

A Blockchain based Decentralized E-Marketplace

Submitted in partial fulfillment of the requirements for the
award of Bachelor of Engineering Degree in
Computer Science and Engineering

By

Deepak Arumugam S (Reg No: 40110302)
Jenish J. Bosco (Reg No: 40110216)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category – 1 University by UGC

**Accredited with Grade “A++” by NAAC | 12 B Status | Approved by
AICTE**

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI – 600119

APRIL 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category - I University by UGC

Accredited "A++" by NAAC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Deepak Arumugam S (40110302)** and **Jenish J. Bosco (40110216)** who carried out the project entitled **"A Blockchain based Decentralized E-Marketplace"** under my supervision from November 2023 to April 2024.

Internal Guide

Ms. Nancy Kirupanithi D, M.E.,

Head of the Department

Dr. L. Lakshmanan, M.E, Ph.D.

HEAD OF THE DEPARTMENT
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Submitted for Viva Voce Examination, held on 12/04/2024
Chennai - 600 119.

Internal Examiner

External Examiner

DECLARATION

Jenish J. Bosco (Reg. No. 40110216) hereby declare that the Project Report entitled on "**A Blockchain based Decentralized E-Marketplace**" done by me under the guidance of **Ms. Nancy Kirupanithi D, M.E.**, at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

Date: 12/04/2024

Place: Chennai


Signature of the Candidate

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to, **Dr. T. Sasikala M.E., Ph.D.**, Dean, School of Computing, **Dr. L. Lakshmanan, M.E., Ph.D.**, Head of the Department, Department of Computer Science and Engineering for providing me the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms. Nancy Kirupanithi D, M.E.**, for her valuable guidance, suggestions which paved the way for the successful completion of my project work.

I wish to express my thanks to all teaching and non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Over 1.8 billion people purchased goods online in 2018, and as a result, 2.8 trillion dollars were spent. Companies like Amazon, eBay, and PayPal thrive on being the middleman between sellers and buyers of online goods. We propose a framework for building a decentralized electronic marketplace for computing resources. The idea is that anyone with spare capacities can offer them on this marketplace, opening up the cloud computing market to smaller players, thus creating a more competitive environment compared to today's market consisting of a few large providers. decentralized E-marketplace platform utilizing the blockchain technology is implemented. We use the self-enforcement of smart contracts to secure the entire process of buying and selling. Our project uses Blockchain Technology to decentralize the online marketplace and remove the middleman as well as the fees associated with it. To do this, we use smart contracts in the Ethereum Blockchain. Each transaction is verified through the blockchain and is recorded to the decentralized ledger. This enables trustless transactions since the smart contract is self-executed. The smart contract is able to perform credible transactions without trusted third parties, and the transactions on the blockchain are trackable and irreversible. Therefore, both the buyer and the seller cannot breach the contract. All processes are recorded on the blockchain.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	v
	LIST OF FIGURES	vii
1	INTRODUCTION	1
	About the concept	1
2	LITERATURE SURVEY	3
	Inferences from the Literature Survey	6
	Open problems in Existing System	7
3	AIM AND SCOPE OF THE PRESENT INVESTIGATION	8
	Aim	9
	3.1 Scope of Present Investigation	10
4	REQUIREMENT ANALYSIS AND DESCRIPTION OF PROPOSED SYSTEM	11
	Feasibility studies of the project	11
	Requirements Specification	12
	Description of Proposed System	13
5	METHODS AND ALGORITHMS USED	14
	Decentralized System	14
	Blockchain	15
	Methodology	21
	System Architecture	23
6	RESULTS AND DISCUSSION	26
	Outputs and Screenshots	26
7	CONCLUSION	29
	REFERENCES	30

LIST OF FIGURES

FIGURE No.	NAME	PAGE No.
	Decentralized system	15
	Blockchain Transaction	17
	Consensus Mechanism	19
	Consensus Working	20
	System flow diagram	22
	System Architecture diagram	23
	Ganache establishing the Contract	25
	Packing In charge	26
	Production In Charge	27
	Customer Listing	27
	Supplier Listing	28
	Transaction Details	28
	Website Homepage	29
	Blockchain Report	29

CHAPTER 1

INTRODUCTION

ABOUT THE CONCEPT

Blockchain technology has been responsible for recent innovations in many industries. Blockchain can be defined as a chain of blocks that store transactions in a decentralized ledger. The use of blockchain would be for any network seeking “decentralization immutability transparency and auditability”. The internet of things (IoT) is characterized as a network of things connected wirelessly through smart sensors. Developments in these technologies have led to many breakthroughs in the agriculture industry and supply chain although many barriers still exist that prevent widespread adoption. This paper aims to explore the barriers preventing the use of blockchain technology along with the use of IoT in the agricultural supply chain.

Apart from the widespread data and service trading between companies and its users, there are specific data marketplaces which facilitate data trade by matching demand with information sources. Information providers or sellers showcase their data to woo potential buyers. Interested buyers search and select the information they want and acquire it in exchange for money. The marketplace (mediator) gets incentive for facilitating the trade and, in some cases, for hosting the data. But as data grow in value, so will the cases of cheating and leaks. A key aspect which the conventional data marketplaces tend to ignore is ensuring fair trade. The involved parties - sellers, buyers and mediators, being strategic players, would often want to collude and cheat in order to gain more money. Also, if the mediator hosts the complete trade, and buyer and seller have no communication, then the mediator is in a position of power. It can equivocate and forge for monetary gains.

The global supply chain market value surpassed. trillion US dollars having

increased at a compound annual growth rate of percentage. The supply chain plays a vital role in the global economy. Supply chain management which refers to the flow management of goods and services including all the processes that transform raw materials into final products along the supply chain is essential for boosting customer services reducing operation costs improving financial positions etc.

Among supply chain management services traceability is essential because it allows product tracking from the sources to end consumers. Supply chain traceability provides opportunities to enhance the supply chain efficiencies meet the regulatory requirements and most importantly to story-tell the consumers about the provenance and journey of products. Regarding the products whose safety is critical e.g. food and pharmaceuticals supply chain traceability is critical and has been pursued for decades by the industries. Despite its importance supply chain traceability is challenging primarily because of its undue reliance on the collaboration of multiple stakeholders who are not motivated to collaborate. More over there is a lack of mechanisms to define the minimum amount of data required from the stakeholders to achieve supply chain traceability. In existing studies, the researchers focused on modeling supply chain traceability especially the data among different stakeholders. The incentives of collaboration are understudied letting alone the safety and quality of the tracing process. The basic concept behind a Blockchain is that it is a decentralized immutable distributed ledger. Once data is written into a block and that block is added to the Blockchain, the data cannot be changed without rewriting the history of each Block that succeeded it. This would only be possible if the Blockchain was controlled by a single centralized entity. In order to guarantee the immutability we instead distribute the data among as many entities as possible in a trustless system where the consensus on what is correct is agreed when the majority says so. It is important to note that this is not just a distributed system since every node in a Blockchain contains the entire Blockchain, and not just parts of it, effectively decentralizing ownership of the Blockchain.

CHAPTER 2

LITERATURE SURVEY

1. **Blockchain Technology for Secure Supply Chain Management: A Comprehensive Review** by Udit Agarwal, Vinay Rishiwal, Sudeep Tanwar, Rashmi Chaudhary, Gulshan Sharma, Pitshou N. Bokoro and Ravi Sharma - This paper offers a comprehensive survey on Blockchain's adoption in Supply Chain Management (SCM), addressing prevalent issues like security, transparency, and stakeholder involvement. Analyzing 97 recent publications, it highlights Blockchain's role in enhancing transparency, traceability, and trust within SCM. The study identifies applications across diverse sectors, illustrating how Blockchain resolves transactional challenges. Additionally, it outlines research gaps, presenting opportunities for newcomers interested in delving into Blockchain technology's potential in SCM.
2. **Blockchain-Based Supply Chain Information Sharing Mechanism** by Sung-Jung Hsiao and Wen-Tsai Sung - This study introduces a blockchain-based approach to address the scarcity of information sharing in supply chains. It outlines methods for blockchain application, supply chain architecture design, and hierarchical information flow modeling. The research focuses on recording internal and external data sources within the supply chain using a structured block recording method, aiming to enhance supply chain integration and adaptability. It presents a five-tier model for supply chain information flow, emphasizing data division, composition, and a hash-based linkage system for blockchain-enabled data integration.
3. **A Blockchain-Based Approach for Drug Traceability in Healthcare Supply Chain** by ad Musamih, Khaled Salah, Raja Jayaraman, Junaid Arshad, Mazin Debe, Yousof AlHammadi and Samer Ellahham - This article addresses healthcare supply chain complexities by proposing an Ethereum blockchain-based solution for drug traceability. Emphasizing the need to combat counterfeit drugs, it introduces smart contracts and decentralized

storage to ensure data authenticity and transparency across the pharmaceutical supply chain. The proposed system offers end-to-end product tracking, eliminating intermediaries, ensuring data provenance, and providing secure transaction histories for stakeholders. It includes an architectural overview, algorithmic framework, and comprehensive testing to validate the system's effectiveness in enhancing pharmaceutical supply chain traceability while discussing smart contract security analysis.

4. **A Trusted Blockchain-Based Traceability System for Fruit and Vegetable Agricultural Products** by Xinting Yang , Mengqi Li, Huajing Yu, Mingting Wang, Daming Xu and Chuanheng Sun - This paper introduces a blockchain-based agricultural product traceability system, addressing issues of centralized management and unreliable data in supply chains. Utilizing blockchain's decentralized and tamper-proof nature, it enhances traceability transparency and credibility. The system implements a dual storage structure (database + blockchain) for on-chain and off-chain traceability data, improving query efficiency and ensuring private information security. It employs cryptography for safe information sharing, incorporates reputation-based smart contracts, and demonstrates enhanced query efficiency, data reliability, and security for supply chain management through practical application and performance analysis.
5. **Smart Contract-Based Agricultural Food Supply Chain Traceability** by Lu Wang, Longqin Xu, Zhiying Zheng, Shuangyin Liu, Xiangtong Li, Liang Cao, Jingbin Li and Chuanheng Sun - This paper introduces a blockchain-based framework for agricultural food supply chains, addressing complexities and distrust among participants. Utilizing a consortium and smart contracts, it tracks and shares supply chain workflows, enhancing transaction integrity and security while eliminating central institutions. Implementing Hyperledger smart contracts and QR codes enables end-to-end traceability, ensuring transparency from cultivation to consumer, thus enhancing trust between consumers and enterprises while preventing data tampering.

6. **User Interface of Blockchain-Based Agri-Food Traceability Applications:** A Review by Atima Tharatipyakul and Suporn Pongnumkul
-This paper scrutinizes blockchain-based agri-food supply chain applications, emphasizing user interface perspectives for usability enhancement. Reviewing 25 papers on blockchain and agri-food supply chains and 39 research papers displaying user interface screenshots, it categorizes user groups, data collection and visualization approaches. However, it highlights inadequate details on interface designs and minimal user involvement in blockchain research, underscoring potential usability issues. The study aims to bridge gaps in user interface design for improved blockchain adoption in agri-food traceability applications.
7. **Big Production Enterprise Supply Chain Endogenous Risk Management Based on Blockchain** by Yonggui Fu and Jianming Zhu -
This study focuses on applying blockchain to manage supply chain risks due to information asymmetry in large production enterprises. It outlines the blockchain's role in mitigating fraud, improving decision-making, and enabling group decisions by providing accurate information. The paper details the system structure, intelligent contract operations, data storage, and access mechanisms within the blockchain, emphasizing economic values like response speed, accuracy, cooperation integrity, and reduced economic costs. By addressing limitations and proposing solutions, this research offers a model for developing blockchain systems in supply chain management, aiming to advance blockchain applications in specific industries.

INFERENCES FROM LITERATURE SURVEY

Blockchain's Potential: The literature consistently highlights the immense potential of blockchain in addressing critical challenges within supply chains, such as transparency, traceability, data integrity, and trust issues

Enhanced Traceability and Transparency: Blockchain's decentralized and immutable nature provides enhanced traceability, transparency, and security, particularly in tracking product movements, ensuring authenticity, and combating counterfeiting in supply chains.

Usability and Interface Challenges: Some studies underscore a lack of emphasis on user interface design and usability considerations in blockchain-based applications. This could potentially hinder widespread adoption due to usability issues.

Risk Management: Blockchain emerges as a promising tool for mitigating risks inherent in supply chains, especially concerning information asymmetry and fraudulent activities among business entities.

Sector-Specific Applications: Research highlights sector-specific applications, demonstrating how blockchain technology can benefit various industries, including healthcare, agriculture, manufacturing, and pharmaceuticals, by improving efficiency and accountability.

Economic Value: Analysis indicates that implementing blockchain in supply chains can yield economic benefits such as improved response speed, supply accuracy, cooperation integrity, reduced costs, better quality, and pricing optimization.

Ongoing Development and Research Needs: While showcasing its potential, there's ongoing research aimed at addressing limitations, such as scalability, data access, and user involvement in blockchain-based solutions. Future studies are needed to fill gaps and enhance the practicality and usability of blockchain in supply

chain management.

