Hotel Management Inventory System (HMIS)

1. project-overview:

The **Hotel Management Inventory System (HMIS)** is a specialized, lightweight software solution designed to digitize the asset tracking processes of small to mid-sized hotels. In the hospitality industry, managing the flow of consumables—such as housekeeping toiletries, kitchen ingredients, and maintenance tools—is critical for operational efficiency.

This project serves as a **standalone console application** that replaces traditional, error-prone paper logbooks. It provides hotel staff with a centralized platform to register products, monitor stock levels in real-time, and record the movement of goods (purchases and issuances) with precision.

By utilizing **In-Memory Data Storage** (Python Lists and Dictionaries), the system offers lightning-fast performance and zero-configuration setup, making it ideal for immediate deployment in environments where installing complex SQL servers is not feasible.

2. Problem-statement:

Hotels manage thousands of individual items daily. Without a proper digital system, they face several operational challenges:

- **Operational Blindness:** Store managers cannot instantly verify if they have enough soap or clean towels without physically walking to the storage room.
- **Revenue Leakage:** Unrecorded consumption or theft often goes unnoticed until the end-of-month audit.
- **Human Error:** Manual calculation of stock (e.g., Previous Stock + Purchase - Issue) is prone to mathematical mistakes when done on paper.
- **Stockouts:** Critical items run out unexpectedly because there is no automated "Low Stock" warning system, leading to guest complaints.

HMIS solves these problems by enforcing a strict digital workflow for every item that enters or leaves the hotel storage.

3.Functional-requirements:

The system is divided into three core functional modules, each handling a specific aspect of inventory control.

Module 1: Item Management (CRUD Operations)

This module allows the hotel administrator to define what items are being tracked.

- **Product Registration:** Users can add new items with specific attributes:
    - **Name:** (e.g., "Shampoo 50ml")
    - **Category:** Linked to specific departments (Housekeeping, Kitchen).
    - **Unit Price:** Cost per item for financial tracking.
    - **Reorder Level:** The minimum quantity threshold before an alert is triggered.
- **Auto-Incrementing IDs:** The system automatically assigns a unique ID to every new product to prevent duplicates.

Module 2: Inventory Tracking

This module provides visibility into the current state of the hotel's assets.

- **Real-Time Dashboard:** A tabular view of all registered products.
- **Status Indicators:** The system intelligently compares the Current Stock against the Reorder Level.
    - If Stock <= Reorder Level: It displays a **"LOW STOCK!"** alert.
    - If Stock > Reorder Level: It displays **"OK"**.
- **Financial Overview:** Displays unit prices to help managers estimate the value of stock on hand.

Module 3: Stock Transactions

This module handles the core logic of inventory flow.

- **Stock In (Procurement):** When a supplier delivers goods, this function adds the quantity to the existing stock.
- **Stock Out (Issuance):** When a department (e.g., Housekeeping) requests items, this function deducts the quantity.
- **Validation Logic:** The system strictly prevents "Negative Stock." You cannot issue 10 items if you only have 5 in stock.

4. Non-functional-requirements:

To ensure the system is robust and user-friendly, the following non-functional constraints have been implemented:

- **Performance:** The system utilizes **In-Memory storage (RAM)**, ensuring that read/write operations (like viewing inventory or updating stock) occur in **O(1) or O(n)** time complexity. This guarantees instant feedback even with hundreds of items.
- **Usability:** The Command Line Interface (CLI) is designed with clear, step-by-step prompts. Error messages are descriptive (e.g., "Insufficient stock" rather than code errors), making it accessible to staff with minimal technical training.
- **Reliability (Data Integrity):** The application includes strict validation rules. It prevents

logical errors such as negative inventory counts or entering duplicate product IDs, ensuring the data remains consistent during the session.

- **Portability:** The system is built using only the **Python Standard Library**. It requires **no external database installation** (like MySQL or PostgreSQL) and runs identically on Windows, macOS, and Linux, making deployment immediate and hassle-free.

5. Technical-architecture:

This project is built using Python 3 and follows a procedural programming paradigm for simplicity and readability.

Data Structures

Instead of a heavy database (SQL), HMIS uses efficient Python native data structures to store data in Random Access Memory (RAM).

1. The Product Repository (products list)
Each product is stored as a Dictionary inside a main List. This mimics the structure of a JSON object or a NoSQL document.

```
products = [
  {
    "id": 1,
    "name": "Bath Towel",
    "category_id": 1,
    "price": 15.50,
    "reorder_level": 10,
    "stock": 100
  },
  {
    "id": 2,
    "name": "Tomato Ketchup",
    "category_id": 2,
    "price": 2.50,
    "reorder_level": 20,
    "stock": 45
  }
]
```

2. The Category Reference (categories list)
To ensure consistency, categories are pre-defined.

```
categories = [
  {"id": 1, "name": "Housekeeping"},
  {"id": 2, "name": "Kitchen"},
  {"id": 3, "name": "Maintenance"}
]
```

Algorithmic Logic

- **Search Algorithm:** When a user wants to update stock for Product ID 5, the system performs a **Linear Search** (O(n)) through the products list to find the matching dictionary.
- **State Persistence:** Variables are maintained in the global scope as long as the script is running. Once the script terminates, the RAM is cleared (Volatile Memory).

