SUPPLY CHAIN MANAGEMENT USING ETHEREUM BLOCKCHAIN

TABLE OF CONTENTS

Certificate	**i**
Declaration	**ii**
Acknowledgement	**iii**
List of Figures	**v**
List of Tables	**vi**
Abstract	**vii**
1. INTRODUCTION	**1**
1.1 Problem Definition	2
1.2 Existing System	3
1.3 Proposed System	3
1.4 Requirements Specification	4
1.4. 1 Software Requirement	4
1.4.2 Hardware Requirements	4
2. LITERATURE SURVEY	**7**
3. METHODOLOGY OF SUPPLYCHAIN USING BLOCKCHAIN	**12**
4. DESIGN AND IMPLEMENTATION	**17**
5. UML DIAGRAMS	**30**
6. RESULTS	**34**
7. CONCLUSION AND FUTURE SCOPE	**37**
REFERENCES	**38**
APPENDIX	**40**

LIST OF FIGURES

Figure 1.1	Ethereum blockchain	5
Figure 1.2	Truffle network	6
Figure 3.1	Smart contracts	13
Figure 3.1.1	Methodology	15
Figure 4.1.1	Smart contracts enabling information availability	18
Figure 4.1.2	Integrated supply chain management	20
Figure 4.2.1	Driving Value in the Supply Chain	22
Figure 4.5.1	Launching Ganache Cli	26
Figure 4.5.2	Compiling Smart Contracts	26
Figure 4.5.3	Truffle Migrate	27
Figure 4.5.4	Truffle Test	28
Figure 4.5.5	MetaMask Confirmation	29
Figure 5.1	Activity diagram	30
Figure 5.2	Sequence diagram	31
Figure 5.3	State diagram	32
Figure 5.4	Data modeling diagram	33
Figure 6.1	DApp User Interface	34
Figure 6.2	Farm Details	34
Figure 6.3	Product Details	35
Figure 6.4	Transaction History	35

LIST OF TABLES

Table-2.1	Literature Survey for Supplychain using blockchain	9

ABSTRACT

SCTS stands for "Supply Chain Tracking System". STCS is a Smart Contract framework to keep tracking of products from the raw materials to the shopping stage. This system is powered by the Ethereum Blockchain, so it's fully distributed, immutable and auditable. Every Product in SCTS is represented in a Smart Contract, which keeps track of actions being made by Ethereum accounts. The community in charge of SCTS (could be a government, a consortium or even a DAO) can register this account as official SCTS Product Handlers, so their identity is stored in the Blockchain being impossible for fake producers or suppliers to cheat and sell products in the name of a registered Brand. Because of this, any customer, government or regulatory institution can scan a QR code in the product packaging, which identifies the Smart Contract representing the exact unit of the Product, or even introduce the address of the Smart Contract, and be able to track and audit all the life of the Product. But there's more. Anyone will be also able to introduce the code of a SCTS Product Handler and see all the actions this individual or institution has performed on SCTS Products, empowering a more transparent and fair global economic system.

Four major issues: traceability and transparency, stakeholder involvement and collaboration, supply chain integration and digitalization, and common frameworks on blockchain-based platforms, are critical for future orientation. Traditional supply chain activities involve several intermediaries, trust, and performance issues. The potential of blockchain can be leveraged to disrupt supply chain operations for better performance, distributed governance, and process automation.

For this project, the implementation of blockchain to facilitate the process of allowing the coffee bean produce to move right from the start of the supply chain (the farmer) to the consumer's hands is shown.

1. INTRODUCTION

In blockchain, decentralization refers to the transfer of control and decision-making from a centralized entity (individual, organization, or group thereof) to a distributed network. Traditional business operations rely heavily on a centralized authority or third parties to promote trust among participants. The emergence of blockchain technology may bring about a rethinking of the design of business operations by virtue of its distributed and decentralized characteristics.

Blockchain, a DLT, refers to a consecutive list of time-stamped records (usually digital transaction data) sequentially linked using cryptography. A peer-to-peer network of participating nodes contributes to the formation and validation of blockchain and manages distributed consensus by network majority. This makes blockchain an immutable, secure, and trustless model where transactions among parties are concerned. Blockchain technology has the potential to rebuild the way businesses conduct their operations. Supply chain management (SCM) is one potential application which could function through a distributed database without third parties.

Supply chain management (SCM) is a pivotal sector of an organization and is described as the supervision and control of all relationships (both upstream and downstream) between manufacturers, suppliers and customers to deliver greater customer value at an economical rate to the supply chain as a whole. SCM operational efficiency is measured through its performance against the greater demand ambiguity, higher supply risk, and increasing competitive intensity. Therefore, increased transparency through better information exchange is imperative. This consistent need for transparent traceability of goods due to inconsistent or unavailable data, lack of interoperability, and limited information on the product’s lifecycle or transport history will continue to present a significant goal for future supply chains. These particular problems in SCM have actually been addressed through the proposition of applying blockchain technology that has shown to counter transparency and security inefficiencies. To do so, we implement a supply chain tracking system which employs blockchain to provide trustable and transparent tracking, which is enabled by a public access to the public Ethereum blockchain.

1.1 Problem Definition

- This project "SUPPLY CHAIN MANAGEMENT USING BLOCKCHAIN" mainly makes use of the Ethereum blockchain. We use smart contracts to deploy them onto the blockchain so that each actor (farmer, producer, transporter) is entirely aware of the movement and state of the product throughout the supply chain.
- The Ethereum blockchain is a community-built technology behind the cryptocurrency ether (ETH) and thousands of applications you can use today. It is the decentralized platform that helps run the smart contracts.
- The main objective of this project is to show why blockchain can be so indispensable to the supply chain industry and to challenge traditional ways of handling product flow from the producer to the consumer.

Project Scope

This project can be useful in many ways:

1. To help unearth existing problems in the supply chain, which are:
    a. Greater demand ambiguity
    b. Higher supply risk
    c. Increasing competitive intensity
2. To increase transparency among the different entities involved in the supply chain and facilitate better information exchange.
3. To improve interoperability between the actors in the supply chain.
4. To provide more information on the product life cycle and transport history.
5. To amplify security within the supply chain so that only the entities involved are exposed to information regarding the product's state, movement and so on.

1.2 Existing System

The supply chain industry has a handful of problems that demand significant attention in order to fuel better productivity throughout the cycle. To be able to efficiently manage customer expectations of a given product can be a challenge. Supply chains also need to manage suppliers and understand how to handle delays and receive orders. Quality and sustainability of the product needs to be understood at every stage of the chain, which requires constant monitoring of the product conditions. All entities of the supply chain need access to the data related to the product as efficiently as possible, which needs a massive infrastructure to do so.

Disadvantage:

1. Lack of transparency between each of the actors in the supply chain.
2. Lack of interoperability between the entities available.
3. Lack of Security of data.

1.3 Proposed System

The proposed system suggests the use of blockchain to fill the gaps left by the existing system, especially in terms of transparency. Blockchain aims to make supply chains transparent and traceable to better protect the end consumer’s interests against counterfeiting, contamination, false claims, and inadequate processes. The actors in the supply chain can also leverage additional visibility to proactively tune their operations in response to upstream or downstream disruptions. Blockchain technology, with its promise of immutability, transparency, and provenance is a natural fit to supply chain problems.

To verify all the transactions taking place within the supply chain, we make use of smart contracts that are deployed on the Ethereum blockchain. It ensures all stakeholders are registered so that all are aware of the information related to the movement, state, etc of the products.

Advantages:

1. Provides transparency between the actors in the supply chain.

2. Improves security of data stored related to the supply chain so as to avoid competitors from learning the ways of the current project.
3. Improves interoperability and hence communication between the entities.

1.4 Requirements Analysis

The following are the requirements for the execution of the project:

1.4.1 Software Requirements

The following are the software requirements for SUPPLY CHAIN MANAGEMENT USING BLOCKCHAIN :

Software: Ethereum, Truffle Framework, Infura

Platform: Ethereum Virtual Machine

Language: Solidity, Javascript, HTML

Packages required:

1) Ganache cli

2) Truffle

3) MetaMask extension

4) IPFS

1.4.2 Hardware Requirements

- OS – Windows 7,8 or 10 (32 or 64 bit)
- RAM – 4GB

Software Tools Used:

Ethereum

Ethereum is software running on a network of computers that ensures that data and small computer programs called smart contracts are replicated and processed on all the computers on the network, without a central coordinator. The vision is to create an unstoppable censorship-resistant self-sustaining decentralized world computer.

Fig 1.1 Ethereum Blockchain

IPFS (InterPlanetary File System)

IPFS is a distributed system for storing and accessing files, websites, applications, and data. When you use IPFS, you don't just download files from someone else — your computer also helps distribute them. IPFS makes this possible for not only web pages but also any kind of file a computer might store, whether it's a document, an email, or even a database record.

Truffle Framework

Truffle is a world-class development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM). Truffle is the most popular development tooling for Ethereum programmers. Easily deploy smart contracts and communicate with their underlying state without heavy client side programming. An especially useful library for the testing and iteration of Ethereum smart contracts.

Infura

Infura provides the tools and infrastructure that allow developers to easily take their blockchain application from testing to scaled deployment - with simple, reliable access to Ethereum and IPFS. A peer-to-peer hypermedia protocol to make the web faster, safer and more open.

Ganache Cli

Ganache CLI is the latest version of TestRPC: a fast and customizable blockchain emulator. It allows you to make calls to the blockchain without the overheads of running an actual Ethereum node. Accounts can be recycled, reset and instantiated with a fixed amount of Ether (no need for faucets or mining).

Fig 1.2 Truffle Network

Solidity

Solidity is an object-oriented, high-level language for implementing smart contracts. Solidity is a curly-bracket language. It is influenced by C++, Python and JavaScript, and is designed to target the Ethereum Virtual Machine (EVM). Solidity is statically typed, supports inheritance, libraries and complex user-defined types among other features.

2. LITERATURE SURVEY

A thorough literature review was carried out in order to acquire a complete grasp of how blockchain can be implemented on the supply chain and how they might be designed efficiently. Several research publications have been read, and the methodologies utilized in those papers have been comprehended.

1. **Blockchain Technology in Supply Chain Management: Preliminary Study**
   Despite significant research, the supply chain management challenges still have a long way to go with respect to solving the issues such as management of product supply information, product lifecycle, transport history, etc. Given the recent rise of blockchain technology in various industrial sectors, our work explores the issues prevalent in each stage of the supply chain and checks their candidacy for the implementation using blockchain technology. The analysis is performed in terms of the characteristics of trust and decentralization with respect to forming a generalized framework. The main contribution of this work is to create a conceptual overview of the areas where blockchain integrates with supply chain management in order to benefit further research and development.

2. **Study on Supply Chain Management using Blockchain Technology**
   The rapid increase in use of new technology known as "block chain technologies" in various fields brings massive benefits to the user with having concerns on decentralization, transparency, and security. Information like data, transactions details, time, and money are recorded and stored end to end within the system. This ensures the security and protects data from getting corrupt. Various patterns and algorithms all met to ensure the security level. Block chain technologies are mostly used in automations like E-voting, supply chain etc. This information can be attacked or corrupted by hackers, so they are highly under risk. To ensure transparency inside the system and security outside the system, block chain technologies are used as a major tool. A detailed study has been presented on the various types of techniques and methods that are used in the field of

supply chain under block chain technologies.

3. **A Platform-independent, Generic-purpose, and Blockchain-based Supply Chain Tracking**

   Supply Chain Tracking (SCT) is considered as a major challenge for stakeholders of heterogeneous production, processing, transporting, storing, and selling systems. Studies on solutions in SCT reveal that many centralized SCT applications lack a design, which can support or even enable main features of a reliable, transparent, and publicly accessible SCT system for end users. This work demonstrates the implementation of a SCT application which employs SC on the Ethereum blockchain (BC). This Decentralized Application (Dapp) provides a hardware-and platform-independent approach that flexibly enables multiple object combinations and transformations to be tracked with a use case-agnostic design and utilization.

4. **When blockchain meets supply chain: A systematic literature review on current development and potential applications**

   This study aims to explore the current status, potential applications, and future directions of blockchain technology in supply chain management. A literature survey, along with an analytical review, of blockchain-based supply chain research was conducted to better understand the trajectory of related research and shed light on the benefits, issues, and challenges in the blockchain-supply-chain paradigm. A selected corpus comprising 106 review articles was analyzed to provide an overview of the use of blockchain and smart contracts in supply chain management. The diverse industrial applications of these technologies in various sectors have increasingly received attention by researchers, engineers, and practitioners. Four major issues: traceability and transparency, stakeholder involvement and collaboration, supply chain integration and digitalization, and common frameworks on blockchain-based platforms, are critical for future orientation. Traditional supply chain activities involve several intermediaries, trust, and performance issues. The potential of blockchain can be leveraged to disrupt supply chain operations for better performance, distributed governance, and process automation. This study contributes to the comprehension of blockchain applications in supply chain management and provides a blueprint for these applications from the perspective of literature analysis. Future efforts regarding technical adoption/diffusion, block-supply chain integration, and their social impacts were highlighted to enrich the research scope.

5. **Blockchains and the supply chain: Findings from a broad study of practitioners**
   Blockchain is an emergent technology that has attracted practitioner attention in the supply chain domain. Multiple motivators lead companies to incorporate blockchain technology into their supply chain. However, a number of barriers and challenges may impede the successful adoption of blockchains. The purpose of this paper is to explore how a variety of motivators and barriers are perceived by different companies from different industries. This paper summarizes survey data gathered from 173 respondents associated with the Association of Supply Chain Management (ASCM), formerly APICS. Summary statistics on blockchain adoption by various types of companies are provided to help practitioners benchmark current practice. The paper presents preliminary findings on these dimensions with some insights provided.