PROBLEMS IN EXISTING SYSTEM

Cannot be changed after configuration. Cannot be implemented real time. May take huge time and economic cost to construct. Solutions have been proved ineffective. Computation burden may limit its further application for real scenarios. Cannot be implemented real time. It is not an easy-to-use method. Additional configuration is required. Difficult to be used in large-scale parallel computing. May take huge time and economic cost to construct. Small Networks can be compromised. High complexity, inaccuracy, and inadequacy Cannot be implemented real time Difficulties to obtain better performance large communication overhead. May take huge time and economic cost to construct. There are several other issues as well as follows:

Limited Digitalized Marketplaces: The text mentions the rarity of finding digitalized marketplaces specifically implemented for trading locally produced goods in developing countries. This implies a lack of existing infrastructure tailored to the needs of Small-Scale Producers (SSPs) in such regions.

Smart Contract Rigidity: While Smart Contracts offer advantages such as automation and transparency, they can also pose challenges due to their perceived rigidity. Bugs in Smart Contract code may be difficult to correct once deployed, potentially leading to issues in transaction execution or security vulnerabilities.

Security Concerns: While blockchain technology provides security advantages inherent to its design, such as immutability and consensus mechanisms, there may still be vulnerabilities to address, including potential attacks like Man in the Middle, data forgery, or eavesdropping.

Legislative Considerations: Legal aspects, such as the equivalent legal status of Smart Contracts to written contracts, need to be addressed. Incorporating

terms and conditions for legally binding contracts in the system's graphical user interface (GUI) can enhance legal clarity and user understanding.

Continuous Monitoring and Development: Given that blockchain technology is still in its early stages, ongoing monitoring and development are necessary to explore new applications and functionalities, ensuring the system remains relevant and effective.

Integration of Other Goods or Services: While the system focuses on trading goods, there's potential for expansion to include services or utilities like water and electricity, provided the necessary infrastructure supports such transfers.

Sensor Reliability and Data Integrity: Ensuring the reliability and integrity of data gathered from sensors is crucial, as any tampering could compromise the effectiveness of blockchain deployment. Measures such as using tamper-proof sensors or physical anti-tamper enhancements are recommended.

Product Information and Confidence: Enhancements are needed in the presentation of product information, including the addition of images and possibly other relevant details to increase customer confidence and improve the overall visual appeal of the website.

System Functionality: The existing system may lack certain functionalities that could enhance user experience and utility, such as a transaction explorer page, user registration and management system, and improved website performance and visual aesthetics.

CHAPTER 3

AIM AND SCOPE OF THE PRESENT INVESTIGATION

3.1 AIM

The primary aim of our project, "A Blockchain based Decentralized e-Marketplace" is to revolutionize the traditional online marketplace model by leveraging the power of blockchain technology and smart contracts. This project aims to create a decentralized crowdfunding platform that uses Ethereum tokens instead of traditional banking systems, which often come with high transaction fees and international transaction restrictions. By doing so, the project seeks to make crowdfunding more accessible, efficient, and secure.

3.1 SCOPE OF PRESENT INVESTIGATION

The scope of the present investigation for this project would involve:

1. Understanding Blockchain and Smart Contracts: A thorough understanding of blockchain technology, Ethereum tokens, and smart contracts is crucial. This includes how they operate, their benefits, and potential challenges in their implementation.
2. System Design: Designing a robust, secure, and user-friendly platform that can handle transactions efficiently. This involves creating an intuitive user interface, a secure transaction system, and a reliable database for storing transaction details.
3. Smart Contract Development: Developing smart contracts that govern the transactions on the platform. These contracts should be designed to automatically execute transactions when certain conditions are met, ensuring transparency and trust among users.
4. Security Measures: Implementing strong security measures to protect against

potential threats such as fraud, hacking, and data breaches. This includes encryption techniques, secure user authentication, and regular system audits.

5. Testing and Deployment: Rigorous testing of the platform to ensure its functionality, performance, and security. After successful testing, the platform can be deployed for public use.

6. Legal and Regulatory Compliance: Investigating the legal and regulatory aspects of using blockchain for crowdfunding, especially considering the use of Ethereum tokens instead of traditional currency. This may involve understanding laws and regulations in different countries, as blockchain regulations can vary widely.

By investigating these areas, your project can contribute significantly to the field of crowdfunding, offering a more accessible, secure, and efficient alternative to traditional methods.

CHAPTER 4

REQUIREMENT ANALYSIS AND DESCRIPTION OF PROPOSED SYSTEM

FEASIBILITY STUDIES OF THE PROJECT

Blockchain itself is known for its safety and security of the transactions, but as a Chinese proverb said “Even the brightest star has its imperfections.”. Following are some of the risks and challenges faced in this project.

Economic Feasibility

- **Regulatory Uncertainty:** The regulatory landscape for blockchain and cryptocurrencies is still evolving in many regions. Projects may face legal challenges and uncertainties related to compliance with financial regulations and securities laws.
- **Funding and Sustainability:** Like any startup, marketplace projects must secure sufficient funding and demonstrate long-term sustainability to succeed.
- **Market Competition:** The crowdfunding space is competitive, and established platforms have a significant market share. Newcomers must find ways to differentiate themselves and attract users.
- **Legal and Tax Implications:** Projects may need to navigate complex legal and tax issues related to the use of blockchain and cryptocurrencies.
- **Funding and Sustainability:** Like any startup, crowdfunding projects must secure sufficient funding and demonstrate long-term sustainability to succeed.

Technical Feasibility

- **Security Concerns:** Blockchain and smart contracts are not immune to security vulnerabilities. Hackers can exploit weaknesses in the code, leading to theft of funds or unauthorized access.
- **Scalability:** Scalability remains a challenge for many blockchain platforms. As

the number of users and transactions increases, some blockchain networks may experience slowdowns and increased costs.

- **Technology Risks:** Projects relying on cutting-edge technologies like blockchain can face technical challenges, including software bugs, network instability, and compatibility issues.
- **Smart Contract Risks:** Projects utilizing smart contracts must ensure the accuracy and security of their code. Bugs or vulnerabilities in smart contracts can lead to financial losses.
- **Operational Challenges:** Maintaining and upgrading a decentralized platform can be complex. Projects must plan for ongoing development, governance, and community management.

Social Feasibility

- **Lack of Trust:** Despite promises of transparency, some users may still lack trust in decentralized systems, especially if they have been victims of previous scams or hacks.

REQUIREMENTS SPECIFICATION

Hardware Requirements

- RAM : 4GB or above
- Processor or Frequency : 1.5GHz or above
- Processor : Intel Pentium 4 or above

Software Requirements

- Operating System : Windows 7 or above
- Languages : Python, Solidity and SQL
- Libraries : Ganache
- Tools used : PyCharm, SQL Server and Ganache

DESCRIPTION OF PROPOSED SYSTEM

In this paper, we will cover these goals and how we tackled them in our Blockchain marketplace by explaining the user interface and workflow. The main features that we wanted our market to have were: List products for sale, browse and buy products, cross-communication between different marketplaces using a decentralized database and Blockchain, transparency for transactions, use of automated escrow services that safely release payments upon delivery, no single point of failure, or censorship, all while maintaining the project as decentralized as possible. For transparency, we need to be able to keep track of all the transactions in the e-marketplace. More concisely, this means checking them for validity, so that participants cannot create fake transactions. They also need to be made persistent in a way such that they cannot be changed or forged afterwards. Finally, the transactions need to be traceable, so it is clear who entered a contract with whom and who is responsible when things start going wrong. In decentralized e-marketplaces the matching of buyers and sellers could be done in a more transparent way: a buyer has more options to choose from, increasing the likelihood of finding a good match. Fair Exchange, even between two parties, is impossible without a trusted third party. In our proposed design we want to use blockchain as an enforcer of rules and a trusted third party. It will contain logs of transactions that would help ascertain source and ownership of data being traded, which remains immutable because of the tamper-resistance property of blockchain. At the same time, it would also transfer incentives and act as an automated arbitrator in case of malicious behavior from parties. The smart contract is a protocol to establish a binding between the seller and the buyer, and the executions of the smart contract are triggered by events.

CHAPTER 5

METHODS AND ALGORITHMS USED

DECENTRALIZED SYSTEM

A decentralized system is a system where there is no single point of control or authority. planning and decision making are distributed evenly among several independent entities or nodes. Each node can act independently and autonomously, without relying on a central coordinator or leader. A decentralized system can be resilient to failures, censorship, and attacks, as there is no single point of failure or vulnerability. A decentralized system can also be scalable, as adding more nodes does not increase the complexity or overhead of the system. The opposite of a decentralized system is a centralized system, where control is exerted by just one entity.

A majority of the web applications we access on a daily basis are centralized, meaning that they are owned by a single entity who is responsible for configuring and managing the software code. This code can be executed on an individual computer, server or even a group of computers. systems like Twitter, Facebook, Instagram, Dropbox and Google Drive are examples of centralized systems.

An example of a decentralized system is Bitcoin blockchain, a decentralized ledger of transactions that uses a peer-to-peer network and a consensus algorithm to verify transactions and maintain a distributed ledger. Bitcoin does not have a central authority or server that controls the network or the currency. Instead, each node in the network can participate in validating transactions and creating new blocks of data. Bitcoin can resist attacks from malicious actors, as they would need to control more than half of the network's computing power to alter the ledger. Bitcoin can also accommodate more users and transactions, as adding more nodes does not affect the performance or security of the system. note this decentralized system is inherently a distributed system as well.

However, a decentralized system also has some trade-offs that need to be considered. For example:

- it may have lower efficiency or consistency than a centralized system, as it may require more communication or coordination among the nodes to reach consensus or agreement.
- it may also have lower usability or accessibility than a centralized system, as it may require more technical knowledge or skills from the users or developers.



Fig 5.1: Decentralized system

BLOCKCHAIN

A blockchain is a distributed ledger with growing lists of records (blocks) that are securely linked together via cryptographic hashes. Each block contains a cryptographic hash of the previous block, a timestamp, and transaction data (generally represented as a Merkle tree, where data nodes are represented by leaves). Since each block contains information about the previous block, they effectively form a chain (compare linked list data structure), with each additional block linking to the ones before it. Consequently, blockchain transactions are irreversible in that, once they are recorded, the data in any given block cannot be altered retroactively without altering all subsequent blocks.

Blockchains are typically managed by a peer-to-peer (P2P) computer network for use as a public distributed ledger, where nodes collectively adhere to a consensus

algorithm protocol to add and validate new transaction blocks. Although blockchain records are not unalterable, since blockchain forks are possible, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance.

A blockchain was created by a person (or group of people) using the name (or pseudonym) Satoshi Nakamoto in 2008 to serve as the public distributed ledger for bitcoin cryptocurrency transactions, based on previous work by Stuart Haber, W. Scott Stornetta, and Dave Bayer. The implementation of the blockchain within bitcoin made it the first digital currency to solve the double-spending problem without the need for a trusted authority or central server. The bitcoin design has inspired other applications and blockchains that are readable by the public and are widely used by cryptocurrencies. The blockchain may be considered a type of payment rail.

Structure and design

A blockchain is a decentralized, distributed, and often public, digital ledger consisting of records called blocks that are used to record transactions across many computers so that any involved block cannot be altered retroactively, without the alteration of all subsequent blocks. This allows the participants to verify and audit transactions independently and relatively inexpensively. A blockchain database is managed autonomously using a peer-to-peer network and a distributed timestamping server. They are authenticated by mass collaboration powered by collective self-interests. Such a design facilitates robust workflow where participants' uncertainty regarding data security is marginal. The use of a blockchain removes the characteristic of infinite reproducibility from a digital asset. It confirms that each unit of value was transferred only once, solving the long-standing problem of double spending. A blockchain has been described as a value-exchange protocol. A blockchain can maintain title rights because, when properly set up to detail the exchange agreement, it provides a record that compels offer and acceptance.

Logically, a blockchain can be seen as consisting of several layers:

- infrastructure (hardware)
- networking (node discovery, information propagation and verification)
- consensus (proof of work, proof of stake)
- data (blocks, transactions)
- application (smart contracts/decentralized applications, if applicable)

Blocks hold batches of valid transactions that are hashed and encoded into a Merkle tree. Each block includes the cryptographic hash of the prior block in the blockchain, linking the two. The linked blocks form a chain. This iterative process confirms the integrity of the previous block, all the way back to the initial block, which is known as the genesis block (Block 0). To assure the integrity of a block and the data contained in it, the block is usually digitally signed.



Fig 5.2: Blockchain Transaction

Consensus Algorithm:

We are going to use Consensus Algorithm instead of Proof-of-Supply chain. It can improve Network Topology, Provides Comprehensive Decentralization Flexible Trust & Asymptotic Security. The main features that we wanted our market to have were: List products for sale, browse and buy products, cross-communication between different marketplaces using a decentralized database and Blockchain, transparency for transactions, use of

automated escrow services that safely release payments upon delivery, no single point of failure, or censorship, all while maintaining the project as decentralized as possible. For transparency, we need to be able to keep track of all the transactions in the e-marketplace. More concisely, this means checking them for validity, so that participants cannot create fake transactions. Utilizing a consensus algorithm instead of Proof-of-SupplyChain (PoSC) for managing blockchain in supply chain applications involves selecting an appropriate consensus mechanism that ensures agreement and validity of transactions across the distributed network. Here are some commonly used consensus algorithms suitable for supply chain management in blockchain:

Proof of Authority (PoA): This consensus model relies on authorized validators, typically selected by the network, who verify transactions and create new blocks. PoA emphasizes identity and authority to ensure transaction validity and maintain network security.

