SWENG 837 Software System Design

Supply Chain Management System

Professor Santosh Nalubandhu

Kevin Malone

Table of Contents

Problem Statement and Requirements

Business Requirements

Problem Definition

Current supply chains lack transparency and traceability, leading to inefficiencies, fraud, and reduced consumer trust.

System Functionalities

- Record Creation: Allow manufacturers to record the creation of products on the blockchain.
- Tracking: Enable tracking of products through the supply chain stages (e.g., shipping, warehousing, retail).
- Verification: Provide a verification mechanism for consumers to authenticate product origin and journey.
- Inventory Management: Support real-time inventory management for stakeholders.
- Reporting: Generate reports for stakeholders on product status, location, and history.

Target Users

- Manufacturers: Record production details and initiate the supply chain.
- Transporters: Update the status of goods during transportation.
- Warehouse Managers: Track inventory and manage stock levels.
- Retailers: Update the arrival of products and manage sales.
- Consumers: Verify product authenticity and trace its journey.

Business Goals

- Increase transparency and trust in the supply chain.
- Improve efficiency by reducing delays and errors.
- Enhance product traceability from origin to consumer.
- Support compliance with regulations and standards.

Non-Functional Requirements

Performance Requirements

- Scalability: The system should handle a large number of transactions and users.
- Response Time: Ensure minimal response time for data retrieval and transactions.
- Throughput: Support high transaction throughput for real-time updates.

Security Requirements

- Authentication: Use robust authentication mechanisms for user access.
- Authorization: Implement role-based access control to ensure data integrity.
- Data Encryption: Encrypt sensitive data both at rest and in transit.

Maintainability Requirements

- Code Modularity: Ensure the code is modular for easy updates and maintenance.
- Documentation: Provide comprehensive documentation for all system components.
- Testing Strategies: Implement unit, integration, and system testing to ensure reliability.

Other Non-Functional Requirements

- Reliability: Ensure the system is reliable with minimal downtime.
- Usability: Design the system to be user-friendly for all stakeholders.
- Compliance: Ensure the system complies with industry standards and regulations.

System Design using Domain Modeling

UML Use Case Diagrams

Create Product Record UC

Use Case Section	Comment
Use Case Name	Create Product Record
Scope	Supply Chain Management System
Level	Primary Function
Primary Actor	Manufacturer
Stakeholder and Interests	**Manufacturer:** Needs to record production details to initiate the supply chain process.
Preconditions	The manufacturer must be authenticated and authorized.
Success Guarantee	The product record is successfully created and stored on the blockchain.
Main Success Scenario	1. Manufacturer logs into the system. 2. Enters product details. 3. System records the product on the blockchain.
Extensions	3a. If there are incomplete details, the system prompts for missing information. 3b. If the blockchain is unavailable, the system queues the request for later.
Special Requirements	System must validate product details and ensure the integrity of blockchain transactions.

Update Product Location UC

Use Case Section	Comment
Use Case Name	Update Product Location
Scope	Supply Chain Management System
Level	Primary Function
Primary Actor	Transporter
Stakeholder and Interests	**Transporter:** Needs to update the status and location of goods during transportation.
Preconditions	The transporter must be authenticated and authorized.
Success Guarantee	The product location and status are successfully updated on the blockchain.
Main Success Scenario	1. Transporter logs into the system. 2. Enters location and status details. 3. System updates the blockchain with new information.
Extensions	3a. If the location details are incomplete, the system prompts for missing information. 3b. If the blockchain is unavailable, the system queues the request for later.
Special Requirements	System must ensure the accuracy and timeliness of location updates.

Manage Inventory UC

Use Case Section	Comment
Use Case Name	Manage Inventory
Scope	Supply Chain Management System
Level	Primary Function
Primary Actor	Warehouse Manager
Stakeholder and Interests	**Warehouse Manager:** Needs to track inventory and manage stock levels efficiently.
Preconditions	The warehouse manager must be authenticated and authorized.
Success Guarantee	Inventory details are successfully updated and tracked in the system.
Main Success Scenario	1. Warehouse Manager logs into the system. 2. Updates inventory details. 3. System records the inventory changes.
Extensions	3a. If the inventory details are incomplete, the system prompts for missing information. 3b. If the system is unavailable, the changes are queued for later.
Special Requirements	System must validate inventory changes and ensure data consistency.