Table 2.1 Literature survey for Supply chain using blockchain

S.No	Name of Author	Title of Paper	Year	Method Used	Limitations
1.	Soha Yousuf, Davor Svetinovic	Blockchain Technology in Supply Chain Management: Preliminary Study	2021	Analysis is performed in terms of the characteristics of trust and decentralization with respect to forming a generalized framework.	Lacks a formal systematic review covering the rest of the SCM stages to formulate a complete framework
2.	Yaswanth Raj, Sowmiya B	Study on Supply Chain Management using Blockchain Technology	2020	Various types of techniques and methods that are used in the field of supply chain under blockchain technologies	Needs evaluation with the existing techniques and how far they are scalable with different blockchain technologie s and methods improved
3.	Sina Rafati Niya, Danijel Dordevic, Atif Ghulam Nabi, Tanbir Mann, Burkhard Stiller	A Platform-independ ent, Generic-purpose, and Blockchain-based Supply Chain Tracking	2020	A SCT application which employs SC on the Ethereum blockchain (BC). Uses DApp ASPIR	Needs evaluation with the existing techniques and how far they are scalable with

					different blockchain technologies and methods.
4.	Shuchih E. Chang and YiChian Chen	When blockchain meets supply chain: A systematic literature review on current development and potential applications	2019	This study aims to explore the current status, potential applications, and future directions of blockchain technology in supply chain management.	Needs evaluation with the existing techniques and how far they are scalable with different blockchain technologies and methods.
5.	Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis	Blockchains and the supply chain: Findings from a broad study of practitioners	2019	Association of Supply Chain Management (ASCM)	Need to investigate the blockchain implementation evolution in the supply chain.

3. METHODOLOGY OF SUPPLY CHAIN MANAGEMENT USING BLOCKCHAIN

A private blockchain-based solution for facilitating information exchange, such as inventory between retailers and suppliers in a network is proposed. The solution adopts blockchain technology to promote transparency, encourage trust, and enhance information security between the stakeholders in the supply chain. Moreover, the system develops smart contracts and uses decentralized storage technology in the Ethereum network. This would allow registered retailers to track the inventory level in real time for each supplier in the network.

The roles of each stakeholder and component in the proposed system is described below.

- **Supplier:**

  The supplier researches the market, determines the necessary goods, and calculates the required amount to fulfill consumer demand while ensuring that the products meet regulatory requirements. Simultaneously, a supplier must be prepared to anticipate incidents that could result in commodity shortages, such as raw material delays during natural disasters or manufacturing disruptions during pandemics.

- **Retailer:**

  In a supply chain, a major responsibility of the retailer is to provide consumers with a variety of goods sourced from numerous producers, wholesalers, dealers, or suppliers. As a result, the retailer is the ultimate step in the supply chain, connecting vendors to consumers.

- **Ethereum Smart Contracts:**

  A code that is created to automatically execute functions based on the terms and conditions decided upon by network stakeholders. It works as a virtual agent that verifies transactions without the need for third-party intervention. Our proposed approach consists of four smart contracts, each of which is focused on a specific task. As depicted in Fig 3.1, the registration contract focuses on registering all network members, while the inventory level contract specifies the amount of inventory left for each supplier as well as the product descriptions offered by suppliers. The order management contract deals with handling and managing orders, and the reputation contract ranks vendors based on their product quality, distribution efficiency, and honesty.

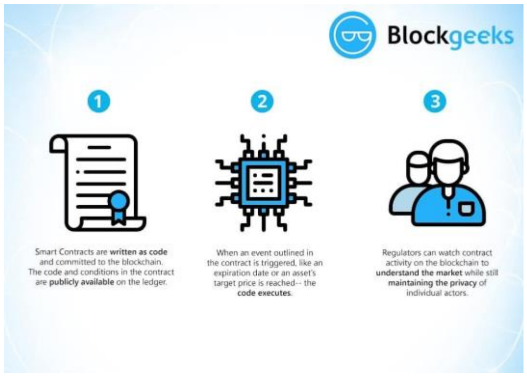

Blockgeeks
1
2
3
Smart Contracts are written as code and committed to the blockchain. The code and conditions in the contract are publicly available on the ledger.
When an event outlined in the contract is triggered, like an expiration date or an asset's target price is reached-- the code executes.
Regulators can watch contract activity on the blockchain to understand the market while still maintaining the privacy of individual actors.

Fig 3.1 Smart Contracts

- **Decentralized Storage Technology:**

  Using a peer-to-peer distributed file system like Filecoin or Interplanetary File System (IPFS) makes it easier to link all nodes in a network to the same file system [33]. Authorized network users can use IPFS to store vast volumes of data and use blockchain transactions to store the permanent IPFS hashes, which are time-stamped and encrypted using cryptographic technologies. This is advantageous because it eliminates the need for consumers to store data on the chain. As a result, the combination of blockchain and distributed storage is advantageous since nodes in the network can only store indexed data, making it easier to find where data is stored.

3.1 METHODOLOGY

The proposed system was built using the information as shown in Fig. 3.2. This system is then captured as a series of functions and events highlighting the interaction between each stakeholder and contracts. The code is then designed and tested.

Initially, each stakeholder in the network is registered using their Ethereum address in the registration contract. Retailers and suppliers are registered by calling the functions registerRetailer() and registerSupplier() respectively. The system then allows the registered suppliers to upload their inventory information, such as product numbers, quantities, descriptions, etc., by calling the additem() function found in the inventory level contract. This triggers an event for all the stakeholders that a particular item has been uploaded successfully. The registered retailers use this information to make a new purchase order stating the product and quantity needed. Then an event is triggered depending on the status of the order, alerting the stakeholders whether the order is accepted or rejected.

When an order is placed with a particular supplier, the number of items in the inventory gets deducted from the inventory level contract automatically. Thus, whenever retailers make a successful order, they are allowed to rate the performance of their suppliers using a reputation contract. The retailers call this contract and provide feedback, which automatically calculates the reputation score of the suppliers and simultaneously notifies all stakeholders that the reputation score has been updated. This allows retailers to work with trusted suppliers while keeping suppliers aware that their actions get recorded in an immutable distributed ledger.

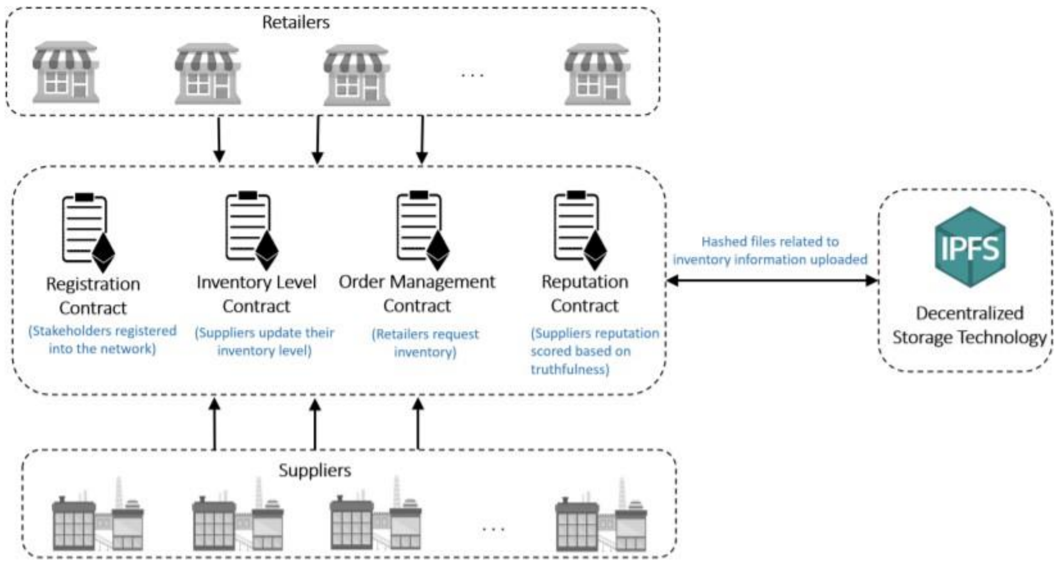

Retailers
Registration Contract
(Stakeholders registered into the network)
Inventory Level Contract
(Suppliers update their inventory level)
Order Management Contract
(Retailers request inventory)
Reputation Contract
(Suppliers reputation scored based on truthfulness)
Hashed files related to inventory information uploaded
IPFS
Decentralized Storage Technology
Suppliers

Fig 3.1.1 Methodology

3.2 PRIVACY, CONFIDENTIALITY AND AUTHORIZATION

Preserving the privacy of the stakeholders is vital while successfully communicating and interacting using the blockchain network functions. Therefore, we have chosen a permissioned blockchain. Each type of permissioned blockchain has its way for maintaining privacy and confidentiality amongst its participants. For instance, Hyperledger Fabric is a permissioned network that is known for its plug and play design. Its versatile architecture relies on Membership Service Providers (MSPs) as well as certificate authorities to provide roles for the participants as well as identities and communication rights. Therefore, authorization is achieved through members allowed in after registering through the MSPs. Moreover, hyperledger fabric also provides channels which are ways for certain stakeholders to subscribe in and communicate privately. Communication within a channel is transparent only to its participants. In addition, any information communicated within a channel can also be encrypted to achieve confidentiality. On the other hand, other permissioned networks such as the java based Ethereum permissioned Besu network does not support permissioning itself. However, it relies on Ethereumcompatible wallet called EthSigner to manage the private keys of its participants. Furthermore, Orion is used to ensure private communication is possible between a set of users. There are many private blockchain based networks that can be used and our solution is easily adaptable to any of them.

We have chosen a permissioned Ethereum network but our solution can also be implemented on Hyperledger Fabric and others.

Using a permissioned network allows the stakeholders to have roles on the ledger and private communication between certain entities based on the need. The communication is signed using the keys of the participants and encrypted if needed. Therefore, this ensures trust and enforces confidentiality as well as privacy. Only authorized entities can communicate in private channels or network subsets and groups

The registered participating entities can execute functions in the smart contracts that they are authorized to only, based on their roles. Unregistered entities cannot take part in executing any transaction. The transaction logs stored in the distributed ledger are accessible to the registered members only. Based on the type of the permissioned network, either channels or groups are created for private transactions taking place between a set of registered members. Hence, transaction logs of private transactions are accessible by members of the channel or group respectively. Therefore, each registered member can only execute their own transactions based on their affiliated role and can only access logs that they are permitted to. Our solution leverages the intrinsic features of the blockchain as well as the permissioned features of the private blockchain.

4. DESIGN AND IMPLEMENTATION

This section presents the implementation details of the permissioned blockchain-based solution for promoting inventory sharing among retailers and suppliers in a network. The system overview presented previously was implemented on a test Ethereum blockchain. The Remix IDE was exploited to deploy smart contracts using the Solidity language.

There are four different types of smart contracts in the solution: registration smart contract, inventory level smart contract, order management smart contract, and reputation smart contract. These smart contracts are used for different purposes, as explained below.

- **Registration smart contract:**The owner of the system deploys this smart contract. The registration smart contract is only implemented once and is used to handle all stakeholder addresses. The Ethereum Address is a 20- byte identifier that is used to identify each stakeholder (EA). This smart contract includes both retailers and suppliers. In addition, it is used by other smart contracts to validate the identity of the entity through its EA. The registration smart contract registers members and only allows authenticated individuals with predefined roles from accessing the smart contract functions and onchain transactions. Our modifiers restrict the registered members based on their roles.

- **Inventory level smart contract:** When a registered supplier’s inventory level changes, this smart contract is updated. Each product is known by its EA, and once the contract is established, all of the product data is contained in it. The type of commodity sold by the seller, its name, the content and material hash file, the CE certificate hash file, the overall quantity initially purchased, and the remaining quantity after distribution to different retailers are all included. Furthermore, this smart contract keeps track of each supplier’s remaining inventory, making it easier for retailers to place orders accordingly. Every adjustment to each remaining quantity is immediately recorded and reflected on the blockchain. The registration smart contract fixed EA is also saved for comparison in order to verify the stakeholder identities.

- **Order management smart contract:** Registered retailers contact suppliers to place their orders for individual products and quantities. If the current supplier with whom the retailer is working does not have the quantity required, the retailer can place an order with another registered supplier in the network. These procedures are carried out by this smart contract, which declares new order placements. Before placing the order, the smart contract verifies the identities of the participants. Following that, all modifications to the orders, including order confirmation and execution, are registered in this smart contract. A special 32-byte number identifies each order here. This identifier is linked to the commodity ID, the quantity requested, the ordering entity address, and the order receiver address.

- **Reputation smart contract:** This smart contract is dedicated to providing trust in the integrity of the suppliers. Trusting the suppliers provides retailers with confidence in the system and the overall quality of service. The owner of the system deploys this contract, just like the registration smart contract. The retailers can access it to learn about the reputation score of the suppliers and to rate their interactions with them.

