**Assignment: Python Programming for GUI Development**

Name: Viyaa M

Register Number: 192372387

Department: CSE (AI)

Date of Submission: 26/08/2024

**PROBLEM 2: 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.

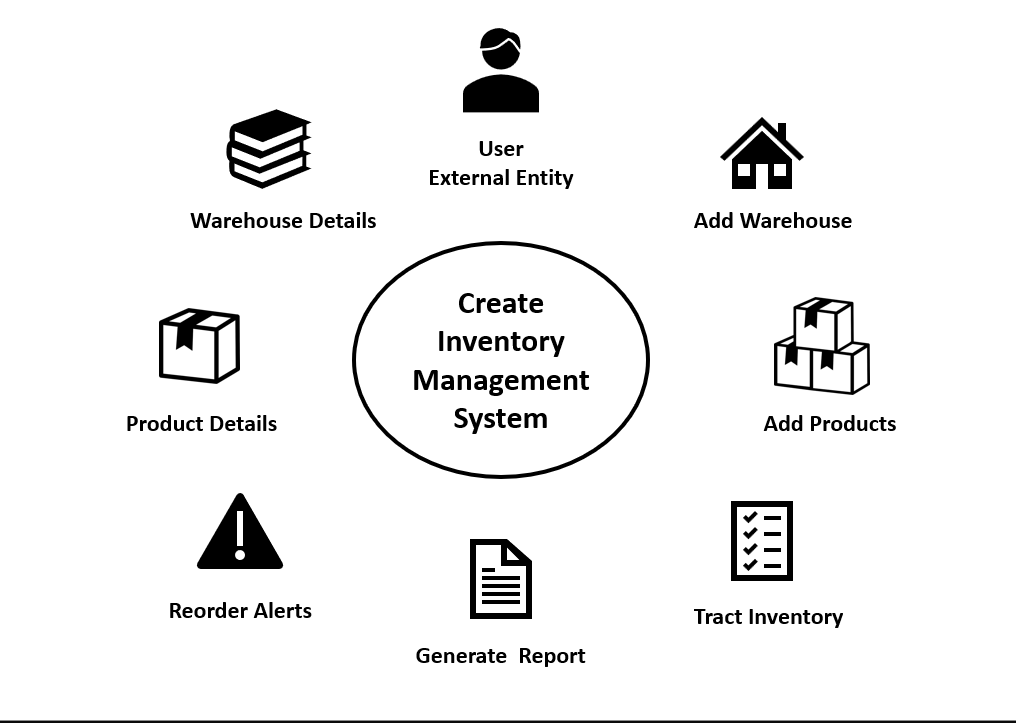
**DELIVERABLES:**

* **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 OPTIMIZATION**

**1. DATA FLOW DIAGRAM:**



**2. PSEUDOCODE AND IMPLIMENTATION:**

**PSEUDOCODE:**

def add\_warehouse(self, warehouse):

self.warehouses[warehouse.warehouse\_id] = warehouse

def track\_inventory(self):

for warehouse in self.warehouses.values():

warehouse.check\_reorder()

def generate\_report(self):

for warehouse in self.warehouses.values():

print(f"Inventory Report for Warehouse {warehouse.warehouse\_id}:")

for product in warehouse.inventory.values():

print(f"Product: {product.name}, Stock Level: {product.stock\_level}, Reorder Point: {product.reorder\_point}")

warehouse1 = Warehouse(1, "Main Warehouse")

product1 = Product(101, "Widget A", 50, 20, 30)

product2 = Product(102, "Widget B", 10, 15, 25)

warehouse1.add\_product(product1)

warehouse1.add\_product(product2)

inventory\_system = InventoryManagementSystem()

inventory\_system.add\_warehouse(warehouse1)

inventory\_system.track\_inventory()

inventory\_system.generate\_report()

