

SMART INVEST USING BLOCKCHAIN

Main Project Report

Submitted by

Abel Tomy

Reg No : FIT20MCA-2002

*Submitted in partial fulfillment of the requirements for the award of
the degree of*

*Master of Computer Applications
Of*

A P J Abdul Kalam Technological University



FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)®

ANGAMALY-683577, ERNAKULAM(DIST)

JUNE 2022

DECLARATION

I, **Abel Tomy** hereby declare that the report of this project work, submitted to the Department of Computer Applications, Federal Institute of Science and Technology (**FISAT**), Angamaly in partial fulfillment of the award of the degree of Master of Computer Application is an authentic record of my original work.

The report has not been submitted for the award of any degree of this university or any other university.

Date :

Signature :

Place : Angamaly

Name :

FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY (FISAT)®
ANGAMALY, ERNAKULAM-683577

DEPARTMENT OF COMPUTER APPLICATIONS



CERTIFICATE

This is to certify that the project report titled **"SMART INVEST USING BLOCKCHAIN"** submitted by **Abel Tomy [Reg No: FIT20MCA-2002]** towards partial fulfillment of the requirements for the award of the degree of Master of Computer Applications is a record of bonafide work carried out by her during the year 2022.

Project Guide

Head of the Department

Submitted for the viva-voice held on at

ACKNOWLEDGEMENT

I am deeply grateful to **Dr. Manoj George**, Principal, FISAT, Angamaly for providing me with adequate facilities, way and means by which I was able to complete my main project work and I express my sincere thanks to **Dr. C Sheela**, Vice Principal FISAT, Angamaly.

My sincere thanks to **Dr. Deepa Mary Mathews**, Head of the department of MCA, FISAT, who had been a source of inspiration. I express heartiest thanks to **Ms. Manju Joy**, my project guide, for the encouragement and valuable suggestions. I express my heartiest gratitude to my scrum master **Dr. Sujesh P Lal** and the faculty members in my department for their constant encouragement and never ending support throughout the project. I would also like to express my sincere gratitude to the lab faculty members for their guidance.

Finally I express my thanks to all my friends who gave me wealth of suggestions for successful completion of this project.

ABSTRACT

As an attempt to collaborate the digital world with the physical one, we maintain a track of the journey of the products from producer to consumer. The final consumer can access a complete record of details and trust that the information is accurate and precise. One of the upcoming technologies, blockchain, is a great way for handling the quality of supply-chain management, since it uses the distributed public general ledger. Blockchain technology proves to be helpful in the supply-chain sector in the following manner: reduce errors, avoid product delays, eliminate fraudulent activities, improve management, increase consumer/supplier trust, and so on. Blockchain technology offers important opportunities for supply chain management. In case of lack of transparency, blockchain provides recorded information about a variety of transactions in goods and/or services, which are recorded and tracked in real time. Blockchain can provide a permanent, shareable, auditable record of products through their supply chain, which improves product traceability, authenticity, and legality in a more cost-effective way.

Contents

1	INTRODUCTION	8
2	PROOF OF CONCEPT	10
2.1	EXISTING SYSTEM	11
2.2	PROPOSED SYSTEM	11
2.3	OBJECTIVES	12
3	IMPLEMENTATION	13
3.1	REQUIREMENTS	15
3.1.1	Hardware Requirements	15
3.1.2	Software Requirements	15
3.2	PREREQUISITES	17
3.2.1	Ganache	17
3.2.2	Ethereum	18
3.2.3	Metamask	18
3.2.4	Web3	18
3.2.5	Solidity	19
3.2.6	React	20
3.2.7	Node.js	21
3.2.8	Truffle	22
3.3	ALGORITHM	23
3.4	SYSTEM ARCHITECTURE	24

4	PROJECT DESCRIPTION	26
4.1	MODULES	27
4.1.1	Assign Roles	27
4.1.2	Users Activity Management	28
4.1.3	Explore Activity And Invoice Management	30
5	CONCLUSION AND FUTURE SCOPE	31
5.1	CONCLUSION	31
5.2	FUTURE SCOPE	32
6	APPENDIX	33
6.1	CODING	33
6.1.1	App.js	33
6.1.2	Migration.sol	36
6.1.3	Manufacture.js	36
6.1.4	Seller.js	40
6.1.5	Delivery Hub.js	43
6.1.6	Customer.js	47
6.2	SCREEN SHOTS	51
7	REFERENCES	60

Chapter 1

INTRODUCTION

A block chain technology is a digital ledger of record of transactions, which based on decentralized network in a peer-to-peer network around the world. Block chain, is focused primarily online transactions and disseminated digital ledger system. While comparing with traditional centralized database, the information cannot be handled due to block chain's feature of distributed nature and confirmed guarantees by the peers. If someone wants to perform a transaction, it goes to the network directly and algorithms find the authenticity of the transaction. After the transaction is verified, the new one is linked with the previous transaction forming a chain of transactions. This chain is called the block chain. The agreement between two people in the form of a computer code is called as Smart Contract. They run on the block chain, so they are stored on a public database and cannot be changed.

The combination of block chain technology and smart contract gives more flexibility to design, develop and implement in real-time with minimum cost. Block chain-based smart contracts provide number of advantages: they are quick and real-time refurbish, minimum cost and lower risk in execution, no intermediaries and high accuracy.

Block Chain Technology in Supply Chain:- The supply chain management system consists of several stages and various sectors of applications in supply chain. Block chain system makes supply chain sector more reliable. The Smart Invest is such an implementation of a supply chain management system which uses blockchain to ensure a transparent and secure transfer of product from the manufacturer to the customer via the online e-commerce websites.

Chapter 2

PROOF OF CONCEPT

The process of verifying that the idea has the potential in a real-world situation. Its aim is to determine whether the project is feasible and will function as planned. Blockchains helps in tracking and identifying provenance (i.e., proof-of-origin) of a product, because blockchain provides a secure and trusted tracking system from one end of the supply chain (the creation or mining of raw materials) all the way to other end of the supply chain (where the end user enjoys the finished product). The implementation of a supply chain management system which uses blockchain to ensure a transparent and secure transfer of product from the manufacturer to the customer via the online e-commerce websites.

2.1 EXISTING SYSTEM

A centralised supply chain is the traditional supply chain model, featuring a central headquarters and warehouse based in a single location. From a supply chain management standpoint, a centralised supply chain operation is typically managed at the headquarters – which handles all up and downstream decisions.

The system provide a centralised system that has the ability to provide a interaction between the main system that generate the product or entity to the end of the cycle. In a centralized system, you'll typically find a single business headquarters and/or a single warehouse full of departmental managers in areas like logistics, distribution and procurement. These managers are responsible for overseeing their specific area throughout the entire supply chain from this central office. If the organization is large, there may be multiple hubs, but these will be spread far enough apart to handle production lines in different time zones.

Centralize Supply Chain Processes are those that can be managed from a remote location, can serve multiple sites, and drive synergy by serving multiple sites. This centralised system shows disadvantages that changes entire structure:

- Cross-country customers who have an urgent need may be charged steep shipping fees. In a market dominated with free 2nd-day delivery, this could force them to look elsewhere for fulfillment.
- Ownership of cost and staffing for all areas of business including systems, security, recruiting and training can be very costly.

To solve the above problems, a blockchain based solution for the supply chain system is proposed. Blockchain is a decentralized network in which unreliable peers interact among themselves and perform transactions.

2.2 PROPOSED SYSTEM

Blockchain provides a distributed ledger or database which is shared among all participants in the network based on the consensus mechanism. The need for a

third party verifier is eliminated, making the system secure and completely decentralized. Any transaction which results in a modification to the Blockchain ledger is digitally signed, verified and validated by miner nodes which keep a duplicate of the ledger. This creates completely decentralized, secure, time-stamped and shared tamper-proof ledgers.

Blockchain technology has been utilized in many industries such as finance, healthcare, supply chain, logistics, document management and accounting [6–8]. Due to its robust and decentralized infrastructure, blockchain technology is applied to handle issues related to trust, efficiency, privacy and data sharing [9]. This technology eliminates the requirement of a third party transaction authority by leveraging the potential of cryptography to provide trustworthy solutions for the entities participating in the chain.

- Transparency in System and Security
- Diminishing Corruption
- Decentralized Database

2.3 OBJECTIVES

- Main aim is to integrating blockchain technology with the platform.
- To provide better transparency and security to data.
- By using a block chain technology, consumers do not need to rely on the trusted third parties to know the source of the purchased product safely.
- To generate an invoice that shows the track of the product which is ordered by the customer.

Chapter 3

IMPLEMENTATION

The proposed system provide a blockchain implementation for better transparency, security and automation. Blockchain is the backbone Technology of Digital Cryptocurrency BitCoin. A Blockchain is a list of records called blocks that are linked together using linked lists and use the cryptographic technique. Each block contains its own digital fingerprint called Hash, the hash of the previous block, a timestamp and the data of the transaction made, making it more secure towards any kind of data breach.

Therefore, if the data of one block is changed then its hash will also change. If the hash is changed, then its hash will be different from the next block that contains the hash of the previous block affecting all the hashes of the blocks after it. Changing of the hashes and then comparing it with other blocks allows us to check the blockchain.

With the use of series of actions and module capabilities, the transaction begins with the initialization and assigning the roles for each manufacturer, seller, delivery hub and the customer. Its a flow chain system that begins with the manufacturer and ends with the customer. The implementation of a supply chain management system which uses blockchain to ensure a transparent and secure transfer of product from the manufacturer to the customer via the online e-commerce websites. Each of the transaction and its corresponding details can be seen using the

unique id that is provided during the registry of the product done by the manufacturer.

