**Assignment: Python Programming for GUI Development**

Name: R.Siva Thanoj Reddy

Register Number: 192311349

Department: CSE

Date of Submission: 29.8.2024

**Problem 1:** **Inventory Management System Optimization**

**Scenario:**

You have been hired by a retail company to optimize their inventory management system. The company wants to minimize stockouts and overstock situations while maximizing inventory turnover and profitability.

**Tasks:**

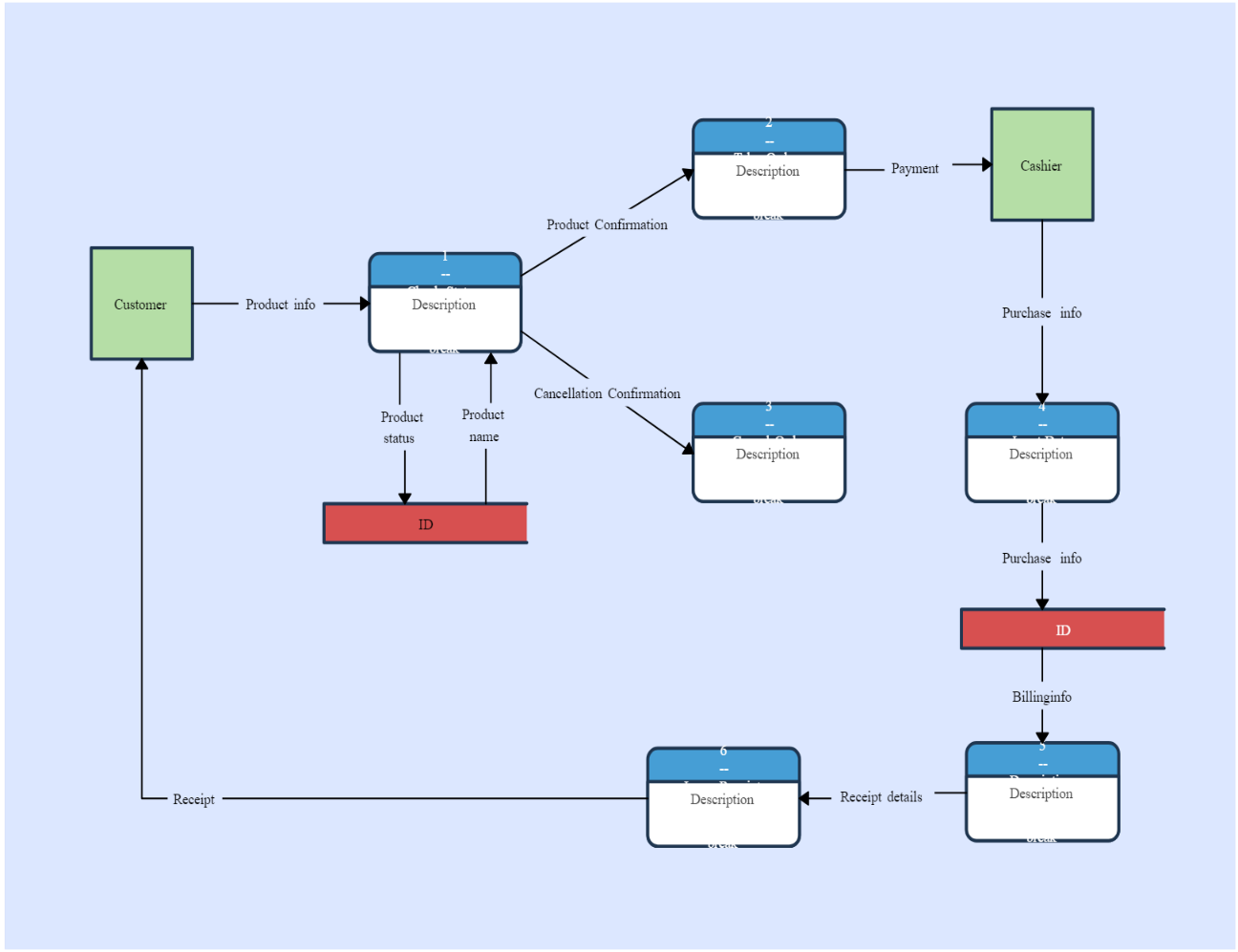
1. **Model the inventory system**: Define the structure of the inventory system, including products, warehouses, and current stock levels.
2. **Implement an inventory tracking application**: Develop a Python application that tracks inventory levels in real-time and alerts when stock levels fall below a certain threshold.
3. **Optimize inventory ordering**: Implement algorithms to calculate optimal reorder points and quantities based on historical sales data, lead times, and demand forecasts.
4. **Generate reports**: Provide reports on inventory turnover rates, stockout occurrences, and cost implications of overstock situations.
5. **User interaction**: Allow users to input product IDs or names to view current stock levels, reorder recommendations, and historical data.

**Deliverables:**

* **Data Flow Diagram**: Illustrate how data flows within the inventory management system, from input (e.g., sales data, inventory adjustments) to output (e.g., reorder alerts, reports).
* **Pseudocode and Implementation**: Provide pseudocode and actual code demonstrating how inventory levels are tracked, reorder points are calculated, and reports are generated.
* **Documentation**: Explain the algorithms used for reorder optimization, how historical data influences decisions, and any assumptions made (e.g., constant lead times).
* **User Interface**: Develop a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.
* **Assumptions and Improvements**: Discuss assumptions about demand patterns, supplier reliability, and potential improvements for the inventory management system's efficiency and accuracy.

