

SUPPLY CHAIN MANAGEMENT SYSTEM USING BLOCKCHAIN

*A Project Report submitted
in partial fulfillment of the requirements
for the award of the degree of*

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

By

1. P. Sanjana(20B01A05F6)

2. R. Sanjana(20B01A05F7)

3. U. Jahnavi(20B01A05H2)

4. V. Sylvia Anand(20B01A05I4)

5. Y. Mahalakshmi(20B01A05J2)

Under the esteemed guidance of
Mr. M. V. V. Rama Rao (Assistant Professor)



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

SHRI VISHNU ENGINEERING COLLEGE FOR WOMEN(A)

(Approved by AICTE, Accredited by NBA & NAAC, Affiliated to JNTU Kakinada)

BHIMAVARAM – 534 202

2023 – 2024

SHRI VISHNU ENGINEERING COLLEGE FOR WOMEN(A)

(Approved by AICTE, Accredited by NBA & NAAC, Affiliated to JNTU Kakinada)
BHIMAVARAM – 534 202

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



CERTIFICATE

*This is to certify that the project entitled "**SUPPLY CHAIN MANAGEMENT USING BLOCKCHAIN**", is being submitted by **P.Sanjana, R.Sanjana, U.Jahnavi, V.Sylvia Anand, Y.Mahalakshmi** bearing the **Regd. No. 20B01A05F6, 20B01A05F7, 20B01A05H2, 20B01A05I4, 20B05A05J2** in partial fulfillment of the requirements for the award of the degree of "**Bachelor of Technology in Computer Science & Engineering**" is a record of bonafide work carried out by her under my guidance and supervision during the academic year 2023–2024 and it has been found worthy of acceptance according to the requirements of the university.*

Internal Guide

Head of the Department

External Examiner

Acknowledgements

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant encouragement and guidance has been a source of inspiration throughout the course of this seminar. We take this opportunity to express our gratitude to all those who have helped us in this seminar.

We wish to place our deep sense of gratitude to **Sri. K. V. Vishnu Raju**, Chairman of SVES, for his constant support on each and every progressive work of mine.

We wish to express our sincere thanks to **Dr. G. Srinivasa Rao**, Principal of SVECW for being a source of inspiration and constant encouragement.

We wish to express our sincere thanks to **Dr. P. Srinivasa Raju**, Vice-Principal of SVECW for being a source of inspirational and constant encouragement.

We wish to place our deep sense of gratitude to **Dr. P. Kiran Sree**, Head of the Department of Computer Science & Engineering for his valuable pieces of advice in completing this project successfully.

We are deeply thankful to our project coordinator **Dr. P. R. Sudha Rani**, Professor for her indispensable guidance and unwavering support throughout our project's completion. Her expertise and dedication have been invaluable to our success.

We are deeply indebted and sincere thanks to our PRC members **Dr. P. B. V. Raja Rao**, **Dr. J. Veeraraghavan** and **Dr. K. Ramachandra Rao** for their valuable advice in completing this project successfully.

Our deep sense of gratitude and sincere thanks to to **Mr. M. V. V. Rama Rao**, Assistant Professor for his unflinching devotion and valuable suggestions throughout my project work.

Project Associates:

1.20B01A05F6 - P. Sanjana

2.20B01A05F7 - R. Sanjana

3.20B01A05H2 - U. Jahnavi

4.20B01A05I4 - V. Sylvia Anand

5.20B01A05J2 - Y. Mahalakshmi

Abstract

The traditional supply chain management is a complex process that is done manually with insufficient data and without any security in transactions. It includes more time and ineffective processes which may cause customers trouble using this application. To store the details of the transactions and history of the products it will be difficult to enter the data manually and to maintain the records for years is arduous. Blockchain can address various challenges in traditional supply chain systems by providing a decentralized and secure ledger that records transactions and data in a tamper-resistant manner. Blockchain provides a transparent and immutable ledger that allows all participants in the supply chain to have a real-time view of transactions and data. Every transaction or event in the supply chain is recorded in a decentralized and distributed manner, reducing the risk of fraud and errors. Blockchain enables end-to-end traceability of products through the entire supply chain.

So, if we include blockchain in this traditional supply chain management then we can increase the efficiency and security of the process. The security in the transactions will be in addition. If we use other technology rather than blockchain, we may get laborious when hacking is involved, because it will be hard to find the hacker. But, with blockchain technology, as security is more, the hacker cannot access the data and no person outside of the organization can modify or insert the data. Digitizing physical assets and creating a decentralized, unchangeable record of all transactions are two ways that blockchain can help supply organizations manage assets more accurately and transparently from manufacturing to delivery or end-user use. We mainly focus on customers and sellers where the product details will be displayed i.e. when it is manufactured. Because of the details only the security and quality of the product will be known.

Keywords: Blockchain, Supply chain management, Ganache, Metamask, Solidity, Ethereum.

Contents

S.No	Topic	Page No.
1.	Introduction	1 - 3
2.	System Analysis	4 - 13
2.1	Existing System	
2.2	Proposed System	
2.3	Feasibility Study	
3.	System Requirements Specification	14 -25
3.1	Software Requirements	
3.2	Hardware Requirements	
3.3	Functional Requirements	
4.	System Design	25-41
4.1	Introduction	
4.2	UML Diagrams	
4.3	Database Design	
4.3.1	ER Diagrams	
4.3.2	Table Structures	
5.	System Implementation	42-56
5.1	Introduction	
5.2	Project Modules	
5.3	Screens	
6.	System Testing	56-65
6.1	Introduction	
6.2	Testing Methods	
6.3	Test Cases	

7.	Conclusion	66-88
8.	References	89-90

List of Figures

S.No	Figure Description	Page No
1.1	Working of Ethereum	2
2.1.1	Traditional Supply Chain Management	5
2.2.1	Architecture diagram	8
3.1.1	Ganache	15
3.1.2	Metamask	16
3.2.1	Flow of Blockchain	25
3.3.1	Block representation in detail	28
3.3.2	Code flow of supply chain management	30
3.3.3	Metamask Interface	31
3.3.4	Ganache Interface	32
3.3.5	Ganache interface(local blockchain)	33
3.3.6	Ganache blockchain creation	33
3.3.7	Ganache local blockchain	34
3.3.8	Truffle installation	35
3.3.9	Truffle installation output	36
3.3.10	npx truffle migrate	36
3.3.11	npx truffle compile	37
3.3.12	npm run start	38
3.3.13	Architecture of supply chain management	40
3.3.14	Truffle management process	45
3.3.15	Objectives of Blockchain	52
4.1.1	How Blockchain works	53
4.2.1	Data Flow Diagram	54
4.2.2	UseCase Diagram	56

