerful, user-friendly tools for optimizing inventory control and streamlining business decision-making.

Table of Contents:

Chapter 1: Introduction

1. Background and Motivation:

Effective Inventory management is one of the most critical components of a business that deals with physical items. Traditional methods, like manual counts, spreadsheets, or outdated desktop software, often can lead to inefficiencies such as data inaccuracy, slow processing and transactions, stockouts, overstocking, and poor visibility into asset utilization. These barriers can directly cause high operational costs and reduce customer satisfaction.

The motivation for developing a **Web-Based Inventory Management System** stems from the need for a modern, accessible, and centralized solution. The main purpose is to provide a real-time, multi-user platform accessible from any standard web browser by leveraging the flexibility and scalability of the Python Flask micro-framework.

2. Problem Statement:

The core problem addressed by this project is the lack of a unified and automated platform for comprehensive inventory control. Especially, Admins lack the necessary tools to perform the following tasks.

- **Lack of Centralized Data System:** Item categorization, types, and storage locations are managed separately, leading to inconsistencies.
- **Inefficient Process:** The process of issuing items to buyers and updating stocks to inventory is prone to errors and delayed stock updates.
- **Delayed Reporting:** Generating accurate historical sales and stock movement reports through manual aggregation, delaying analysis and business decision-making. And it can be inaccurate.

3. Project Objectives:

The main objectives of the **Web-Based Inventory Management System** are:

- Develop a Unified Data Module and User-friendly Interfaces to allow admins to create and manage Item Types, Categories, Stocks, and Storage Locations.
- Establish a secure platform for data security by enabling proper authentication methods.
- Implement real-time data processing with a robust database logic that provides accurate, instant stock levels according to any updates.
- Implement User-friendly interfaces and integrate with the database to check all stock-related details and sales-related details.
- Implement the sales reports download feature for Administrators to facilitate sales-related decision-making.
- Implement the notifications checking feature for auditing.

4. Scope of the Project:

Included in Scope:

- Secure login and administrative access control.
- CRUD operations for Item Master data (Types, Categories, Locations, Stoks)
- Real-time stock tracking and inventory level display.
- Transaction recording for item issuance (outbound stock movements).
- Generation and export (download) of sales/issuance reports in CSV format.

Excluded from Scope:

- Integration with external systems
- Advanced features like automatic re-order point calculation, demand forecasting, or supplier management.
- Barcode or QR code scanning functionality for item tracking.
- User roles beyond a single Administrator level.
- Customer friendly Item buying interface.

Chapter 2: Literature Review / Background Study

1. Inventory Management Concepts

Inventory management is the strategic process of ordering, storing, tracking, and selling a company's stock (raw materials, WIP, finished goods) to meet demand efficiently, balancing costs and availability by preventing stockouts (lost sales) and overstocking (tied-up capital). Key concepts include Just-In-Time (JIT), Economic Order Quantity (EOQ), ABC Analysis, setting Reorder Points, using Safety Stock, and tracking Inventory Turnover, all supported by software for accuracy and data-driven decisions to boost cash flow, profitability, and customer satisfaction.

2. Existing Systems and Solutions

- **Manual/Spreadsheet Systems:** Have limitations like being prone to human error, difficult to scale, a lack of real-time visibility, and complex report generation.
- **Commercial Off-The-Shelf (COTS) Solutions:** While these are robust, they often incur high licensing costs, have complex interfaces, and lack flexibility for specialized applications.
- **Desktop Applications:** These applications have limitations for accessibility, multi-user support, and reduced deployment complexity.

3. Technology Stack Review:

I. Backend: Python and Flask

Python has been used for the backend because of its readability, vast library ecosystem, and rapid development capabilities.

Flask has been chosen as the framework, because of its lightweight, modular, and flexible nature. Ideal for small to medium-sized projects like an administrative inventory system, where developers only need core functionalities, not an all-inclusive framework. And allows the developers to choose specific components and libraries (e.g., specific database ORM, templating engine) tailored to the project's exact needs.

II. Database Layer: SQLAlchemy and ORM

Used SQLite because SQLite support for simple development/testing. SQLAlchemy allows Developers to interact with the database using Python objects and classes instead of writing raw SQL queries. This makes the code cleaner, more secure (helps prevent SQL injections), and more manageable.

III. Frontend Technologies

HTML, CSS, JavaScript: Standard web technologies necessary for structuring, styling, and adding interactivity to the user interface.

Chapter 3: System Analysis and Design

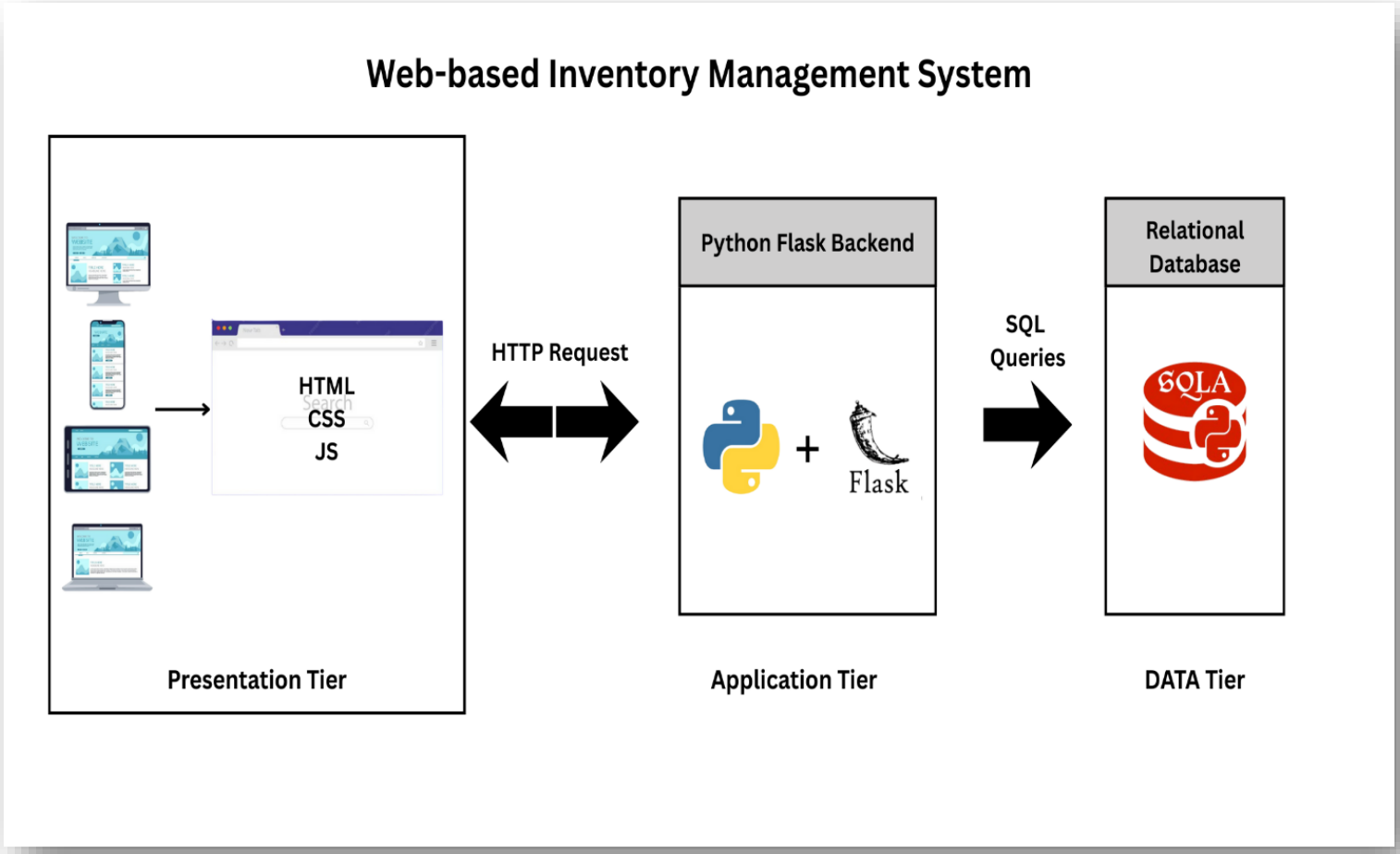
1. **Requirements Analysis:**
    a. **Functional Requirements (Features):**

