Blockchain Shipment Management Tracking System

Table of Contents

0. Pre-Phase I: Software Project Management Plan (SPMP)

0.1 Gantt Chart

1. Phase I: Requirements Analysis

1.1 System Domain & Objectives
1.2 System Major Actors/Components
1.3 Glossary
1.4 Table of Functional Requirements (FR)
1.5 Table of Non-Functional Requirements (NFR)
1.6 Table of Assumptions (A)
1.7 Table of Responsibilities (R)
1.8 Table of Use Cases (UC)
1.9 Use Case Diagrams (UCD)
1.10 Use Case Scenarios (STD)
1.11 Activity Diagrams (AD)

2. Phase II: System Design

2.1 Apply Fixes on Phase I
2.2 Class Diagram (Initial)
2.3 Discuss and Apply selected Design Pattern(s)
2.4 Class Diagram (Updated)
2.5 Sequence Diagrams (SD)

3. Phase III & IV: System Implementation & Testing

3.1 Overview of Implementation
3.2 Code Structure Summary
3.3 Implementation of System Objects
3.4 Implementation of System UIs
3.5 Test Plan & Test Cases with Sample Results

0. Pre-Phase I: Software-Project Management Plan (SPMP)

0.1 Gantt Chart

The Gantt chart below provides a clear visual timeline of all project tasks, showing how responsibilities were distributed across team members, ensuring balanced workload distribution and helping us monitor progress against deadlines.

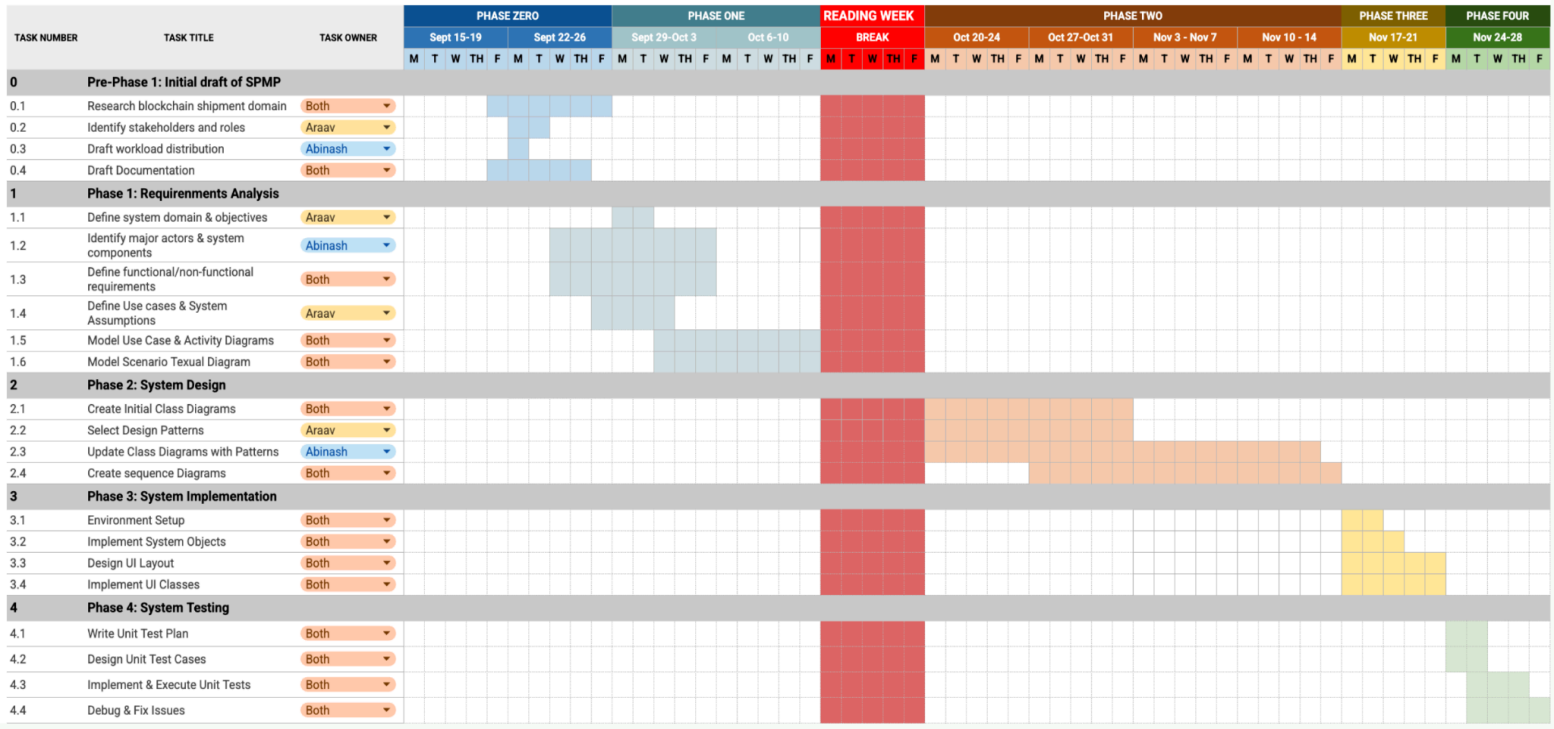
GANTT CHART

PROJECT TITLE Blockchain Shipment Management Tracking

TEAM 12 Araav Sehgal & Abinash Jegatheeswaran

DATE 11/14/25

TASK NUMBER	TASK TITLE	TASK OWNER
0	**Pre-Phase 1: Initial draft of SPMP**	
0.1	Research blockchain shipment domain	Both
0.2	Identify stakeholders and roles	Araav
0.3	Draft workload distribution	Abinash
0.4	Draft Documentation	Both
1	**Phase 1: Requirenments Analysis**	
1.1	Define system domain & objectives	Araav
1.2	Identify major actors & system components	Abinash
1.3	Define functional/non-functional requirements	Both
1.4	Define Use cases & System Assumptions	Araav
1.5	Model Use Case & Activity Diagrams	Both
1.6	Model Scenario Texual Diagram	Both
2	**Phase 2: System Design**	
2.1	Create Initial Class Diagrams	Both
2.2	Select Design Patterns	Araav
2.3	Update Class Diagrams with Patterns	Abinash
2.4	Create sequence Diagrams	Both
3	**Phase 3: System Implementation**	
3.1	Environment Setup	Both
3.2	Implement System Objects	Both
3.3	Design UI Layout	Both
3.4	Implement UI Classes	Both
4	**Phase 4: System Testing**	
4.1	Write Unit Test Plan	Both
4.2	Design Unit Test Cases	Both
4.3	Implement & Execute Unit Tests	Both
4.4	Debug & Fix Issues	Both



1. Phase I: Requirements Analysis

1.1 System Domain and Objectives

System Domain:

The **Blockchain Shipment Management Tracking System** falls under the supply chain and logistics domain and focuses on shipment management and product tracking. Traditional tracking systems experience several challenges, ultimately leading to its inefficient usage. These include:

- **Lack of Transparency:** Stakeholders not having full visibility/accessibility to shipment data
- **Data Manipulation/Fraud**: Inaccurate data records (eg. Inaccurate status of shipment) with no ability to be verified
- **Limited Traceability:** Data being scattered across systems causing issues in error traceability (when and where errors occurred) with scenarios such as delays, recalls, or disputes, etc.
- **Security:** with traditional systems being vulnerable, they are prone to attacks (ie; hackers) causing data compromisation and/or lost data.

Structurally, the Blockchain Shipment Management Tracking System will act as an efficient platform, allowing interaction between the stakeholders (ie: Shippers, carriers, buyers, warehouses, etc). It will consist of a **permissioned blockchain**, allowing only users with granted access to use the platform, **identity management**, **off-chain storage** (for handling all sizes of documents), **smart contracts** (ensuring no tampering/changing of existing data, along with an automated agreement generation process) and etc. Behaviorally, use cases of the system include validating transitions through the use of smart contracts, recording/updating shipment events (ie delivery, pickup, in-transit), and presenting users with a shared, interactable and real-time view of the shipment data, ensuring transparency. At a higher level, this system will automate the actions of shipment lifecycles to maintain traceability.

System Objectives:

As mentioned above, the Blockchain Shipment Management Tracking System will act as a platform to address the errors and challenges that are presented through traditional tracking systems. Implementing this Shipment Management Tracking System with Blockchain, acts as an enhanced, stable and secure system, that is the answer to all the challenges listed above. It addresses trust, transparency, efficiency, traceability, and security. Through including tamper-proof records, immutable event logs, and end-to-end traceability, the platform will enable trust among stakeholders, a reduced cost of minimizing disputes, reduced fraud, and enhanced security. Ultimately, the Blockchain Shipment Management Tracking System enhances the user experience, and seeks to deliver a more secure, and cost-effective solution for shipment management & product tracking.

1.2 System Major Components/Actors

The following is a sample list of the major components and actors in the system:

Major Actors & Components (A → Actor | C → Component)	Major Responsibilities and Roles
Manufacturer (A)	Records product details, creates shipment entries, and adds them to the blockchain.
Supplier (A)	Registers shipment details and ensures handover is captured in the blockchain.
Logistics Provider (A)	Updates transport milestones such as pickup, in-transit, and delivery events.
Warehouse (C)	Records storage, receipt, and dispatch activities in the system.
Customer (A)	Tracks shipment status, verifies product authenticity, and confirms final delivery
Administrator (A)	Maintains system security, configures access permissions, and resolves issues.
User Interfaces (Web-based) (C)	Provides dashboards for manufacturers, suppliers, logistics providers, and customers to input and view shipment data.
Customs/Regulatory Authority (A)	Validates shipments, conducts inspections, and logs approvals or holds.
Retailer (A)	Confirms receipt of shipments and verifies authenticity on the blockchain.
Blockchain Network (C)	Stores all shipment records in a tamper-resistant manner, accessible to authorized actors.
Payment Service (C)	Processes payments once deliveries or contractual conditions are fulfilled.
Smart Contract (C)	Automates rules such as delivery confirmation, compliance checks, or payment release.
Auditor (A)	Monitors shipment records for compliance and transparency, with read-only access to ensure accountability.

1.3 Glossary

The following is a list of the identifiers for all diagrams, tables, and items in this document. The # symbol represents a unique number for each member of the category.

Category	Identifier
Use Case	UC#
Assumption	A#
Functional Requirement	FR#
Non-Functional Requirement	NFR#
Scenario Textual Description	STD#
Sequence Diagram	SD#
Communication Diagram	CD#
Activity Diagram	AD#
Test Case	TC#

The following is a list of the definitions of the technical terms in the document:

Term	Definition
Manufacturer	Produces goods and records shipment details in the system.
Supplier	Prepares shipments and logs handover to logistics providers or warehouses.
Logistics Provider	Moves shipments and updates their status during transit.
Warehouse	Stores shipments, logs receipt and dispatch events.
Customer	Tracks shipments, confirms delivery, and checks product authenticity.
System Administrator	Configures user access, maintains security, and ensures the system works smoothly.
User Interfaces (Web-based)	Web screens where stakeholders enter and view shipment data.
Customs/Regulatory Authority	Inspects and validates shipments to ensure compliance.
Retailer	Receives shipments, verifies authenticity, and records

	receipt.
Blockchain	A secure shared record book where shipment records are stored and cannot be changed once saved.
Payment Service	Handles payments between parties when shipment conditions are met.
Smart Contract	Rules stored on the blockchain automatically perform actions, such as confirming delivery or releasing payment.
Auditor	Monitors shipment records for correctness and transparency (read-only).
Traceability	Ability to see the full history of a shipment from origin to delivery.
Provenance	The full history of a shipment, including where it has been and who handled it.
Ledger Entry	A single shipment record stored on the blockchain.
Immutable Record	A blockchain record that cannot be changed or deleted.

1.4 Table of Functional Requirements (FR)

The following is a sample list of the Functional Requirements in the system:

(A) → Actor
(C) → Component

ID	Description: FRs for the Manufacturer (Shipper) (A)	Traceability
FR1	The system must allow Manufacturers to record product details and create shipment entries via the User Interface (C), with all records stored immutably on the Blockchain Network (C).	UC01, R01, R11, NFR1
FR2	The system must enable Manufacturers to update shipment origin and dispatch details through the User Interface (C), validated by Smart Contracts (C), before being stored on the Blockchain Network (C).	UC01, UC03, R04, R12
FR3	The system must allow Manufacturers to assign Suppliers or Logistics Providers for shipments through the User Interface (C), with assignments enforced via Smart Contracts (C).	UC01, UC03, R04, R13
FR4	The system must allow manufacturers to review shipment progress and status via the User Interface (C), retrieving data from the Blockchain Network (C).	UC04, R05, R15, NFR2
ID	**Description: FRs for the Supplier (Shipper) (A)**	**Traceability**
FR5	The system must allow Suppliers to register shipment details and link them to Blockchain entries through the User Interface (C).	UC01, R02, R11
FR6	The system must allow Suppliers to verify handover events to Logistics Providers or Warehouses via the User Interface (C), with confirmations stored on the Blockchain Network (C).	UC03, R22, R25
FR7	The system must ensure that Suppliers validate shipment documentation before dispatch, with document hashes stored on the Blockchain Network (C).	UC02, R02, R12
FR8	The system must provide Suppliers with access to shipment updates through the User Interface (C), retrieving data from the Blockchain Network (C).	UC04, R05, R15
FR9	The system must send alerts to Suppliers regarding shipment events (e.g., pickup confirmed, handover completed) via the User Interface (C), with data retrieved	UC03, UC04, R12, R15, NFR9

	from the Blockchain Network (C).	
ID	**Description: FRs for the Logistics Provider (A)**	**Traceability**
FR10	The system must allow Logistics Providers to record pickup, in-transit, and delivery events via the User Interface (C), storing results on the Blockchain Network (C).	UC03, R22, R12
FR11	The system must allow Logistics Providers to update delays or route deviations in real time through the User Interface (C), with data recorded on the Blockchain Network (C).	UC03, R22, R15
FR12	The system must ensure that delivery confirmations are captured and stored immutably on the Blockchain Network (C), enforced by Smart Contracts (C).	UC11, R23, R25, R12
FR13	The system must enforce role-based permissions so that Logistics Providers can only update records for shipments assigned to them, with enforcement handled by Smart Contracts (C) and interactions through the User Interface (C).	UC08, UC15, R14, R29, NFR5
ID	**Description: FRs for the Warehouse (C)**	**Traceability**
FR14	The system must allow Warehouses to record the receipt of shipments from Suppliers or Logistics Providers through the User Interface (C), with data stored on the Blockchain Network (C).	UC03, R33, R12
FR15	The system must allow Warehouses to log storage details and conditions (e.g., timestamps, duration) through the User Interface (C), with records secured in the Blockchain Network (C).	UC03, R33, R15
FR16	The system must allow Warehouses to confirm dispatch events to the next stakeholder via the User Interface (C), storing confirmation on the Blockchain Network (C).	UC03, R34, R12
FR17	The system must provide Warehouses with real-time visibility into storage status via the Blockchain Network (C) through the User Interface (C).	UC04, R35, R15, NFR2
ID	**Description: FRs for the Customer (Buyer) (A)**	**Traceability**
FR18	The system must allow Customers to track shipment progress and current location via the User Interface (C), retrieving data from the Blockchain Network (C).	UC04, R06, R15
FR19	The system must allow Customers to verify product authenticity using document and record hashes stored on the Blockchain Network (C).	UC09, R09, R15, NFR6

FR20	The system must allow Customers to confirm receipt of goods via the User Interface (C), with confirmations immutably stored on the Blockchain Network (C).	UC11, R07, R12
FR21	The system must allow Customers to report discrepancies (e.g., delays or damaged items) via the User Interface (C), with records logged in the Blockchain Network (C).	UC05, R10, R15
FR22	The system must display shipment delay, delivery confirmation, or discrepancy messages to Customers via the User Interface (C), with data retrieved from the Blockchain Network (C).	UC04, UC11, R12, R15, NFR9
FR23	The system must allow Customers to initiate dispute resolution processes through the User Interface (C), with resolution workflows enforced by Smart Contracts (C).	UC05, R10, R16
ID	**Description: FRs for the Administrator (A)**	**Traceability**
FR24	The system must allow the Administrator to manage user roles and permissions via the User Interface (C), with enforcement by Smart Contracts (C).	UC08, UC15, R29, NFR5
FR25	The system must provide Administrators with tools to monitor system performance and review logs through the User Interface (C).	UC08, R31, NFR4
FR26	The system must allow Administrators to resolve technical issues and ensure system uptime through the User Interface (C).	UC08, R32
FR27	The system must allow the Administrator to configure blockchain nodes and system policies via the User Interface (C), with changes applied directly to the Blockchain Network (C).	UC08, R30, R31
FR28	The system must allow Administrators to review and manage detailed system audit logs, stored immutably on the Blockchain Network (C).	UC12, R31, R32, R15
FR29	The system must display real-time error or downtime alerts to Administrators through the User Interface (C), based on monitoring data from the Blockchain Network (C).	UC08, UC12, R31, NFR4
ID	**Description: FRs for the Customs/Regulatory Authority (A)**	**Traceability**
FR30	The system must allow Regulatory Authorities to validate shipment details during inspections via the User Interface (C), with document hashes verified against the Blockchain Network (C).	UC10, R17, R15

FR31	The system must allow Regulatory Authorities to record approvals, rejections, or holds in the system via the User Interface (C), with entries stored on the Blockchain Network (C).	UC10, R18, R19, R12
FR32	The system must ensure compliance records from Regulatory Authorities are stored immutably on the Blockchain Network (C).	UC10, UC07, R12, NFR8
FR33	The system must allow Regulatory Authorities to generate inspection reports for stakeholders via the User Interface (C).	UC07, UC10, R20, R28
ID	**Description: FRs for the Retailer (Buyer) (A)**	**Traceability**
FR34	The system must allow Retailers to confirm receipt of shipments from Suppliers or Logistics Providers via the User Interface (C), storing results on the Blockchain Network (C).	UC11, R07, R12
FR35	The system must allow Retailers to verify product authenticity through Blockchain event logs accessed via the User Interface (C).	UC09, R09, R15
FR36	The system must provide Retailers with access to shipment records for dispute resolution via the User Interface (C), with records retrieved from the Blockchain Network (C).	UC05, UC06, R10, R15
FR37	The system must allow Retailers to track delivery timelines through the User Interface (C), retrieving event data from the Blockchain Network (C).	UC04, R06, R15
FR38	The system must display delivery update and payment release alerts to Retailers via the User Interface (C), triggered by Smart Contracts (C).	UC11, UC14, R12, R15
ID	**Description: FRs for the Blockchain Network (C)**	**Traceability**
FR39	The Blockchain Network must maintain a tamper-proof ledger of all shipment records.	UC01–UC14, R12, NFR8
FR40	The Blockchain Network must validate all transactions before recording them.	UC01–UC03, UC10, R11
FR41	The Blockchain Network must allow role-based read access to authorized stakeholders only.	UC08, UC15, R14, NFR5
FR42	The Blockchain Network must ensure consensus among nodes for all shipment events.	UC01–UC14, R11, NFR9
ID	**Description: FRs for the Payment Service (C)**	**Traceability**

FR44	The system must process payments after delivery confirmation through the Payment Service (C), triggered by Smart Contracts (C).	UC11, UC14, R16
FR45	The system must ensure that all transactions between stakeholders are secured by the Payment Service (C).	UC11, UC14, NFR6, R12
FR46	The system must integrate the Payment Service (C) with Smart Contracts (C) to trigger automatic payments.	UC14, R13
FR47	The system must generate receipts for completed transactions via the Payment Service (C).	UC11, UC14, R12
ID	**Description: FRs for the Smart Contract (C)**	**Traceability**
FR48	The system must validate shipment creation and updates against predefined rules enforced by Smart Contracts (C).	UC01, UC03, R13
FR49	The system must enforce delivery confirmation before releasing payments through Smart Contracts (C).	UC11, UC14, R13, NFR6
FR50	The system must automate compliance checks for Regulatory Authorities using Smart Contracts (C).	UC07, UC10, R13
FR51	The system must prevent modification of completed shipment records by enforcing immutability through Smart Contracts (C).	UC01–UC14, R12, NFR8
ID	**Description: FRs for the Auditor (A)**	**Traceability**
FR52	The system must allow Auditors to access shipment histories through the User Interface (C), with read-only rights from the Blockchain Network (C).	UC06, UC12, R26, NFR5
FR53	The system must allow Auditors to verify the authenticity and traceability of shipment records through the Blockchain Network (C).	UC09, UC12, R27, R15, NFR6
FR54	The system must allow Auditors to generate compliance and transparency reports via the User Interface (C).	UC07, R28
FR55	The system must allow Auditors to ensure accountability across stakeholders by analyzing Blockchain data via the User Interface (C).	UC07, UC12, R28
ID	**Description: FRs for the User Interfaces (Web-based) (C)**	**Traceability**
FR56	The system must provide dashboards tailored to each stakeholder role (Manufacturer, Supplier, etc.) through the User Interface (C).	UC08, UC15, R29, NFR5

FR57	The system must allow users to enter shipment creation, updates, and confirmations through the User Interface (C), with data stored in the Blockchain Network (C).	UC01–UC03, UC11, R01–R07
FR58	The system must allow users to track shipments in real time via the User Interface (C), retrieving data from the Blockchain Network (C).	UC04, R06, R15, NFR2
FR59	The system must ensure secure login and identity management for all users via the User Interface(C).	UC15, R36, NFR5
FR60	The system must generate summary reports and shipment analytics for authorized stakeholders through the User Interface (C).	UC07, UC12, R28
FR61	The system must display alerts and system messages for exceptions, delays, or required actions through the User Interface (C), based on data from the Blockchain Network (C).	UC03, UC04, UC05, UC11, R15, R16, NFR9

Functional Requirements of the Manufacturer (Shipper):

1. The system must allow Manufacturers to record product details and create shipment entries via the User Interface (C), with all records stored immutably on the Blockchain Network (C).
2. The system must enable Manufacturers to update shipment origin and dispatch details through the User Interface (C), validated by Smart Contracts (C), before being stored on the Blockchain Network (C).
3. The system must allow Manufacturers to assign Suppliers or Logistics Providers for shipments through the User Interface (C), with assignments enforced via Smart Contracts (C).
4. The system must allow Manufacturers to review shipment progress and status via the User Interface (C), retrieving data from the Blockchain Network (C).

Functional Requirements of the Supplier (Shipper):

1. The system must allow Suppliers to register shipment details and link them to Blockchain entries through the User Interface (C).
2. The system must allow Suppliers to verify handover events to Logistics Providers or Warehouses via the User Interface (C), with confirmations stored on the Blockchain Network (C).
3. The system must ensure that Suppliers validate shipment documentation before dispatch, with document hashes stored on the Blockchain Network (C).
4. The system must provide Suppliers with access to shipment updates through the User Interface (C), retrieving data from the Blockchain Network (C).
5. The system must send alerts to Suppliers regarding shipment events (e.g., pickup confirmed, handover completed) via the User Interface (C), with data retrieved from the Blockchain Network (C).

Functional Requirements of the Logistics Provider:

1. The system must allow Logistics Providers to record pickup, in-transit, and delivery events via the User Interface (C), storing results on the Blockchain Network (C).
2. The system must allow Logistics Providers to update delays or route deviations in real time through the User Interface (C), with data recorded on the Blockchain Network (C).
3. The system must ensure that delivery confirmations are captured and stored immutably on the Blockchain Network (C), enforced by Smart Contracts (C).
4. The system must enforce role-based permissions so that Logistics Providers can only update records for shipments assigned to them, with enforcement handled by Smart Contracts (C) and interactions through the User Interface (C).

Functional Requirements of the Warehouse (C):

1. The system must allow Warehouses to record the receipt of shipments from Suppliers or Logistics Providers through the User Interface (C), with data stored on the Blockchain Network (C).
2. The system must allow Warehouses to log storage details and conditions (e.g., timestamps, duration) through the User Interface (C), with records secured in the Blockchain Network (C).
3. The system must allow Warehouses to confirm dispatch events to the next stakeholder via the User Interface (C), storing confirmation on the Blockchain Network (C).
4. The system must provide Warehouses with real-time visibility into storage status via the Blockchain Network (C) through the User Interface (C).