4.2.3	Class Diagram	57
4.2.4	Sequence Diagram	58
4.2.6	Collaboration Diagram	60
4.2.7	Deployment Diagram	61
4.2.8	Activity Diagram	63
4.2.9	Component Diagram	65
4.3.1.1	ER Diagram	67
5.3.1	npx truffle migrate command output	74
5.3.2	npx truffle compile command output	74
5.3.3	npm run start command output	75
5.3.4	This is the Ganache interface which is the local blockchain where we have different ethereum addresses for actors included in the project.	75
5.3.5	This is the Home page of our project which consists of navigation bar with Register, Order Products, Control Supply Chain and Track Products Options.	76
5.3.6	This is the Registration page of the project where the actors need to be registered.	77
5.3.7	This is the Order Products Page where the products are ordered by using Product Name and Product Description.	77
5.3.8	This is the Control Supply chain Page where the product transferring happens.	79
5.3.9	This is the tracking page of the project where the ordered product is tracked by using its ID.	79
5.3.10	This is the final output screen where the product details along with actors is displayed.	80

1.Introduction

Supply chain management is the movement of goods or products between manufacturers and customers. This flow will follow the process from the core development to the overall development of the product or good [1]. It also includes transporting the product or goods between sources like suppliers, manufacturers, distributors, retailers, customers, etc. As the strategic framework that coordinates the smooth movement of commodities, information, and funds among a network of interrelated entities, supply chain management (SCM) is essential to the modern corporate environment [1]. In today's globalized and highly competitive markets, organizations recognize the critical importance of SCM in enhancing operational resilience, minimizing costs, and optimizing overall performance.

This advancement of the system using blockchain explores the fundamental ideas, new developments, and the revolutionary potential of digital technologies as it digs into the complex dynamics of supply chain management. Through examining the opportunities and problems associated with supply chain management, this research seeks to provide industry and academics with insightful knowledge that will deepen our comprehension of the always-changing nature of supply chain operations

Several connected processes make up the supply chain management process, which makes it easier for products and services to be moved from suppliers to consumers. The procedure entails: Identifying and choosing reliable suppliers based on factors such as quality, cost, and reliability. Establish and maintain strong relationships with suppliers to ensure a smooth flow of materials and information [2]. Negotiate contracts, terms, and conditions with suppliers. Place orders for raw materials or finished goods based on demand forecasts and inventory levels. Convert raw materials into finished goods through manufacturing processes. Optimize production efficiency and quality control measures. Monitor and manage inventory levels to avoid stockouts or excess inventory.

Implement just-in-time (JIT) or other inventory optimization strategies. Plan and execute the transportation of goods from manufacturing facilities to distribution centers. Optimize transportation routes and modes to reduce costs and enhance efficiency. Store and manage inventory in warehouses or distribution centers. Implement efficient warehouse layouts and technology for order fulfillment. Receive and process customer orders.

Communicate order details to the relevant departments for fulfillment. Coordinate the movement of goods from distribution centers to retailers or end customers. Utilize various transportation modes, including trucks, ships, planes, etc [2]. Manage inventory at retail locations. Ensure that products are appropriately displayed and available for customers. Pick, pack, and ship products to fulfill customer orders.

Provide tracking information and manage returns if necessary. Address customer inquiries, complaints, or issues promptly. Gather feedback to improve processes and enhance customer satisfaction. Handle product returns and manage the reverse flow of goods. Evaluate returned products for possible refurbishment or recycling.

Utilize technologies such as RFID, barcoding, and advanced software systems for real-time tracking and visibility. Implement data analytics for demand forecasting and continuous process improvement. Regularly assess and refine supply chain processes. Adapt to market changes and incorporate new technologies or strategies for increased efficiency.

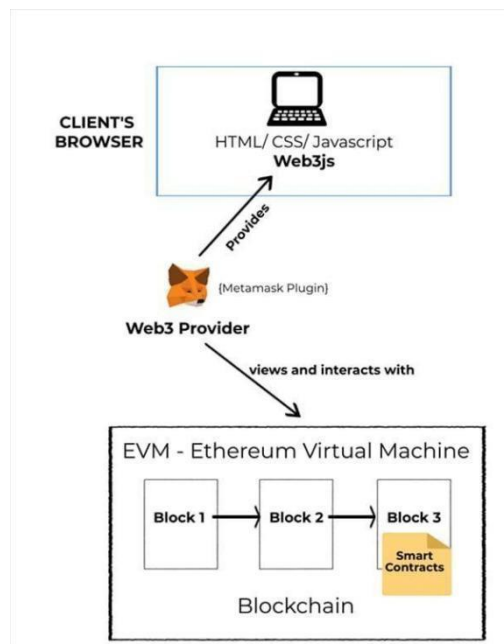


Fig. 1.1. Working of Ethereum

Blockchain's decentralized control is a paradigm shift from traditional supply chain models. Control is shared among all network members rather than depending on a single entity. This decentralization eliminates single points of failure, enhances resilience, and ensures that the supply chain remains operational even in the face of disruptions.

Improved traceability is another significant advantage of implementing blockchain in supply chain management. A unique identifier that is recorded on the blockchain can be issued to each product or batch, allowing precise tracing of its path from the point of origin to the point of destination [3]. This traceability is invaluable in identifying the source of issues such as recalls or counterfeit products, allowing for swift corrective actions. Increasing efficiency is a natural outcome of adopting blockchain in supply chain processes.

The streamlined and automated execution of smart contracts within the blockchain network reduces the need for intermediaries, minimizes paperwork, and accelerates transaction times. This efficiency translates to cost savings and quicker response to market demands. Blockchain can help identify and mitigate risks by providing a clear and accurate picture of the entire supply chain. With real-time data and traceability, companies can respond more effectively to disruptions, such as recalls or supply chain interruptions.

Integrating blockchain technology into supply chain management holds the promise of revolutionizing traditional processes, offering a transparent, traceable, and efficient framework [3]. By leveraging the decentralized and tamper-resistant nature of blockchain, supply chain participants can benefit from increased transparency, streamlined operations, and enhanced trust across the entire ecosystem.

The collaborative nature of blockchain technology fosters better communication and collaboration among supply chain participants, creating a more cohesive and responsive network. Ultimately, the integration of blockchain in supply chain management not only addresses current challenges but also sets the stage for a more resilient, agile, and trustworthy supply chain ecosystem.

As industries continue to explore and adopt blockchain solutions, the potential benefits of enhanced transparency, traceability, and efficiency are poised to redefine the future landscape of supply chain management. One significant financial benefit of using blockchain technology in supply chain management is reduced expenses. Blockchain helps to save costs at every stage of the supply chain lifecycle by eliminating the need for middlemen, lowering errors, and improving overall efficiency. System analysis in supply chain management using blockchain involves a comprehensive evaluation of existing processes and the integration of blockchain technology to optimize efficiency and transparency. The analysis encompasses identifying key pain points within the supply chain, assessing data flow, and evaluating the potential benefits of blockchain in addressing specific challenges.

2. System Analysis

System analysis in supply chain management using blockchain involves a comprehensive evaluation of existing processes and the integration of blockchain technology to optimize efficiency and transparency. The analysis encompasses identifying key pain points within the supply chain, assessing data flow, and evaluating the potential benefits of blockchain in addressing specific challenges.