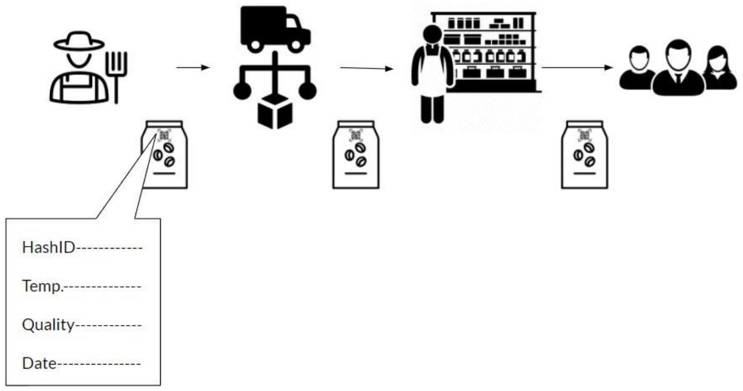


Fig 4.1.1 Smart contracts enabling information availability

The proposed approach for inventory sharing comprises several smart contracts that contribute to the different processes in this solution. As shown in Fig 4.1.1, it presents how the suppliers can add items to their inventory and increase their stock of existing items. Assuming the supplier is already registered in the Registration smart contract, the supplier deploys only once a different type of smart contract, called the Inventory smart contract. This is responsible for keeping track of the supplier inventory and modifying it. Naturally, only the Ethereum client that deploys such a smart contract can modify it. When the suppliers want to add an item, they enter its number, quantity, and price. However, if the item already exists, the supplier enters the item number and the quantity to be added. In addition, the price of an item can be adjusted through this smart contract. Finally, the newly updated quantity is sent out as an event for retailers to be updated.

All retailers can contact the Order Management smart contract to submit their purchase order requests. This smart contract manages the communication between the retailers and the suppliers and their inventory smart contracts. The retailer provides the Ethereum address of the inventory smart contract it wishes to purchase from, along with the item number and the quantity ordered. The Order Management smart contract logs this request and checks whether the provided Ethereum address corresponds to a legitimate inventory smart contract. Consequently, the stock in this inventory is verified to have enough items for this order. The smart contract that manages this inventory then modifies the quantity available to account for the purchase order. Only the owner of the inventory smart contract is authorized to modify the quantity of the items available. The supplier is informed of this transaction through an event that includes the request details. The transaction is reverted at any point in the event of errors in the order, such as incorrect inventory addresses or unavailable items.

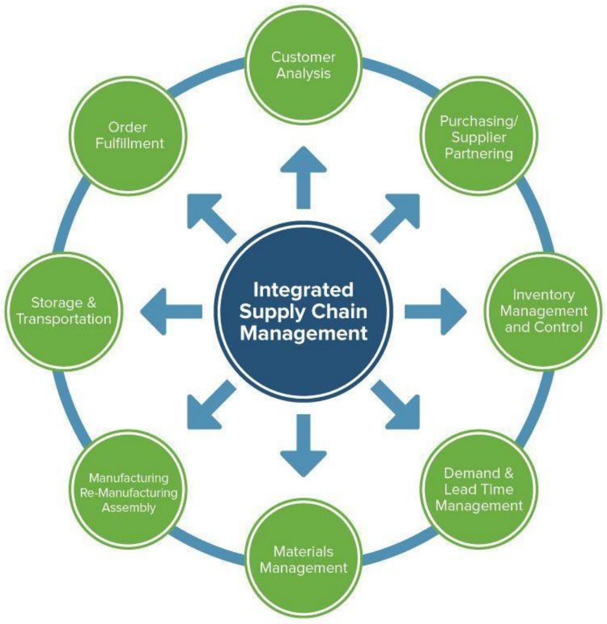

Customer Analysis
Purchasing/ Supplier Partnering
Inventory Management and Control
Demand & Lead Time Management
Materials Management
Manufacturing Re-Manufacturing Assembly
Storage & Transportation
Order Fulfillment
Integrated Supply Chain Management

Fig 4.1.2 Integrated Supply Chain Management

Fig 4.1.2 refers to an enterprise resource planning approach to supply chain management. A business facilitates relationships with all of its suppliers and manages all distribution and logistics activities through a centralized system rather than having multiple systems within the organization. Concentrated professional expertise and cost efficiency are core benefits of the integrated supply chain process, but developing collaboration is an obstacle.

Supply chain management (SCM) is a coordinated system of managing the transportation and logistics activities in a manufacturing, wholesale or retail business. The primary purpose of SCM is to optimize efficiency in supply chain distribution activities. Historically, each supply chain member took a concentrated view of its role in moving goods to the next stage. With an integrated supply chain model, all chain members collaborate with the end goal of delivering the best value to consumers.

A supply chain is a collection of suppliers required to create one specific product for a company. Each supplier is a “link” in the chain that adds time and monetary costs. Supply chain management is the collection of methodologies, theories, and practices that go towards keeping a supply chain running and improving its efficiency for the benefit of most, if not all of the links.

Supply chain integration is a large-scale business strategy that brings as many links of the chain as possible into a closer working relationship with each other. The goal is to improve response time, production time, and reduce costs and waste. Every link in the chain benefits. An integration may be done tightly through a merger with another firm in the supply chain, or loosely through sharing information and working more exclusively with particular suppliers and customers. In the latter case, the supply chain isn't truly "owned" by one company, but the various links operate almost as if one company wanted to increase efficiency and benefit everyone through steady, reliable business.

A separate smart contract is deployed to merely handle the reputation scores of suppliers. The suppliers need to be pre-registered in this smart contract. During registration, the suppliers are given a preliminary reputation score that is adjusted based on their performance. Following a transaction with the supplier, a retailer can provide feedback about the quality of service of this supplier. Retailers are allowed to provide feedback about each supplier to keep the reputation score as fair as possible. The retailers provide their feedback about whether the interaction was satisfactory or not. According to this feedback, the reputation score is modified based on a constant set by the smart contract deployer. This constant determines how strict or lenient the smart contract is towards positive or negative feedback. In the case of positive feedback, the reputation score is multiplied by 0.95/16 and added to the original reputation score. In the case of negative feedback, the reputation score is multiplied by 0.95/24 and subtracted from the original reputation score.

4.2 DRIVING VALUE IN THE SUPPLY CHAIN

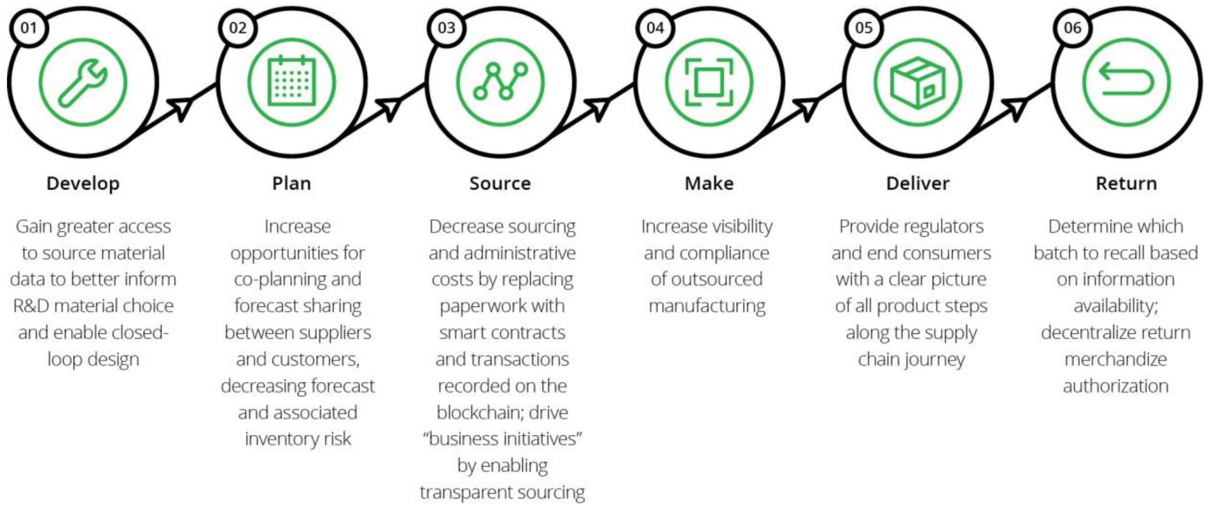

01
02
03
04
05
06
Develop
Gain greater access to source material data to better inform R&D material choice and enable closed-loop design
Plan
Increase opportunities for co-planning and forecast sharing between suppliers and customers, decreasing forecast and associated inventory risk
Source
Decrease sourcing and administrative costs by replacing paperwork with smart contracts and transactions recorded on the blockchain; drive "business initiatives" by enabling transparent sourcing
Make
Increase visibility and compliance of outsourced manufacturing
Deliver
Provide regulators and end consumers with a clear picture of all product steps along the supply chain journey
Return
Determine which batch to recall based on information availability; decentralize return merchandize authorization

Fig 4.2.1 Driving Value in the Supply Chain

Blockchain can provide increased supply chain transparency, as well as reduced cost and risk across the supply chain. Specifically, blockchain supply chain innovations can deliver the following key benefits. As depicted in Fig 4.2.1, **Blockchain can enable more transparent and accurate end-to-end tracking in the supply chain**: Organizations can digitize physical assets and create a decentralized immutable record of all transactions, making it possible to track assets from production to delivery or use by end user.

Primary potential benefits

- Increase traceability of material supply chain to ensure corporate standards are met
- Lower losses from counterfeit/gray market trading
- Improve visibility and compliance over outsourced contract manufacturing
- Reduce paperwork and administrative costs

Secondary potential benefits

- Strengthen corporate reputation through providing transparency of materials used in products

- ➢ Improve credibility and public trust of data shared
- ➢ Reduce potential public relations risk from supply chain malpractice
- ➢ Engage stakeholders

4.3 BLOCK CHAIN USE CASES IN SUPPLY CHAIN BLOCKCHAIN

Enterprise blockchain technology can transform the supply chain with these three use cases:

- Traceability
- Transparency
- Tradeability

Traceability improves operational efficiency by mapping and visualizing enterprise supply chains. A growing number of consumers demand sourcing information about the products they buy. Blockchain helps organizations understand their supply chain and engage consumers with real, verifiable, and immutable data.

Transparency builds trust by capturing key data points, such as certifications and claims, and then provides open access to this data publicly. Once registered on the Ethereum blockchain, its authenticity can be verified by third-party attestors. The information can be updated and validated in real-time.

Tradeability is a unique blockchain offering that redefines the conventional marketplace concept. Using blockchain, one may “tokenize” an asset by splitting an object into shares that digitally represent ownership. Similar to how a stock exchange allows trading of a company’s shares, this fractional ownership allows tokens to represent the value of a shareholder’s stake of a given object. These tokens are tradeable, and users can transfer ownership without the physical asset changing hands.

4.4 SECURITY ANALYSIS

The key properties of the proposed blockchain inventory sharing solution by addressing major security and privacy concerns related to trust, data integrity, availability, non-repudiation, and vulnerability to cyberattacks are explained

- **Confidentiality:** In terms of blockchain core functionality, pseudonyms are used to secure participant identities while also protecting their privacy. As valid transactions are time-stamped and stored in the blockchain, this encourages accountability so all players in the supply chain can be aware of what is going on at any given time. Furthermore, supply chain visibility is critical because it allows you to see how much inventory each registered supplier has. This is useful in situations where demand surges unexpectedly, such as pandemics, so retailers will know which suppliers have the appropriate goods in stock. Additionally, under the currently proposed system, all retailers and vendors are referred to by their EAs, ensuring that their names remain protected.

- **Data Integrity:**

  In a blockchain-based structure, integrity ensures that data stored in the network is accurate and that no one in the network is able to change or delete it. Furthermore, data integrity is protected when cryptographic protocols are used to encrypt valid transactions in a block. This is important since any order made by a retailer in the proposed system is deposited in the ledger as a valid contract, which becomes immutable and cannot be canceled or reversed. Simultaneously, the inventory levels of the suppliers will be reduced in real-time.

- **Availability:**

This relates to the ability of the blockchain system to survive attacks and stay operational even when malicious code, such as a denial of service (DoS) attack, is present. This is attainable due to the hierarchical structure of blockchain technologies. As a result, even though certain nodes in the blockchain network are attacked, all functionalities such as ordering products, updating inventory information, and selling products would still be functional.

- **Non-repudiation:**

  Users that have been registered in the blockchain network cannot deny sending or accepting a particular transaction. This is advantageous because the supply chain network suppliers and retailers cannot deny that orders were placed improperly or that the inventory level is not reflected accurately.

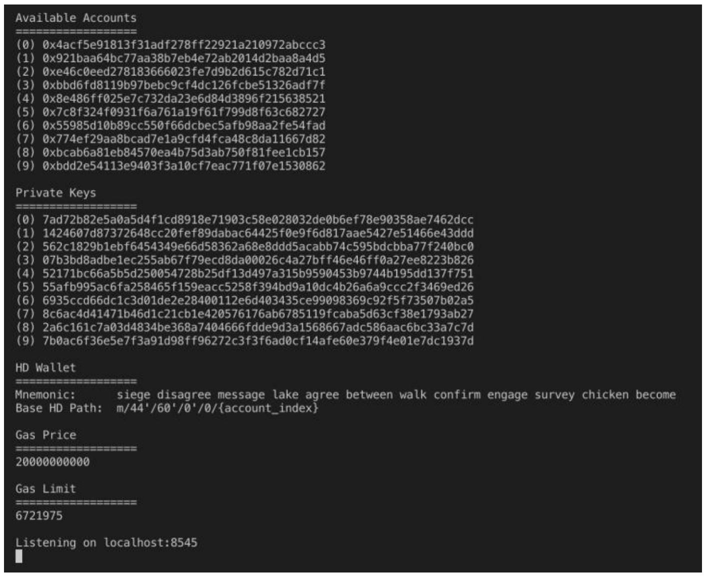
- **Vulnerability to cyberattacks:**

  Cryptographic technology is used to digitally encrypt and secure transactions on the Ethereum network. As a result, any effort to tamper with any transfers will require knowledge of the private key. Furthermore, the attacker is unable to target a specific block since all blocks are cryptographically linked together. Furthermore, cyber-attacks like Man-In-The-Middle (MITM) and distributed denial-of-service (DDoS) are becoming almost impossible.

