Capstone Project - Supply Chain Tracker

Sarai	Deeplang	6322770569
Napaphat	Sittijanyawan	6322772656
Suthisa	Panto	6322773985

DES484 Blockchain Development

Section 1

Dr. Watthanasak Jeamwatthanachai

Sirindhorn International Institute of Technology

Thammasat University

December 18, 2023

Table of Content

Background

What is Decentralized application?

Decentralized applications (DApps) revolutionize computing through peer-to-peer networks, often integrated with blockchain. They operate on distributed trust, storing data across network nodes rather than a central server. This eradicates central authority, granting users control and transparency via blockchain-based smart contracts. Consequently, DApps stand resilient against censorship, ensuring heightened security and bypassing intermediaries, yet struggling with scalability and complexity challenges. Nevertheless, their potential to reshape sectors like finance remains profound. As blockchain technology progresses, DApps are anticipated to evolve into more user-friendly interfaces, contributing to a secure and transparent online environment. This evolution holds promise for a future where DApps play a pivotal role in transforming digital interactions across various industries.

What is Smart Contract?

A smart contract, a cornerstone of blockchain technology, represents a self-executing agreement embedded within a distributed ledger. Functioning autonomously, it meticulously executes predefined conditions without the involvement of intermediaries, operating within a trustless framework. This paradigm leverages the robust security mechanisms inherent in blockchain, ensuring transactions occur devoid of centralized control, thus fortifying the system's resilience.

The essence of a smart contract lies in its transparency and immutability. Its code, openly accessible on the blockchain, lays bare the operational framework, cultivating trust among involved parties. By deploying these contracts, sectors spanning finance, supply chains, and even voting systems benefit from their adaptable nature. Contrary to being confined to a singular blockchain infrastructure like Ethereum, smart contracts seamlessly integrate with diverse blockchain platforms, showcasing their versatility and compatibility.

Supply Chain Management Domain

Case Study and Issue

Supply chain management (SCM) encompasses the intricate network of activities involved in bringing a product from its raw material state to the end consumer. This process involves a multitude of stakeholders, including manufacturers, distributors, logistics providers, and retailers, all collaborating to ensure smooth and efficient delivery. However, the complexity of these details can make the supply chain susceptible to disruptions and inefficiencies, resulting in various challenges:

1. Lack of transparency: Insufficient visibility in the flow of information can lead to delays, errors, and difficulties in identifying responsible parties for issues.
2. Counterfeiting and fraud: Unscrupulous actors can infiltrate the supply chain, introducing fake or tampered products, raising safety and ethical concerns.
3. Inefficiencies and waste: Poor visibility into inventory levels and logistics can result in overstocking, understocking, and inefficient transportation, thereby increasing costs and environmental impact.

These challenges underscore the necessity for robust and transparent systems to track and manage the movement of goods throughout the supply chain. This is where blockchain technology emerges as a promising solution. With its distributed ledger technology and cryptographic security, blockchain offers a unique set of features to address the challenges of traditional supply chain management including:

1. Transparency and immutability: All transactions and movements within the supply chain are recorded on a shared ledger accessible to authorized participants. This fosters trust and accountability, facilitating the tracking of the origin and journey of products.
2. Security and tamper-proofing: Data stored on the blockchain is encrypted and distributed across a network of nodes, making it highly resistant to manipulation or fraud. This ensures the integrity of information and protects against counterfeiting.
3. Efficiency and automation: Smart contracts can automate specific processes within the supply chain, such as payments and document verification, streamlining workflows and reducing manual errors.

A Real-World Example: Tracking Food from Farm to Fork

Consider the journey of a fresh-cut salad from farm to supermarket shelf. Traditionally, tracking its origin, freshness, and handling practices can be challenging. However, a blockchain-based supply chain tracker can record every step of the process including its origin, its distribution and transportation, its retailer, and its customer. For example,

1. Planting: The planting date, seed variety, and any fertilizers used can be recorded on the blockchain.
2. Harvesting and processing: The harvesting date, processing methods, and temperature controls can be documented, ensuring proper handling.
3. Distribution and transportation: Logistics details, including truck temperature and location data, can be tracked, guaranteeing freshness and safety.
4. Retailer and consumer: Upon arrival at the supermarket, the salad's origin and journey can be readily accessed by consumers through QR codes or smartphone apps, promoting transparency.

Backend Development

Key Requirements

SMART Goal for Supply Chain Management

SMART goals ensure that collaborators are on the same page by clarifying the vision and the ideas, allowing them to deploy resources productively, especially when working with numerous stakeholders with large supplier ecosystems has given us extensive insight into the value of setting these SMART goals with suppliers; they ensure that initiatives and projects are given initial focus at the outset and that they stay on track.

By adopting this technique means setting goals that are:

Specific: Develop and deploy a blockchain-based Supply Chain Tracker system that traces product movement from manufacturer to end-user.

Measurable: Track product through the entire supply chain lifecycle, recording key details like address, shipment status, and purchasing status.

Achievable: Engage with key stakeholders, including manufacturers, retailers and more, to integrate the tracking system. Utilize existing blockchain frameworks and establish data-sharing protocols.

Relevant: Improve supply chain visibility, minimize delays, and enhance inventory management. Address industry demands for transparency, compliance, and customer satisfaction.

Time-bound: Complete system development, integration, and testing, followed by a decent pilot phase for real-time tracking and validation.