The flow of the entire system is shown below:

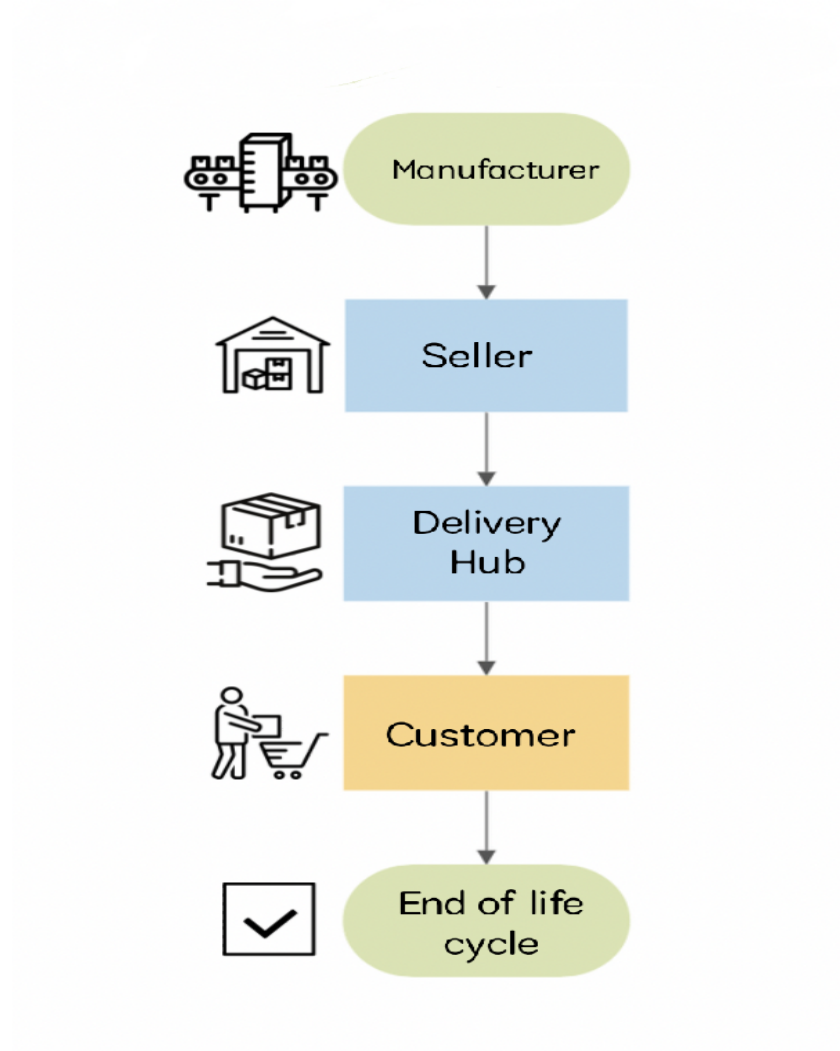


Figure 3.1: Smart Invest Flow

3.1 REQUIREMENTS

3.1.1 Hardware Requirements

Selection of hardware configuration is very important task related to the software development. The processor should be powerful to handle all the operations. The hard disk should have the sufficient capacity to compile and migrate the rules and run the react application.

Processor	core i5/i7
RAM	8 GB
Storage	1TB HDD
Display	15.6" (39.62 cm) display, 1366 x 768 p
Operating System	Windows 10/11

3.1.2 Software Requirements

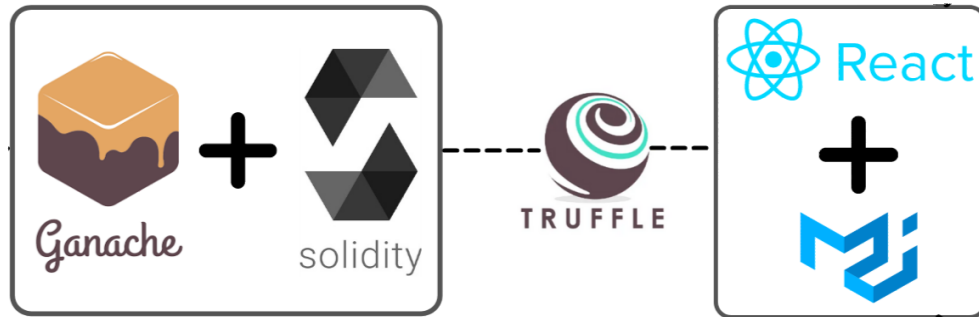
HTML: The Hyper Text Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be The HyperText Markup Language, or HTML is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript. Web browsers receive HTML documents from a web server or from local storage and render the documents into multimedia web pages.

CSS : Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML.CSS is a cornerstone technology of the World Wide Web, HTML and JavaScript. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts.

JavaScript: JavaScript, often abbreviated as JS, is a programming language that conforms to the ECMAScript specification. JavaScript is high-level, often just-in-time compiled, and multi-paradigm. It has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions. Alongside HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. All major web browsers have a dedicated JavaScript engine to execute the code on the user's device. As a multi-paradigm language, JavaScript supports event-driven, functional, and imperative programming styles.

Android Studio: Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on IntelliJ IDEA software and designed especially for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is placed for the Eclipse Android Development Tools (ADT) as primary IDE for native Android application development.

3.2 PREREQUISITES



3.2.1 Ganache

Quickly fire up a personal Ethereum blockchain which you can use to run tests, execute commands, and inspect state while controlling how the chain operates. Ganache is an Ethereum simulator that makes developing Ethereum applications faster, easier, and safer. It includes all popular RPC functions and features (like events) and can be run deterministically to make development a breeze.

- Zero-config Mainnet Forking
- Ethereum JSON-RPC support
- Snapshot/revert state
- Mine blocks instantly, on demand, or at an interval
- Fast-forward time
- Impersonate any account (no private keys required!)
- Listens for JSON-RPC 2.0 requests over HTTP/WebSockets

3.2.2 Ethereum

Ethereum is a technology that's home to digital money, global payments, and applications. The community has built a booming digital economy, bold new ways for creators to earn online, and so much more. It's open to everyone, wherever you are in the world – all you need is the internet.

Ethereum is a decentralized, open-source blockchain with smart contract functionality. Ether (ETH or) is the native cryptocurrency of the platform. Among cryptocurrencies, Ether is second only to Bitcoin in market capitalization.

3.2.3 Metamask

MetaMask is a software cryptocurrency wallet used to interact with the Ethereum blockchain. It allows users to access their Ethereum wallet through a browser extension or mobile app, which can then be used to interact with decentralized applications. MetaMask is developed by ConsenSys Software Inc., a blockchain software company focusing on Ethereum-based tools and infrastructure.

MetaMask allows users to store and manage account keys, broadcast transactions, send and receive Ethereum-based cryptocurrencies and tokens, and securely connect to decentralized applications through a compatible web browser or the mobile app's built-in browser.

3.2.4 Web3

Web 3.0, also known as the third-generation internet, is the next evolution of the World Wide Web. It provides a data-driven Semantic Web employing a machine-based understanding of data with the objective of developing a more intelligent and connected web experience for users. The Web of today is static and unable to adjust to the individual needs of each person experiencing it. Web 3.0 promises to be more dynamic and interactive. By implementing artificial intelligence and blockchain technology, it will redefine the web experience with structural changes to ensure democratization across all aspects of the internet.

In Web 3.0, data is stored securely and distributed across many devices, removing the need for centralized servers. Such a design also reduces the risks of massive data leaks because data is no longer centrally stored — making it more resilient to compromise.

Here are the layers of web3:

- **Decentralized Network:** Decentralized data networks allow different data producers to sell or exchange their data without losing ownership, jeopardizing privacy, or relying on middlemen.
- **Edge Computing:** While web 2.0 modifies presently personal computer technology in data centers, web 3.0 is pushing the data center out to the edge, that is edge computing, and perhaps right into our hands.
- **Artificial Intelligence and Machine Learning:** Artificial intelligence and machine learning algorithms have progressed to the point that they can now make useful, and sometimes life-saving, predictions and actions.

3.2.5 Solidity

Solidity is an object-oriented programming language for implementing smart contracts on various blockchain platforms, most notably, Ethereum. Solidity is compiled to bytecode (or portable code) that is executable on the Ethereum Virtual Machine (EVM), the runtime environment for smart contracts in Ethereum. Although other languages, including Serpent, Viper and Mutan, can also be compiled into EVM machine-level bytecode to run on Ethereum nodes for payment, Solidity is the most widely adopted.

Solidity was created to be easily learned because it employs many concepts — such as variables, functions, classes, arithmetic operations and string manipulation — that appear in popular modern programming languages. An intentionally pared-down, loosely-typed language, Solidity draws from C, C++, C, JavaScript, PowerShell and Python.

- Solidity is a high-level programming language designed for implementing smart contracts.
- It is statically typed object-oriented(contract-oriented) language.
- Solidity supports complex user-defined programming, libraries and inheritance.
- Solidity is primary language for blockchains running platforms.
- Solidity can be used to creating contracts like voting, blind auctions, crowd-funding, multi-signature wallets, supplychain etc.

3.2.6 React

React. js is an open-source JavaScript library that is used for building user interfaces specifically for single-page applications. It's used for handling the view layer for web and mobile apps. React also allows us to create reusable UI components. The advantages of using react are the following:

- **Intuitive** - ReactJS is extremely intuitive to work with and provides interactivity to the layout of any UI. Plus, it enables fast and quality assured application development that in turn saves tome for both - clients and developers.
- **Declarative** - ReactJS enables significant data changes that result in automatic alteration in the selected parts of user interfaces. Owing to this progressive functionality, there is no additional function that you need to perform to update your user interface.
- **Provides Reusable Components** - ReactJS provides reusable components that developers have the authority to reuse and create a new application . Reusability is exactly like a remedy for developers. This platform gives the

developers the authority to reuse the components build for some other application having the same functionality. Thereby, reducing the development effort and ensuring a flawless performance.

- **JavaScript library** - A strong blend of JavaScript and HTML syntax is always used, which automatically simplifies the entire process of writing code for the planned project. The JS library consists several functions including one that converts the HTML components into required functions and transforms the entire project so that it is easy to understand.
- **Components Support** - ReactJS is a perfect combination of JavaScript and HTML tags. The usage of the HTML tags and JS codes, make it easy to deal with a vast set of data containing the document object model. During this time, ReactJS works as a mediator which represents the DOM and assists to decide which component needs changes to get the exact results.

3.2.7 Node.js

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser, which was designed to build scalable network applications. Node.js has an event-driven architecture capable of asynchronous I/O. These design choices aim to optimize throughput and scalability in web applications with many input/output operations, as well as for real-time Web applications.

Node.js offers the following advantages:

- js enables asynchronous event-driven programming, which eliminated blocking processes. This improves scalability.
- It's a highly performant runtime environment. Developers with JavaScript expertise can easily learn Node.js, and it improves their productivity.
- Data streaming is easier to implement with Node.js.

- Thousands of sharable open-source modules and tools enrich the ecosystem, which also boasts of a vibrant developer community.
- Open-source NoSQL databases like MongoDB use JavaScript, therefore, a JavaScript developer can easily implement these.