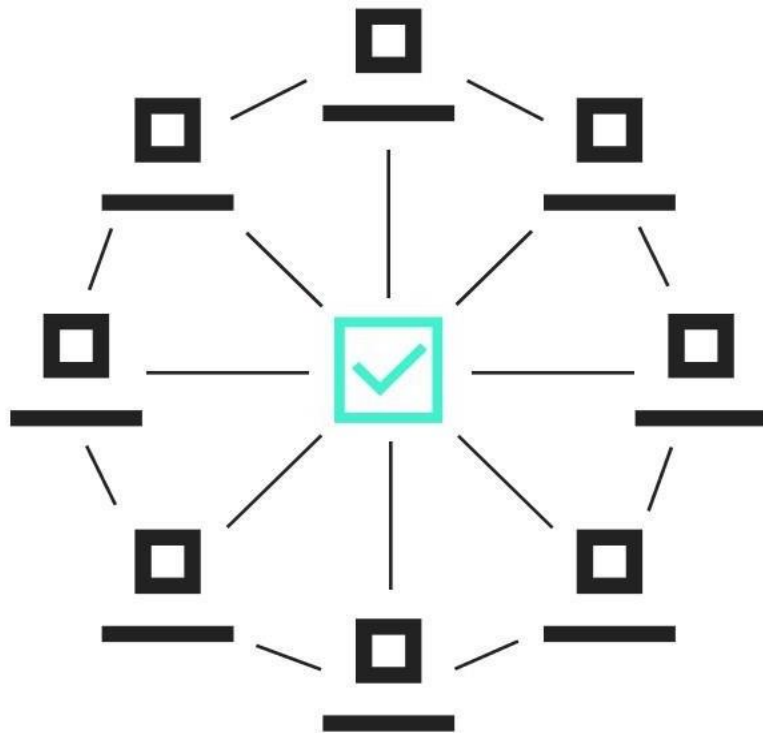
Proof of Stake (PoS): PoS allocates block creation rights based on the amount of cryptocurrency participants hold or "stake." It aims to achieve consensus by considering the economic stake participants have in the network.

Delegated Proof of Stake (DPoS): DPoS operates similarly to PoS but introduces a delegated voting system where stakeholders elect representatives to validate transactions on their behalf. It aims for faster transactions and scalability.

Practical Byzantine Fault Tolerance (PBFT): PBFT relies on a deterministic algorithm where nodes reach consensus by communicating and agreeing on a single decision, even in the presence of faulty or malicious nodes.

Proof of Elapsed Time (PoET): PoET is a consensus mechanism developed by Intel that uses a random leader election process. It aims to reduce energy consumption by allowing nodes to "wait" for a randomly selected time before proposing a new block.

Selecting the right consensus algorithm depends on various factors like scalability requirements, network participants' trust levels, energy efficiency, and the desired level of decentralization. It's essential to assess these factors against the specific needs and goals of the supply chain application to determine the most suitable consensus mechanism to ensure efficient and secure transaction validation in the blockchain network.



Decentralized Consensus

Fig 5.3: Consensus Mechanism

Consensus Mechanism



Fig 5.4: Consensus Working

METHODOLOGY

Module 1: Record Creation

The fundamental stakeholders of a data marketplace are the sellers and the buyers. The buyers and sellers should be able to easily sell and buy data via the marketplace.

Buyer: We assume that there are a set of buyers interested in buying data from a data marketplace.

E.g. the buyers can be stakeholders from the recycling domain who can use the data to enhance their products and the recycling process.

Seller: We suppose that there are sellers who are ready to sell their data in the marketplace.

Module 2: Information Query

We differentiate the actual data transfer and the transaction. The challenge associated with the transaction is that there is a requirement of a secure platform also the parties involved has to trust each other as the transaction may be tampered or denied later. For this problem we propose smart contract as a medium for the transactions. A smart Contract has self-executing scripts that reside on the blockchain. When a pre-configured condition in a smart contract among participating entities is met then the parties involved in a contractual agreement can be automatically made payments as per the contract in a transparent manner. Smart contract facilitates transparency of transaction to all concerned parties as once executed its immutable.

Module 3: Blockchain Creation

Blockchain technology is a distributed database of public or private records or ledgers of digital events (transactions) that have been shared and executed among the participants. The basis of blockchain technology is distributed ledger technology. Blockchain technology has four key characteristics that distinguish it from other available information systems. These characteristics include lack of localization (decentralization) security controllability and auditability smart implementation (smart contract).

In Blockchain a participant creates a new transaction to add it to the Blockchain. The new transaction is distributed throughout the network to be verified and audited. When the majority of blockchain nodes verify the transaction(according to predefined rules) the new transaction is added to the chain as a new block. A transaction record is

distributed and saved among several nodes to ensure security. At the same time smart contract as one of the most important features of blockchain provides valid transactions without requiring the involvement of third parties. Decentralization is another significant feature of Blockchain technology that detects and prevents fraud in information and consequently increases the validity of the data

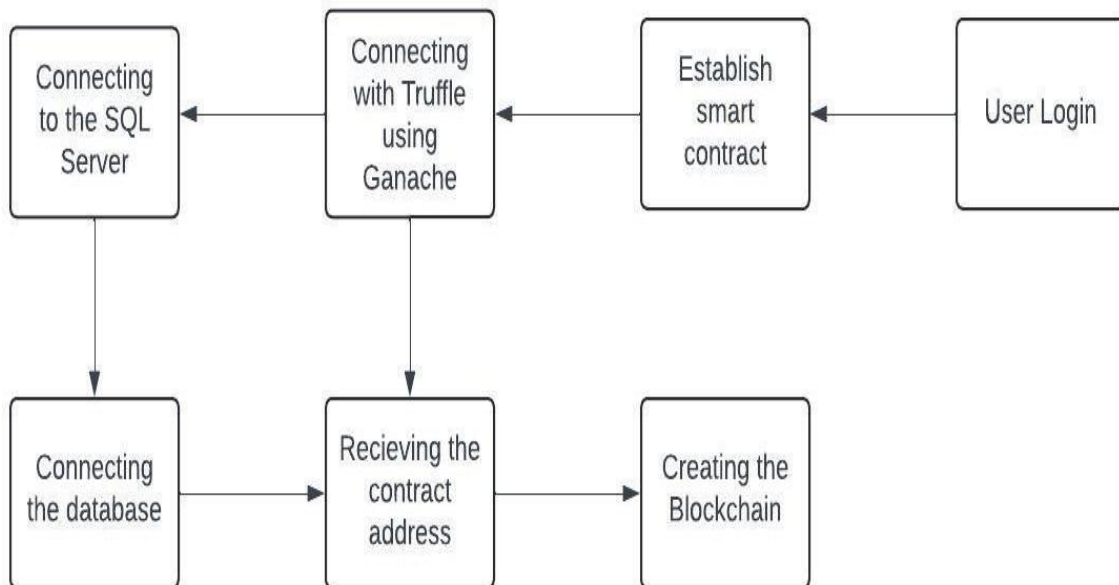


Fig 5.5: System flow diagram

SYSTEM ARCHITECTURE

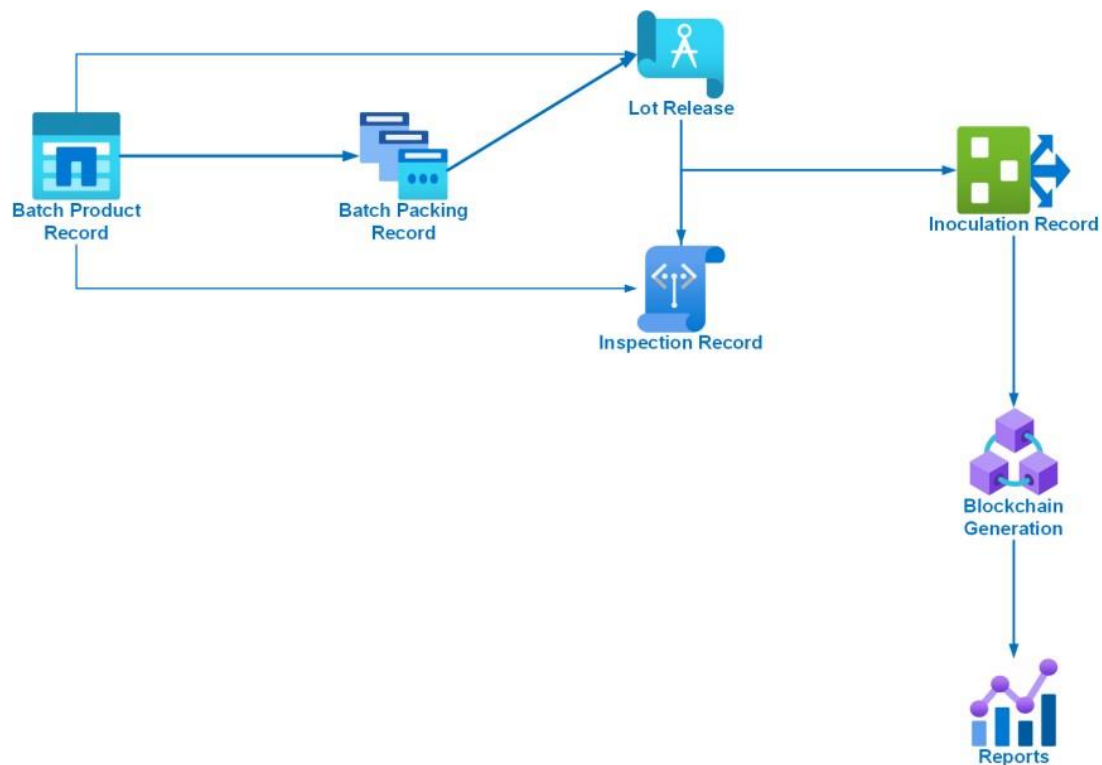


Fig 5.6: System Architecture diagram

In the block diagram of the above system the user logs in through the blockchain wallet that he/she owns and that connects to the smart contract through the front end which is created using ReactJS and Thirdweb SDK. The Thirdweb SDK helps us to mutate the data stored in the blockchain by connecting to the smart contract from the frontend. It sends and receives data from the backend which is created using Solidity. After this the transaction information and the money (cryptocurrency) is transferred from user A to user B through the Ethereum network (This happens in the backend).

The implementation of our crowdfunding platform involves the integration of various technologies and tools. We will utilize the following components:

1) Solidity:

Solidity is an object-oriented, high-level programming language for implementing smart contracts that run on the Ethereum Virtual Machine (EVM). Solidity is inspired by well-known programming languages like C++ and JavaScript. As a web3 developer, working with Solidity is beneficial to learning web3 development.

This section introduces you to the history of Solidity, establishes similarities between Solidity and other programming languages, and explains the main use cases for Solidity.

2) SQL:

SQL (Structured Query Language) is a language to operate databases; it includes Database Creation, Database Deletion, Fetching Data Rows, Modifying & Deleting Data rows, etc.

SQL stands for Structured Query Language which is a computer language for storing, manipulating and retrieving data stored in a relational database. SQL was developed in the 1970s by IBM Computer Scientists and became a standard of the American National Standards Institute (ANSI) in 1986, and the International Organization for Standardization (ISO) in 1987.

3) Python:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse

4) Ganache:

Ganache is a personal blockchain for rapid Ethereum and Filecoin distributed application development. You can use Ganache across the entire development cycle; enabling you to develop, deploy, and test your dApps in a safe and

deterministic environment.

Ganache comes in two flavors: a UI and CLI. Ganache UI is a desktop application supporting Ethereum and Filecoin technology. Our more robust command-line tool, ganache, is available for Ethereum development.