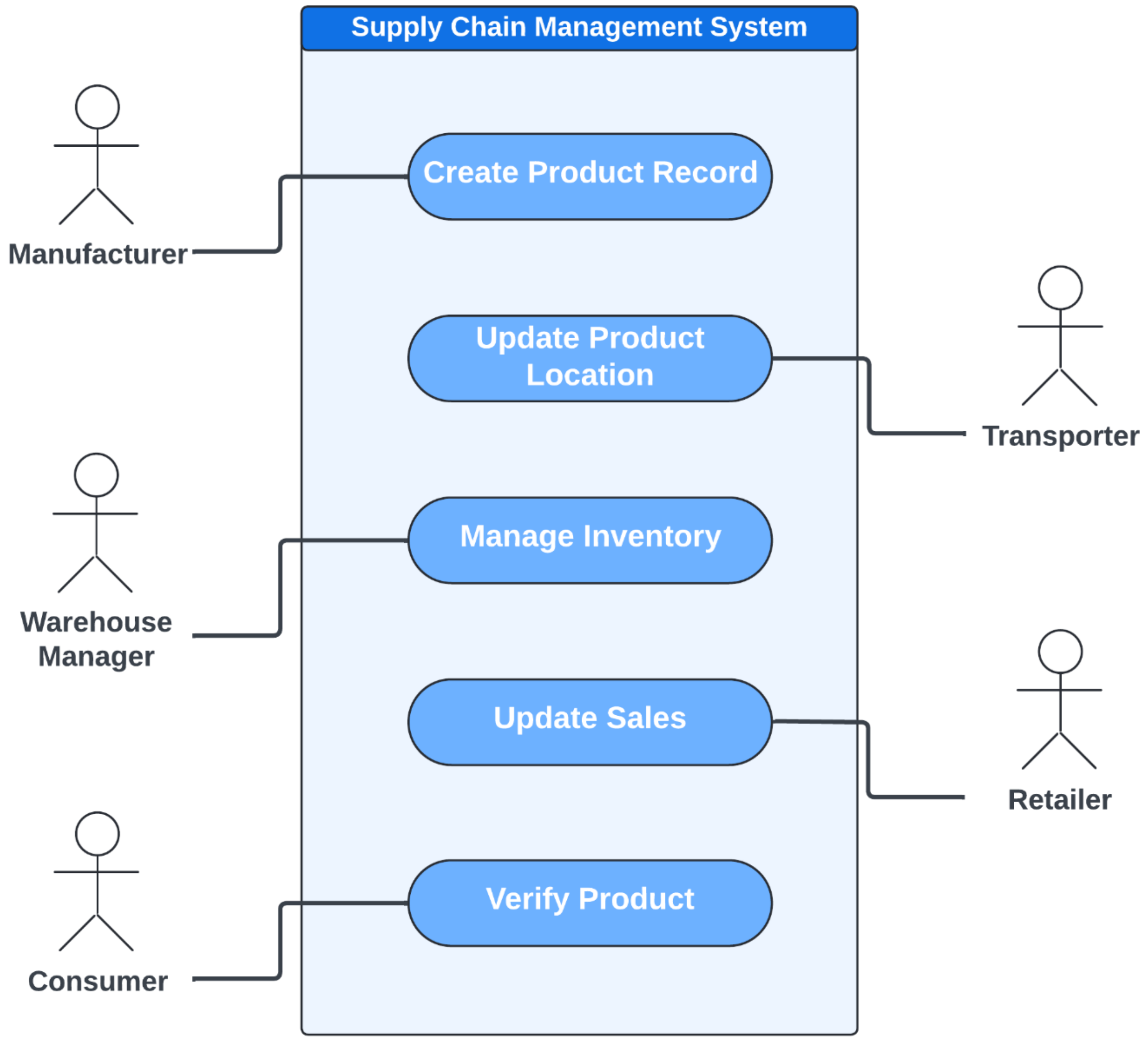
Update Sales UC

Use Case Section	Comment
Use Case Name	Update Sales
Scope	Supply Chain Management System
Level	Primary Function
Primary Actor	Retailer
Stakeholder and Interests	**Retailer:** Needs to update product sales and manage sales data.
Preconditions	The retailer must be authenticated and authorized.
Success Guarantee	Sales details are successfully updated and recorded in the system.
Main Success Scenario	1. Retailer logs into the system. 2. Enters sales details. 3. System updates sales records.
Extensions	3a. If the sales details are incomplete, the system prompts for missing information. 3b. If the system is unavailable, the changes are queued for later.
Special Requirements	System must validate sales entries and ensure data accuracy.

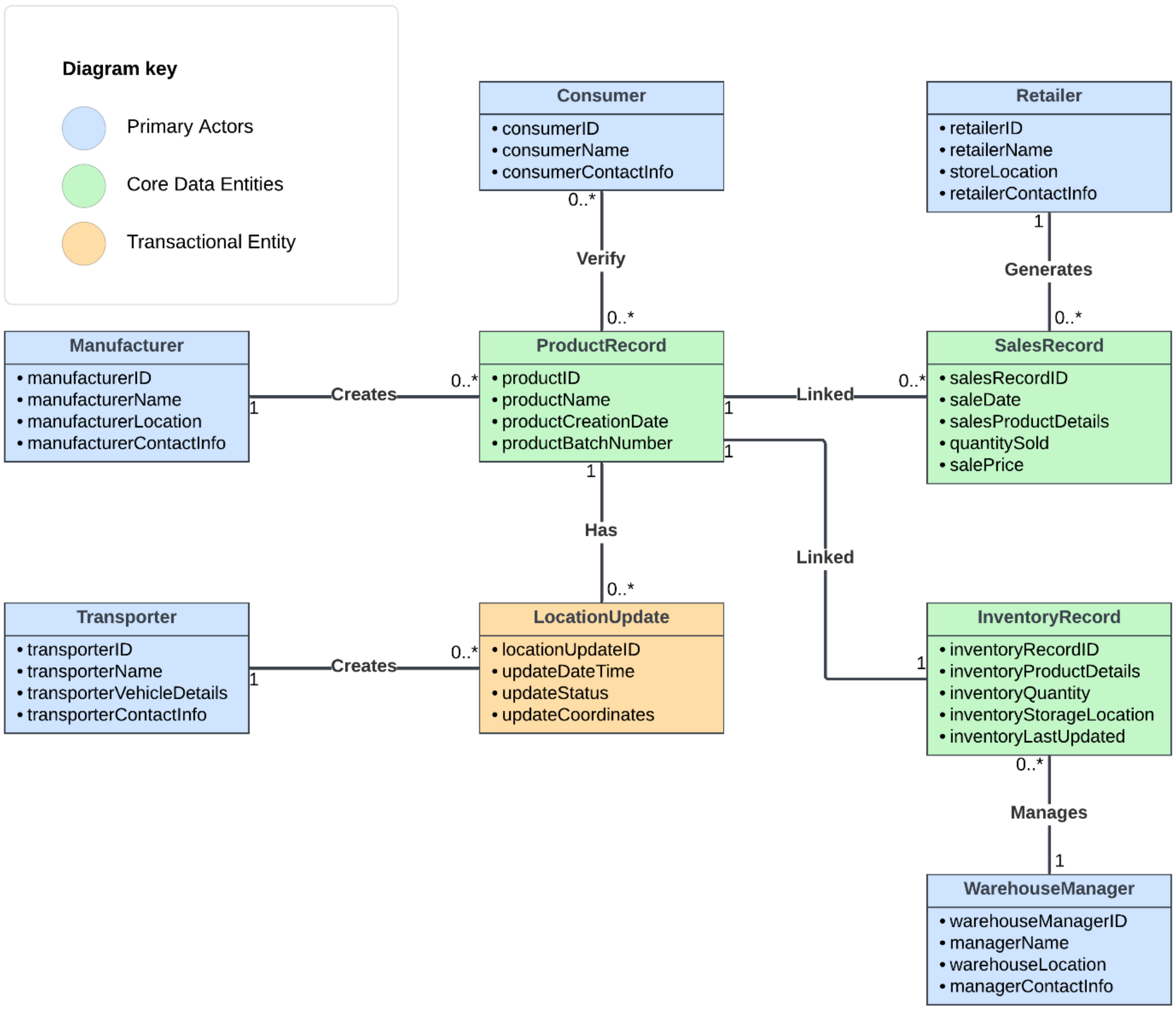
Verify Product UC

Use Case Section	Comment
Use Case Name	Verify Product
Scope	Supply Chain Management System
Level	Primary Function
Primary Actor	Consumer
Stakeholder and Interests	**Consumer:** Needs to verify product authenticity and trace its journey.
Preconditions	The consumer must have access to product information.
Success Guarantee	Product details are successfully verified and displayed to the consumer.
Main Success Scenario	1. Consumer accesses the system. 2. Enters product details. 3. System retrieves and displays the product information.
Extensions	3a. If the product details are incomplete, the system prompts for missing information. 3b. If the system is unavailable, the verification is queued for later.
Special Requirements	System must ensure the security and accuracy of the product verification process.

