2025

# BILLING MANAGEMENT SYSTEM

PROJECT:
HARSH VARDHAN SINGH (25BCY10124)

## Introduction

Billing Management System (BMS) automates and streamlines the checkout process for a retail apparel store, replacing manual calculations with a fast, error-free method for receipt generation. Implemented entirely in Python using standard data structures, its core functionality manages a fixed product catalog, processes user input (product codes and quantities), calculates the total, and produces a formatted, itemized receipt.

## Problem Statement

Manual retail billing is slow, error-prone, and lacks proper itemization. This project addresses the need for a simple, efficient, and accurate sales transaction processor. The system must provide immediate access to pricing, allow rapid item entry, instantly calculate totals, and generate a clear, professional transaction record for the customer.

## Functional Requirements (FRs)

The following requirements define the core capabilities of the implemented system:

| ID | Requirement Description | Status |
|---|---|---|
| **FR-01** | Display complete, itemized catalog (Code, Name, Price). | Implemented (show_catalog) |
| **FR-02** | Allow user to add items by entering a valid Product Code. | Implemented (add_to_cart) |
| **FR-03** | Validate and prompt for positive integer quantity for each item. | Implemented (req_qty) |
| **FR-04** | Calculate line total for each item and the grand total. | Implemented (compute_total) |
| **FR-05** | Generate a final receipt detailing Qty, Code, Name, Price, and Total. | Implemented (make_receipt) |
| **FR-06** | Provide a clear mechanism (e.g., 'BILL') to finalize transaction. | Implemented (add_to_cart) |

## Non-functional Requirements (NFRs)

These requirements describe the qualities and constraints of the system:

| ID | Requirement Description | Priority |
|---|---|---|
| **NFR-01** | **Usability:** Simple, text-based CLI for rapid data entry. | High |
| **NFR-02** | **Performance:** Instantaneous catalog lookups and calculations. | High |
| **NFR-03** | **Maintainability:** Modular codebase with focused functions. | Medium |
| **NFR-04** | **Reliability:** Robust input handling prevents crashes from invalid quantities. | High |
| **NFR-05** | **Data Integrity:** Pricing data must be consistent and non-mutable during transaction. | High |

**System Architecture**

The BMS uses a **Monolithic, Single-Tier Architecture**. The **Presentation Layer** uses the Python console (print()/input()). **Business Logic** resides in Python functions (add_to_cart, compute_total, etc.). The **Data Layer** utilizes in-memory Python dictionaries (PRODUCT_CATALOG) and lists (live_cart) for fixed product data and temporary transaction data, respectively. This design is fast and ideal for a lightweight, single-user application.