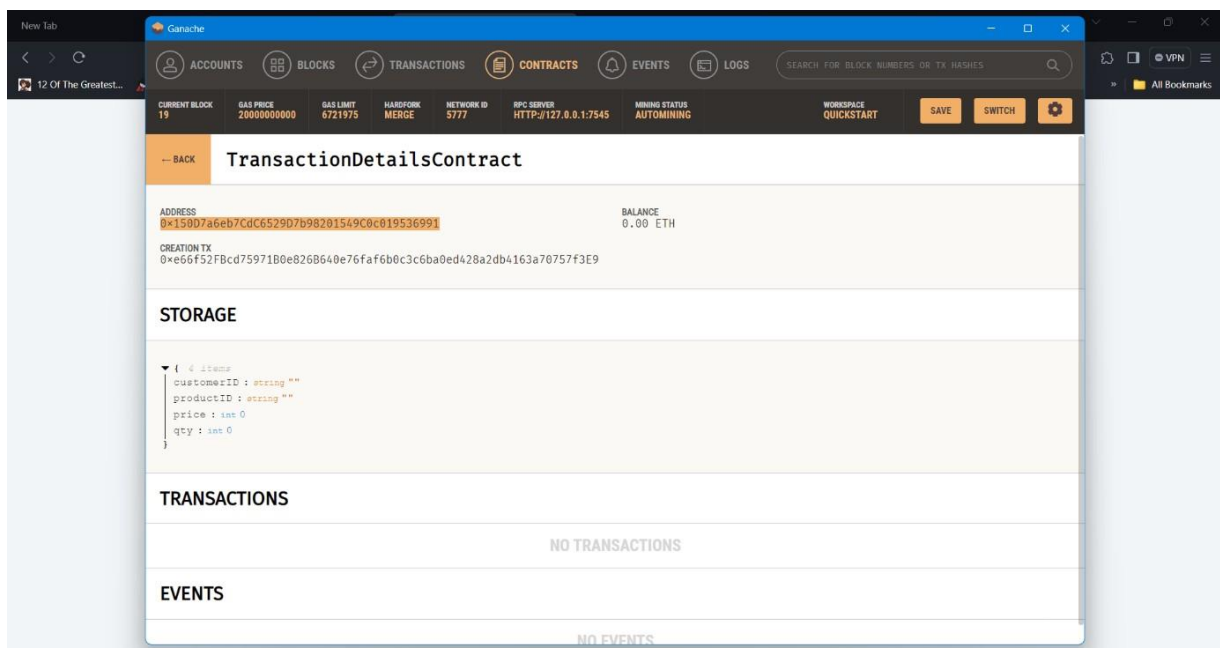


Fig 5.7: Ganache establishing the Contract

CHAPTER 6

RESULTS AND DISCUSSION

Having a decentralized way for transactions is better in many ways, such as the transaction time and its fees being much lower compared to what they would have been if it were done through centralized means, i.e., a bank.

Furthermore, the security, transparency, and availability of a blockchain-based system are leagues better than those of a centralized one. As a blockchain network is near impossible to hack into due to its peer-to-peer connectivity and decentralized nature, the transaction is permanent and cannot be changed or modified once done. This provides a much better and safer alternative to the usual methods.

The screenshots of the project are as follows:

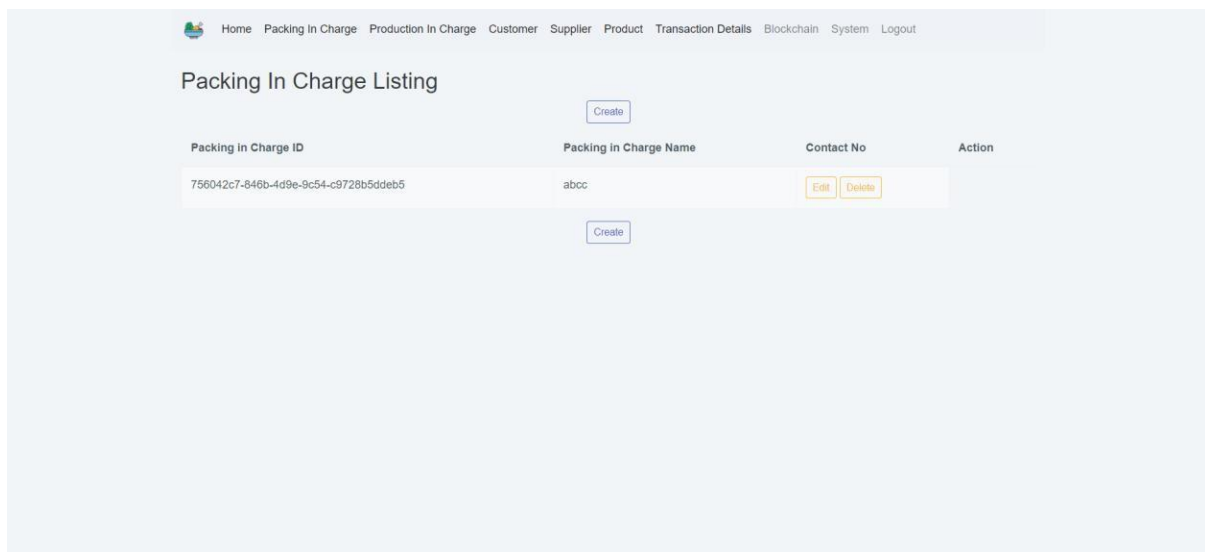


Fig 6.1: Packing In charge

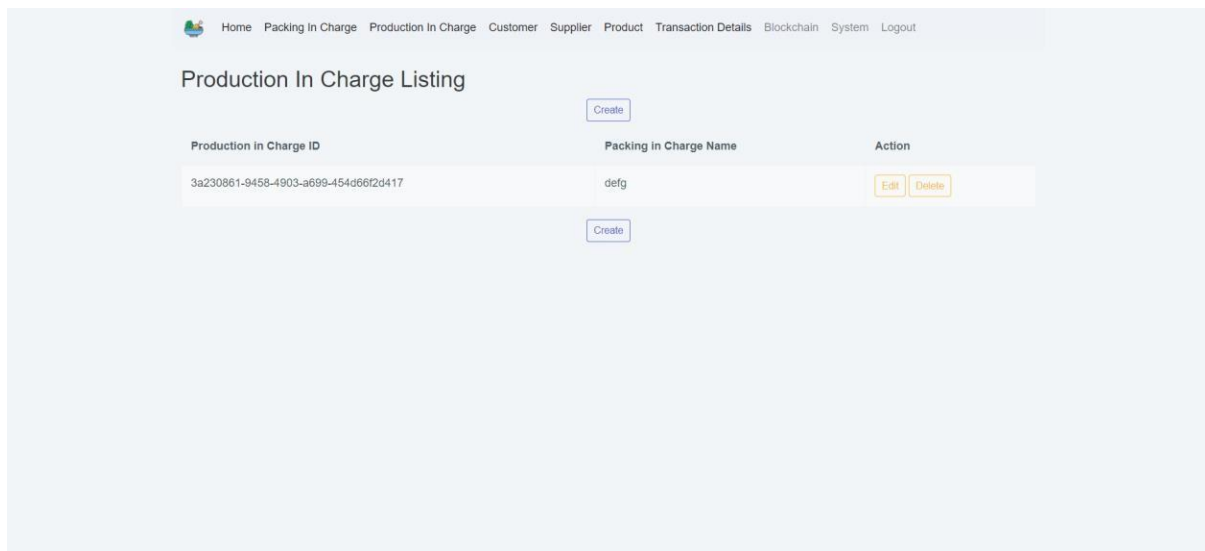


Fig 6.2: Production In Charge

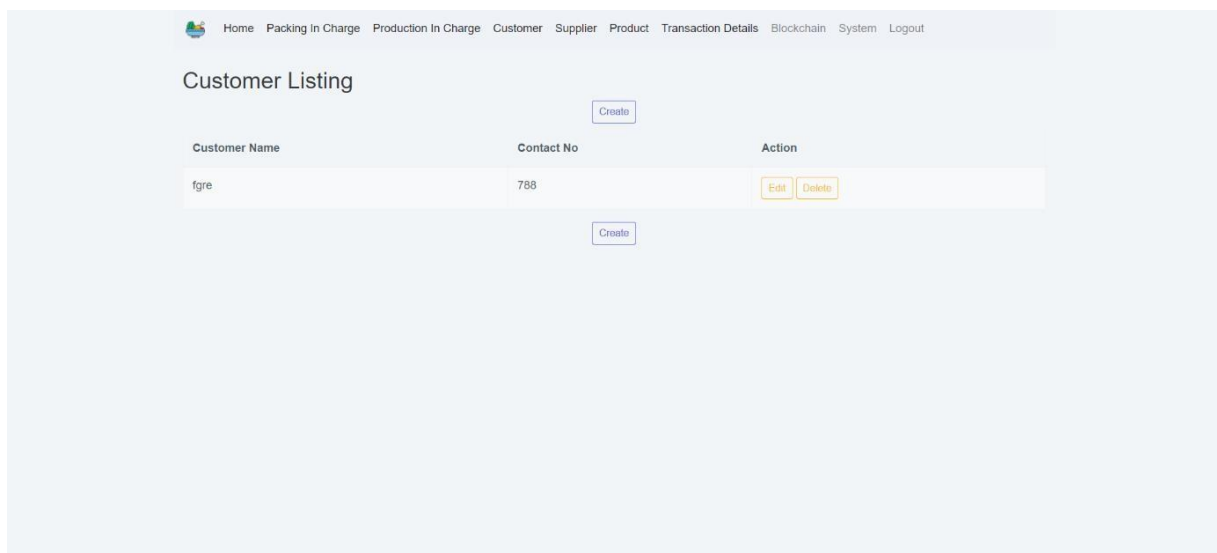


Fig 6.3: Customer Listing

	Home	Packing In Charge	Production In Charge	Customer	Supplier	Product	Transaction Details	Blockchain	System	Logout
Supplier Listing										
Create										
Supplier Name		Contact No		Action						
xdxd		312456		EditDelete						
Create										

: *Supplier Listing*

	Home	Packing In Charge	Production In Charge	Customer	Supplier	Product	Transaction Details	Blockchain	System	Logout
Transaction Details Listing										
Create										
Sales Order ID			Eff Date			Action				
454d3a90-a0d8-4d12-b30d-64c151c24431			2024-03-26			EditDelete				
Create										