Use Case Diagram



UML Domain Model



UML Class Diagram

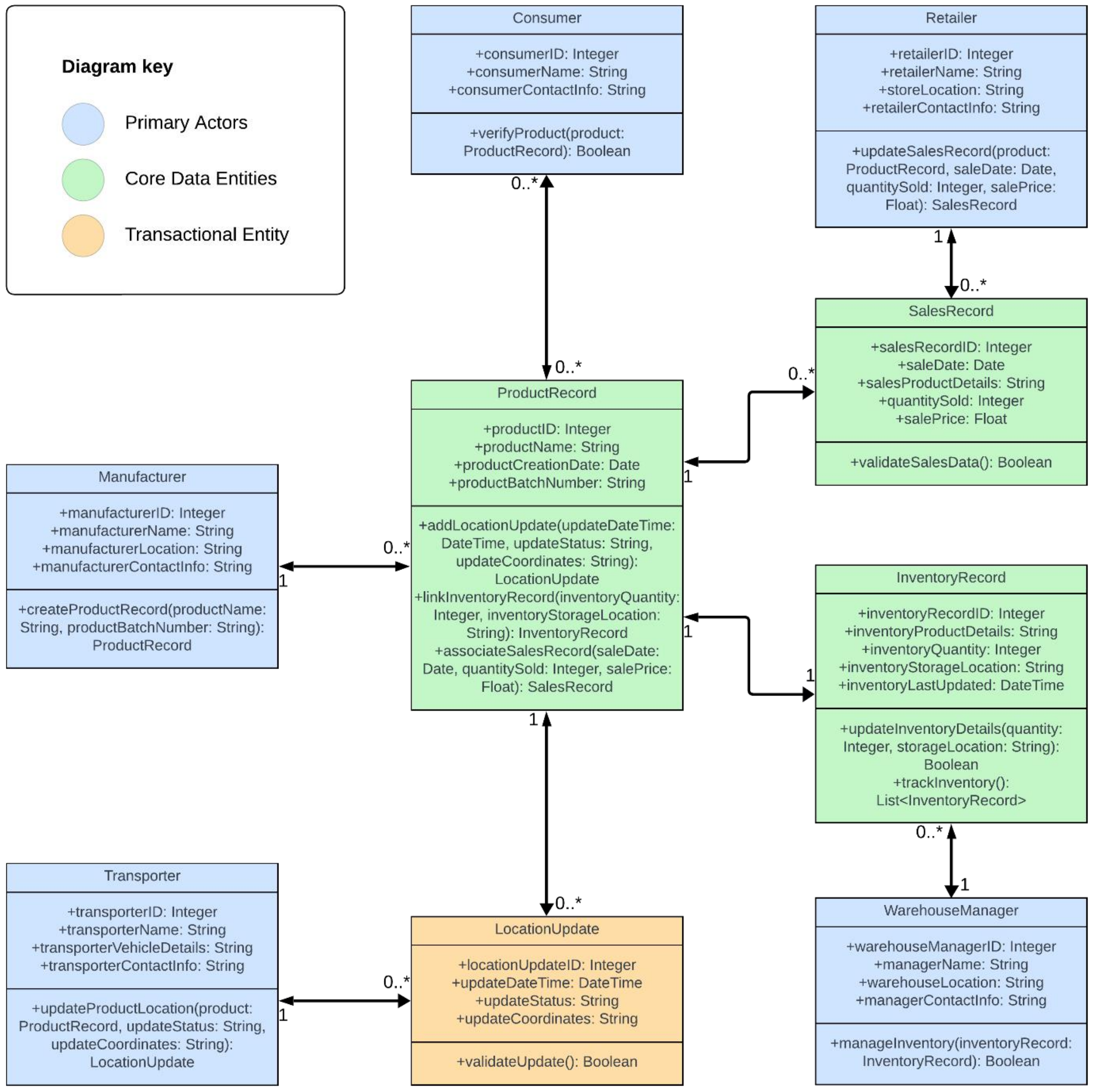
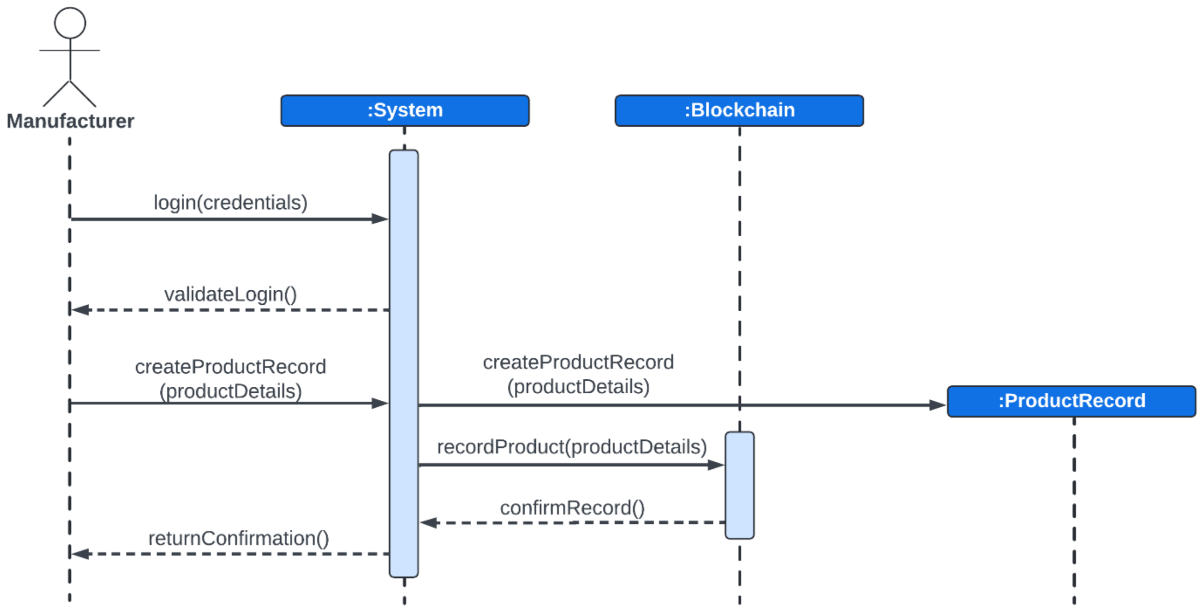