Requirement ID	Module	Description
FR-001	User Access	The system must implement a secure login mechanism (using password hashing) for the administrator to access the inventory control panels.
FR-002	Master Data	The administrator must be able to perform **CRUD** operations on Item Types, Categories, and Storage Locations. This ensures the foundation for accurate data entry.
FR-003	Inventory	The system must provide a form for adding new stock items, capturing essential details like SKU, initial quantity, category (FK), and location (FK).
FR-004	Stock Levels	The system must display the current quantity-on-hand for every item, dynamically calculated from the initial stock and all recorded transactions.
FR-005	Issuance/Sales	The system must facilitate the issuance of items to buyers. This process involves validating stock availability, recording the transaction details (buyer, quantity), and automatically reducing the corresponding item's stock level.
FR-006	Reporting	The system must generate a queryable report summarizing all item issuance transactions, filterable by date range and category.
FR-007	Reporting	The system must provide functionality to **download** the generated sales/issuance reports in a standardized format, CSV.
FR-008	Reporting	The system must provide functionality to **check** all the notifications for auditing and security processes.

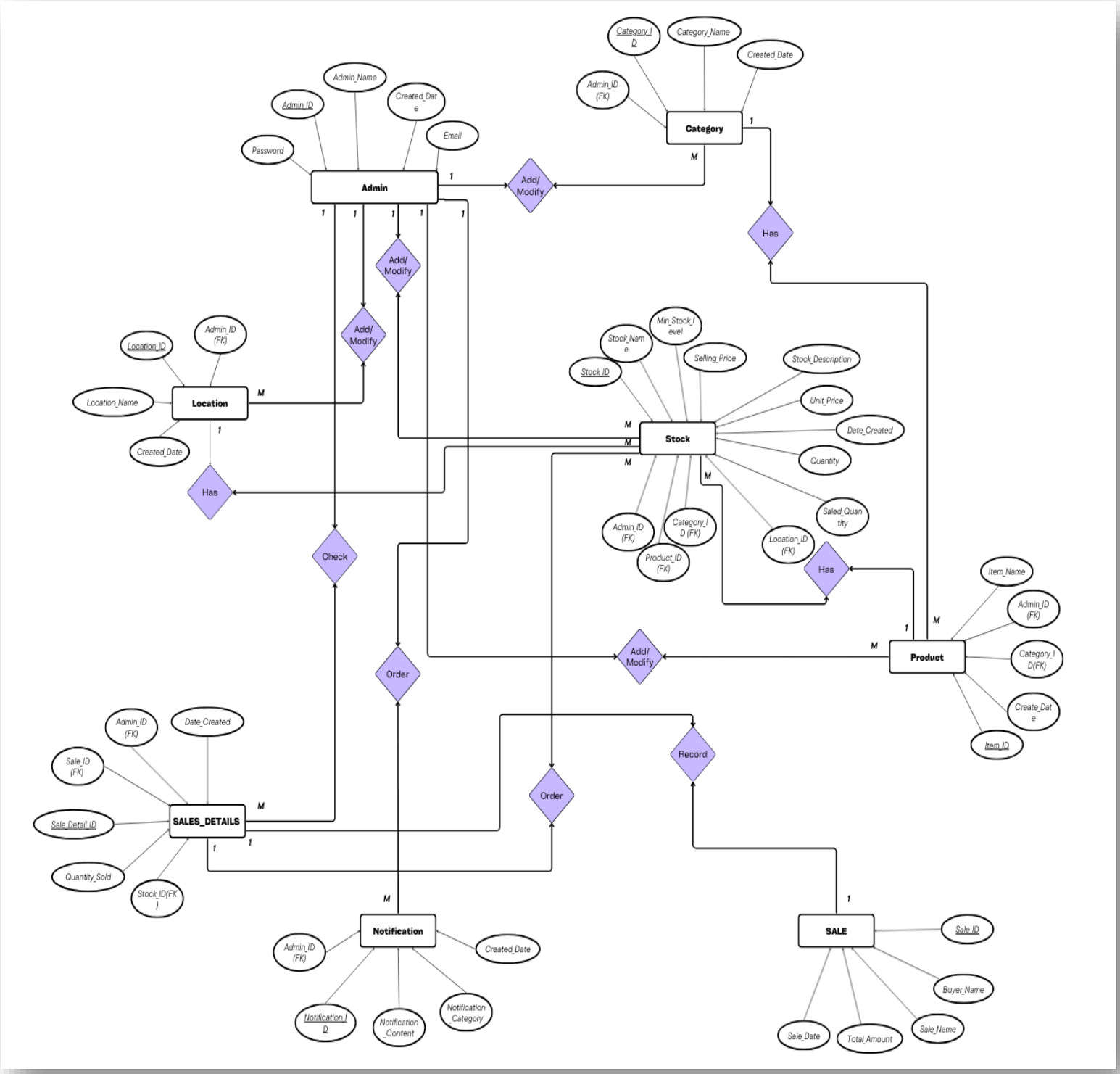
b. Non-Functional Requirements:

- **Performance**: The system must retrieve item lists and generate standard reports in less than 3 seconds for up to 1,000 inventory records.
- **Security:** All administrative communication must be authenticated. Passwords must be stored using secure hashing.
- **Usability:** The administrative interface must be intuitive, featuring clear navigation and a responsive design for easy use on various screen sizes.