4.5 IMPLEMENTATION

This is the implementation of our project on the local machine for development and testing purposes. We have already installed ganache-cli, Truffle and the MetaMask extension on our Chrome browser.

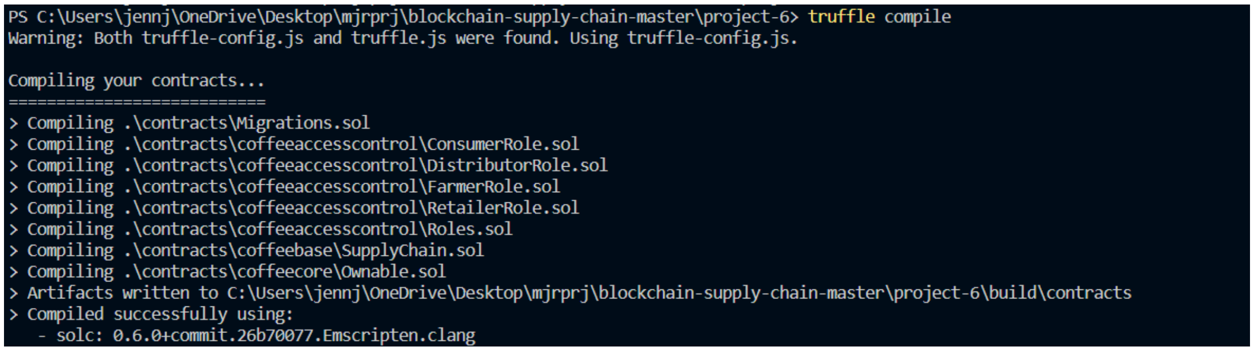
1. When we first launch Ganache, using the mnemonic, the terminal should be as depicted in Fig 4.4.1:



```
Available Accounts
==================
(0) 0x4acf5e91813f31adf278ff22921a210972abccc3
(1) 0x921baa64bc77aa38b7eb4e72ab2014d2baa8a4d5
(2) 0xe46c0eed278183666023fe7d9b2d615c782d71c1
(3) 0xbbd6fd8119b97bebc9cf4dc126fcbe51326adf7f
(4) 0x8e486ff025e7c732da23e6d84d3896f215638521
(5) 0x7c8f324f0931f6a761a19f61f799d8f63c682727
(6) 0x55985d10b89cc550f66dcbec5afb98aa2fe54fad
(7) 0x774ef29aa8bcad7e1a9cfd4fca48c8da11667d82
(8) 0xbcab6a81eb84570ea4b75d3ab750f81fee1cb157
(9) 0xbdd2e54113e9403f3a10cf7eac771f07e1530862

Private Keys
==================
(0) 7ad72b82e5a0a5d4f1cd8918e71903c58e028032de0b6ef78e90358ae7462dcc
(1) 1424607d87372648cc20fef89dabac64425f0e9f6d817aae5427e51466e43ddd
(2) 562c1829b1ebf6454349e66d58362a68e8ddd5acabb74c595bdcbba77f240bc0
(3) 07b3bd8adbe1ec255ab67f79ecd8da00026c4a27bff46e46ff0a27ee8223b826
(4) 52171bc66a5b5d250054728b25df13d497a315b9590453b9744b195dd137f751
(5) 55afb995ac6fa258465f159eacc5258f394bd9a10dc4b26a6a9ccc2f3469ed26
(6) 6935ccd66dc1c3d01de2e28400112e6d403435ce99098369c92f5f73507b02a5
(7) 8c6ac4d41471b46d1c21cb1e420576176ab6785119fcaba5d63cf38e1793ab27
(8) 2a6c161c7a03d4834be368a7404666fdde9d3a1568667adc586aac6bc33a7c7d
(9) 7b0ac6f36e5e7f3a91d98ff96272c3f3f6ad0cf14afe60e379f4e01e7dc1937d

HD Wallet
==================
Mnemonic:      siege disagree message lake agree between walk confirm engage survey chicken become
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Listening on localhost:8545
█
```

Fig 4.5.1 Launching Ganache Cli

2. In a separate window, we compile the smart contracts:

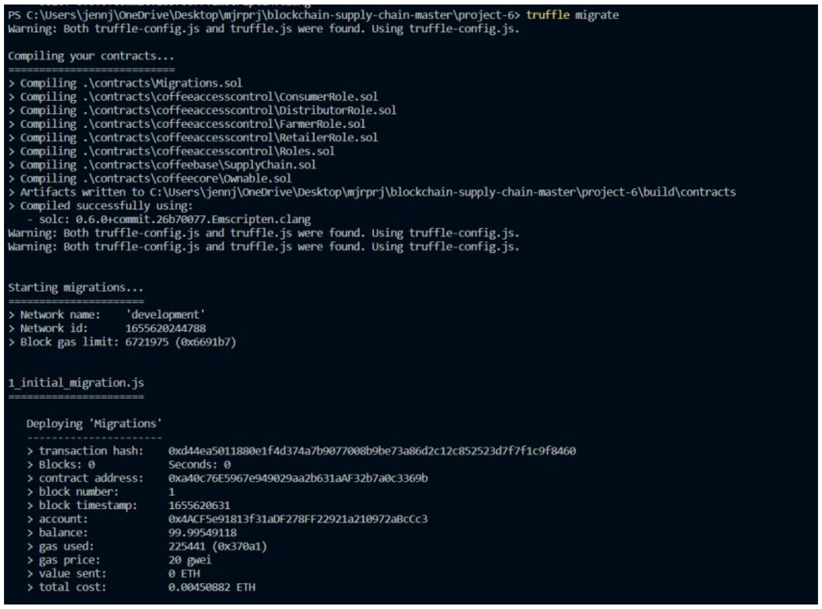


```
PS C:\Users\jennj\OneDrive\Desktop\mjrprj\blockchain-supply-chain-master\project-6> truffle compile
Warning: Both truffle-config.js and truffle.js were found. Using truffle-config.js.

Compiling your contracts...
===========================
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\coffeeaccesscontrol\ConsumerRole.sol
> Compiling .\contracts\coffeeaccesscontrol\DistributorRole.sol
> Compiling .\contracts\coffeeaccesscontrol\FarmerRole.sol
> Compiling .\contracts\coffeeaccesscontrol\RetailerRole.sol
> Compiling .\contracts\coffeeaccesscontrol\Roles.sol
> Compiling .\contracts\coffeebase\SupplyChain.sol
> Compiling .\contracts\coffeecore\Ownable.sol
> Artifacts written to C:\Users\jennj\OneDrive\Desktop\mjrprj\blockchain-supply-chain-master\project-6\build\contracts
> Compiled successfully using:
   - solc: 0.6.0+commit.26b70077.Emscripten.clang
```

Fig 4.5.2 Compiling smart contracts

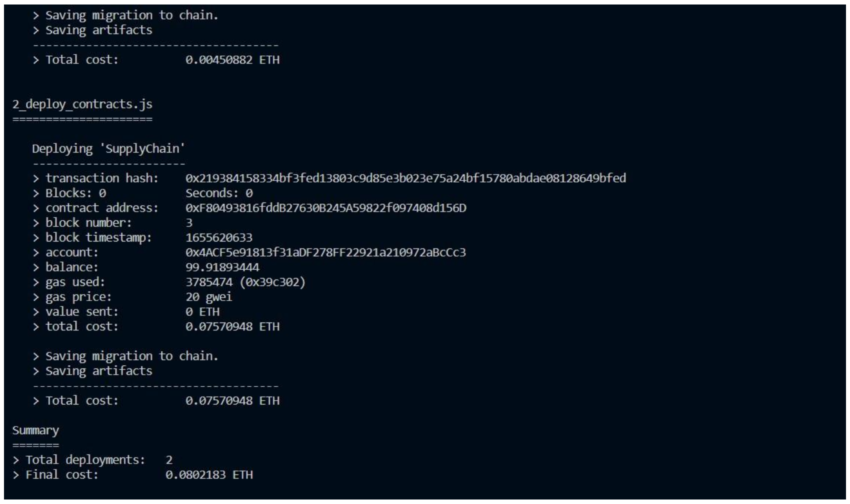
As shown in Fig 4.4.2, this will create the smart contracts artifacts folder build\contracts.

3. Migrate smart contracts to the locally running blockchain, ganache-cli:



```
PS C:\Users\jennj\OneDrive\Desktop\mjrprj\blockchain-supply-chain-master\project-6> truffle migrate
Warning: Both truffle-config.js and truffle.js were found. Using truffle-config.js.

Compiling your contracts...
===========================
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\coffeeaccesscontrol\ConsumerRole.sol
> Compiling .\contracts\coffeeaccesscontrol\DistributorRole.sol
> Compiling .\contracts\coffeeaccesscontrol\FarmerRole.sol
> Compiling .\contracts\coffeeaccesscontrol\RetailerRole.sol
> Compiling .\contracts\coffeeaccesscontrol\Roles.sol
> Compiling .\contracts\coffeebase\SupplyChain.sol
> Compiling .\contracts\coffeecore\Ownable.sol
> Artifacts written to C:\Users\jennj\OneDrive\Desktop\mjrprj\blockchain-supply-chain-master\project-6\build\contracts
> Compiled successfully using:
   - solc: 0.6.0+commit.26b70077.Emscripten.clang
Warning: Both truffle-config.js and truffle.js were found. Using truffle-config.js.
Warning: Both truffle-config.js and truffle.js were found. Using truffle-config.js.


Starting migrations...
======================
> Network name:    'development'
> Network id:      1655620244788
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0xd44ea5011880e1f4d374a7b9077008b9be73a86d2c12c852523d7f7f1c9f8460
   > Blocks: 0            Seconds: 0
   > contract address:    0xa40c76E5967e949029aa2b631aAF32b7a0c3369b
   > block number:        1
   > block timestamp:     1655620631
   > account:             0x4ACF5e91813f31aDF278FF22921a210972aBcCc3
   > balance:             99.99549118
   > gas used:            225441 (0x370a1)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.00450882 ETH
```

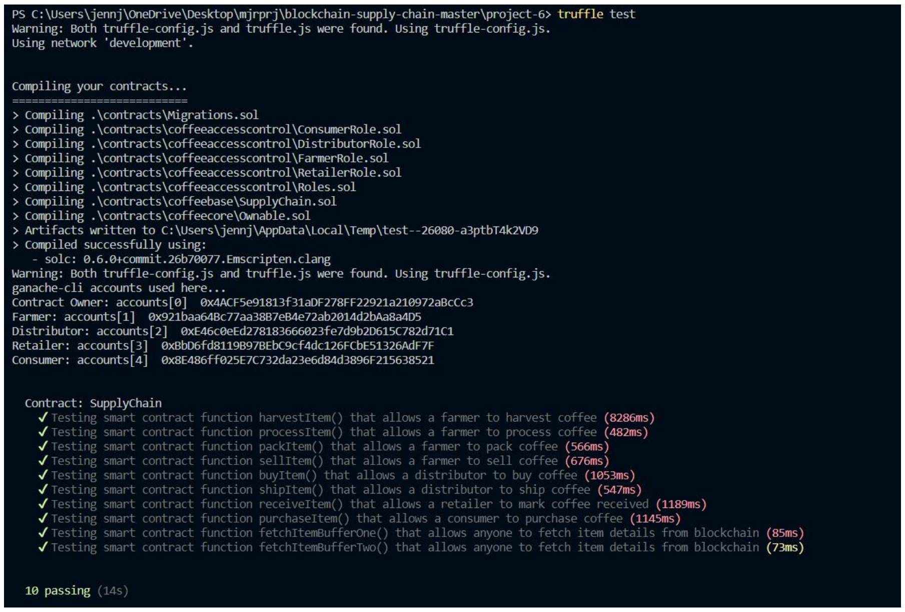
Fig 4.5.3 Truffle Migrate



```
   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.00450882 ETH


2_deploy_contracts.js
=====================

   Deploying 'SupplyChain'
   -----------------------
   > transaction hash:    0x219384158334bf3fed13803c9d85e3b023e75a24bf15780abdae08128649bfed
   > Blocks: 0            Seconds: 0
   > contract address:    0xF80493816fddB27630B245A59822f097408d156D
   > block number:        3
   > block timestamp:     1655620633
   > account:             0x4ACF5e91813f31aDF278FF22921a210972aBcCc3
   > balance:             99.91893444
   > gas used:            3785474 (0x39c302)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.07570948 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:          0.07570948 ETH

Summary
=======
> Total deployments:   2
> Final cost:          0.0802183 ETH
```

As shown in Fig 4.4.3, each of the smart contracts have now been deployed onto the blockchain, on the local network. It is now made available for the application.