3.2.8 Truffle

Truffle is a world-class development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM), aiming to make life as a developer easier. Truffle is widely considered the most popular tool for blockchain application development with over 1.5 million lifetime downloads. Truffle supports developers across the full lifecycle of their projects, whether they are looking to build on Ethereum, Hyperledger, Quorum, or one of an ever-growing list of other supported platforms. Paired with Ganache, a personal blockchain, and Drizzle, a front-end dApp development kit, the full Truffle suite of tools promises to be an end-to-end dApp development platform.

- Built-in smart contract compilation, linking, deployment and binary management.
- Scriptable, extensible deployment migrations framework.
- Network management for deploying to any number of public private networks.
- Interactive console for direct contract communication.
- Configurable build pipeline with support for tight integration.

3.3 ALGORITHM

SHA-3 Algorithm: SHA-3 (Secure Hash Algorithm 3) is the latest member of the Secure Hash Algorithm family of standards, released by NIST on August 5, 2015. Although part of the same series of standards, SHA-3 is internally different from the MD5-like structure of SHA-1 and SHA-2.

The hash function is responsible for converting the plaintext to its respective hash digest. They are designed to be irreversible, which means your digest should not provide you with the original plaintext by any means necessary. Hash functions also provide the same output value if the input remains unchanged, irrespective of the number of iterations.

Some of the standout features of the SHA algorithm are as follows:

- **Message Length:** The length of the cleartext should be less than 264 bits. The size needs to be in the comparison area to keep the digest as random as possible.
- **Digest Length:** The length of the hash digest should be 256 bits in SHA 256 algorithm, 512 bits in SHA-512, and so on. Bigger digests usually suggest significantly more calculations at the cost of speed and space.
- **Irreversible:** By design, all hash functions such as the SHA 256 are irreversible. You should neither get a plaintext when you have the digest beforehand nor should the digest provide its original value when you pass it through the hash function again.

3.4 SYSTEM ARCHITECTURE

The blockchain architecture consists of the elements like a node - user or computer that has a complete copy of the blockchain ledger, block - a data structure used for keeping a set of transactions, and transaction - the smallest building block of a blockchain system (records, information, etc.).

Smart Contracts: A smart contract is a computer code that can be built into the blockchain to facilitate, verify, or negotiate a contract agreement. Smart contracts operate under a set of conditions to which users agree. When those conditions are met, the terms of the agreement are automatically carried out.

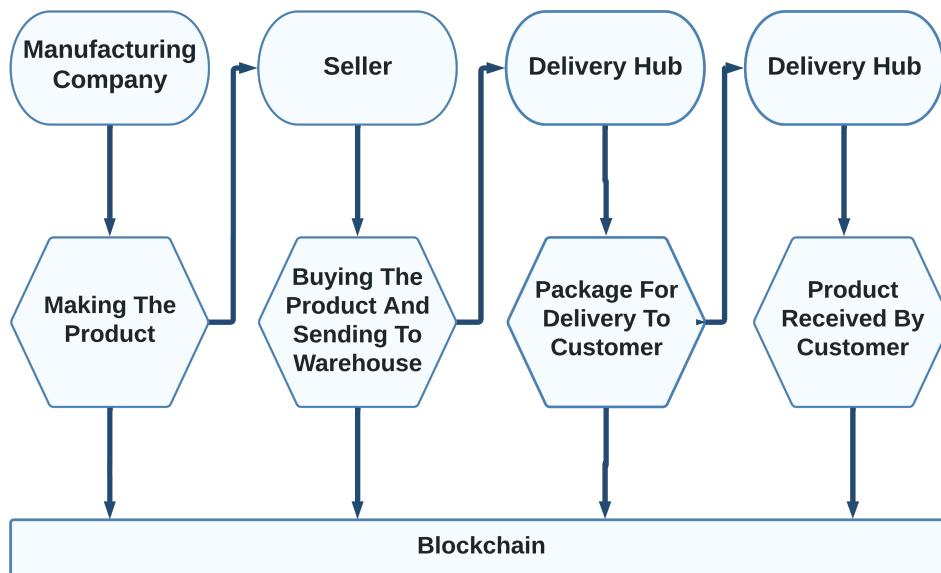


Figure 3.2: System Architecture

The smart contract is being written with Solidity which is then compiled, migrated and deployed using Truffle.js on the local blockchain network created using Ganache-cli. The frontend uses Web3.js to communicate with the smart contract and local blockchain network and is written using React.js framework for better component and state lifecycle management. The requests from user are forwarded

to frontend through Nginx(load balancer) and Express.js for dynamic routing.

Benefits of blockchain

- Accuracy of the Chain.

Transactions on the blockchain network are approved by a network of thousands of computers. This removes almost all human involvement in the verification process, resulting in less human error and an accurate record of information. Even if a computer on the network were to make a computational mistake, the error would only be made to one copy of the blockchain.

- Decentralization

Blockchain does not store any of its information in a central location. Instead, the blockchain is copied and spread across a network of computers. Whenever a new block is added to the blockchain, every computer on the network updates its blockchain to reflect the change. By spreading that information across a network, rather than storing it in one central database, blockchain becomes more difficult to tamper with. If a copy of the blockchain fell into the hands of a hacker, only a single copy of the information, rather than the entire network, would be compromised.

- Secure Transactions

Once a transaction is recorded, its authenticity must be verified by the blockchain network. Thousands of computers on the blockchain rush to confirm that the details of the purchase are correct. After a computer has validated the transaction, it is added to the blockchain block. Each block on the blockchain contains its own unique hash, along with the unique hash of the block before it.

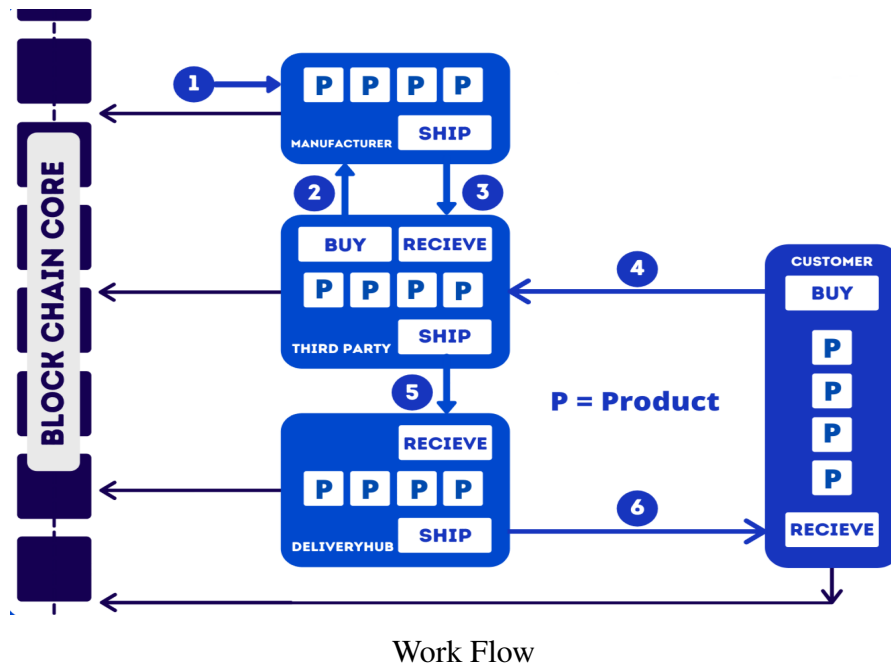
Chapter 4

PROJECT DESCRIPTION

The product starts when `manufactureProduct` is called(while making an entry) after the final product is manufactured and the product and manufacturer details are entered in the blockchain. The `productHistory` gets initialized and the current product data is stored with the current owner(manufacturer). Now this product shall be available to the Seller for purchase. On being purchased by a third party seller, the `purchasedBySeller` gets called where the owner is set to seller and the present data gets pushed to the `productHistory` (which helps us to track the origin and handling of the product).

Simultaneously, the product is shipped by the manufacturer (`shipToSeller`) and is received by the Seller where `receivedBySeller` is called and the details of the seller are entered. Each of these checkpoint's data is stored in product history with the state being updated at each step. The online purchase of the product takes place from the Third Party. When the customer orders the product, it is shipped by the Seller and received by the delivery hub where the `receivedByDeliveryHub` is called. Here the customer address is stored, owner is set to Delivery Hub, details of the Delivery Hub are fed and the current data state gets pushed to the `productHistory`.

Finally the product is shipped by the Delivery Hub (shipByDeliveryHub) and received by the customer where the receivedByCustomer is called and the current and final state gets pushed to the productHistory. All of these juncture functions shall be called only after complete verification of product and productHistory while entering a checkpoint.



4.1 MODULES

4.1.1 Assign Roles

Roles are created for various job functions. The permissions to perform certain operations are assigned to specific roles. Members or staff (or other system users) are assigned particular roles. Since users are not assigned permissions directly, but only acquire them through their role (or roles), management of individual user rights becomes a matter of simply assigning appropriate roles to the user's

account; this simplifies common operations, such as adding a user, or changing a user's department.

The roles that are assigned using the address that are provided by the personal blockchain network(Ganache). Each address are unique and are provided to each of the users. The users in which address are given:

- Manufacturer Address
- Seller Address
- Delivery Hub Address
- Customer Address

4.1.2 Users Activity Management

MANUFACTURER

A manufacturer is a person or company that produces finished goods from raw materials using a variety of tools, equipment and processes and then sells the products to consumers, wholesalers, distributors, retailers or other manufacturers for the manufacturing process.

- Create A Product Entry: The life-cycle of a product starts when manufacture product is called(while making an entry) after the final product is manufactured and the product and manufacturer details are entered in the blockchain. Now this product shall be available to the Third Party for purchase.
- Shipment To Seller: The request from the seller is received buy the manufacturer. This request is accepted and the product is shipped to the corresponding seller address.
- View All Products: Here, All the products that are registered and made visible.

SELLER/ THIRD PARTY

A seller represents a network consisting of a seller who wholesales a product to a buyer, who in turn retails it to the consumer.

- **Buy Products From Manufacturer:** As the product is made registered and made into the blockchain, the product is now visible to all the sellers. The seller buys the products by making a request to the manufacture ship the product.
- **Receive Product From Manufacturer:** The seller receives the shipment from the manufacturer.
- **Shipment To Delivery Hub:** The request from the customer is received regarding the product the seek to buy, the seller takes the order and ship the product to the delivery hub.

