**Problem 2: Inventory Management System Optimization**

**Scenario:**

**A retail company has enlisted your help to optimize their inventory management system. They aim to reduce stockouts and overstock scenarios while enhancing 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 to monitor inventory levels in real-time and send alerts when stock levels drop below a specified threshold.**

**3. Optimize inventory ordering: Create algorithms to calculate optimal reorder points and quantities based on historical sales data, lead times, and demand forecasts.**

**4. Generate reports: Produce reports on inventory turnover rates, stockout occurrences, and the cost implications of overstock situations.**

**5. User interaction: Enable users to input product IDs or names to view current stock levels, reorder recommendations, and historical data.**

**Deliverables:**

**Data Flow Diagram: Show how data moves within the inventory management system, from inputs (e.g., sales data, inventory adjustments) to outputs (e.g., reorder alerts, reports).**

**Pseudocode and Implementation: Provide pseudocode and actual code that demonstrate how inventory levels are tracked, reorder points are calculated, and reports are generated.**

**Documentation: Explain the algorithms used for reorder optimization, the influence of historical data on decisions, and any assumptions made (e.g., constant lead times).**

**User Interface: Create a user-friendly interface for accessing inventory information, viewing reports, and receiving alerts.**

**Assumptions and Improvements: Discuss assumptions about demand patterns, supplier reliability, and suggest potential improvements for enhancing the inventory management system's efficiency and accuracy.**

**Flow diagram**

+----------------------------+

| Sales Data |

+----------------------------+

|

| (1) Sales Transactions

v

+----------------------------+

| Inventory Management System |

+----------------------------+

|

| (2) Update Stock Levels

v

+--------------------------------------+

| Database |

| - Products |

| - Warehouses |

| - Current Stock Levels |

| - Historical Sales Data |

+--------------------------------------+

|

+----------------------+--------------------+

| |

v v

+-----------------------------+ +---------------------------+

| Inventory Tracking Module | | Inventory Optimization |

+-----------------------------+ | Module |

| +---------------------------+

| | |

| | (4) Calculate Reorder Points |

v | and Quantities |

+-----------------------------+ | |

| User Interface (UI) | v |

| - Input Product IDs/Names | <------- +---------------------------+

| - View Current Stock Levels | | Reporting Module |

| - View Reorder Recommendations| (3) Reorder | - Generate Inventory |

| - View Historical Data | Alerts | Turnover Reports |

| - View Reports | | - Generate Stockout |

+-----------------------------+ | Occurrences Reports |

| - Generate Overstock Cost |

| Reports |

**Python Code**

class Product:

def \_\_init\_\_(self, product\_id, name, category, current\_stock, reorder\_point, reorder\_quantity, lead\_time):

self.product\_id = product\_id

self.name = name

self.category = category

self.current\_stock = current\_stock

self.reorder\_point = reorder\_point

self.reorder\_quantity = reorder\_quantity

self.lead\_time = lead\_time

self.historical\_sales = []

def add\_sales\_data(self, sales):

self.historical\_sales.append(sales)

self.current\_stock -= sales # Update current stock after sale

class Warehouse:

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

self.warehouse\_id = warehouse\_id

self.location = location

self.products = {}

def add\_product(self, product):

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

def track\_inventory(self):

print("\n--- Inventory Tracking ---")

for product in self.products.values():

if product.current\_stock < product.reorder\_point:

print(f"Reorder Alert for: {product.name}")

print(f"Current Stock: {product.current\_stock}")

print(f"Recommended Order Quantity: {product.reorder\_quantity}")

else:

print(f"{product.name} is sufficiently stocked.")

def generate\_report(self):

print("\n--- Inventory Report ---")

for product in self.products.values():

print(f"Product: {product.name}")

print(f"Current Stock: {product.current\_stock}")

print(f"Turnover Rate: {self.calculate\_turnover\_rate(product)}")

def calculate\_turnover\_rate(self, product):

total\_sales = sum(product.historical\_sales)

average\_stock = (product.current\_stock + product.current\_stock) / 2

turnover\_rate = total\_sales / average\_stock if average\_stock > 0 else 0

return turnover\_rate

def calculate\_eoq(annual\_demand, ordering\_cost, holding\_cost):

if holding\_cost > 0:

return (2 \* annual\_demand \* ordering\_cost / holding\_cost) \*\* 0.5

return 0

def main():

warehouse = Warehouse(1, "Main Warehouse")

# Example products

product1 = Product(101, "Laptop", "Electronics", 15, 5, 20, 2)

product1.add\_sales\_data(3)

product1.add\_sales\_data(4)

product2 = Product(102, "Smartphone", "Electronics", 30, 10, 15, 3)

product2.add\_sales\_data(5)

product2.add\_sales\_data(6)

warehouse.add\_product(product1)

warehouse.add\_product(product2)

while True:

print("\nOptions:")

print("1. Track Inventory")

print("2. Generate Report")

print("3. Add Sales Data")

print("4. Calculate EOQ")

print("5. Exit")

choice = input("Select an option: ")

if choice == '1':

warehouse.track\_inventory()

elif choice == '2':

warehouse.generate\_report()

elif choice == '3':

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

sales = int(input("Enter sales data: "))

if product\_id in warehouse.products:

warehouse.products[product\_id].add\_sales\_data(sales)

print("Sales data updated.")

else:

print("Product not found.")

elif choice == '4':

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

if product\_id in warehouse.products:

annual\_demand = sum(warehouse.products[product\_id].historical\_sales)

ordering\_cost = 50 # Example ordering cost

holding\_cost = 2 # Example holding cost per unit

eoq = calculate\_eoq(annual\_demand, ordering\_cost, holding\_cost)

print(f"Optimal Order Quantity (EOQ) for {warehouse.products[product\_id].name}: {eoq:.2f}")

else:

print("Product not found.")

elif choice == '5':

break

else:

print("Invalid option, try again.")

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

main()

Pseudocode:

1. Define the `Product` class:

- Initialize with `product\_id`, `name`, and `current\_stock`.

- Initialize an empty list for `historical\_sales`.

2. Define the `Warehouse` class:

- Initialize with `warehouse\_id` and `name`.

- Initialize an empty dictionary for inventory.

3. Define the `InventorySystem` class:

- Initialize with empty dictionaries for `products` and `warehouses`.

4. Define the `add\_product(product)` method:

- Add the product to the `products` dictionary.

5. Define the `add\_warehouse(warehouse)` method:

- Add the warehouse to the `warehouses` dictionary.

6. Define the `update\_stock(product\_id, warehouse\_id, quantity)` method:

- Retrieve the warehouse using `warehouse\_id`.

- Update the inventory for `product\_id` in the warehouse.

- Retrieve the product using `product\_id`.

- Update the product's `current\_stock`.

7. Define the `record\_sale(product\_id, quantity)` method:

- Retrieve the product using `product\_id`.

- Decrease the product's `current\_stock` by `quantity`.