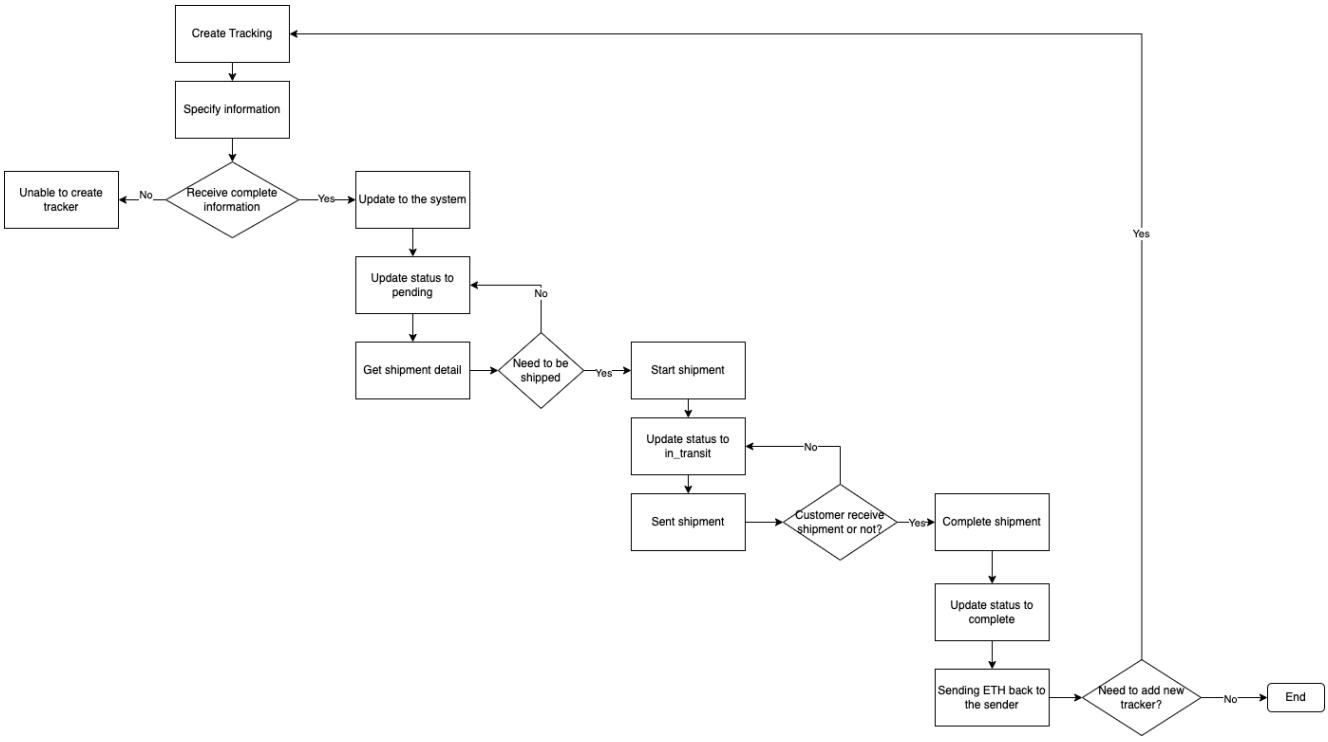
Architecture

The architecture for a shipment tracking smart contract on the Ethereum blockchain. begins by defining the core elements: the Shipment struct encapsulates essential shipment details such as sender, receiver, pickup and delivery times, distance, price, status, and payment status.

The contract utilizes an enum type, ShipmentStatus, to enumerate the various stages a shipment can undergo: PENDING, IN_TRANSIT, and DELIVERED. The use of mappings enables efficient storage and retrieval of shipment data, with shipments grouped by sender addresses. Additionally, the contract implements key functionalities such as creating, initiating, completing shipments, and updating delivery times, all triggered externally through specified functions. Events, like ShipmentCreated and ShipmentDelivered, provide external visibility and transparency into the contract's operations.

This architecture presents a structured framework delineating data organization, state transitions, and external interaction points essential for a functional and transparent shipment tracking system on the Ethereum blockchain.

The system's workflow is depicted through the presented flowchart, illustrating the sequential steps and interactions within the system.



Smart Contract Development Tools

1. **JavaScript (JS)**

   JavaScript(JS) is a high-level, interpreted programming language primarily used for front-end web development. It adds interactivity, dynamic content, and behavior to websites. JavaScript is crucial for interacting with smart contracts, enabling the sending of transactions and retrieving data from the blockchain. This facilitates the querying of information from the blockchain, offering real-time updates to users. Moreover, JavaScript also sets up event listeners to detect and react to user actions, such as button clicks or form submissions, and can handle events emitted by smart contracts on the blockchain. Libraries like Web3.js (for Ethereum) or ethers.js further enhance these interactions by providing tools and APIs.

2. **Node.js**

   Node.js is a runtime environment that allows the execution of JavaScript code on the server side. It is designed to be lightweight, efficient, and scalable, making it well-suited for building server-side applications, including web servers and networked software. Node.js plays several roles for DApp including acting as a backend server, handling server-side logic, data processing, and communication with databases.

   In our DApp, we utilized Node.js with the Tailwind CSS framework to streamline web application development. In addition, it facilitates interaction with the blockchain network, enabling transaction sending and smart contract state queries. Node.js can also establish event listeners responsive to specific smart contract-triggered events, allowing the DApp to dynamically respond to changes on the blockchain.

3. **Solidity**

   Solidity is a programming language designed for developing smart contracts on blockchain platforms, notably Ethereum. Its key roles in smart contract development include defining contract logic, managing state variables, handling data types and structures, supporting contract inheritance for modular code, incorporating events for signaling changes, enabling interaction with external contracts, and addressing security considerations. Solidity is fundamental for expressing the rules and functionalities of smart contracts, crucial for decentralized application development on the blockchain.

Smart Contract Development

The supply chain tracker smart contract is designed to serve as an efficient tracking system for shipments on the blockchain. It manages the shipment lifecycle through a series of functions catering to different stages of the process. The 'createShipment' function acts as the inception point, initializing a new shipment by capturing pivotal details like sender, receiver, pickup time, distance, and price. This function is instrumental in creating a record of the shipment and incrementing the overall shipment count, marking the commencement of the shipment journey.