DELIVERY HUB

Delivery Hub refers to a central warehouse that's used to deliver to various destinations in SCM. That may be customers, stores, or other smaller warehouses. It's an important part of the supply chain strategy. Hub is used to optimizing the overall transportation costs.

- **Receive Product From Seller:** The delivery hub receives the product/shipment from the seller. The entire transaction till the product reaches the delivery hub is made visible.
- **Shipment To Customer:** The Product is shipped to the corresponding customer according to the time period of the delivery.

CUSTOMER

From the beginning of an order until order delivery, customers are involved in the process. The customer not only pays for the product or service, but they also

decide whether or not to do business with your company again based on their experience.

- **Purchase Product From Seller:** All the products that are available in the seller are listed. The customer purchase the product from the seller buy making an order for the product.
- **Receive Product From Delivery Hub:** The delivery hub deliver the product to the customer with entire transaction details.
- **View all Products:** The customer is able to see all the products that are bought by the account.

4.1.3 Explore Activity And Invoice Management

Explore Activity: The explore activity basically tract the entire the blockchain transaction. From the manufacture area to the customer part, the are blocks developed and the transactions are shown by the unique id generated at the time of product entry.

While the unique id is called by the user, the entire details of the product is displayed. This include the unique id, manufacturer address, time and place of product register, price, product id, seller address, delivery hub address, customer address and the hash value.

Invoice Management: Invoice management is an internal business function linked to procurement and is responsible for managing and processing invoice documents from vendors and suppliers. Invoice management is important to prevent delays and errors in receiving or paying for goods and services. Invoice management keeps track of sales/supplies and prevents wastage and delays.

The whole transactions and its details are provided in the explore activity. This helps in creating an invoice to the customer with all the details that are provided in the explore activity function.

Chapter 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

The main aim of the project provide a better understanding of Blockchain technology and the smart ivest supplychain management. The adoption of the Blockchain technology in the supply chain is a promising enhancement, suitable to provide benefits to all the different actors involved in the process. One of the most critical issues to be addressed in the implementation of Blockchain in the supply chain is the need to include all the different actors. Moreover, the sharing of the information along the entire Blockchain could lead to inertia in adopting the solution. For this reason, a correct implementation of the Blockchain technology in the supply chain must start from an analysis of the needs and the objectives of the different actors involved, with the aim to create a business model capable of highlighting the returns (both in economic and customer satisfaction terms) of this solution.

5.2 FUTURE SCOPE

Blockchain has emerged as the latest technology for solving various issues like lack of security, transparency, integrity etc. The encryption is done through cryptography to eliminate vulnerabilities such as unauthorized data tampering. Supply chain is always hard to manage and requires a lot of administrative machinery. However, when managed with smart contracts using blockchain, a lot of the paperwork is reduced. Also it leads to an increase in the transparency and helps to build an efficient Root of Trust. Various fields where the supply chain method are implemented can transact their way of action. So it will be very useful if blockchain can be integrated with those procedures. Hence most of the security as well as threat issues can be solved.

Chapter 6

APPENDIX

6.1 CODING

6.1.1 App.js

```
import Explorer from './pages/Explorer';
import Home from './pages/Home';
class App extends Component {
  state = { web3: null, accounts: null, contract:
  null,
  mRole: null, tpRole: null, dhRole: null, cRole:
  null };
  componentDidMount = async () => {
    try {
      const web3 = await getWeb3();
      const accounts = await web3.eth.getAccounts();
      const networkId = await web3.eth.net.getId();
      const deployedNetwork = SupplyChainContract.
        networks[networkId];
      const mRole = localStorage.getItem("mRole");
```

```
const tpRole = localStorage.getItem("tpRole");
const dhRole = localStorage.getItem("dhRole");
const cRole = localStorage.getItem("cRole");
this.setState({ web3, accounts, contract:
instance,mRole: mRole, tpRole: tpRole, dhRole:
dhRole, cRole: cRole }, this.runExample);
} catch (error) {
alert(
'Failed to load web3, accounts, or contract.
Check console for details.',
);
console.error(error);
} }];
runExample = async () => {
const { contract } = this.state;
console.log(contract);
};
render() {
if (!this.state.web3) {
return <div>Loading Web3, accounts, and contract...
</div>;
}
return (
<div className="App">
<ThemeProvider theme={theme}>
<RoleDataContextProvider mRole={this.state.mRole}
tpRole={this.state.tpRole} dhRole={this.state.
dhRole} cRole={this.state.cRole}>
<Router history={createBrowserHistory()}>
<Switch>
```

```
<Route exact path="/roleAdmin">
<RoleAdmin accounts={this.state.accounts}
supplyChainContract={this.state.contract} />
</Route>
<Route exact path="/explorer">
<Explorer accounts={this.state.accounts}
supplyChainContract={this.state.contract}
</Route>
<Route exact path="/manufacturer/ship">
{this.state.mRole !== "" ?
<ShipManufacture accounts={this.state.accounts}
supplyChainContract={this.state.contract} />
<h1>Assign Manufacturer Role at /RoleAdmin</h1> }
<Route exact path="/Customer/buy">
{this.state.cRole !== "" ?
<PurchaseCustomer accounts={this.state.accounts}
supplyChainContract={this.state.contract} />
<h1>Assign Customer Role at /RoleAdmin</h1> }
<Route exact path="/DeliveryHub/ship">
{this.state.dhRole !== "" ?
<ShipDeliveryHub accounts={this.state.accounts}
supplyChainContract={this.state.contract} />
<h1>Assign Delivery Hub Role at /RoleAdmin</h1> }
<Route exact path="/Customer/allReceived">
{this.state.cRole !== "" ?
<ReceivedByCustomer accounts={this.state.accounts}
supplyChainContract={this.state.contract} />
<h1>Assign Customer Role at /RoleAdmin</h1> }
</Route></Switch></Router>
</RoleDataContextProvider>
```

```
</ThemeProvider>
</div>
);
}
}
export default App;
```

6.1.2 Migration.sol

```
pragma solidity >=0.4.21 <0.9.0;
contract Migrations {
  address public owner;
  uint public last_completed_migration;
  modifier restricted() {
    if (msg.sender == owner) _;
  }
  constructor() public {
    owner = msg.sender;
  }
  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  } }
}
```

6.1.3 Manufacture.js

```
import React from "react";
export default function Manufacture(props) {
  const supplyChainContract = props.supplyChainContract;
  const classes = useStyles();
  const { roles } = useRole();
```

```
const [loading, setLoading] = React.useState(false);
const [fvalid, setfvalid] = React.useState(false);
const navItem = [
  ["Add Product", "/manufacturer/manufacture"],
  ["Ship Product", "/manufacturer/ship"],
  ["All Products", "/manufacturer/allManufacture"],
];
const [manuForm, setManuForm] = React.useState({
  id: 0
const handleChangeManufacturerForm = async (e) => {
  setManuForm({
    ...manuForm,
    [e.target.name]: e.target.value,
  });
const handleSubmitManufacturerForm = async () => {
  setLoading(true);
  await supplyChainContract.methods.manufactureProduct
    (manuForm.manufacturerName, manuForm.manufacturerDetails,
    manuForm.productName, parseInt(manuForm.productCode),
    parseInt(manuForm.productPrice), manuForm.productCategory)
    .send({ from: roles.manufacturer, gas: 999999 })
    // .then(console.log)
    .on('transactionHash', function (hash) {
      handleSetTxhash(hash);
    });
  setManuForm({
    id: 0,
    manufacturerName: "",
    manufacturerDetails: "",
    productName: "",
```

```
productCode: 0,
productPrice: 0,
productCategory: "",
})
} else {
  setfvalid(true);
}
setLoading(false);
};
const handleSetTxhash = async (hash) => {
  await supplyChainContract.methods
    .setTransactionHashOnManufacture(hash)
    .send({ from: roles.manufacturer, gas: 900000 });
};
<div className={classes.FormWrap}>
  <h1 className={classes.pageHeading}>Add Product </h1>
  <Grid container spacing={3}>
    <Grid item xs={12}>
      <TextField
        required
        name="manufacturerName"
        variant="outlined"
        value={manuForm.manufacturerName}
        onChange={handleChangeManufacturerForm}
        label="Manufacturer Name"
        style={{ width: "100%" }}/>
    </Grid>
    <Grid item xs={12}>
      <TextField
        required
        name="productName"
        variant="outlined"
        value={manuForm.productName}
        onChange={handleChangeProductForm}
        label="Product Name"
        style={{ width: "100%" }}/>
    </Grid>
  </Grid>
</div>
```

```
</Grid>
<Grid item xs={6}>
<TextField
  required
  name="productCode"
  variant="outlined"
  value={manuForm.productCode}
  onChange={handleChangeManufacturerForm}
  label="Product Code"
  style={{ width: "100%" }}
/>
</Grid>
<Grid item xs={6}>
<TextField
  required
  name="productPrice"
  variant="outlined"
  value={manuForm.productPrice}
  label="Product Category"
  style={{ width: "100%" }}/>
</Grid></Grid><br />
<p><b style={{ color: "red" }}>{fvalid ? "Please
enter all data" : ""}</b></p>
<Button
  type="submit"
  variant="contained"
  color="primary"
  onClick={handleSubmitManufacturerForm}>
  SUBMIT</Button>
<br /><br /></div>
```

```
</>
)} </Navbar ></>;
}
```