Functional Requirements of the Customer (Buyer):

1. The system must allow Customers to track shipment progress and current location via the User Interface (C), retrieving data from the Blockchain Network (C).
2. The system must allow Customers to verify product authenticity using document and record hashes stored on the Blockchain Network (C).
3. The system must allow Customers to confirm receipt of goods via the User Interface (C), with confirmations immutably stored on the Blockchain Network (C).
4. The system must allow Customers to report discrepancies (e.g., delays or damaged items) via the User Interface (C), with records logged in the Blockchain Network (C).
5. The system must display shipment delay, delivery confirmation, or discrepancy alerts to Customers via the User Interface (C), with data retrieved from the Blockchain Network (C).
6. The system must allow Customers to initiate dispute resolution processes through the User Interface (C), with resolution workflows enforced by Smart Contracts (C).

Functional Requirements of the Administrator:

1. The system must allow the Administrator to manage user roles and permissions via the User Interface (C), with enforcement by Smart Contracts (C).
2. The system must provide Administrators with tools to monitor system performance and review logs through the User Interface (C).
3. The system must allow Administrators to resolve technical issues and ensure system uptime through the User Interface (C).
4. The system must allow the Administrator to configure blockchain nodes and system policies via the User Interface (C), with changes applied directly to the Blockchain Network (C).

5. The system must allow Administrators to review and manage detailed system audit logs, stored immutably on the Blockchain Network (C).
6. The system must display real-time error or downtime alerts to Administrators through the User Interface (C), based on monitoring data from the Blockchain Network (C).

Functional Requirements of the Customs/Regulatory Authority:

1. The system must allow Regulatory Authorities to validate shipment details during inspections via the User Interface (C), with document hashes verified against the Blockchain Network (C).
2. The system must allow Regulatory Authorities to record approvals, rejections, or holds in the system via the User Interface (C), with entries stored on the Blockchain Network (C).
3. The system must ensure compliance records from Regulatory Authorities are stored immutably on the Blockchain Network (C).
4. The system must allow Regulatory Authorities to generate inspection reports for stakeholders via the User Interface (C).

Functional Requirements of the Retailer (Buyer):

1. The system must allow Retailers to confirm receipt of shipments from Suppliers or Logistics Providers via the User Interface (C), storing results on the Blockchain Network (C).
2. The system must allow Retailers to verify product authenticity through Blockchain event logs accessed via the User Interface (C).
3. The system must provide Retailers with access to shipment records for dispute resolution via the User Interface (C), with records retrieved from the Blockchain Network (C).
4. The system must allow Retailers to track delivery timelines through the User Interface (C), retrieving event data from the Blockchain Network (C).
5. The system must display delivery update and payment release alerts to Retailers via the User Interface (C), triggered by Smart Contracts (C).

Functional Requirements of the Blockchain Network (C):

1. The Blockchain Network must maintain a tamper-proof ledger of all shipment records.
2. The Blockchain Network must validate all transactions before recording them.
3. The Blockchain Network must allow role-based read access to authorized stakeholders only.
4. The Blockchain Network must ensure consensus among nodes for all shipment events.

Functional Requirements of the Payment Service (C):

1. The system must process payments after delivery confirmation through the Payment Service (C), triggered by Smart Contracts (C).
2. The system must ensure that all transactions between stakeholders are secured by the Payment Service (C).
3. The system must integrate the Payment Service (C) with Smart Contracts (C) to trigger automatic payments.
4. The system must generate receipts for completed transactions via the Payment Service (C).

Functional Requirements of the Smart Contract (C):

1. The system must validate shipment creation and updates against predefined rules enforced by Smart Contracts (C).
2. The system must enforce delivery confirmation before releasing payments through Smart Contracts (C).
3. The system must automate compliance checks for Regulatory Authorities using Smart Contracts (C).
4. The system must prevent modification of completed shipment records by enforcing immutability through Smart Contracts (C).

Functional Requirements of the Auditor:

1. The system must allow Auditors to access shipment histories through the User Interface (C), with read-only rights from the Blockchain Network (C).
2. The system must allow Auditors to verify the authenticity and traceability of shipment records through the Blockchain Network (C).
3. The system must allow Auditors to generate compliance and transparency reports via the User Interface (C).
4. The system must allow Auditors to ensure accountability across stakeholders by analyzing Blockchain data via the User Interface (C).

Functional Requirements of the User Interfaces (Web-based) (C):

1. The system must provide dashboards tailored to each stakeholder role (Manufacturer, Supplier, etc.) through the User Interface (C).
2. The system must allow users to enter shipment creation, updates, and confirmations through the User Interface (C), with data stored in the Blockchain Network (C).
3. The system must allow users to track shipments in real time via the User Interface (C), retrieving data from the Blockchain Network (C).
4. The system must ensure secure login and identity management for all users via the User Interface(C).
5. The system must generate summary reports and shipment analytics for authorized stakeholders through the User Interface (C).
6. The system must display alerts and system messages for exceptions, delays, or required actions through the User Interface (C), based on data from the Blockchain Network (C).

1.5 Table of Non-Functional Requirements (NFR)

The following is a sample list of the Non-Functional Requirements in the system:

ID	Description	Category	Traceability
NFR1	The blockchain should record a new shipment event in less than 5 seconds	Performance	UC01, UC03, UC10, UC11, R11, R12
NFR2	The UI of the system should allow	Performance	UC04, UC06, UC11,

	stakeholders to track shipment status updates in real-time		R05, R06, R15
NFR3	Documents stored in off-chain storage should have a quick retrieval process, despite document size (~ <4 seconds)	Performance	UC02, UC06, UC07, UC09, R02, R15
NFR4	The system should maintain maximum uptime availability across all nodes to ensure continuous access	Reliability/Ability	UC08, UC12, UC14, R31, R32
NFR5	Role-based authentication should be required to access the system	Security	UC08, UC15, R29, R36
NFR6	All transactions should be digitally signed	Security	UC02, UC09, UC11, UC12, R03, R07, R19
NFR7	The system must support a high number of concurrent users (~ 1000) accessing shipment data without performance issues	Scalability	UC04, UC08, UC15
NFR8	The blockchain ledger should be immutable, with all shipment records maintained	Security/Reliability	UC01–UC14, UC12, R12, R15
NFR9	Events status' must be consistent & takes a maximum of ~3 seconds to synchronize across all nodes	Performance	UC03, UC10, UC11, UC14, R11, R15

1.6 Table of Assumptions (A)

The following is a sample list of possible assumptions for our system.

ID	Description	Traceability
A01	All stakeholders (manufacturers, suppliers, logistics providers, etc.) have reliable computers and internet access to use the web-based system.	UC01–UC15
A02	The blockchain network nodes are always operational and synchronized.	UC01–UC14, R11, R15
A03	Smart contracts execute correctly and without errors when triggered.	UC03, UC10, UC13, UC14, R13
A04	All users are trained to use the system and understand their role-specific dashboards.	UC15

A05	The Payment Service provider is trusted and always available to process transactions.	UC11, UC14, R16
A06	System administrators have the authority to configure roles and permissions correctly.	UC08, UC15, R29
A07	Regulatory authorities are available to validate shipments promptly.	UC10, R17–R20
A08	Warehouses and logistics providers have the hardware (scanners, devices, etc.) needed to log updates.	UC03, R22, R33
A09	Once written to the blockchain, data cannot be altered or removed (immutability assumption).	UC01–UC14, UC12, R12
A10	Off-chain storage services are reliable and secure for large documents.	UC02, UC09, R02
A11	There will be no major power failures or system downtimes affecting critical blockchain nodes.	UC08, UC12, R31
A12	Auditors and regulators have read-only access and cannot tamper with shipment records.	UC06, UC07, UC12, R26

1.7 Table of Responsibilities (R)

The following is a sample list of the Responsibilities of our system:

ID	Description	Traceability
	Responsibilities of the Shipper (Manufacturer/Supplier)	
R01	The Shipper makes a new shipment record through the Blockchain on the User Interface	UC01, FR1
R02	Shipping documents are uploaded to the off-chain storage by the Shipper and store an immutable hash on the Blockchain	UC02, FR7
R03	Shipper digitally signs shipment events to ensure authenticity	UC02, UC11, FR12
R04	Expected delivery date & handover details are updated by the Shipper	UC03, FR2, FR3
R05	Shipment status updating/querying is performed by the Shipper	UC03, UC04, UC11, FR4
	Responsibilities of the Buyer (Retailer/Customer)	
R06	Buyer queries shipment status in real time through the UI	UC04, FR18

R07	Receipt is signed digitally on the Blockchain by the Buyer upon delivery	UC11, FR20, FR34
R08	Shipment history is reviewed to ensure accurate traceability by the Buyer	UC06, FR36
R09	Buyer checks hashes on the Blockchain to verify document authenticity	UC09, FR19, FR35
R10	Buyer raises any concerns/disputes if shipment conditions are not met	UC05, FR21, FR23
	Responsibilities of the Blockchain Network	
R11	Blockchain performing nodes monitor each transaction (event) and return shipment ID based on new events	UC01–UC14, FR39–FR42
R12	Blockchain stores shipment (off-chain if required) events immutably & is synchronized across all performing nodes	UC01–UC14, FR39, FR51
R13	Smart contracts enforce shipment lifecycle rules (e.g., the label cannot be shown as clearance until the goods are delivered)	UC03, UC10, UC14, FR48–FR50
R14	Performing Blockchain nodes ensure secure & role-based access to transactions (events)	UC08, UC15, FR13, FR41
R15	Blockchain updates/maintains a full history per shipment event	UC03, UC04, UC06, FR39
R16	Blockchain Network manages post-dispute workflows, including resolution, escalation, and insurance claim processing.	UC05, UC14, FR23, FR44
	Responsibilities of Customs/Regulators	
R17	Customs officer verifies compliance documents by comparing the file hash on the Blockchain	UC10, FR30
R18	Clearance event (e.g., Approved/Rejected/Pending) is updated by a customs officer on the Blockchain	UC10, FR31
R19	Clearance approval is digitally signed by customs to ensure authenticity	UC10, FR31
R20	Customs officer queries shipment history to ensure compliance and prevent fraud	UC06, UC10, FR33
R21	Customs officers raise any alerts/concerns on any incomplete/suspicious shipments	UC05, UC10, FR31
	Responsibilities of the Logistics Provider	

R22	Logistics Provider updates shipment pickup and delivery events on the Blockchain	UC03, UC11, FR10–FR13
R23	Logistics Provider uploads proof-of-delivery or condition documents (hashes stored on-chain)	UC03, UC11, FR10–FR13
R24	Logistics Provider queries shipment status for assignments they are responsible for	UC03, UC11, FR10–FR13
R25	Logistics Provider digitally signs handover confirmations	UC03, UC11, FR10–FR13
	Responsibilities of the Auditor	
R26	Auditor queries Blockchain history for compliance and accountability checks	UC06–UC12, FR52–FR55
R27	Auditor verifies the integrity of shipment records by checking digital signatures and hashes	UC06–UC12, FR52–FR55
R28	Auditor generates compliance or anomaly reports based on Blockchain data	UC06–UC12, FR52–FR55
	Responsibilities of the Administrator	
R29	The administrator manages user accounts and roles for actors in the system	UC08, UC15, FR24–FR29
R30	The administrator ensures smart contracts are deployed and updated as needed	UC08, UC15, FR24–FR29
R31	The administrator monitors system performance and node synchronization	UC08, UC15, FR24–FR29
R32	The administrator resolves technical issues and supports other actors	UC08, UC15, FR24–FR29
	Responsibilities of the Warehouse	
R33	Warehouse records receipt of shipments from Suppliers or Logistics Providers, including timestamps, duration, and storage conditions.	UC03, FR14–FR17
R34	Warehouse confirms the dispatch of shipments to the assigned Logistics Provider, Retailer, or Customer.	UC03, FR14–FR17
R35	The warehouse maintains visibility of current storage status and shipment locations.	UC03, FR14–FR17
	Additional Responsibitlies	
R36	The system authenticates users and validates their credentials	UC15, FR59

	before granting access.	

1.8 Table of Use Cases (UC)

The following is a sample list of the Use Cases (goals) of our system:

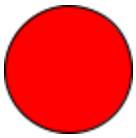
ID	Name	Description	Traceability
UC01	Create Shipment Record	Shipper (Manufacturer or Supplier) records a new shipment on the blockchain with product details.	FR1, R01, NFR1, NFR8
UC02	Upload Shipment Documents	Shipper (Manufacturer or Supplier) uploads documents to off-chain storage, storing the hash on the blockchain.	FR7, R02, NFR3, NFR6
UC03	Update Shipment Status	Shipper (Manufacturer or Supplier), Logistics Provider, or Warehouse updates shipment milestones (pickup, in-transit, delivery, dispatch).	FR10–FR13, R05, R12, NFR9
UC04	Query Shipment Status	Buyer (Customer or Retailer), Shipper (Manufacturer or Supplier), or Retailer queries shipment status in real time through the UI.	FR4, FR18, FR37, R06, R15, NFR2
UC05	Raise Dispute/Concerns	Buyer (Customer or Retailer) or Customs raises concerns/disputes if shipment conditions aren't met.	FR21, FR23, R10, R16, NFR8
UC06	Review Shipment History	Buyer (Customer or Retailer), Auditor, or Customs reviews shipment history for traceability.	FR36, FR52, R08, R15, R26
UC07	Generate Compliance Report	Auditors generate compliance and transparency reports from blockchain data.	FR54, FR55, R28, NFR3, NFR8
UC08	Manage User Access & Roles	Admin assigns roles to users and controls blockchain permissions	FR24–FR29, R29–R32, NFR5
UC09	Document Authenticity Verification	Integrity of Documents is validated by comparing recalculated hashes with the immutable ones stored on the blockchain	FR19, FR35, FR53, R09, R15, NFR6
UC10	Customs Clearance Approval	Customs officers record clearance or hold decisions directly on the blockchain for trusted verification	FR30–FR33, R17–R19, NFR1, NFR8

UC11	Confirmation of Delivery	Delivery is confirmed on the blockchain by the receiver, bringing the shipment lifecycle to an end	FR20, FR34, R07, R12, NFR6, NFR9
UC12	Generate Audit Trail	Immutable blockchain transaction history is compiled into regulator/auditor-approved audit trails.	FR28, FR55, R26–R28, NFR3, NFR8
UC13	System off-chain fraud detection	Verifying the off-chain stored documents with hashes on the blockchain to prevent fraud & data tampering	FR51, R27, R29, NFR3, NFR8
UC14	Insurance Claim Automation	If the receiver's goods are delayed/damaged, smart contracts trigger insurance claims automatically	FR44–FR47, R13, R16, NFR4, NFR8
UC15	User Authentication	System validates user credentials and assigns access rights based on roles (e.g. Shipper, Buyer, Custom Officer) before granting access to system functionalities.	FR59, R36, R14, NFR5

1.9 Use Case Diagrams

The following is the list of the Use Case Diagrams, rounding out the Blockchain Shipment Management Tracking:

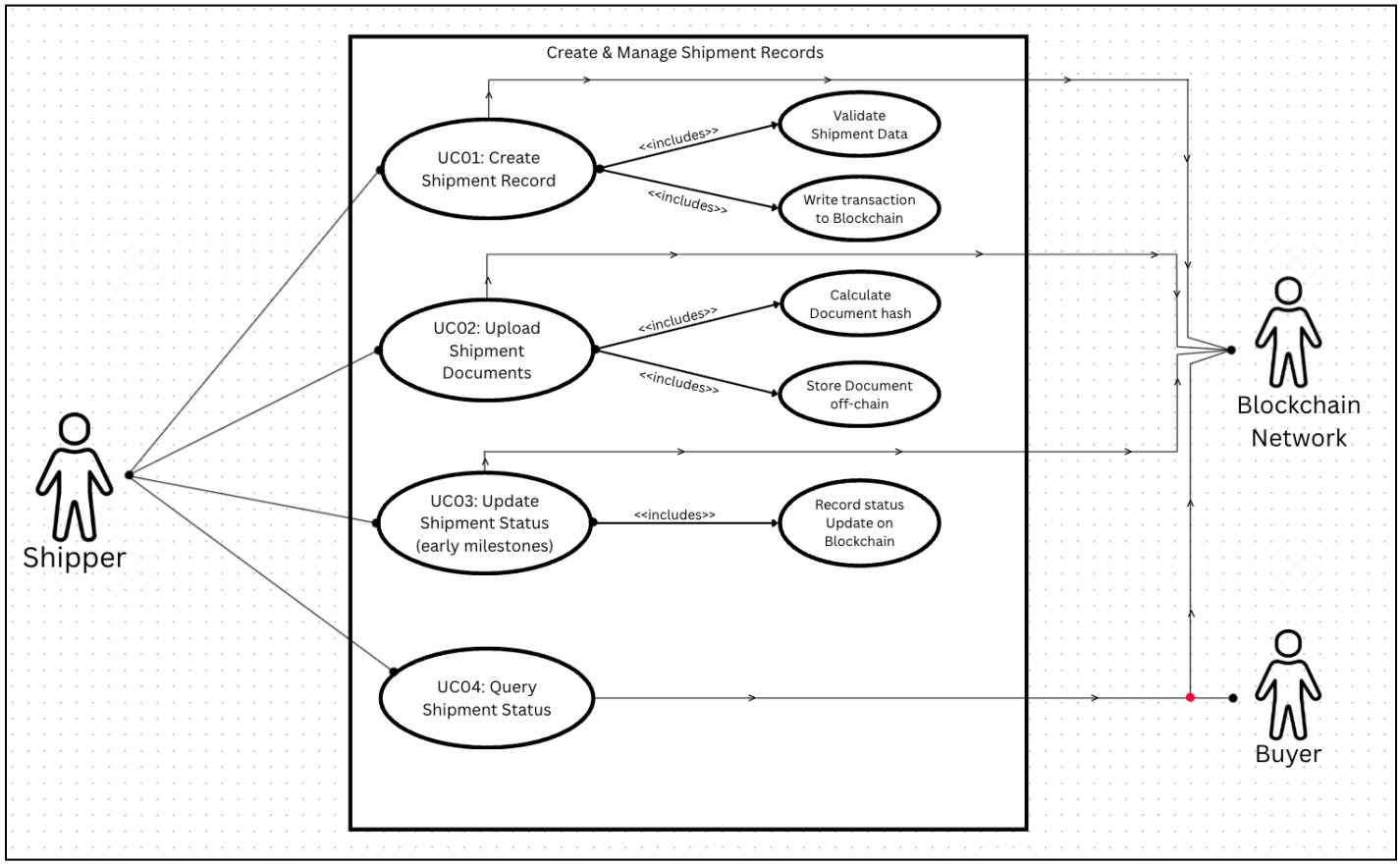
The following red circle occurs multiple times in the Use Case Diagrams and is used to denote an additional path added in the process of mapping out the use cases to their secondary actors.



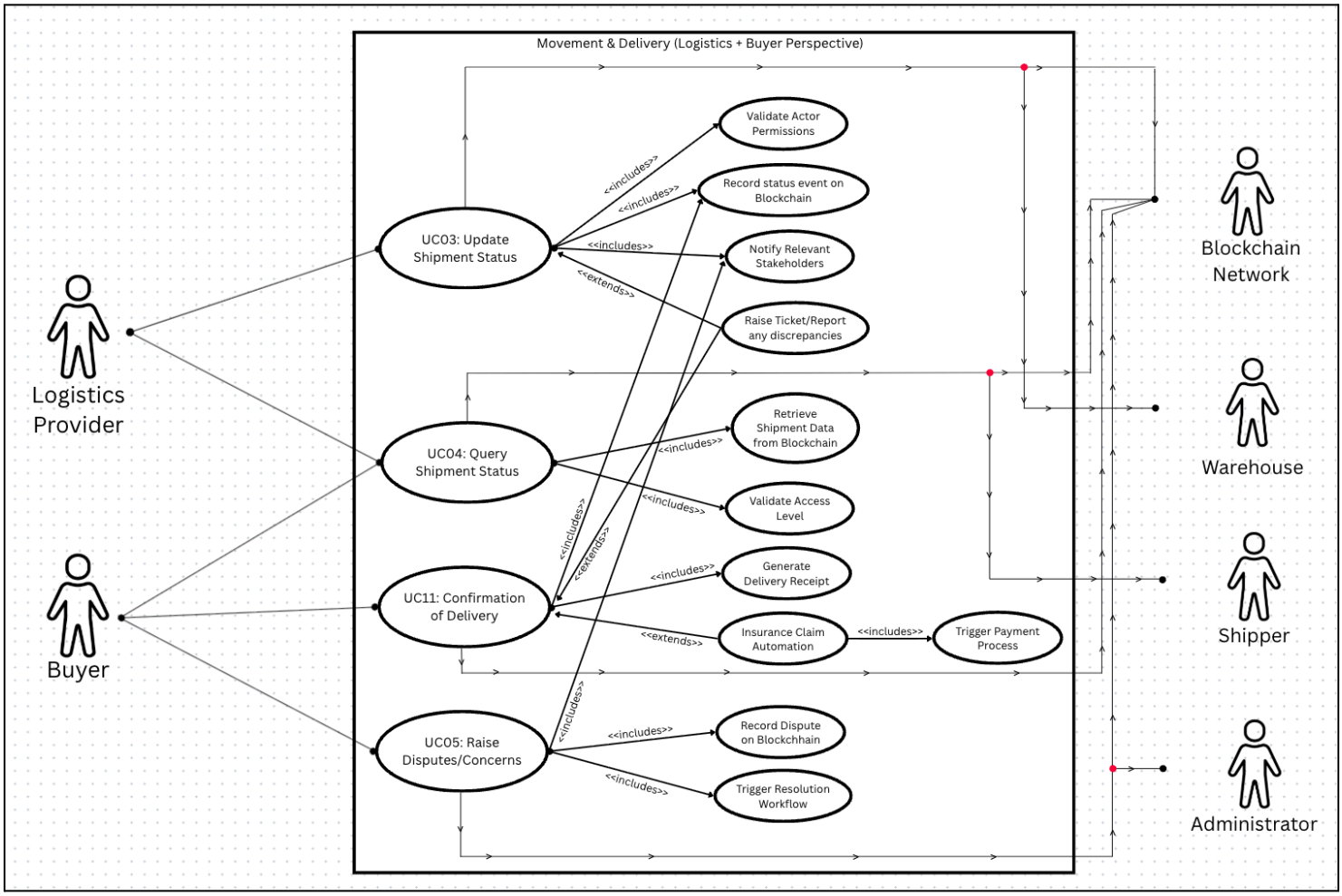
For example: in **UCD01**, the red circle is present in the connection between UC04: Query Shipment Status and secondary actors (Buyer (Customer or Retailer) & Blockchain Network). The circle shows that we can count these as two different paths, one to Buyer (Customer or Retailer) and the other to Blockchain Network. Rather than showing two different lines from UC04: Query Shipment Status, there is one, with the red circle which marks the second path to the other secondary actor.

The ^ > v < symbols have been used for better visibility of the mapped paths from each use case to its secondary actors.