2.1. Existing System:

Traditional supply chain management involves a series of interconnected activities aimed at the production and delivery of goods and services from manufacturers to end consumers. The process typically begins with the procurement of raw materials, followed by manufacturing, distribution, and retail, and ultimately concludes with the delivery of products to customers. In the conventional model, information flow and coordination between various stakeholders rely heavily on manual processes, often resulting in inefficiencies and challenges.

The procurement stage initiates the supply chain, where organizations source raw materials required for production. This phase involves negotiations with suppliers, order placement, and logistics planning. The manufacturing phase follows, where raw materials are transformed into finished goods. During this stage, production schedules, quality control, and inventory management play crucial roles in ensuring smooth operations.

Once products are manufactured, they move into the distribution phase, involving warehousing, packaging, and transportation [4]. Traditional supply chains often face challenges related to inventory management, as organizations strive to balance the costs of carrying excess inventory against the risk of stockouts. Additionally, the reliance on manual tracking systems can lead to inaccuracies and delays in identifying product locations within the supply chain.

The retail stage involves the sale of products to consumers through various channels such as brick-and-mortar stores or online platforms. Marketing strategies, demand forecasting, and customer relationship management are vital components at this stage. In a traditional supply chain, forecasting accuracy may be limited due to the lack of real-time data, leading to overstocking or under stocking issues.

The final stage of the traditional supply chain is the delivery of products to end consumers. Logistics and transportation become critical factors in ensuring timely and cost-effective delivery. Challenges arise in optimizing routes, minimizing transit times, and managing the complexities of last-mile delivery [4].

The Traditional Supply Chain

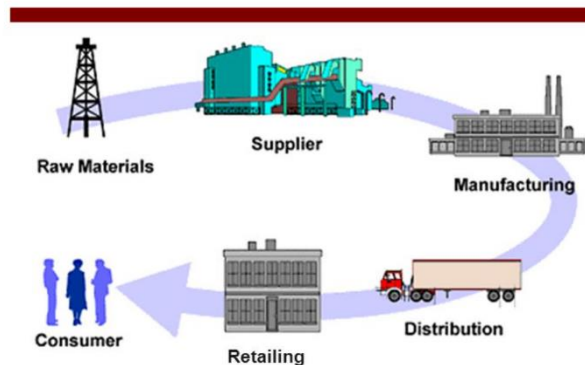


Fig. 2.1.1. Traditional Supply Chain Management

A traditional supply chain is a linear process that involves several steps as shown in Figure: 2.1.1.

1. **Raw materials:** This is the first step in the supply chain. Raw materials are the basic ingredients that are used to produce a good or service. In the image, they are sourced from a supplier.
2. **Manufacturing:** In this stage, the raw materials are transformed into a finished product. This can involve a variety of processes, such as assembly, packaging, and labeling.
3. **Distribution:** Once the products are manufactured, they need to be distributed to retailers or wholesalers. This can involve transportation, warehousing, and storage.
4. **Retailing:** This is the final stage in the supply chain, where the products are sold to consumers. Retailers can be brick-and-mortar stores or online businesses.
5. Traditional supply chains are typically focused on efficiency and cost-effectiveness. However, they can also be inflexible and slow to respond to changes in demand.

Despite its widespread use, traditional supply chain management has several disadvantages. One significant challenge is the lack of real-time visibility into the entire supply chain. Manual data entry and reliance on paper-based documentation make it difficult to track the movement of goods accurately, leading to delays in decision-making and response times. Furthermore, the absence of a centralized data repository often results in information silos among different supply chain partners, hindering collaboration and communication.

Disadvantages:

1. **Lack of Real-Time Visibility: Limited Flexibility and Responsiveness:** The traditional supply chain often grapples with a lack of real-time visibility, making it challenging for stakeholders to obtain up-to-date information on the status of products and processes. Manual data entry and paper-based documentation contribute to delays and inaccuracies, hindering the ability to make informed and timely decisions. This limitation can lead to inefficiencies, disruptions, and difficulties in quickly responding to changes in demand or supply chain dynamics.
2. **Limited Flexibility and Responsiveness:** Traditional supply chains can be rigid and slow to adapt to changes in market conditions or unforeseen disruptions. The lack of flexibility and responsiveness may result in difficulties in adjusting production schedules, reallocating resources, or quickly responding to shifts in customer demands. In a rapidly changing business environment, this inflexibility can lead to missed opportunities, increased costs, and challenges in maintaining competitiveness.
3. **Inventory Management Challenges:** Balancing optimal inventory levels is a perpetual challenge in traditional supply chain management. Overstocking ties up valuable capital and storage space, while understocking can lead to stockouts and dissatisfied customers. Manual inventory tracking systems and disconnected databases contribute to inaccuracies and inefficiencies in managing stock levels. This can result in increased holding costs, higher chances of stockouts, and challenges in meeting customer demands efficiently.
4. **Quality Control Issues:** Traditional supply chains may face difficulties in maintaining stringent quality control standards throughout the entire process. The lack of real-time visibility and traceability can impede the quick identification and resolution of quality issues. This limitation poses risks of defective or substandard products reaching end

consumers, potentially damaging brand reputation and customer trust. Quality control challenges may arise from inconsistent processes, manual record-keeping, and difficulties in enforcing standards across the supply chain.

5. **Sustainability and Environmental Impact:** Traditional supply chains may not align with modern sustainability goals. Inefficient transportation routes, excess packaging, and overreliance on non-renewable resources contribute to an environmental impact that is increasingly at odds with sustainability objectives.

In conclusion, while traditional supply chain management has been the cornerstone of global trade, it is not without its drawbacks. The limitations in real-time visibility, flexibility, inventory management, quality control, and sustainability have led industries to explore innovative solutions, including the integration of advanced technologies, to address these challenges and propel the supply chain into a more efficient and sustainable future.

2.2. Proposed System:

Supply chain management (SCM) using blockchain is a revolutionary paradigm that leverages decentralized and transparent ledger technology to address challenges inherent in traditional supply chain systems [5]. The foundational concept of blockchain is to create a decentralized and tamper-resistant ledger shared among all participants in the supply chain network. Unlike traditional centralized databases, blockchain ensures that every stakeholder has access to a single version of the truth, reducing errors, fraud, and delays.

This transformative approach enhances traceability and visibility across the entire supply chain. Each transaction is securely recorded, allowing stakeholders to trace the journey of products from origin to the end consumer. This level of transparency is particularly crucial in industries where tracking the provenance of goods is essential, such as food, pharmaceuticals, and luxury goods [5].

Smart contracts, a core feature of blockchain, automate and enforce predefined contractual agreements in supply chain management. These contracts streamline processes like order fulfillment, payment settlements, and compliance verification, reducing reliance on intermediaries and enhancing overall operational efficiency.