6.1.4 Seller.js

```
import React from "react";
export default function PurchaseThirdParty(props) {
  const classes = useStyles();
  const supplyChainContract = props.supplyChainContract;
  const { roles } = useRole();
  const navItem = [
    ["Buy Product", "/ThirdParty/allProducts"],
    ["Receive Product", "/ThirdParty/receive"],
    ["Ship Products", "/ThirdParty/ship"],
  ];
  React.useEffect(() => {
    (async () => {
      setLoading(true);
      const cnt = await supplyChainContract.methods
        .fetchProductCount().call();
      setCount(cnt);
    })();
    (async () => {
      const arr = [];
      for (var i = 1; i < count; i++) {
        const prodState = await supplyChainContract.methods
          .fetchProductState(i)
          .call();
        if (prodState === "0") {
```



```
const prodData = [];  
const a = await supplyChainContract.methods  
.fetchProductPart1(i, "product", 0)  
.call();  
const b = await supplyChainContract.methods  
.fetchProductPart2(i, "product", 0)  
.call();  
const c = await supplyChainContract.methods  
.fetchProductPart3(i, "product", 0)  
.call();  
prodData.push(a);  
prodData.push(b);  
prodData.push(c);  
arr.push(prodData);  
}  
}  
return (  
<div classname={classes.pageWrap}>  
<Navbar pageTitle={"Seller"} navItems={navItem}>  
{loading ? (  
<Loader />  
) : (  
<TableCell className={classes.TableHead} align="left">  
Universal ID  
</TableCell>  
<TableCell className={classes.TableHead} align="center">  
Product Code  
</TableCell>  
<TableCell className={classes.TableHead} align="center">  
Manufacturer
```

```
</TableCell>
<TableCell className={classes.TableHead} align="center">
  Manufacture Date
</TableCell>
<TableCell className={classes.TableHead} align="center">
  Product Name
</TableCell>
<TableCell
  className={clsx(
    classes.TableHead,
    classes.AddressCell
  )}
  align="center">Owner
</TableCell><TableCell
  className={clsx(classes.TableHead)}
  align="center">Buy
</TableCell></TableRow></TableHead><TableBody>
{allProducts.length !== 0 ? (
  key={prod[0][0]}>
<TableCell
  className={classes.TableCell}
  component="th"
  align="left"
  align="center"
  onClick={() => handleClick(prod)}>
{d.toDateString() + " " + d.toTimeString()}
</TableBody>
</Table>
</TableContainer><TablePagination
  rowsPerPageOptions={[10, 25, 100]}
```

```
component="div"
count={allProducts.length}
rowsPerPage={rowsPerPage}
page={page}
onChangePage={handleChangePage}
onChangeRowsPerPage={handleChangeRowsPerPage}
/>
</Paper></div></>
)}
</Navbar>
</div>
);}
```

6.1.5 Delivery Hub.js

```
import React, { useState } from "react";
export default function ReceiveDeliveryHub(props) {
  const supplyChainContract = props.supplyChainContract;
  const { roles } = useRole();
  const [count, setCount] = React.useState(0);
  const [allReceiveProducts, setAllReceiveProducts] =
    React.useState([]);
  const [modalData, setModalData] = useState([]);
  const [open, setOpen] = useState(false);
  const classes = useStyles();
  const [loading, setLoading] = React.useState(false);
  const navItem = [
    ["Receive Product", "/DeliveryHub/receive"],
    ["Ship Product", "/DeliveryHub/ship"],
  ];
}
```

```
const [alertText, setalertText] = React.useState("");
React.useEffect(() => {
  (async () => {
    setLoading(true);
    (async () => {
      const arr = [];
      for (var i = 1; i < count; i++) {
        const handleSetTxhash = async (id, hash) => {
          await supplyChainContract.methods
            .setTransactionHash(id, hash)
            .send({ from: roles.manufacturer, gas: 900000 });
        };
        const handleReceiveButton = async (id, long, lat) => {
          try {
            await supplyChainContract.methods
              .receiveByDeliveryHub(parseInt(id), long, lat)
              .send({ from: roles.deliveryhub, gas: 1000000 })
              .on("transactionHash", function (hash) {
                handleSetTxhash(id, hash);
              });
            setCount(0);
            setOpen(false);
          } catch {
            setalertText("You are not the owner of the Product");
            const handleChangeRowsPerPage = (event) => {
              setRowsPerPage(+event.target.value);
              setPage(0);
            };
          }
          const handleClose = () => setOpen(false);
          const handleClick = async (prod) => {
```

```
await setModalData(prod);
setOpen(true);
};
return (
<div classname={classes.pageWrap}>
<Navbar pageTitle={"Delivery Hub"} navItems={navItem}>
{loading ? (
<Loader />
handleClose={handleClose}
handleReceiveButton={handleReceiveButton}
aText={alertText}/>
<h1 className={classes.pageHeading}>Products To be
Received </h1>
<h3 className={classes.tableCount}>
Total : {allReceiveProducts.length}
</h3></div>
<Paper className={classes.TableRoot}>
<TableContainer className={classes.TableContainer}>
<Table stickyHeader aria-label="sticky table">

className={clsx(
classes.TableHead,
classes.AddressCell
)}
align="center">Owner
</TableCell><TableCell
className={clsx(classes.TableHead)}
align="center">
RECEIVE
</TableCell></TableRow></TableHead><TableBody>
```

```
{allReceiveProducts.length !== 0 ? (
  allReceiveProducts
    .slice(
      page * rowsPerPage,
      page * rowsPerPage + rowsPerPage
    )
    .map((prod) => {
      const d = new Date(parseInt(prod[1][0] * 1000));
      return (
        <TableRow
          hover
          role="checkbox"
        >
        <TableCell
          className={clsx(classes.TableCell)}
          align="center">
        <Button
          type="submit"
          variant="contained"
          color="primary"
          component="div"
          count={allReceiveProducts.length}
          rowsPerPage={rowsPerPage}
          page={page}
          onChangePage={handleChangePage}
          onChangeRowsPerPage={handleChangeRowsPerPage}
        />
      </Paper></div></Navbar></div>
    );}
```

6.1.6 Customer.js

```
import React, { useState } from "react";
export default function ReceiveCustomer(props) {
  const supplyChainContract = props.supplyChainContract;
  const { roles } = useRole();
  const [count, setCount] = React.useState(0);
  const [allReceiveProducts, setAllReceiveProducts] =
    React.useState([]);
  const [modalData, setModalData] = useState([]);
  const [open, setOpen] = useState(false);
  const classes = useStyles();
  const [loading, setLoading] = React.useState(false);
  const navItem = [
    ["Purchase Product", "/Customer/buy"],
    ["Receive Product", "/Customer/receive"],
    ["Your Products", "/Customer/allReceived"],
  ];
  const [alertText, setalertText] = React.useState("");
  React.useEffect(() => {
    (async () => {
      setLoading(true);
      (async () => {
        const arr = [];
        for (var i = 1; i < count; i++) {
          const prodState = await supplyChainContract.methods
            .fetchProductState(i)
            .call();
          if (prodState === "7") {
            const prodData = [];
            const a = await supplyChainContract.methods
```

```
.fetchProductPart1(i, "product", 0)
.call();
const b = await supplyChainContract.methods
.fetchProductPart2(i, "product", 0)
.call();
const c = await supplyChainContract.methods
.fetchProductPart3(i, "product", 0)
.setLoading(false);
})();
}, [count]);
const handleReceiveButton = async (id) => {
try {
await supplyChainContract.methods
.receiveByCustomer(parseInt(id))
.send({ from: roles.customer, gas: 1000000 })
.on("transactionHash", function (hash) {
handleSetTxhash(id, hash);
});
const [page, setPage] = React.useState(0);
const [rowsPerPage, setRowsPerPage] = React.useState(10);
const handleChangePage = (event, newPage) => {
setPage(newPage);
};
const handleChangeRowsPerPage = (event) => {
setRowsPerPage(+event.target.value);
setPage(0);
};
const handleClose = () => setOpen(false);
const handleClick = async (prod) => {
await setModalData(prod);
```



```
setOpen( true );
};
return (
<div classname={ classes .pageWrap}>
<Navbar pageTitle={"Customer"} navItems={navItem}>
{loading ? (
<Loader />
) : (
<TableRow>
<TableCell className={ classes .TableHead} align="left">
Universal ID</TableCell>
<TableCell className={ classes .TableHead} align="center">
Product Code</TableCell>
<TableCell className={ classes .TableHead} align="center">
Manufacturer </TableCell>
<TableCell className={ classes .TableHead} align="center">
Manufacture Date </TableCell>
<TableCell className={ classes .TableHead} align="center">
Product Name</TableCell><TableCell
className={ clsx (
classes .TableHead ,
classes .AddressCell
)} align="center">Owner
</TableCell><TableCell
className={ clsx ( classes .TableHead )}
align="center">RECEIVE
align="center"
onClick={() => handleClick( prod)}>
{prod[0][4]} </TableCell><TableCell
align="center"
```

```
onClick={() => handleClick(prod)}>
{d.toString() + " " + d.toString()}
</TableCell><TableCell
className={classes.TableCell}
align="center"
onClick={() => handleClick(prod)}>
{prod[1][1]}
</TableCell><TableCell
className={clsx(
classes.TableCell, classes.AddressCell
)}
align="center"
onClick={() => handleClick(prod)}>
{prod[0][2]} </TableCell><TableCell
className={clsx(classes.TableCell)}
align="center">
<Button
type="submit
/></Paper>
</div>
</Navbar>
</div>
);
}
```