4. Now, as shown in Fig 4.4.4, we can test the smart contracts. All 10 tests should pass:

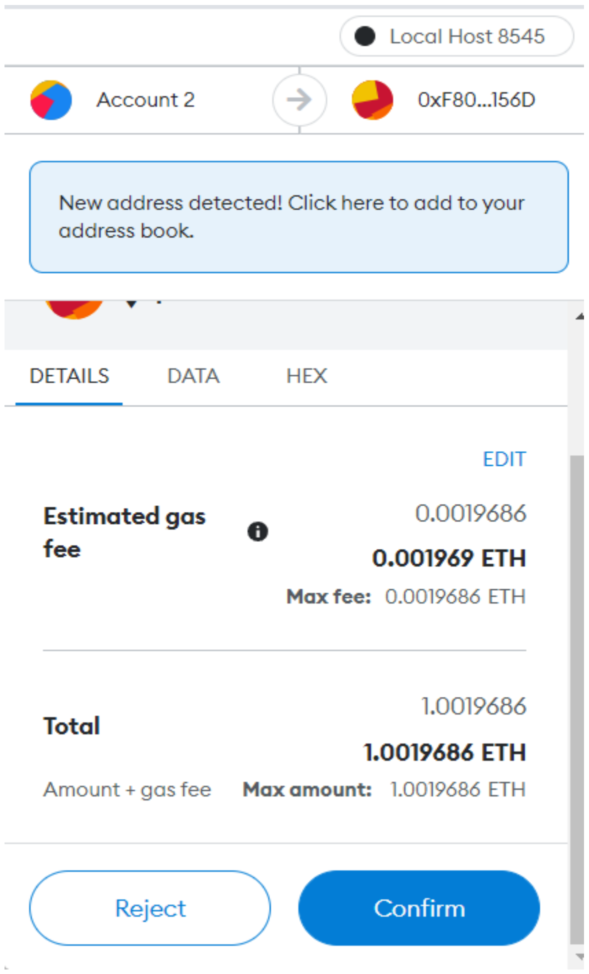


```
PS C:\Users\jennj\OneDrive\Desktop\mjrprj\blockchain-supply-chain-master\project-6> truffle test
Warning: Both truffle-config.js and truffle.js were found. Using truffle-config.js.
Using network 'development'.


Compiling your contracts...
===========================
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\coffeeaccesscontrol\ConsumerRole.sol
> Compiling .\contracts\coffeeaccesscontrol\DistributorRole.sol
> Compiling .\contracts\coffeeaccesscontrol\FarmerRole.sol
> Compiling .\contracts\coffeeaccesscontrol\RetailerRole.sol
> Compiling .\contracts\coffeeaccesscontrol\Roles.sol
> Compiling .\contracts\coffeebase\SupplyChain.sol
> Compiling .\contracts\coffeecore\Ownable.sol
> Artifacts written to C:\Users\jennj\AppData\Local\Temp\test--26080-a3ptbT4k2VD9
> Compiled successfully using:
   - solc: 0.6.0+commit.26b70077.Emscripten.clang
Warning: Both truffle-config.js and truffle.js were found. Using truffle-config.js.
ganache-cli accounts used here...
Contract Owner: accounts[0]  0x4ACF5e91813f31aDF278FF22921a210972aBcCc3
Farmer: accounts[1]  0x921baa64Bc77aa38B7eB4e72ab2014d2bAa8a4D5
Distributor: accounts[2]  0xE46c0eEd278183666023fe7d9b2D615C782d71C1
Retailer: accounts[3]  0xBbD6fd8119B97BEbC9cf4dc126FCbE51326AdF7F
Consumer: accounts[4]  0x8E486ff025E7C732da23e6d84d3896F215638521


  Contract: SupplyChain
    ✓ Testing smart contract function harvestItem() that allows a farmer to harvest coffee (8286ms)
    ✓ Testing smart contract function processItem() that allows a farmer to process coffee (482ms)
    ✓ Testing smart contract function packItem() that allows a farmer to pack coffee (566ms)
    ✓ Testing smart contract function sellItem() that allows a farmer to sell coffee (676ms)
    ✓ Testing smart contract function buyItem() that allows a distributor to buy coffee (1053ms)
    ✓ Testing smart contract function shipItem() that allows a distributor to ship coffee (547ms)
    ✓ Testing smart contract function receiveItem() that allows a retailer to mark coffee received (1189ms)
    ✓ Testing smart contract function purchaseItem() that allows a consumer to purchase coffee (1145ms)
    ✓ Testing smart contract function fetchItemBufferOne() that allows anyone to fetch item details from blockchain (85ms)
    ✓ Testing smart contract function fetchItemBufferTwo() that allows anyone to fetch item details from blockchain (73ms)


  10 passing (14s)
```

Fig 4.5.4 Truffle test

Now, we can test our DApp. The application, when launched will automatically open MetaMask. This is a good indication that both our app as well as the MetaMask account are properly connected to the local server.

Whenever we make any transactions, we will receive a notification from MetaMask confirming our transaction, along with the gas fee used to create the transaction.


Local Host 8545

Account 2 → 0xF80...156D

New address detected! Click here to add to your address book.

Estimated gas fee 0.0019686
0.001969 ETH
Max fee: 0.0019686 ETH

Total 1.0019686
1.0019686 ETH
Amount + gas fee Max amount: 1.0019686 ETH

Fig 4.5.5 MetaMask confirmation

This way, we can know that all our transactions are authenticated and every single process in the supply chain will be accounted for, for every entity on this permissioned blockchain to witness.

5. UML DIAGRAMS

Activity Diagram:

Activity diagram is **a behavioral diagram** i.e. it depicts the behavior of a system. An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.

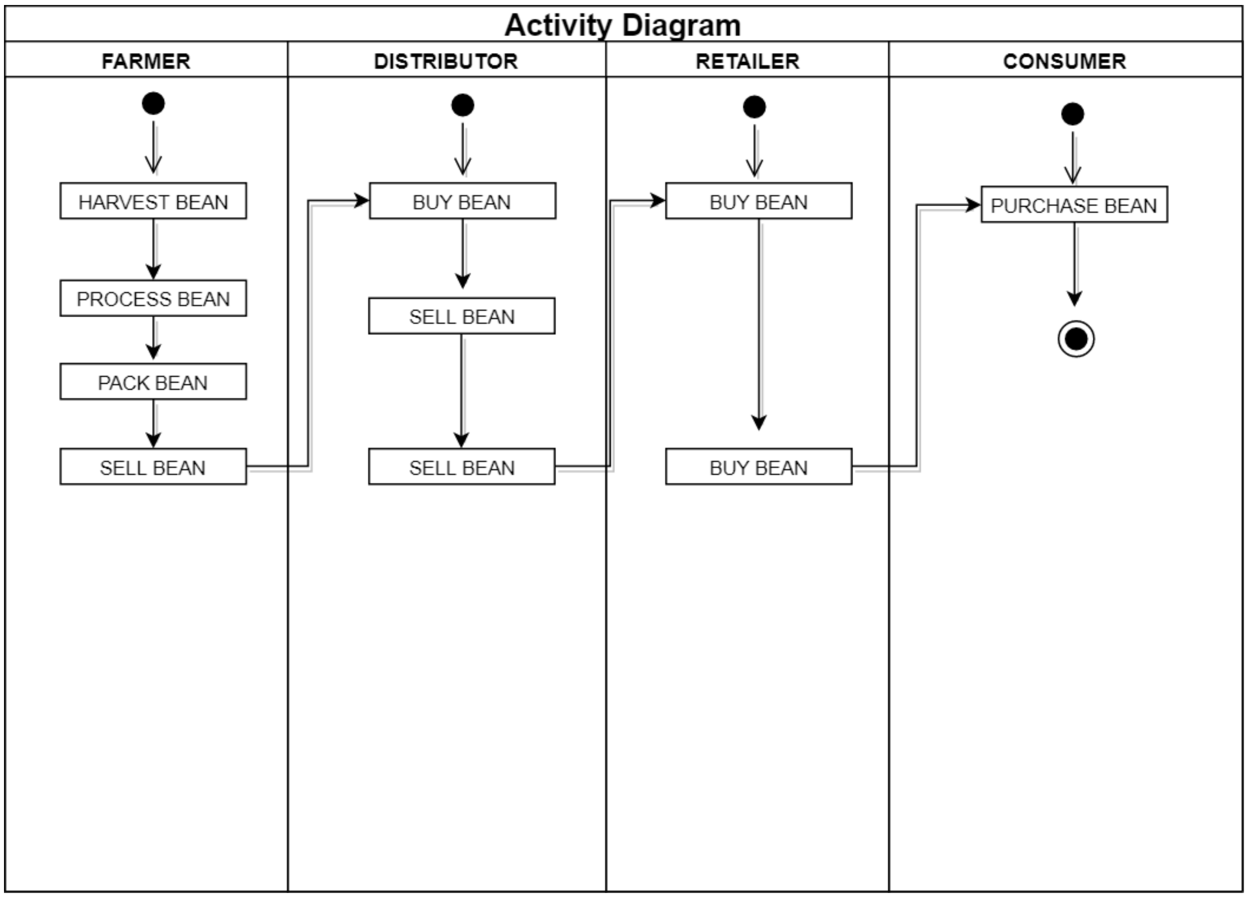


Figure 5.1: Activity Diagram.

Figure 5.1: Describes the flow of the supply chain that involves growing the beans, harvesting, hulling, drying, packing, bulking, blending and finally roasting. In between this process, the beans go through international transporters, export sellers and retailers like grocery stores, cafes and specialty shops.

Sequence Diagram:

The system sequence diagram shows us object interactions arranged in time sequence. The objects involved here are the user, dataset and system and we are able to see the messages exchanged between the respective objects.

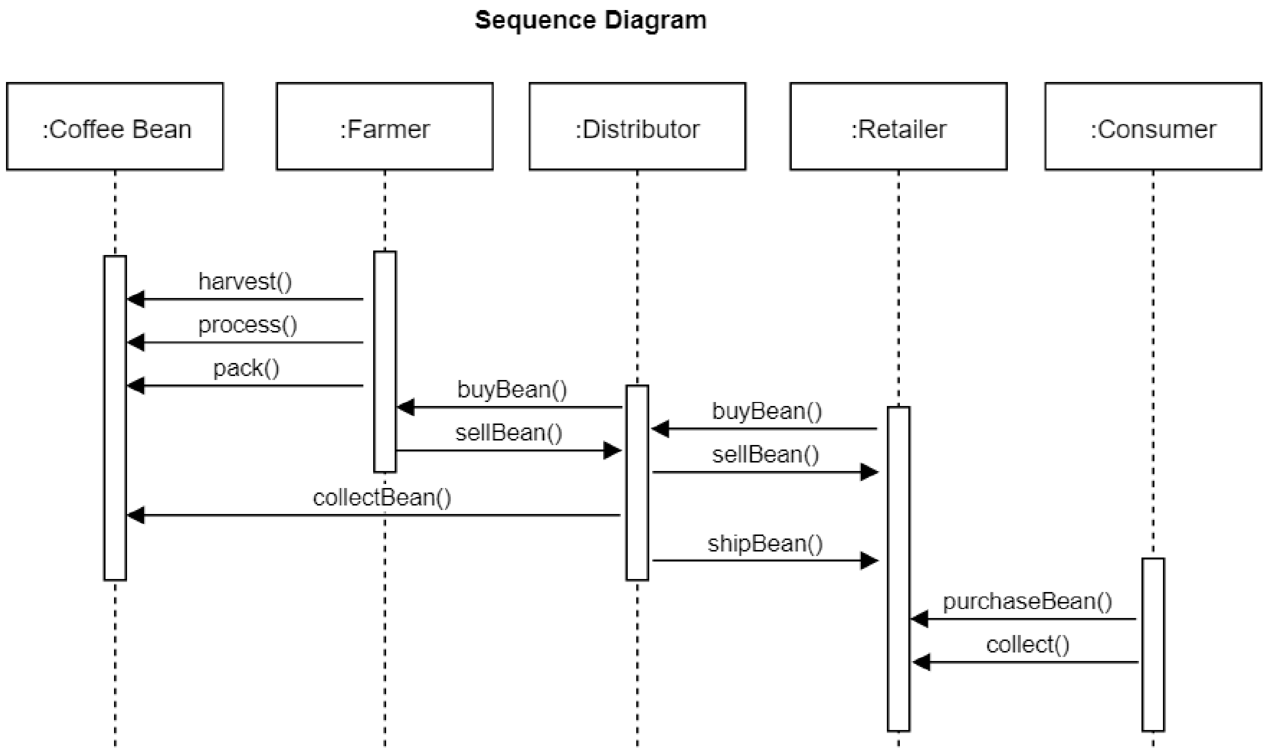


Figure 5.2: Sequence Diagram

Figure 5.2:This is the uml sequence diagram of the supply chain management system which shows the interaction between objects of Supplier,Distributor,Consumer and Retailer.

State Diagram:

A state diagram is **a type of diagram used in computer science and related fields to describe the behavior of systems**. State diagrams require that the system described is composed of a finite number of states; sometimes, this is indeed the case, while at other times this is a reasonable abstraction.