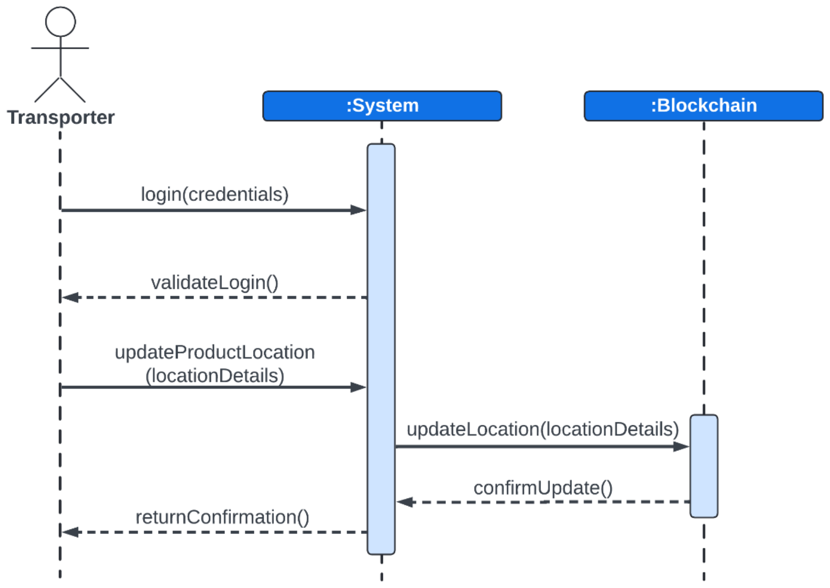
Diagram key
Primary Actors
Core Data Entities
Transactional Entity
Consumer
+consumerID: Integer
+consumerName: String
+consumerContactInfo: String
+verifyProduct(product: ProductRecord): Boolean
Retailer
+retailerID: Integer
+retailerName: String
+storeLocation: String
+retailerContactInfo: String
+updateSalesRecord(product: ProductRecord, saleDate: Date, quantitySold: Integer, salePrice: Float): SalesRecord
1
0..*
SalesRecord
+salesRecordID: Integer
+saleDate: Date
+salesProductDetails: String
+quantitySold: Integer
+salePrice: Float
+validateSalesData(): Boolean
0..*
0..*
ProductRecord
+productID: Integer
+productName: String
+productCreationDate: Date
+productBatchNumber: String
+addLocationUpdate(updateDateTime: DateTime, updateStatus: String, updateCoordinates: String): LocationUpdate
+linkInventoryRecord(inventoryQuantity: Integer, inventoryStorageLocation: String): InventoryRecord
+associateSalesRecord(saleDate: Date, quantitySold: Integer, salePrice: Float): SalesRecord
1
1
1
0..*
Manufacturer
+manufacturerID: Integer
+manufacturerName: String
+manufacturerLocation: String
+manufacturerContactInfo: String
+createProductRecord(productName: String, productBatchNumber: String): ProductRecord
1
0..*
InventoryRecord
+inventoryRecordID: Integer
+inventoryProductDetails: String
+inventoryQuantity: Integer
+inventoryStorageLocation: String
+inventoryLastUpdated: DateTime
+updateInventoryDetails(quantity: Integer, storageLocation: String): Boolean
+trackInventory(): List<InventoryRecord>
1
0..*
1
WarehouseManager
+warehouseManagerID: Integer
+managerName: String
+warehouseLocation: String
+managerContactInfo: String
+manageInventory(inventoryRecord: InventoryRecord): Boolean
1
0..*
Transporter
+transporterID: Integer
+transporterName: String
+transporterVehicleDetails: String
+transporterContactInfo: String
+updateProductLocation(product: ProductRecord, updateStatus: String, updateCoordinates: String): LocationUpdate
1
0..*
0..*
LocationUpdate
+locationUpdateID: Integer
+updateDateTime: DateTime
+updateStatus: String
+updateCoordinates: String
+validateUpdate(): Boolean