The reduction of counterfeiting and fraud is a significant advantage of blockchain in supply chain management. The decentralized and tamper-resistant nature of blockchain ensures the

integrity of product information, allowing participants to authenticate products at every stage. This safeguards consumer trust and protects brands from reputational damage caused by counterfeit goods entering the market.

Improved data security and privacy are critical aspects of blockchain technology. Robust cryptographic techniques secure information stored in the ledger, protecting sensitive data from unauthorized access and tampering. The decentralized architecture further reduces vulnerability, addressing security concerns associated with centralized databases in traditional supply chain systems.

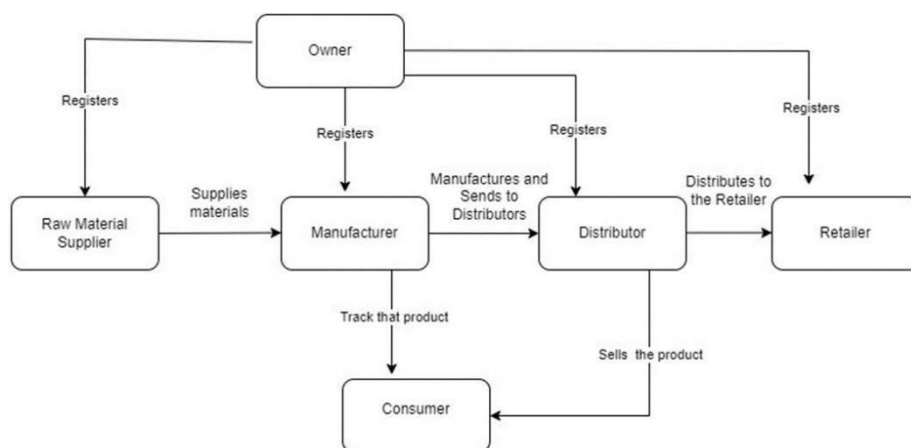


Fig. 2.2.1. Architecture diagram

The above Figure:2.2.1 is the Architecture of the Supply Chain Management system Using Block Chain.

Supplier: The process begins with a supplier, who provides raw materials to the manufacturer.

Manufacturer: The manufacturer receives the raw materials and uses them to create the product. The diagram shows the manufacturer sending the finished product to distributors.

Distributor: The distributor receives the product from the manufacturer and distributes it to retailers.

Retailer: The retailer sells the product to the consumer.

Consumer: The consumer purchases and uses the product.

Advantages:

1. **Transparency and Traceability:** Blockchain introduces unprecedented transparency and traceability into supply chain management. Each transaction is securely recorded and time-stamped on a decentralized ledger, providing a clear and immutable history of the product's journey from its origin to the end consumer. This level of transparency ensures that stakeholders can easily trace and verify the authenticity of products, fostering trust among participants and enabling compliance with regulatory requirements [6].
2. **Enhanced Security and Data Integrity:** The decentralized and tamper-resistant nature of blockchain technology significantly enhances the security and integrity of supply chain data. Traditional supply chains are vulnerable to data manipulation and fraud, but block chain's cryptographic algorithms make it extremely challenging for unauthorized parties to alter information. This heightened security reduces the risk of counterfeit products, unauthorized access, and data breaches, ensuring the reliability of information throughout the supply chain.
3. **Efficiency Through Smart Contracts:** Smart contracts, self-executing contracts with predefined rules and conditions, automate various processes within the supply chain. These contracts facilitate and enforce agreements between parties, such as payment settlements, order fulfillment, and compliance verification [6]. By automating these processes, block chain streamlines operations, reduces the need for intermediaries, and increases overall efficiency, leading to faster and more cost-effective supply chain management.
4. **Real-time Visibility and Decision-making: Block chain** enables real-time access to accurate and up-to-date information across the supply chain network. Stakeholders can monitor inventory levels, production status, and logistics in real-time, allowing for informed decision-making. The availability of timely data enhances agility and responsiveness, enabling organizations to adapt quickly to changes in demand, supply chain disruptions, or market conditions, ultimately improving overall operational efficiency.
5. **Reduction of Counterfeiting and Fraud:** One of the significant advantages of blockchain in supply chain management is the reduction of counterfeiting and fraud. The

transparent and tamper-resistant nature of the decentralized ledger ensures the integrity of product information. Participants can verify the authenticity of products at each stage of the supply chain, minimizing the risk of counterfeit goods entering the market. This not only protects consumer trust but also preserves the reputation of brands, contributing to long-term sustainability and success.

Blockchain fosters collaboration, supports ethical practices, and increases resilience to disruptions. As industries continue to adopt and adapt block chain solutions, the benefits of a more efficient, transparent, and resilient supply chain become increasingly apparent.

2.3. Feasibility Study:

The feasibility of the project is analyzed in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- **ECONOMICAL FEASIBILITY**
- **TECHNICAL FEASIBILITY**
- **SOCIAL FEASIBILITY**
- **OPERATIONAL FEASIBILITY**
- **FINANCIAL FEASIBILITY**
- **LEGAL AND REGULATORY FEASIBILITY**

Economic feasibility:

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

Technical feasibility:

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

Social feasibility:

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it [7]. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

Operational Feasibility:

Operational feasibility focuses on determining whether the proposed system can be effectively integrated into the company's operations without disrupting existing processes. This involves assessing the impact on workflow, training requirements for employees, and any potential resistance to change [7]. Operational feasibility also considers the scalability and flexibility of the proposed system to accommodate future growth and evolving business needs.

Financial Feasibility:

Financial feasibility evaluates the economic viability of the proposed system, considering both the initial investment and ongoing operational costs. This includes estimating the costs associated with software development, hardware procurement, training, maintenance, and any potential savings or revenue generated by the system. Cost-benefit analysis is conducted to determine the return on investment (ROI) and ensure that the proposed system offers tangible benefits that outweigh its costs.

Legal and Regulatory Feasibility:

Legal and regulatory feasibility involves assessing whether the proposed system complies with relevant laws, regulations, and industry standards. This includes considerations for data privacy, security, intellectual property rights, and compliance with certifications or accreditations. Legal and regulatory feasibility ensures that the proposed system mitigates any potential legal risks and operates within the bounds of applicable regulations.

Preliminary Business Proposal:

Based on the feasibility analysis, a preliminary business proposal is formulated, outlining the general plan for the project and providing cost estimates.

The business proposal includes:

Project Scope and Objectives: Clearly defining the scope and objectives of the project, including the desired outcomes and deliverables.

System Requirements: Detailing the technical, operational, and regulatory requirements for the proposed system, based on the feasibility analysis.

Implementation Plan: Outlining the timeline, milestones, and tasks required to implement the proposed system, including resource allocation and project management strategies.

Cost Estimates: Providing estimates for the initial investment and ongoing operational costs associated with developing, implementing, and maintaining the proposed system.

Risk Analysis: Identifying potential risks and mitigation strategies to minimize disruptions and ensure the successful implementation of the proposed system.