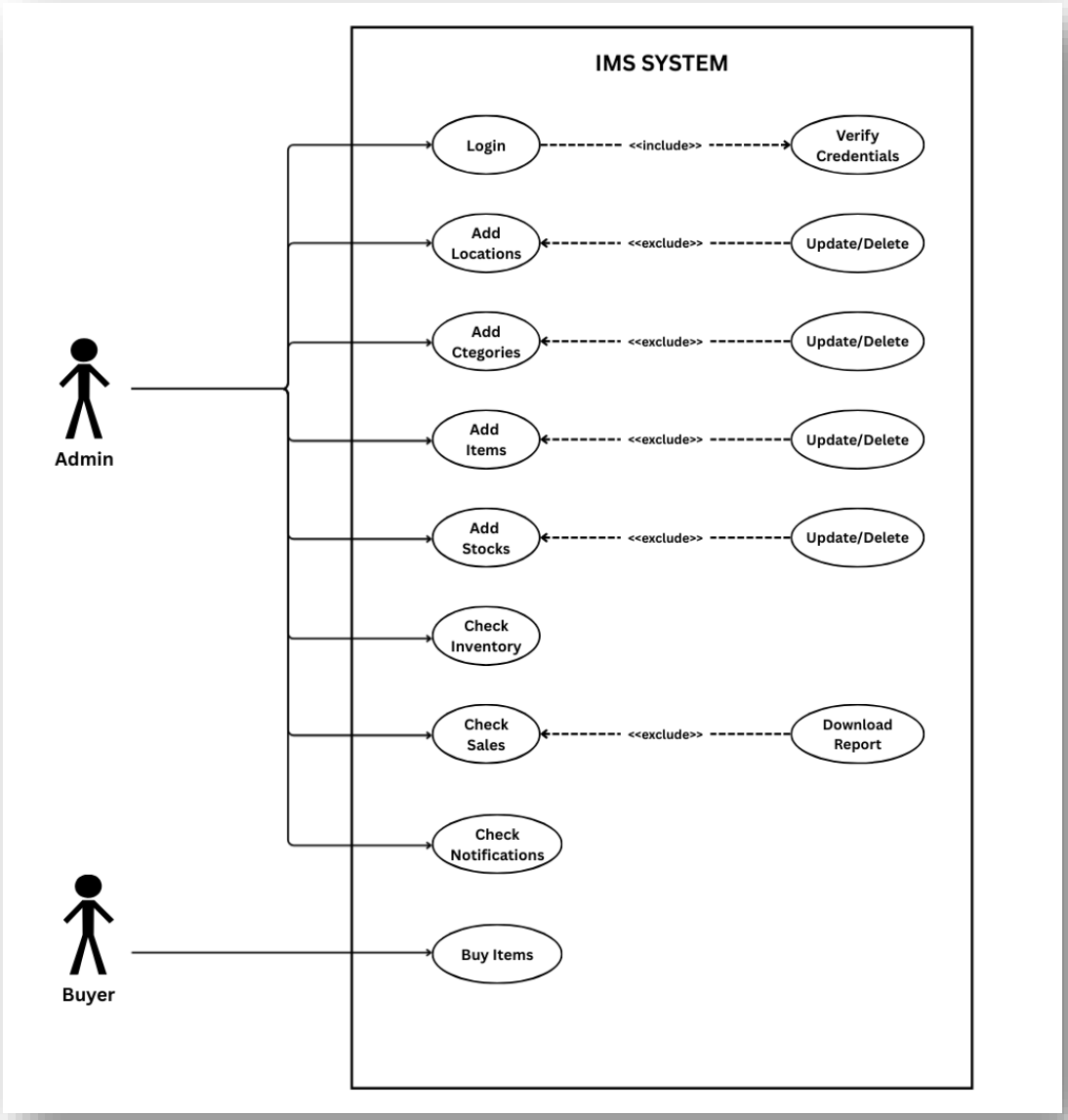
2. **System Design:**
    a. **System Architecture:**



b. Database Design (ER Diagram):