Moving along the lifecycle, the 'startShipment' function plays a pivotal role by transitioning the status of the shipment to 'IN_TRANSIT.' Its activation is contingent on both the sender and receiver, signifying the commencement of the shipment's journey from a pending state to being actively in transit.

As the shipment reaches its destination, the 'completeShipment' function comes into play. It culminates the shipment process by marking it as 'DELIVERED' and meticulously recording the delivery time. Crucially, this function ensures the appropriate payment to the sender, preventing any duplication of payments for a single shipment.

Furthermore, the 'updateDeliveryTime' function offers flexibility by empowering the sender to modify the delivery time if the shipment hasn't yet reached the 'DELIVERED' status. This feature is crucial for maintaining accuracy in timestamps and preserving the integrity of the recorded data.

In conjunction with these critical functions, 'getShipment' and 'getShipmentsCount' serve as key retrieval mechanisms. 'getShipment' enables users to access specific details about a shipment by providing the sender's address and index, offering transparency regarding the shipment's various aspects. On the other hand, 'getShipmentsCount' provides an overview by fetching the total count of shipments associated with a specific sender's address, delivering insights into the volume of shipments initiated by that sender.

When combining them together, these functions facilitate comprehensive tracking and management of shipments on the blockchain, ensuring transparency and accuracy throughout the shipment process.

Frontend Development

Frontend Development Tools

1. **JavaScript (JS)**

   JavaScript (JS) is a high-level programming language recognized for its role in web development, bringing interactivity and dynamic behavior to websites. JavaScript is a versatile and widely adopted programming language, and its compatibility with web browsers, support for asynchronous programming, integration with Web3.js for blockchain interaction, and a rich ecosystem of tools and frameworks make it a strong choice for developing decentralized applications (DApps).

2. **React**

   React is a open-source frontend JavaScript library developed by Meta (formerly Facebook) that is widely used for building Single Page Applications (SPAs). It simplifies intricate application development through its component-based architecture and Virtual DOM optimization. Component-based approach promotes modular and reusable code, enhancing organization and maintainability. The Virtual DOM ensured efficient updates to the actual DOM, contributing to enhanced performance. React is a preferred frontend library with a large, supportive community and strengths like declarative syntax, flexibility, and JSX for readability. Its component-based, declarative, and flexible nature, along with efficient DOM management, makes it a top choice for developers creating sophisticated and maintainable applications. Continuous development and support from Meta further enhance its appeal.

3. **HTML**

   HTML, or HyperText Markup Language, is a fundamental language in web development. It structures the content of web pages using elements and tags. In the context of decentralized applications (DApps), HTML establishes the layout and user interface, encompassing headers, footers, navigation bar, and content areas. It incorporates form elements for user input, buttons, and hyperlinks for triggering actions. We utilized HTML with CSS for styling and JavaScript for dynamic behavior, contributing to a visually appealing and responsive DApp interface. And it supports embedding media elements, such as images to display content to the DApp. HTML's semantic elements enhance accessibility, making it a foundational tool in DApp frontend development.

4. **Next.js**

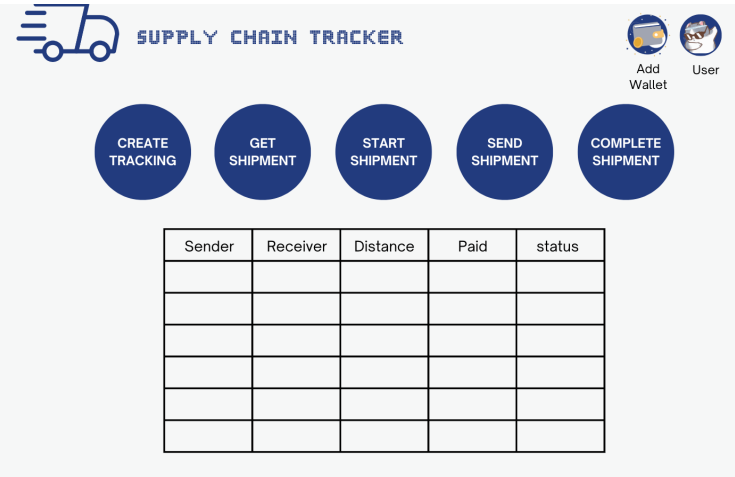
Next.js is a React framework for building web applications, offering server-side rendering, statistical site generation, and streamlined routing. Ideal for DApps, it ensures an efficient workflow, boosts performance, and integrates seamlessly with blockchain APIs. Next.js simplifies React tooling, automating bundling and compilation, allowing developers to focus on application development rather than configuration intricacies. Its comprehensive features contribute to a smoother development experience, and optimizes the overall application structure.

5. **Tailwind CSS**

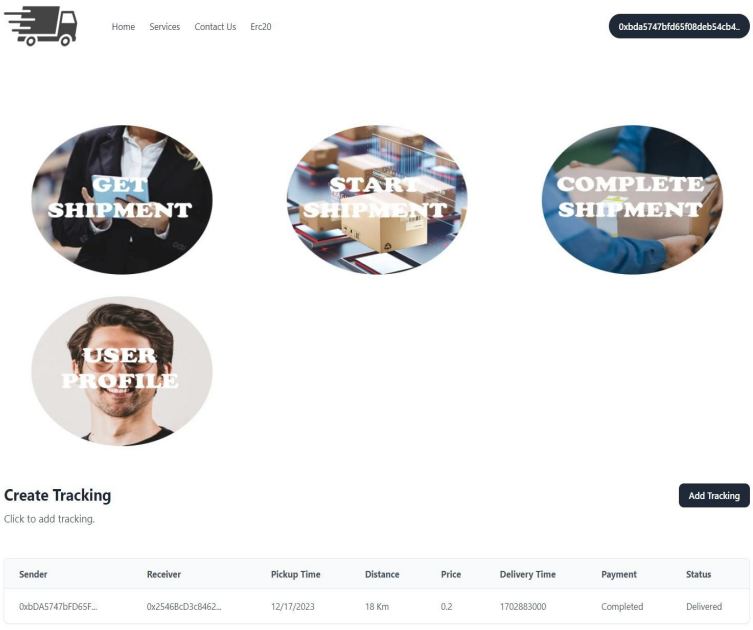
Tailwind CSS is a utility-first CSS framework designed to simplify and streamline the process of styling web applications. Its utility classes, applied directly in HTML, offer flexibility, customization, and a composable design system. Considered to use Tailwind because it is suitable for DApp development due to its responsiveness, ease of use, and seamless integration with React can make it an ideal choice for building visually appealing and customizable interfaces.

