NAME:SENTHIL NATHAN S
DEPARTMENT:II-ECE
NM ID:aut1133eca

AI-Enhanced Supply Chain Optimization

Objective

The objective of Phase 3 is to implement key components of the AI-Enhanced Supply Chain Optimization platform as envisioned in Phase 2. This includes deploying AI for demand forecasting and supplier risk analysis, setting up IoT-based tracking infrastructure, piloting blockchain for procurement, and implementing basic data security protocols.

1. AI Model Development

AI models play a critical role in optimizing inventory levels, predicting demand, and evaluating supplier risk.

- Demand Forecasting Model: A time-series forecasting AI trained on historical sales, external market data,and real-time inputs. The model incorporates adaptive learning to refine predictions.
- Supplier Risk Analysis: Uses NLP to extract insights from vendor reports and performance data, generatinga dynamic risk score.

Outcome:
Forecasting product demand with reasonable accuracy and identifying high-risk suppliers.

2. IoT Integration for Real-Time Visibility

IoT devices improve logistics transparency by enabling real-time tracking and predictive alerts for goods in transit.

- Device Setup: Basic integration of GPS and RFID sensors on sample shipments.
- Tracking Dashboard: A centralized dashboard to visualize shipment status and generate delay alerts.

Outcome:

The system tracks shipments and notifies users of delays or deviations.

3. Blockchain Pilot for Secure Transactions

Blockchain enhances trust and transparency in procurement by creating immutable, auditable records.

- Smart Contract Templates: Basic smart contracts created to simulate procurement agreements.
- Blockchain Ledger: A private blockchain pilot records transaction data between stakeholders.

Outcome:

Simulated secure procurement transactions using blockchain technology.

4. Data Security Implementation

Given the sensitivity of supply chain data, basic data protection measures are essential.

- Encryption: All user inputs, transaction logs, and forecasting data will be stored using AES encryption.
- Access Controls: Admin roles and permission levels will be implemented in the platform.

Outcome:

Confidential data protected and restricted access ensured.

5. Testing and Feedback Collection

Initial testing will validate the functionality, accuracy, and usability of the platform.

- Test Scenarios: Realistic test cases for forecasting, shipment tracking, and supplier scoring.
- Feedback Forms: Stakeholders provide input on system performance and user experience.

Outcome:

Collected feedback informs refinements and scalability planning.

Challenges and Solutions

1. Data Integration: Use middleware and standardized APIs.

2. Blockchain Complexity: Begin with simple smart contracts and expand gradually.
3. Device Availability: Simulate real-time tracking if hardware is unavailable.