6. Prerequisites--installation:

Prerequisites

- A computer running **Windows, macOS, or Linux**.
- **Python 3.6 or higher** installed.

Installation Steps

1. Clone the Repository:
   Open your terminal or command prompt and run:
   ```
   git clone
   https://github.com/pratul-kashyap1234/Hotel-Management-Inventory-Database-System
   ```
2. **Navigate to the Directory:**
   ```
   cd hotel-inventory-system
   ```

3. Verify Python Installation:
   Ensure Python is correctly installed by checking the version:
   ```
   python --version
   ```

   (Should output Python 3.x.x)

7. Usage:

Starting the Application

Run the main script file:

```
python main.py
```

You will be greeted by the Main Menu:

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

Adding a New Product

Use this option to register a newly purchased item type that hasn't been tracked before.

1. Select **Option 1**.
2. **Enter Name:** Type the product name (e.g., "Hand Soap").
3. **Select Category:** Choose the ID corresponding to the department (e.g., 1 for Housekeeping).
4. **Enter Price:** Input the cost per unit (e.g., 1.50).
5. **Enter Reorder Level:** Set the safety stock limit (e.g., 10).
6. *System Confirmation:* "Success! 'Hand Soap' added to inventory with ID 1."

Viewing Inventory

Use this option to check what is currently in the store.

1. Select **Option 2**.
2. The system prints a formatted table.
3. **Check the 'Status' column:**
   - Look for "LOW STOCK!" warnings to identify items that need immediate reordering.

Updating Stock (In/Out)

Use this option when physical goods move.

1. Select **Option 3**.
2. The system first shows the inventory list so you can see the **Product IDs**.
3. **Enter Product ID:** Type the ID of the item you want to update (e.g., 1).
4. **Action Type:**
   - Type IN if you bought more items (Stock Purchase).
   - Type OUT if you are giving items to staff (Stock Issue).
5. **Quantity:** Enter the amount (e.g., 50).
6. *Validation:* If you try to OUT 50 items but only have 10, the system will error: "Insufficient stock! You only have 10."

8. code-structure--explanation:

The application is contained within a single file main.py for portability. Here is a breakdown of the functions:

- **add_product()**:
  - Captures user input.
  - Calculates a new unique ID using len(products) + 1.
  - Constructs a dictionary and appends it to the products list.
- **view_inventory()**:
  - Iterates using a for loop over the products list.
  - Uses Python's f-string formatting (:<20, :<10) to create a clean, aligned table layout.
  - Contains logic if p["stock"] <= p["reorder_level"] to determine the status message.
- **update_stock()**:
  - Locates the specific product dictionary.
  - Modifies the stock integer value directly (+= or -=).
  - Acts as the gatekeeper against invalid transactions (Negative Stock).
- **hotel_inventory_system()**:
  - The "Driver Code".
  - Contains the while True loop that keeps the program running until the user selects "Exit".

9. Testing-instructions:

Since there are no automated unit tests, follow this manual testing script to verify the system integrity:

Test Case 1: The "New Item" Flow

1. Start the app.
2. Add "Test Item" with 0 stock and reorder level 5.
3. View Inventory. Verify "Test Item" exists and Status is **"LOW STOCK!"**.

Test Case 2: The "Stock In" Flow

1. Select Update Stock.
2. Choose "Test Item".
3. Type IN, Quantity 20.
4. View Inventory. Verify Stock is 20 and Status is **"OK"**.

Test Case 3: The "Stock Out" Logic

1. Select Update Stock.
2. Choose "Test Item".
3. Type OUT, Quantity 5.
4. View Inventory. Verify Stock is 15.

Test Case 4: The "Safety" Check

1. Select Update Stock.
2. Choose "Test Item" (Current Stock: 15).

3. Type OUT, Quantity 100.
4. **Expected Result:** System prints "ERROR: Insufficient stock!" and does not change the stock level.

10. Limitations--known-issues:

Data Volatility (In-Memory):

- **Issue:** All data (products, stock counts) is lost immediately when the application is closed or the computer is restarted.
- **Reason:** No file system or database persistence is implemented in this version to keep the code lightweight.

1. **Input Sensitivity:**
   - **Issue:** If the user enters "Ten" instead of "10", the program will crash.
   - **Reason:** Explicit removal of try/except error handling blocks (as per project requirements) means inputs must be perfect.
2. **Single User:**
   - **Issue:** Only one person can use the terminal at a time. It is not networked.

11. future-enhancements:

In the next version (v2.0), we plan to introduce:

1. **File Persistence:** Save data to a inventory_data.json or .csv file so stock levels are remembered after restart.
2. **Search Functionality:** Allow users to search by "Name" (e.g., "Soap") instead of just ID.
3. **Transaction History:** A new menu option to view a log of all past "IN" and "OUT" movements.
4. **GUI:** A graphical interface using Tkinter or PyQt for better user experience.

12. License--contribution:

License:

Contributor:

- **[Your Name]** - Lead Developer & Documentation

Acknowledgements:

- Project guidelines provided by [University Name/Course Name].
- Inspired by real-world Hotel Operations Management workflows.