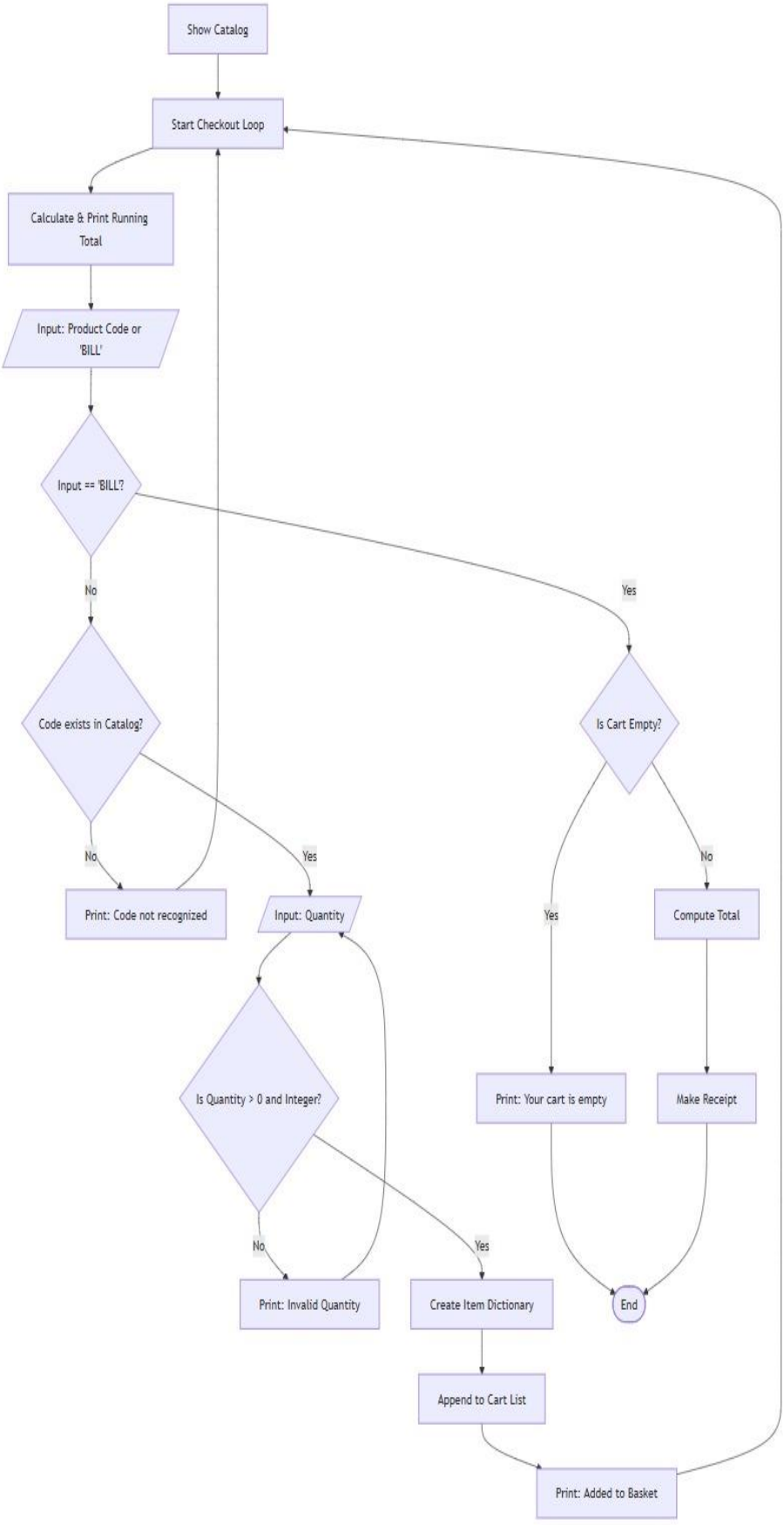
**Design Diagrams**

**Use Case Diagram**

The primary actor is the **Cashier** (User).

| Use Case | Description |
| --- | --- |
| **Manage Billing Transaction** | Core process for creating a new sale. |
| **View Product Catalog** | Display items and prices. |
| **Add Item to Cart** | Select product and specify quantity. |
| **Calculate Total** | Sum costs of all cart items. |
| **Generate Receipt** | Print final itemized bill. |

## Workflow Diagram



**Sequence Diagram**

This diagram focuses on the interaction between the User and the core system functions during the checkout process.

**Class/Component Diagram**

Since this is not an Object-Oriented project, the diagram represents the major components (data structures and functions) and their interactions.

| Component | Type | Responsibility |
|---|---|---|
| **PRODUCT_CATALOG** | Global Dictionary (Data) | Stores all available product details (Code, Name, Price). |
| **live_cart** | Global List (Data) | Stores the items added during the current transaction. |
| **show_catalog()** | Function | Handles catalog display (FR-01). |
| **req_qty()** | Function | Handles Quantity validation (FR-03, NFR-04). |
| **add_to_cart()** | Function | Main transaction logic, handles item lookup and cart update (FR-02). |
| **compute_total()** | Function | Calculates the running total and final grand total (FR-04). |
| **make_receipt()** | Function | Formats and prints the final bill (FR-05). |

**ER Diagram (if storage used)**

As the system uses in-memory data storage (Python Dictionaries and Lists) without a persistent database, a formal Entity-Relationship (ER) Diagram is not strictly applicable. Instead, we outline the **Conceptual Data Structure:**

| Data Structure | Type | Key Field | Attributes |
|---|---|---|---|
| **Product** (Catalog) | Dictionary | Product Code (WST01, IND08) | name, price |
| **Cart Item** (live_cart entry) | Dictionary | N/A | code, name, price, qty |

