PROJECT REPORT

Topic: Electronic Store Management System

Domain: Retail & E-commerce

Language: Python 3.x

1. Abstract

The Electronic Store Management System is a console-based application designed to digitize the sales and inventory tracking of a small retail business. By transitioning from manual paper records to a file-based digital system, the project aims to reduce human error and improve operational efficiency. The system allows store owners to view real-time stock, process customer orders with automated calculations, and generate permanent digital invoices while simultaneously maintaining accurate inventory levels through automated updates.

2. Problem Definition

Small retail outlets often struggle with manual record-keeping. Using physical ledgers for sales is slow, prone to calculation errors, and fails to provide an immediate view of stock levels.

The proposed solution addresses three critical business needs:

1. **Stock Visibility:** The system must read inventory data directly from a central file to ensure that customers and staff always see accurate availability.
2. **Billing Automation:** The system removes the need for manual math by automatically calculating totals, applying complex percentage-based discounts, and generating the final bill.
3. **Data Persistence:** The most critical requirement is ensuring that once an item is sold, it is permanently removed from the "Available Quantity" in the database to prevent overselling.

3. System Architecture

3.1 Modular Design

To ensure the software is maintainable and organized, the project follows a **Modular Programming** approach. Rather than writing a single long script, the system is divided into distinct functional blocks:

- **Data Retrieval Module:** Responsible exclusively for accessing the database file and loading stock information into memory.
- **Transaction Module:** Handles the "Shopping Cart" logic, user interactions, and the mathematical rules for discounts.

- **Persistence Module:** Manages file output, specifically generating unique invoice files and overwriting the master stock file with updated quantities.
- **Controller Module:** Acts as the central hub that connects the retrieval, transaction, and persistence modules in a continuous loop.

3.2 Data Management Strategy

The system bridges theoretical computer science concepts with real-world application through specific data structures:

- **Sequential Storage (Lists):** Inventory data is loaded into a mutable list structure. This allows the system to temporarily modify stock numbers in the computer's memory while a customer is shopping, before committing those changes to the file.
- **Key-Value Mapping (Dictionaries):** The "Shopping Cart" is implemented as a dictionary. This maps specific product names to the quantity selected by the user, allowing for efficient retrieval and modification of order details before the final checkout.

4. Implementation Logic

The core functionality operates on a continuous **Read-Process-Write** cycle:

A. Initialization and Data Loading

Upon starting, the system accesses the external text file containing the product database. It reads the raw text, sanitizes it by removing formatting characters, and organizes it into a structured table (Product Name, Price, Quantity) for display to the user.

B. Transaction Processsing and Validation

The system prompts the user to select items. During this phase critical validation logic is applied:

- **Input Validation:** The system checks if the user entered a valid number for quantity. If text is entered by mistake, the system catches the error and requests a number again, preventing a crash.
- **Stock Validation:** Before adding an item to the cart, the system compares the requested quantity against the available stock. If the request exceeds the supply, the order is rejected.

C. Discount Algorithm

Once the shopping is complete, the total cost is calculated. The system then applies a tiered discount logic:

- Purchases falling within a specific mid-range bracket receive a standard percentage discount (e.g., 5%).
- High-value purchases exceeding the upper threshold qualify for a premium discount (e.g., 10%).

- If a user requests a discount tier they do not qualify for, the algorithm automatically reverts to the base rate.

D. File Persistence (The "Write" Phase)

The final phase involves two critical file operations:

1. **Invoice Generation:** A unique text file is generated using the precise date and time (down to the second) as the filename. This ensure that no previous invoices is ever overwritten.
2. **Inventory Update:** The system subtracts the sold items from the original stock levels and rewrites the entire product database file. This ensures that the next time the program runs, the stock levels reflect the latest sales.

5. Testing and Output

Test Case 1: Standard Transaction

- **Scenario:** A customer purchases a valid quantity of a product.
- **Outcome:** The system correctly calculates the total cost, generates a timestamped invoice file, and the main product file shows a reduced quantity for that item.

Test Case 2: Invalid Input Handling

- **Scenario:** The user inputs text (e.g-Five) instead of a number when asked for quantity.
- **Outcome:** The error handling mechanism activates, displaying a warning message and reprompting the user without crashing the program.

Test Case 3: Out of Stock Prevention

- **Scenario:** The user attempts to purchase more units than are currently available in the database.
- **Outcome:** The logic successfully detects the deficit and prevents the item from being added to the cart, informing the user of the limited stock.

6. Conclusion

This project successfully demonstrates the application of Python in solving a tangible business problem. By moving away from hard-coded values and implementing dynamic file handling, the system creates a persistent and reliable tool for inventory management. The development process reinforced key programming concepts, specifically the importance of modular design for code clarity and exception handling for user experience. Future enhancements could include moving from text files to a SQL database to handle larger inventoriees.

Prepared by :- Yash Gupta [25BCE10190]