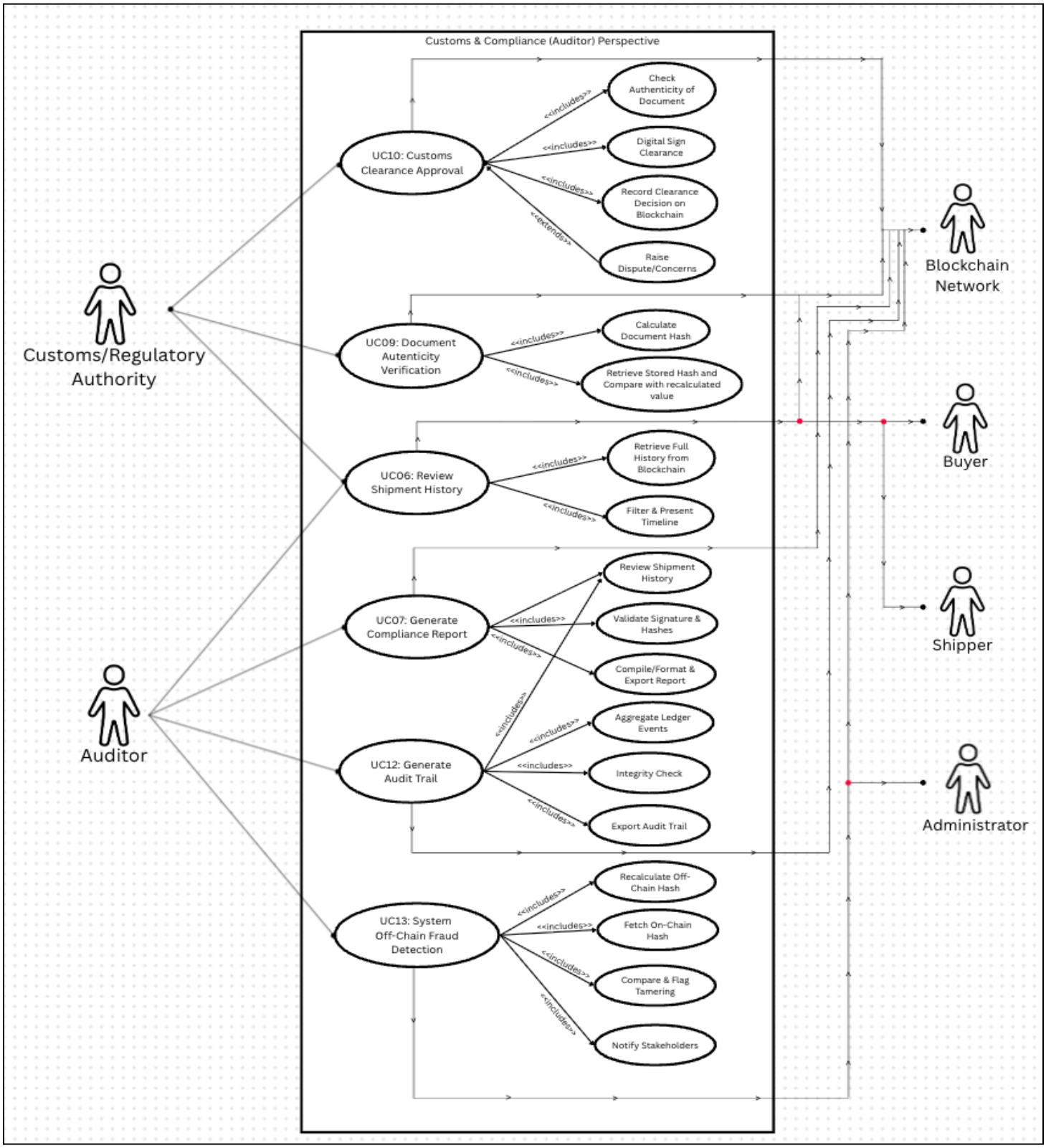
UCD01 - Create & Manage Shipment Records



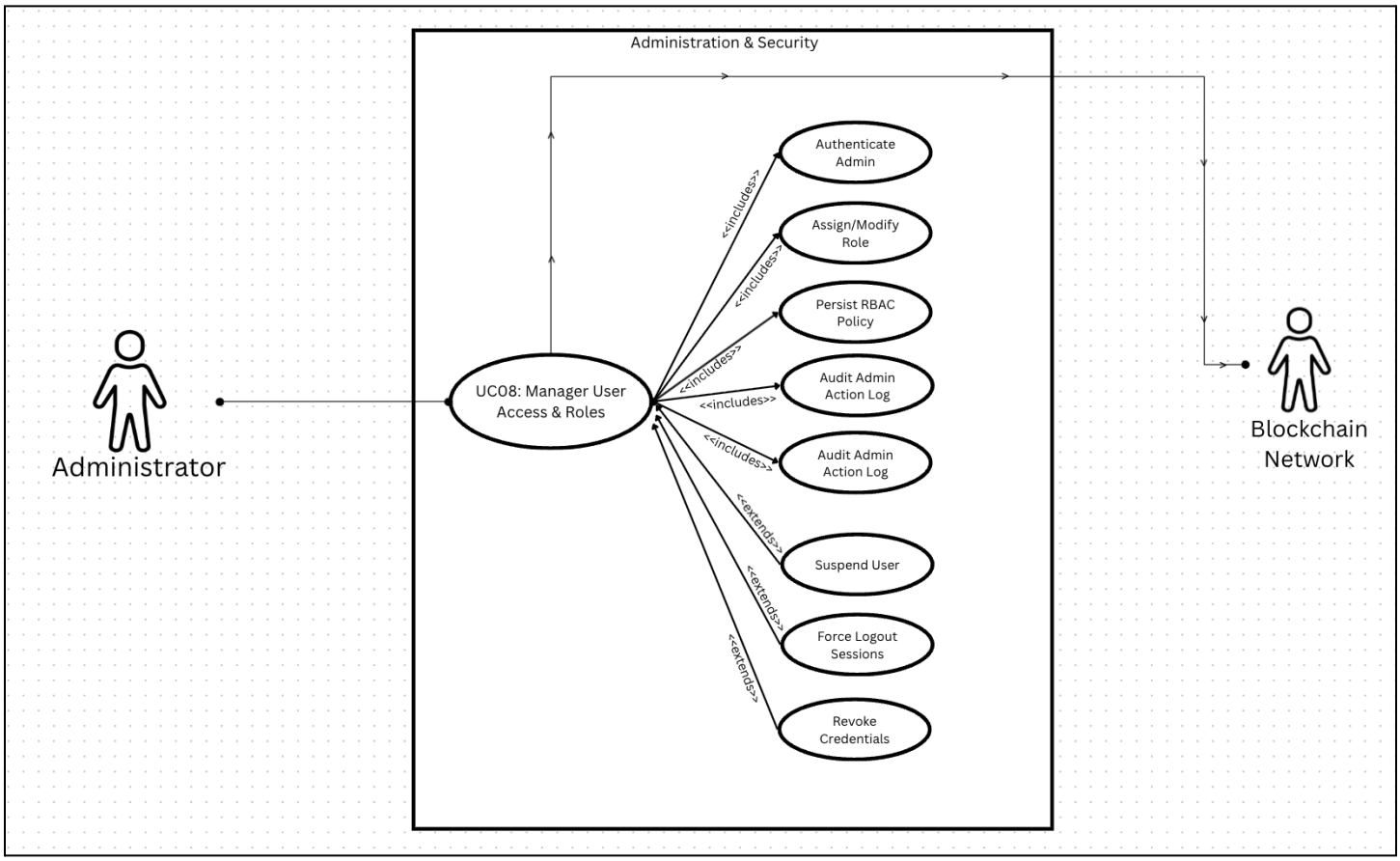
UCD02 - Movement & Delivery (Logistics + Buyer (Customer or Retailer) Perspective)



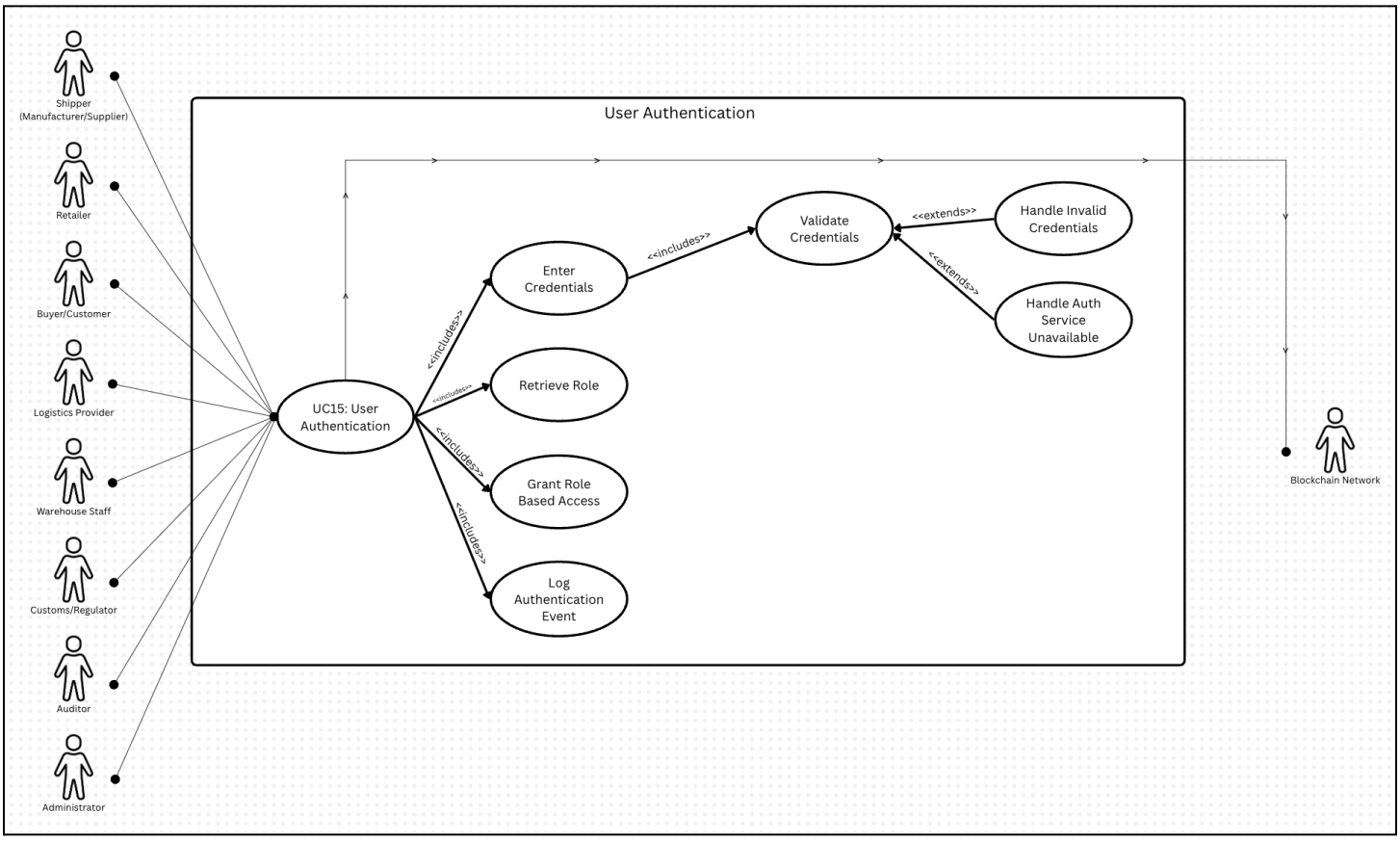
UCD03 - Customs Compliance (Auditor) Perspective



UCD04 - Administration & Security



UCD05 - User Authentication



1.10 Use Case Scenarios (STD)

The following is a sample list of the Use Case Scenarios (ie the Scenario Textual Descriptions → STD) for the system:

STD01: Create Shipment Record	**Tracebility**
Use Case Name: UC#01	UC01, FR1-FR4, R01, R02, R04, R11, NFR1, NFR8, A02, A03, A09
Scope & Description: Shipper (Manufacturer or Supplier) records a new shipment on the blockchain with product details, including shipment ID, origin, destination, product type, and participants.	
Primary Actors: Shipper (Manufacturer or Supplier)	
Stakeholders & Interests: - Shipper (Manufacturer or Supplier): wants to immutably register shipment details - Buyer (Customer or Retailer): expects accuracy of shipment details - Customs: relies on trusted shipment records	
Pre-condition: Shipper (Manufacturer or Supplier) must be authenticated and has permission to create shipment records	
Triggering Event: Shipper (Manufacturer or Supplier) initiates a new shipment entry through UI	
Main Scenario Steps: 1. Shipper (Manufacturer or Supplier) selects “Create Shipment” in the UI **(R01)** 2. System prompts for shipment details (product, origin, destination, parties) **(R04)** 3. Shipper (Manufacturer or Supplier) submits shipment details **(R02)** a. The system writes transactions to the blockchain. 4. Blockchain confirms the transaction and returns the shipment ID and is now available for status querying **(R11)**	
Post Condition: Shipment record is immutably stored on blockchain and assigned a unique ID.	
Extensions: If a blockchain transaction fails, an error is logged, and a retry is possible.	
Non-Functional Requirements: - Blockchain must store shipment records within 5 seconds **(NFR1)** - Immutable storage **(NFR8)**	
Technology & Data Variations List: - Smart contract for shipment creation - blockchain transaction validation	

Frequency of Occurrence: Every New Shipment	
Miscellaneous Comments: First step for every shipment lifecycle.	

STD02: Upload Shipment Documents	**Tracebility**
Use Case Name: UC#02	UC02, FR7, R01, R02, R12, R15, NFR3, NFR5, A09, A10
Scope & Description: Shipper (Manufacturer or Supplier) uploads supporting documents (e.g., bill of lading, invoice) to off-chain storage and stores the hash on the blockchain	
Primary Actors: Shipper (Manufacturer or Supplier)	
Stakeholders & Interests: - Shipper (Manufacturer or Supplier): provides proof documents - Customs: validates document authenticity - Buyer (Customer or Retailer): checks integrity of docs	
Pre-condition: Shipper (Manufacturer or Supplier) must be authenticated and shipment record (UC#01) already exists	
Triggering Event: Shipper (Manufacturer or Supplier) uploads a shipment-related document.	
Main Scenario Steps: 1. Shipper (Manufacturer or Supplier) selects the "Upload Document" option in the UI **(R01)** 2. Shipper (Manufacturer or Supplier) uploads files (off-chain if required) **(R02)** a. The system calculates the hash of the file. 3. Hash is stored on blockchain, file stored off-chain **(R12)** 4. System confirms document successfully recorded **(R15)**	
Post Condition: Document hash stored immutably; file retrievable off-chain	
Extensions: If hash mismatch occurs during later verification, the document is flagged as tampered.	
Non-Functional Requirements: - Secure hashing **(NFR5)** - Off-chain storage must be highly available **(NFR3)**	
Technology & Data Variations List: - IPFS/Cloud storage integration - blockchain hash anchoring	
Frequency of Occurrence: Every New Shipment or during customs clearance	

Miscellaneous Comments: Integrity check enables anti-fraud compliance	

STD03: Update Shipment Status	**Tracebility**
Use Case Name: UC#03	UC03, FR10-13, R05, R12, R15, NFR9, A02, A03, A08
Scope & Description: Shipper (Manufacturer or Supplier), logistics provider, or warehouse updates shipment milestones on blockchain	
Primary Actors: Shipper (Manufacturer or Supplier), Carrier, Warehouse Operator	
Stakeholders & Interests: - Shipper (Manufacturer or Supplier): need real-time update capability - Buyer (Customer or Retailer): requires visibility of status - Customs: monitors cross-border progress	
Pre-condition: Actor must be authenticated, Shipment exists on blockchain (UC#01)	
Triggering Event: Actor changes shipment milestone (e.g., "Picked Up," "In Transit," "Delivered")	
Main Scenario Steps: 1. The actor selects "Update Status." The actor enters a milestone + timestamp **(R05)** 2. Transaction (event) recorded on blockchain **(R12)** 3. Blockchain validates and updates immutable records **(R15)** 4. Alerts triggered for subscribed parties	
Post Condition: Shipment milestone updated immutably on blockchain	
Extensions: If node consensus fails, update retried	
Non-Functional Requirements: - Status updates must propagate to all nodes within 3 seconds **(NFR9)**	
Technology & Data Variations List: - Blockchain event triggers, smart contract milestone logic	
Frequency of Occurrence: Multiple times during the shipment lifecycle	
Miscellaneous Comments: Enables trusted real-time visibility	

STD04: Query Shipment Status	**Tracebility**
Use Case Name: UC#04	UC04, FR4, FR18,

Scope & Description: Buyer (Customer or Retailer) or Shipper (Manufacturer or Supplier) queries shipment status in real time through the system's UI, retrieving the most recent immutable blockchain record of the shipment.	FR37, R05, R06, R12, R15, NFR1, NFR4, A01, A02
Primary Actors: Buyer (Customer or Retailer), Shipper (Manufacturer or Supplier)	
Stakeholders & Interests: - Customer/Retailer: wants transparency about delivery progress and assurance of product location. - Manufacturer/Supplier: confirms goods are being tracked accurately and needs reliable data for supply chain planning.	
Pre-condition: Actor must be authenticated and shipment record exists on blockchain (UC#01)	
Triggering Event: Actor requests status lookup for a shipment ID.	
Main Scenario Steps: 1. The Customer or Retailer selects "Track Shipment" in the UI **(R06)**. 2. The Manufacturer or Supplier enters the shipment ID or searches by criteria **(R05)**. 3. The system queries blockchain records for the latest status and milestones **(R15)**. 4. The system displays immutable shipment status (e.g., "In Transit," "Cleared Customs," "Delivered") **(R12)**.	
Post Condition: User views accurate, blockchain-verified status in real time.	
Extensions: - If the shipment ID does not exist, the system returns an error message. - If the blockchain query fails, a fallback retry is triggered.	
Non-Functional Requirements: - Query must return results within 5 seconds **(NFR1)** - High availability of query services **(NFR4)**	
Technology & Data Variations List: - Blockchain data indexing - UI search integration	
Frequency of Occurrence: Frequently during the shipment lifecycle.	
Miscellaneous Comments: Supports transparency across the supply chain.	

STD05: Raise Dispute/Concerns	**Tracebility**

Use Case Name: UC#05	UC05, FR21, FR23, R10, R15, R16, NFR5, NFR8, A03, A09
Scope & Description: Buyer (Customer or Retailer) or Customs raises concerns/disputes if shipment conditions are not met (e.g., delays, missing documents, damaged goods). Disputes are logged on-chain for accountability.	
Primary Actors: Buyer (Customer or Retailer), Customs	
Stakeholders & Interests: - Customer/Retailer: ensures goods meet contractual conditions. - Customs: raises compliance concerns for cross-border shipments. - Manufacturer/Supplier: needs visibility into issues raised.	
Pre-condition: Actor must be authenticated, and shipment record exists and has progress/milestones logged.	
Triggering Event: Actor submits a dispute or concern via the system.	
Main Scenario Steps: 1. The Customer or Retailer selects the "Raise Dispute" option **(R10)**. 1.1. The Customer or Retailer enters the shipment ID and describes the issue (e.g., damage, delay) **(R10)**. 2. The system records the dispute on the blockchain for immutability **(R15)**. 3. The system allows further workflow, such as resolution, escalation, or insurance claim **(R16)**.	
Post Condition: Dispute immutably recorded; stakeholders alerted.	
Extensions: - If an invalid shipment ID is provided, the dispute is rejected.	
Non-Functional Requirements: - Disputes are recorded securely and immutably **(NFR8)** - Confidentiality of sensitive dispute details is ensured **(NFR5)**	
Technology & Data Variations List: - Blockchain logging for disputes - Alert/Message service integration	
Frequency of Occurrence: Occasionally, depending on shipment issues.	
Miscellaneous Comments: Forms foundation for dispute resolution workflows.	

STD06: Review Shipment History	**Tracebility**
Use Case Name: UC#06	UC06, FR36, FR52, R08, R15, R20,

Scope & Description: Buyer (Customer or Retailer), Auditor, or Customs reviews complete shipment history recorded on blockchain, ensuring transparency and traceability.	R26, NFR3, A02, A09
Primary Actors: Buyer (Customer or Retailer), Auditor, Customs	
Stakeholders & Interests: - Buyer (Customer or Retailer): verifies the full history of purchased goods. - Auditor: ensures regulatory compliance and integrity of records. - Customs: checks historical shipment activities for cross-border validation.	
Pre-condition: Actor must be authenticated, and shipment record exists and milestones/documents are logged.	
Triggering Event: Actor requests to view shipment history.	
Main Scenario Steps: 1. The Buyer (Customer or Retailer), Auditor, or Customs Officer selects "View Shipment History" from the system's UI **(R08)** 2. The actor enters a Shipment ID to retrieve the full historical record **(R20)** 3. The Blockchain Network retrieves all related immutable events and associated documents. **(R15)** 4. The system displays a chronological shipment history in a verified, tamper-proof format **(R26)**	
Post Condition: Full shipment history viewable in trusted, tamper-proof format.	
Extensions: - If shipment ID is invalid, error returned. - If blockchain node query fails, retry performed.	
Non-Functional Requirements: - Shipment history retrieval should complete within 5 seconds, ensuring efficient access to historical data **(NFR3)**	
Technology & Data Variations List: - Blockchain data query & indexing - UI history visualization	
Frequency of Occurrence: During audits, customs inspections, or buyer review.	
Miscellaneous Comments: Critical for trust and transparency in the supply chain.	

STD07: Generate Compliance Report	**Tracebility**

Use Case Name: UC#07	UC07, FR54, FR55, R26, R28, NFR3, NFR8, A02, A03
Scope & Description: Auditor generates compliance and transparency reports from blockchain data to ensure all transactions and records adhere to regulatory standards.	
Primary Actors: Auditor	
Stakeholders & Interests: - Auditor: Needs accurate, tamper-proof records for compliance validation - Administrator: Ensures report generation functions securely and efficiently - Regulators: Rely on verifiable data for audits	
Pre-condition: Auditor must be authenticated and blockchain contains verified shipment and transaction data.	
Triggering Event: Auditor requests to view the compliance report.	
Main Scenario Steps: 1. Auditor navigates to the "Compliance Reports" section in the system **(R26)** 1.1. Auditor selects filters such as date range, shipment type, or entity 1.2. System queries the blockchain ledger for all relevant records 1.3. System retrieves shipment, transaction, and clearance data for the specified criteria 2. System compiles the retrieved data into a structured compliance report **(R28)** 2.1. System formats the report with verifiable data fields (timestamps, digital signatures) 2.2. System displays or exports the report in a tamper-proof format	
Post Condition: Compliance report is available for viewing, export, or audit submission.	
Extensions: - If the blockchain query fails, the retry mechanism triggers.	
Non-Functional Requirements: - Report generation must complete within 4 seconds **(NFR3)** - Report format must be immutable and digitally verifiable **(NFR8)**	
Technology & Data Variations List: - Blockchain query aggregation - Report export to PDF/CSV	
Frequency of Occurrence: Periodic audits, quarterly reviews, or	

regulatory inspections.	
Miscellaneous Comments: Supports audit readiness and transparency goals.	

STD08: Manage User Access & Roles	**Tracebility**
Use Case Name: UC#08	UC08, FR24-FR29, R29, R30, NFR5, A06
Scope & Description: Administrator manages user accounts, assigns roles, and controls system permissions for blockchain access.	
Primary Actors: Administrator	
Stakeholders & Interests: - Administrator: Maintains security and access control. - All Actors: Depend on correct role assignments for proper permissions.	
Pre-condition: Admin must be authenticated with system-level privileges.	
Triggering Event: Admin selects the "User Management" screen on the System UI	
Main Scenario Steps: 1. Admin opens the "Manage Users & Roles" section **(R29)** 1.1. Admin adds or selects a user account 1.2. Admin assigns a specific role to the user 1.3. Confirmation message displays on the System UI. 2. System updates blockchain permission records accordingly **(R30)**	
Post Condition: Updated access permissions stored and enforced across the system.	
Extensions: - If the user does not exist, the system prompts for new account creation. - If the permission update fails, rollback to the previous configuration.	
Non-Functional Requirements: - System must ensure secure authentication **(NFR5)**	
Technology & Data Variations List: - Access control management - Role-based permission smart contracts	
Frequency of Occurrence: During onboarding, offboarding, or security audits	

Miscellaneous Comments: Critical for maintaining the integrity and security of blockchain operations.	

STD09: Document Authenticity Verification	**Tracebility**
Use Case Name: UC#09	UC09, FR19, FR35, R09, R15, NFR3, NFR5, NFR6, NFR8, A09, A10
Scope & Description: Actors validate document authenticity by recalculating the document hash and comparing it with the one stored on the blockchain.	
Primary Actors: Buyer (Customer or Retailer), Auditor, Customs	
Stakeholders & Interests: - Buyer (Customer or Retailer): Confirms received documents are genuine. - Auditor: Ensures compliance with authenticity standards. - Customs: Validates import/export documents.	
Pre-condition: Actor must be authenticated, and document and corresponding blockchain hash record exist.	
Triggering Event: Actor initiates action to verify documents.	
Main Scenario Steps: 1. Actor selects a document and requests verification **(R09)** 1.1. System recalculates the document hash locally 2. System retrieves stored hash from blockchain **(R15)** 2.1. System compares both hashes 2.1.1. If the hashes match, the system confirms "Document Authentic." 2.1.2. If hashes differ, the system flags "Document Tampered."	
Post Condition: Document authenticity verified and audit trail updated.	
Extensions: - If the blockchain node is unavailable, the system retries verification. - If the document format is unsupported, the system alerts the user.	
Non-Functional Requirements: - Verification must be completed under 4 seconds **(NFR3)** - Data integrity must be ensured through digital signatures and immutable storage **(NFR6, NFR8)** - Confidentiality of document access must be enforced via role-based authentication **(NFR5)**	
Technology & Data Variations List: - Hashing algorithms	

- Blockchain hash retrieval	
Frequency of Occurrence: Every document submission, validation, or audit review.	
Miscellaneous Comments: Essential for trust and data integrity in the supply chain ecosystem.	