**IMPLEMENTATION:**

class Product:

def \_\_init\_\_(self, product\_id, name, stock\_level, reorder\_point, reorder\_quantity):

self.product\_id = product\_id

self.name = name

self.stock\_level = stock\_level

self.reorder\_point = reorder\_point

self.reorder\_quantity = reorder\_quantity

class Warehouse:

def \_\_init\_\_(self, warehouse\_id, location):

self.warehouse\_id = warehouse\_id

self.location = location

self.inventory = {}

def add\_product(self, product):

self.inventory[product.product\_id] = product

def update\_stock(self, product\_id, quantity):

if product\_id in self.inventory:

self.inventory[product\_id].stock\_level += quantity

def check\_reorder(self):

for product in self.inventory.values():

if product.stock\_level < product.reorder\_point:

self.reorder\_product(product)

def reorder\_product(self, product):

print(f"Reorder Alert: {product.name} (ID: {product.product\_id}) needs to be reordered. Suggested quantity: {product.reorder\_quantity}")

class InventoryManagementSystem:

def \_\_init\_\_(self):

self.warehouses = {}

def add\_warehouse(self, warehouse):

self.warehouses[warehouse.warehouse\_id] = warehouse

def track\_inventory(self):

for warehouse in self.warehouses.values():

warehouse.check\_reorder()

def generate\_report(self):

for warehouse in self.warehouses.values():

print(f"Inventory Report for Warehouse {warehouse.warehouse\_id}:")

for product in warehouse.inventory.values():

print(f"Product: {product.name}, Stock Level: {product.stock\_level}, Reorder Point: {product.reorder\_point}")

def get\_product\_info():

product\_id = int(input("Enter Product ID: "))

name = input("Enter Product Name: ")

stock\_level = int(input(f"Enter Stock Level for {name}: "))

reorder\_point = int(input(f"Enter Reorder Point for {name}: "))

reorder\_quantity = int(input(f"Enter Reorder Quantity for {name}: "))

return Product(product\_id, name, stock\_level, reorder\_point, reorder\_quantity)

def main():

inventory\_system = InventoryManagementSystem()

warehouse\_id = int(input("Enter Warehouse ID: "))

location = input("Enter Warehouse Location: ")

warehouse = Warehouse(warehouse\_id, location)

inventory\_system.add\_warehouse(warehouse)

num\_products = int(input("How many products do you want to add? "))

for \_ in range(num\_products):

product = get\_product\_info()

warehouse.add\_product(product)

inventory\_system.track\_inventory()

inventory\_system.generate\_report()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**3.DOCUMENTATION:**

**Reorder Optimization Algorithm**

**Basic Mechanism:**

1. **Reorder Point (ROP):**
   * Each product in the system has a predefined reorder point (ROP). This is the stock level at which the system triggers a reorder.
   * When the stock level of a product falls below this reorder point, the system generates a reorder alert and suggests a quantity to reorder (defined as reorder\_quantity in the product data).
2. **Reorder Quantity:**
   * The reorder quantity is predefined for each product. It’s the quantity that the system suggests ordering when the reorder point is reached.
   * This quantity might be based on factors such as economic order quantity (EOQ), which is a formula used to minimize the total cost of inventory, including holding costs and ordering costs.

**Historical Data Influence**

In more advanced inventory systems, historical data can significantly enhance reorder decisions.

1. **Demand Forecasting:**
   * Historical sales data can be analyzed to predict future demand for each product. This helps in setting more accurate reorder points and reorder quantities.
   * For example, if a product has shown increasing demand over the last few months, the reorder point might be adjusted upwards to prevent stockouts.
2. **Safety Stock Calculation:**
   * Safety stock is an additional quantity of inventory held to reduce the risk of stockouts. Historical data on demand variability and lead times can be used to calculate an appropriate level of safety stock.
3. **Dynamic Reorder Points:**
   * Historical data allows for dynamic adjustment of reorder points based on trends. For instance, seasonal demand variations can be factored into the reorder point.
4. **Lead Time Consideration:**
   * Historical data on supplier lead times (the time taken between placing an order and receiving the stock) can influence reorder points. If lead times are long or inconsistent, the reorder point may be set higher.

**Assumptions in the Algorithm**

**Constant Lead Times:**

* The algorithm assumes that the lead time (the time between ordering and receiving inventory) is constant and known. This simplifies the calculation of reorder points.
* In reality, lead times can vary, and assuming a constant lead time might lead to either stockouts (if the lead time is longer than expected) or overstock (if the lead time is shorter).