6.2 SCREEN SHOTS

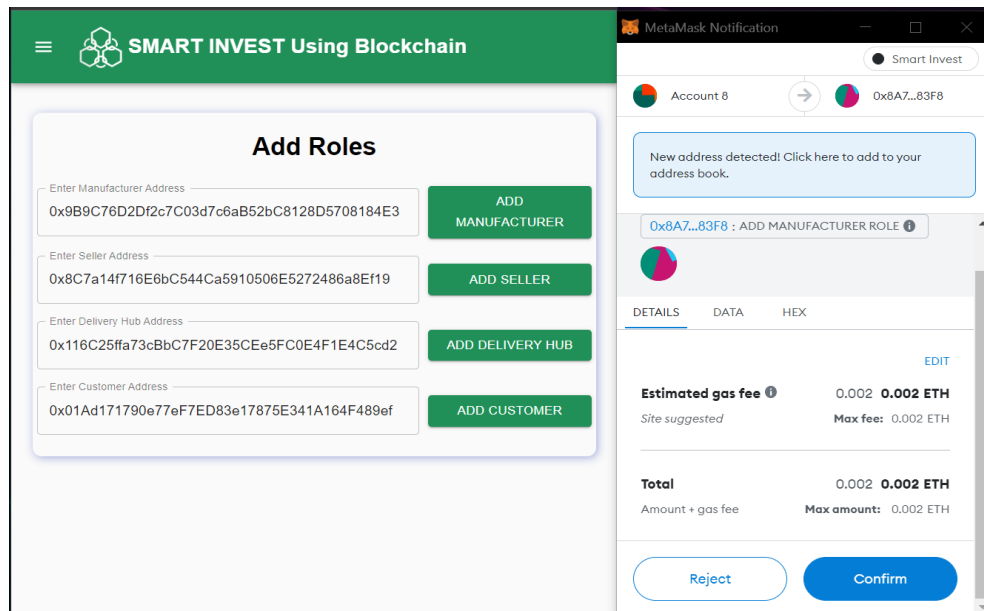


Figure 7.1: Assign roles

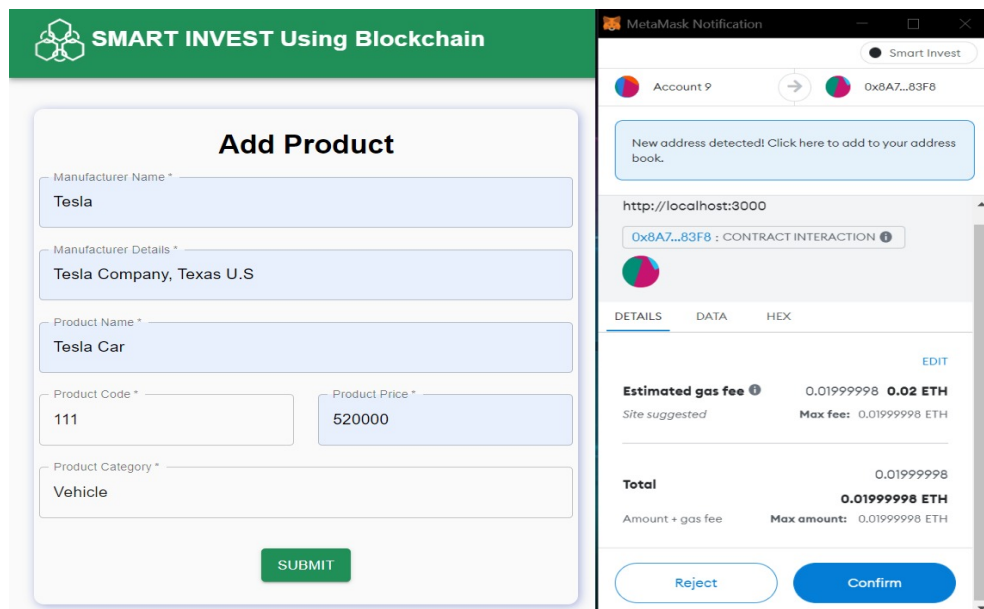


Figure 7.2: Product register

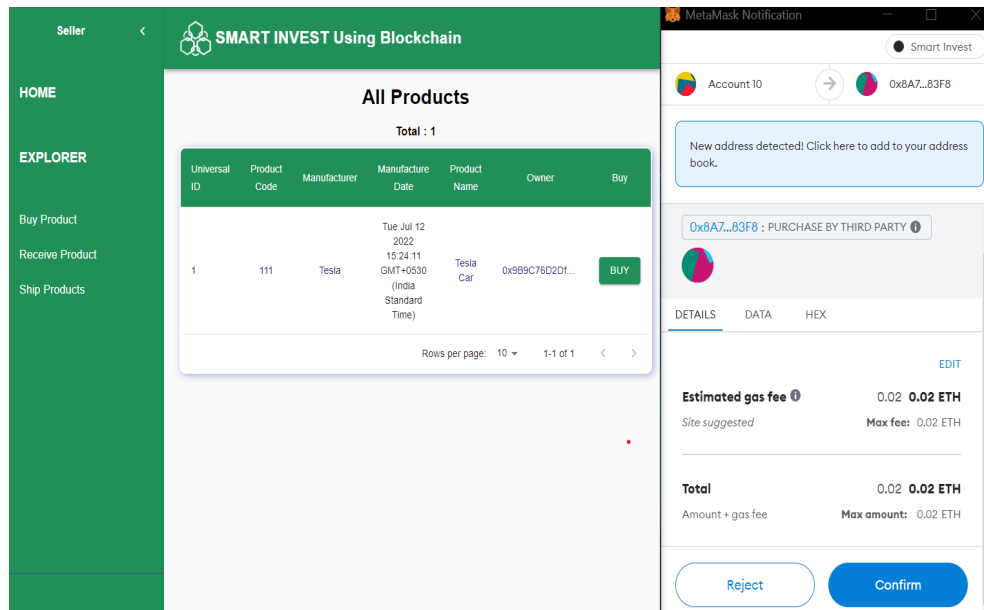


Figure 7.3: Buy products from manufacturer

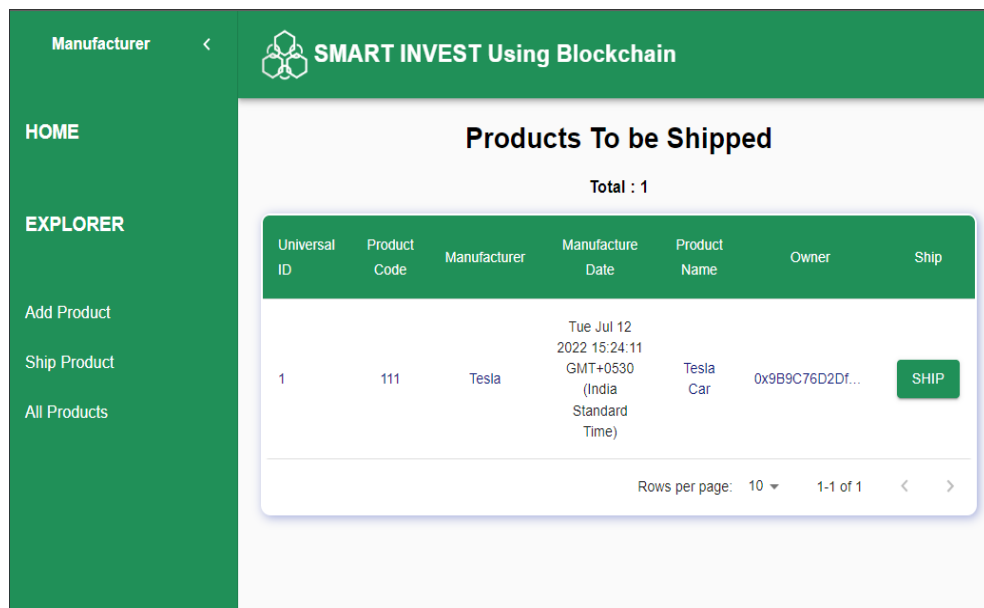


Figure 7.4: Ship product to seller

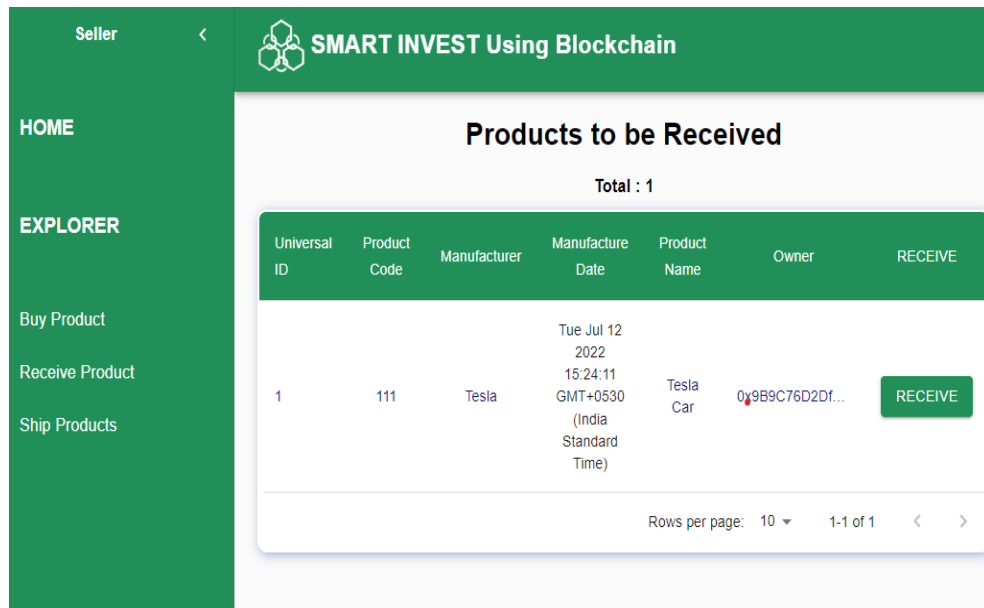


Figure 7.5: Receive product from manufacturer

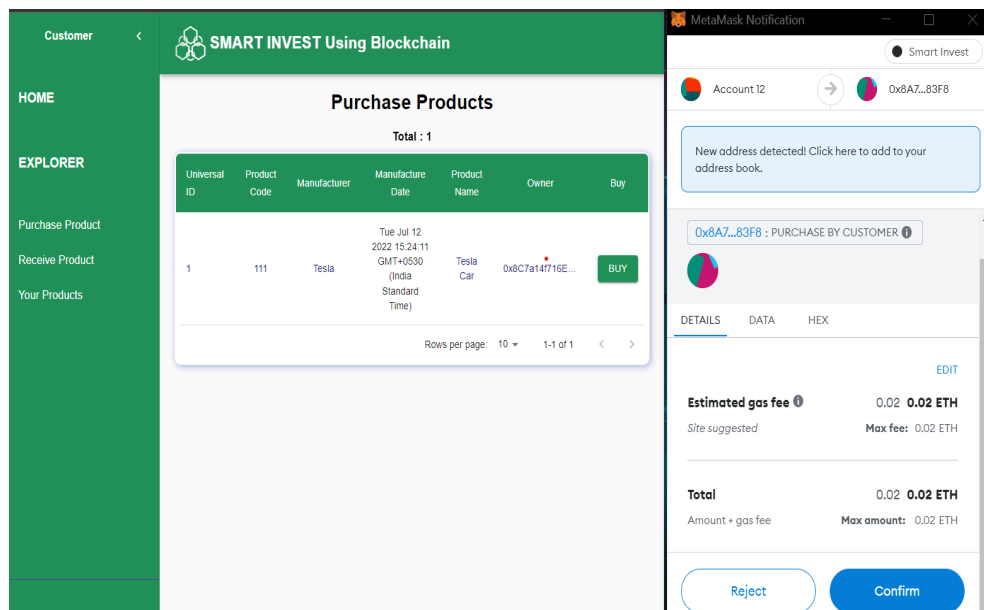
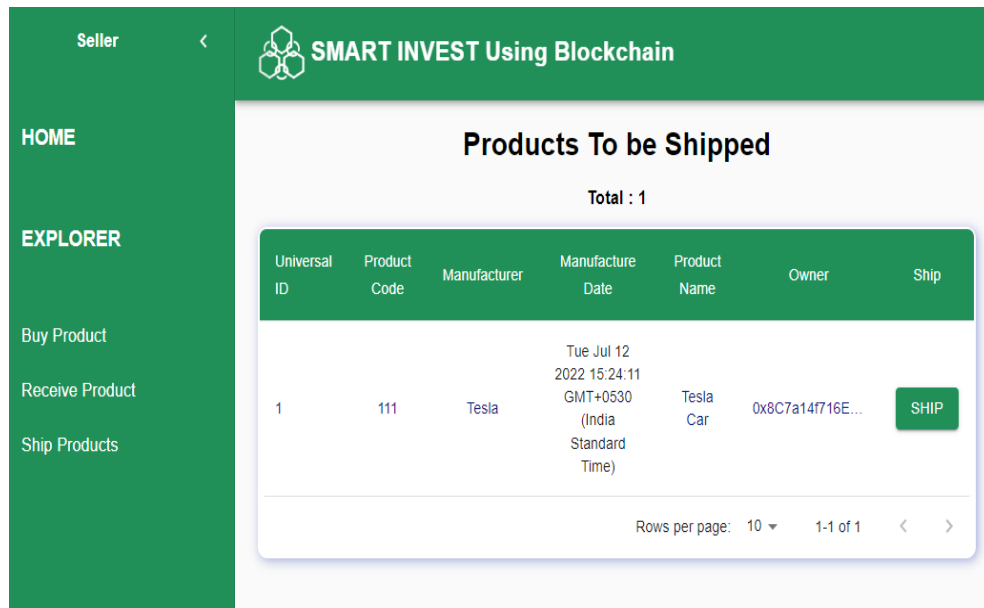


