VIT®

BHOPAL

HOTEL MANAGEMENT INVENTORY DATABASE SYSTEM

NAME:- PRATUL KASHYAP

REGISTRATION NO. :- 25BCE10322

INTRODUCTION:

Hotel Management Inventory Database System

Overview:

This project is a comprehensive Inventory Management System designed specifically for the hospitality industry. It replaces manual logbooks with a robust SQL-based database to track assets, manage suppliers, and monitor stock levels in real-time. This solution addresses the issues of revenue leakage and operational inefficiencies caused by traditional paper-based tracking.

Technologies Used:

- **Language: Python (for application logic)**
- **Database: SQL (MySQL / SQLite)**
- **Tools: VS Code, Git, Mermaid.js (for documentation)**

Steps to Install & Run:

Clone the repository:

git clone: https://github.com/pratul-kashyap1234/Hotel-Management-Inventory-Database-System

Install dependencies:(Ensure you have Python installed)

pip install -r requirements.txt

Initialize the database: Run the SQL script to create the tables and relationships.

 # If using SQLite via a setup script

python setup_database.py

 # OR import 'database_schema.sql' into your MySQL Workbench

Run the application:

python main.py

Instructions for Testing:

Test Add Item:

- **Navigate to the "Add Item" menu.**
- **Input a new product (e.g., "Liquid Soap 5L") with a quantity of 10.**
- **Verify it appears in the "View All Inventory" list.**

Test Transaction (Stock Out):

- **Select "Issue Stock".**
- **Choose the item "Liquid Soap 5L" and issue 2 units to "Housekeeping".**
- **Check the "View All Inventory" list again; the quantity should now be 8.**

Test Low Stock Alert:

- **Manually issue enough stock so the remaining quantity falls below the reorder level (e.g., below 5).**
- **Navigate to "Reports" -> "Low Stock"; the item should appear there.**

Functional Requirements:

- **Product Management: Create, Read, Update, and Delete (CRUD) inventory items with categorization.**
- **Stock Tracking: Real-time updates of stock levels via "Check-in" (Purchase) and "Check-out" (Issue) transactions.**
- **Reporting: Automatic detection of items below reorder levels (Low Stock Alerts).**
- **Department Allocation: Track which department (Kitchen, Laundry, etc.) is consuming the most resources.**
- **Role-Based Access: Security separation between Admin (full access) and General Staff (transaction access only).**

Non-Functional Requirements:

- **Portability: The system runs as a single Python script with no external dependencies or database installations required.**
- **Performance: Since data is stored in RAM (In-Memory), retrieval and updates are instantaneous (microsecond latency).**
- **Usability: The Command Line Interface (CLI) prompts are designed to be intuitive, guiding the user step-by-step.**
- **Robustness: The system handles basic logic errors, such as preventing users from consuming non-existent stock.**

Problem Statement:

Hotels manage thousands of consumable items daily, from toiletries and linens to food ingredients. Manual tracking of this inventory using paper logs or disconnected spreadsheets leads to operational inefficiencies, including:

1. **Revenue Leakage: Due to unrecorded consumption or theft.**
2. **Operational Delays: Running out of critical stock (e.g., soap, fresh towels) negatively impacts guest experience.**
3. **Data Silos: Purchasing teams do not know what the Housekeeping team actually needs in real-time.**

Scope of the Project:

The scope of this project is limited to the internal inventory control of a single hotel property. It covers the lifecycle of a product from purchase (entry) to issuance (exit). It does not cover room booking or guest management.