Overall, the feasibility analysis and business proposal provide essential insights into the viability of the proposed system and lay the foundation for informed decision-making and planning. By addressing key requirements and considerations, the company can assess the potential benefits and risks of implementing the proposed blockchain-based Supply chain management system

3.System Requirements Specification

Requirement analysis is a vital step for determining the success of a system or software project. The importance of requirement analysis lies in its ability to ensure alignment between the project's objectives and the solutions being developed. By meticulously identifying and prioritizing requirements, potential misunderstandings and scope creep are minimized, thus reducing the risk of project failure and budget overruns.

3.1.Software Requirements:

Software requirements The importance of requirement analysis lies in its ability to ensure alignment between the project's objectives and the solutions being developed. By meticulously identifying and prioritizing requirements, potential misunderstandings and scope creep are minimized, thus reducing the risk of project failure and budget overruns. for a project refer to a comprehensive description of the functionalities, capabilities, constraints, and qualities that a software system must possess in order to meet the needs and expectations of its stakeholders. These requirements serve as the foundation for the design, development, and testing of the software solution. They typically encompass both functional requirements, which describe specific behaviors or functions the software must perform, and non-functional requirements, which define attributes such as performance, security, usability, and reliability.

Functional Requirements:

Functional requirements define specific behaviors or functions that the software must perform to fulfill its intended purpose. These requirements outline the desired features, actions, and interactions of the software system. Functional requirements answer questions such as "What should the software do?" and include:

User Features: Describing the features and capabilities that users expect from the software, such as user interfaces, data input and output, and navigation.

System Features: Identifying the core functionalities and operations that the software system must support, including data processing, calculations, and business logic.

Integration Requirements: Specifying how the software interacts with external systems, databases, APIs, or third-party services to exchange data and information.

Error Handling: Defining how the software should respond to various error conditions, exceptions, and unexpected events to ensure robustness and reliability.

Non-Functional Requirements:

Non-functional requirements define attributes or qualities of the software system that are not directly related to specific functionalities but are critical for its overall performance, usability, security, and reliability. These requirements address aspects such as:

Performance: Establishing performance benchmarks, response times, throughput, and scalability requirements to ensure the software meets performance expectations under varying workloads.

Security: Outlining security measures, access controls, encryption standards, and compliance requirements to protect sensitive data and prevent unauthorized access or breaches.

Usability: Describing user interface design principles, accessibility standards, and user experience guidelines to ensure the software is intuitive, easy to use, and accessible to all users.

Reliability: Specifying reliability metrics, fault tolerance mechanisms, recovery procedures, and system availability targets to ensure the software operates consistently and reliably under normal and adverse conditions.

Maintainability: Defining code quality standards, documentation requirements, and version control practices to facilitate software maintenance, updates, and enhancements over time.

S/W CONFIGURATION:

The software configuration process typically involves selecting the appropriate software components based on the project requirements and compatibility considerations. It also involves configuring these components to work together seamlessly, ensuring optimal

performance and reliability. Additionally, software configuration encompasses aspects such as version control, dependency management, and environment setup to maintain consistency across development, testing, and deployment stages.

- **Operation system:** Windows 8 or later/ MacOS Sierra 10.12 or later/64-bit Ubuntu 14.04+
- **Front end:** Java script, HTML, CSS, React.js
- **Blockchain Technologies:**

Ganache:

Ganache is used for the overall development of the project. This tool is used to develop the project by giving different scenarios and testing those scenarios whether they are working fine or not.

If those test cases are successfully working then we can deploy them directly into the project. This is the main part of the blockchain project where we can create a strong base for the project with different scenarios and test cases useful for the project. This tool is also used for debugging to check the created test cases. Our project contains a separate truffle config file, which is mainly used for the ganache. Based on this file, the ganache will provide the addresses and keys for the people who are involved in developing the product or goods. So, this is the main tool for the application. The developers who are developing the blockchain project should mainly concentrate on the ganache because they are the people who are experimenting with the test cases with different scenarios.

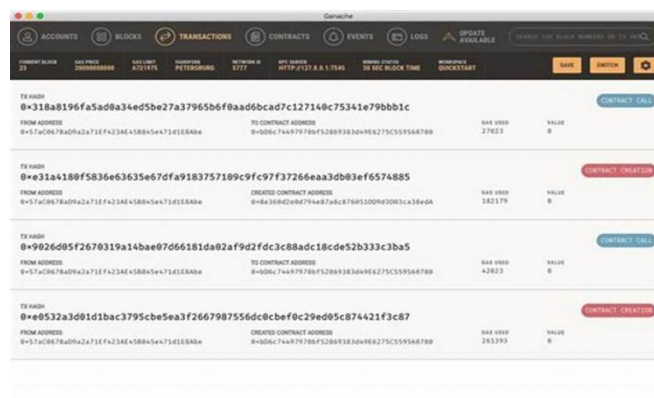


Fig. 3.1.1. Ganache

MetaMask:

An addition to the blockchain wallet that handles contracts and transactions is called MetaMask. MetaMask is essential to the field of supply chain management because it allows users to communicate with blockchain apps, particularly those developed on the Ethereum network. We have to add the extension of the metamask to Chrome or other search engines. We have to create an Ethereum project in the metamask. The addresses and private keys created in the ganache are added to the metamask to connect the project. Based on the private keys added, the transactions will occur between the different sources.

Finally, the products or goods will reach the customer after all these stages. We can also use other tools which work similar to these. These transactions and contracts will be secure and safe as they can only be restricted to the people who are using the project. They cannot be edited and modified. All the transactions are stored in the cloud or database for verification purposes. This guarantees the transparency and immutability of the recorded data on the Ethereum blockchain in addition to improving supply chain activities' efficiency.

Smart contracts built on Ethereum can be easily integrated into supply chain operations thanks to MetaMask. Order fulfillment, record-keeping, and payment settlement are just a few of the supply chain processes that smart contracts may automate. MetaMask acts as a bridge, enabling users to interact with these smart contracts through a user-friendly interface.

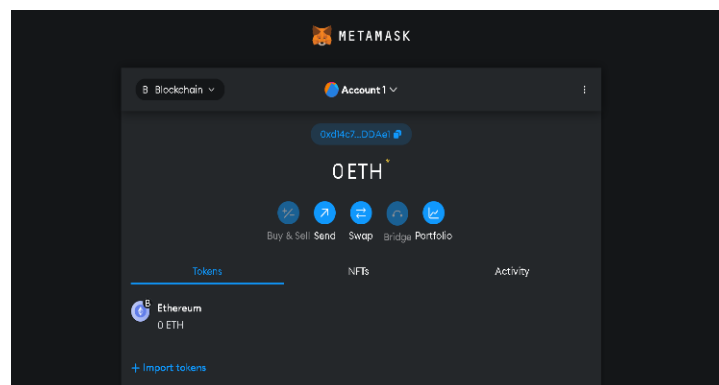


Fig. 3.1.2. Metamask

This guarantees the transparency and immutability of the recorded data on the Ethereum blockchain in addition to improving supply chain activities' efficiency.