Figure 7.6: Customer purchase product from seller

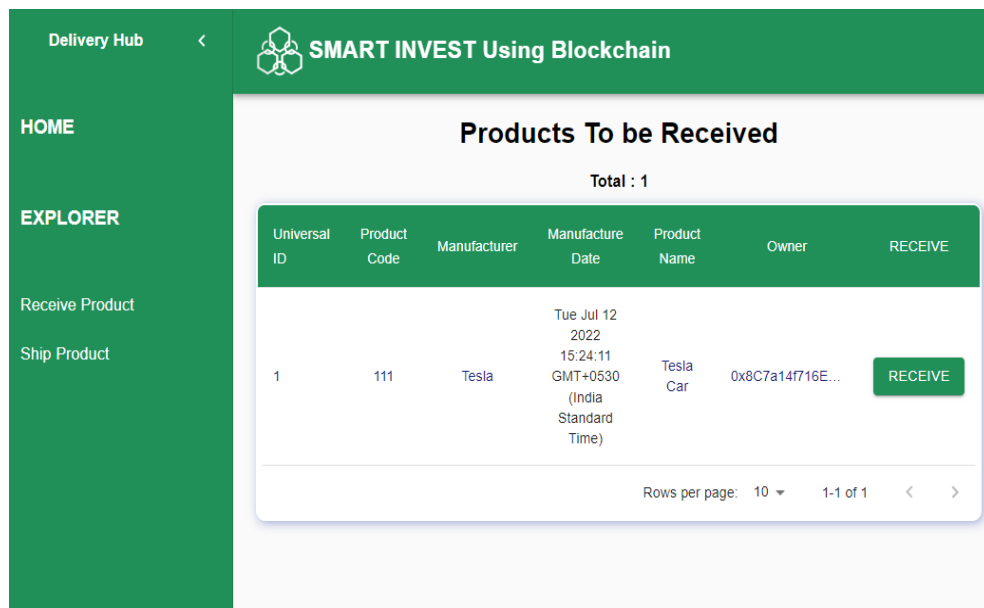


The screenshot shows the 'Seller' interface of the 'SMART INVEST Using Blockchain' system. The left sidebar contains a 'HOME' section and an 'EXPLORER' section with options: 'Buy Product', 'Receive Product', and 'Ship Products'. The main content area is titled 'Products To be Shipped' and shows a total of 1 product. A table lists the product details, and a 'SHIP' button is visible next to the product entry.

Universal ID	Product Code	Manufacturer	Manufacture Date	Product Name	Owner	Ship
1	111	Tesla	Tue Jul 12 2022 15:24:11 GMT+0530 (India Standard Time)	Tesla Car	0x8C7a14f716E...	SHIP

Rows per page: 10 ▼ 1-1 of 1 < >

Figure 7.7: Seller ship product to delivery hub



The screenshot shows the 'Delivery Hub' interface of the 'SMART INVEST Using Blockchain' system. The left sidebar contains a 'HOME' section and an 'EXPLORER' section with options: 'Receive Product' and 'Ship Product'. The main content area is titled 'Products To be Received' and shows a total of 1 product. A table lists the product details, and a 'RECEIVE' button is visible next to the product entry.

Universal ID	Product Code	Manufacturer	Manufacture Date	Product Name	Owner	RECEIVE
1	111	Tesla	Tue Jul 12 2022 15:24:11 GMT+0530 (India Standard Time)	Tesla Car	0x8C7a14f716E...	RECEIVE

Rows per page: 10 ▼ 1-1 of 1 < >

Figure 7.8: Delivery hub receives the product

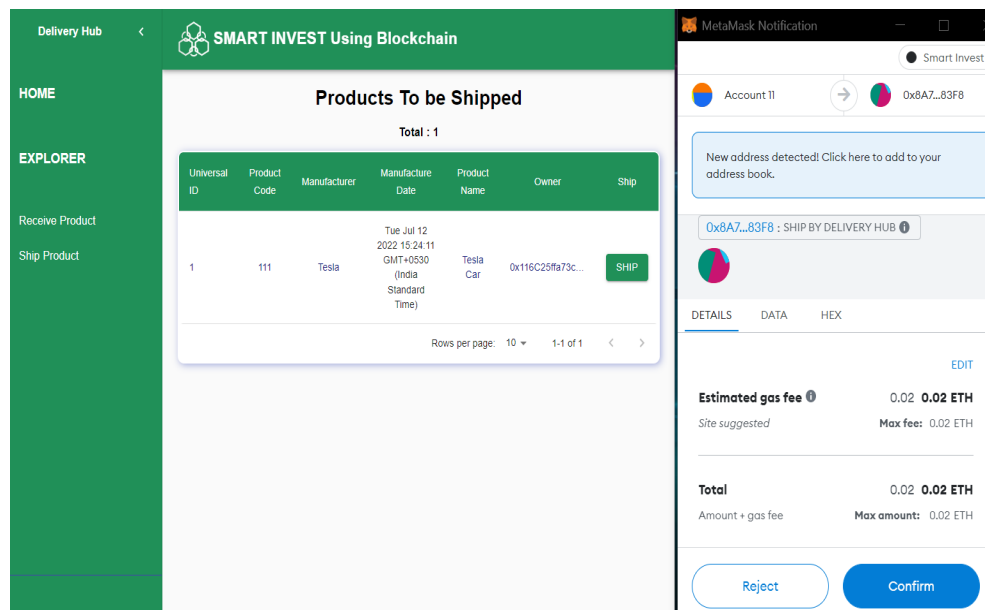


Figure 7.9: Delivery hub ship product to customer

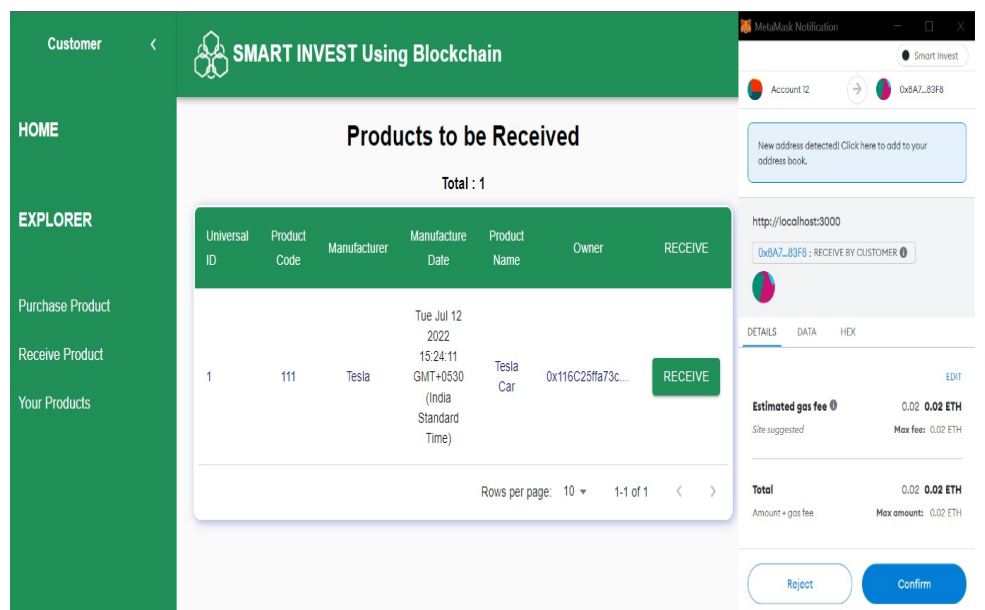
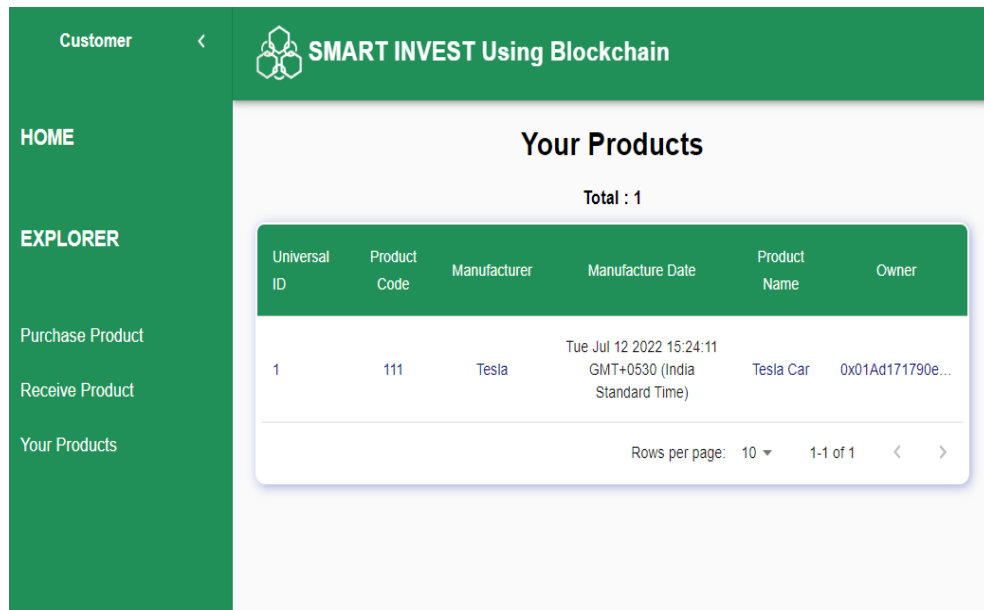