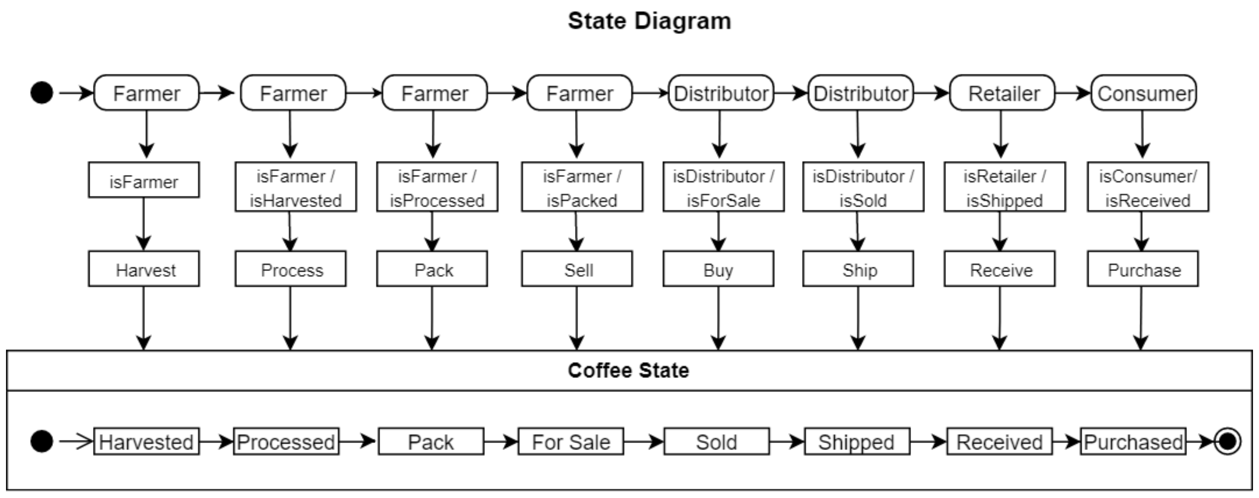


Figure 5.3: State Diagram

Figure 5.3 shows the different states of each object and the events or conditions that change those states.

Data Modeling Diagram:

The Data Modeling diagram is **used to create or view graphical models of relational database system schemas** including a range of database objects. The diagrams can be drawn at a logical or a physical level.

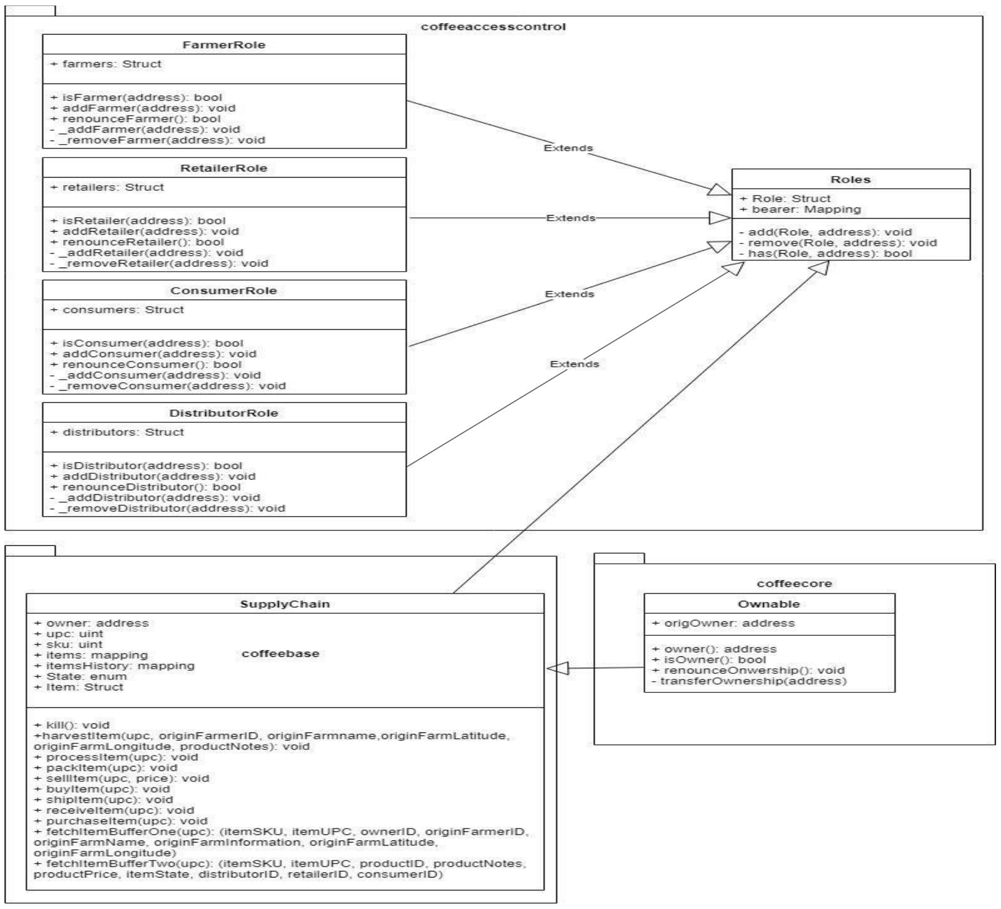

coffeeaccesscontrol
FarmerRole
+ farmers: Struct
+ isFarmer(address): bool
+ addFarmer(address): void
+ renounceFarmer(): bool
- _addFarmer(address): void
- _removeFarmer(address): void
RetailerRole
+ retailers: Struct
+ isRetailer(address): bool
+ addRetailer(address): void
+ renounceRetailer(): bool
- _addRetailer(address): void
- _removeRetailer(address): void
ConsumerRole
+ consumers: Struct
+ isConsumer(address): bool
+ addConsumer(address): void
+ renounceConsumer(): bool
- _addConsumer(address): void
- _removeConsumer(address): void
DistributorRole
+ distributors: Struct
+ isDistributor(address): bool
+ addDistributor(address): void
+ renounceDistributor(): bool
- _addDistributor(address): void
- _removeDistributor(address): void
Extends
Extends
Extends
Extends
Roles
+ Role: Struct
+ bearer: Mapping
- add(Role, address): void
- remove(Role, address): void
- has(Role, address): bool
SupplyChain
coffeebase
+ owner: address
+ upc: uint
+ sku: uint
+ items: mapping
+ itemsHistory: mapping
+ State: enum
+ Item: Struct
+ kill(): void
+harvestItem(upc, originFarmerID, originFarmname,originFarmLatitude, originFarmLongitude, productNotes): void
+ processItem(upc): void
+ packItem(upc): void
+ sellItem(upc, price): void
+ buyItem(upc): void
+ shipItem(upc): void
+ receiveItem(upc): void
+ purchaseItem(upc): void
+ fetchItemBufferOne(upc): (itemSKU, itemUPC, ownerID, originFarmerID, originFarmName, originFarmInformation, originFarmLatitude, originFarmLongitude)
+ fetchItemBufferTwo(upc): (itemSKU, itemUPC, productID, productNotes, productPrice, itemState, distributorID, retailerID, consumerID)
coffeecore
Ownable
+ origOwner: address
+ owner(): address
+ isOwner(): bool
+ renounceOnwership(): void
- transferOwnership(address)

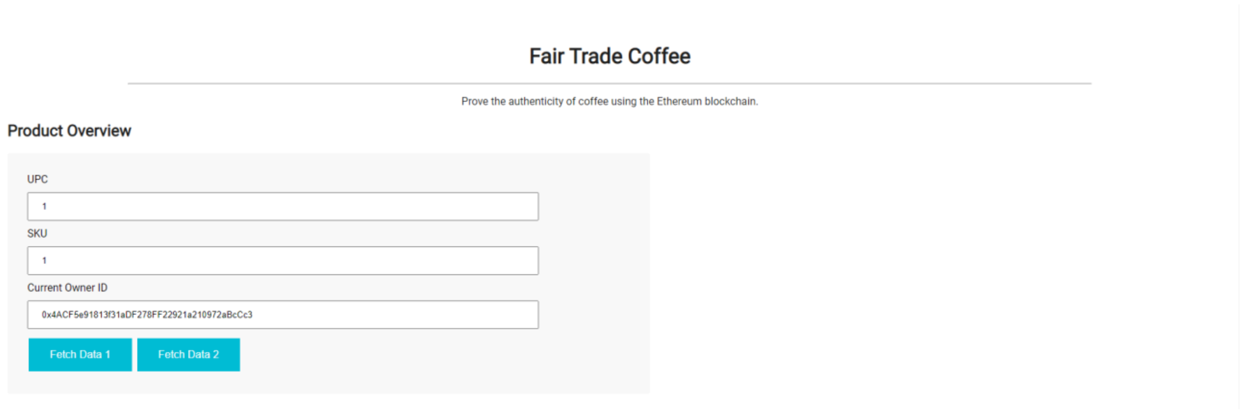
Figure 5.4: Data Modeling Diagram

Figure 5.4 shows the different Smart Contracts and the relation between them.

6. RESULTS

The user story followed through the implementation of this project is similar to any commonly used supply chain process. A Seller can add items to the inventory system stored in the blockchain. A Buyer can purchase such items from the inventory system. Additionally, a Seller can mark an item as Shipped, and similarly a Buyer can mark an item as Received.

The DApp user interface when running should look like this:


Fair Trade Coffee

Prove the authenticity of coffee using the Ethereum blockchain.

Product Overview

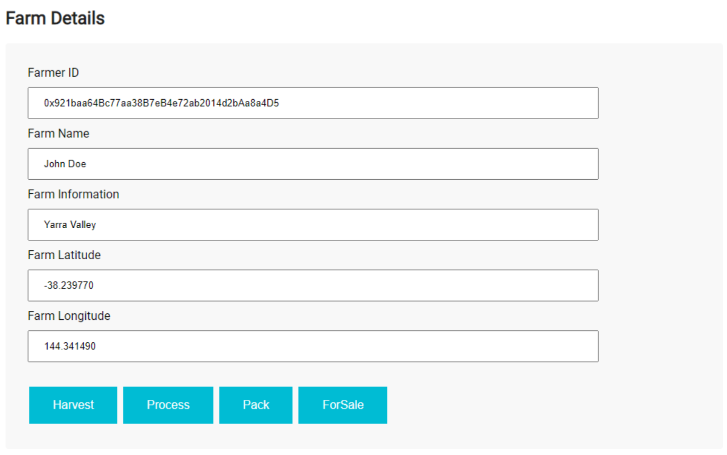
UPC: 1

SKU: 1

Current Owner ID: 0x4ACF5e91813f31aDF278FF22921a210972aBcCc3

Fig 6.1 DApp user interface

In figure 6.1, through the user interface, we can fetch details pertaining to the current owner ID, when we provide product UPC and SKU.


Farm Details

Farmer ID: 0x921baa64Bc77aa38B7eB4e72ab2014d2bAa8a4D5

Farm Name: John Doe

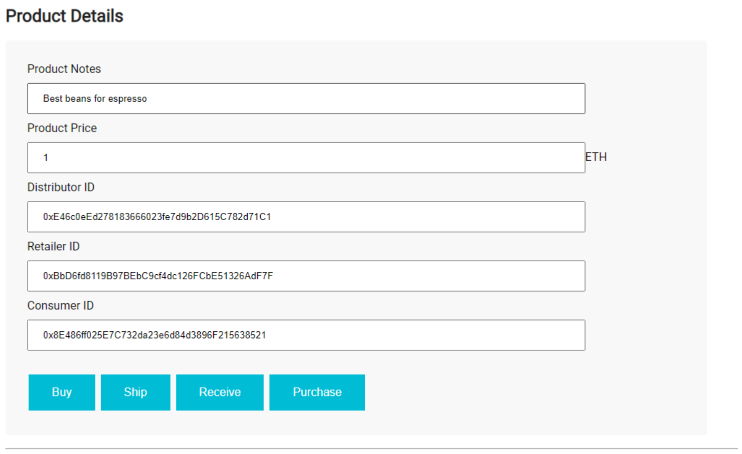
Farm Information: Yarra Valley

Farm Latitude: -38.239770

Farm Longitude: 144.341490

Fig 6.2 Farm Details

In Fig 6.2, the farmer is allowed to register the state of the product, thus creating a transaction that will eventually be stored in the transaction history. This will allow other entities on the permissioned blockchain to track the product's status throughout the supply chain.

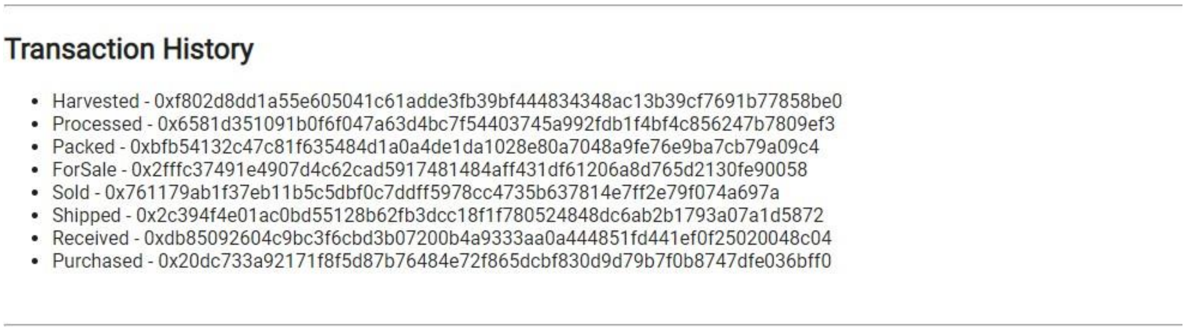


Product Details

Product Notes	Best beans for espresso
Product Price	1 ETH
Distributor ID	0xE46c0eEd278183666023fe7d9b2D615C782d71C1
Retailer ID	0xBbD6fd8119B97BEbC9cf4dc126FCbE51326AdF7F
Consumer ID	0x8E486ff025E7C732da23e6d84d3896F215638521

Fig 6.3 Product Details

In Fig 6.3, the distributor, retailer and consumer Ids are to be given and the product's status throughout the supply chain is also register for all entities on the permissioned blockchain to make note of.