c. Use-case Diagram



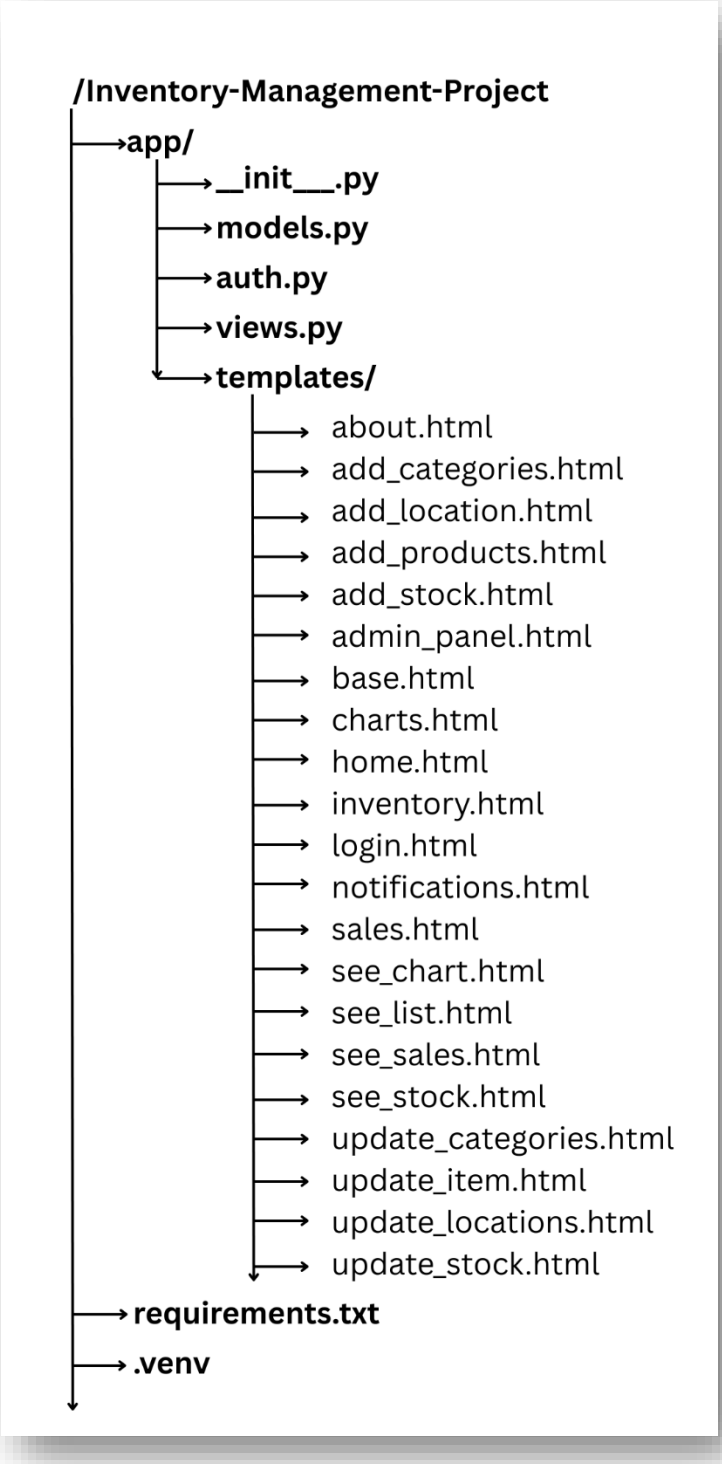
Chapter 4: Implementation and Testing

1. Development Environment:

Component	Specification	Purpose
Operating System	Windows 11	Development platform.
Programming Language	Python 3.13.2	Core backend logic.
IDE	VS Code	Code development and debugging.
Web Framework	Flask 3.1.0	Routing and application structure.
Database	SQLite	Persistent storage of inventory data.
ORM	Flask-SQLAlchemy	Managing database interactions using Python objects.
Frontend	HTML5, CSS3, Bootstrap, js	User interface and responsive design.
Security	Werkzeug.security	Password hashing for Admin accounts.

2. Key Implementation Details:

a. Project Structure and Routing



b. Item Issuance and Stock Update Logic (FR-005)

Process Flow:

- Admin submits the Issuance Form (/sale_process)
- The Flask route handler receives the request and **validates** the input.
- **Stock Check:** Query the database to ensure the current_stock for the item_id is sufficient (current_stock >= quantity_issued).
- **Transaction Record:** A new record is created in the SaleDetail table and recording the quantity, date, and buyer details.
- **Stock Adjustment:** The current_stock field in the Items table is decremented as
  Current Stock = Stock – Quantity Issued
- The user is redirected back to the Home with a success/failure message.

c. Reports Generation Implementation (FR-006 & FR-007)

- Data Query: A Flask route /download_sales_report receives date range filters.
- It queries the SaleDetail table using these filters, often joining with the Stock and the Sale tables to get readable names.
- **Data Processing:** Python structures the query results into a clean tabular format.
- **CSV Output: CSV** library takes the structured data to generate the data is formatted as a CSV string.
- **File Download:** The Flask route uses /download_sales_report to stream the generated file back to the user's browser, enabling the download.

3. Testing

a. Unit Testing

Test Function	Component	Description	Result
test_password_hashing ()	Authentication Logic	Verify that a given password hashes correctly and that the hash can be correctly verified against the original password.	Returns True for a correct password, False for an incorrect one.
Login_required ()	Authentication Logic	Test function that checks whether the users are authenticated when accessing the system resources.	Return True for authenticated users. Otherwise, return False.
test_stock_check ()	Business Logic	Test the function that compares the requested issuance quantity against the current stock level for a specific item ID.	Returns True if the requested quantity <= current stock. Otherwise, returns False.
test_real_time_ saletotal_updating ()	Business Logic	Test the function that updates the total amount with the given item and quantity.	Return the total amount in real-time correctly.
test_low_stocks _notification ()	Reporting Logic	Test the function that notify admin when stocks are low.	Return a notification when stocks are going low.
test_data_sorting ()	Reporting Logic	Test a database query function that accepts id, quantity, price, date, and name parameters.	Returns sorted data within related parameters.

b. Integration Testing

Integration Test 01:

Test Case: Master Data CRUD

Preconditions: The system has no category named "Peripherals".

Steps:

- Admin navigates to Category Management.
- Adds new category: "Peripherals".
- Edits the name to "PC Accessories".
- Deletes "PC Accessories".

Expected Result: The category is successfully added, modified, and then removed from the Categories table.

Actual Result: Category CRUD successful.

Status: Pass

Integration Test 02:

Test Case: Report Generation

Preconditions: Transactions exist for dates **26/11/2025** to **08/12/2025**.

Steps:

- Admin navigates to Sales Report.
- Clicks "Download Sales Report (CSV)" to "Generate Report".

Expected Result: The generated PDF file includes **all** transactions.

Actual Result: Report is accurate and downloadable.

Status: Pass

Integration Test 03:

Test Case: Item Issuance

Preconditions: Stock ID 1 ("Keychron Q1 Pro") has a current_stock of **15**.

Steps:

- Admin logs in.
- Navigates to the Issue Item form.
- Selects "Keychron Q1 Pro" (01).
- Enter quantity **10** and a Buyer Name.
- Submits.

Expected Result:

- A new StockTransaction record has been created for 10 units OUT.
- Stock ID 1's current_stock is updated to **5**.