Figure 7.10: Customer receives the product



Customer < SMART INVEST Using Blockchain

HOME

EXPLORER

Purchase Product

Receive Product

Your Products

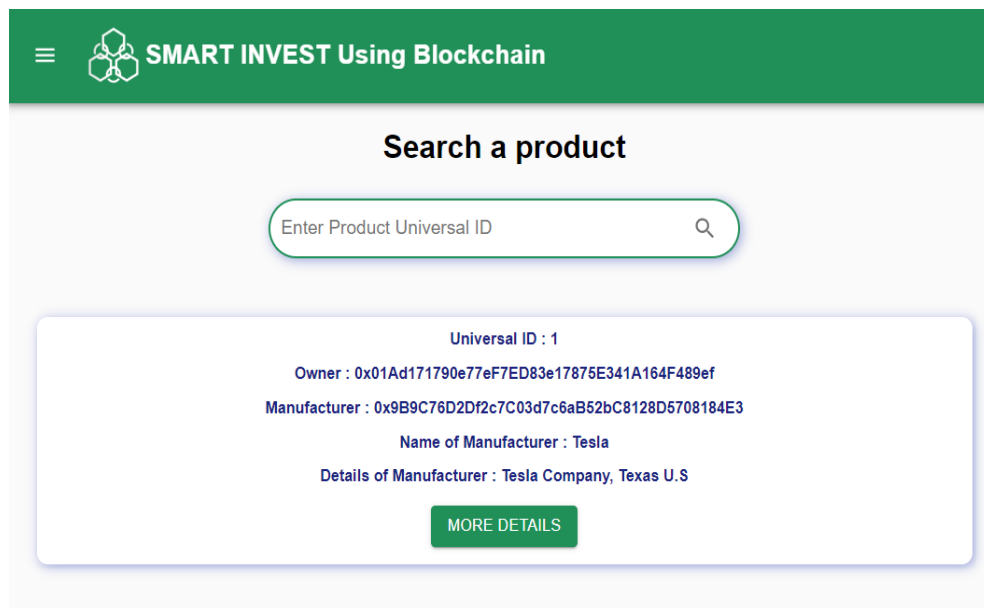
Your Products

Total : 1

Universal ID	Product Code	Manufacturer	Manufacture Date	Product Name	Owner
1	111	Tesla	Tue Jul 12 2022 15:24:11 GMT+0530 (India Standard Time)	Tesla Car	0x01Ad171790e...

Rows per page: 10 1-1 of 1 < >

Figure 7.11: View customer orders



≡ SMART INVEST Using Blockchain

Search a product

Enter Product Universal ID 🔍

Universal ID : 1

Owner : 0x01Ad171790e77eF7ED83e17875E341A164F489ef

Manufacturer : 0x9B9C76D2Df2c7C03d7c6aB52bC8128D5708184E3

Name of Manufacturer : Tesla

Details of Manufacturer : Tesla Company, Texas U.S

MORE DETAILS

Figure 7.12: Search a product

Product History							
Universal ID	Manufacturer	Date	Product Name	Price	Owner	Last Action	Details
1	Tesla	Tue Jul 12 2022 15:24:11 GMT+0530 (India Standard Time)	Tesla Car	520000	0x9B9C76D2Df2c7...	Manufactured	DETAILS
1	Tesla	Tue Jul 12 2022 15:24:11 GMT+0530 (India Standard Time)	Tesla Car	520000	0x9B9C76D2Df2c7...	Bought By Seller	DETAILS
1	Tesla	Tue Jul 12 2022 15:24:11 GMT+0530 (India Standard Time)	Tesla Car	520000	0x9B9C76D2Df2c7...	Shipped From Manufacturer	DETAILS
1	Tesla	Tue Jul 12 2022 15:24:11 GMT+0530 (India Standard Time)	Tesla Car	520000	0x8C7a14f716E6b...	Received By Seller	DETAILS
1	Tesla	Tue Jul 12 2022 15:24:11 GMT+0530 (India Standard Time)	Tesla Car	520000	0x8C7a14f716E6b...	Bought By Customer	DETAILS
1	Tesla	Tue Jul 12 2022 15:24:11 GMT+0530	Tesla Car	520000	0x8C7a14f716E6b...	Shipped By	DETAILS

Figure 7.13: View product history

Invoice Details

Universal ID:	1
Owner:	0x01Ad171790e77eF7ED83e17875E341A164F489ef
Manufacturer:	0x9B9C76D2Df2c7C03d7c6aB52bC8128D5708184E3
Name of Manufacturer:	Tesla
Manufactured date:	Tue Jul 12 2022 15:24:11 GMT+0530 (India Standard Time)
Details of Manufacturer:	Tesla Company, Texas U.S
Product Name:	Tesla Car
Product Code:	111
Product Price:	520000
Product Category:	Vehicle
Product State:	8
Seller Address:	0x8C7a14f716E6bC544Ca5910506E5272486a8Ef19
Delivery Hub Address:	0x116C25ffa73cBbC7F20E35CEe5FC0E4F1E4C5cd2
Customer Address:	0x01Ad171790e77eF7ED83e17875E341A164F489ef
Tx Hash:	

[DOWNLOAD INVOICE](#)

Figure 7.14: Invoice generated

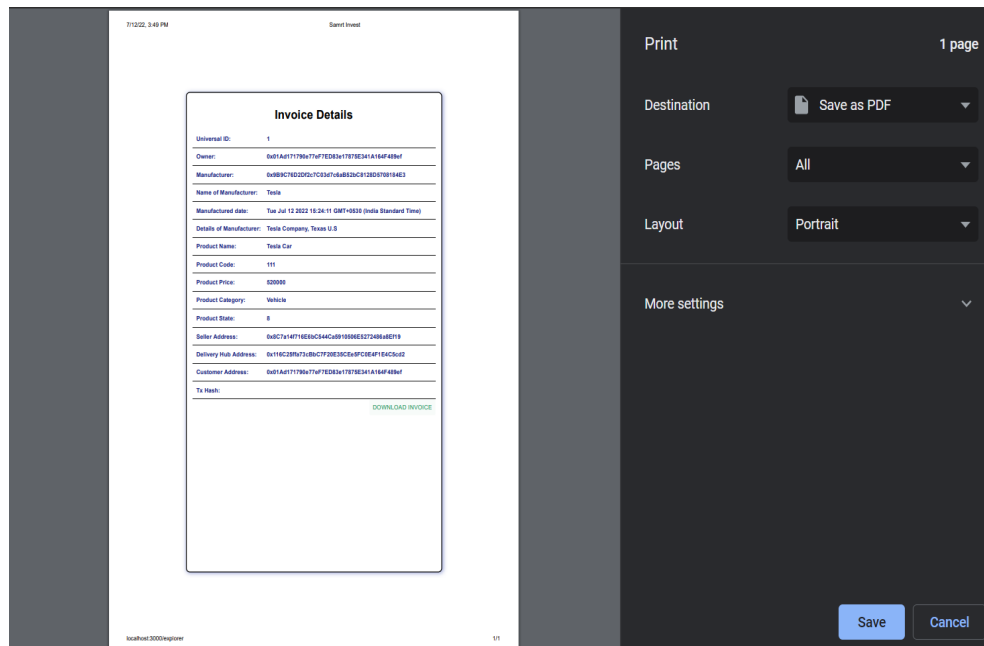


Figure 7.15: Invoice download

Ganache			
ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS
CURRENT BLOCK 26	GAS PRICE 20000000000	GAS LIMIT 6721975	HARDFORK MUIRGLACIER
NETWORK ID 5777	RPC SERVER HTTP://127.0.0.1:7545	MINING STATUS AUTOMINING	WORKSPACE TEST 2
EVENT NAME ReceivedByCustomer			
CONTRACT SupplyChain	TX HASH 0x7b2be006a0b8976c100d8b2b94c6959dcfe89e793c0e56c69b1f1fcd9e95f6b9d	LOG INDEX 0	BLOCK TIME 2022-07-12 15:45:03
EVENT NAME ShippedByDeliveryHub			
CONTRACT SupplyChain	TX HASH 0x1d06d51bba5f67ca5e199c5074c3b8496d3ddb79ec9ea3f984ee8de102933888	LOG INDEX 0	BLOCK TIME 2022-07-12 15:43:01
EVENT NAME ReceivedByDeliveryHub			
CONTRACT SupplyChain	TX HASH 0x3f96a6032dd8df29f55be26745f530d02162dc96a2a46e9c88f6bffe030a5bd9	LOG INDEX 0	BLOCK TIME 2022-07-12 15:42:09
EVENT NAME ShippedByThirdParty			

Figure 7.16: Ganache events and transaction

Chapter 7

REFERENCES

1. <https://www.mhi.org/downloads/learning/cicmhe/blockchain-and-supply-chain-management>
2. Saveen A Abeyratne and Radmehr P Monfared. Blockchain ready manufacturing supply chain using distributed ledger. 2016
3. <https://www.forbes.com/sites/forbestechcouncil/2021/11/08/blockchain/?sh237e034c4e1a>
4. Feng Tian. An agri-food supply chain traceability system for china based on rfid blockchain technology. In Service Systems and Service Management (ICSSSM), 2016 13th International Conference on, pages 1–6. IEEE, 2016.
5. Abey Ratne, Saveen A., Monfared, Radmehr P., “Blockchain ready manufacturing supply chain using distributed ledger”. eSAT; 2321-7308.
6. https://www.tutorialspoint.com/ethereum/ethereum_ganache_blockchain.html