Solidity:

Ethereum is one of the most well-known blockchain systems, and Solidity is a high-level programming language made especially for creating smart contracts on these networks. Solidity is essential to supply chain management because it enables smart contracts to automate and carry out business logic. Smart contracts facilitate transparency, immutability, and trust among supply chain participants by enacting terms directly into code and functioning as self-executing agreements.

The development of smart contracts that correspond to different supply chain phases is one method Solidity is applied to supply chain management. One way to use smart contracts is to configure them to run automatically in response to specific events, such as when a product is sent, received, or goes through quality control inspections. Because of this automation, there is a decreased need for middlemen, a lower chance of mistakes, and a more secure blockchain record of supply chain activities.

Furthermore, Solidity makes it possible to integrate supply chain elements like transparency and traceability. The blockchain allows for the transparent and unchangeable recording of every event and transaction in the supply chain, making it available to all authorized parties. By improving accountability and lowering the likelihood of fraud or counterfeiting, this transparency aids in tracking the origin, transportation, and status of products. Overall, Solidity's capability to create robust, decentralized, and automated smart contracts makes it a powerful tool for improving efficiency and accountability in supply chain management on blockchain platforms.

This SCM with blockchain can be used by any sector like agriculture, industries, railways, factories, etc. For every sector where the products or goods are developing and selling to the customers, we can use this supply chain process project.

3.2.HARDWARE REQUIREMENTS:

Hardware requirements for a project refer to the specifications and characteristics of the physical components necessary to support and run the software solution effectively. These requirements detail the minimum and recommended hardware configurations needed for the

software to operate optimally, considering factors such as processing power, memory (RAM), storage capacity, network connectivity, and peripheral devices.

In any software project, defining the hardware requirements is essential to ensure that the software can run effectively and efficiently on the underlying physical infrastructure. Hardware requirements outline the specifications and characteristics of the physical components necessary to support and execute the software solution. These requirements consider various factors such as processing power, memory (RAM), storage capacity, network connectivity, and peripheral devices. Here's a detailed elaboration on hardware requirements:

1. Processing Power:

The processing power of the hardware, typically measured in terms of CPU (Central Processing Unit) speed and number of cores, is crucial for executing the software efficiently. The hardware must have sufficient processing power to handle the computational demands of the software, including data processing, calculations, and other tasks.

2. Memory (RAM):

RAM (Random Access Memory) is essential for storing and accessing data and program instructions temporarily while the software is running. The hardware must have an adequate amount of RAM to support the software's memory requirements and ensure smooth performance without excessive delays or slowdowns.

3. Storage Capacity:

Storage capacity refers to the amount of data that can be stored on the hardware device, typically measured in gigabytes (GB) or terabytes (TB). The hardware should have sufficient storage capacity to accommodate the software installation files, databases, and other data storage requirements. Additionally, considerations may include the type of storage (e.g., SSDs for faster access or HDDs for larger capacity) based on performance and cost factors.

4. Network Connectivity:

Network connectivity is essential for communication between the software system and other devices, servers, or networks. The hardware should support reliable and high-speed network connections, such as Ethernet or Wi-Fi, to ensure seamless data exchange and connectivity.

5. Peripheral Devices:

Peripheral devices, such as printers, scanners, monitors, keyboards, and mice, may be required to interact with the software system. The hardware should have compatible interfaces and ports to connect these peripheral devices and support their functionality.

Minimum and Recommended Configurations:

Hardware requirements typically include both minimum and recommended configurations. The minimum configuration specifies the least amount of hardware resources needed for the software to run, while the recommended configuration outlines optimal hardware specifications for optimal performance and user experience.

Hardware requirements are critical for ensuring that the software solution can operate effectively and efficiently on the underlying physical infrastructure. By specifying the necessary processing power, memory, storage capacity, network connectivity, and peripheral devices, stakeholders can ensure that the hardware environment adequately supports the software's functionality and performance requirements. Additionally, considering both minimum and recommended configurations helps in providing flexibility and scalability to accommodate varying user needs and usage scenarios.

H/W CONFIGURATION:

Hardware configuration refers to the setup and arrangement of physical components required to support the software and execute the project tasks efficiently. This includes servers, networking equipment, storage devices, peripherals, and any other hardware resources needed to host and run the software application or system.

Hardware configuration involves selecting hardware components that meet the project's performance, scalability, and reliability requirements. Factors such as processing power, memory capacity, storage capacity, and network bandwidth are carefully considered during the hardware configuration process. Additionally, hardware configuration may involve setting up redundancy, load balancing, and other mechanisms to ensure high availability and fault tolerance.

- Processor - I3/Intel Processor
- Hard Disk -160GB
- Key Board - Standard Windows Keyboard

- Mouse - Two or Three Button Mouse
- Monitor - SVGA
- RAM - 4Gb

3.2.Functional and Non-functional requirements:

Requirement analysis is a very critical process that enables the success of a system or software project to be assessed. Requirements are generally split into two types: Functional and non-functional requirements.

Functional Requirements:

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Functional requirements constitute the core features and capabilities that end users expect from the system to fulfill their specific needs and demands. These requirements are essential functionalities that users require as basic facilities, and they must be incorporated into the system as part of the contractual agreement. Functional requirements are typically represented or stated in terms of the input required by the system, the operations to be performed, and the expected output. Unlike non-functional requirements, which focus on attributes like performance, security, and usability, functional requirements directly impact the system's behavior and functionality.

Here's an elaboration on functional requirements:

1. Input Requirements:

Functional requirements specify the inputs that users need to provide to the system to execute certain operations or tasks. These inputs could include various forms of data, commands, parameters, or user interactions required to initiate specific functionalities within

the system.

2. Operations to be Performed:

Functional requirements outline the specific operations, actions, or tasks that the system should perform in response to the provided inputs. These operations represent the core functionalities and behaviors that users expect from the system, such as data processing, calculations, data manipulation, or business logic execution.

3. Expected Output:

Functional requirements define the expected outputs or results that the system should generate after performing the specified operations. These outputs could include processed data, reports, notifications, user interface updates, or any other feedback provided to the user in response to their inputs.

4. Direct User Requirements:

Functional requirements are directly derived from the needs and demands of the end users, representing the functionalities that users can directly observe and interact with in the final product. These requirements reflect the user's expectations and serve as the basis for designing and developing the system to meet user needs effectively.

5. Contractual Obligations:

Functional requirements are often included as part of the contractual agreement between the stakeholders, including the users, clients, and development team. These requirements serve as binding commitments that the development team must fulfill to deliver a satisfactory product that meets the user's expectations and requirements.

Functional requirements form the foundation of the system's functionality, outlining the specific features, operations, and outputs that users expect from the system. These requirements are directly derived from the user's demands and serve as contractual obligations that must be met to deliver a successful and satisfactory product. By incorporating functional requirements into the system design and development process, stakeholders can ensure that the system effectively meets user needs and delivers the desired functionalities and capabilities.