: *Transaction Details*

6.1 OUTPUTS AND SCREENSHOTS

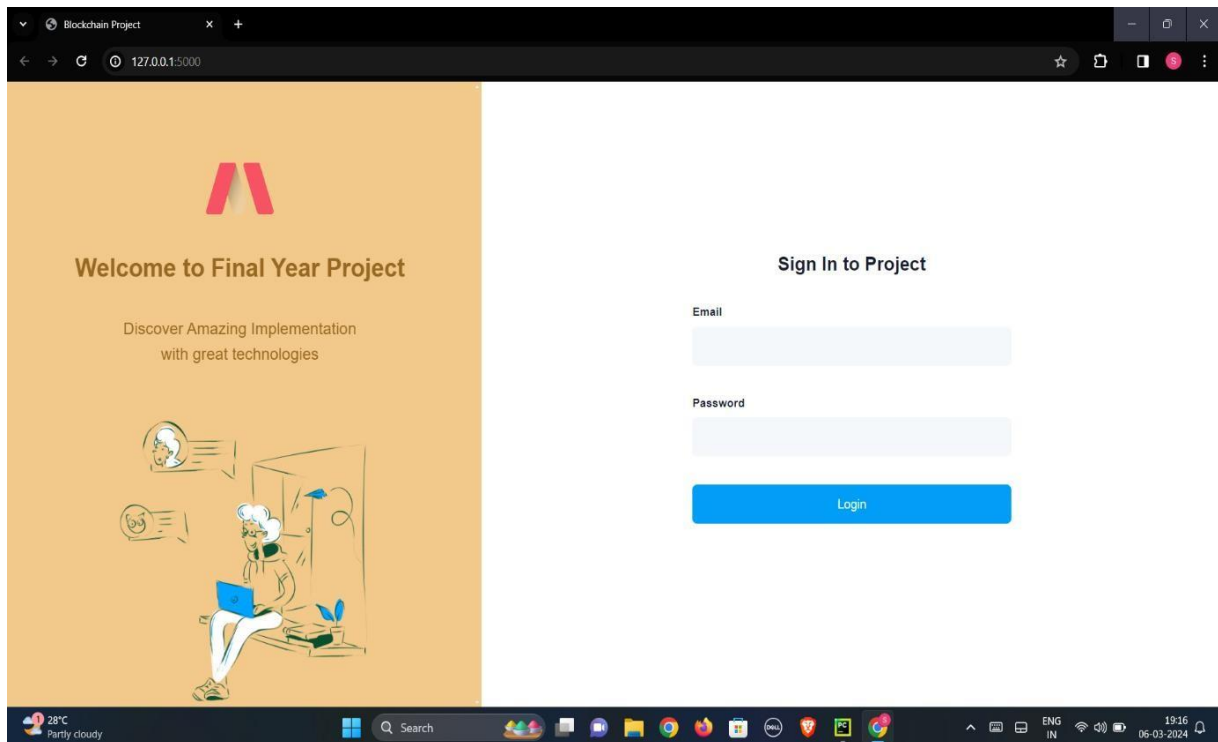


Fig 6.6: Website Homepage

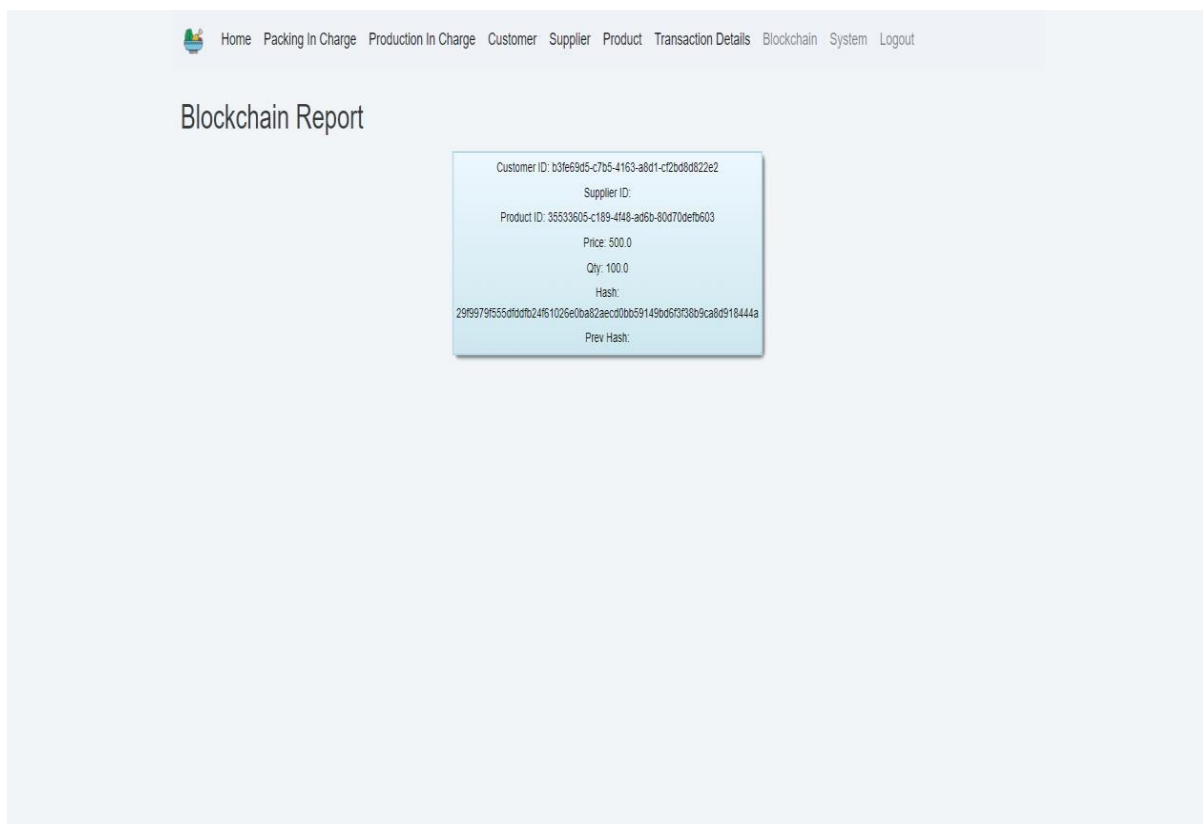


Fig 6.7: Blockchain Report

CHAPTER 7

CONCLUSION AND FUTURE WORK

We have enumerated the properties desired of any practical data marketplace, outlined ways and means to design such a marketplace which facilitates trade. Primarily being an idea paper, the exact instantiation of methods and possible tools to be used are not concretely specified. The system design in its entirety is under construction. Implementation of a prototype is ongoing work through which we aim to further understand implementation issues like scalability and performance. The long-term goal is to design and implement a production grade marketplace enabling data trade.

This study has examined the procedures involved as well as the needs of the marketplace system using a solidity program. This study could compile a smart contract composed of data structures and functions, and future research could concentrate on putting smart contracts to the test in the performance section.

REFERENCES

- [1] Yonggui Fu, Jianming Zhu. Big Production Enterprise Supply Chain Endogenous Risk Management Based on Blockchain IEEE Access, 2019.
- [2] Udit Agarwal, Vinay Rishiwal, Sudeep Tanwar, Rashmi Chaudhary, Gulshan Sharma, Pitshou N. Bokoro, Ravi Sharma. Blockchain Technology for Secure Supply Chain Management: A Comprehensive Review IEEE Access, 2022.
- [3] Sung-Jung Hsiao, Wen-Tsai Sung. Blockchain-Based Supply Chain Information Sharing Mechanism IEEE Access, 2022.
- [4] H.C. Pfohl, P. Gallus and D. Thomas, Interpretive structural modeling of supply chain risks, *Int. J. Phys. Distrib. Logistics Manage.*, vol. 41, pp. 839-859, 2011.
- [5] M. L. Xu, X. Z. Nie and H. Y. Jian, Pricing decision of dual-channel supply chain with risk preference, *Control Decis.*, vol. 31, no. 1, pp. 91-97, 2016.
- [6] J.S. Dai, W. D. Meng and B. Fan, Supply chain coordination with risk aversion via buy-back contracts, *J. Manage. Sci. China*, vol. 18, no. 5, pp. 57-65, 2015.
- [7] G. Zhao, Y. B. Yang and X. Bao, Dynamic model for the risk spreading in supply chain network and its application, *Syst. Eng.-Theory Pract.*, vol. 35, no. 8, pp. 2014-2024, 2015.
- [8] B. Y. Wang, Research on corporate social responsibility risk and transmission mechanism in supply chain system, *Electron. Problems*, no. 3, pp. 80-84, 2015.
- [9] M.J. Labaran and M. Hamma-Adama, The Nigerian pharmaceutical supply chain: Blockchain adoption counterfeit drugs and successful deployment of COVID-19 vaccine in Nigeria, *J. Sci. Res. Rep.*, vol. 27, no. 2, pp. 20-36, Mar. 2021.
- [10] M. Sharma and G. Sikka, Blockchain based approaches for preventing drug counterfeit: A survey, *Int. J. Eng. Res. Technol.*, vol. 9, no. 8, pp. 1-6, 2021.
- [11] A. Batwa and A. Norrman, Blockchain technology and trust in supply chain management: A literature review and research agenda, *Oper. Supply Chain Manage. Int. J.*, vol. 14, no. 2, pp. 203-220, Jan. 2021.
- [12] R. Raman, J. Sushmitha and M. K. Nalini, A survey paper on blockchain technologies in supply chain management, *Int. J. Res. Eng. Sci.*, vol. 9, no. 6, pp. 79-86, 2021.

APPENDIX

CODE:

PACKING MODEL:

```
from Constants import connString
import pyodbc
import datetime
import uuid
import time

class PackingInChargeModel:
    def __init__(self, packingInChargeID = "", packingInChargeName = "", isActive =
False):
        self.packingInChargeID = packingInChargeID
        self.packingInChargeName = packingInChargeName
        self.isActive = isActive

    @staticmethod
    def get_all():
        conn = pyodbc.connect(connString, autocommit=True)
        cursor = conn.cursor()
        sqlcmd1 = "SELECT * FROM PackingInCharge ORDER BY
packingInChargeName"
        cursor.execute(sqlcmd1)
        records = []
        for dbrow in cursor.fetchall():
            row = PackingInChargeModel(dbrow[0], dbrow[1], dbrow[2])
            records.append(row)
```

```
cursor.close()
conn.close()
return records
```

```
@staticmethod
def get_name_id():
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT packingInChargeID, packingInChargeName FROM
PackingInCharge ORDER BY packingInChargeName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = PackingInChargeModel(dbrow[0],dbrow[1])
        records.append(row)
    cursor.close()
    conn.close()
    return records
```

```
@staticmethod
def get_by_id(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM PackingInCharge WHERE packingInChargeID
= ?"
    cursor.execute(sqlcmd1, unique_id)
    record = None
    for dbrow in cursor.fetchall():
        record = PackingInChargeModel(dbrow[0],dbrow[1],dbrow[2])
    cursor.close()
    conn.close()
    return record
```

```
@staticmethod
```

```

def insert(obj):
    obj.packingInChargeID = str(uuid.uuid4())
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "INSERT INTO PackingInCharge
(packingInChargeID,packingInChargeName,isActive) VALUES(?,?,?)"
    cursor.execute(sqlcmd1,
(obj.packingInChargeID,obj.packingInChargeName,obj.isActive))
    cursor.close()
    conn.close()

```

```

@staticmethod
def update(obj):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "UPDATE PackingInCharge SET packingInChargeName =
?,isActive = ? WHERE packingInChargeID = ?"
    cursor.execute(sqlcmd1,
(obj.packingInChargeName,obj.isActive,obj.packingInChargeID))
    cursor.close()
    conn.close()

```

```

@staticmethod
def delete(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "DELETE FROM PackingInCharge WHERE packingInChargeID =
?"
    cursor.execute(sqlcmd1, (unique_id))
    cursor.close()
    conn.close()

```

PRODUCTION MODEL:

```
from Constants import connString
import pyodbc
import datetime
import uuid
import time
```

```
class ProductionInChargeModel:
```

```
    def init____(self, productionInChargeID = "",productionInChargeName =
    "",isActive = False):
        self.productionInChargeID = productionInChargeID
        self.productionInChargeName = productionInChargeName
        self.isActive = isActive
```

```
@staticmethod
```

```
def get_all():
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM ProductionInCharge ORDER BY
productionInChargeName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = ProductionInChargeModel(dbrow[0],dbrow[1],dbrow[2])
        records.append(row)
    cursor.close()
    conn.close()
    return records
```

```

@staticmethod
def get_name_id():
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT productionInChargeID, productionInChargeName
FROM ProductionInCharge ORDER BY productionInChargeName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = ProductionInChargeModel(dbrow[0],dbrow[1])
        records.append(row)
    cursor.close()
    conn.close()
    return records

```

```

@staticmethod
def get_by_id(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM ProductionInCharge WHERE
productionInChargeID = ?"
    cursor.execute(sqlcmd1, unique_id)
    record = None
    for dbrow in cursor.fetchall():
        record = ProductionInChargeModel(dbrow[0],dbrow[1],dbrow[2])
    cursor.close()
    conn.close()
    return record

```

```

@staticmethod
def insert(obj):
    obj.productionInChargeID = str(uuid.uuid4())
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()

```

```

        sqlcmd1 = "INSERT INTO ProductionInCharge
(productionInChargeID,productionInChargeName,isActive) VALUES(?,?,?)"
        cursor.execute(sqlcmd1,
(obj.productionInChargeID,obj.productionInChargeName,obj.isActive))
        cursor.close()
        conn.close()

```

```

@staticmethod
def update(obj):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "UPDATE ProductionInCharge SET productionInChargeName =
?,isActive = ? WHERE productionInChargeID = ?"
    cursor.execute(sqlcmd1,
(obj.productionInChargeName,obj.isActive,obj.productionInChargeID))
    cursor.close()
    conn.close()

```

```

@staticmethod
def delete(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "DELETE FROM ProductionInCharge WHERE
productionInChargeID = ?"
    cursor.execute(sqlcmd1, (unique_id))
    cursor.close()
    conn.close()

```

PRODUCT MODEL:

```
from Constants import connString
import pyodbc
import datetime
import uuid
import time
```

```
class ProductModel:
    def __init__(self, productID = "",productName = "",packageSize = "",price = 0):
        self.productID = productID
        self.productName = productName
        self.packageSize = packageSize
        self.price = price
```

```
@staticmethod
```

```
def get_all():
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM Product ORDER BY productName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = ProductModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3])
        records.append(row)
    cursor.close()
    conn.close()
    return records
```

```
@staticmethod
```

```

def get_name_id():
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT productID, productName FROM Product ORDER BY
productName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = ProductModel(dbrow[0],dbrow[1])
        records.append(row)
    cursor.close()
    conn.close()
    return records

```

@staticmethod

```

def get_by_id(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM Product WHERE productID = ?"
    cursor.execute(sqlcmd1, unique_id)
    record = None
    for dbrow in cursor.fetchall():
        record = ProductModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3])
    cursor.close()
    conn.close()
    return record

```

@staticmethod

```

def insert(obj):
    obj.productID = str(uuid.uuid4())
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "INSERT INTO Product
(productID,productName,packageSize,price) VALUES(?,?,?,?)"