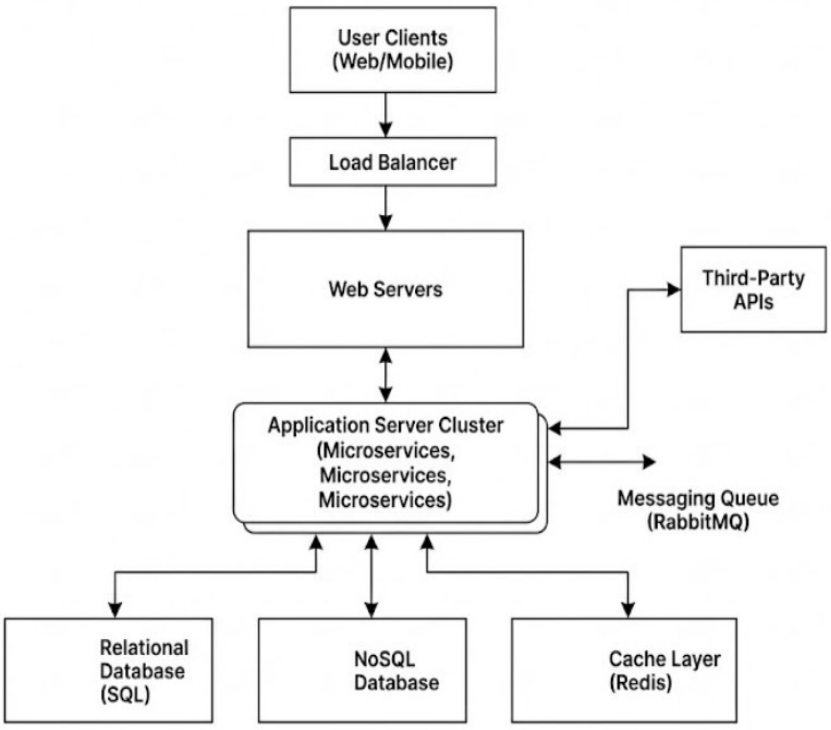
Target Users:

- **Hotel Store Manager: Responsible for purchasing and maintaining stock levels.**
- **Department Heads: (e.g., Head Chef, Housekeeping Supervisor) who request items from the store.**
- **Hotel Administrator: Who oversees the overall efficiency and costs.**

High-Level Features:

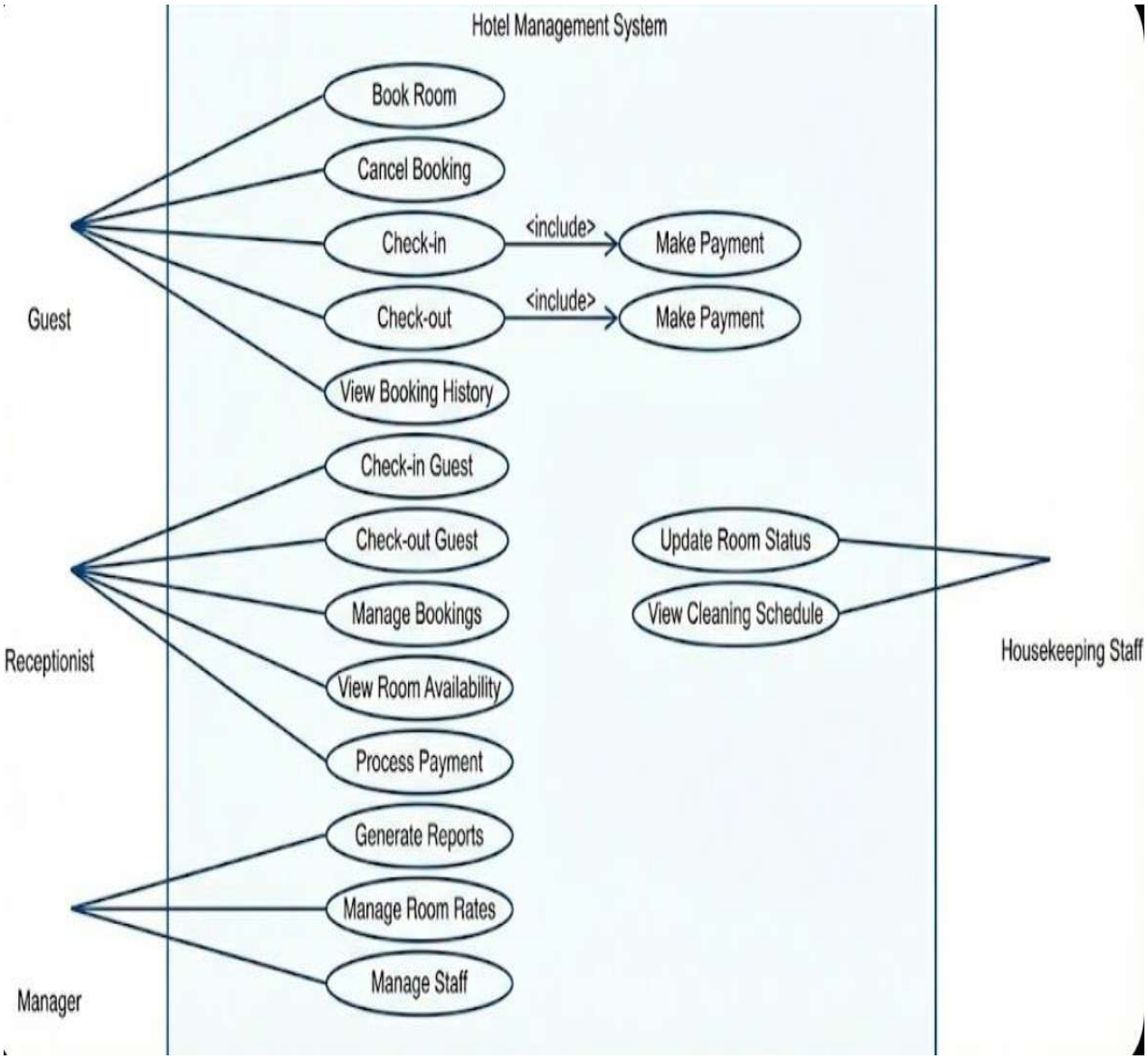
- **Centralized Database: A single source of truth for all hotel assets.**
- **Transaction Logging: Immutable history of every item moved in or out.**
- **Supplier Management: Linking items to specific vendors for easier reordering.**
- **Alert System: Proactive notifications for low inventory.**