Examples of functional requirements:

- 1) Authentication of user whenever he/she logs into the system
- 2) System shutdown in case of a cyber-attack
- 3) A verification email is sent to the user whenever he/she registers for the first time on some software system.

Non-functional requirements:

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements.

Non-functional requirements, also known as quality constraints, define the attributes and characteristics of the system that contribute to its overall quality, usability, and performance. Unlike functional requirements, which specify the system's behavior and functionalities, non-functional requirements focus on the qualitative aspects of the system that impact its performance, reliability, security, and user experience. These requirements are essential for ensuring that the system meets the project contract's quality standards and user expectations. Here's an elaboration on non-functional requirements:

1. Quality Constraints:

Non-functional requirements specify the quality constraints or attributes that the system must satisfy according to the project contract. These constraints encompass various aspects such as performance, reliability, security, usability, maintainability, and scalability, which collectively contribute to the overall quality of the system.

2. Varying Priorities:

The priority or extent to which non-functional requirements are implemented may vary from one project to another based on project objectives, constraints, and stakeholders' preferences. Some non-functional requirements may be considered critical and must be implemented rigorously, while others may have lower priority or flexibility in their implementation.

3. Non-behavioral Requirements:

Non-functional requirements are sometimes called non-behavioral requirements because they do not directly specify the system's functionalities or behaviors. Instead, they focus on the system's qualities, attributes, and characteristics that influence its performance and user experience.

4. Examples of Non-functional Requirements:

Performance: Specifies the system's response time, throughput, and scalability requirements under various load conditions.

Reliability: Defines the system's ability to perform consistently and reliably over time without failure or errors.

Security: Outlines the measures and controls to protect the system from unauthorized access, data breaches, and security vulnerabilities.

Usability: Describes the ease of use, intuitiveness, and user experience of the system for end users.

Maintainability: Specifies the ease of maintenance, updates, and enhancements to the system's codebase, architecture, and documentation.

Scalability: Defines the system's ability to handle increasing workload and user demands by scaling its resources and infrastructure accordingly.

5. Contractual Obligations:

Non-functional requirements are integral components of the project contract, serving as contractual obligations that the development team must adhere to. These requirements ensure that the system meets the specified quality standards and user expectations, ultimately contributing to the project's success and stakeholders' satisfaction.

Non-functional requirements play a crucial role in ensuring the overall quality, performance, and user experience of the system. By defining quality constraints and attributes, these requirements help to meet the project contract's quality standards and fulfill stakeholders' expectations. Incorporating non-functional requirements into the system development

process ensures that the system meets the desired quality criteria and delivers a satisfactory user experience.

They basically deal with issues like:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

Examples of non-functional requirements:

- 1) Emails should be sent with a latency of no greater than 12 hours from such an activity.
- 2) The processing of each request should be done within 10 seconds
- 3) The site should load in 3 seconds whenever of simultaneous users are > 10000

Blockchain is a decentralized, distributed ledger technology that enables secure and transparent recording of transactions across a network of computers. The process begins with a transaction request being broadcasted to the network. This transaction is then verified by multiple participants, known as nodes, through a consensus mechanism such as Proof of Work or Proof of Stake. Once verified, the transaction is grouped with other transactions into a block. Each block contains a cryptographic hash of the previous block, creating a chain of blocks, hence the name "blockchain." The block is then added to the existing chain, forming an immutable record of transactions. This decentralized nature ensures transparency, security, and immutability of data, making blockchain a promising technology for various industries, including finance, supply chain, and healthcare.

By lowering the possibility of fraud, unauthorized access, and data manipulation, this improves the overall security of the supply chain data and transactions. MetaMask simplifies the blockchain's complexity, making it easier for users to interact with it. Non-technical users may interact with smart contracts, track product provenance, and confirm authenticity throughout the supply chain more easily because of its user-friendly interface

and straightforward architecture.

The entire supply chain ecosystem is made more transparent, secure and efficient through the use of Solidity, a programming language created for Ethereum blockchain smart contracts. By utilizing Solidity smart contracts, stakeholders within the supply chain can automate and streamline various processes, including tracking and verifying the authenticity of products, recording transactions, and enforcing contractual agreements

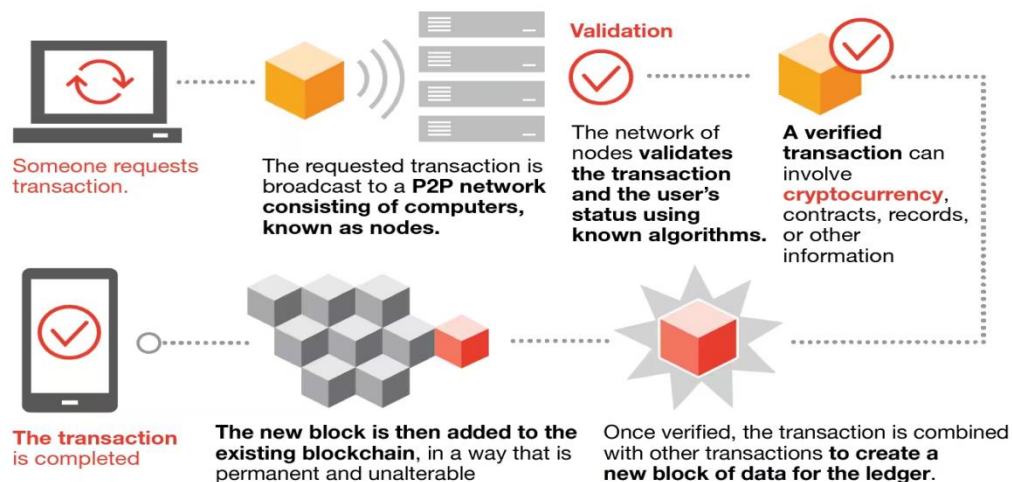


Fig. 3.2.1. Flow of Blockchain

Supply chain players are more likely to trust one another because of the immutability and decentralized nature of blockchain, which guarantee transparent and tamper-resistant data. 6A groundbreaking approach that solves a number of the problems with conventional supply chain systems is Solidity for Supply Chain Management, which leverages blockchain.

An ecosystem for the global supply chain that is more dependable and trustworthy is made possible by the integration of smart contracts written in Solidity, which increases efficiency, transparency, and security.

In summary, a new era of effectiveness, openness, and confidence in supply chain management is ushered in by the combination of blockchain with Ganache, MetaMask, and solidity. By providing a safe and efficient method of handling the intricate network of exchanges and transactions in the supply chain, this technological synergy resolves long-standing problems in the sector. The potential advantages of this technology are enormous for both consumers and businesses, and it will open the door for a more

responsive and robust global supply chain ecosystem as it develops and gains traction.