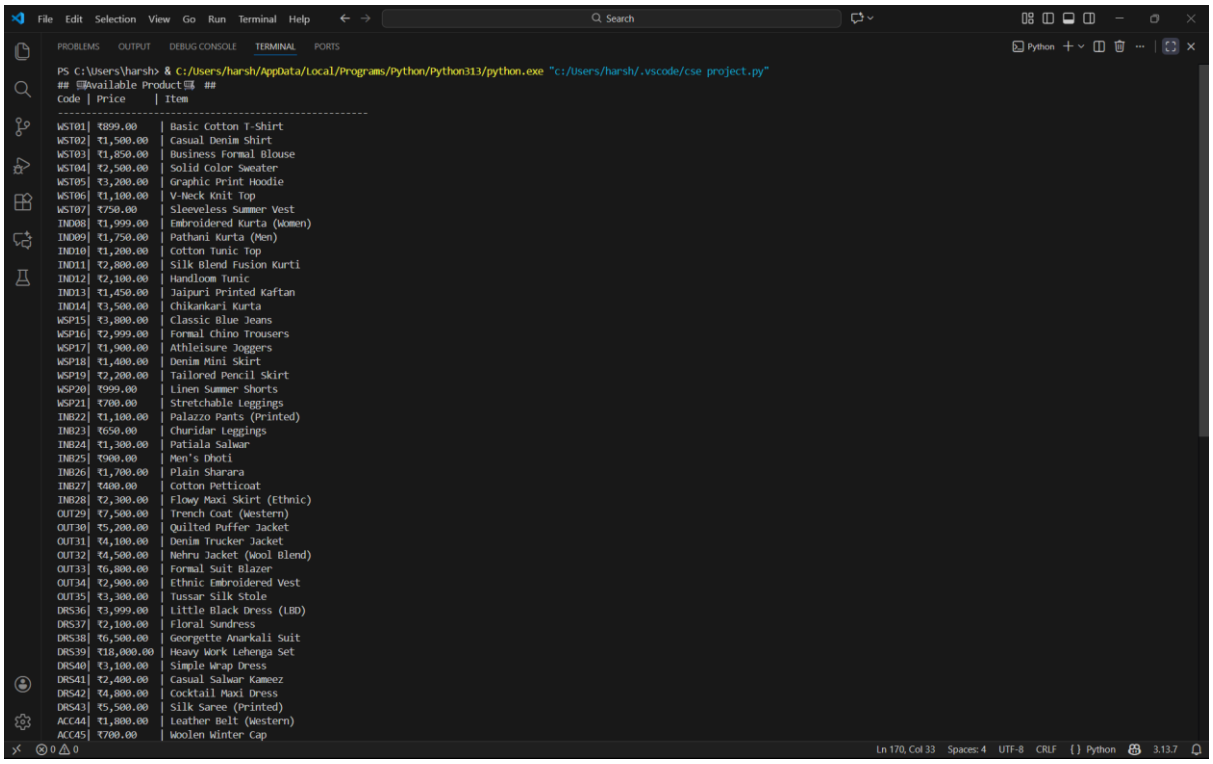
**Design Decisions & Rationale**

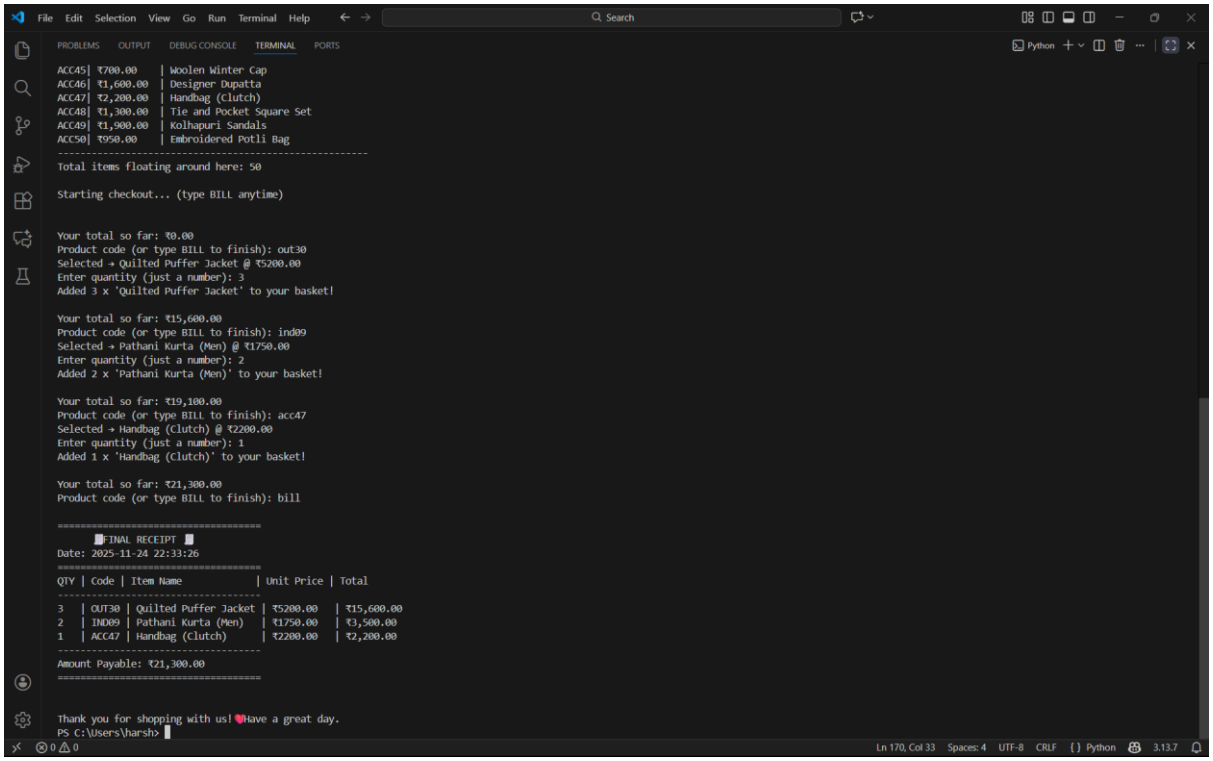
| Decision | Rationale |
|---|---|
| **In-Memory Catalog** | Provides **O(1)** (constant time) lookups via Python dictionary for fast performance (NFR-02). |
| **Modular Functions** | Small, focused functions (req_qty, compute_total, etc.) enhance maintainability and testability (NFR-03). |
| **While Loop for Billing** | while True loop with a sentinel (BILL) simplifies continuous item addition, simulating real checkout. |
| **Separate Quantity Function** | Centralizes error handling and validation logic, cleaning the main billing flow (FR-03, NFR-04). |
| **String Formatting** | Use of f-strings/format specifiers ensures neat, aligned, and professional output (NFR-01). |