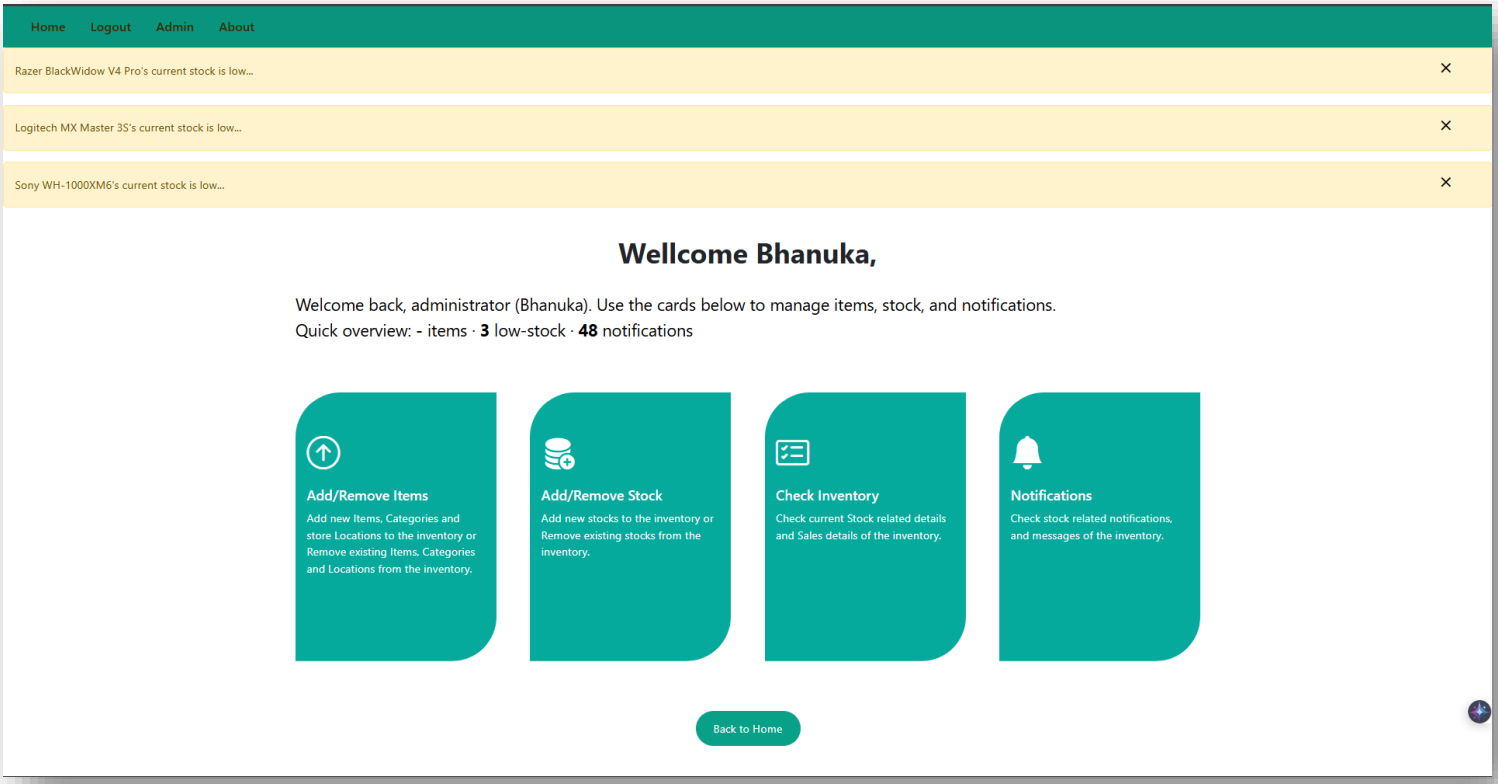
Actual Result:

- Transaction logged.
- Current Stock is 5**.**

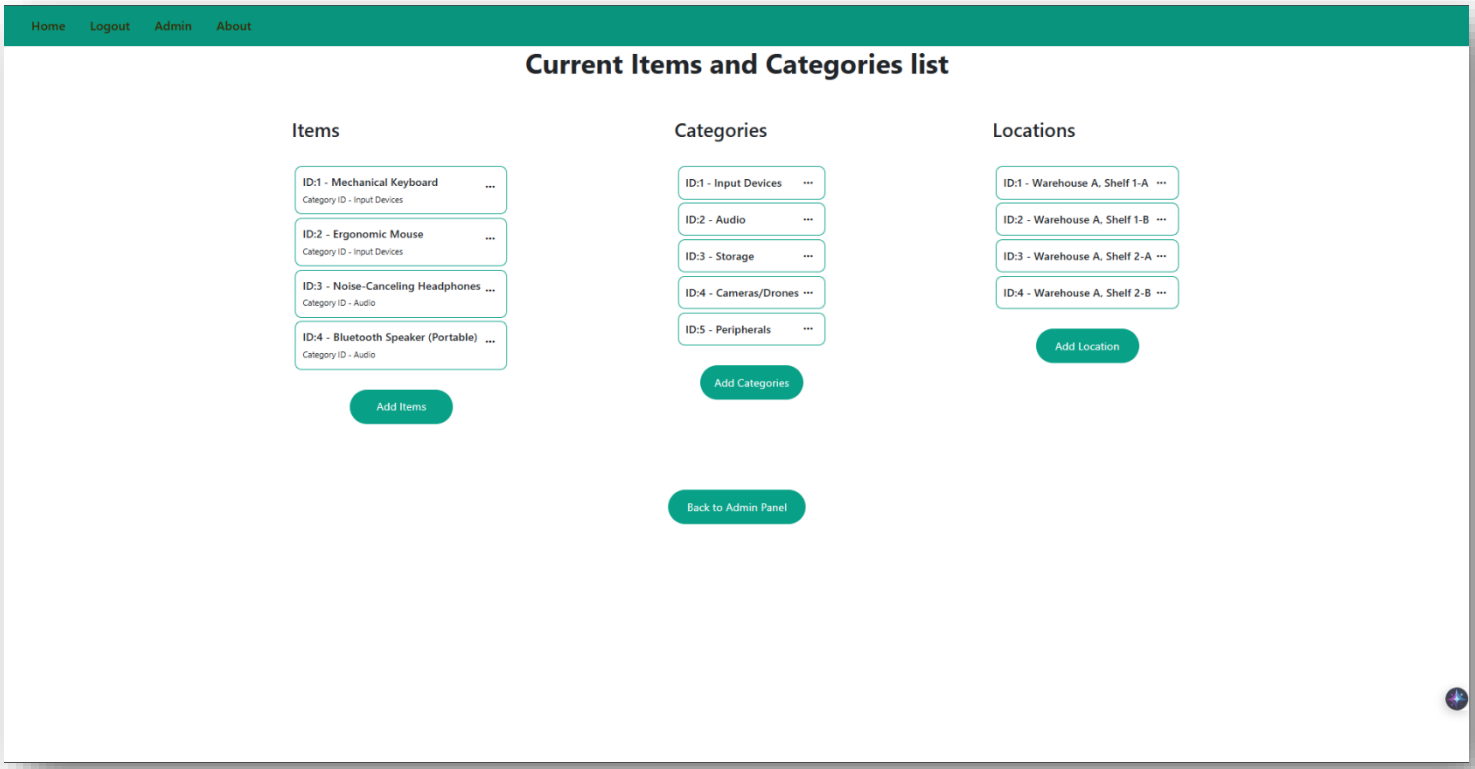
Status: Pass

Chapter 5: Results and Discussion

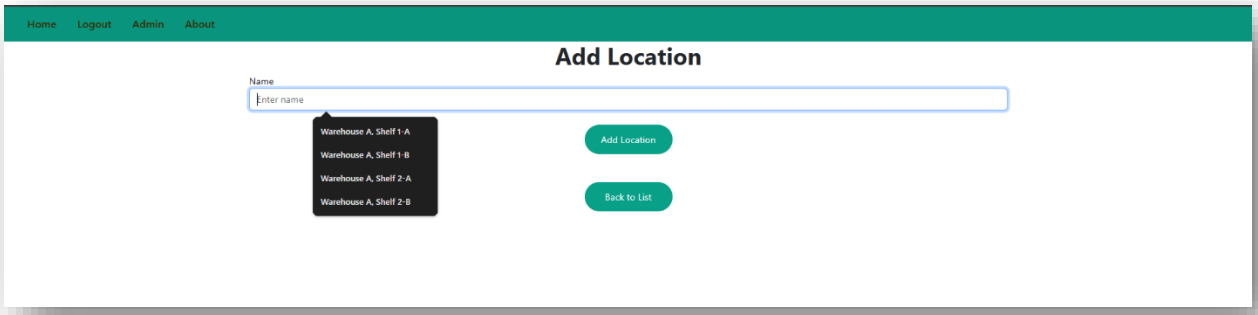
1. **System Demonstration:**
    a. **Admin Dashboard –** Shows the overall inventory's key features and notifies Admins of low stock items through notifications.