STD10: Customs Clearance Approval	**Tracebility**
Use Case Name: UC#10	UC10, FR30-FR33, R12, R17-R19, NFR1, NFR8, A02, A07
Scope & Description: Customs officers record shipment clearance or hold decisions directly on the blockchain for transparent and verifiable tracking.	
Primary Actors: Customs	
Stakeholders & Interests: - Customs: Ensures legitimate shipment processing. - Buyer (Customer or Retailer): Needs assurance that the shipment has passed customs. - Shipper (Manufacturer or Supplier): Needs trusted confirmation before proceeding with delivery. - Auditor: Verifies the authenticity of clearance records.	
Pre-condition: Custom Officer must be authenticated and shipment record exists with associated documentation.	
Triggering Event: Customs opens a shipment record and selects the Record Clearance Decision screen.	
Main Scenario Steps: 1. Customs officer accesses shipment details **(R17)** 2. Officer records decision: Cleared or Held for Review **(R18)** 2.1. System notifies relevant parties (Buyer (Customer or Retailer), Supplier) 3. Clearance approval is digitally signed **(R19)** 4. System logs the decision and timestamp to the blockchain **(R12)** 4.1. Blockchain immutably stores the clearance record for audit	
Post Condition: Clearance decision permanently recorded and accessible to all authorized actors.	
Extensions: - If a blockchain transaction fails, the system retries automatically.	
Non-Functional Requirements: - Decision recording must finalize within 5 seconds **(NFR1)** - Blockchain transaction integrity ensured **(NFR8)**	

Technology & Data Variations List: - Smart contract clearance function - Blockchain event alerts	
Frequency of Occurrence: Every shipment inspection or cross-border clearance.	
Miscellaneous Comments: Enhances trust and reduces fraudulent clearance claims.	

STD11: Confirmation of Delivery	**Tracebility**
Use Case Name: UC#11	UC11, FR20, FR34, R05-R07, R11, R12, NFR6, NFR8, NFR9, A03, A09
Scope & Description: Delivery is confirmed on the blockchain by the receiver, finalizing the shipment lifecycle.	
Primary Actors: Buyer (Customer or Retailer)	
Stakeholders & Interests: - Buyer (Customer or Retailer): Wants to confirm receipt securely and immutably - Shipper (Manufacturer or Supplier): Wants proof of delivery and transparency - Blockchain Network: Ensures delivery confirmation event is valid and immutable	
Pre-condition: Buyer (Customer or Retailer) must be authenticated and shipment record exists and is marked “In Transit”	
Triggering Event: Buyer (Customer or Retailer) receives shipment and confirms delivery through the blockchain UI.	
Main Scenario Steps: 1. Buyer (Customer or Retailer) selects “Confirm Delivery” **(R06)** 2. System prompts for shipment ID and confirmation **(R07)** 3. Buyer (Customer or Retailer) digitally signs the confirmation **(R07)** 4. Blockchain validates and records delivery confirmation **(R11, R12)** 5. Shipment status updates to “Delivered” **(R05)**	
Post Condition: Delivery event is stored immutably on the blockchain; shipment lifecycle is marked “Completed.”	
Extensions: - If digital signature fails or blockchain node validation is delayed, the system retries transaction automatically.	
Non-Functional Requirements: - Transactions must be digitally signed **(NFR6)**	

- Event synchronization across nodes within 3 seconds (**NFR9)** - Immutable storage **(NFR8)**	
Technology & Data Variations List: - Smart contract validates delivery confirmation. - Digital signature validation on-chain.	
Frequency of Occurrence: Once per shipment, upon delivery	
Miscellaneous Comments: Marks the official end of a shipment's lifecycle and proof-of-receipt for compliance and insurance.	

STD12: Generate Audit Trail	**Tracebility**
Use Case Name: UC#12	UC12, FR28, FR55, R15, R26-R28, NFR3, NFR4, NFR8, A02, A03
Scope & Description: Immutable blockchain transaction history is compiled into regulator/auditor-approved audit trails.	
Primary Actors: Auditor	
Stakeholders & Interests: - Auditor: Needs verifiable shipment history for compliance and accountability. - Regulators: Require transparency in shipment transactions.	
Pre-condition: Auditor must be authenticated and blockchain ledger contains transaction history for completed or ongoing shipments.	
Triggering Event: Auditor initiates "Generate Audit Trail" request through UI.	
Main Scenario Steps: 1. Auditor selects "Generate Audit Trail" **(R26)** 2. System fetches shipment transaction data from blockchain **(R15)** 3. Auditor verifies signatures and hashes for each event **(R27)** 4. System compiles report summarizing shipment history **(R28)** 5. Auditor exports or reviews the report for compliance review **(R28)**	
Post Condition: Audit trail report generated and stored securely; available for export or regulator submission.	
Extensions: - If a node is unsynchronized or a transaction is missing, the system re-queries backup nodes.	
Non-Functional Requirements: - Ledger must remain immutable **(NFR8)** - Data retrieval from blockchain ~ <4 seconds **(NFR3)** - High availability across nodes **(NFR4)**	

Technology & Data Variations List: - Smart contract API for audit report generation. - Automated report template for regulators.	
Frequency of Occurrence: Periodically (monthly, quarterly, or on request).	
Miscellaneous Comments: Critical for compliance audits and operational transparency.	

STD13: System off-chain fraud detection	**Tracebility**
Use Case Name: UC#13	UC13, FR51, R11, R12, R16, R27, R29, NFR3, NFR6, NFR8, A03, A09, A10
Scope & Description: System verifies off-chain stored documents by matching recalculated hashes with on-chain hashes to detect fraud or data tampering.	
Primary Actors: Blockchain Network	
Stakeholders & Interests: - Auditor: Needs tamper detection for compliance. - Customs/Regulators: Require validation to prevent document fraud. - System Admin: Oversees fraud detection logs.	
Pre-condition: Documents are stored off-chain, and their hash values are stored on-chain.	
Triggering Event: System or Auditor initiates a hash verification process.	
Main Scenario Steps: 1. System recalculates off-chain document hash **(R27)** 2. Blockchain retrieves original on-chain hash **(R12)** 3. Hash values are compared for verification **(R11)** 4. If a mismatch is found, an alert is generated **(R16)** 5. Admin reviews discrepancy and triggers follow-up **(R29)**	
Post Condition: Verified documents confirmed as authentic or flagged for tampering.	
Extensions: - If off-chain file retrieval fails, backup retrieval is attempted; an alert is logged for the admin.	
Non-Functional Requirements: - Data retrieval ~ <4 seconds **(NFR3)** - Digital signatures for transaction authenticity **(NFR6)** - Blockchain immutability **(NFR8)**	

Technology & Data Variations List: - Hash verification algorithms (SHA-256). - Smart contracts auto-trigger fraud alerts.	
Frequency of Occurrence: On-demand or scheduled verification cycles.	
Miscellaneous Comments: Enhances trust and compliance by detecting tampering between on-chain and off-chain records.	

STD14: Insurance Claim Automation	**Tracebility**
Use Case Name: UC#14	UC14, FR44-FR47, R11, R13, R16, R27, R29, NFR4, NFR8, NFR9, A03, A05, A09
Scope & Description: If receiver's goods are delayed or damaged, smart contracts automatically trigger insurance claim processing.	
Primary Actors: Blockchain Network (Smart Contract)	
Stakeholders & Interests: - Buyer (Customer or Retailer): Wants automatic compensation for losses. - Supplier: Needs verified claim events before payout. - Logistics Provider: Wants to minimize delays triggering unnecessary claims.	
Pre-condition: Delivery or shipment milestone exists and is linked with an active insurance policy.	
Triggering Event: Smart contract detects condition for claim (e.g., delay, damage, or non-delivery).	
Main Scenario Steps: 1. Blockchain monitors shipment milestones **(R11)** 2. Delay/damage threshold met **(R16)** 3. Smart contract automatically generates claim event **(R13)** 4. Claim validation and verification process triggered **(R27)** 5. System logs payout confirmation and notifies stakeholders **(R29)**	
Post Condition: Insurance claim initiated, verified, and processed automatically via smart contract.	
Extensions: - If a smart contract fails or the external API is not reachable, the system retries or alerts the supplier manually.	
Non-Functional Requirements: - Smart contracts ensure reliability **(NFR4)** - Event synchronization ~ <3 seconds **(NFR9)** - Immutable record of claim **(NFR8)**	

Technology & Data Variations List: - Smart contract linked to the Supplier dispute module. - Blockchain event listener for delay triggers.	
Frequency of Occurrence: Only when claim conditions are met.	
Miscellaneous Comments: Supports automation of post-shipment workflows, reducing manual claim processing time.	

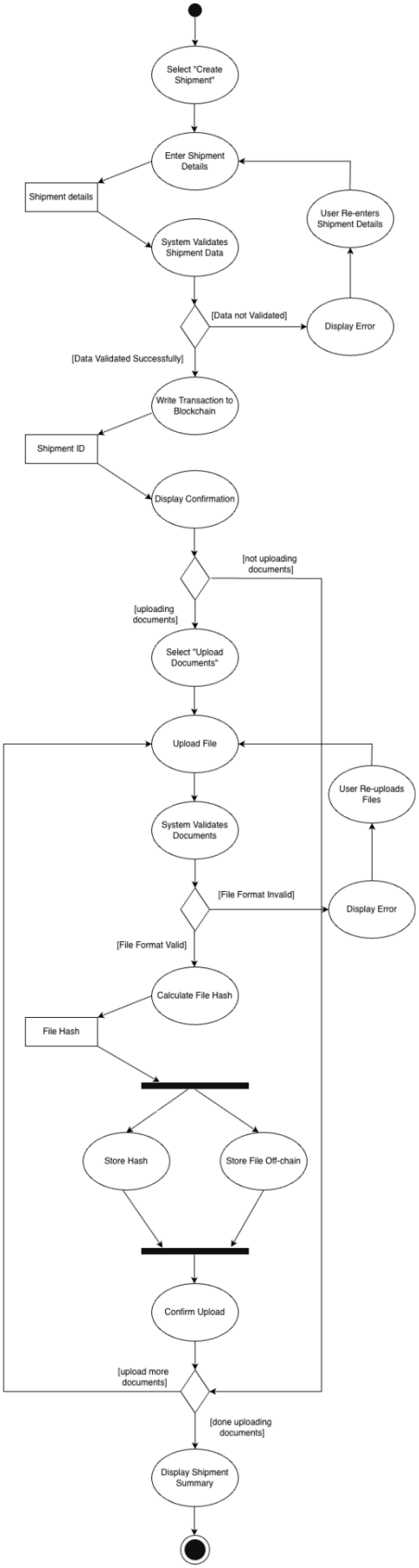
STD15: User Authentication	**Tracebility**
Use Case Name: UC#15	UC15, FR59, R14, R15, R29, R36, NFR4, NFR5, A01, A06
Scope & Description: System validates user credentials and grants access to authorized functionalities based on predefined roles (e.g., Buyer, Shipper, Customs, Auditor, Admin).	
Primary Actors: All Users (Buyer, Shipper, Customs Officer, Auditor, Admin)	
Stakeholders & Interests: - Admin: Ensures proper role-based access control and system security, and needs to prevent unauthorized access or privilege escalation. - All Users: Require secure authentication to perform system operations according to their permissions.	
Pre-condition: User has a valid registered account in the system with assigned roles and credentials.	
Triggering Event: User attempts to log in via the system's authentication interface.	
Main Scenario Steps: 1. User enters credentials (username and password) into the login interface **(R36)** 2. System validates credentials against stored records **(R36)** 2.1. If username and password matches, then the user is successfully authenticated 2.2. If username and password does not match, then the system displays an error message and the user is prompted to enter their credentials again 3. If credentials are valid, the system retrieves the user's assigned role **(R29)** 4. System grants access to permitted modules and dashboards based on role. **(R14)** 5. System logs the authentication event with timestamp and user ID on the blockchain for auditability **(R15)**	

Post Condition: User is successfully authenticated and granted access to authorized functionalities. Authentication record is securely logged.	
Extensions: - If credentials are invalid, the system displays an "Invalid Credentials" error and prompts for re-entry - If authentication service is unavailable, a retry or maintenance notification is triggered	
Non-Functional Requirements: - Login process should be fast and reliable **(NFR4)** - The system should allow only authorized users to access their respective modules **(NFR5)**	
Technology & Data Variations List: - Simple database check for username and password. - Role assignment handled within the system (e.g., Admin, Buyer, Shipper).	
Frequency of Occurrence: Every time a user accesses the system or during re-authentication after session timeout.	
Miscellaneous Comments: User authentication is a prerequisite for all protected operations in the system. It ensures integrity, confidentiality, and non-repudiation within blockchain-based workflows.	

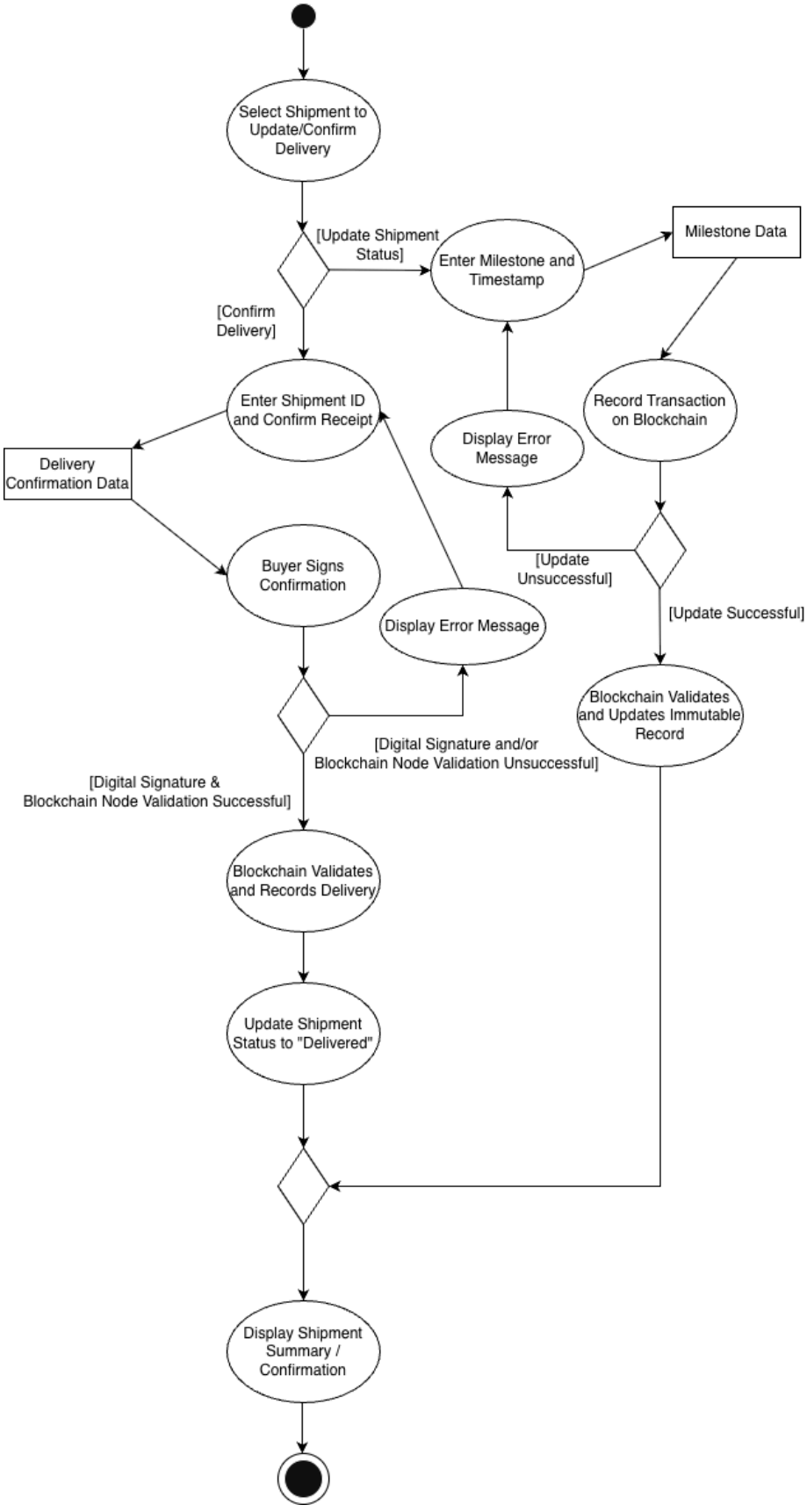
1.11 Activity Diagrams

The following is a list of the Activity Diagrams that illustrate the key workflows and dynamic behaviours within the Blockchain Shipment Management Tracking System. Each diagram represents the flow of control between system processes, user actions, and actors, highlighting automation, decision points, and data synchronization across stakeholders.

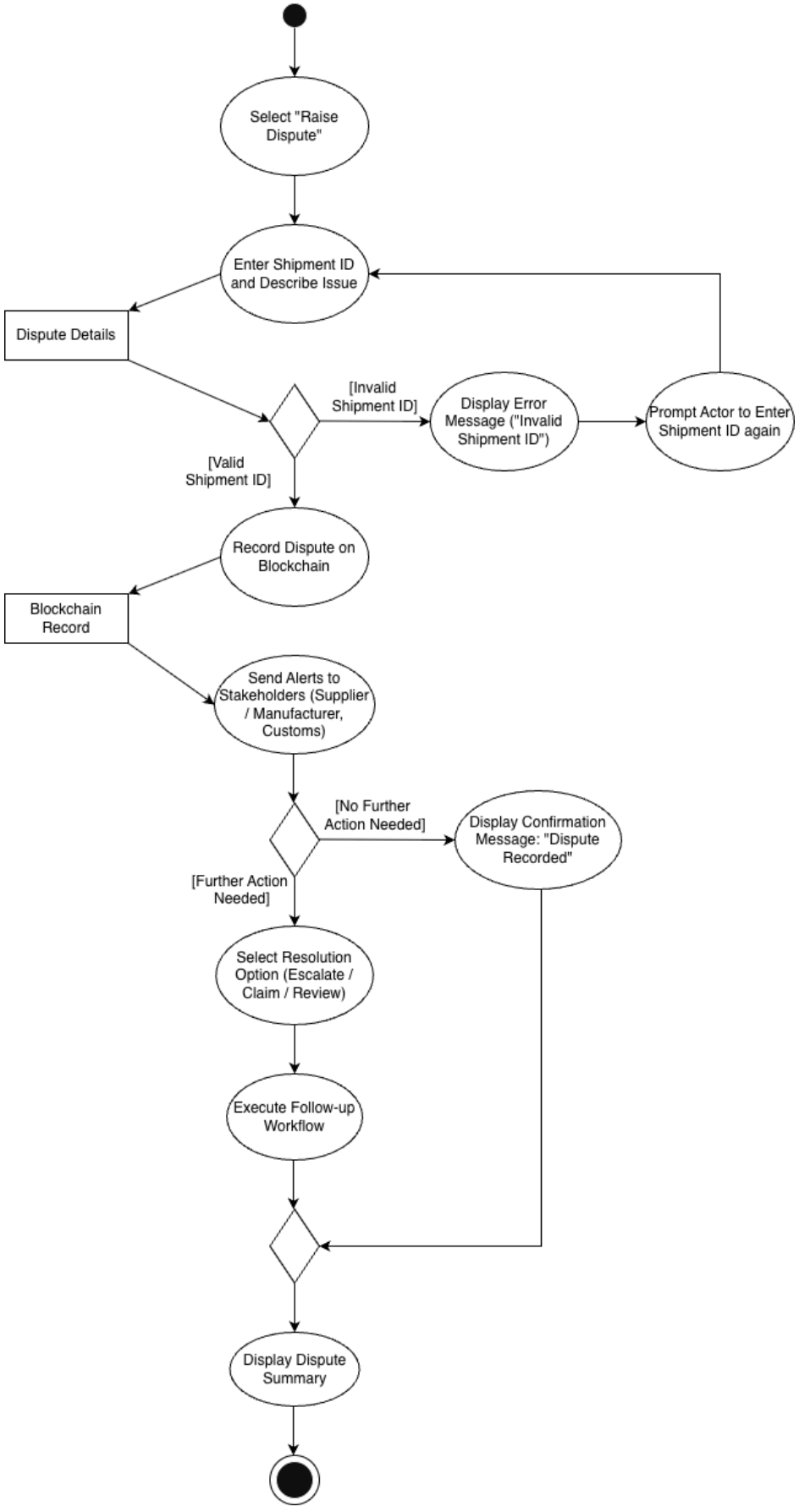
AD01 - Shipment Creating and Document Upload (UC01 & UC02)



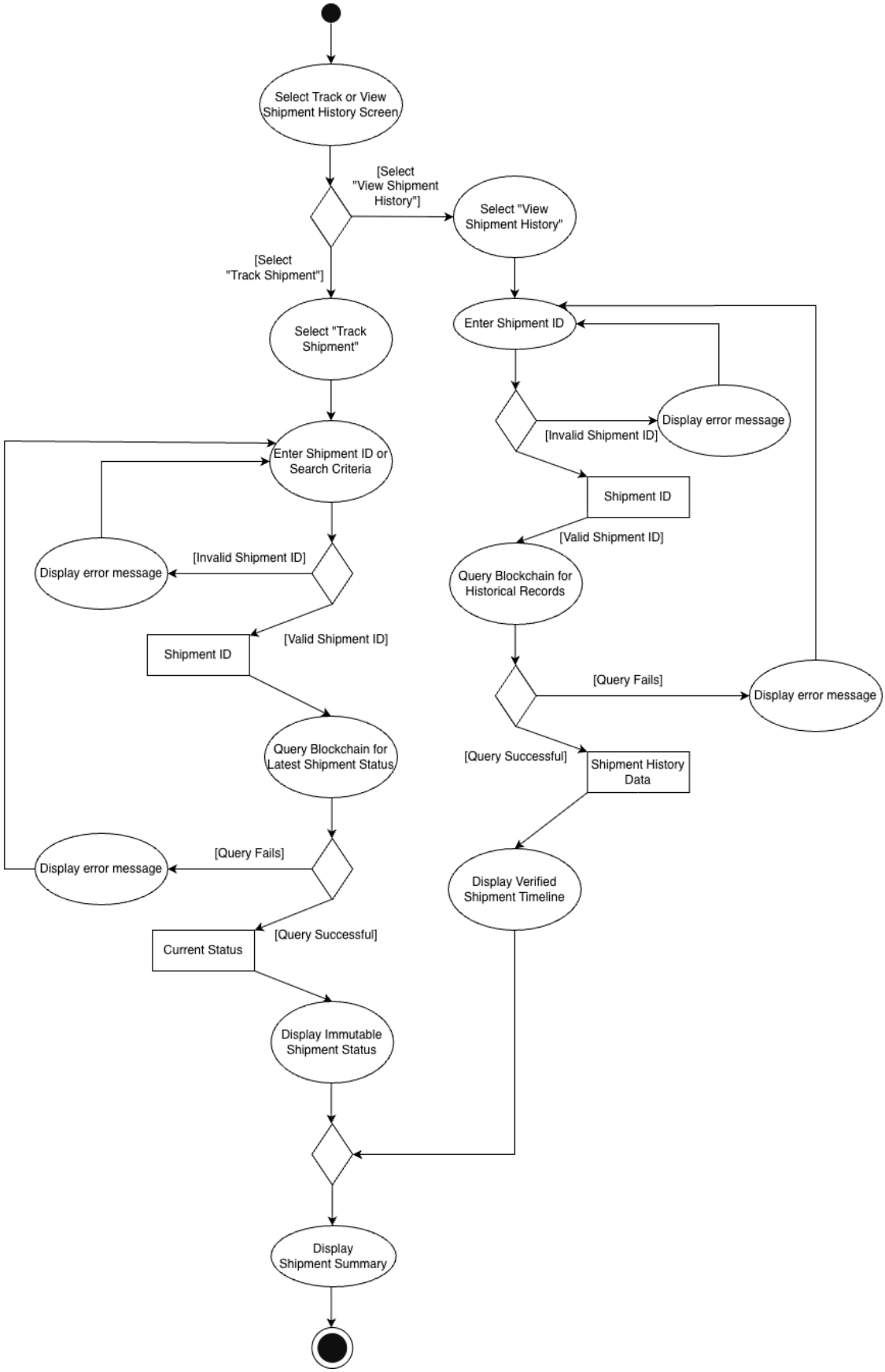
AD02 - Shipment Status Update and Delivery Confirmation (UC03 & UC11)