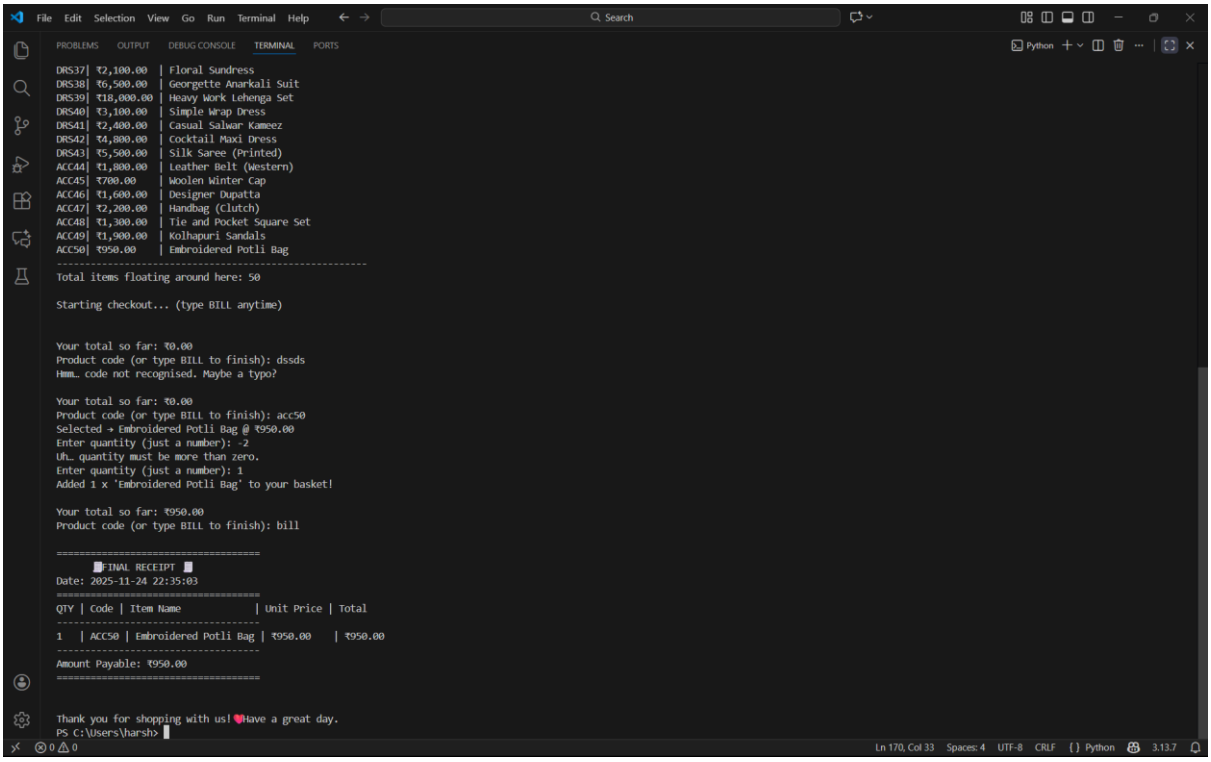
**Implementation Details**

The system is implemented within a single Python script using standard features. **Data Structures** include the PRODUCT_CATALOG dictionary (product codes as keys) and the live_cart list of item dictionaries (code, name, price, qty) for transaction details. **I/O** uses standard input()/print(). **Control Flow** is managed by a while loop. Key logic includes **Error Handling** in req_qty() using a try-except to manage non-integer inputs, and **Total Calculation** in compute_total(), which iterates the cart and sums line totals (price * qty).

## 10. Screenshots / Results




File Edit Selection View Go Run Terminal Help
Search
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
ACC45| ₹700.00 | Woolen Winter Cap
ACC46| ₹1,600.00 | Designer Dupatta
ACC47| ₹2,200.00 | Handbag (Clutch)
ACC48| ₹1,300.00 | Tie and Pocket Square Set
ACC49| ₹1,900.00 | Kolhapuri Sandals
ACC50| ₹950.00 | Embroidered Potli Bag
Total items floating around here: 50
Starting checkout... (type BILL anytime)
Your total so far: ₹0.00
Product code (or type BILL to finish): out30
Selected → Quilted Puffer Jacket @ ₹5200.00
Enter quantity (just a number): 3
Added 3 x 'Quilted Puffer Jacket' to your basket!
Your total so far: ₹15,600.00
Product code (or type BILL to finish): ind09
Selected → Pathani Kurta (Men) @ ₹1750.00
Enter quantity (just a number): 2
Added 2 x 'Pathani Kurta (Men)' to your basket!
Your total so far: ₹19,100.00
Product code (or type BILL to finish): acc47
Selected → Handbag (Clutch) @ ₹2200.00
Enter quantity (just a number): 1
Added 1 x 'Handbag (Clutch)' to your basket!
Your total so far: ₹21,300.00
Product code (or type BILL to finish): bill
FINAL RECEIPT
Date: 2025-11-24 22:33:26
QTY | Code | Item Name | Unit Price | Total
3 | OUT30 | Quilted Puffer Jacket | ₹5200.00 | ₹15,600.00
2 | IND09 | Pathani Kurta (Men) | ₹1750.00 | ₹3,500.00
1 | ACC47 | Handbag (Clutch) | ₹2200.00 | ₹2,200.00
Amount Payable: ₹21,300.00
Thank you for shopping with us! Have a great day.
PS C:\Users\harsh>
Ln 170, Col 33 Spaces: 4 UTF-8 CRLF { } Python 3.13.7