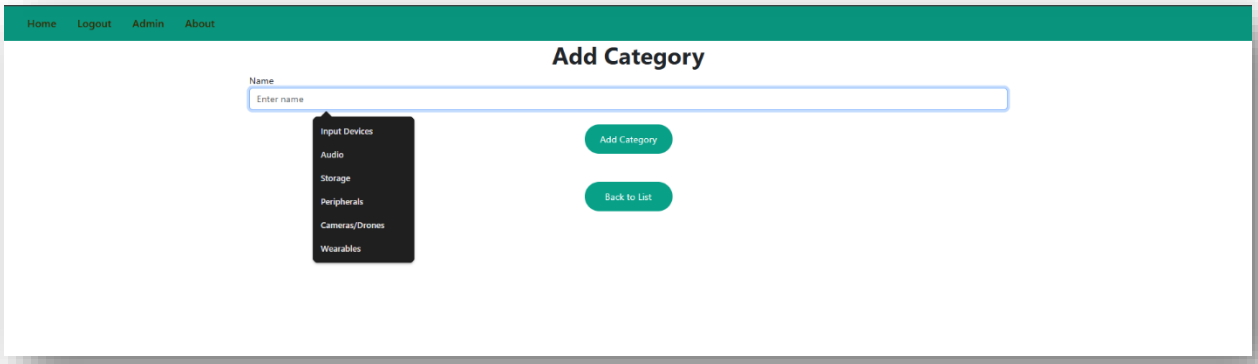


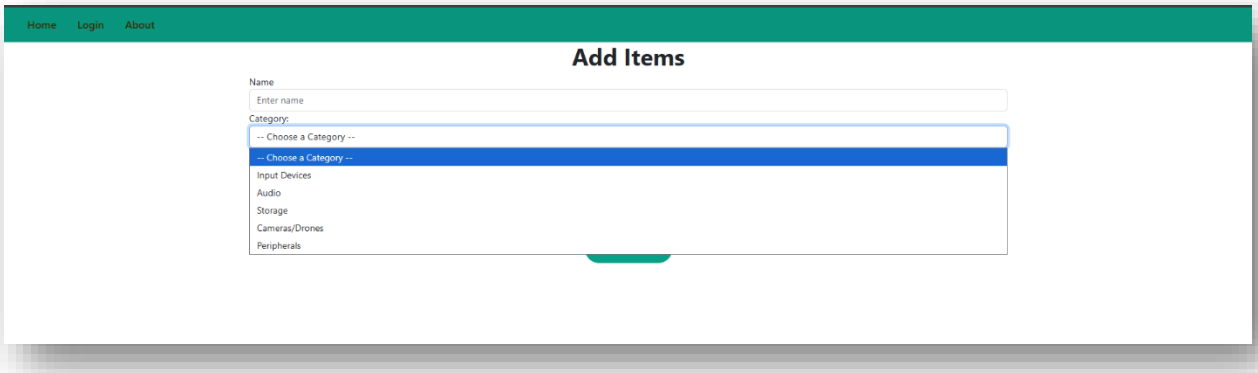
b. Current Items and Categories list – Show Current Items, Categories, and Locations that stock stores.



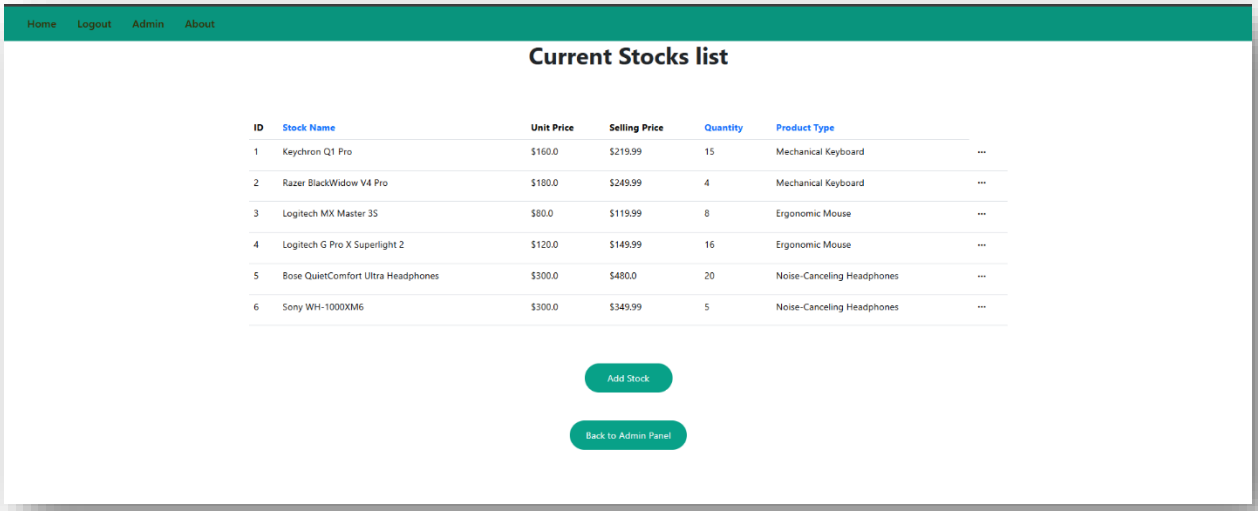
c. **Add Location, Add Category, and Add Items –** Allows Admins to add Locations, Categories, and Items to Inventory.