Interface

To activate the smart contract, an interactive interface is required to execute tasks. We have designed a simple interface with key features, including user profile, connect wallet, create tracking, get shipment, start shipment, send shipment, and complete shipment to enhance convenience of user. The primary focus is on placing the key features of the decentralized application (DApp) at the center. To ensure user comprehension of the process, we have organized the information of each tracking list in a table below the key features, making it easier to check the status.



And the figure below is the final interface of the DApp.



Testing and Deployment

Testing and Deployment Tools

1. **Hardhat**

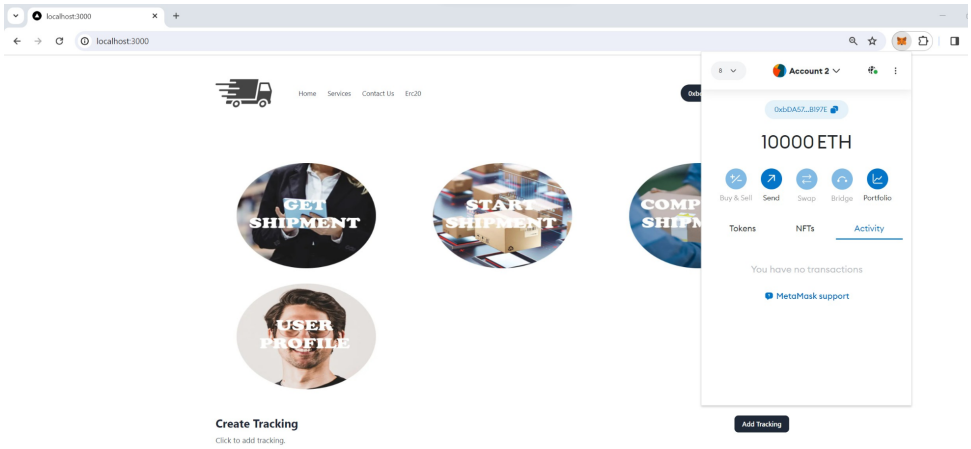
   Hardhat is a development environment and task runner for editing, compiling, testing and deploying Ethereum smart contracts.It streamlines DApps development with features like a run solidity locally, debugging-first, extreme flexibility, fully extensible, Ethereum network management, and a plugin system. It provides a convenient and flexible environment for writing, testing and deploying smart contracts, making it a popular choice in the Ethereum development ecosystem. Hardhat was used to provide virtual address, virtual private key, and virtual ETH coin for connecting to MetaMask. It provides 10000 ETH for each address which makes it more convenient for operating the system.

2. **MetaMask**

   MetaMask is a browser extension cryptocurrency wallet, facilitating DApp interaction, token storage, and Ethereum trading. Its security features eliminate the need for private key entry, ensuring safe cryptocurrency management and transactions. Popular among crypto gamers, developers, and newcomers, MetaMask's utility extends to its Web3 integration, network switching, and open-source nature. Using metamask for DApp, it provides a user-friendly experience, secure transaction signing, and compatibility across browsers, enhancing accessibility and familiarity for users in the decentralized application ecosystem.

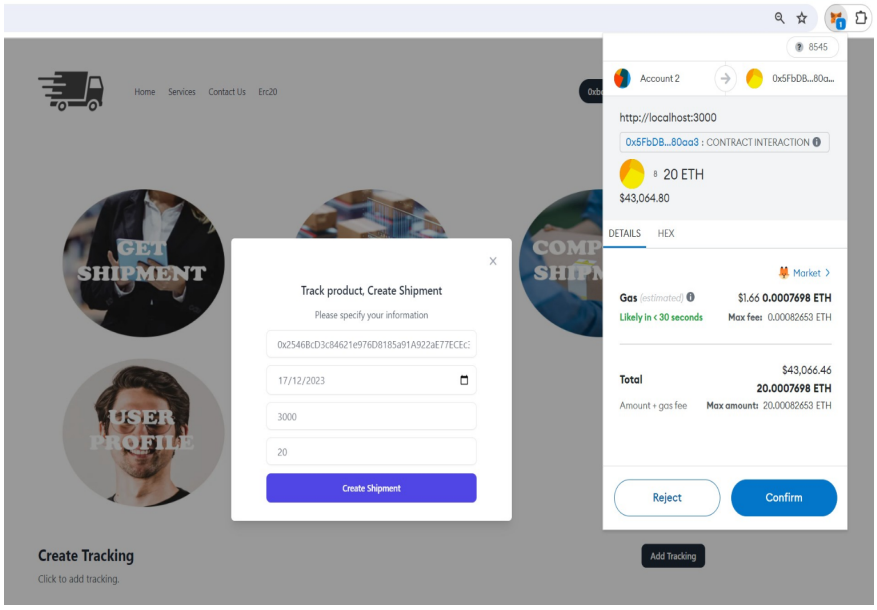
Testing :

