PIM SCM DApp

John Black

Introduction:

My project is a decentralized product information management (PIM) and supply chain management (SCM) system. It enables manufacturers to share information about their products with their retailers and sales representatives securely and efficiently in real-time. There is also functionality for customers to place orders for products through the app, and the orders are stored on the blockchain.

The dApp allows manufacturers to input and manage product information, including descriptions, images, and pricing information. The app would be accessed by suppliers, distributors, retailers, and sales representatives who can use the information available to make informed purchasing and sales decisions. Manufacturers are able to track inventory levels through the application, and entities looking to purchase goods will be able to view inventory levels when making purchases.

The frontend uses React, a popular JavaScript library for building user interfaces, to create a seamless and intuitive user experience. I use React to create components for the app, which allows for more streamlined development.

To interact with the Ethereum blockchain, I integrated Web3.js, a JavaScript library that provides the necessary tools and APIs to communicate with smart contracts. Web3.js allows us to send transactions to the blockchain, read data from smart contracts, and receive events triggered by the contract's state changes. This integration allows users to directly interact with the smart contract.

For the smart contract development, I used Solidity, a smart contract programming language. This contract serves as the backbone of the application, providing the necessary functions and data structures to add, retrieve, and manage product information.

Process and Implementation:

The development process of the application involved several key steps, including smart contract design, frontend development, integration with Web3.js, and testing. Throughout the process, I encountered various challenges that required problem-solving and iterative development.

Solidity Smart Contract:

The first step of development was the creation of the smart contract, which is responsible for storing and managing product information on the Ethereum blockchain. The smart contract was created using Solidity, and the main data structure used in the contract is the Product struct, which contains product attributes such as description, image, and price. The contract includes functions to add products, retrieve product details, and manage inventory. Since Solidity is a compiled language, I needed to compile the smart contract before using it. I tried to use the solc and truffle compilers, but I was having trouble with the compilation process due to the terminal not being able to recognize the compilers names. To solve this problem, I used the Remix Ethereum IDE, which is a web application that is used to compile Solidity project. I used this compiler to create “ProductInformationContract.json”, which is needed by Web3.js to interact with the smart contract.

The frontend of the application is built using React, providing a user-friendly interface for interacting with the smart contract. I created three components, AddProductForm, PlaceOrderForm, and ProductInformation. The components are organized in a modular and reusable manner, promoting code efficiency and maintainability.

AddProductForm:

The AddProductForm component serves as a form for adding new products to the system. It includes input fields for entering the product details such as the product ID, description, image URL, price, and the sender's Ethereum address. The component utilizes the useState hook to manage the form's state, capturing the user's input values. When the form is submitted, the handleSubmit function is triggered, which interacts with the smart contract's addProduct function to store the product information on the blockchain.

const AddProductForm = ({ contractInstance, onAddProduct }) => {

Declare state variables for product information

const handleFormSubmit () {

try {

Add product with contractInstance to blockchain

} catch (error) {

Add new product with onAddProduct if web3 cannot connect

}}

return HTML code for add product form

};

PlaceOrderForm:

The PlaceOrderForm component provides a form for placing orders for products. Similar to the AddProductForm, it utilizes the useState hook to manage the form's state. The user can enter the desired product ID and the quantity they wish to order. Upon form submission, the handleSubmit function interacts with the smart contract's placeOrder function to initiate the order placement process on the blockchain.

const PlaceOrderForm = ({ contractInstance, onPlaceOrder }) => {

Declare state variables for order information

const handleFormSubmit () {

try {

Add product with contractInstance to blockchain

} catch (error) {

Add new product with onPlaceOrder if web3 cannot connect

}}

return HTML code for place order form

};

ProductInformation:

The ProductInformation component is responsible for displaying the product details and inventory levels. It retrieves the product information and inventory data from the smart contract and renders it in a user-friendly manner. The component utilizes the map function to iterate over the product data array and dynamically generate the HTML elements for each product. This component provides a clear view of the available products, their descriptions, images, prices, and current inventory levels.

One of the challenges I faced during frontend development was handling form submissions and integrating them with the smart contract. I will touch more on this issue in the Web3.js section.

const ProductInformation = ({ contractInstance, dummyProducts, dummyOrders }) => {

Declare state variables for products and orders

useEffect(() => {

const fetchData = async () => {

try {

Fetch product information from the smart contract

Fetch inventory levels from the smart contract

Fetch order status from the smart contract

} catch (error) {

Set products and orders manually if web3 cannot connect

setProducts(dummyProducts);

setOrders(dummyOrders)

}

};

fetchData();

},

);

return HTML code for product and order information

};

Integration with Web3.js:

To interact with the deployed smart contract and the Ethereum blockchain, I integrated Web3.js into the frontend code. Web3.js provides APIs for connecting to the blockchain network, sending transactions, and reading data from smart contracts. I initialized a Web3 instance and connected it to a local or remote Ethereum provider. I then use the contract's ABI (Application Binary Interface), which came from the Remix Ethereum IDE compilation, and contract address to instantiate a contract instance. This allows us to call its functions and retrieve data from the blockchain.

The majority of my challenges with the project came from interfacing with Web3.js. I was able to create a functional smart contract which was tested using the GUI of the Remix Etheruem IDE, which allowed me to directly call functions in my smart contract on a mock blockchain after compilation. After thorough testing of the smart contract, I was ready to integrate it with the frontend using Web3.js. However, I was unable to connect to the app through a web browser, as Web3 was never able to make a connection to a wallet local to the browser. Because of this, I had to create additional variables in the React app to handle taking the user inputs directly without adding them to the smart contract.

Results:A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Displayed above is the frontend of the application. Everything works in displaying the information, besides updating the inventory and the status of the order. I was unable to implement these features because the information in the above example is not being properly stored on the blockchain, as Google Chrome was unable to connect to the MetaMask wallet extension using Web3.js. The form with the add product submission button is used for adding products to inventory. The form with the place order button is used for placing an order of one of the items in the inventory.

Conclusion:

During the development of this project, I gained valuable insights and learned several important lessons. One of the important things I learned during this project was how to create a smart contract using Solidity. There was a bit of a learning curve due to the unique syntax, but I was able to learn a lot about creating a smart contract once I understood how to write code with Solidity.

Working with Web3.js, the JavaScript library for interacting with the Ethereum blockchain, was another significant learning experience. I learned how to establish a connection with the blockchain network, send transactions, and retrieve data from smart contracts. Although I was never able to integrate Web3.js with the frontend, I still learned about connecting a smart contract to the frontend of an app.

I could firstly improve on the site by getting the frontend to display real data from the blockchain. I was unable to get this functionality in the app at the time of submission, but that is something that would be added in the future. The site would also need a login page and separate pages for people who are inputting product and people who are placing orders.

Overall, I learned many new things about creating a decentralized application over the course of completing this project, including creating a smart contract and interfacing with it through a React frontend app, and gained a deeper understanding about how a dApp functions on the Ethereum blockchain.