UML Sequence Diagrams

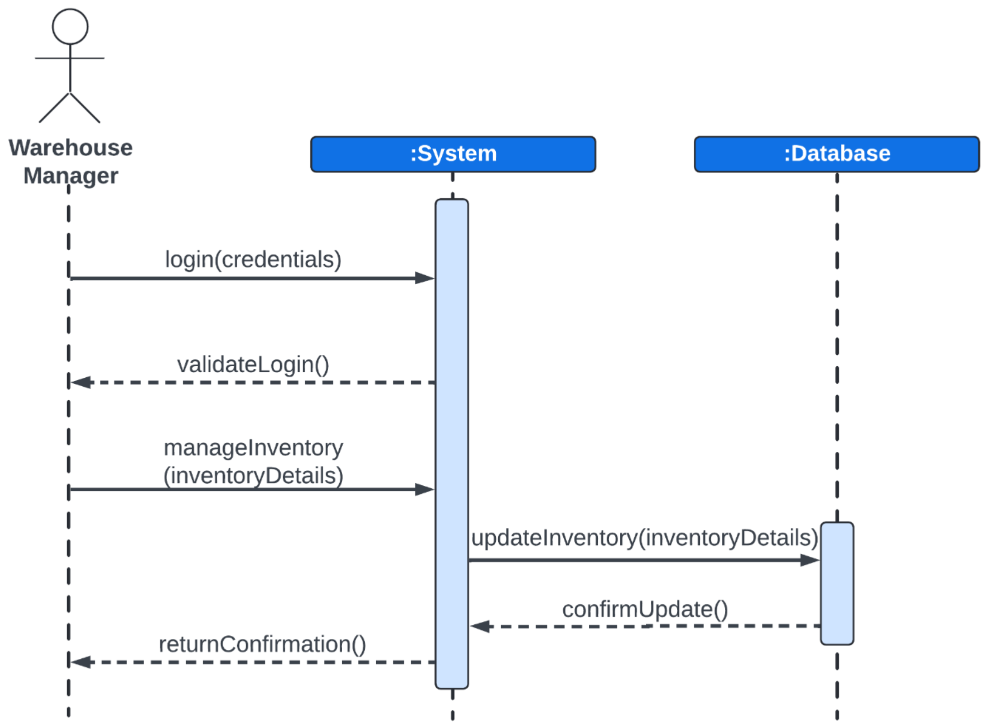
Create Product Record SD



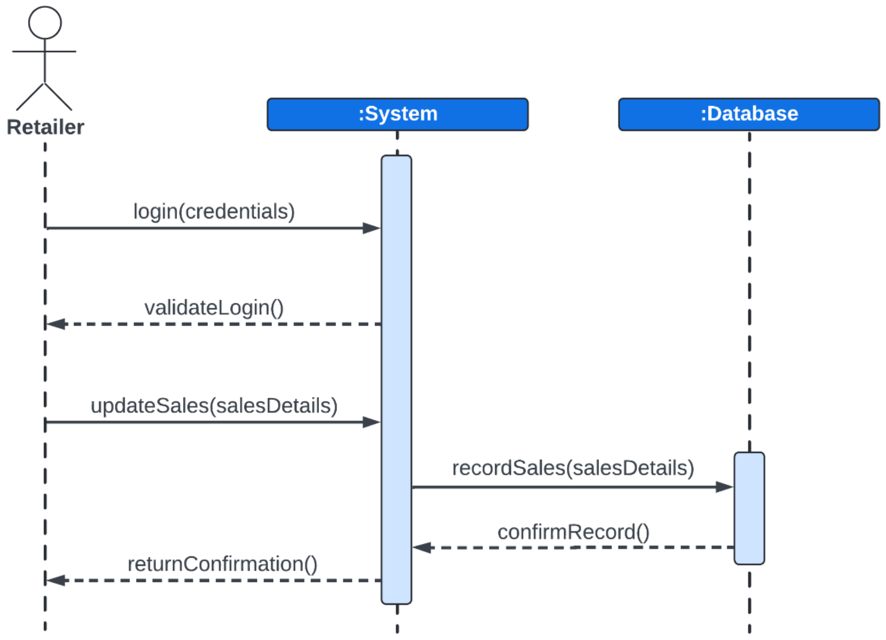
Update Product Location SD



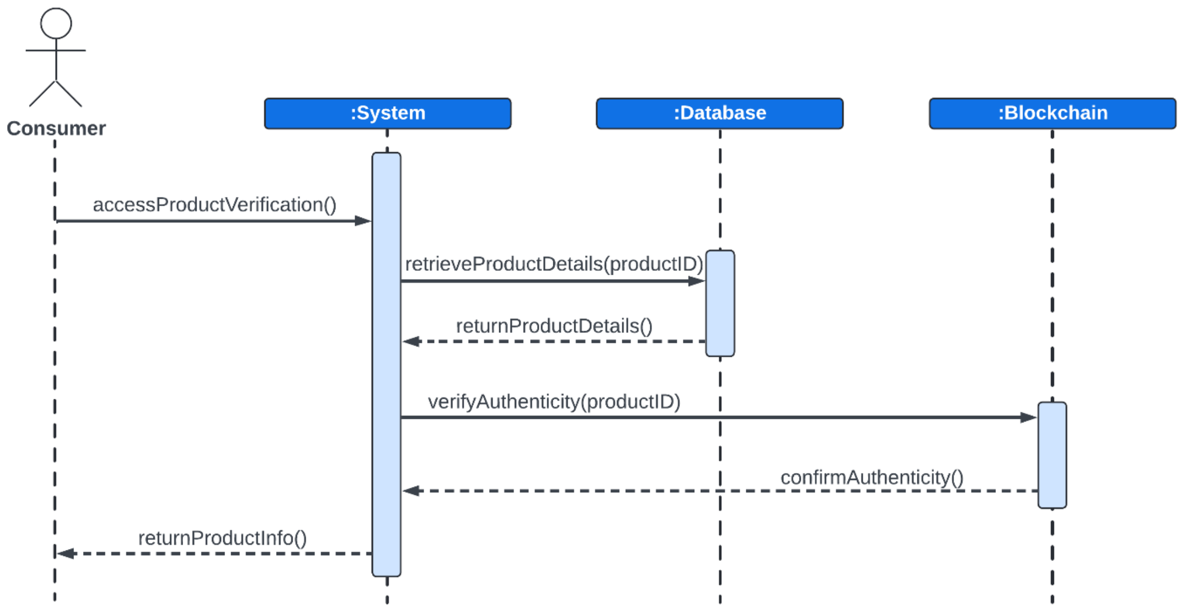
Manage Inventory SD



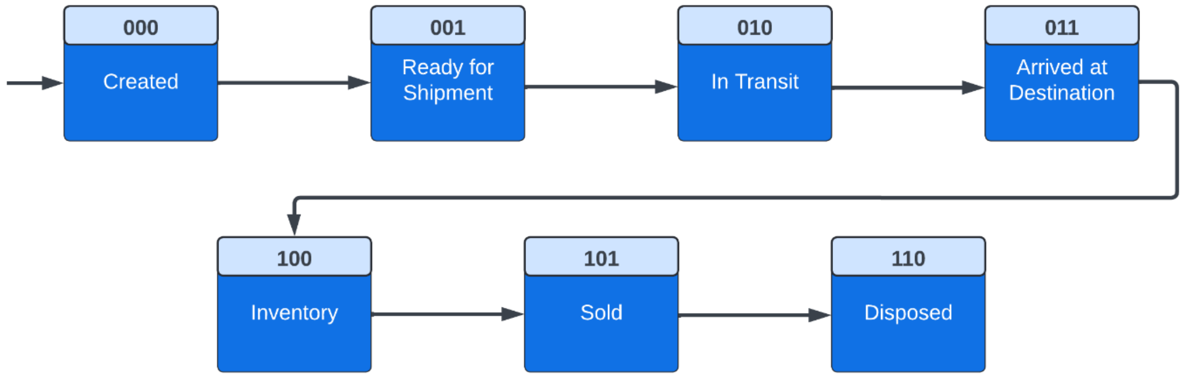
Update Sales SD



Verify Product SD

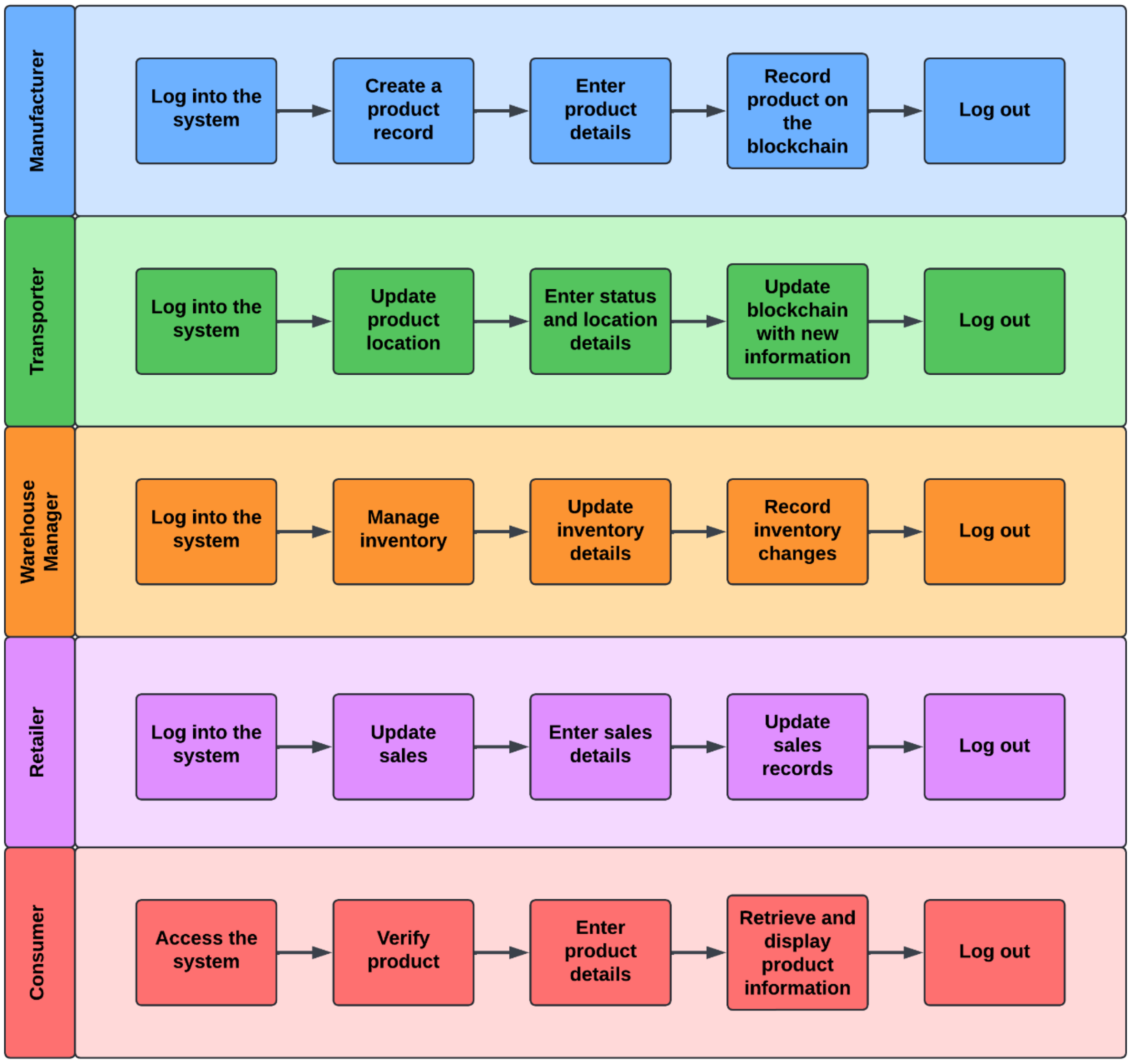


UML State Diagram

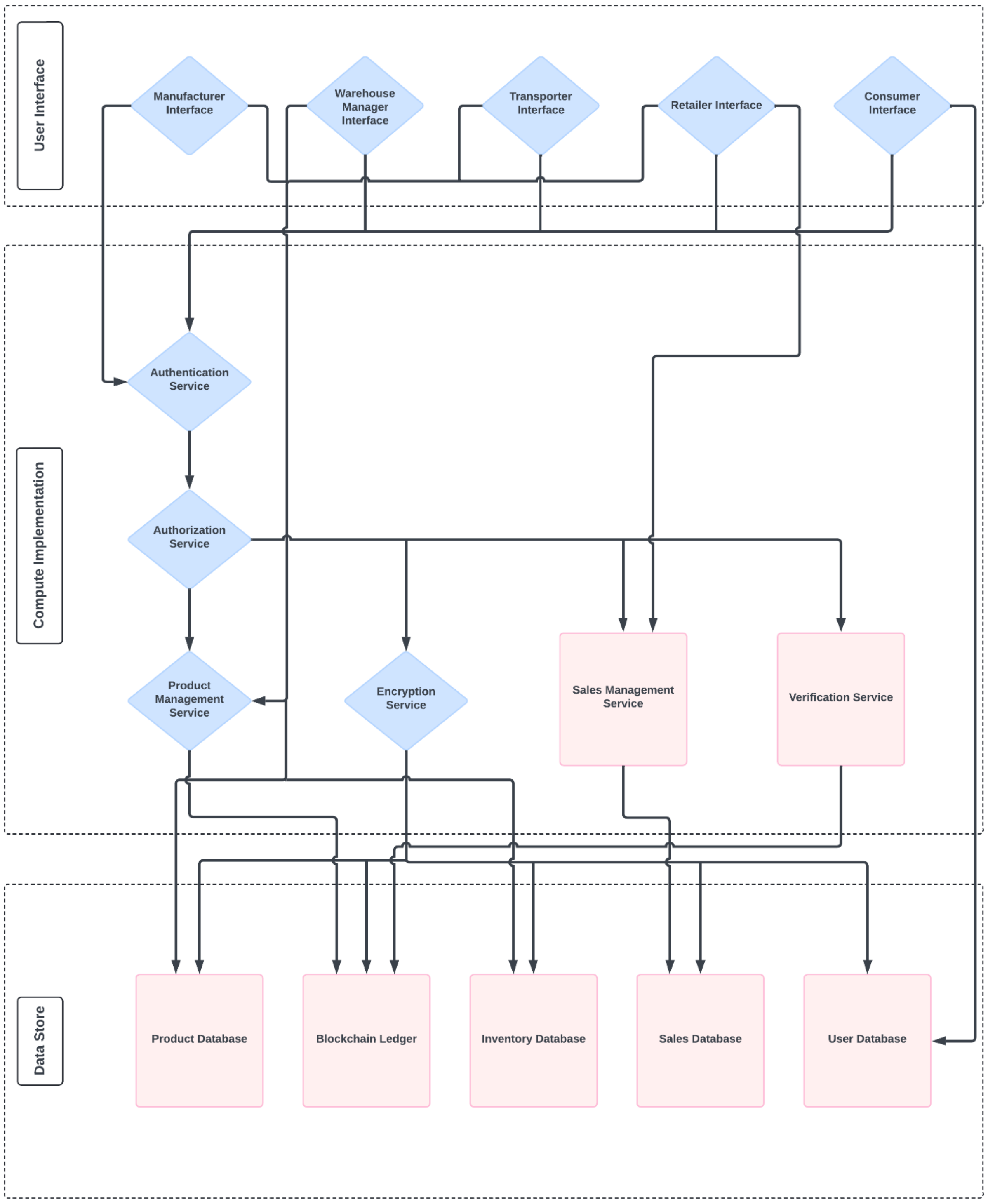


State	State Name	Description	Transition
000	Created	The product is created and a record is entered into the system.	Move to the next state once the product is ready for shipment.
001	Ready for Shipment	The product is packaged and ready to be shipped.	When a shipment order is confirmed, transition to the next state.
010	In Transit	The product is being transported to the next location (warehouse, retailer, etc.).	Move to the next state once the product arrives at its destination.
011	Arrived at Destination	The product has reached a warehouse, store, or final destination.	Depending on the destination, the product can transition to either "Inventory" or "Sold".