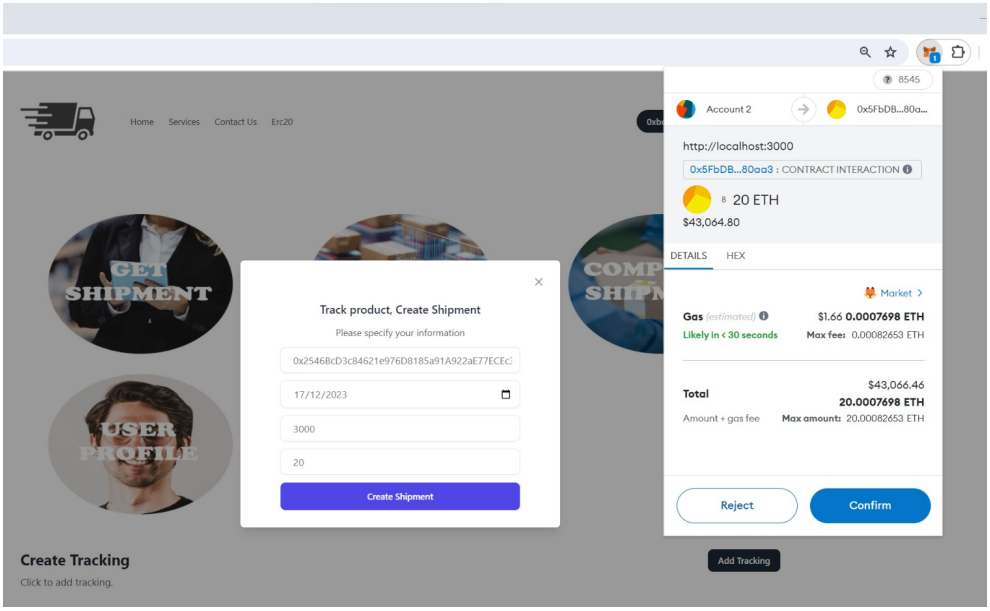
The testing phase forms the foundation of ensuring the reliability and robustness of a smart contract. Their role is to confirm collective functionality and interaction, ensuring that functions operate harmoniously as intended, solidifying the operational reliability of the contract. Additionally, robustness checks are integral, assessing the contract's responses across diverse scenarios and error-handling mechanisms, fortifying its resilience to handle unexpected situations and ensuring its solidity. The following steps explain how to ensure the functionalities of the smart contract:



This picture shows the initial amount of ETH

- createShipment Test:
  - Confirm that a shipment is created correctly with valid input data.
  -





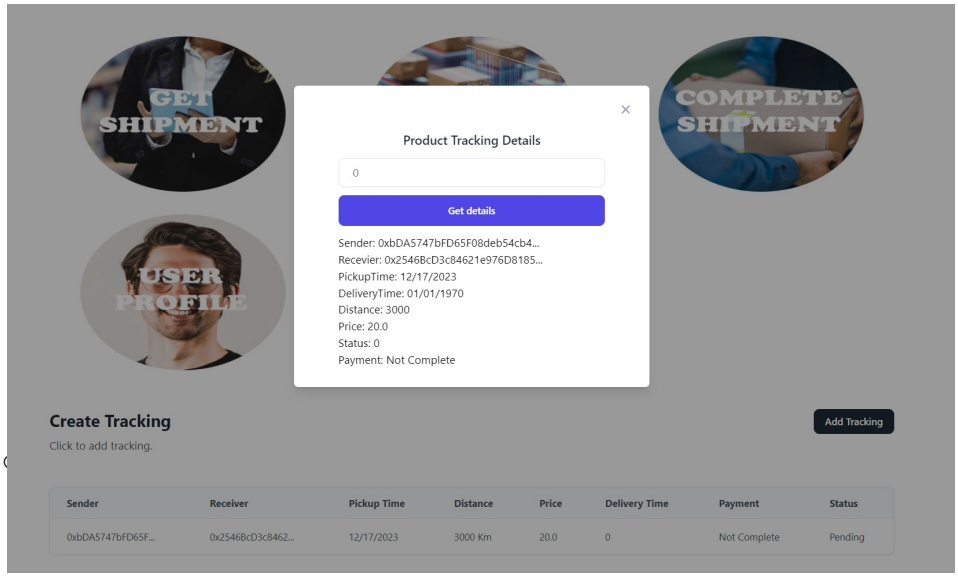
- Verify the emission of the ShipmentCreated event.

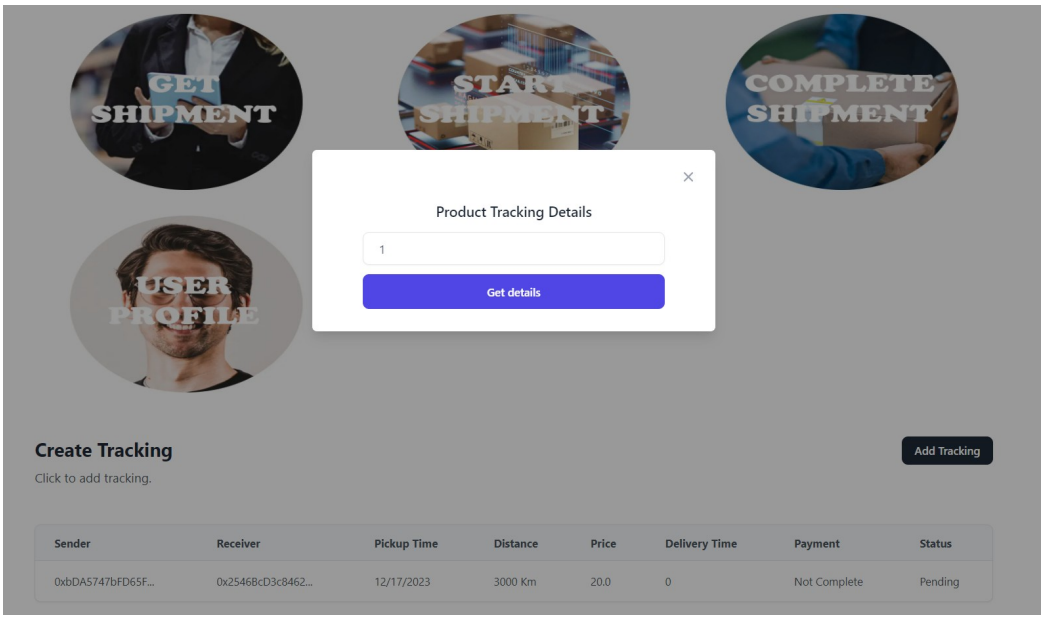
Create Tracking Add Tracking

Click to add tracking.