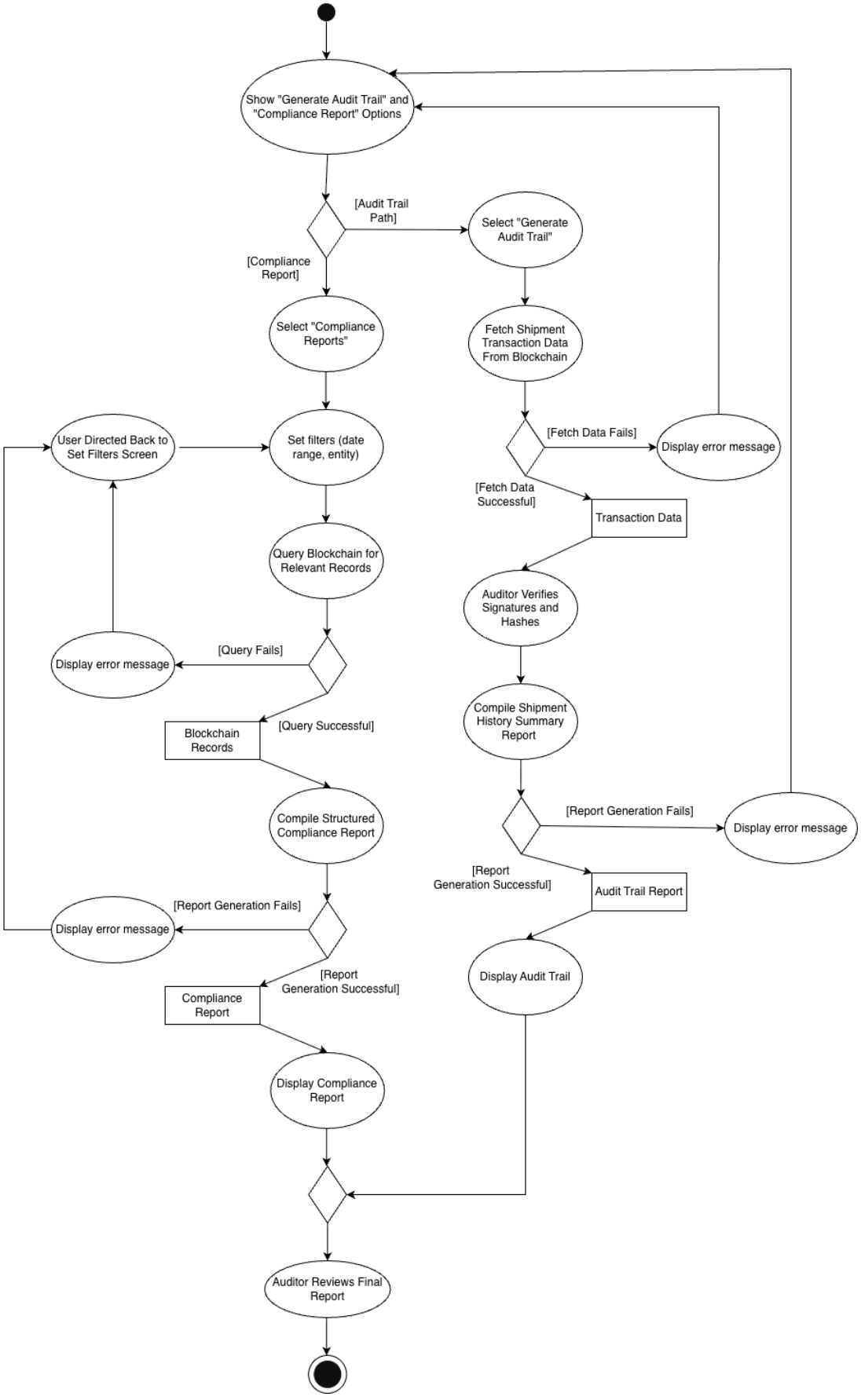
AD03 - Dispute Handling Workflow (UC05)



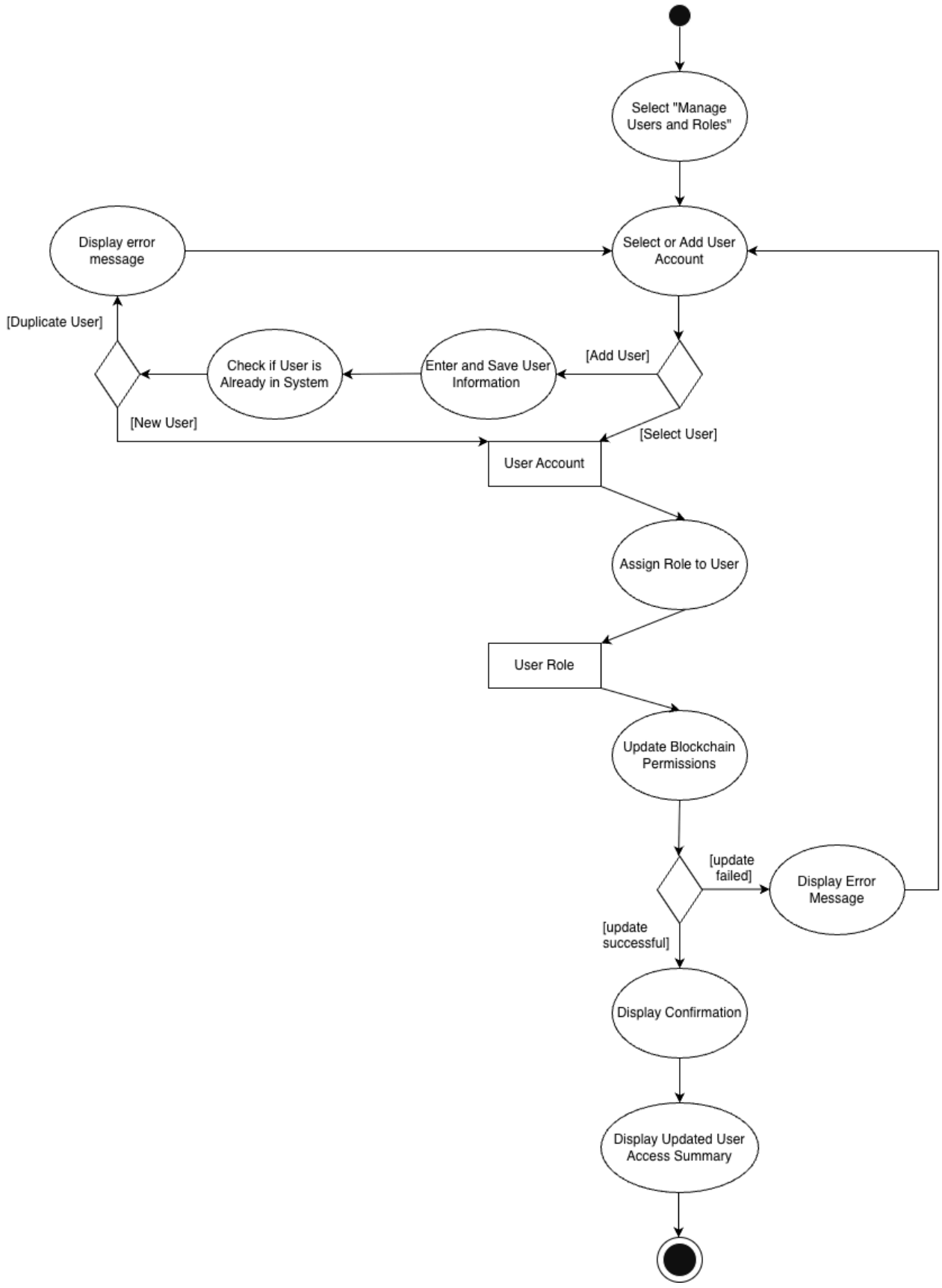
AD04 - Shipment Tracking and Review (UC04 & UC06)



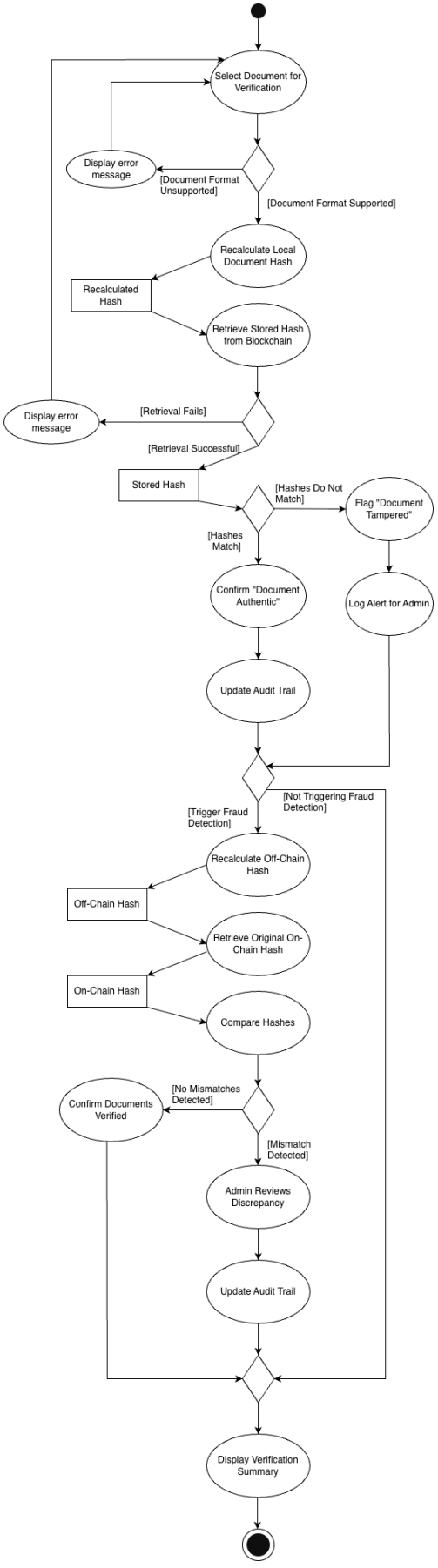
AD05 - Compliance and Audit Reporting (UC07 & UC12)



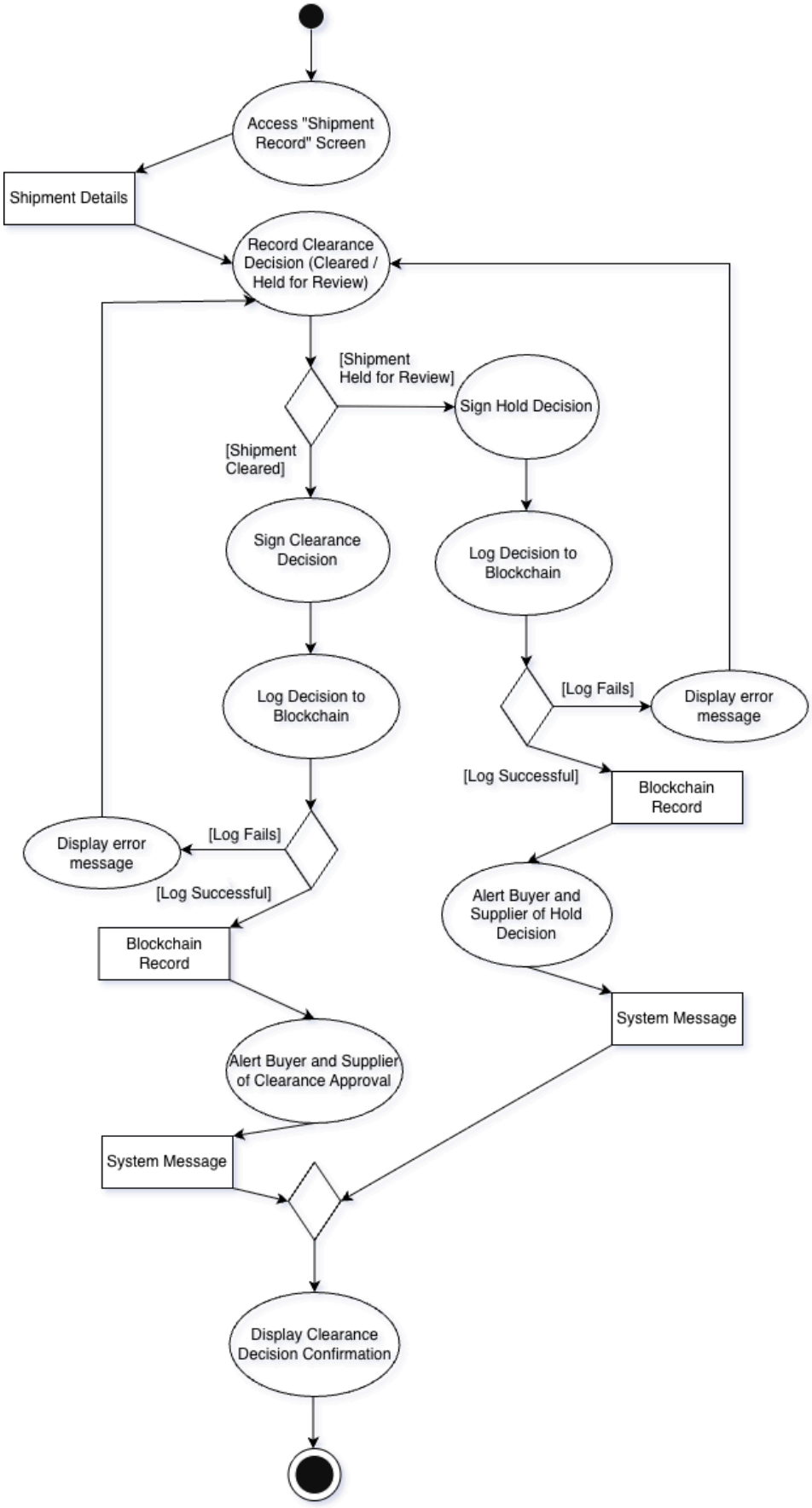
AD06 - User Management and Roles (UC08)



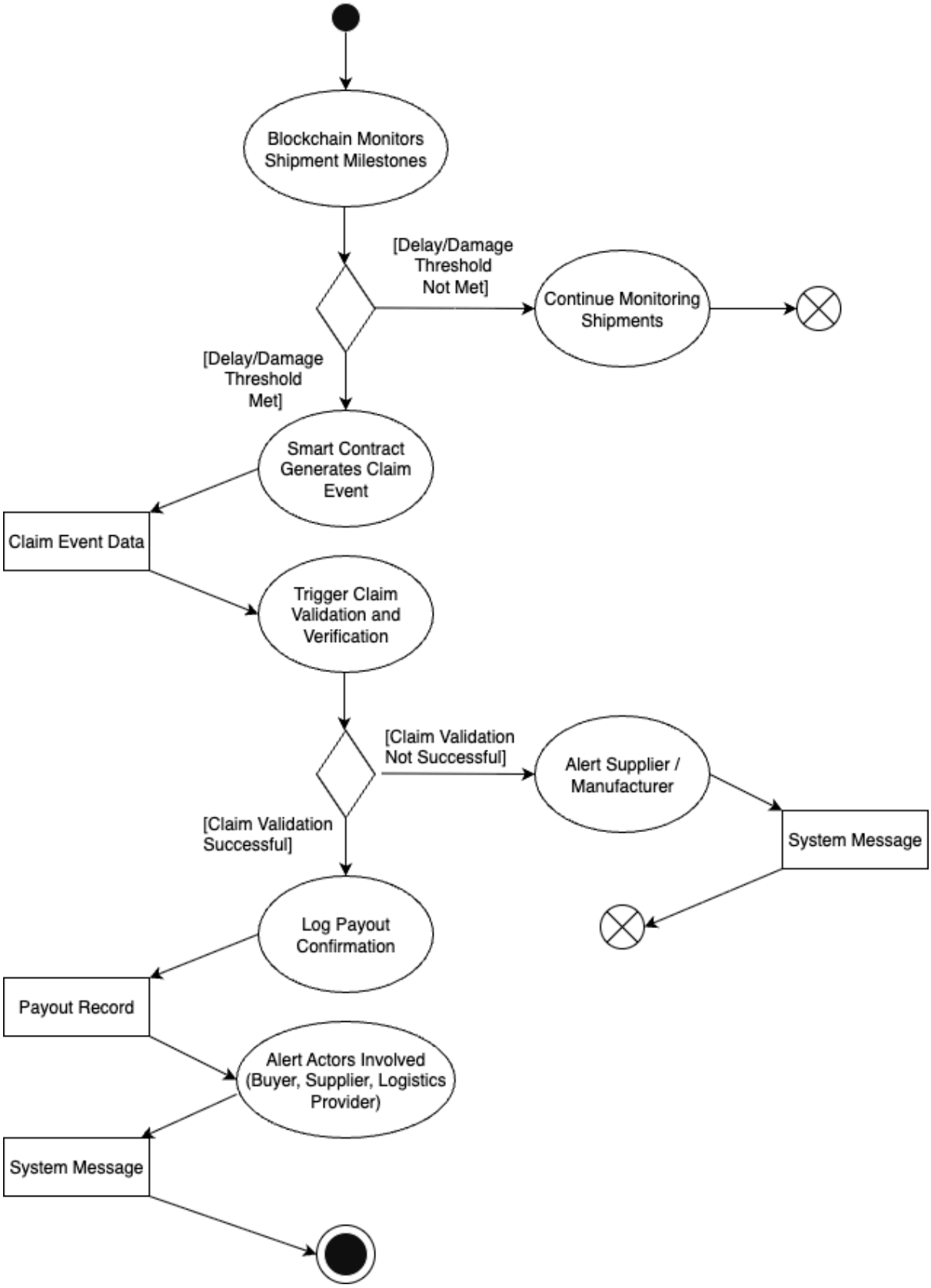
AD07 - Document Verification and Fraud Detection (UC09 & UC13)



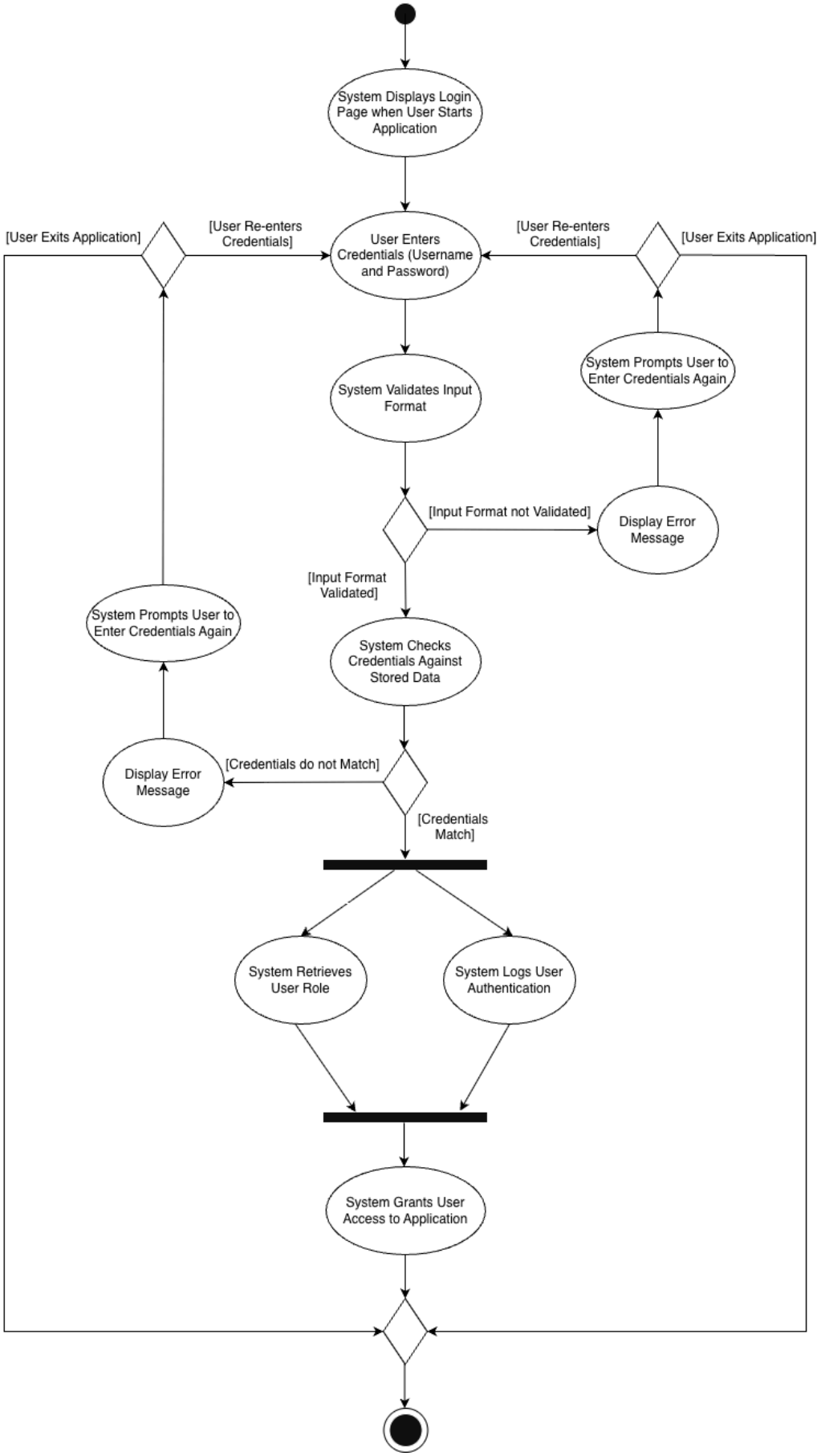
AD08 - Customs Clearance Approval (UC10)



AD09 - Insurance Claim Automation (UC14)



AD10 - User Authentication (UC15)