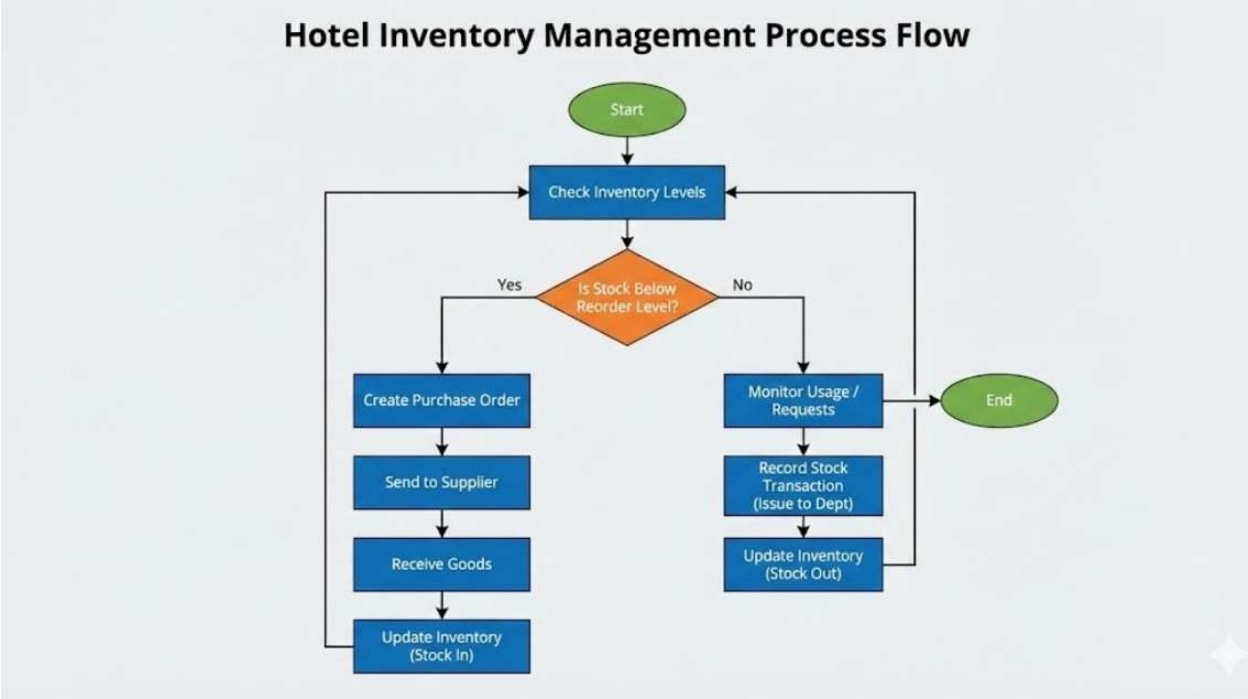
SYSTEM ARCHITECTURE:



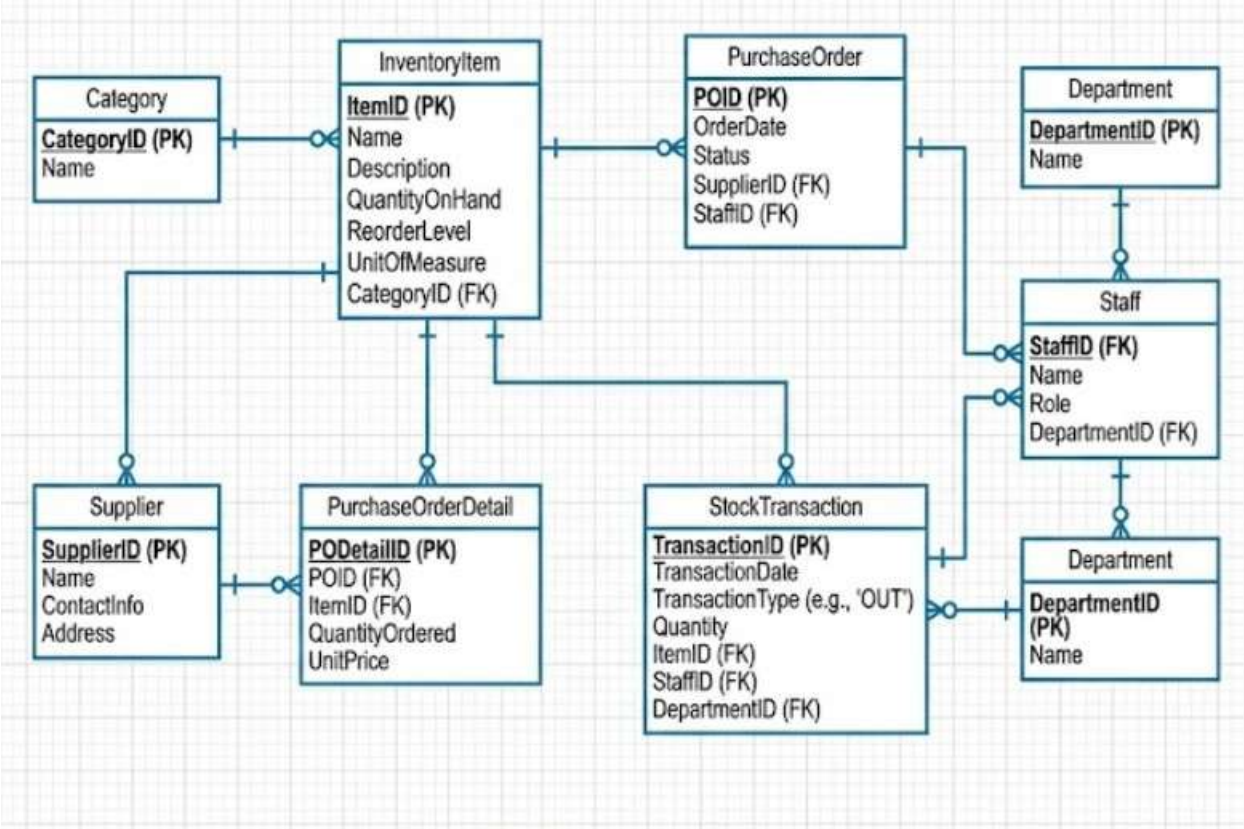
DESIGN DIAGRAMS:

USE CASE DIAGRAM:




Hotel Inventory Management Process Flow
Start
Check Inventory Levels
Is Stock Below Reorder Level?
Yes
No
Create Purchase Order
Send to Supplier
Receive Goods
Update Inventory (Stock In)
Monitor Usage / Requests
Record Stock Transaction (Issue to Dept)
Update Inventory (Stock Out)
End

ER.DIAGRAM:


Category
CategoryID (PK)
Name
InventoryItem
ItemID (PK)
Name
Description
QuantityOnHand
ReorderLevel
UnitOfMeasure
CategoryID (FK)
PurchaseOrder
POID (PK)
OrderDate
Status
SupplierID (FK)
StaffID (FK)
Department
DepartmentID (PK)
Name
Staff
StaffID (FK)
Name
Role
DepartmentID (FK)
Supplier
SupplierID (PK)
Name
ContactInfo
Address
PurchaseOrderDetail
PODetailID (PK)
POID (FK)
ItemID (FK)
QuantityOrdered
UnitPrice
StockTransaction
TransactionID (PK)
TransactionDate
TransactionType (e.g., 'OUT')
Quantity
ItemID (FK)
StaffID (FK)
DepartmentID (FK)
Department
DepartmentID (PK)
Name

Implementation Details:

The core logic relies on list manipulation.