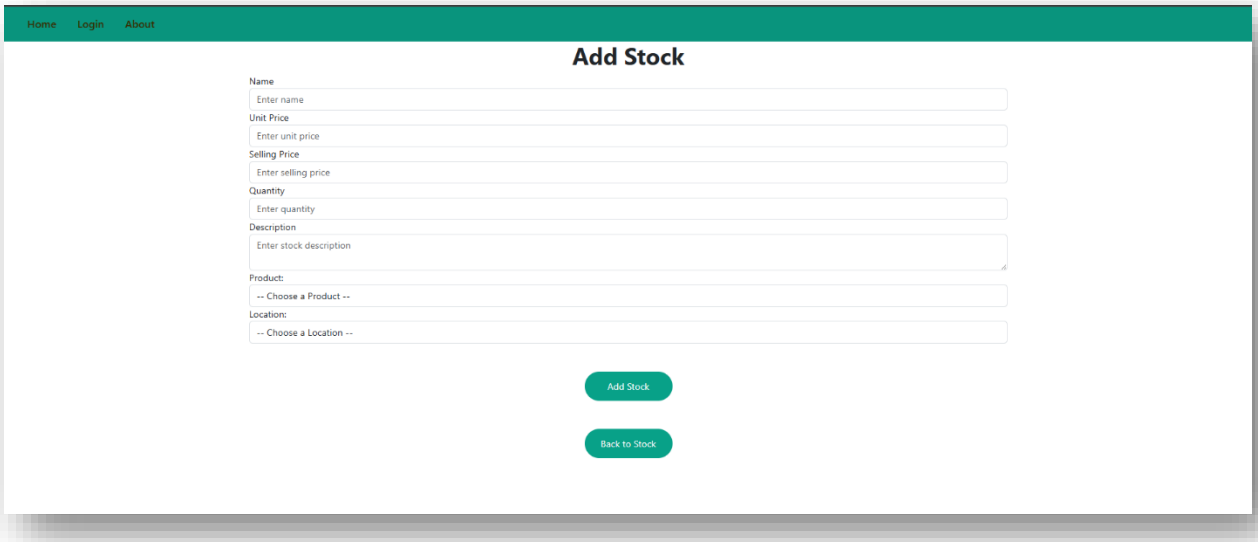




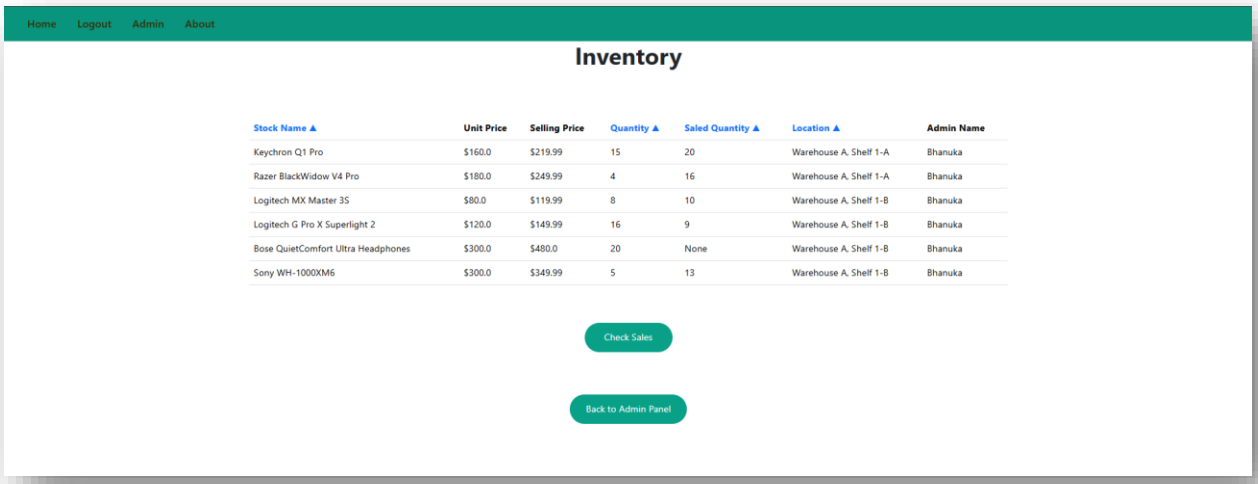
d. **Current Stocks list -** Show Current stocks in the Inventory.



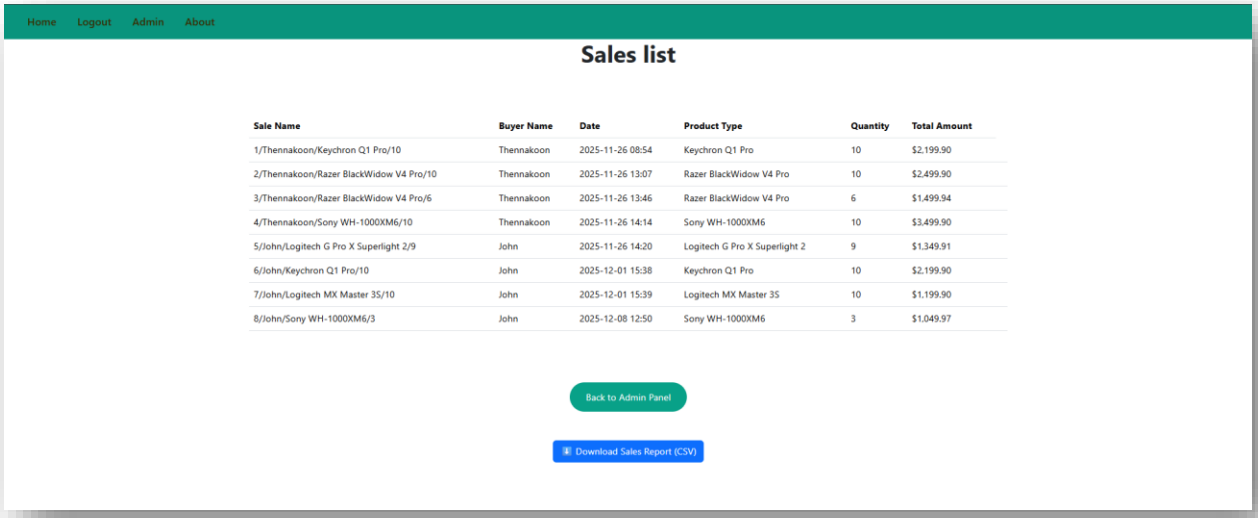
e. **Add Stock -** Allows Admins to add stocks with existing Items, Categories, and Locations.