```

```
        cursor.execute(sqlcmd1,  
(obj.productID,obj.productName,obj.packageSize,obj.price))  
        cursor.close()  
        conn.close()
```

```
@staticmethod  
def update(obj):  
    conn = pyodbc.connect(connString, autocommit=True)  
    cursor = conn.cursor()  
    sqlcmd1 = "UPDATE Product SET productName = ?,packageSize = ?,price =  
? WHERE productID = ?"  
    cursor.execute(sqlcmd1,  
(obj.productName,obj.packageSize,obj.price,obj.productID))  
    cursor.close()  
    conn.close()
```

```
@staticmethod  
def delete(unique_id):  
    conn = pyodbc.connect(connString, autocommit=True)  
    cursor = conn.cursor()  
    sqlcmd1 = "DELETE FROM Product WHERE productID = ?"  
    cursor.execute(sqlcmd1, (unique_id))  
    cursor.close()  
    conn.close()
```

CUSTOMER MODEL:

```
from Constants import connString
import pyodbc
import datetime
import uuid
import time
```

```
class CustomerModel:
```

```
    def __init__(self, customerID = "",customerName = "",contactNbr = "",emailID =
    "",address = "",city = "",county = "",postcode = "",country = "",experience = 0,emailModel
    = None):
```

```
        self.customerID = customerID
```

```
        self.customerName = customerName
```

```
        self.contactNbr = contactNbr
```

```
        self.emailID = emailID
```

```
        self.address = address
```

```
        self.city = city
```

```
        self.county = county
```

```
        self.postcode = postcode
```

```
        self.country = country
```

```
        self.experience = experience
```

```
        self.emailModel = emailModel
```

```
@staticmethod
```

```
def get_all():
```

```
    conn = pyodbc.connect(connString, autocommit=True)
```

```
    cursor = conn.cursor()
```

```
    sqlcmd1 = "SELECT * FROM Customer ORDER BY customerName"
```

```
    cursor.execute(sqlcmd1)
```

```

records = []
for dbrow in cursor.fetchall():
    row = CustomerModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],dbrow[6],
dbrow[7],dbrow[8],dbrow[9])
    records.append(row)
cursor.close()
conn.close()
return records

```

```

@staticmethod
def get_name_id():
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT customerID, customerName FROM Customer ORDER
BY customerName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = CustomerModel(dbrow[0],dbrow[1])
        records.append(row)
    cursor.close()
    conn.close()
    return records

```

```

@staticmethod
def get_by_id(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM Customer WHERE customerID = ?"
    cursor.execute(sqlcmd1, unique_id)
    record = None
    for dbrow in cursor.fetchall():
        record =

```

```

CustomerModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],dbrow[6],
dbrow[7],dbrow[8],dbrow[9])
    cursor.close()
    conn.close()
    return record

@staticmethod
def insert(obj):
    obj.customerID = str(uuid.uuid4())
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "INSERT INTO Customer
(customerID,customerName,contactNbr,emailID,address,city,county,postcode,co
untry,experience) VALUES(?,?,?,?,?,?,?,?,?,?)"
    cursor.execute(sqlcmd1,
(obj.customerID,obj.customerName,obj.contactNbr,obj.emailID,obj.address,obj.cit
y,obj.county,obj.postcode,obj.country,obj.experience))
    cursor.close()
    conn.close()

```

```

@staticmethod
def update(obj):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "UPDATE Customer SET customerName = ?,contactNbr =
?,emailID = ?,address = ?,city = ?,county = ?,postcode = ?,country = ?,experience
= ? WHERE customerID = ?"
    cursor.execute(sqlcmd1,
(obj.customerName,obj.contactNbr,obj.emailID,obj.address,obj.city,obj.county,obj
.postcode,obj.country,obj.experience,obj.customerID))
    cursor.close()
    conn.close()

```

```

@staticmethod
def delete(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "DELETE FROM Customer WHERE customerID = ?"
    cursor.execute(sqlcmd1, (unique_id))
    cursor.close()
    conn.close()

```

PACKING INCHARGE MODEL:

```

from Constants import connString
import pyodbc
import datetime
import uuid
import time

class PackingInChargeModel:
    def __init__(self, packingInChargeID = "", packingInChargeName = "", isActive = False):
        self.packingInChargeID = packingInChargeID
        self.packingInChargeName = packingInChargeName
        self.isActive = isActive

    @staticmethod
    def get_all():
        conn = pyodbc.connect(connString, autocommit=True)
        cursor = conn.cursor()
        sqlcmd1 = "SELECT * FROM PackingInCharge ORDER BY packingInChargeName"

```

```

cursor.execute(sqlcmd1)
records = []
for dbrow in cursor.fetchall():
    row = PackingInChargeModel(dbrow[0],dbrow[1],dbrow[2])
    records.append(row)
cursor.close()
conn.close()
return records

```

```

@staticmethod
def get_name_id():
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT packingInChargeID, packingInChargeName FROM
PackingInCharge ORDER BY packingInChargeName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = PackingInChargeModel(dbrow[0],dbrow[1])
        records.append(row)
    cursor.close()
    conn.close()
    return records

```

```

@staticmethod
def get_by_id(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM PackingInCharge WHERE packingInChargeID
= ?"
    cursor.execute(sqlcmd1, unique_id)
    record = None
    for dbrow in cursor.fetchall():
        record = PackingInChargeModel(dbrow[0],dbrow[1],dbrow[2])

```

```

cursor.close()
conn.close()
return record

```

```

@staticmethod
def insert(obj):
    obj.packingInChargeID = str(uuid.uuid4())
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "INSERT INTO PackingInCharge
(packingInChargeID,packingInChargeName,isActive) VALUES(?,?,?)"
    cursor.execute(sqlcmd1,
(obj.packingInChargeID,obj.packingInChargeName,obj.isActive))
    cursor.close()
    conn.close()

```

```

@staticmethod
def update(obj):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "UPDATE PackingInCharge SET packingInChargeName =
?,isActive = ? WHERE packingInChargeID = ?"
    cursor.execute(sqlcmd1,
(obj.packingInChargeName,obj.isActive,obj.packingInChargeID))
    cursor.close()
    conn.close()

```

```

@staticmethod
def delete(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "DELETE FROM PackingInCharge WHERE packingInChargeID =
?"

```

```
cursor.execute(sqlcmd1, (unique_id))
cursor.close()
conn.close()
```

ROLE MODEL:

```
from Constants import connString
import pyodbc
import datetime
import uuid
import time
```

```
class RoleModel:
    def __init__(self, roleID = 0, roleName = "", canRole = False, canUsers =
False, canCustomer = False, canPackingInCharge = False, canProduct =
False, canProductionInCharge = False, canSupplier = False, canTransactionDetails
= False):
        self.roleID = roleID
        self.roleName = roleName
        self.canRole = canRole
        self.canUsers = canUsers
        self.canCustomer = canCustomer
        self.canPackingInCharge = canPackingInCharge
        self.canProduct = canProduct
        self.canProductionInCharge = canProductionInCharge
        self.canSupplier = canSupplier
        self.canTransactionDetails = canTransactionDetails
```

```
@staticmethod
def get_all():
```

```

conn = pyodbc.connect(connString, autocommit=True)
cursor = conn.cursor()
sqlcmd1 = "SELECT * FROM Role ORDER BY roleName"
cursor.execute(sqlcmd1)
records = []
for dbrow in cursor.fetchall():
    row = RoleModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],dbrow[6],dbrow[7],dbrow[8],dbrow[9])
    records.append(row)
cursor.close()
conn.close()
return records

```

@staticmethod

def get_name_id():

```

conn = pyodbc.connect(connString, autocommit=True)
cursor = conn.cursor()
sqlcmd1 = "SELECT roleID, roleName FROM Role ORDER BY roleName"
cursor.execute(sqlcmd1)
records = []
for dbrow in cursor.fetchall():
    row = RoleModel(dbrow[0],dbrow[1])
    records.append(row)
cursor.close()
conn.close()
return records

```

@staticmethod

def get_by_id(unique_id):

```

conn = pyodbc.connect(connString, autocommit=True)
cursor = conn.cursor()
sqlcmd1 = "SELECT * FROM Role WHERE roleID = ?"
cursor.execute(sqlcmd1, unique_id)

```

```

        record = None
        for dbrow in cursor.fetchall():
            record = RoleModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],dbrow[6],dbrow[7],dbrow[8],dbrow[9])
        cursor.close()
        conn.close()
        return record

```

```

@staticmethod
def insert(obj):
    obj.roleID = str(uuid.uuid4())
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "INSERT INTO Role"
    (roleName,canRole,canUsers,canCustomer,canPackingInCharge,canProduct,canProductionInCharge,canSupplier,canTransactionDetails)
    VALUES(?,?,?,?,?,?,?,?)"
    cursor.execute(sqlcmd1,
    (obj.roleName,obj.canRole,obj.canUsers,obj.canCustomer,obj.canPackingInCharge,obj.canProduct,obj.canProductionInCharge,obj.canSupplier,obj.canTransactionDetails))
    cursor.close()
    conn.close()

```

```

@staticmethod
def update(obj):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "UPDATE Role SET roleName = ?,canRole = ?,canUsers = ?,canCustomer = ?,canPackingInCharge = ?,canProduct = ?,canProductionInCharge = ?,canSupplier = ?,canTransactionDetails = ? WHERE roleID = ?"

```

```

        cursor.execute(sqlcmd1,
(obj.roleName,obj.canRole,obj.canUsers,obj.canCustomer,obj.canPackingInCharge,obj.canProduct,obj.canProductionInCharge,obj.canSupplier,obj.canTransactionDetails,obj.roleID))
        cursor.close()
        conn.close()

```

@staticmethod

```

def delete(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "DELETE FROM Role WHERE roleID = ?"
    cursor.execute(sqlcmd1, (unique_id))
    cursor.close()
    conn.close()

```

SUPPLIER MODEL:

```

from Constants import connString
import pyodbc
import datetime
import uuid
import time

```

class SupplierModel:

```

    def __init__(self, supplierID = "",supplierName = "",contactNbr = "",emailID =
"",address = "",city = "",county = "",postcode = "",country = "",experience = 0,emailModel
= None):
        self.supplierID = supplierID
        self.supplierName = supplierName
        self.contactNbr = contactNbr
        self.emailID = emailID
        self.address = address
        self.city = city

```

```

self.county = county
self.postcode = postcode
self.country = country
self.experience = experience
self.emailModel = emailModel

```

@staticmethod

def get_all():

```

    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM Supplier ORDER BY supplierName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = SupplierModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],dbrow[6],dbrow[7],dbrow[8],dbrow[9])
        records.append(row)
    cursor.close()
    conn.close()
    return records

```

@staticmethod

def get_name_id():

```

    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT supplierID, supplierName FROM Supplier ORDER BY supplierName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = SupplierModel(dbrow[0],dbrow[1])

```

```

        records.append(row)
    cursor.close()
    conn.close()
    return records

```

```

@staticmethod
def get_by_id(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM Supplier WHERE supplierID = ?"
    cursor.execute(sqlcmd1, unique_id)
    record = None
    for dbrow in cursor.fetchall():
        record = SupplierModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],dbrow[6],dbrow[7],dbrow[8],dbrow[9])
    cursor.close()
    conn.close()
    return record

```

```

@staticmethod
def insert(obj):
    obj.supplierID = str(uuid.uuid4())
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "INSERT INTO Supplier"
    (supplierID,supplierName,contactNbr,emailID,address,city,county,postcode,country,experience) VALUES(?,?,?,?,?,?,?,?,?,?)"
    cursor.execute(sqlcmd1,
    (obj.supplierID,obj.supplierName,obj.contactNbr,obj.emailID,obj.address,obj.city,obj.county,obj.postcode,obj.country,obj.experience))
    cursor.close()
    conn.close()

```

```

@staticmethod
def update(obj):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "UPDATE Supplier SET supplierName = ?,contactNbr = ?,emailID
= ?,address = ?,city = ?,county = ?,postcode = ?,country = ?,experience = ?
WHERE supplierID = ?"
    cursor.execute(sqlcmd1,
(obj.supplierName,obj.contactNbr,obj.emailID,obj.address,obj.city,obj.county,obj.
postcode,obj.country,obj.experience,obj.supplierID))
    cursor.close()
    conn.close()

```

```

@staticmethod
def delete(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "DELETE FROM Supplier WHERE supplierID = ?"
    cursor.execute(sqlcmd1, (unique_id))
    cursor.close()
    conn.close()

```

TRANSACTION DETAILS MODEL:

```

from Constants import connString
import pyodbc
import datetime
import uuid
import time
from Constants import contract_address
from web3 import Web3, HTTPProvider
import json
import pprint

```

```

class TransactionDetailsModel:
    def __init__(self, salesOrderID = "",gstNbr = "",customerID = "",supplierID =
    "",effDate = None,productID = "",price = 0,qty = 0,isBlockChainGenerated =
    False,hash = "",prevHash = "",sequenceNumber = 0,customerModel =
    None,supplierModel = None,productModel = None):
        self.salesOrderID = salesOrderID
        self.gstNbr = gstNbr
        self.customerID = customerID
        self.supplierID = supplierID
        self.effDate = effDate
        self.productID = productID
        self.price = price
        self.qty = qty
        self.isBlockChainGenerated = isBlockChainGenerated
        self.hash = hash
        self.prevHash = prevHash
        self.sequenceNumber = sequenceNumber
        self.customerModel = customerModel
        self.supplierModel = supplierModel
        self.productModel = productModel

```

@staticmethod

```

def get_all():
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM TransactionDetails ORDER BY customerID"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row =
TransactionDetailsModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],