Transaction History

- Harvested - 0xf802d8dd1a55e605041c61adde3fb39bf444834348ac13b39cf7691b77858be0
- Processed - 0x6581d351091b0f6f047a63d4bc7f54403745a992fdb1f4bf4c856247b7809ef3
- Packed - 0xbfb54132c47c81f635484d1a0a4de1da1028e80a7048a9fe76e9ba7cb79a09c4
- ForSale - 0x2fffc37491e4907d4c62cad5917481484aff431df61206a8d765d2130fe90058
- Sold - 0x761179ab1f37eb11b5c5dbf0c7ddff5978cc4735b637814e7ff2e79f074a697a
- Shipped - 0x2c394f4e01ac0bd55128b62fb3dcc18f1f780524848dc6ab2b1793a07a1d5872
- Received - 0xdb85092604c9bc3f6cbd3b07200b4a9333aa0a444851fd441ef0f25020048c04
- Purchased - 0x20dc733a92171f8f5d87b76484e72f865dcbf830d9d79b7f0b8747dfe036bff0

Fig 6.4 Transaction History

As shown in Fig 6.4, the entire transaction history is maintained for all stages of the supply chain, thus maintaining transparency and a measure of authenticity for this process.

The metamask account used is imported from the available accounts when ganache-cli is started. This account contains some test ethers which were used to test each transaction taking place. All transactions are recorded such that information is made available to all members of this permissioned blockchain.

7. CONCLUSION AND FUTURE SCOPE

The importance of inventory sharing among various stakeholders in a supply chain has been indicated. The relevance of information sharing to running an effective and profitable supply chain operation is discussed. The proposed solution combines blockchain and decentralized storage technologies to improve trust, efficiency and transparency in complex supply chains. Additionally, it helps strengthen the coordination between partners while reducing unnecessary delays. The proposed solution can be adapted to various product categories in the supply chain where the system architecture, algorithms, sequence diagrams, and testing scenarios can be suitably customized. The solution permits only registered stakeholders to communicate with the smart contract, maintaining the integrity of transactions, stakeholder trust, and accountability. The security of the proposed solution in terms of data integrity, non-repudiation, has been discussed. The various transaction costs involved for each stakeholder are analyzed. This work demonstrates that blockchain-based information sharing solutions reduce inefficiencies, are economical, commercially viable, and provide improved information connectivity among supply chain stakeholders in a trusted and secure way.

Developing decentralized applications to facilitate full automation of other relevant processes affecting all supply chain stakeholders as potential future work is proposed. We also acknowledge that as blockchain is still in its infancy, problems such as scalability, governance, and energy consumption still remain as open challenges to be addressed in the future.

REFERENCES

[1] C. YU, Y. ZHAN and Z. LI, "Using Blockchain and Smart Contract for Traceability in Agricultural Products Supply Chain," 2020 International Conference on Internet of Things and Intelligent Applications (ITIA), 2020, pp. 1-5, doi: 10.1109/ITIA50152.2020.9312315.

[2] S. E. Chang and Y. Chen, "When Blockchain Meets Supply Chain: A Systematic Literature Review on Current Development and Potential Applications," in IEEE Access, vol. 8, pp. 62478-62494, 2020, doi: 10.1109/ACCESS.2020.2983601.

[3] S. Saberi, M. Kouhizadeh and J. Sarkis, "Blockchains and the Supply Chain: Findings from a Broad Study of Practitioners," in IEEE Engineering Management Review, vol. 47, no. 3, pp. 95-103, 1 third quarter,Sept. 2019, doi: 10.1109/EMR.2019.2928264.

[4] S. R. Niya, D. Dordevic, A. G. Nabi, T. Mann and B. Stiller, "A Platform-independent, Generic-purpose, and Blockchain-based Supply Chain Tracking," 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2019, pp. 11-12, doi: 10.1109/BLOC.2019.8751415.

[5] S. Yousuf and D. Svetinovic, "Blockchain Technology in Supply Chain Management: Preliminary Study," 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), 2019, pp. 537-538, doi: 10.1109/IOTSMS48152.2019.8939222.

[6] S. Su, K. Wang and H. S. Kim, "Smart Supply: Smart Contract Based Validation for Supply Chain Blockchain," 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 988-993, doi: 10.1109/Cybermatics_2018.2018.00186.

[7] S. Sahai, N. Singh and P. Dayama, "Enabling Privacy and Traceability in Supply Chains using Blockchain and Zero Knowledge Proofs," 2020 IEEE International Conference on Blockchain (Blockchain), 2020, pp. 134-143, doi: 10.1109/Blockchain50366.2020.00024.

[8] Y. Raj and S. B, "Study on Supply Chain Management using Blockchain Technology," 2021 6th International Conference on Inventive Computation Technologies (ICICT), 2021, pp. 1243-1247, doi: 10.1109/ICICT50816.2021.9358768.

[9] Y. Raj and S. B, "Study on Supply Chain Management using Blockchain Technology," 2021 6th International Conference on Inventive Computation Technologies (ICICT), 2021, pp. 1243-1247, doi: 10.1109/ICICT50816.2021.9358768.

[10] Martin Westerkamp, Friedhelm Victor, Axel Küpper, Tracing manufacturing processes using blockchain-based token compositions, Digital Communications and Networks, Volume 6, Issue 2, 2020

[11] Omar, Ilhaam & Jayaraman, Raja & Debe, Mazin & Hasan, Haya & Salah, Khaled & Omar, Mohammed. (2021). Supply Chain Inventory Sharing Using Ethereum Blockchain and Smart Contracts. IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3139829.

[12] https://hbr.org/2020/05/building-a-transparent-supply-chain

[13] https://www.kaleido.io/blockchain-platform/truffle

APPENDIX

CONTRACTS:

- Coffeeaccesscontrol:

```
 >ConsumerRole.sol:
pragma solidity 0.6.0;

// Import the library 'Roles'
import "./Roles.sol";

// Define a contract 'ConsumerRole' to manage this role - add, remove, check
contract ConsumerRole {

  using Roles for Roles.Role;

  // Define 2 events, one for Adding, and other for Removing
  event ConsumerAdded(address indexed account);
  event ConsumerRemoved(address indexed account);
  // Define a struct 'consumers' by inheriting from 'Roles' library, struct Role
  Roles.Role private consumers;

  // In the constructor make the address that deploys this contract the 1st consumer
  constructor() public {
    _addConsumer(msg.sender);
  }

  // Define a modifier that checks to see if msg.sender has the appropriate role
  modifier onlyConsumer() {
    require(consumers.has(msg.sender), "This account has no Consumer Role");
    _;
  }

  // Define a function 'isConsumer' to check this role
  function isConsumer(address account) public view returns (bool) {
    return consumers.has(account);
  }

  // Define a function 'addConsumer' that adds this role
  function addConsumer(address account) public onlyConsumer {
    _addConsumer(account);
  }

  // Define a function 'renounceConsumer' to renounce this role
  function renounceConsumer() public {
    _removeConsumer(msg.sender);
  }

  // Define an internal function '_addConsumer' to add this role, called by 'addConsumer'
  function _addConsumer(address account) internal {
    consumers.add(account);
    emit ConsumerAdded(account);
  }

  // Define an internal function '_removeConsumer' to remove this role, called by
'removeConsumer'
  function _removeConsumer(address account) internal {
    consumers.remove(account);
    emit ConsumerRemoved(account);
  }
}

>DistributorRole.sol:
pragma solidity ^0.6.0;

// Import the library 'Roles'
import "./Roles.sol";

// Define a contract 'DistributorRole' to manage this role - add, remove, check
contract DistributorRole {

  using Roles for Roles.Role;

  // Define 2 events, one for Adding, and other for Removing
  event DistributorAdded(address indexed account);
  event DistributorRemoved(address indexed account);

  // Define a struct 'distributors' by inheriting from 'Roles' library, struct Role
  Roles.Role private distributors;
  // In the constructor make the address that deploys this contract the 1st distributor
  constructor() public {
    // The first distributor will be the person deploying this contract
    _addDistributor(msg.sender);
  }

  // Define a modifier that checks to see if msg.sender has the appropriate role
  modifier onlyDistributor() {
      require(distributors.has(msg.sender), "This account has no Distributor Role");
    _;
  }

  // Define a function 'isDistributor' to check this role
  function isDistributor(address account) public view returns (bool) {
       return distributors.has(account);
  }

  // Define a function 'addDistributor' that adds this role
  function addDistributor(address account) public onlyDistributor {
      _addDistributor(account);
  }

  // Define a function 'renounceDistributor' to renounce this role
  function renounceDistributor() public {
    _removeDistributor(msg.sender);
  }

  // Define an internal function '_addDistributor' to add this role, called by 'addDistributor'
  function _addDistributor(address account) internal {
    distributors.add(account);
    emit DistributorAdded(account);
  }

  // Define an internal function '_removeDistributor' to remove this role, called by
'removeDistributor'
  function _removeDistributor(address account) internal {
    distributors.remove(account);
    emit DistributorRemoved(account);
  }
}
```

```
>FarmerRole.sol:
pragma solidity ^0.6.0;

// Import the library 'Roles'
import "./Roles.sol";

// Define a contract 'FarmerRole' to manage this role - add, remove, check
contract FarmerRole {
  using Roles for Roles.Role;

  // Define 2 events, one for Adding, and other for Removing
  event FarmerAdded(address indexed account);
  event FarmerRemoved(address indexed account);

  // Define a struct 'farmers' by inheriting from 'Roles' library, struct Role
  Roles.Role private farmers;

  // In the constructor make the address that deploys this contract the 1st farmer
  constructor() public {
    _addFarmer(msg.sender);
  }

  // Define a modifier that checks to see if msg.sender has the appropriate role
  modifier onlyFarmer() {
    require(isFarmer(msg.sender), "This account has no a Farmer Role");
    _;
  }

  // Define a function 'isFarmer' to check this role
  function isFarmer(address account) public view returns (bool) {
    return farmers.has(account);
  }

  // Define a function 'addFarmer' that adds this role
  function addFarmer(address account) public onlyFarmer {
    _addFarmer(account);
  }

  // Define a function 'renounceFarmer' to renounce this role
  function renounceFarmer() public {
    _removeFarmer(msg.sender);
  }

  // Define an internal function '_addFarmer' to add this role, called by 'addFarmer'
  function _addFarmer(address account) internal {
    farmers.add(account);
    emit FarmerAdded(account);
  }

  // Define an internal function '_removeFarmer' to remove this role, called by
'removeFarmer'
  function _removeFarmer(address account) internal {
    farmers.remove(account);
    emit FarmerRemoved(account);
  }
}

>RetailerRole.sol:
pragma solidity ^0.6.0;

// Import the library 'Roles'
import "./Roles.sol";

// Define a contract 'RetailerRole' to manage this role - add, remove, check
contract RetailerRole {

  using Roles for Roles.Role;

  // Define 2 events, one for Adding, and other for Removing
  event RetailerAdded(address indexed account);
  event RetailerRemoved(address indexed account);

  // Define a struct 'retailers' by inheriting from 'Roles' library, struct Role
  Roles.Role private retailers;

  // In the constructor make the address that deploys this contract the 1st retailer
  constructor() public {
    _addRetailer(msg.sender);
  }
  // Define a modifier that checks to see if msg.sender has the appropriate role
  modifier onlyRetailer() {
    require(retailers.has(msg.sender), "This account has no Retailer role");
    _;
  }

  // Define a function 'isRetailer' to check this role
  function isRetailer(address account) public view returns (bool) {
    return retailers.has(account);
  }

  // Define a function 'addRetailer' that adds this role
  function addRetailer(address account) public onlyRetailer {
    _addRetailer(account);
  }

  // Define a function 'renounceRetailer' to renounce this role
  function renounceRetailer() public {
    _removeRetailer(msg.sender);
  }

  // Define an internal function '_addRetailer' to add this role, called by 'addRetailer'
  function _addRetailer(address account) internal {
    retailers.add(account);
    emit RetailerAdded(account);
  }

  // Define an internal function '_removeRetailer' to remove this role, called by
'removeRetailer'
  function _removeRetailer(address account) internal {
    retailers.remove(account);
    emit RetailerRemoved(account);
  }
}

>Roles.sol:
pragma solidity ^0.6.0;

/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
  struct Role {
    mapping (address => bool) bearer;
  }

  /**
   * @dev give an account access to this role
   */
  function add(Role storage role, address account) internal {
    require(account != address(0), "Address cannot be zero address");
    require(!has(role, account), "This address already has this role");
    role.bearer[account] = true;
  }

  /**
   * @dev remove an account's access to this role
   */
  function remove(Role storage role, address account) internal {
    require(account != address(0), "Address cannot be zero address");
    require(has(role, account), "This address does not have this role yet");

    role.bearer[account] = false;
  }

  /**
   * @dev check if an account has this role
   * @return bool
   */
  function has(Role storage role, address account)
    internal
    view
    returns (bool)
  {
    require(account != address(0), "Address cannot be zero address");
    return role.bearer[account];
  }
}
```

- coffeecore
  >Ownable.sol:

```
pragma solidity ^0.6.0;

/// Provides basic authorization control
contract Ownable {
    address payable private origOwner;

    // Define an Event
    event TransferOwnership(address indexed oldOwner, address indexed newOwner);

    /// Assign the contract to an owner
    constructor() internal {
        origOwner = msg.sender;
        emit TransferOwnership(address(0), origOwner);
    }

    // Define a function 'kill' if required
    function kill() public {
        if (msg.sender == origOwner) {
        selfdestruct(origOwner);
        }
    }

    /// Look up the address of the owner
    function owner() public view returns (address) {
        return origOwner;
    }

    /// Define a function modifier 'onlyOwner'
    modifier onlyOwner() {
        require(isOwner(), "Only the owner can perform this operation");
        _;
    }

    /// Check if the calling address is the owner of the contract
    function isOwner() public view returns (bool) {
        return msg.sender == origOwner;
    }
    /// Define a function to renounce ownerhip
    function renounceOwnership() public onlyOwner {
        emit TransferOwnership(origOwner, address(0));
        origOwner = address(0);
    }

    /// Define a public function to transfer ownership
    function transferOwnership(address payable newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /// Define an internal function to transfer ownership
    function _transferOwnership(address payable newOwner) internal {
        require(newOwner != address(0), "The new Owner cannot be address 0");
        emit TransferOwnership(origOwner, newOwner);
        origOwner = newOwner;
    }
}
```