100	Inventory	The product is stored in inventory at a warehouse or retail location.	When the product is sold or shipped to another location, transition to "Sold" or back to "In Transit".
101	Sold	The product has been sold to a customer.	Final state, representing the end of the product's lifecycle in the supply chain.
110	Disposed	The product has reached the end of its useful life and is disposed of or recycled.	**NOT APPLICABLE (FINAL STATE)**

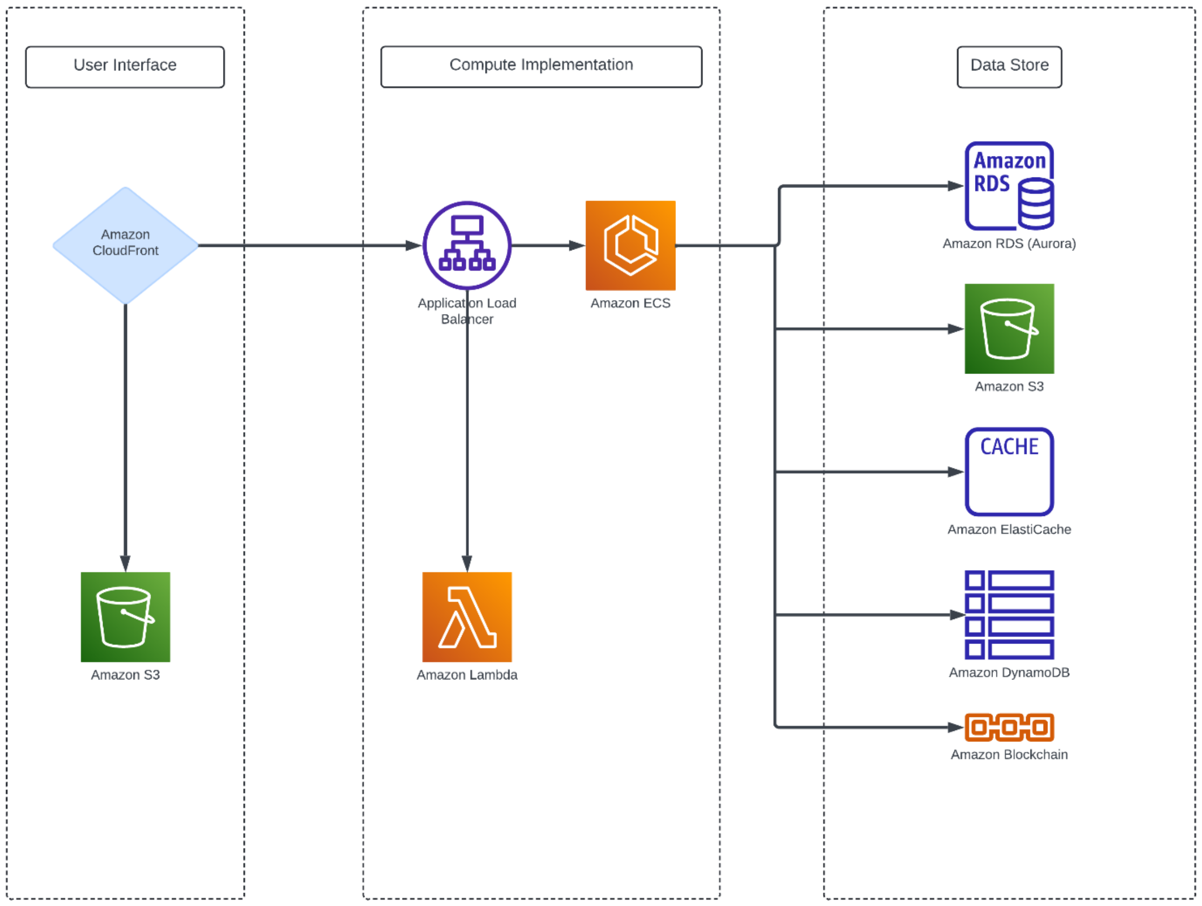
UML Activity Diagram (Swimlane Diagram)



UML Component Diagram



Cloud Deployment Diagram



Skeleton Classes and Tables Definition

Class: ProductRecord

- **Attributes:**
  - productID (Integer): Unique identifier for the product.
  - productName (String): Name of the product.
  - productCreationDate (Date): Date the product was created.
  - productBatchNumber (String): Batch number for tracking the product.
- **Methods:**
  - addLocationUpdate(DateTime updateDateTime, String updateStatus, String updateCoordinates): Adds location and status updates for the product.
  - linkInventoryRecord(Integer inventoryQuantity, String inventoryStorageLocation): Links the product to an inventory record.
  - associateSalesRecord(Date saleDate, Integer quantitySold, Float salePrice): Associates a sales record with the product.

Class: Consumer

- **Attributes:**
  - consumerID (Integer): Unique identifier for the consumer.
  - consumerName (String): Name of the consumer.
  - consumerContactInfo (String): Contact information for the consumer.
- **Methods:**
  - verifyProduct(ProductRecord product): Verifies the details of a product.

Class: Retailer

- **Attributes:**
  - retailerID (Integer): Unique identifier for the retailer.
  - retailerName (String): Name of the retailer.
  - storeLocation (String): Location of the retailer's store.
  - retailerContactInfo (String): Contact information for the retailer.
- **Methods:**
  - updateSalesRecord(ProductRecord product, Date saleDate, Integer quantitySold, Float salePrice): Updates the sales record for a product.

Class: Manufacturer

- **Attributes:**
  - manufacturerID (Integer): Unique identifier for the manufacturer.
  - manufacturerName (String): Name of the manufacturer.
  - manufacturerLocation (String): Location of the manufacturer.
  - manufacturerContactInfo (String): Contact information for the manufacturer.