2. Phase II: System Design

2.1 Fixes Applied on Phase-I

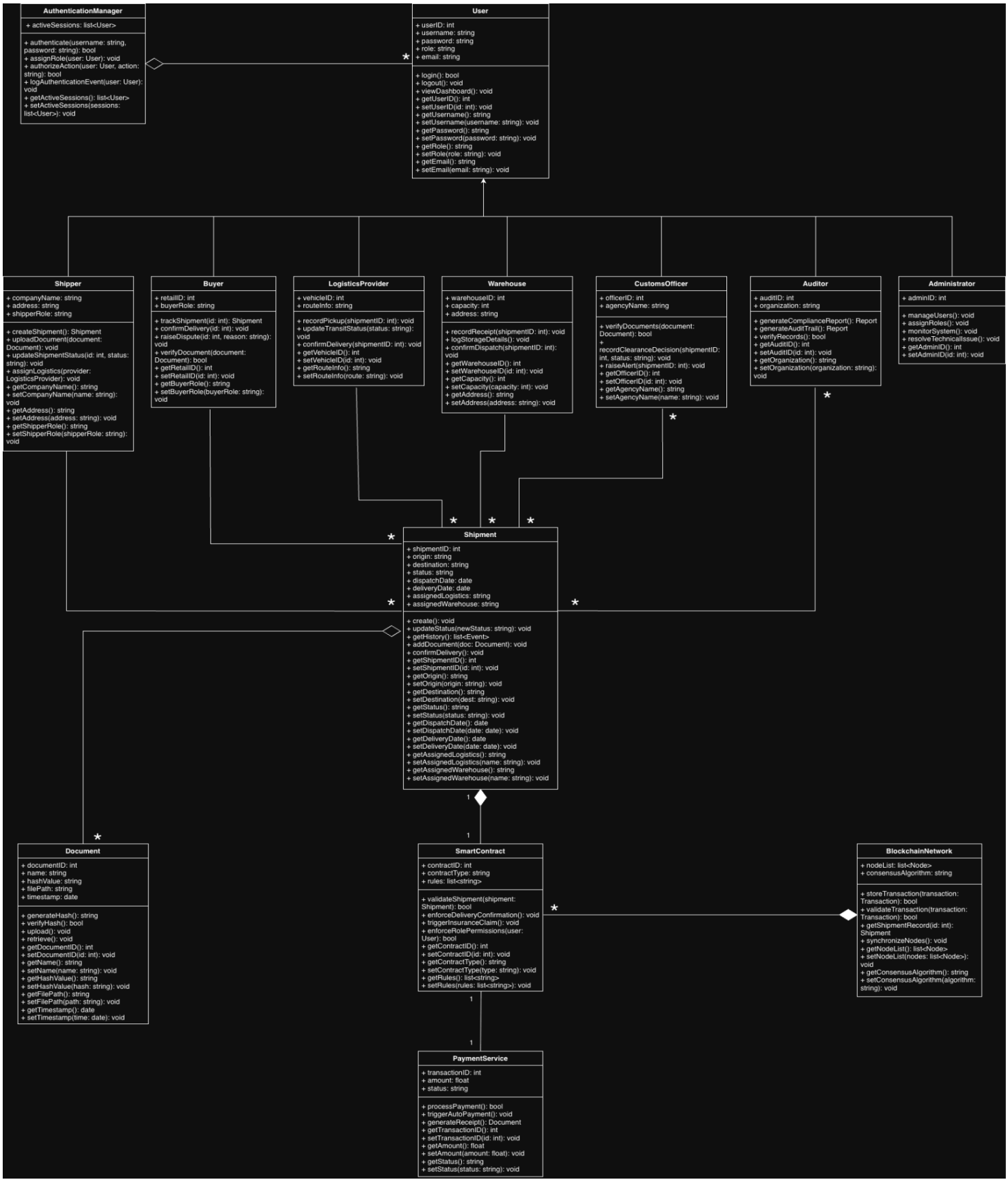
Below is a table that consists of the fixes we made to Phase-I:

Fix #	Affected Diagram/Table/Item	Issue	Applied Fixes
1	UC15 - User Autentication (New)	Make user authentication its own use case since authentication flow was added to a few activity diagrams (from TA feedback).	Added new use case for user authentication.
2	R36 - The system authenticates users and validates their credentials before granting access. (New)	Missing responsibility for the system to validate credentials, which is needed for the scenario steps for STD15.	Added new responsibility for the system to validate user credentials, which can be used for the scenario steps in STD15.
3	UCD05 - User Authentication (New)	Missing use case diagram for UC15 - User Authentication (from TA feedback).	Added new use case diagram for user authentication.
4	STD15 - User Authentication (New)	Missing scenario textual description for UC15 - User Authentication (from TA feedback).	Added new scenario textual description for user authentication.
5	AD10 - User Authentication (New)	Missing activity diagram for UC15 - User Authentication (from TA feedback).	Added new activity diagram for user authentication.
6	AD01 - Shipment Creating and Document Upload	Remove authentication flow since we made it its own use case and activity diagram, and missing extensions from STD01 and STD02.	Removed authentication flow, added extensions from STD01 & STD02, and added user authentication as pre-condition in STD01 and STD02.
7	AD02 - Shipment Status Update and Delivery Confirmation	Remove authentication flow since we made it its own use case and activity	Removed authentication flow, added extensions from

		diagram, and missing extensions from STD03 and STD11.	STD03 and STD11, and added user authentication as pre-condition in STD03 and STD11.
8	AD03 - Dispute Handling Workflow	Remove authentication flow since we made it its own use case and activity diagram, and missing extensions from STD05.	Removed authentication flow, added extensions from STD05, and added user authentication as pre-condition in STD05.
9	AD04 - Shipment Tracking and Review	Remove authentication flow since we made it its own use case and activity diagram, and missing extensions from STD04 and STD06.	Removed authentication flow, added extensions from STD04 and STD06, and added user authentication as pre-condition in STD04 and STD06.
10	AD05 - Compliance and Audit Reporting	Remove authentication flow since we made it its own use case and activity diagram, and missing extensions from STD07 and STD12.	Removed authentication flow, added extensions from STD07 and STD12, and added user authentication as pre-condition in STD07 and STD12.
11	AD06 - User Management and Roles	Remove authentication flow since we made it its own use case and activity diagram, and missing extensions from STD08.	Removed authentication flow, added extensions from STD08, and added user authentication as pre-condition in STD08.
12	AD07 - Document Verification and Fraud Detection	Missing extensions from STD09 and STD13.	Added extensions from STD09 and STD13, and added user authentication as pre-condition in STD09 and STD13.
13	AD08 - Customs Clearance Approval	Missing extensions from STD10.	Added extensions from STD10.
14	AD09 - Insurance Claim Automation	Missing extensions from	Added extensions from

		STD14.	STD14.

2.2 Class Diagram (Initial)



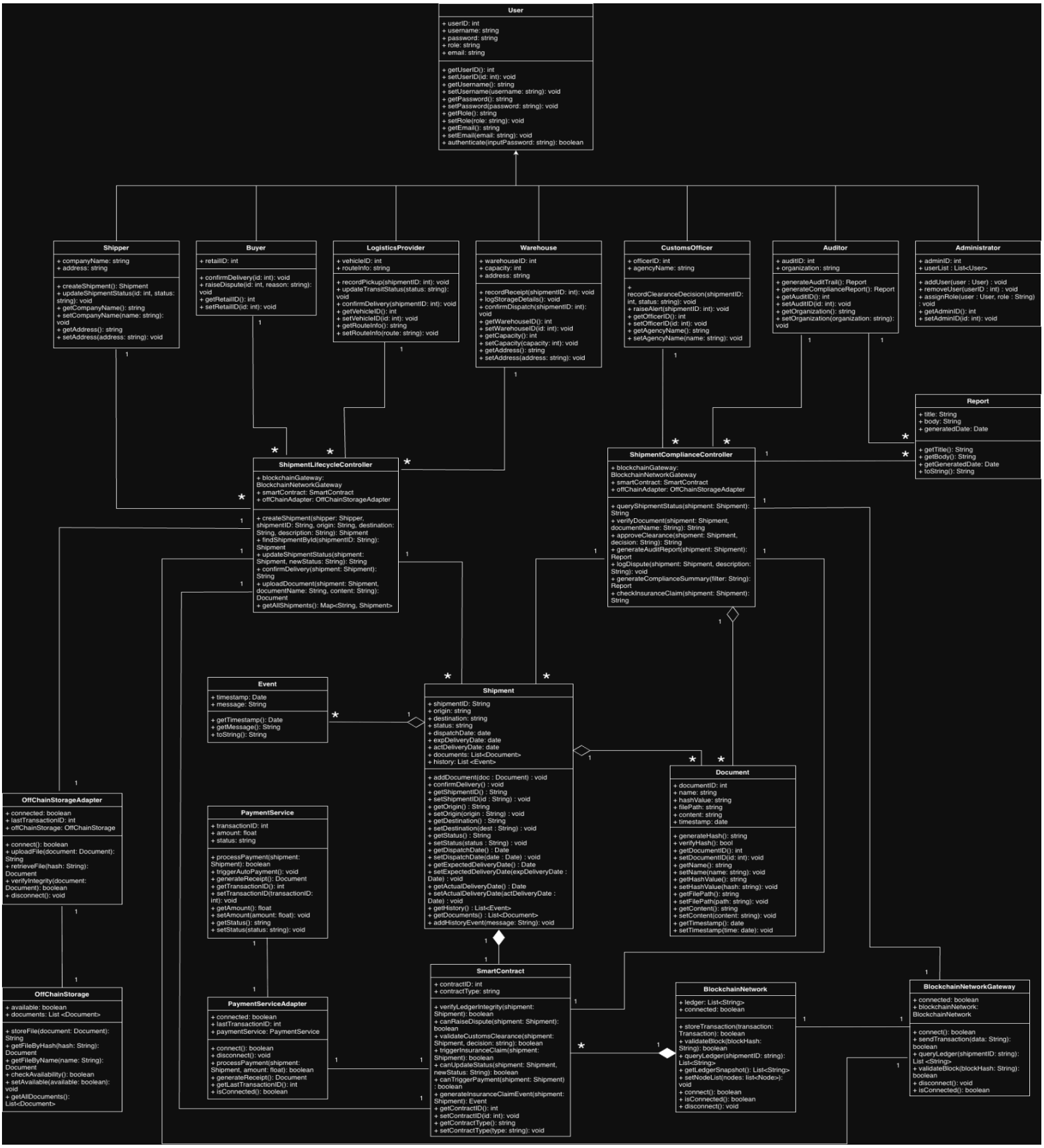
2.3 Discuss and Apply selected Design Pattern(s)

Context, Problem and Solution:

In our blockchain shipment tracking system, different users like shippers, buyers, and customs officers interact with the platform through the user interface to perform tasks such as creating shipments, updating delivery status, or confirming delivery. Each of these actions requires coordination between the interface, the blockchain, and the off-chain storage components. If the user interface directly managed all these interactions, the code would become tightly coupled and harder to maintain. The UI would handle both business logic and communication with the blockchain, making it difficult to update or scale the system when new features or user roles are added. The **Controller pattern** solves this by introducing a dedicated class (e.g., **`ShipmentLifecycleController`**, **`ShipmentComplianceController`**) that serves as an intermediary between the UI and the system logic. The controller receives user input, coordinates with the smart contracts, and manages responses. This keeps the user interface simple while maintaining low coupling and high cohesion. It also centralizes key operations like validation and transaction handling, making the system easier to extend and maintain over time.

Our system communicates with several external components such as the blockchain network, off-chain document storage, and payment services. These technologies are often updated or replaced, and directly connecting our core logic to them would make the system fragile and difficult to adapt to changes. Direct connections between core modules and external systems create dependencies that are hard to manage. If one external service changes its protocol or API, multiple parts of the system must be rewritten, increasing maintenance time and risk of errors. The **Indirection pattern** introduces intermediary classes like **`BlockchainGateway`**, **`OffChainStorageAdapter`**, and **`PaymentServiceAdapter`** to handle communication between the core system and external technologies. This approach decouples the core logic from specific implementations, allowing easier upgrades and testing. It ensures flexibility, maintainability, and scalability, which are essential for a blockchain-based system where components like storage or node configurations can evolve quickly.

2.4 Class Diagram (Updated)

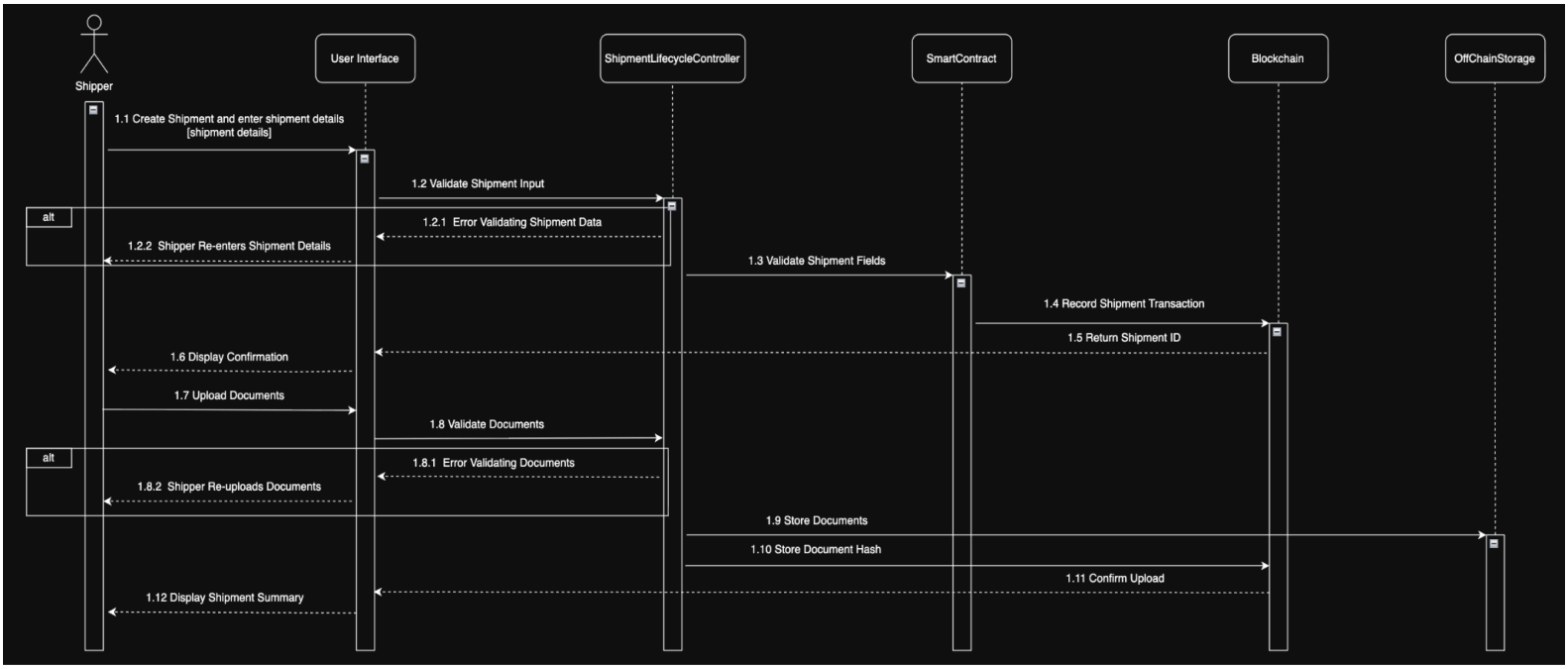


The updated class diagram integrates the **Controller** and **Indirection** design patterns to improve modularity and reduce coupling. **`ShipmentLifecycleController`** and **`ShipmentComplianceController`** were both added to separate workflow logic from the **`Shipment`** class. **`ShipmentLifecycleController`** handles shipment creation, status updates, and delivery confirmations, while the **`ShipmentComplianceController`** manages audits, customs clearance, and insurance claims. This ensures the UI interacts only with the controllers, which coordinate with other system components, keeping the architecture cohesive and maintainable.

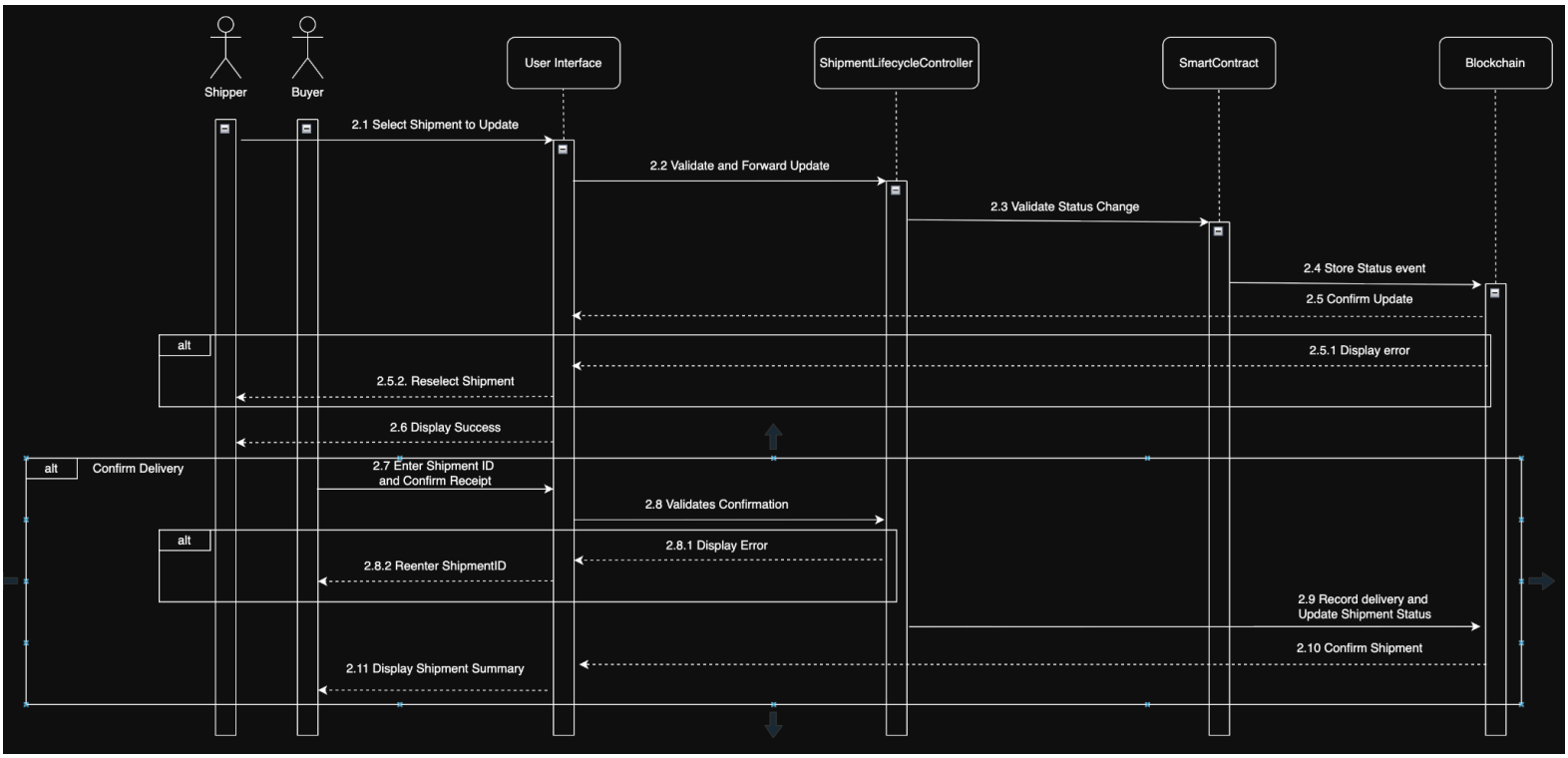
To separate the system from external dependencies, the **Indirection** pattern was applied by adding **`BlockchainGateway`**, **`OffChainStorageAdapter`**, and **`PaymentServiceAdapter`** classes. They are intermediary classes between the core system and external technologies, such as the blockchain network, document storage, and payment services. An **`OffChainStorage`** class was also added to represent the external document repository accessed via the adapter. These changes simplify future updates, isolate external dependencies, and make the system more scalable and adaptable to technological advancements.

2.5 Sequence Diagrams (SD)

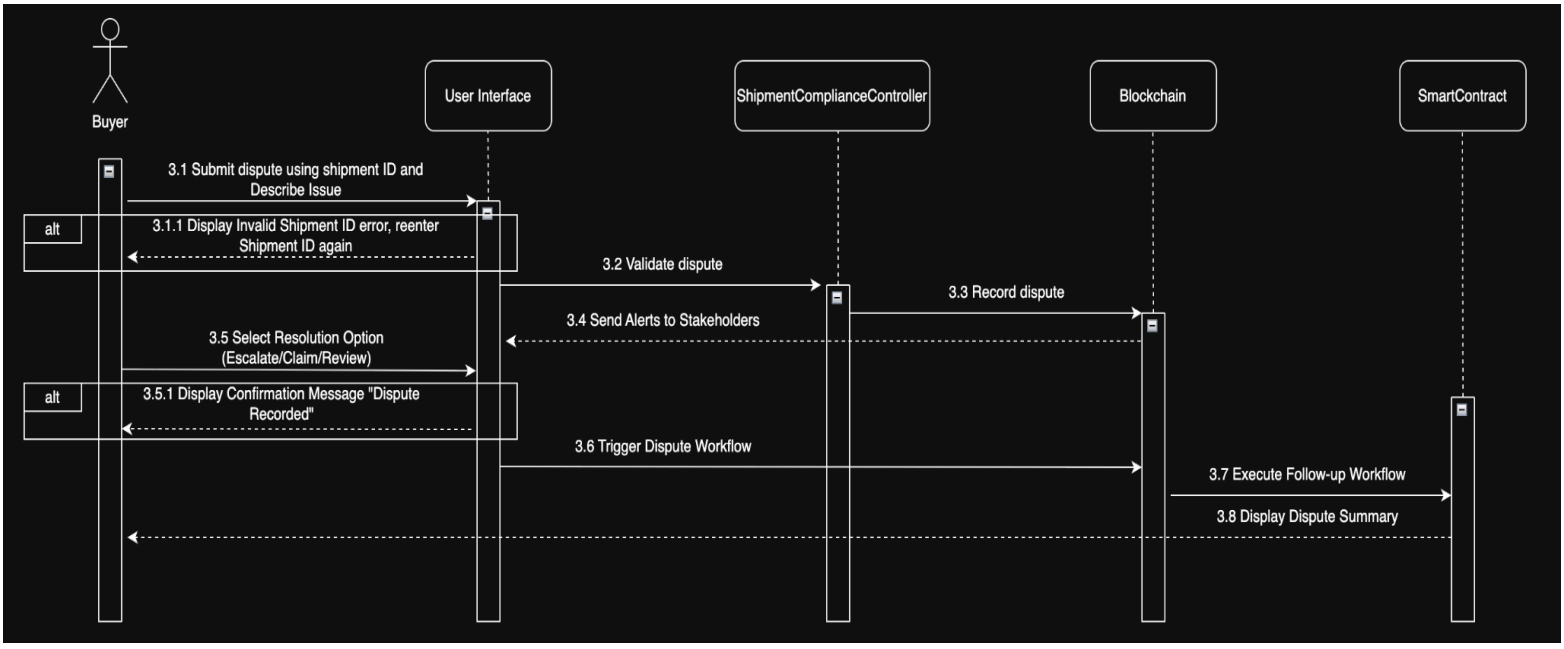
SD01 - Shipment Creating and Document Upload



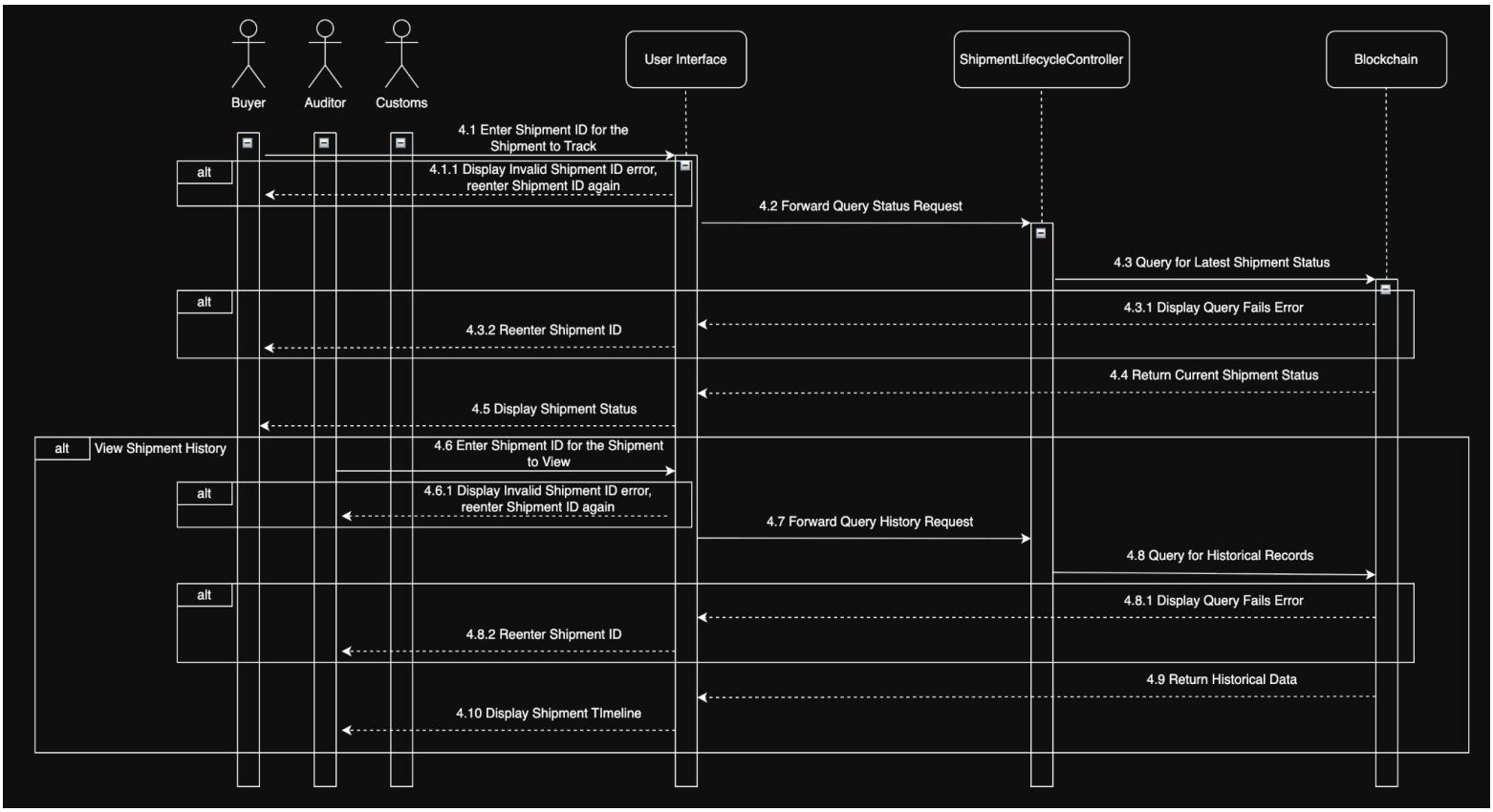
SD02 - Shipment Status Update and Delivery Confirmation