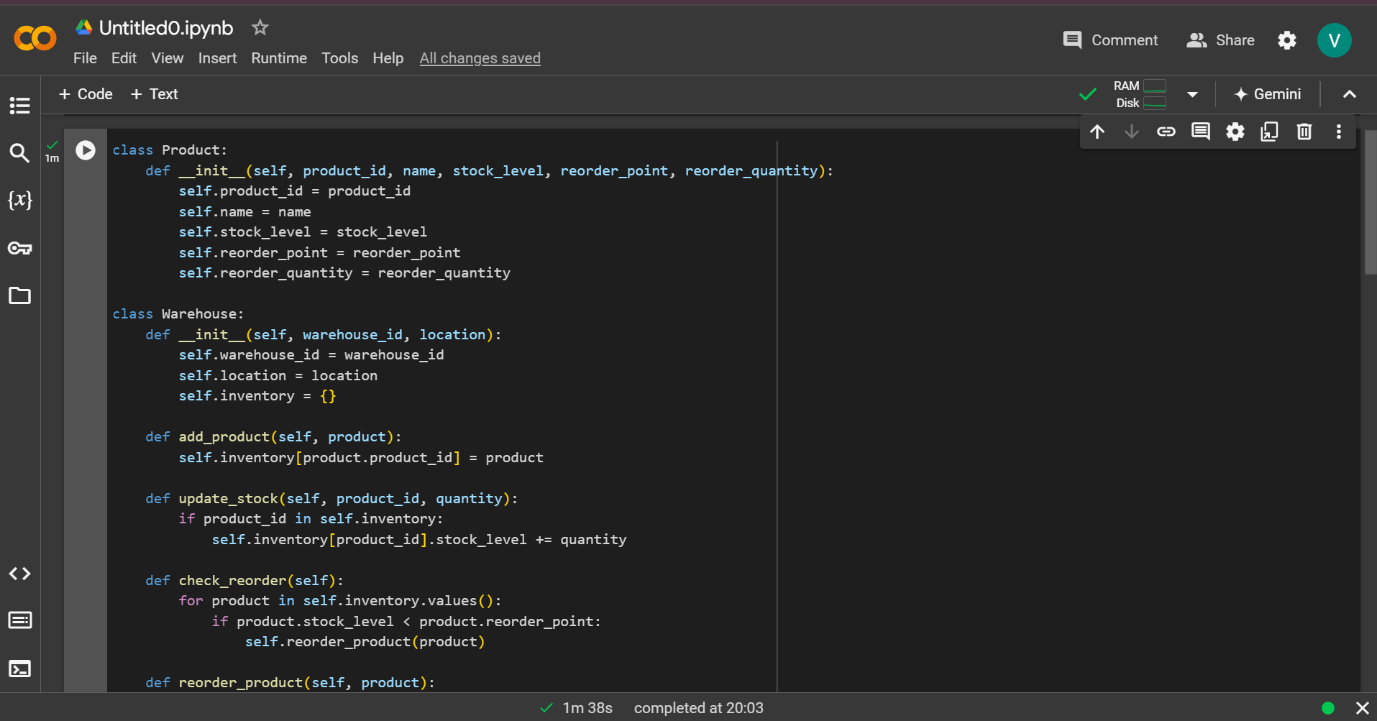
**4. USER INTERFACE:**

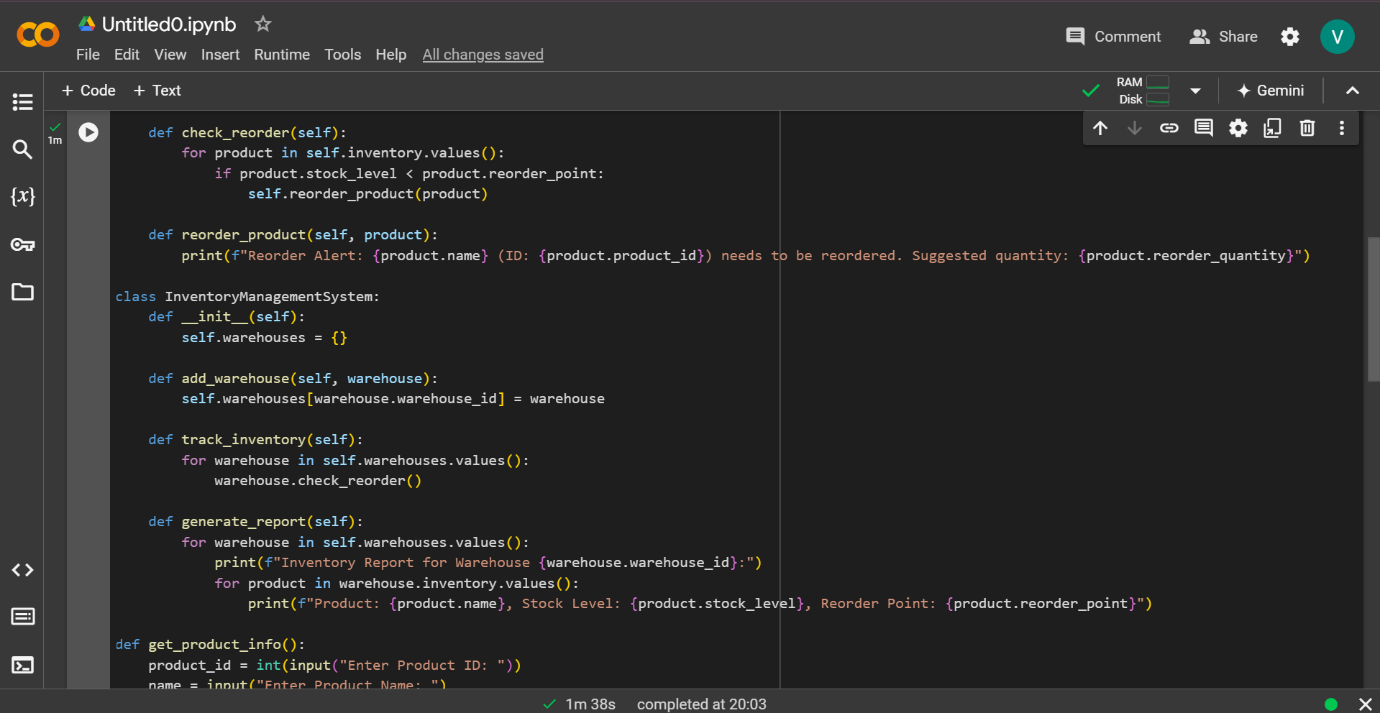
**User interactive program:**

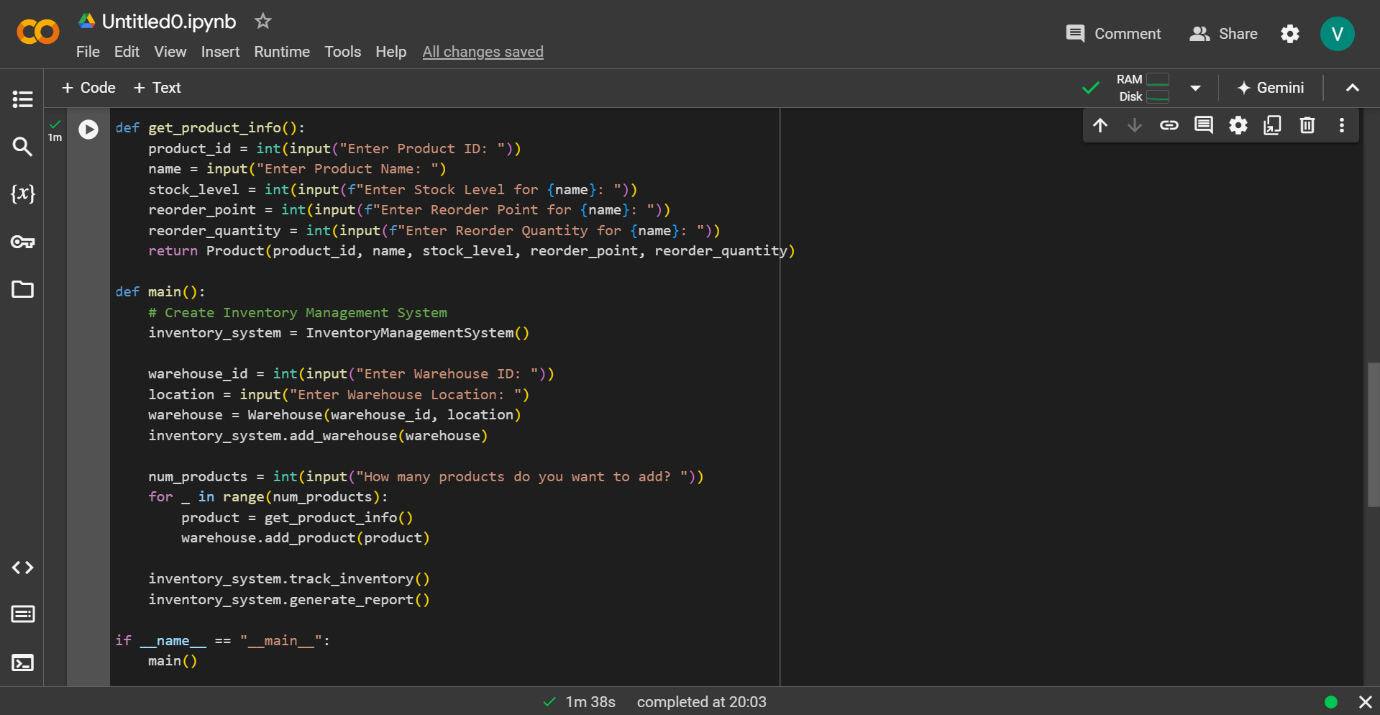
This program allows the user to inter the input values and allows the user to interact freely with the program even though the user would have to give a lot of inputs for the working of the program