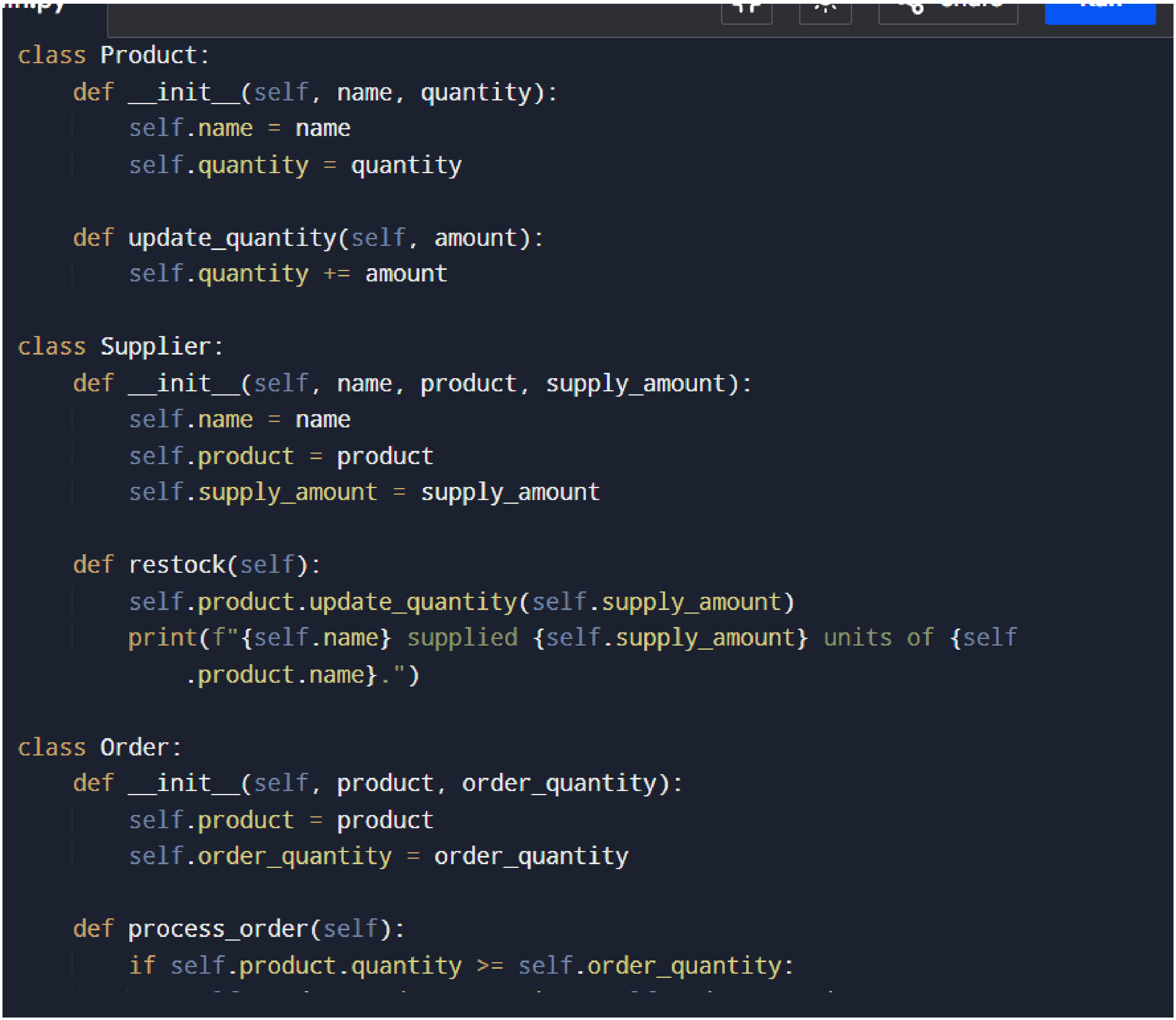
Outcomes of Phase 3

1. Functional AI Modules: Demand forecasting and supplier risk scoring.
2. IoT Integration: Real-time visibility on sample shipments.3. Blockchain Pilot: Procurement contracts stored securely.
4. Data Security: Encrypted data and access controls in place.
5. Initial User Feedback: Guides enhancements in Phase 4.

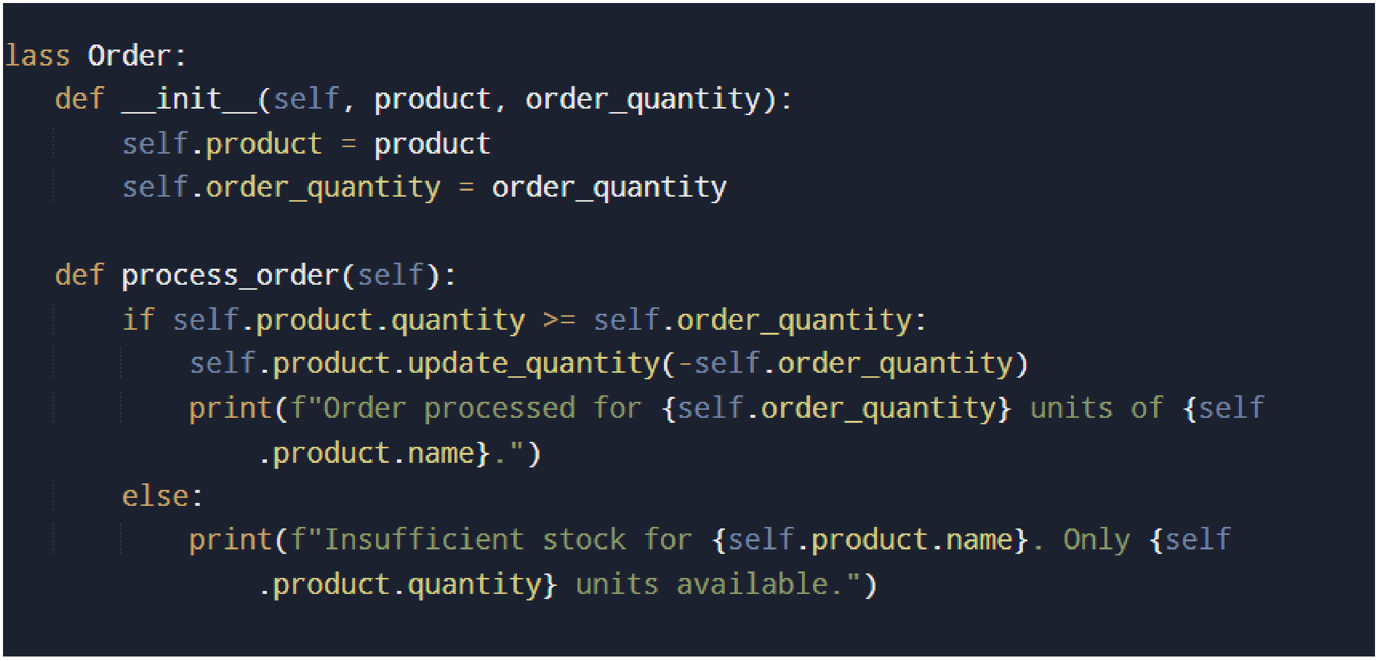
Next Steps for Phase 4

1. Model Refinement: Improve AI accuracy.
2. IoT Expansion: Broader deployment.
3. Blockchain Scaling: Contracts and payments.
4. Scalability Testing: Enterprise-level readiness.