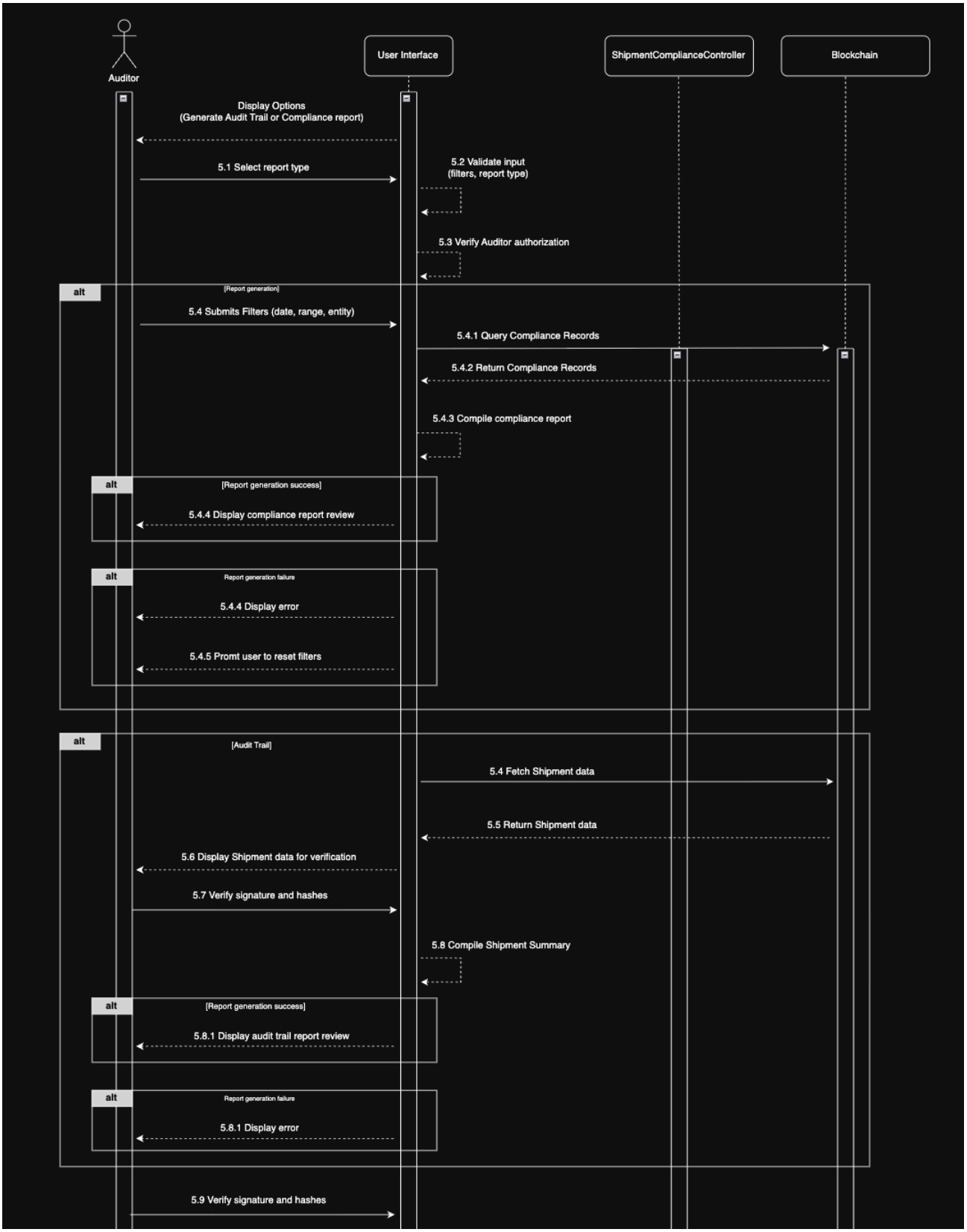
SD03 - Dispute Handling Workflow



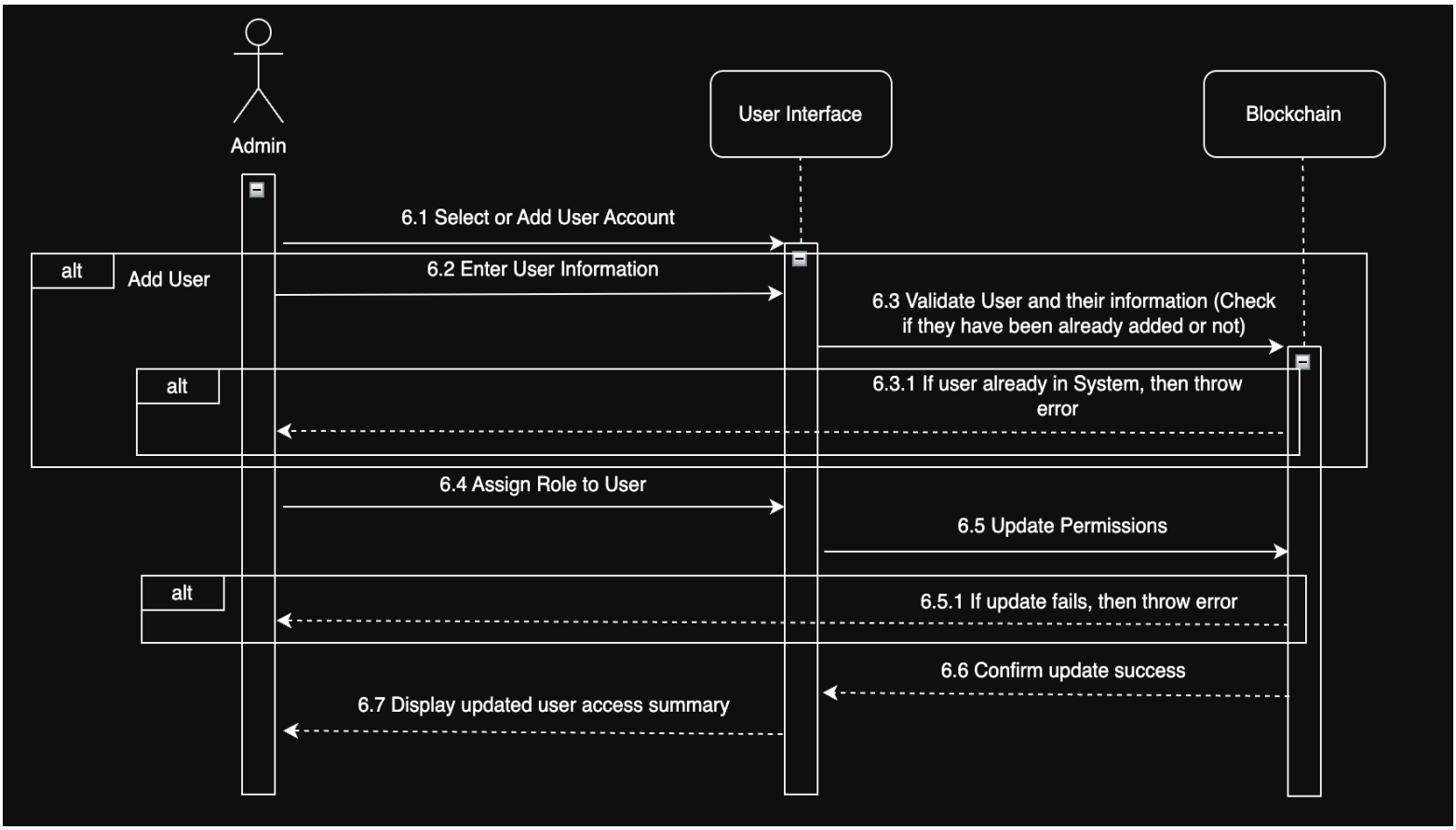
SD04 - Shipment Tracking and Review



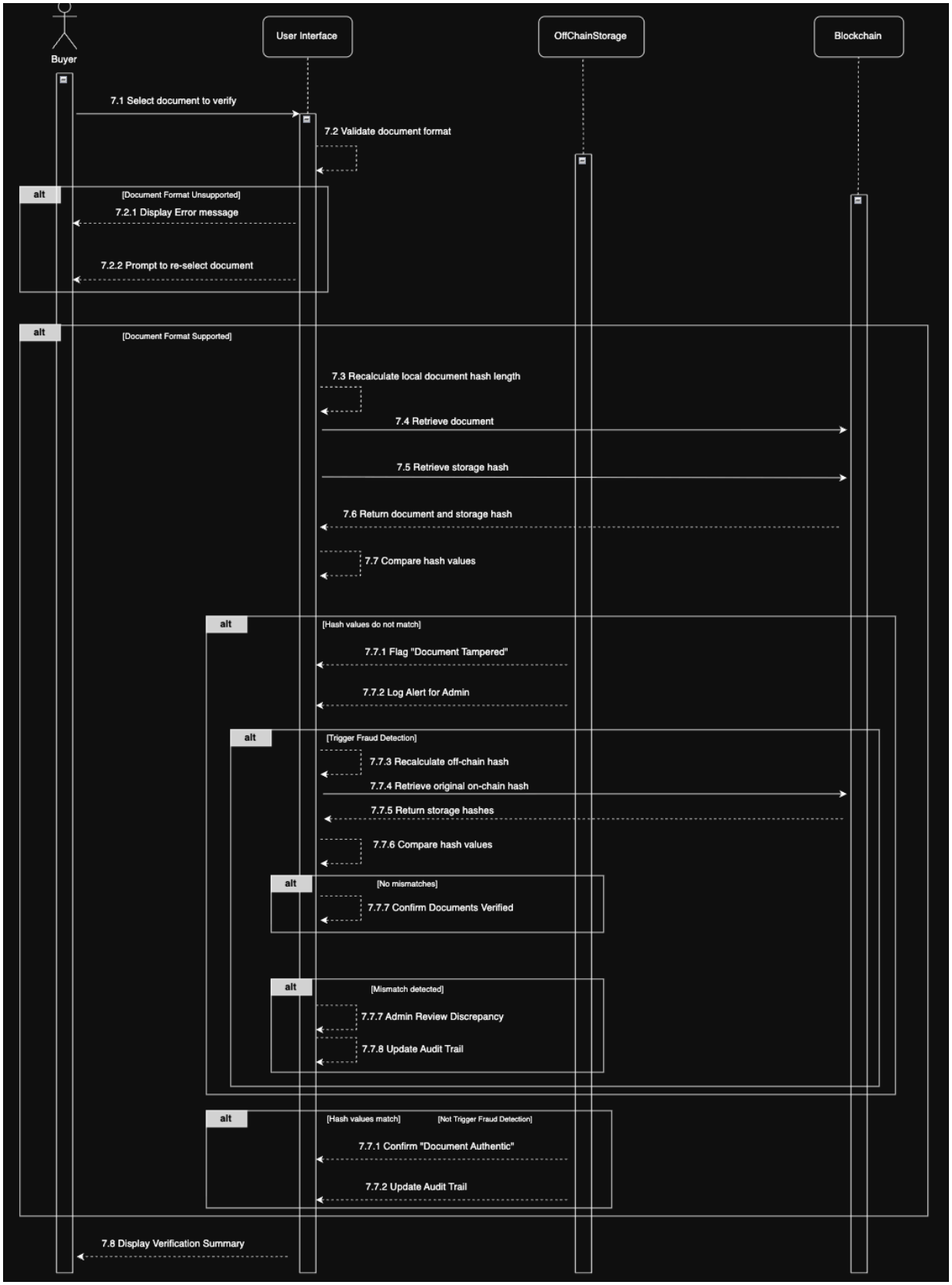
SD05 - Compliance and Audit Reporting



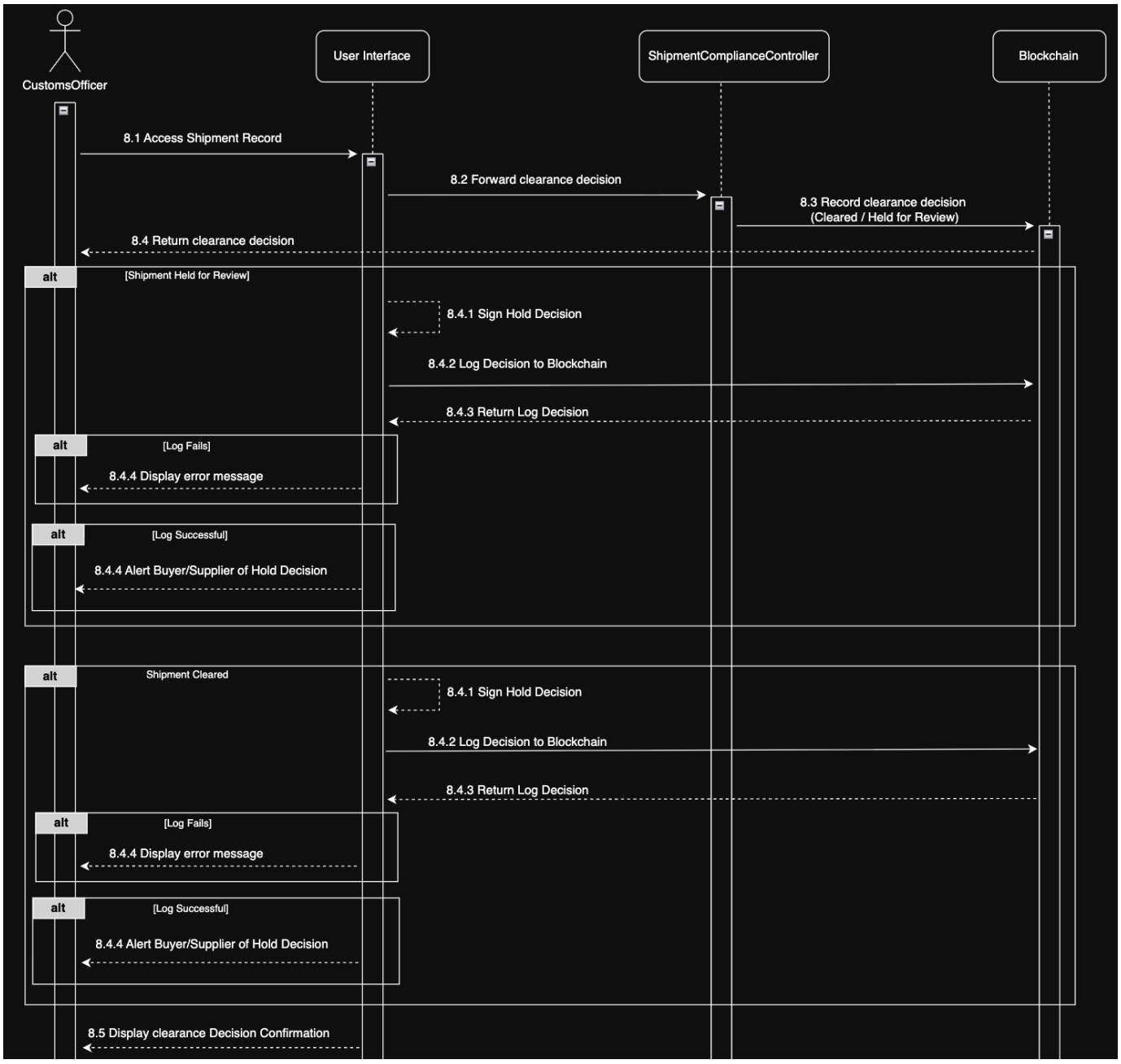
SD06 - User Management and Roles



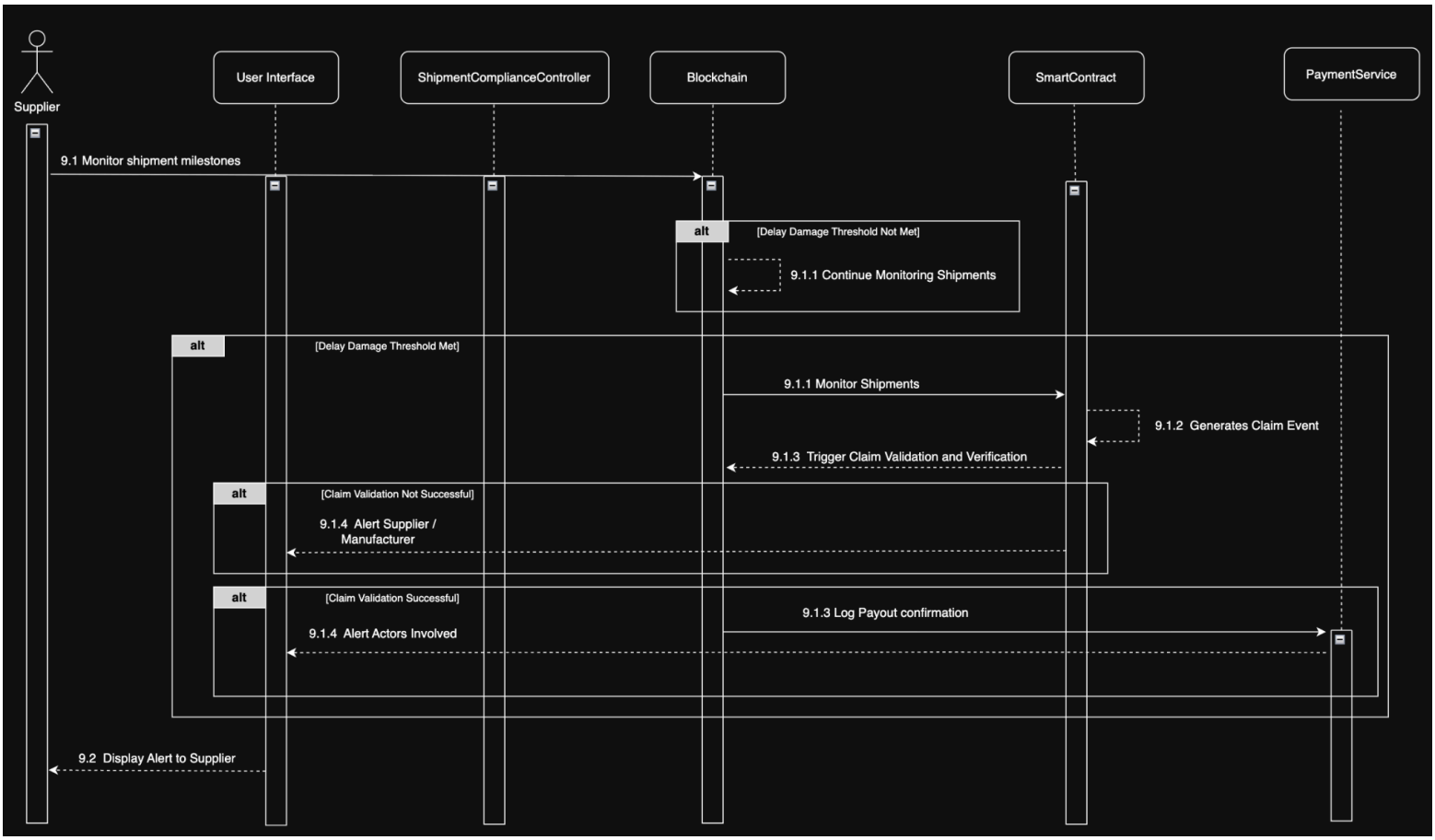
SD07 - Document Verification and Fraud Detection



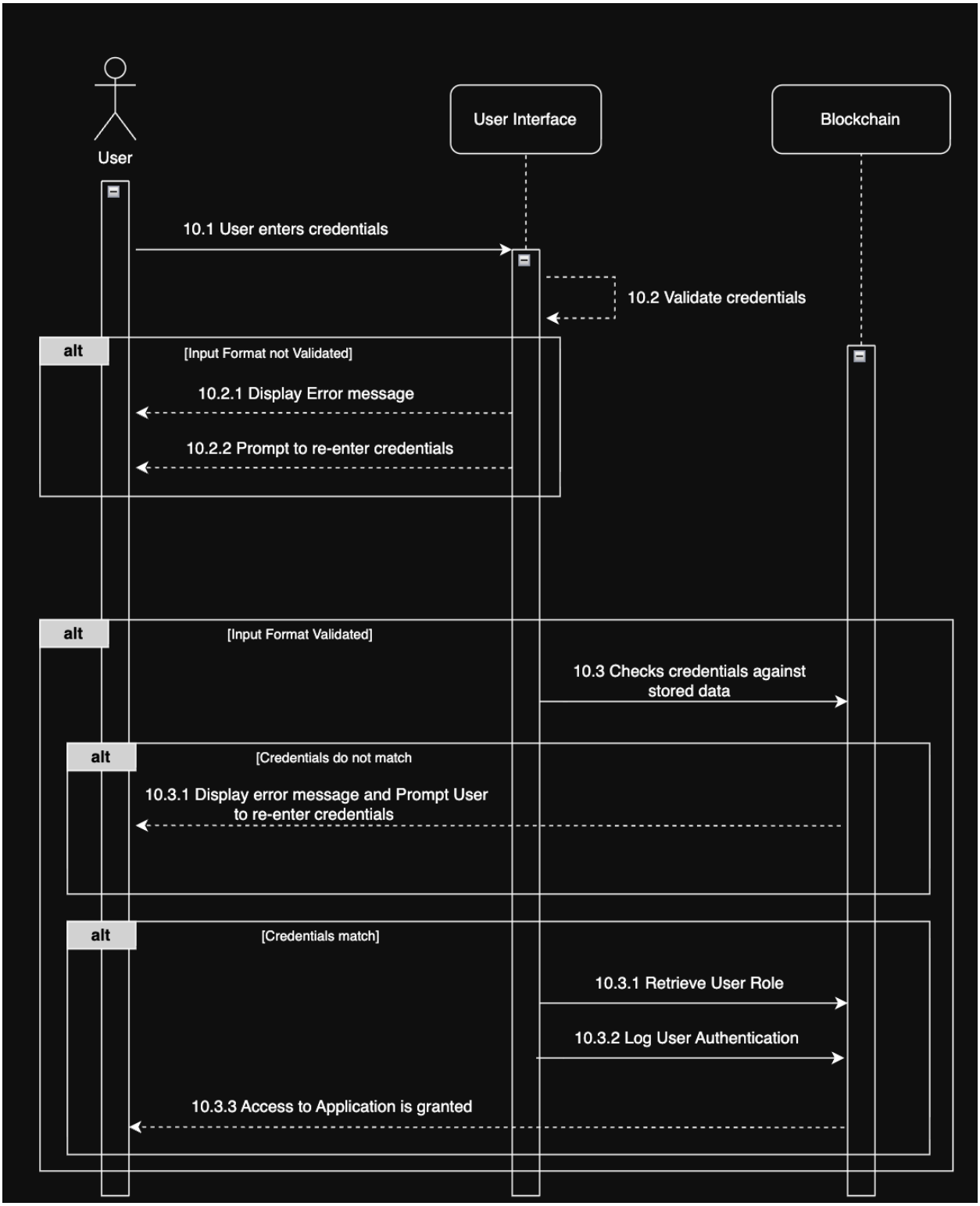
SD08 - Customs Clearance Approval



SD09 - Insurance Claim Automation



SD10 - User Authentication



3. Phase III & IV: System Implementation & Testing

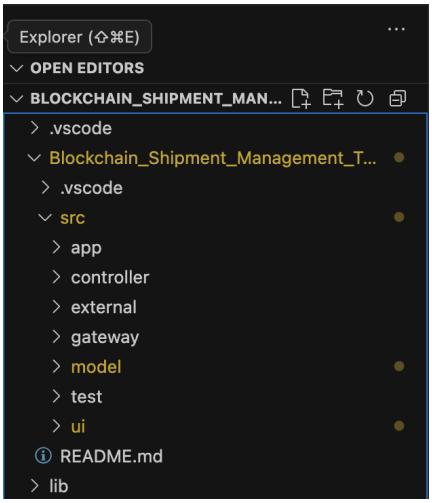
3.1 Overview of Implementation

Our system is fully implemented in **Java**, using a combination of **Swing** for graphical user interface development and custom modules to simulate core blockchain behaviours. The backend integrates a simplified blockchain ledger, off-chain storage, and an automated payment service through a collection of gateways and adapters that follow our design patterns. This architecture allowed us to separate external dependencies from our main logic and ensured clean interactions between uI components, controllers, and lower-level system services.