File Edit Selection View Go Run Terminal Help
Search
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
DRS37| ₹2,100.00 | Floral Sundress
DRS38| ₹6,500.00 | Georgette Anarkali Suit
DRS39| ₹18,000.00 | Heavy Work Lehenga Set
DRS40| ₹3,100.00 | Simple Wrap Dress
DRS41| ₹2,400.00 | Casual Salwar Kameez
DRS42| ₹4,800.00 | Cocktail Maxi Dress
DRS43| ₹5,500.00 | Silk Saree (Printed)
ACC44| ₹1,800.00 | Leather Belt (Western)
ACC45| ₹700.00 | Woolen Winter Cap
ACC46| ₹1,600.00 | Designer Dupatta
ACC47| ₹2,200.00 | Handbag (Clutch)
ACC48| ₹1,300.00 | Tie and Pocket Square Set
ACC49| ₹1,900.00 | Kolhapuri Sandals
ACC50| ₹950.00 | Embroidered Potli Bag
Total items floating around here: 50
Starting checkout... (type BILL anytime)
Your total so far: ₹0.00
Product code (or type BILL to finish): dssds
Hmm.. code not recognised. Maybe a typo?
Your total so far: ₹0.00
Product code (or type BILL to finish): acc50
Selected → Embroidered Potli Bag @ ₹950.00
Enter quantity (just a number): -2
Uh.. quantity must be more than zero.
Enter quantity (just a number): 1
Added 1 x 'Embroidered Potli Bag' to your basket!
Your total so far: ₹950.00
Product code (or type BILL to finish): bill
FINAL RECEIPT
Date: 2025-11-24 22:35:03
QTY | Code | Item Name | Unit Price | Total
1 | ACC50 | Embroidered Potli Bag | ₹950.00 | ₹950.00
Amount Payable: ₹950.00
Thank you for shopping with us! Have a great day.
PS C:\Users\harsh>
Ln 170, Col 33 Spaces: 4 UTF-8 CRLF { } Python 3.13.7

## Testing Approach

Testing employed a two-pronged approach: **1. Unit Testing (Black-Box)** focused on core functions. compute_total() was verified for accuracy across cart sizes. req_qty() was rigorously tested with valid, non-numeric, and zero/negative inputs for robust validation. Catalog lookup integrity was confirmed. **2. System Testing (End-to-End)** executed a full billing cycle to ensure component integration. An edge case test confirmed graceful handling of an empty cart (typing BILL immediately).

## Challenges Faced

Challenges stemmed from the limitations of a console application proof-of-concept: **Persistence** is absent, as all data (transactions/catalog changes) is lost upon program termination, mandating future database integration. **Scalability** is limited by the in-memory catalog for large inventory sizes. **Feature Scope** is constrained, preventing advanced business logic like tax calculation, discounts, or user authentication.

## Learnings & Key Takeaways

Key takeaways reinforced fundamental software concepts: **Input Robustness** is vital, proven by req_qty()'s error handling to prevent crashes. **Modular Design** enhanced maintainability by separating concerns (calculation, I/O, cart logic). **Data Structure Choice** favored dictionaries for optimal, fast, key-based catalog lookups. Finally, careful **Formatting for Usability** using Python f-strings ensured a professional and readable CLI output.

## Future Enhancements

To evolve this proof-of-concept into a production system, key enhancements are: **1. Database Integration:** Migrate catalog and history to a persistent database (e.g., SQLite, Firestore) for data retention. **2. Advanced Pricing:** Implement support for discounts and automatic tax calculation (e.g., GST/VAT). **3. User Interface (UI):** Move from CLI to a modern web (React/Angular) or desktop GUI. **4. Inventory Management:** Track stock levels and decrement inventory post-billing. **5. Reporting:** Generate sales summaries and best-seller reports from transaction data.

## References


1. Python Documentation, Version 3.11. \url{https://docs.python.org/}
2. Introduction to Computer Science and Programming, Data Structures and Algorithms.
3. Software Engineering Concepts: Requirements Analysis, System Architecture, and Design.