- **Methods:**
    - createProductRecord(String productName, String productBatchNumber): Creates a new product record.

Class: Transporter

- **Attributes:**
    - transporterID (Integer): Unique identifier for the transporter.
    - transporterName (String): Name of the transporter.
    - transporterVehicleDetails (String): Details about the transporter's vehicle.
    - transporterContactInfo (String): Contact information for the transporter.
- **Methods:**
    - updateProductLocation(ProductRecord product, String updateStatus, String updateCoordinates): Updates the location and status of a product during transport.

Class: InventoryRecord

- **Attributes:**
    - inventoryRecordID (Integer): Unique identifier for the inventory record.
    - inventoryProductDetails (String): Details of the product in inventory.
    - inventoryQuantity (Integer): Quantity of the product in the inventory.
    - inventoryStorageLocation (String): Location where the inventory is stored.
    - inventoryLastUpdated (DateTime): Date and time when the inventory was last updated.
- **Methods:**
    - updateInventoryDetails(Integer quantity, String storageLocation): Updates the details of the inventory.
    - trackInventory(): Tracks the status of the inventory.

Class: SalesRecord

- **Attributes:**
    - salesRecordID (Integer): Unique identifier for the sales record.
    - saleDate (Date): Date of the sale.
    - salesProductDetails (String): Details of the product sold.
    - quantitySold (Integer): Quantity of the product sold.
    - salePrice (Float): Price at which the product was sold.
- **Methods:**
    - validateSalesData(): Validates the data in the sales record.

Class: LocationUpdate

- **Attributes:**
    - locationUpdateID (Integer): Unique identifier for the location update.

    - updateDateTime (DateTime): Date and time of the location update.
    - updateStatus (String): Status update associated with the location.
    - updateCoordinates (String): Coordinates of the location update.
- **Methods:**
    - validateUpdate(): Validates the location update data.

Class: WarehouseManager

- **Attributes:**
    - warehouseManagerID (Integer): Unique identifier for the warehouse manager.
    - managerName (String): Name of the warehouse manager.
    - warehouseLocation (String): Location of the warehouse.
    - managerContactInfo (String): Contact information for the warehouse manager.
- **Methods:**
    - manageInventory(InventoryRecord record): Manages the inventory record.

Table: ProductRecords

- **Columns:**
    - productID (INTEGER, Primary Key): Unique identifier for the product.
    - productName (VARCHAR): Name of the product.
    - productCreationDate (DATE): Date the product was created.
    - productBatchNumber (VARCHAR): Batch number for the product.

Table: Consumers

- **Columns:**
    - consumerID (INTEGER, Primary Key): Unique identifier for the consumer.
    - consumerName (VARCHAR): Name of the consumer.
    - consumerContactInfo (VARCHAR): Contact information for the consumer.

Table: Retailers

- **Columns:**
    - retailerID (INTEGER, Primary Key): Unique identifier for the retailer.
    - retailerName (VARCHAR): Name of the retailer.
    - storeLocation (VARCHAR): Store location.
    - retailerContactInfo (VARCHAR): Contact information for the retailer.

Table: Manufacturers

- **Columns:**
    - manufacturerID (INTEGER, Primary Key): Unique identifier for the manufacturer.
    - manufacturerName (VARCHAR): Name of the manufacturer.
    - manufacturerLocation (VARCHAR): Location of the manufacturer.
    - manufacturerContactInfo (VARCHAR): Contact information for the manufacturer.

Table: Transporters

- **Columns:**
    - transporterID (INTEGER, Primary Key): Unique identifier for the transporter.
    - transporterName (VARCHAR): Name of the transporter.
    - transporterVehicleDetails (VARCHAR): Details of the transporter's vehicle.
    - transporterContactInfo (VARCHAR): Contact information for the transporter.

Table: InventoryRecords

- **Columns:**
    - inventoryRecordID (INTEGER, Primary Key): Unique identifier for the inventory record.
    - inventoryProductDetails (VARCHAR): Product details.
    - inventoryQuantity (INTEGER): Quantity of the product in inventory.
    - inventoryStorageLocation (VARCHAR): Storage location of the inventory.
    - inventoryLastUpdated (DATETIME): Date and time of the last update.

Table: SalesRecords

- **Columns:**
    - salesRecordID (INTEGER, Primary Key): Unique identifier for the sales record.
    - saleDate (DATE): Date of the sale.
    - salesProductDetails (VARCHAR): Product details.
    - quantitySold (INTEGER): Quantity sold.
    - salePrice (FLOAT): Sale price.

Table: LocationUpdates

- **Columns:**
    - locationUpdateID (INTEGER, Primary Key): Unique identifier for the location update.
    - updateDateTime (DATETIME): Date and time of the update.
    - updateStatus (VARCHAR): Status associated with the update.
    - updateCoordinates (VARCHAR): Coordinates of the location.

Table: WarehouseManagers

- **Columns:**
    - warehouseManagerID (INTEGER, Primary Key): Unique identifier for the warehouse manager.
    - managerName (VARCHAR): Name of the manager.
    - warehouseLocation (VARCHAR): Warehouse location.
    - managerContactInfo (VARCHAR): Contact information for the manager.

Design Patterns

Design Pattern	Usage	Justification
Controller (GRASP)	Centralizes control in managing complex operations related to product records, inventory management, and location updates	By using a SupplyChainController, which coordinates actions like creating product records, updating locations, and verifying products, we maintain a clear separation of concerns, making the system easier to manage and extend.
Single Responsibility Principle (SRP) (SOLID)	Ensures that each class, such as ProductRecord, InventoryRecord, and SalesRecord, has a specific responsibility.	This principle is applied across the system to ensure that each class handles only one part of the business logic, like tracking product locations or managing inventory, which makes maintenance and future enhancements more straightforward.
Factory Method (GoF)	A factory method could be used to create different types of objects needed in the system, such as different types of database connections (e.g., for blockchain interaction or regular SQL databases).	This pattern supports the flexibility required to adapt to various backend services, enhancing the system's modularity and making it easier to switch or extend database services as needed.
Observer (GoF)	Could be used for monitoring changes across the supply chain, such as when a product's location is updated or when inventory levels change.	By implementing observers for key events like location updates, stakeholders (e.g., Warehouse Managers, Retailers) can be notified in real-time, improving responsiveness and data accuracy.
Repository (DDD)	Centralizes data access logic for entities such as ProductRecord, InventoryRecord, and SalesRecord.	This pattern encapsulates data access and simplifies testing by decoupling the system's business logic from the underlying data sources, making the system more resilient to changes in data storage technology.