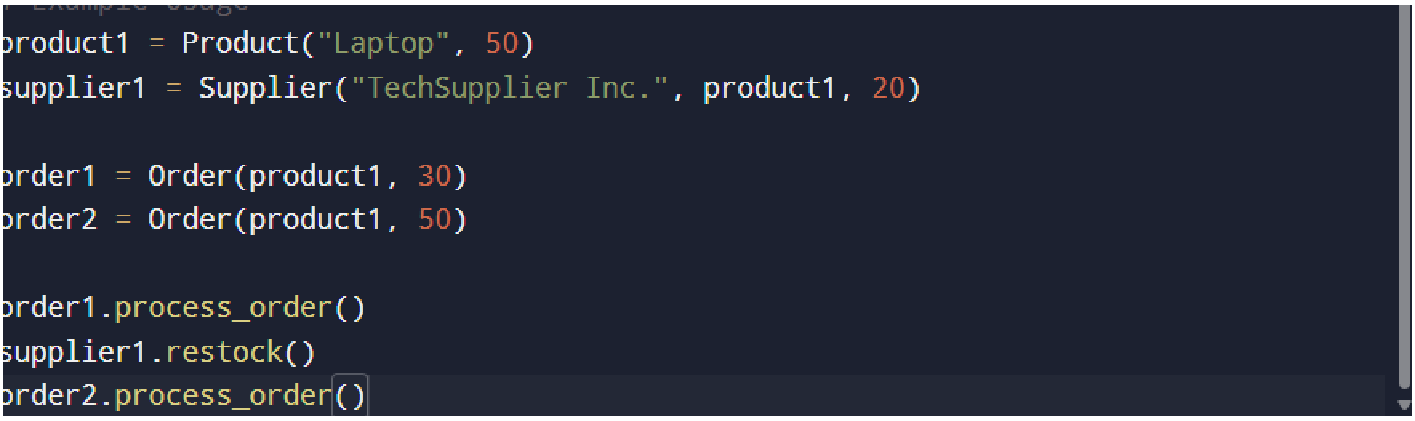
PROGRAM FOR THE MODEL:



```
class Product:
    def __init__(self, name, quantity):
        self.name = name
        self.quantity = quantity

    def update_quantity(self, amount):
        self.quantity += amount

class Supplier:
    def __init__(self, name, product, supply_amount):
        self.name = name
        self.product = product
        self.supply_amount = supply_amount

    def restock(self):
        self.product.update_quantity(self.supply_amount)
        print(f"{self.name} supplied {self.supply_amount} units of {self
            .product.name}.")

class Order:
    def __init__(self, product, order_quantity):
        self.product = product
        self.order_quantity = order_quantity

    def process_order(self):
        if self.product.quantity >= self.order_quantity:
```

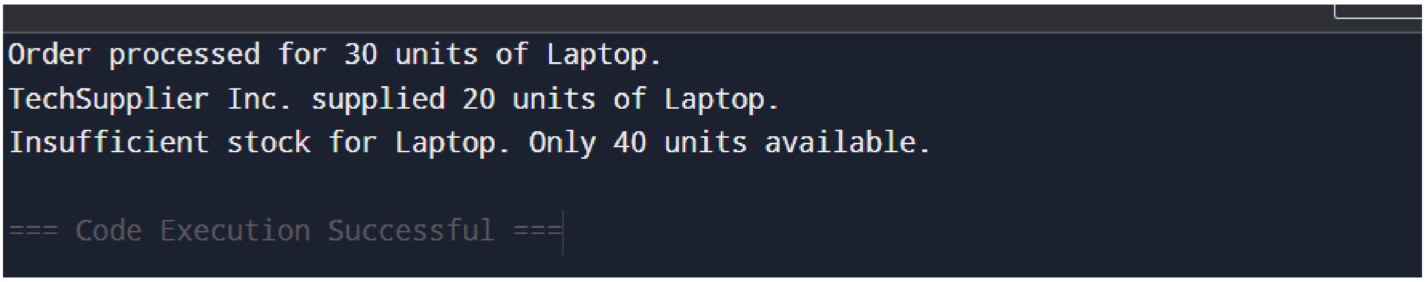


```
lass Order:
   def __init__(self, product, order_quantity):
       self.product = product
       self.order_quantity = order_quantity

   def process_order(self):
       if self.product.quantity >= self.order_quantity:
           self.product.update_quantity(-self.order_quantity)
           print(f"Order processed for {self.order_quantity} units of {self
               .product.name}.")
       else:
           print(f"Insufficient stock for {self.product.name}. Only {self
               .product.quantity} units available.")
```



```python
# Example usage
product1 = Product("Laptop", 50)
supplier1 = Supplier("TechSupplier Inc.", product1, 20)

order1 = Order(product1, 30)
order2 = Order(product1, 50)

order1.process_order()
supplier1.restock()
order2.process_order()
```

OUTPUT:



```
Order processed for 30 units of Laptop.
TechSupplier Inc. supplied 20 units of Laptop.
Insufficient stock for Laptop. Only 40 units available.

=== Code Execution Successful ===
```