```

```
dbrow[6],dbrow[7],dbrow[8],dbrow[9],dbrow[10],dbrow[11])
```

```
    records.append(row)
```

```
    cursor.close()
```

```
    conn.close()
```

```
    return records
```

```
@staticmethod
```

```
def get_name_id():
```

```
    conn = pyodbc.connect(connString, autocommit=True)
```

```
    cursor = conn.cursor()
```

```
    sqlcmd1 = "SELECT salesOrderID, customerID FROM TransactionDetails  
ORDER BY customerID"
```

```
    cursor.execute(sqlcmd1)
```

```
    records = []
```

```
    for dbrow in cursor.fetchall():
```

```
        row = TransactionDetailsModel(dbrow[0],dbrow[1])
```

```
        records.append(row)
```

```
    cursor.close()
```

```
    conn.close()
```

```
    return records
```

```
@staticmethod
```

```
def get_by_id(unique_id):
```

```
    conn = pyodbc.connect(connString, autocommit=True)
```

```
    cursor = conn.cursor()
```

```
    sqlcmd1 = "SELECT * FROM TransactionDetails WHERE salesOrderID = ?"
```

```
    cursor.execute(sqlcmd1, unique_id)
```

```
    record = None
```

```
    for dbrow in cursor.fetchall():
```

```
        record = TransactionDetailsModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],
```

```
dbrow[6],dbrow[7],dbrow[8],dbrow[9],dbrow[10],dbrow[11])
```

```
    cursor.close()
```

```
    conn.close()
```

```

return record

@staticmethod
def insert(obj):
    obj.salesOrderID = str(uuid.uuid4())
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "INSERT INTO TransactionDetails
(salesOrderID,gstNbr,customerID,supplierID,effDate,productID,price,qty,isBlockC
hainGenerated,hash,prevHash) VALUES(?,?,?,?,?,?,?,?,?,?)"
    cursor.execute(sqlcmd1,
(obj.salesOrderID,obj.gstNbr,obj.customerID,obj.supplierID,datetime.datetime.str
ptime(obj.effDate.replace('T', ' '), '%Y-%m-
%d'),obj.productID,obj.price,obj.qty,obj.isBlockChainGenerated,obj.hash,obj.prev
Hash))
    cursor.close()
    conn.close()

w3 = Web3(HTTPProvider('http://localhost:7545'))

compiled_contract_path = '../..../FoodSupplyChain-
Truffle/build/contracts/TransactionDetailsContract.json'
deployed_contract_address = contract_address

with open(compiled_contract_path) as file:
    contract_json = json.load(file)
    contract_abi = contract_json["abi"]

contract = w3.eth.contract(address=deployed_contract_address,
abi=contract_abi)

accounts = w3.eth.accounts

```

```

tx_hash      =      contract.functions.perform_transactions(obj.customerID,
obj.productID, int(obj.price), int(obj.qty)).transact({'from': accounts[0]})
receipt = w3.eth.wait_for_transaction_receipt(tx_hash)

```

```

@staticmethod
def update(obj):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "UPDATE TransactionDetails SET gstNbr = ?,customerID =
?,supplierID = ?,effDate = ?,productID = ?,price = ?,qty = ?,isBlockChainGenerated
= ?,hash = ?,prevHash = ? WHERE salesOrderID = ?"
    cursor.execute(sqlcmd1,
(obj.gstNbr,obj.customerID,obj.supplierID,datetime.datetime.strptime(obj.effDate.r
eplace('T',
        ' '),
        '%Y-%m-
%d'),obj.productID,obj.price,obj.qty,obj.isBlockChainGenerated,obj.hash,obj.prev
Hash,obj.salesOrderID))
    cursor.close()
    conn.close()

```

```

@staticmethod
def delete(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "DELETE FROM TransactionDetails WHERE salesOrderID = ?"
    cursor.execute(sqlcmd1, (unique_id))
    cursor.close()
    conn.close()

```

USERS MODEL:

```
from Constants import connString
import pyodbc
import datetime
import uuid
import time
```

```
class UsersModel:
```

```
    def __init__(self, userID = 0,userName = "",emailid = "",password = "",contactNo =
    "",isActive = False,roleID = 0,roleModel = None):
```

```
        self.userID = userID
        self.userName = userName
        self.emailid = emailid
        self.password = password
        self.contactNo = contactNo
        self.isActive = isActive
        self.roleID = roleID
        self.roleModel = roleModel
```

```
@staticmethod
```

```
def get_all():
```

```
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM Users ORDER BY userName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
```

```
        row =
```

```
UsersModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],dbrow[6])
        records.append(row)
```

```

cursor.close()
conn.close()
return records

```

@staticmethod

```

def get_name_id():
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT userID, userName FROM Users ORDER BY userName"
    cursor.execute(sqlcmd1)
    records = []
    for dbrow in cursor.fetchall():
        row = UsersModel(dbrow[0],dbrow[1])
        records.append(row)
    cursor.close()
    conn.close()
    return records

```

@staticmethod

```

def get_by_id(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "SELECT * FROM Users WHERE userID = ?"
    cursor.execute(sqlcmd1, unique_id)
    record = None
    for dbrow in cursor.fetchall():
        record =
UsersModel(dbrow[0],dbrow[1],dbrow[2],dbrow[3],dbrow[4],dbrow[5],dbrow[6])
    cursor.close()
    conn.close()
    return record

```

@staticmethod

```

def insert(obj):

```

```

obj.userID = str(uuid.uuid4())
conn = pyodbc.connect(connString, autocommit=True)
cursor = conn.cursor()
sqlcmd1 = "INSERT INTO Users
(userName,emailid,password,contactNo,isActive,roleID) VALUES(?,?,?,?,?,?)"
cursor.execute(sqlcmd1,
(obj.userName,obj.emailid,obj.password,obj.contactNo,obj.isActive,obj.roleID))
cursor.close()
conn.close()

```

```

@staticmethod
def update(obj):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "UPDATE Users SET userName = ?,emailid = ?,password =
?,contactNo = ?,isActive = ?,roleID = ? WHERE userID = ?"
    cursor.execute(sqlcmd1,
(obj.userName,obj.emailid,obj.password,obj.contactNo,obj.isActive,obj.roleID,obj.
userID))
    cursor.close()
    conn.close()

```

```

@staticmethod
def delete(unique_id):
    conn = pyodbc.connect(connString, autocommit=True)
    cursor = conn.cursor()
    sqlcmd1 = "DELETE FROM Users WHERE userID = ?"
    cursor.execute(sqlcmd1, (unique_id))
    cursor.close()
    conn.close()

```

SCREENSHOTS:

```
80
81 @app.route("/")
82 def index():
83     global user_id, emailid
84     return render_template("Login.html")
85
86 @app.route("/processLogin", methods=["POST"])
87 def processLogin():
88     global user_id, emailid, role_object
89     emailid = request.form["emailid"]
90     password = request.form["password"]
91     conn1 = pyodbc.connect(connString, autocommit=True)
92     cur1 = conn1.cursor()
93     sqlcmd1 = "SELECT * FROM Users WHERE emailid = '" + emailid + "' AND password = '" + password + "' AND isActive = 1"
94     cur1.execute(sqlcmd1)
95     row = cur1.fetchone()
96
97     cur1.commit()
98     if not row:
99         return render_template(template_name_or_list="Login.html", processResult="Invalid Credentials")
100     user_id = row[0]
101
102     cur2 = conn1.cursor()
103     sqlcmd2 = "SELECT * FROM Role WHERE RoleID = '" + str(row[6]) + "'"
104     cur2.execute(sqlcmd2)
105     row2 = cur2.fetchone()
106
107     if not row2:
108         return render_template(template_name_or_list="Login.html", processResult="Invalid Role")
109
110 Users_operation() if not canUsers
```

Server of the Application

```
429
430 if canProductionInCharge == False:
431     message = "You Don't Have Permission to Access ProductionInCharge"
432     msgType = "Error"
433     return redirect(url_for("Information"))
434
435 records = ProductionInChargeModel.get_all()
436
437 return render_template(template_name_or_list="ProductionInChargeListing.html", records=records)
438
439 @app.route("/ProductionInChargeOperation")
440 def ProductionInCharge_operation():
441     global user_id, user_name, message, msgType, role_object
442     if role_object is None:
443         message = "Application Error Occurred. Logout"
444         msgType = "Error"
445         return redirect(url_for("Information"))
446     canProductionInCharge = process_role(3)
447
448     if not canProductionInCharge:
449         message = "You Don't Have Permission to Access ProductionInCharge"
450         msgType = "Error"
451         return redirect(url_for("Information"))
452     unique_id = ""
453     operation = request.args.get("operation")
454     row = ProductionInChargeModel(productionInChargeID="", productionInChargeName="")
455     ProductionInCharge = ProductionInChargeModel.get_all()
456
457 if operation != "Create":
```

Server of the Application (2)

PRESENTATION CERTIFICATE:



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
CATEGORY-1 UNIVERSITY BY UGC
Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE



www.sathyabama.ac.in

SCHOOL OF COMPUTING

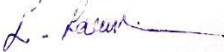
Department of Computer Science and Engineering

Certificate of Presentation

This is to certify that
DEEPAK ARUMUGAM S
of
Sathyabama Institute of science and Technology.
has presented a paper entitled
Blockchain-based decentralized e-marketplace - A smart contract-driven Framework.
in the
International Conference on Cognitive Computing and Artificial Intelligence (ICCCAI-2024) in association with
Taylor's University, Malaysia held on 7th & 8th March 2024.



Dr. Sayan Kumar Ray
Head, School of Computer Science
Taylor's University



Dr. L. Lakshmanan
HOD, CSE Dept
Sathyabama Institute of Science and Technology



Dr. T. Sasikala
Dean, SOC
Sathyabama Institute of Science and Technology

Blockchain-based decentralized e-marketplace

A Smart Contract-Driven Framework

Deepak Arumugam S¹
Computer Science and Engineering
Sathyabama Institute of Science and
Technology
Chennai, India
deepakarumugam03@gmail.com

Ms. Nancy Kirupanithi D³
Computer Science and Engineering
Sathyabama Institute of Science and
Technology
Chennai, India
nancykirupanithi@sathyabama.ac.in

Jenish J. Bosco²
Computer Science and Engineering
Sathyabama Institute of Science and
Technology
Chennai, India
jenishbosco@gmail.com

Abstract— e-Marketplaces are undergoing a transformation into sophisticated and intricate networks, and are emerging as a significant source of potential advantages in the contemporary world. Simultaneously, purchasers are currently displaying a greater inclination towards product excellence. Nevertheless, it is difficult to monitor the origin of data and uphold its ability to be traced across the eMarketplace network. Traditional marketplaces are centralized and rely on a third party for transactions. These centralized systems exhibit a deficiency in transparency, accountability, and auditability. This paper presents a decentralized blockchain-based solution for automating e-marketplace processes, including buying and selling products and facilitating sharing of data among all stakeholders involved in their management. The proposed solution ensures complete security, transparency, traceability, and trustworthiness. We combine the Ethereum blockchain with decentralised storage to securely retrieve, store, and distribute the data associated with the e-marketplace and its management. The intelligent contract code was created and verified in the Ganache environment. We provide a cost analysis of the expenses borne by the stakeholders involved in the supply chain. Implementing a blockchain-based solution for eMarketplace operations in supply chains is financially feasible and offers an efficient, secure, reliable, and open method of communication between different participants.

Keywords— *Ethereum e-Marketplace,*

I. A GENERAL SYSTEM

Blockchain is a comprehensive framework that influences the e-marketplace through technological, legal, organisational, quality, and consumer aspects. The advancement of technology presents novel prospects, but also requires a thorough assessment of existing business procedures, including those that handle confidential customer information or the configuration of communication pathways within the supply chain. E-marketplaces employ various strategies to enhance network effects, such as customising service offerings, implementing recommendation systems for products and services, establishing trust mechanisms, and streamlining transactions.

E-marketplaces[6] flourish as a result of the distributed effects they enable when buyers and sellers engage in trade with each other. Matching can be facilitated either through a promotion driven by the seller or the marketplace, or through a search initiated by the customer. Buyers gain an advantage as the marketplace decreases the expenses associated with searching for products or services. Sellers gain advantages from the marketplace due to the absence of additional expenses for product listings and the sharing of storage and transportation expenses. E-marketplaces

enable global access for buyers and sellers, allowing transactions to occur at any time.

E-marketplaces[6] decrease the expenses associated with searching for buyers by effectively cataloguing and collecting products and services in databases. Marketplaces not only decrease costs, but also boost revenue by influencing users' purchasing preferences through tactics such as product suggestions, packaging, and other manipulations related to products or services.

In decentralised e-marketplaces, the process of matching buyers and sellers can be conducted with greater transparency. This allows buyers to have a wider range of choices to pick from, thereby increasing the probability of finding a suitable match. Furthermore, the payments could be transferred directly from the buyer to the seller, bypassing any intermediaries. Why are there no peer-to-peer e-marketplaces specifically designed for trading computational power? A significant factor is the absence of trust. The prominent service providers have cultivated a commendable standing, and there are also established legal frameworks in existence to safeguard customers. Smaller entities face significant obstacles when trying to enter this market, primarily due to formidable barriers such as the need to establish adequate infrastructure and build a reputable image.

II. ETHEREUM BLOCKCHAIN

The technology behind Blockchain[5][6][10] is a decentralised and distributed system that enables easy data sharing. Each member of the network possesses an electronic version of all legitimate transactions in the ledger, ensuring transparency and accessibility. Transactions are incorporated into the system as blocks, with each block containing multiple transactions. Miners add these blocks to the global blockchain.

Ethereum[9][8][1] functions as a decentralised state machine. Ethereum records the changes in the state of a versatile data repository, specifically a repository capable of storing any data represented as a combination of a key and a value. A key-value data store stores arbitrary values, each of which is referenced by a specific key. Ethereum possesses a memory that retains both information and code, and it utilises the blockchain of Ethereum to monitor

the alterations in this memory over time. Ethereum, similar to a versatile stored-program computer, has the ability to load code into its state machine and execute that code, while recording the resulting changes in its blockchain. Two key distinctions of Ethereum[1][6] from typical general-purpose computers are that state changes in Ethereum are regulated by consensus rules and the state is globally distributed.