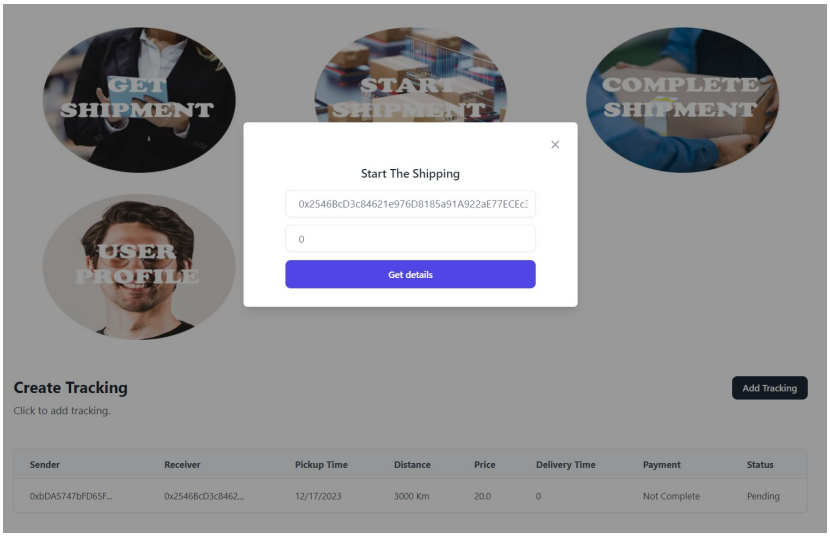
Sender	Receiver	Pickup Time	Distance	Price	Delivery Time	Payment	Status
0xbDA5747bFD65F...	0x2546BcD3c8462...	12/17/2023	3000 Km	20.0	0	Not Complete	Pending

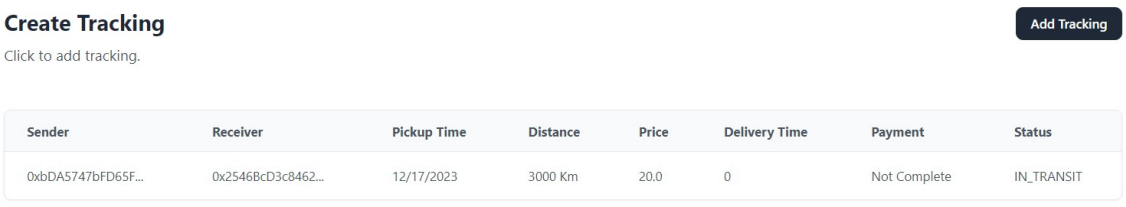
- getShipment Test:
  - Fetch a specific shipment and verify that the returned data matches the expected values.



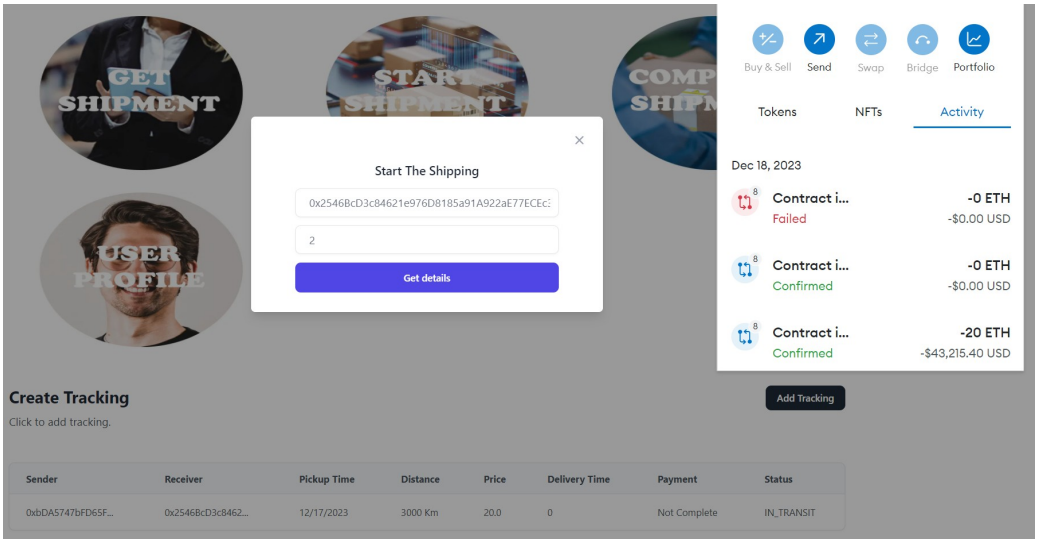


- startShipment Test:
    - Test the transition of a shipment's status from PENDING to IN_TRANSIT.





- Ensure the sender cannot start a shipment that is already in transit or if it is not included



- Validate the emission of the ShipmentInTransit event.