- **Adding Items:** `products.append(new_dictionary)` **adds a new record.**
- **Updating Stock: The system iterates through the** `products` **list to find the matching** `id`**. Once found, it directly modifies the** `stock` **key.**

```
# Code Snippet: Stock Logic
if action == 'IN':
    selected_product["stock"] += qty
elif action == 'OUT':
    if selected_product["stock"] >= qty:
        selected_product["stock"] -= qty
```

Implementation Details (Data Structures)

- **Products List:** Stores the main inventory.
  products = [
  {"id": 1, "name": "Soap", "stock": 50, ...},
  {"id": 2, "name": "Towel", "stock": 20, ...}
]

- **Categories List:** Hardcoded categories to simplify data entry.
  categories = [

  {"id": 1, "name": "Housekeeping"},
  {"id": 2, "name": "Kitchen"}

  ]

SCREENSHOTS:

```
=== HOTEL INVENTORY SYSTEM (In-Memory) ===
1. Add New Item
2. View Inventory
3. Update Stock (In/Out)
4. Exit
```

```
=== HOTEL INVENTORY SYSTEM (In-Memory) ===
1. Add New Item
2. View Inventory
3. Update Stock (In/Out)
4. Exit
Select an option (1-4): 1

--- ADD NEW PRODUCT ---
Enter Product Name (e.g., Soap): nirma

Select Category:
1. Housekeeping
2. Kitchen
3. Maintenance
Enter Category ID: 2
Enter Unit Price: 2.4
Enter Reorder Level (default 5): 5
Success! 'nirma' added to inventory with ID 1.
```

```
=== HOTEL INVENTORY SYSTEM (In-Memory) ===
1. Add New Item
2. View Inventory
3. Update Stock (In/Out)
4. Exit
Select an option (1-4): 2

--- CURRENT INVENTORY STATUS ---
ID    Name                 Stock      Price      Status
-------------------------------------------------------
1     nirma                0          $2.4       LOW STOCK!
```

```
=== HOTEL INVENTORY SYSTEM (In-Memory) ===
1. Add New Item
2. View Inventory
3. Update Stock (In/Out)
4. Exit
Select an option (1-4): 3

--- UPDATE STOCK ---

--- CURRENT INVENTORY STATUS ---
ID    Name                 Stock      Price      Status
-------------------------------------------------------
1     nirma                0          $2.4       LOW STOCK!

Enter Product ID to update: 1
Type 'IN' to add stock (Purchase) or 'OUT' to issue stock: IN
Enter Quantity: 67
Stock updated successfully.
```

```
=== HOTEL INVENTORY SYSTEM (In-Memory) ===
1. Add New Item
2. View Inventory
3. Update Stock (In/Out)
4. Exit
Select an option (1-4): 2

--- CURRENT INVENTORY STATUS ---
ID    Name                 Stock      Price      Status
-------------------------------------------------------
1     nirma                67         $2.4       OK

=== HOTEL INVENTORY SYSTEM (In-Memory) ===
1. Add New Item
2. View Inventory
3. Update Stock (In/Out)
4. Exit
Select an option (1-4): 4
Exiting system. Goodbye!
```

Testing Approach:

- **Functional Testing:**
  - ***Test:*** **Added "Soap" with 0 stock. Performed "Stock In" of 10.**
  - ***Result:*** **View Inventory showed 10. Success.**

- **Boundary Testing:**
  - ***Test:*** **Tried to "Stock Out" 15 units when only 10 were available.**
  - ***Result:*** **System displayed "Insufficient stock" error message. Success.**

- **Logic Verification:**
  - ***Test:*** **Verified that Reorder Level alerts appear exactly when stock matches or falls below the limit.**

Challenges Faced:

- **Data Volatility:** **Since we removed the database, all data is lost when the program closes. This was a trade-off for simplicity and ease of setup.**

- **Input Handling:** Without `try/except` blocks, the program requires the user to be careful (e.g., ensuring they type numbers, not letters).

- **List Iteration:** Finding the correct item in a list required writing a manual loop, which taught me about list traversal.

Learnings & Key Takeaways:

- **Data Structures:** Gained a deep understanding of how to use Lists of Dictionaries to model real-world objects.

- **Logic Flow:** Learned how to structure a `while True` loop to create a continuous application experience.

- **State Management:** Understood how variables hold the "state" of the application during its runtime.

Future Enhancements:

- **File I/O:** Implement saving data to a `.txt` or `.csv` file so that inventory is remembered after closing the program.

- **Search Function:** Add a feature to search for products by Name, not just ID.

- **GUI:** Build a visual interface using `Tkinter` for better user experience.

References:

1. Python 3 Documentation (Data Structures).

2. "Automate the Boring Stuff with Python" - List/Dictionary manipulation.

3. Standard Hotel Inventory Procedures.