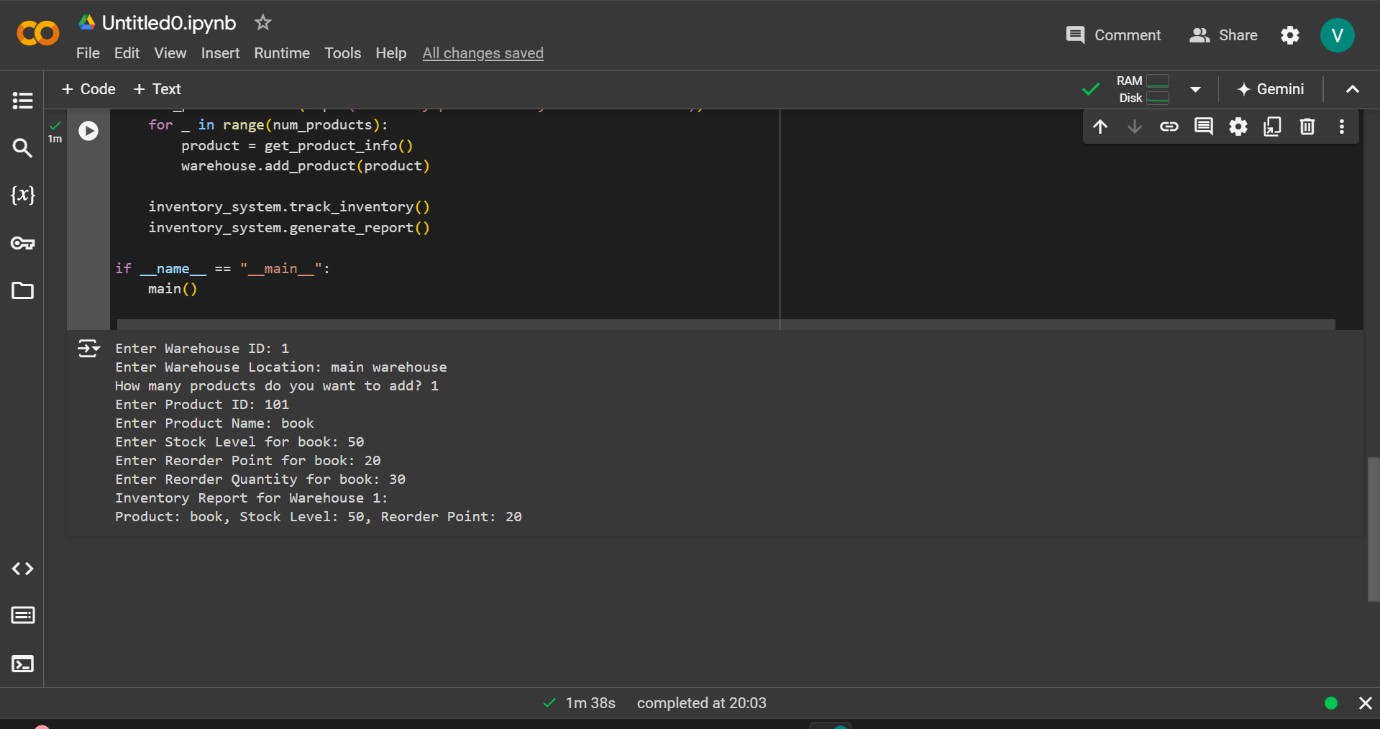
**Sample input and output:**

**(from google colab)**

****

****

****



**5. ASSUMPTIONS AND IMPROVEMENTS:**

**Assumptions in the Inventory Management System**

1. **Assumption about Demand Patterns:**
   * **Constant Demand:** The system implicitly assumes that demand for each product is relatively constant or predictable. The reorder point and reorder quantity are fixed values, suggesting that the system expects a stable demand pattern without significant fluctuations.
   * **No Seasonality:** The system does not account for seasonal variations or trends in demand, which could lead to either stockouts or excess inventory during periods of unexpected demand changes.
2. **Assumption about Supplier Reliability:**
   * **Consistent Lead Times:** The system assumes that lead times—the time between placing an order and receiving the goods—are constant and known. This assumption simplifies the reorder decision but does not account for variability in supplier performance.
   * **Uninterrupted Supply Chain:** The system operates under the assumption that suppliers are always able to fulfill orders as expected, without delays, shortages, or other disruptions.
3. **Assumption about Reorder Quantities:**
   * **Fixed Reorder Quantities:** The system uses fixed reorder quantities for each product, assuming that these quantities are optimal regardless of changing circumstances such as bulk order discounts, storage constraints, or short-term demand spikes.

**Potential Improvements for Efficiency and Accuracy**

1. **Incorporate Demand Forecasting:**
   * **Use Historical Data:** Implementing demand forecasting models that utilize historical sales data can help predict future demand more accurately. This would allow for dynamic adjustment of reorder points and quantities based on expected changes in demand.
   * **Seasonality and Trends:** Incorporate seasonality and trend analysis to adjust inventory levels proactively. For instance, increasing stock levels before a known peak season or adjusting reorder points based on long-term trends.