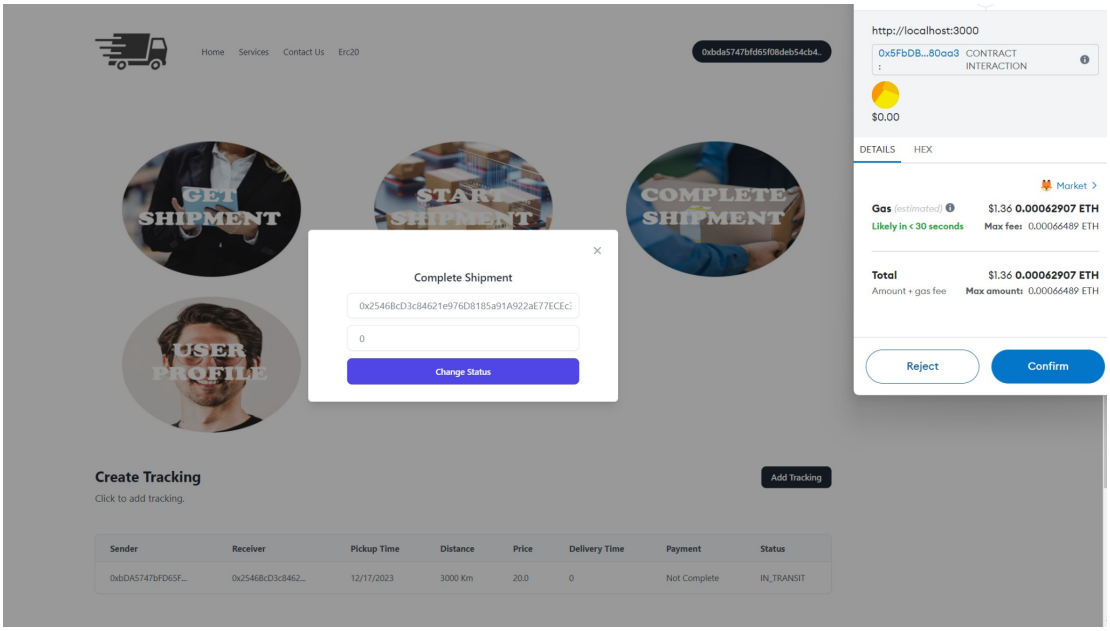
Create Tracking

Click to add tracking.

Add Tracking

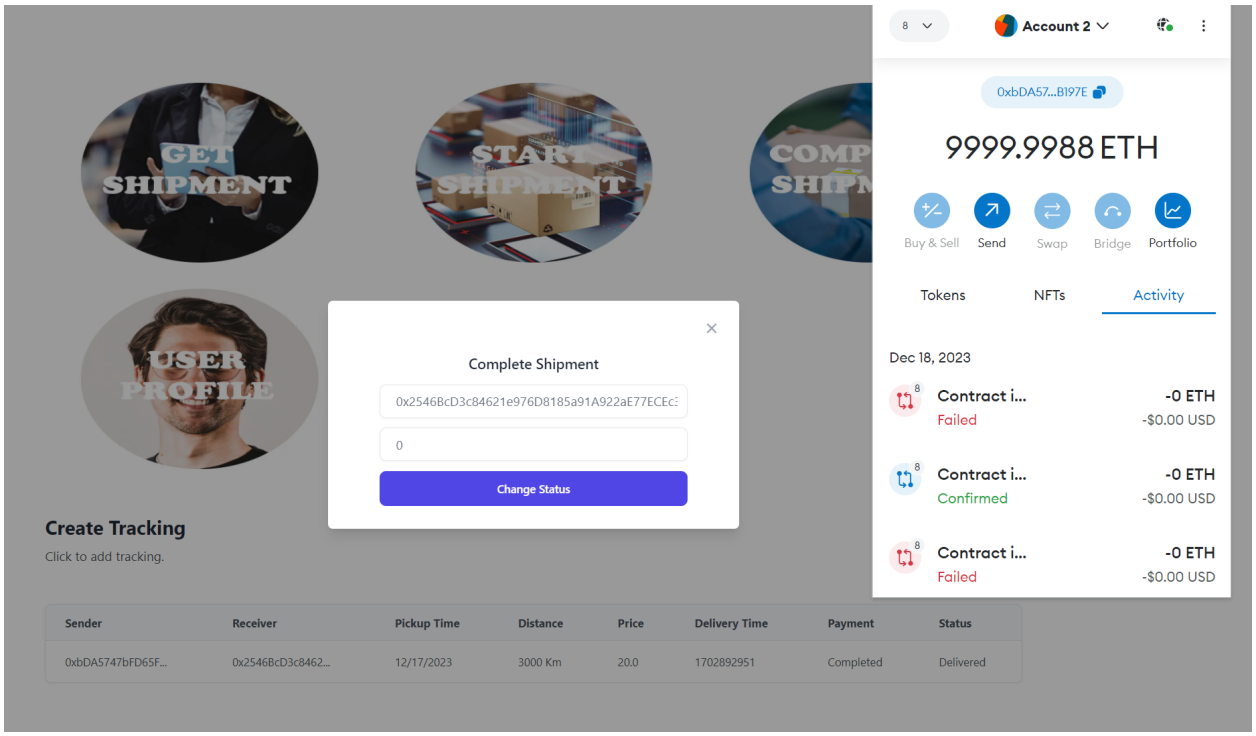
Sender	Receiver	Pickup Time	Distance	Price	Delivery Time	Payment	Status
0xbDA5747bFD65F...	0x2546BcD3c8462...	12/17/2023	3000 Km	20.0	1702892951	Completed	Delivered

- completeShipment Test:
  - Verify the transition of a shipment's status to DELIVERED upon completion.

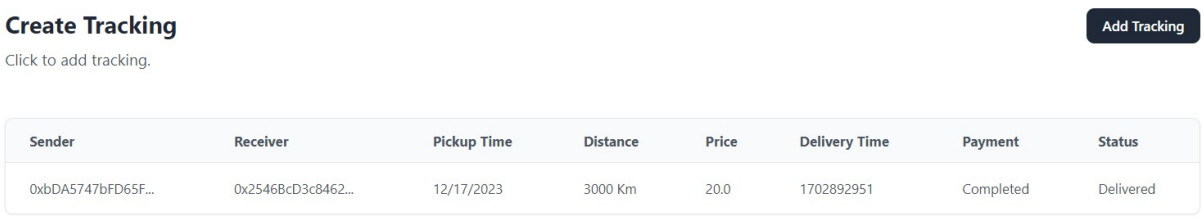


- Test the payment transfer to the sender when completing the shipment.