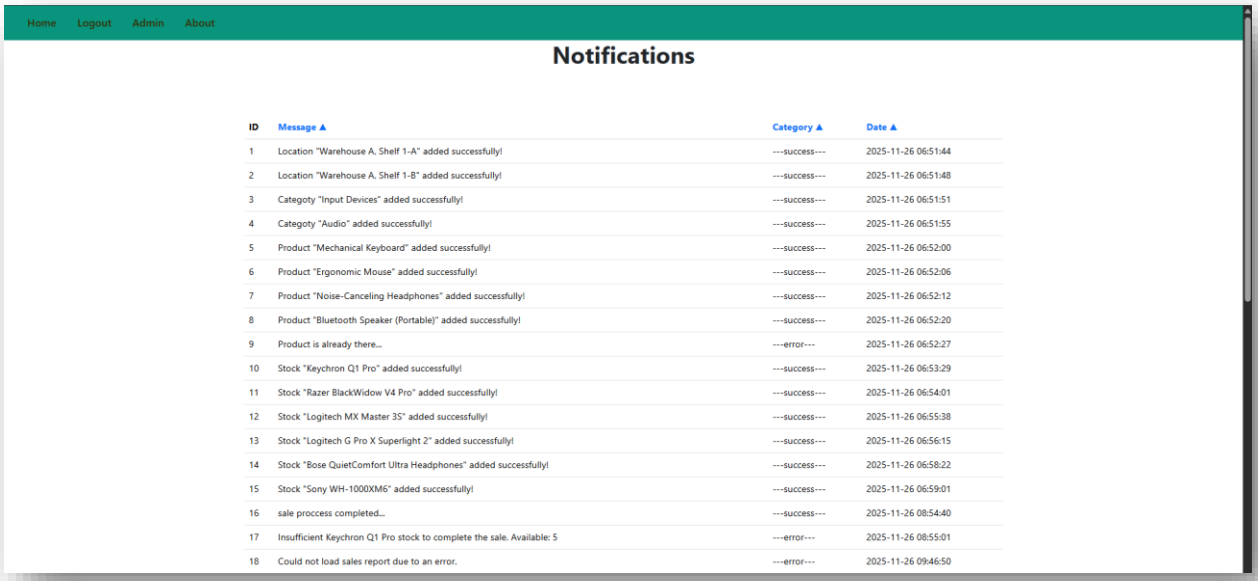
f. Inventory – Shows Current Inventory.



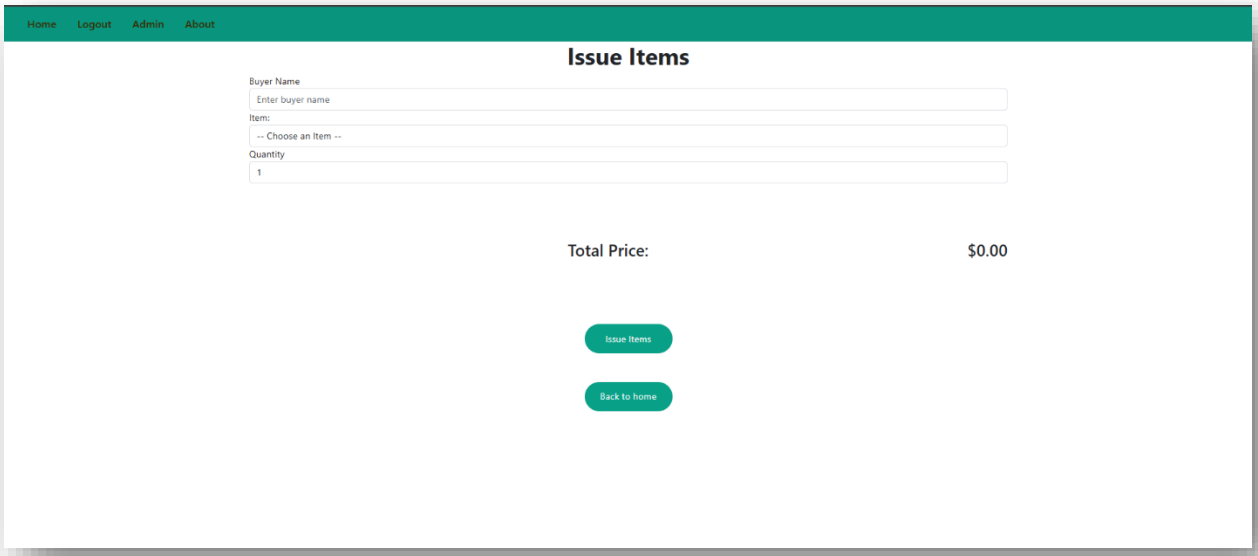
g. Sales list – Shows all Sales-related data and allows admins to download the sales report in CSV format.



h. Notifications – Show all Notifications for auditing processes.



i. Issue Items – Allows admins to issue items from the inventory to Buyers.



2. Discussion of Results:

a. System Strengths:

i. **Accessibility:** Web-based deployment offers access from any location.

ii. **Efficiency:** Automated transaction recording saves significant time compared to manual methods.

iii. **Low Cost:** Developed using open-source technologies (Python, Flask, open-source DB).

iv. **Alerts:** The system provides automatic notifications for low stock levels, not requiring manual checks.

v. **Report Download:** The system allows downloading of sales reports, and it can be used in some tools for Business Insights.

b. System Limitations:

i. **Scalability:** While Flask is scalable, the current design might not be optimal for very high-volume enterprises (e.g., thousands of transactions per second).

ii. **User Roles:** The system currently only supports a single **'Admin'** role; implementing hierarchical user roles (e.g., Supervisor, Clerk) would enhance security.

Chapter 6: Conclusion and Future Work

1. Conclusion:

The development of the **Web-Based Inventory Management System**, utilizing the Python Flask framework, was completed, resulting in a functional, user-friendly, and centralized administrative tool. The system effectively addressed the problems of manual tracking and delayed reporting by implementing robust features for master data management, real-time stock monitoring, low-stock indicating, and automated issuance processes (FR-005).

By successfully integrating the Presentation, Application, and Data tiers, the system provides a reliable platform for controlling inventory assets. The implementation meets all stated functional requirements, particularly the crucial ability for administrators to accurately issue items and instantly generate downloadable sales reports (FR-007). This project demonstrates the efficiency and suitability of lightweight Python frameworks like Flask for developing critical business applications.

2. Future Enhancements:

- **Stock Receiving Module:** Extend the transaction functionality to record **inbound** stock movements (receipts from suppliers), providing a complete ledger of all stock history.
- **Barcode/QR Code Scanning Support:** Integrate frontend functionality to allow administrators to scan item barcodes using a standard scanner or mobile device camera. This would accelerate data entry during item issuance.
- **Hierarchical User Roles:** Implement granular access controls (e.g., 'Supervisor' who can view reports but not manage stock, or 'Clerk' who can only issue items).
- **Data Visualization:** Enhance the reporting module with interactive charts and graphs on the dashboard to visualize sales trends and low-stock items.
- **Supplier Management:** Add a dedicated module to manage supplier information and link items to their primary suppliers, simplifying future purchasing decisions.