A. About Smart Contracts

Smart contracts[2][3] play a crucial role within a blockchain framework. By utilizing them, individuals can engage in online transactions without the requirement of an intermediary. They are not subject to control by either central authorities or human intervention.

Smart contracts are automated agreements in which the contractual terms are encoded directly into lines of code, allowing for self-execution.

Smart contracts enable secure transactions and agreements between different, unidentified parties without relying on central legitimacy, system of law, or external means of enforcement. They make transactions easily trackable, clear, and permanent.

Smart contracts[1][11] are essentially computer programs. In this context, the term "contract" lacks any legal significance. Once a smart contract is deployed, its code becomes immutable. Contrary to traditional software, the sole method of altering a smart contract is by implementing a fresh instance. The result of executing a smart contract is identical for all individuals who execute it, based on the transaction context that triggered its execution and the current state of the Ethereum blockchain at the time of execution. Smart contracts function within a highly restricted execution [4]environment. They have the ability to retrieve information about their own state, the historical context of the operation that invoked them, and certain details regarding the most current blocks.

B. Smart Contracts Life Cycle

Smart contracts are commonly composed in a high-level programming language, especially Solidity. However, in order to execute, these programmes need to

be transformed into low-level bytecode that is compatible with the EVM[6].

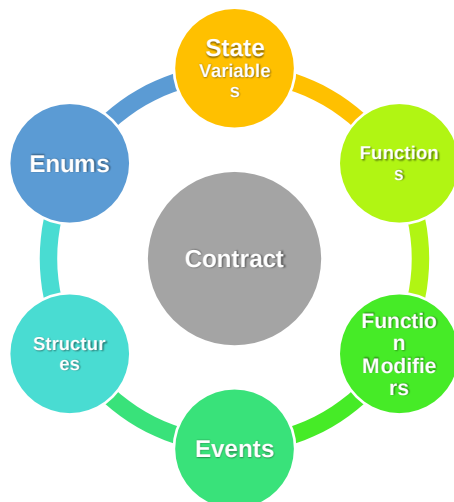
After compilation, the programmes are deployed on the Ethereum network through a specific contract creation transaction. This transaction is distinguished by being sent to the designated contract creation address, which is 0x0.

Every contract is identified only by an Ethereum address, which is determined based on the contract creation transaction and the originating account's nonce. An Ethereum contract's[3] address can be utilised in a transaction to either send payments to the contract or invoke one of its functions. It is important to understand that, in contrast to externally owned accounts (EOAs)[4], an account that is established for an original smart contract does not have any associated keys.

As the individual responsible for creating the contract, you do not possess any exceptional permissions at the protocol level, although you have the ability to explicitly incorporate these into the smart contract[8]. The user cannot obtain the private key for the agreement's account, as it does not exist. It can be stated that smart agreement accounts are self-owned.

Crucially, contracts are only executed[9] when they are invoked by a transaction. Every smart contract in Ethereum is ultimately executed through an activity initiated from an Externally Owned Account (EOA).

Every smart contract file contains six important building blocks:



A contract can invoke another contract, which in turn can invoke other agreement, and so forth. However, it is important to note that the initial contract in this sequence of processing will constantly have been invoked by a transaction from an Externally Owned Account (EOA). Contracts[7] do not operate autonomously or passively. Contracts remain inactive until a transaction activates them, in either the direct or indirect manner as part of a series of contract calls. It is important to mention that smart contracts are not executed concurrently in any way. The Ethereum virtual machine[10] can be regarded as a machine that operates on a single thread.

Transactions are indivisible, regardless of the number of contracts they invoke or the actions performed by those contracts when invoked. Transactions are executed completely, and any modifications to the global state (such as contracts and accounts) are only recorded if the entire execution is successful[2]. Successful termination refers to the condition where the programme runs without encountering any errors and completes its execution. In the event of a failure during execution caused by an error, all modifications to the system's state are reversed, effectively nullifying the transaction[7]. A failed transaction is logged as an attempted transaction, and the ether used for gas during the execution is subtracted from the original account. However, it does not have any other impact on the state of the contract or account.

III. DETAILS OF PROPOSED OPERATIONS

The primary contributions of this study are outlined as follows:

Our proposal entails utilizing the Ethereum blockchain to create a solution that records the essential interactions among trading partners in a supply chain. This will be achieved through the implementation of a solidity smart contract written on Ethereum and a decentralized storage system.

We introduce and analyze the primary smart contract code and its mathematical algorithms that establish the fundamental principles of the planned blockchain approach. In addition, we offer a comprehensive sequence

diagram to elucidate the blockchain-based eMarketplace solution[6].

It serves as a superior solution to the traditional marketplace application providing robust and auditable solutions. This document outlines the algorithms used in smart contracts and assesses their susceptibility to

exploitation, as well as the associated gas and Ether costs when executed on the Ethereum network[5][9].

We suggest a versatile prototype solution that can be applied to public Ethereum blockchains with minimal adjustments, depending on the specific needs and preferences of the existing eMarketplace[6].

EMarketPlace-Truffle E:\Athene\1WiseniTSolutions\03.WebSites\CodeGenerator\CodeGeneratorV1\EMarketPlace\EMarketPlace-Truffle

NAME BuyerContract	ADDRESS 0x24Cd87A49C9171dC3894Bcdd064655e5059cAf9a	TX COUNT 0	DEPLOYED
NAME Migrations	ADDRESS 0x4a654e58c23D1fda0B519469fF4aeaAE0d509FcC	TX COUNT 6	DEPLOYED
NAME ProductContract	ADDRESS 0x58A38338549f9d1b15744551deaE84A61A122344	TX COUNT 0	DEPLOYED
NAME PurchaseTransactionContract	ADDRESS 0x0d555301F90205D82ee801DC3E26C34760158400	TX COUNT 0	DEPLOYED
NAME RoleContract	ADDRESS 0x64512B5576C8fE19d728EB928c5abD131c310a98	TX COUNT 0	DEPLOYED
NAME SellerContract	ADDRESS 0x54a4D78e3e3FAc67AF54BCf26f6829839337eF45	TX COUNT 0	DEPLOYED
NAME UsersContract	ADDRESS 0xD4218Ba632c127A8a060b5C70deBb6C4cB9eC55E	TX COUNT 0	DEPLOYED

A. System Entities and Components

- **Seller:** The primary responsibility of a seller is to record and maintain information regarding their goods and deliveries on the blockchain. The crucial information regarding goods and materials encompasses the product's name, type, cost, manufacturing date, design, composition, and the certifying agency. A seller typically conducts market research to identify the current demand and specifications to effectively sell their products. It ensures that all essential documents, such as specifications and certification records, for all products are accessible. Additionally, it is imperative that the seller is authentic and provides full transparency regarding the valid licence, making all relevant information publicly accessible.
- **Buyer:** A buyer is a consumer who purchases and uses products from retailers. Prior to purchasing products, a Buyer assesses the reliability of a seller by utilising a reputation system.
- **Smart contracts[3][6][8]11]** serve as a replacement for intermediaries and ensure the enforcement of agreed-upon rules by the entities involved. This paper presents the implementation of four primary smart contracts: the buyer smart contract, the seller smart contract, the product smart contract, and the purchase transaction smart contract. The smart contracts in question encompass a collection of operations that are invoked upon the occurrence of specific events within the eMarketplace. These contracts necessitate the inclusion of the addresses of all preceding contracts to uphold a continuous sequence of transactions. In order to obtain their

respective addresses, all four contracts are deployed for this specific purpose. Thus, the solution facilitates the end buyer in attaining comprehensive data provenance maintenance.

B. Application Binary Interface (ABI)

An application binary interface (ABI) is a software interface that facilitates communication between two program modules, typically between the OS and user programs. An Application Binary Interface (ABI) specifies the rules and conventions for accessing data and its structures and functions in machine code.

The ABI (Application Binary Interface) is utilized in Ethereum to create smart agreement calls for the EVM (Ethereum Virtual Machine) and to extract data from transactions. An ABI serves the purpose of specifying the functions within a contract that can be called and detailing the way each function will receive input and produce output.

The contract's ABI[4][5][6] is defined as a JSON array consisting of function descriptions. A function description is a JSON object that includes the following fields: type, name, inputs, outputs, constant, and payable. A description object for an event contains the following fields: type, name, inputs, and anonymous.

C. Gas Usage

On the Ethereum platform[2][10][4], the execution of every smart contract and transaction incurs costs in the form of Gas, which represents monetary value. The gas usage by a smart contract is determined by the number of computational operations and the amount of storage space it needs. Hence, the proficient implementation of a smart contract necessitates enough available Gas, as the execution of the contract will be constrained by the limitations of Gas. In addition, the cost of Gas can limit the number of computation steps performed by a smart contract, thus serving as a safeguard against DoS attacks. Gas is a limiting factor that determines the maximum computational capacity that Ethereum permits for a transaction. Users are discouraged from invoking functions with a high petrol cost since they are responsible for paying the petrol fees associated with the transaction. Therefore, it is advantageous for the programmer to reduce

the price of gas because of a contract's functions. To minimize the price of gas of an operation call, it is advisable to follow certain practices when creating smart contracts.

IV. CONCLUSIONS

This paper explores the advantages of implementing blockchain technology in eMarketplace operations. Implementing a blockchain-based eMarketplace system enhances productivity and performance by facilitating communication among trading partners and promoting transparency. It establishes a robust value chain collaboration that enhances trust among stakeholders. Furthermore, it enhances the precision of order fulfilment by preserving data integrity, which has the potential to boost profits for all stakeholders involved in the eMarketplace. The integration of blockchain technology into eMarketplaces provides a financially advantageous and effective method for implementing strategies across different eMarketplaces. This illustrates the viability of the proposed solution, as each stakeholder contributes a fraction of the cost compared to traditional methods that require payment to third-party service providers.

Our future objective is to develop and construct decentralised applications that will entirely streamline the eMarketplace procedure for all parties involved. Our objective is to enhance the smart contract capabilities to cater to the diverse requirements of the eMarketplace sector.

V. REFERENCES

- [1] D. K. C. Lee, Ed., *Handbook of Digital Currency: Bitcoin, Innovation, Financial Instruments, and Big Data*. New York, NY, USA: Academic, 2015.
- [2] A. Seitz, D. Henze, D. Miehle, B. Bruegge, J. Nickles, and M. Sauer, "Fog computing as enabler for blockchain-based IIoT app marketplaces—A case study," in *Proc. 5th Int. Conf. Internet Things: Syst., Manage. Secur.*, Oct. 2018, pp. 182–188, doi: 10.1109/IoTSMS.2018.8554484.
- [3] J.-S. Park, T.-Y. Youn, H.-B. Kim, K.-H. Rhee, and S.-U. Shin, "Smart contract-based review system for an IoT data marketplace," *Sensors*, vol. 18, no. 10, p. 3577, Oct. 2018, doi: 10.3390/s18103577.
- [4] B.-G. Jeong, T.-Y. Youn, N.-S. Jho, and S. U. Shin, "Blockchain-based data sharing and trading model for the connected car," *Sensors*, vol. 20, no. 11, p. 3141, Jun. 2020, doi: 10.3390/s20113141.

- [5] H. Chen, Z. Chen, F. Lin, and P. Zhuang, "Effective management for blockchain-based agri-food supply chains using deep reinforcement learning," *IEEE Access*, vol. 9, pp. 36008–36018, 2021, doi: 10.1109/ACCESS.2021.3062410.
- [6] Subramanian H. Decentralized blockchain-based electronic marketplaces. *Comm. of the ACM*. 2018;61(1):78–84.
- [7] S. Saberi, M. Kouhizadeh, J. Sarkis, and L. Shen, ""Blockchain technology and its relationships to sustainable supply chain management," *Int. J. Prod. Res.*, vol. 57, no. 7, pp. 2117-2135, Apr. 2019.
- [8] S. A. Abeyratne and R. P. Monfared, ""Blockchain ready manufacturing supply chain using distributed ledger," *Int. J. Res. Eng. Technol.*, vol. 5, no. 9, pp. 1-10, Sep. 2016.
- [9] P. Praitheeshan, L. Pan, J. Yu, J. Liu, and R. Doss, ""Security analysis methods on Ethereum smart contract vulnerabilities: A survey," 2019, arXiv:1908.08605. [Online]. Available: <http://arxiv.org/abs/1908.08605>
- [10] K. Baird, S. Jeong, Y. Kim, B. Burgstaller, and B. Scholz, ""The economics of smart contracts," 2019, arXiv:1910.11143. [Online]. Available: <http://arxiv.org/abs/1910.11143>.
- [11] N. Nizamuddin, K. Salah, M. A. Azad, J. Arshad, and M. H. Rehman, ""Decentralized document version control using ethereum blockchain and IPFS," *Comput. Electr. Eng.*, vol. 76, pp. 183-197, Jun. 2019.