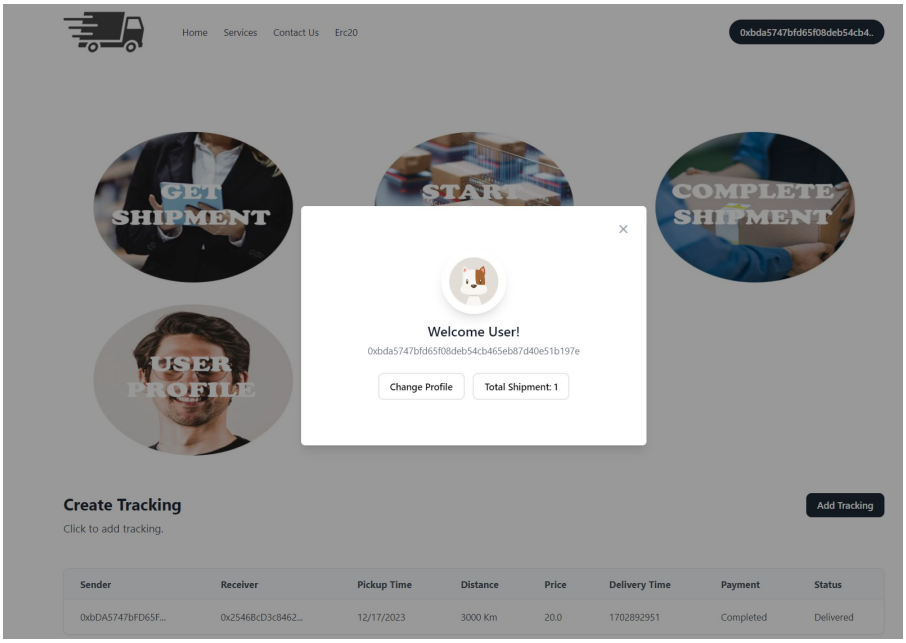
- Confirm that a shipment cannot be completed if it is not in transit or if it is already paid.



- Check the emission of the ShipmentDelivered and ShipmentPaid events.



- getShipmentsCount Test:
    - Validate that the count of shipments matches the expected count for a given sender.



Deployment:

Deployment Preparation represents a critical phase involving meticulous steps to prepare the contract for deployment onto the blockchain network. The following steps cover the preparation for deploying this contract onto the blockchain:

- Selection of Network:
    - Run Ethereum blockchain on local node

```
PS C:\Users\M_kavinsky\Desktop\Capstone Project> npx hardhat node
WARNING: You are using a version of Node.js that is not supported, and it may work incorrectly
, or not work at all. See https://hardhat.org/nodejs-versions


Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts
========

WARNING: These accounts, and their private keys, are publicly known.
Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39Fd6e51aad88F6F4ce6aB8827279cfffFb92266 (10000 ETH)
Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

Account #1: 0x70997970C51812dc3A010C7d01b50e0d17dc79C8 (10000 ETH)
Private Key: 0x59c6995e998f97a5a0044966f0945389dc9e86dae88c7a8412f4603b6b78690d

Account #2: 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC (10000 ETH)
Private Key: 0x5de4111afa1a4b94908f83103eb1f1706367c2e68ca870fc3fb9a804cdab365a

Account #3: 0x90F79bf6EB2c4f870365E785982E1f101E93b906 (10000 ETH)
Private Key: 0x7c852118294e51e653712a81e05800f419141751be58f605c371e15141b007a6

Account #4: 0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65 (10000 ETH)
Private Key: 0x47e179ec197488593b187f80a00eb0da91f1b9d0b13f8733639f19c30a34926a
```

- Compilation:
    - Use a compatible Solidity compiler to compile the code to bytecode.
- Deployment Scripts/Tools:
    - Write deployment scripts using Hardhat as a tool
    - Ensure that the deployment scripts include necessary details like gas limits, constructor arguments, and other deployment specifics.

```
scripts > JS deploy.js > ...
1   const hre = require("hardhat");
2
3   async function main() {
4     const Tracking = await hre.ethers.getContractFactory("Tracking");
5     const tracking = await Tracking.deploy();
6
7     await tracking.deployed();
8
9     console.log(`Tracking deployed to ${tracking.address}`);
10  }
11
12  main().catch((error) => {
13    console.error(error);
14    process.exitCode = 1;
15  });
```

- Deployment:
    - Deploy the contract onto the chosen network using the deployment scripts

```
PS C:\Users\M_kavinsky\Desktop\Capstone Project> npx hardhat run --network localhost scripts
/deploy.js
WARNING: You are using a version of Node.js that is not supported, and it may work incorrect
ly, or not work at all. See https://hardhat.org/nodejs-versions


Compiled 1 Solidity file successfully
Tracking deployed to 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

- Verification:
    - Verify the contract's deployment status using blockchain explorers to confirm successful deployment.
    - Interact with the deployed contract on the blockchain to validate its functionality in the live environment.

```
PS C:\Users\M_kavinsky\Desktop\Capstone Project> npm run dev

> tracking@0.1.0 dev
> next dev

ready - started server on 0.0.0.0:3000, url: http://localhost:3000
warn  - Your project has `@next/font` installed as a dependency, please use the built-in `ne
xt/font` instead. The `@next/font` package will be removed in Next.js 14. You can migrate by
 running `npx @next/codemod@latest built-in-next-font .`. Read more: https://nextjs.org/docs
/messages/built-in-next-font
Browserslist: caniuse-lite is outdated. Please run:
  npx update-browserslist-db@latest
  Why you should do it regularly: https://github.com/browserslist/update-db#readme
event - compiled client and server successfully in 1014 ms (383 modules)
wait  - compiling...
event - compiled successfully in 86 ms (318 modules)
```