2. **Enhance Supplier Management:**
   * **Variable Lead Times:** Introduce mechanisms to account for variable lead times, such as calculating safety stock based on historical lead time variability. This would help mitigate the risk of stockouts due to late deliveries.
   * **Supplier Reliability Metrics:** Track supplier performance over time and adjust reorder strategies accordingly. Reliable suppliers might allow for lower safety stock, while unreliable suppliers might necessitate higher stock levels or alternate suppliers.
3. **Optimize Reorder Quantities:**
   * **Dynamic Reorder Quantities:** Instead of using fixed reorder quantities, the system could implement an Economic Order Quantity (EOQ) model that minimizes the total cost of inventory, including holding costs, ordering costs, and stockout costs.
   * **Bulk Order Discounts:** Consider bulk order discounts offered by suppliers. The system could automatically adjust reorder quantities to take advantage of these discounts, balancing cost savings with storage capacity and demand forecasts.
4. **Implement Safety Stock:**
   * **Safety Stock Calculation:** Introduce safety stock levels that account for demand variability and lead time uncertainty. This buffer stock helps protect against unexpected demand spikes or delays in supply.
   * **Dynamic Safety Stock:** Adjust safety stock levels dynamically based on recent demand patterns, supplier performance, and lead time variability.
5. **Leverage Technology for Real-Time Updates:**
   * **Real-Time Inventory Tracking:** Implement real-time tracking of inventory levels, possibly integrating with automated ordering systems that trigger reorders as soon as stock levels approach the reorder point.
   * **Automated Supplier Communication:** Use automated systems to communicate with suppliers, ensuring that orders are placed immediately when needed and confirming order status to anticipate potential delays.