- coffeebase
  >SupplyChain.sol:

```
pragma solidity ^0.6.0;


import '../coffeeaccesscontrol/FarmerRole.sol';
import '../coffeeaccesscontrol/DistributorRole.sol';
import '../coffeeaccesscontrol/RetailerRole.sol';
import '../coffeeaccesscontrol/ConsumerRole.sol';
import '../coffeecore/Ownable.sol';


// Define a contract 'Supplychain'
contract SupplyChain is Ownable, FarmerRole, DistributorRole, RetailerRole,
ConsumerRole {

  // Define a variable called 'upc' for Universal Product Code (UPC)
  uint  upc;

  // Define a variable called 'sku' for Stock Keeping Unit (SKU)
  uint  sku;

  // Define a public mapping 'items' that maps the UPC to an Item.
  mapping (uint => Item) items;

  // Define a public mapping 'itemsHistory' that maps the UPC to an array of TxHash,
  // that track its journey through the supply chain -- to be sent from DApp.
  mapping (uint => string[]) itemsHistory;

  // Define enum 'State' with the following values:
  enum State
  {
    Harvested,  // 0
    Processed,  // 1
    Packed,     // 2
    ForSale,    // 3
    Sold,       // 4
    Shipped,    // 5
    Received,   // 6
    Purchased   // 7
    }

  State constant defaultState = State.Harvested;

  // Define a struct 'Item' with the following fields:
  struct Item {
    uint    sku;  // Stock Keeping Unit (SKU)
    uint    upc; // Universal Product Code (UPC), generated by the Farmer, goes on the
package, can be verified by the Consumer
    address payable ownerID;  // Metamask-Ethereum address of the current owner as the
product moves through 8 stages
    address payable originFarmerID; // Metamask-Ethereum address of the Farmer
    string  originFarmName; // Farmer Name
    string  originFarmInformation;  // Farmer Information
    string  originFarmLatitude; // Farm Latitude
    string  originFarmLongitude;  // Farm Longitude
    uint    productID;  // Product ID potentially a combination of upc + sku
    string  productNotes; // Product Notes
    uint    productPrice; // Product Price
    State   itemState;  // Product State as represented in the enum above
  address payable distributorID;  // Metamask-Ethereum address of the Distributor
  address payable retailerID; // Metamask-Ethereum address of the Retailer
  address payable consumerID; // Metamask-Ethereum address of the Consumer
}

// Define 8 events with the same 8 state values and accept 'upc' as input argument
event Harvested(uint upc);
event Processed(uint upc);
event Packed(uint upc);
event ForSale(uint upc);
event Sold(uint upc);
event Shipped(uint upc);
event Received(uint upc);
event Purchased(uint upc);


// Define a modifer that verifies the Caller
modifier verifyCaller (address _address) {
  require(msg.sender == _address, "This account is not the owner of this item");
  _;
}

// Define a modifier that checks if the paid amount is sufficient to cover the price
modifier paidEnough(uint _price) {
  require(msg.value >= _price, "The amount sent is not sufficient for the price");
  _;
}

// Define a modifier that checks the price and refunds the remaining balance
modifier checkValueForDistributor(uint _upc) {
  _;
  uint _price = items[_upc].productPrice;
  uint amountToReturn = msg.value - _price;
  items[_upc].distributorID.transfer(amountToReturn);
}

  // Define a modifier that checks the price and refunds the remaining balance
  // to the Consumer
modifier checkValueForConsumer(uint _upc) {
  _;
```

```
    uint _price = items[_upc].productPrice;
    uint amountToReturn = msg.value - _price;
    items[_upc].consumerID.transfer(amountToReturn);
  }

  // Define a modifier that checks if an item.state of a upc is Harvested
  modifier harvested(uint _upc) {
    require(items[_upc].itemState == State.Harvested, "The Item is not in Harvested
state!");
    _;
  }

  // Define a modifier that checks if an item.state of a upc is Processed
  modifier processed(uint _upc) {
    require(items[_upc].itemState == State.Processed, "The Item is not in Processed
state!");
    _;
  }

  // Define a modifier that checks if an item.state of a upc is Packed
  modifier packed(uint _upc) {
    require(items[_upc].itemState == State.Packed, "The Item is not in Packed state!");
    _;
  }

  // Define a modifier that checks if an item.state of a upc is ForSale
  modifier forSale(uint _upc) {
    require(items[_upc].itemState == State.ForSale, "The Item is not in ForSale state!");
    _;
  }

  // Define a modifier that checks if an item.state of a upc is Sold
  modifier sold(uint _upc) {
    require(items[_upc].itemState == State.Sold, "The Item is not in Sold state!");
    _;
  }

  // Define a modifier that checks if an item.state of a upc is Shipped
  modifier shipped(uint _upc) {
    require(items[_upc].itemState == State.Shipped, "The Item is not in Shipped state!");
    _;
  }

  // Define a modifier that checks if an item.state of a upc is Received
  modifier received(uint _upc) {
    require(items[_upc].itemState == State.Received, "The Item is not in Received state!");
    _;
  }

  // Define a modifier that checks if an item.state of a upc is Purchased
  modifier purchased(uint _upc) {
    require(items[_upc].itemState == State.Purchased, "The Item is not in Purchased
state!");
    _;
  }

  // and set 'sku' to 1
  // and set 'upc' to 1
  // Using Ownable to define the ownwerm
  constructor() public payable {
    sku = 1;
    upc = 1;
  }

  // Define a function 'harvestItem' that allows a farmer to mark an item 'Harvested'
  function harvestItem(
  uint _upc,
  address payable _originFarmerID,
  string memory _originFarmName,
  string memory _originFarmInformation,
  string memory _originFarmLatitude,
  string memory _originFarmLongitude,
  string memory productNotes) public
  //Only Farmer
  onlyFarmer
  {
    // Add the new item as part of Harvest
    Item memory newItem;
    newItem.upc = _upc;
    newItem.ownerID = _originFarmerID;
    newItem.originFarmerID = _originFarmerID;
    newItem.originFarmName = _originFarmName;
    newItem.originFarmInformation = _originFarmInformation;
    newItem.originFarmLatitude = _originFarmLatitude;
    newItem.originFarmLongitude = _originFarmLongitude;
    newItem.productNotes = productNotes;
    newItem.sku = sku;
    newItem.productID = _upc + sku;
    // Increment sku
    sku = sku + 1;
    // Setting state
    newItem.itemState = State.Harvested;
    // Adding new Item to map
    items[_upc] = newItem;
    // Emit the appropriate event
    emit Harvested(_upc);
  }

  // Define a function 'processtItem' that allows a farmer to mark an item 'Processed'
  function processItem(uint _upc) public
  //Only Farmer
  onlyFarmer
  // Call modifier to check if upc has passed previous supply chain stage
  harvested(_upc)
  // Call modifier to verify caller of this function
  verifyCaller(items[_upc].originFarmerID)
  {
    // Update the appropriate fields
    Item storage existingItem = items[_upc];
    existingItem.itemState = State.Processed;
    // Emit the appropriate event
    emit Processed(_upc);
  }

  // Define a function 'packItem' that allows a farmer to mark an item 'Packed'
  function packItem(uint _upc) public
  //Only Farmer
  onlyFarmer
  // Call modifier to check if upc has passed previous supply chain stage
  processed(_upc)
```

```
  // Call modifier to verify caller of this function
  verifyCaller(items[_upc].originFarmerID)
  {
    // Update the appropriate fields
    Item storage existingItem = items[_upc];
    existingItem.itemState = State.Packed;
    // Emit the appropriate event
    emit Packed(_upc);
  }

  // Define a function 'sellItem' that allows a farmer to mark an item 'ForSale'
  function sellItem(uint _upc, uint _price) public
  //Only Farmer
  onlyFarmer
  // Call modifier to check if upc has passed previous supply chain stage
  packed(_upc)
  // Call modifier to verify caller of this function
  verifyCaller(items[_upc].originFarmerID)
  {
    // Update the appropriate fields
    Item storage existingItem = items[_upc];
    existingItem.itemState = State.ForSale;
    existingItem.productPrice = _price;
    // Emit the appropriate event
    emit ForSale(_upc);
  }

  // Define a function 'buyItem' that allows the disributor to mark an item 'Sold'
  // Use the above defined modifiers to check if the item is available for sale, if the buyer
has paid enough,
  // and any excess ether sent is refunded back to the buyer
  function buyItem(uint _upc) public payable
    // Only Distributor
    onlyDistributor
    // Call modifier to check if upc has passed previous supply chain stage
    forSale(_upc)
    // Call modifer to check if buyer has paid enough
    paidEnough(items[_upc].productPrice)
    // Call modifer to send any excess ether back to buyer
    checkValueForDistributor(_upc)
  {
  // Update the appropriate fields - ownerID, distributorID, itemState
  Item storage existingItem = items[_upc];
  existingItem.ownerID = msg.sender;
  existingItem.itemState = State.Sold;
  existingItem.distributorID = msg.sender;
  // Transfer money to farmer
  uint productPrice = items[_upc].productPrice;
  items[_upc].originFarmerID.transfer(productPrice);
  // emit the appropriate event
  emit Sold(_upc);
}

// Define a function 'shipItem' that allows the distributor to mark an item 'Shipped'
// Use the above modifers to check if the item is sold
function shipItem(uint _upc) public
  // Only Distributor
  onlyDistributor
  // Call modifier to check if upc has passed previous supply chain stage
  sold(_upc)
  // Call modifier to verify caller of this function
  verifyCaller(items[_upc].distributorID)
  {
  // Update the appropriate fields
  Item storage existingItem = items[_upc];
  existingItem.itemState = State.Shipped;
  // Emit the appropriate event
  emit Shipped(_upc);
}

// Define a function 'receiveItem' that allows the retailer to mark an item 'Received'
// Use the above modifiers to check if the item is shipped
function receiveItem(uint _upc) public
  // Only Retailer
  onlyRetailer
  // Call modifier to check if upc has passed previous supply chain stage
  shipped(_upc)
  // Access Control List enforced by calling Smart Contract / DApp
  {
  // Update the appropriate fields - ownerID, retailerID, itemState
  Item storage existingItem = items[_upc];
  existingItem.ownerID = msg.sender;
  existingItem.itemState = State.Received;
  existingItem.retailerID = msg.sender;
  // Emit the appropriate event
  emit Received(_upc);
}

// Define a function 'purchaseItem' that allows the consumer to mark an item 'Purchased'
// Use the above modifiers to check if the item is received
function purchaseItem(uint _upc) public payable
  //Only Consumer
  onlyConsumer
  // Call modifier to check if upc has passed previous supply chain stage
  received(_upc)
  // Make sure paid enough
  paidEnough(items[_upc].productPrice)
  // Access Control List enforced by calling Smart Contract / DApp
  checkValueForConsumer(_upc)
  {
  // Update the appropriate fields - ownerID, consumerID, itemState
    Item storage existingItem = items[_upc];
    existingItem.ownerID = msg.sender;
    existingItem.itemState = State.Purchased;
    existingItem.consumerID = msg.sender;
  // Emit the appropriate event
    emit Purchased(_upc);
}

// Define a function 'fetchItemBufferOne' that fetches the data
function fetchItemBufferOne(uint _upc) public view returns
(
uint      itemSKU,
uint      itemUPC,
address ownerID,
address originFarmerID,
string  memory originFarmName,
string  memory originFarmInformation,
string  memory originFarmLatitude,
string  memory originFarmLongitude
```

```
)
{
// Assign values to the 8 parameters
Item memory existingItem = items[_upc];

itemSKU = existingItem.sku;
itemUPC = existingItem.upc;
ownerID = existingItem.ownerID;
originFarmerID = existingItem.originFarmerID;
originFarmName = existingItem.originFarmName;
originFarmInformation = existingItem.originFarmInformation;
originFarmLatitude = existingItem.originFarmLatitude;
originFarmLongitude = existingItem.originFarmLongitude;

return
(
itemSKU,
itemUPC,
ownerID,
originFarmerID,
originFarmName,
originFarmInformation,
originFarmLatitude,
originFarmLongitude
);
}

// Define a function 'fetchItemBufferTwo' that fetches the data
function fetchItemBufferTwo(uint _upc) public view returns
(
uint     itemSKU,
uint     itemUPC,
uint    productID,
string  memory productNotes,
uint    productPrice,
uint    itemState,
address distributorID,
address retailerID,
address consumerID
)
```

```
  {
    // Assign values to the 9 parameters
  Item memory existingItem = items[_upc];
  itemSKU = existingItem.sku;
  itemUPC = existingItem.upc;
  productID = existingItem.productID;
  productNotes = existingItem.productNotes;
  productPrice = existingItem.productPrice;
  itemState = uint(existingItem.itemState);
  distributorID = existingItem.distributorID;
  retailerID = existingItem.retailerID;
  consumerID = existingItem.consumerID;

  return
  (
  itemSKU,
  itemUPC,
  productID,
  productNotes,
  productPrice,
  itemState,
  distributorID,
  retailerID,
  consumerID
  );
  }

}
```