# Solution:

Inventory Management System DFD:



# 1.Data Flow Diagram

# 2. Implementation

Track inventory levels and alert when stock levels fall below a threshold.

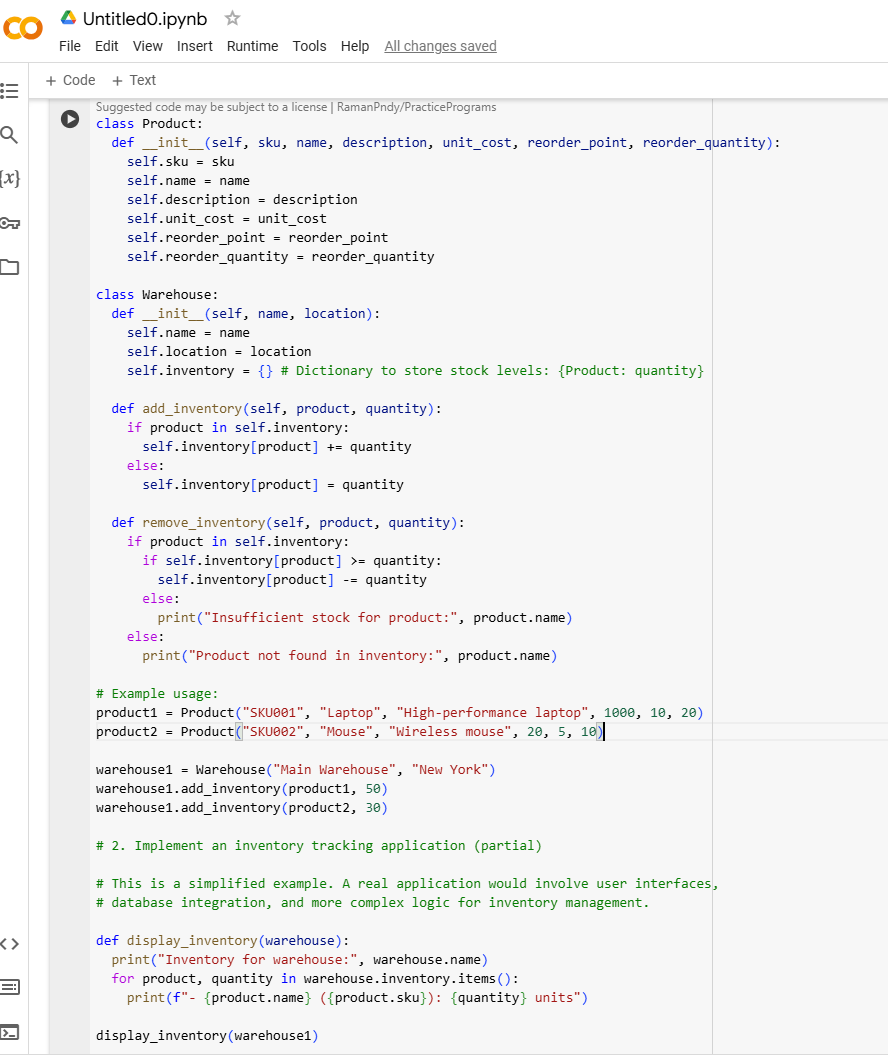
|  |
| --- |
| class InventoryTracking:  def \_\_init\_\_(self, inventory\_system):  self.inventory\_system = inventory\_system  def check\_stock(self, warehouse\_id, product\_id, threshold):  warehouse = self.inventory\_system.warehouses.get(warehouse\_id)  if warehouse:  stock\_level = warehouse.get\_stock\_level(product\_id)  if stock\_level < threshold:  print(f"Alert: Stock for Product ID {product\_id} in Warehouse ID {warehouse\_id} is below threshold! Current level: {stock\_level}") |

# 3.Display the Inventory information:

**- Laptop (SKU001): 50 units**

**- Mouse (SKU002): 30 units**

# 4.User Input:



**5.Documentation:**

**Algorithm Explanation:**

The reorder point is calculated based on the average daily demand and lead time. Safety stock is added to account for demand variability. The reorder quantity is optimized to minimize the total cost, balancing ordering and holding costs.

**Historical Data Influence:**

* Demand forecasts rely on past sales data, which must be accurate and adjusted for seasonality.
* Lead time consistency affects reorder points, so assumptions about supplier reliability should be documented.

**Assumptions and Improvements**

**Assumptions:**

* Lead times and demand are constant for simplicity.
* The Economic Order Quantity (EOQ) model is used for simplicity; real-world scenarios may require more sophisticated methods.

**Improvements:**

* Incorporate more advanced forecasting methods.
* Use historical sales data to dynamically adjust reorder points and quantities.
* Implement integration with real-time sales data for better stock level tracking.
* Use a database to manage products, warehouses, and inventory levels for scalability.

**Conclusion**

Reorder optimization algorithms, influenced by historical data, aim to ensure efficient inventory management by balancing costs, demand, and supply constraints. Assumptions such as constant lead times and demand rates simplify the system but may need periodic adjustment to account for real-world variability. These algorithms are key to maintaining optimal stock levels and avoiding costly stockouts or excess inventory.