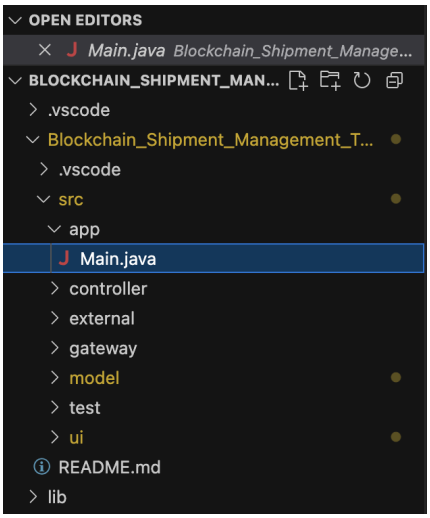
The core modules implemented include the domain model layer (`User, Shipment, Document, Event, SmartContract,` etc.), the controller layer (`ShipmentLifecycleController` and `ShipmentComplianceController`), and the adapter/gateway layer (`BlockchainNetworkGateway, OffChainStorageAdapter, PaymentServiceAdapter`). Each module was developed to map directly to the responsibilities outlined in our class diagram. The controllers act as the central coordinators, enforcing SmartContract rules, updating shipment states, generating logs, triggering insurance claims, and releasing payments. The architecture aligns closely with our design models, ensuring that each class in the implementation corresponds to a defined component in our UML diagrams and that interactions between modules follow the relationships and multiplicities originally specified.

The following steps represent the process of running/visualizing the Blockchain Shipment Management Tracking System:

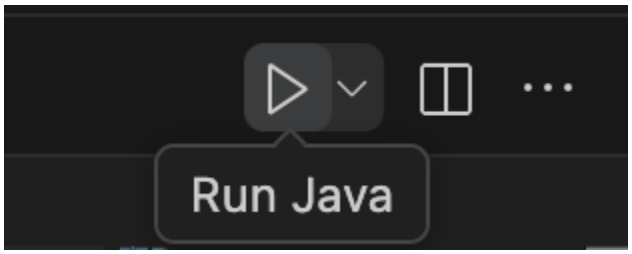
1. Open folder containing all implemented files in your preferred IDE. The screenshot below is what should be visible on the left panel after opening the folder.



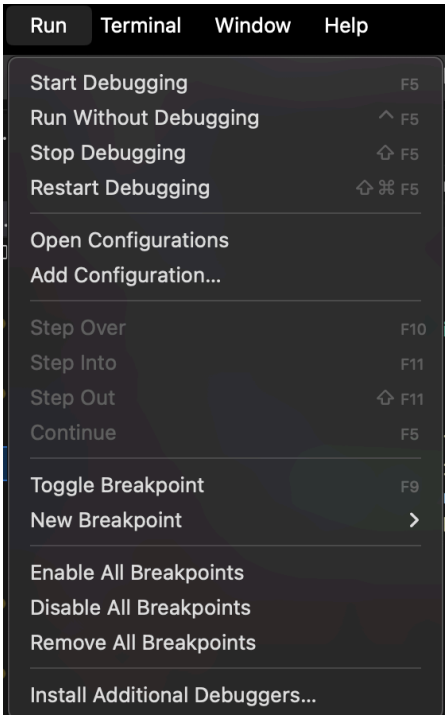
2. Navigate to the `src/app` folder and open `Main.java` to open it.



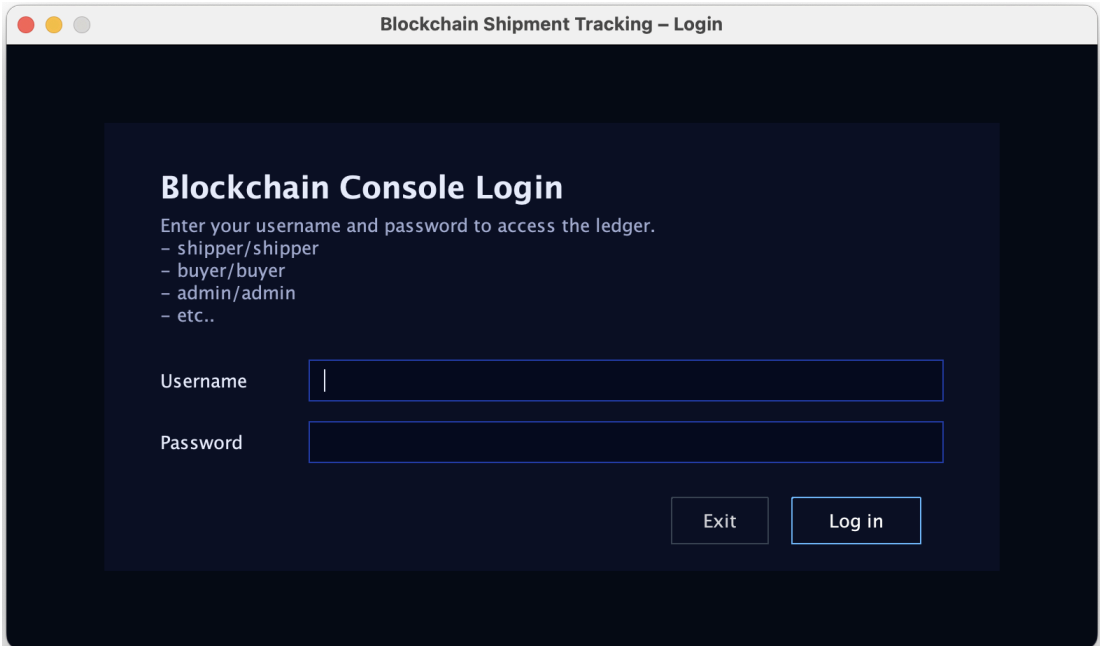
3. Once `Main.java` is open, locate the run button in the top right corner of the file.



   a. Alternatively, locate the "*Run*" tab at the top banner and select "*Run Without Debugging*".



4. The program will open in a new window and the user should have visibility on the application (as follows):



3.2 Code Structure Summary

Below is a description of the main packages in our implementation. This provides a clear mapping between the system design and the actual codebase.

Package	Description
model	Contains all domain entity classes such as `User` (along with its subclasses like `Shipper` and `Buyer`), `Shipment`, `Document`, `Event`, `SmartContract`, and related business objects used throughout the system.
controller	Includes the core controllers (`ShipmentLifecycleController` and `ShipmentComplianceController`) responsible for coordinating user actions, enforcing SmartContract rules, updating shipment states, and interacting with adapters/gateways.
gateway	Holds all adapter/gateway classes that interface with external systems, including `BlockchainNetworkGateway`, `OffChainStorageAdapter`, and `PaymentServiceAdapter`, ensuring separation between controllers and external services.
external	Simulated external system modules such as `BlockchainNetwork`, `OffChainStorage`, and `PaymentService`. These represent third-party or infrastructure components outside the main application logic.
ui	Contains the Java Swing interface: `MainUI` and `LoginFrame`. `LoginFrame` handles login and user authentication, while `MainUI` contains the full Swing interface with all system screens and role-based menus. All UI actions delegate to controller

	classes, keeping the UI free of business logic.
test	Contains unit test classes for validating core functionality in controllers, model classes, and adapters/gateways. This package includes **JUnit** test cases that verify correct behaviour, edge conditions, and integration of system modules.

3.3 Implementation of System Objects

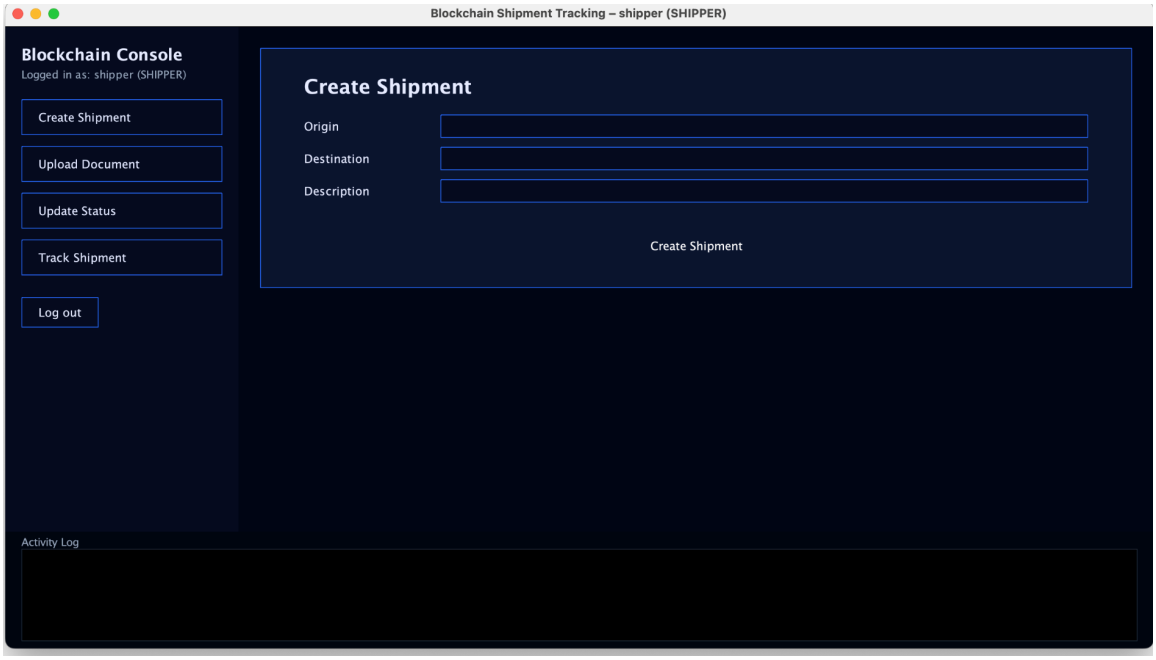
The system objects were implemented according to the design models established in Phase ll, ensuring that each class fulfills the responsibilities defined in the class diagram. As mentioned in the Code Structure Summary, the **model layer** (model package) contains the core domain entities, with each entity encapsulating state and business logic relevant to the shipment lifecycle. Object interactions remain consistent with the UML: shipments aggregate documents and events, controllers mediate between the UI and domain objects, and `SmartContract` enforces rules such as valid status transitions, delivery confirmation permissions, payment processing eligibility, and insurance claim detection.

The **controller layer** (controller package) coordinates all major system operations. The `ShipmentLifecycleController` manages creation, tracking, status updates, document uploads, and delivery confirmation. The `ShipmentComplianceController` handles audit trail generation, customs clearance, dispute logging, status verification, fraud detection, and insurance claim checks. Each controller interacts with the blockchain, off-chain storage, and payment service through their respective gateways/adapters, maintaining loose coupling and clean separation of concerns.

The **gateway layer** (gateway package) provides abstraction between the application and simulated external systems. The `BlockchainNetworkGateway` sends immutable state updates to the blockchain, the `OffChainStorageAdapter` handles document storage and hashing, and the `PaymentServiceAdapter` manages secure payment processing and receipt generation. These adapters ensure that external system changes do not impact core business logic. Overall, the implementation closely aligns with the system's architectural design, resulting in a modular, maintainable, and extensible system.

3.4 Implementation of System UIs

The system's User Interface is implemented in **Java Swing** and is divided into two main classes: **`LoginFrame`** and **`MainUI`**. `LoginFrame` handles user login, user authentication, and initializes all backend components. Once a user logs in, MainUI provides the full application interface using a CardLayout, displaying different screens depending on the user's role (e.g. Create Shipment for Shipper, Confirm Delivery for Buyer, Compliance Reports for Auditors). The menu displayed on the left dynamically changes depending on the logged-in user's role, ensuring that each actor only sees the functions they are authorized to perform. For example, below is what the menu looks like for a user with the Shipper Role.



To keep the UI lightweight and maintainable, the **Controller pattern** is applied. `MainUI` does not perform business logic. Instead, UI actions are sent to the `ShipmentLifecycleController` or `ShipmentComplianceController`. These controllers handle validation, smart contract rules, and interaction with the blockchain and offchain storage.

Communication with external components, such as the blockchain network, payment service, and offchain storage, are separated through the **indirection pattern** using gateway/adapter classes (as mentioned in the code structure summary). This ensures MainUI interacts only with controllers and never with external systems directly.

Overall, the UI provides a role-based, organized interface while maintaining low coupling, high cohesion, and strong alignment with the architecture and design patterns used throughout the system.

3.5 Test Plan & Test Cases with Sample Results

To ensure that our blockchain-based shipment management system works reliably from end to end, we created a series of unit tests that validate everything from basic blockchain connectivity to the business rules inside our smart contract. These five test classes collectively check the core layers of the system:

1. The Blockchain Gateway → `BlockchainNetworkGatewayTest.java`
2. The Shipment Lifecycle → `ShipmentLifeCycleControllerTest.java`
3. Compliance Processes → `ShipmentComplianceControllerTest.java`
4. Data Models → `ShipmentTest.java`
5. SmartContract Logic → `SmartContractTest.java`

making sure that each part behaves correctly on its own before being integrated into the full application. Together, they help confirm that our tracking system is secure, consistent, and capable of supporting transparent supply-chain operations on a blockchain. Below is a breakdown of each of these classes along with their functionality and significance. As mentioned in code structure summary, these test classes can be found under the **test package** in our code.

`------ BlockchainNetworkGatewayTest.java ------`

This test class checks whether the `BlockchainNetworkGateway`, the system's bridge between controller logic and the underlying blockchain, behaves the way it is supposed to. It validates the basic connectivity workflow (connect/disconnect), ensures that transactions can only be submitted if the gateway is connected, and confirms that ledger queries and block validation run safely without breaking the system. This is important because the gateway is responsible for enforcing safe communication with the blockchain layer. If it fails, every higher-level controller (shipping lifecycle, compliance, audits) would behave inconsistently. These tests guarantee that the "entry point to the blockchain" is robust and predictable.

Below are the test cases along with their results:

Test Case	Purpose	Expected Result	Actual Result
testConnectAndDisconnect	Ensure gateway toggles connection state correctly	connect() sets connected=true, and disconnect() sets false	PASS
testSendTransactionRequiresConnect	Validate that transactions only send when connected	Sending fails when disconnected, succeeds after connect()	PASS
testQueryLedger	Ensure ledger queries return a safe, non-null list	Empty list when disconnected; non-null list after connect + send	PASS
testValidateBlockRequiresConnect	Validate block only when connected	Returns false when disconnected; runs without errors when connected	PASS

`------ ShipmentLifeCycleControllerTest.java ------`

This test class focuses on the shipment operational workflow, including creating shipments, retrieving them by ID, updating statuses, and uploading off-chain documents. It checks that lifecycle operations integrate correctly with the blockchain gateway, off-chain storage, and smart contract rules. This is significant because the lifecycle controller represents the “everyday functionality” of the shipment tracking system, through which shippers directly interact with. Ensuring that lifecycle functions work smoothly guarantees that all shipments have a consistent flow from creation → transit → delivery and that documents (such as invoices or certificates) remain properly linked to shipments.

Below are the test cases along with their results:

Test Case	Purpose	Expected Result	Actual Result
testCreateShipment	Create shipment with correct fields and initialization	Shipment object created with correct ID/origin/destination	PASS
testFindShipmentById	Retrieve shipment from in-memory registry	Returns correct shipment instance	PASS
testUpdateShipmentStatus	Ensure status update follows smart contract rules	Status changes & response message contains "updated"	PASS
testUploadDocument	Verify a document attaches to shipment	Document count increases; name matches uploaded	PASS

`------ ShipmentComplianceControllerTest.java ------`

This test class tests the shipment compliance features, which include querying shipment status, generating audit trails, and logging disputes. The tests simulate a shipper creating a shipment and then verify that the compliance controller reflects updates and produces meaningful audit reports accurately. The compliance layer is the part of the blockchain shipment management tracking system that interacts with the blockchain for verification, auditability, and dispute handling. This highlights the significant use cases of the system and further reveals why blockchain is useful in supply-chain management. These tests make sure that the compliance logic produces transparent and traceable outputs, aligning with the trust and accountability requirements of the major actors involved.

Below are the test cases along with their results:

Test Case	Purpose	Expected Result	Actual Result
testQueryShipmentStatus	Confirm controller returns correct shipment status string	Output includes shipment ID + current status	PASS
testGenerateAuditTrail	Validate audit trail creation with correct content	Report contains “Audit trail” + updated status	PASS
testLogDispute	Ensure disputes are recorded properly	Returns "Dispute filed..." message containing shipment ID	PASS

`------ ShipmentTest.java ------`

This test class tests the Shipment model itself, making sure that documents and events can be added and that status updates behave correctly. These tests validate that the shipment object maintains accurate history, updates its state consistently, and stores metadata (like documents) without errors. This is significant since the shipment model serves as the application's fundamental data structure. If the model behaves unpredictably, every controller and smart contract would also behave incorrectly, resulting in inaccurate results. These tests ensure that each shipment/transaction is recorded accurately, is reliable, and is fully traceable.

Below are the test cases along with their results:

Test Case	Purpose	Expected Result	Actual Result
testAddDocument	Confirm documents are stored in shipment	Document list size increases to 1	PASS
testAddEvent	Ensure events append correctly to history	History size increases by 1	PASS
testStatusUpdates	Verify status mutator works properly	Status updates to "IN_TRANSIT"	PASS

`------ SmartContractTest.java ------`

This test class verifies all the business rules enforced by the linked `SmartContract.java` file, such as valid status transitions, dispute triggers, payment eligibility, insurance claims, and customs clearance checks. It ensures that once a shipment is marked as DELIVERED, no illegal transitions occur, and that special events (like disputes or customs actions) abide by the rules as per the application's protocol. This is significant because the SmartContract is the core logic that brings blockchain value into this shipment management tracking system. It enforces trust, prevents tampering, and encodes process rules. By testing it independently, it is proven that the system's logic is consistent and that shipments follow secure, rule-based workflows across their lifecycle.

Below are the test cases along with their results:

Test Case	Purpose	Expected Result	Actual Result
canUpdateStatus_allowsTransitionWhenNotDelivered	Verify allowed transitions before delivery	Returns true	PASS
canUpdateStatus_blocksChangeAfterDelivered	Prevent illegal transitions after delivery	Delivered → other status is blocked	PASS
canTriggerPayment_onlyWhenDelivered	Payment only allowed after delivery	Returns false before delivery, true after	PASS
verifyLedgerIntegrity_emptyHistoryIsValid	Empty history should pass integrity check	Returns true	PASS
canRaiseDispute_onlyWhenNotDelivered	Disputes only allowed before delivery	True before delivery, false after	PASS
validateCustomsClearance_approveFromCreatedIsAllowed	Approve is allowed from CREATED	Returns true	PASS
validateCustomsClearance_notAllowedOnceDelivered	Clearance cannot occur after delivery	Returns false	PASS
triggerInsuranceClaim_whenDamaged_returnsTrue	Damage should trigger claim	Returns true	PASS


OUTPUT
DEBUG CONSOLE
TEST RESULTS
TERMINAL
PORTS
GITLENS
martContractTest]/[method:canTriggerPayment_onlyWhenDelivered()]
%TSTTREE7,canUpdateStatus_allowsTransitionWhenNotDelivered(test.SmartContractTest),false,1,false,2,canUpdateStatus_allowsTransitionWhenNotDelivered(),,[engine:ju
nit-jupiter]/[class:test.SmartContractTest]/[method:canUpdateStatus_allowsTransitionWhenNotDelivered()]
%TSTTREE8,canUpdateStatus_blocksChangeAfterDelivered(test.SmartContractTest),false,1,false,2,canUpdateStatus_blocksChangeAfterDelivered(),,[engine:junit-jupiter
/[class:test.SmartContractTest]/[method:canUpdateStatus_blocksChangeAfterDelivered()]
%TSTTREE9,verifyLedgerIntegrity_emptyHistoryIsValid(test.SmartContractTest),false,1,false,2,verifyLedgerIntegrity_emptyHistoryIsValid(),,[engine:junit-jupiter]/[
class:test.SmartContractTest]/[method:verifyLedgerIntegrity_emptyHistoryIsValid()]
%TSTTREE10,validateCustomsClearance_notAllowedOnceDelivered(test.SmartContractTest),false,1,false,2,validateCustomsClearance_notAllowedOnceDelivered(),,[engine:j
unit-jupiter]/[class:test.SmartContractTest]/[method:validateCustomsClearance_notAllowedOnceDelivered()]
%TESTS 3,validateCustomsClearance_approveFromCreatedIsAllowed(test.SmartContractTest)
%TESTE 3,validateCustomsClearance_approveFromCreatedIsAllowed(test.SmartContractTest)
%TESTS 4,canRaiseDispute_onlyWhenNotDelivered(test.SmartContractTest)
%TESTE 4,canRaiseDispute_onlyWhenNotDelivered(test.SmartContractTest)
%TESTS 5,triggerInsuranceClaim_whenDamaged_returnsTrue(test.SmartContractTest)
%TESTE 5,triggerInsuranceClaim_whenDamaged_returnsTrue(test.SmartContractTest)
%TESTS 6,canTriggerPayment_onlyWhenDelivered(test.SmartContractTest)
%TESTE 6,canTriggerPayment_onlyWhenDelivered(test.SmartContractTest)
%TESTS 7,canUpdateStatus_allowsTransitionWhenNotDelivered(test.SmartContractTest)
%TESTE 7,canUpdateStatus_allowsTransitionWhenNotDelivered(test.SmartContractTest)
%TESTS 8,canUpdateStatus_blocksChangeAfterDelivered(test.SmartContractTest)
%TESTE 8,canUpdateStatus_blocksChangeAfterDelivered(test.SmartContractTest)
%TESTS 9,verifyLedgerIntegrity_emptyHistoryIsValid(test.SmartContractTest)
%TESTE 9,verifyLedgerIntegrity_emptyHistoryIsValid(test.SmartContractTest)
%TESTS 10,validateCustomsClearance_notAllowedOnceDelivered(test.SmartContractTest)
%TESTE 10,validateCustomsClearance_notAllowedOnceDelivered(test.SmartContractTest)
%RUNTIME49
Test Runner for Java
SmartContractTest $(symbol-namespace) test ‹ $(project) Block
canRaiseDispute_onlyWhenNotDelivered() $(symbol-class) Sm
canTriggerPayment_onlyWhenDelivered() $(symbol-class) Sm
canUpdateStatus_allowsTransitionWhenNotDelivered() $(sym
canUpdateStatus_blocksChangeAfterDelivered() $(symbol-cl
triggerInsuranceClaim_whenDamaged_returnsTrue() $(symb
validateCustomsClearance_approveFromCreatedIsAllowed()
validateCustomsClearance_notAllowedOnceDelivered() $(sy
verifyLedgerIntegrity_emptyHistoryIsValid() $(symbol-class) S
23 older results