3.3. Software Installation for Supply Chain Management System using Block chain :

Solidity:

Solidity is a high-level programming language primarily used for writing smart contracts on blockchain platforms, notably Ethereum. It is designed to enable the creation of secure and decentralized applications (DApps) by facilitating the implementation of business logic within smart contracts. Solidity is statically typed, supports inheritance, libraries, and complex user-defined types among other features, making it suitable for a wide range of decentralized applications [9].

One of the key characteristics of Solidity is its close integration with the Ethereum Virtual Machine (EVM), the runtime environment for executing smart contracts on the Ethereum blockchain. Solidity code is compiled into bytecode that can be executed by the EVM, enabling the deployment and execution of smart contracts on the Ethereum network.

Solidity allows developers to define data structures, functions, and events within smart contracts, enabling the creation of complex decentralized applications. It also provides mechanisms for handling ether (the cryptocurrency of the Ethereum network) and interacting with other smart contracts and external services.

Due to its importance in the Ethereum ecosystem, Solidity has become the de facto standard for writing smart contracts and is widely used by blockchain developers. However, it's important to note that writing secure Solidity code requires careful attention to best practices and security considerations, as vulnerabilities in smart contracts can lead to significant financial losses and other adverse consequences. As such, ongoing efforts are made to improve the language and tooling surrounding it to enhance security and usability for developers. Ethereum is a decentralized, open-source blockchain platform that enables the development of decentralized applications (DApps) and smart contracts. It was proposed by Vitalik Buterin in late 2013 and went live on July 30, 2015. Unlike Bitcoin, which primarily serves as a digital currency and payment system, Ethereum is a programmable blockchain platform that supports a wide range of applications beyond simple transactions [9].

Ethereum:

Ethereum is a decentralized, open-source blockchain platform that enables the development of decentralized applications (DApps) and smart contracts. It was proposed by Vitalik Buterin in late 2013 and went live on July 30, 2015. Unlike Bitcoin, which primarily serves as a digital currency and payment system, Ethereum is a programmable blockchain platform that supports a wide range of applications beyond simple transactions.

One of Ethereum's key features is its ability to execute smart contracts, which are self-executing contracts with the terms of the agreement directly written into code. Smart contracts enable developers to create decentralized applications that can autonomously execute predefined actions without the need for intermediaries [10].

Ethereum uses a cryptocurrency called Ether (ETH) as a means of value exchange within the network [10]. Ether is used to compensate participants who perform computations and validate transactions on the Ethereum blockchain, a process known as mining or, as Ethereum transitions to a proof-of-stake consensus mechanism, staking.

Ethereum's architecture consists of a global network of nodes, each running the Ethereum software and participating in the consensus process. This decentralized network ensures the security and integrity of the platform, making it resistant to censorship and single points of failure.

The Ethereum ecosystem is vibrant and dynamic, with a large and active community of developers, entrepreneurs, and enthusiasts continuously building and innovating on the platform [10]. Ethereum's flexibility and programmability have led to the development of diverse applications, including decentralized finance (DeFi), non-fungible tokens (NFTs), decentralized exchanges (DEXs), and more.

However, Ethereum faces challenges such as scalability, high transaction fees during periods of network congestion, and environmental concerns due to its energy-intensive mining process [10]. Efforts are underway to address these challenges through upgrades such as Ethereum 2.0, which aims to transition the network to a more scalable and sustainable proof-of-stake consensus mechanism.

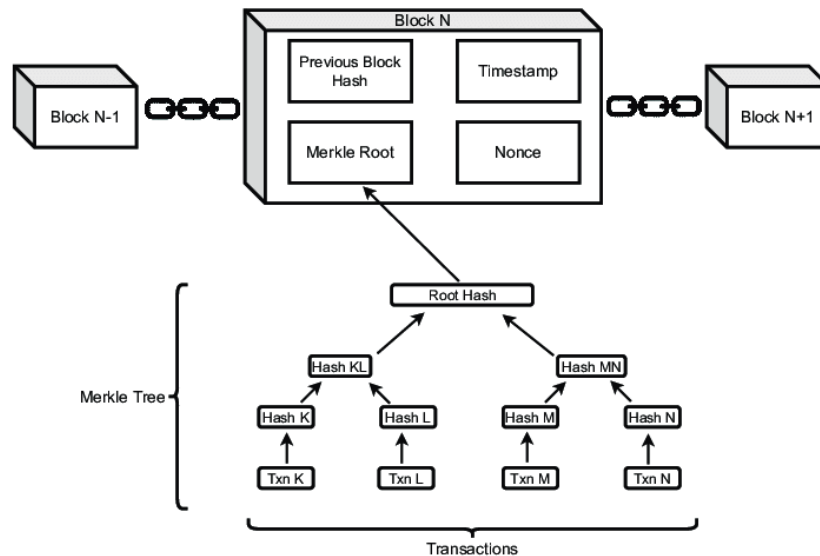


Fig. 3.3.1. Block representation in detail

Overall, Ethereum has emerged as a leading platform for decentralized innovation, empowering developers to build the next generation of decentralized applications and reshape various industries through blockchain technology.

Smart Contracts:

A smart contract represents a significant innovation in the realm of blockchain technology, serving as a self-executing digital agreement with predefined terms directly coded into its structure. Unlike traditional contracts, which rely on intermediaries to enforce agreements, smart contracts leverage blockchain platforms such as Ethereum to automatically execute actions when specified conditions are met. This autonomy is a fundamental characteristic of smart contracts, enabling them to operate without the need for human intervention once deployed. This aspect not only streamlines processes but also minimizes the potential for errors or disputes, thereby enhancing efficiency and reducing costs associated with contract execution and enforcement.

Moreover, smart contracts operate in a trustless environment, meaning that transactions conducted through smart contracts do not require trust in a central authority or intermediary. Instead, trust is placed in the underlying blockchain protocol and the code of the smart contract itself. The transparency of smart contracts further contributes to this trustless nature, as the code is typically open-source and immutable, allowing all parties to inspect the terms and logic of the contract. This transparency fosters a higher degree of

confidence in the integrity and fairness of transactions, promoting greater trust and reliability in decentralized systems.

Security is another crucial aspect of smart contracts, as they leverage cryptographic techniques to ensure the security and integrity of transactions. Once deployed, the code of a smart contract cannot be altered, reducing the risk of fraud or manipulation. However, it's important to acknowledge that smart contracts are not without risks. Security vulnerabilities in smart contract code can lead to financial losses or exploitation by malicious actors. Therefore, thorough testing, auditing, and best practices in smart contract development are essential to mitigate these risks and ensure the reliability and integrity of smart contract-based systems.

In conclusion, smart contracts offer a wide range of benefits, including autonomy, trustlessness, transparency, efficiency, and security. Their versatility and applicability across various industries make them a powerful tool for revolutionizing numerous aspects of business operations, finance, supply chain management, and more. However, it's crucial to approach smart contract development with caution and adhere to rigorous security standards to mitigate potential risks and ensure the successful implementation of smart contract-based solutions.

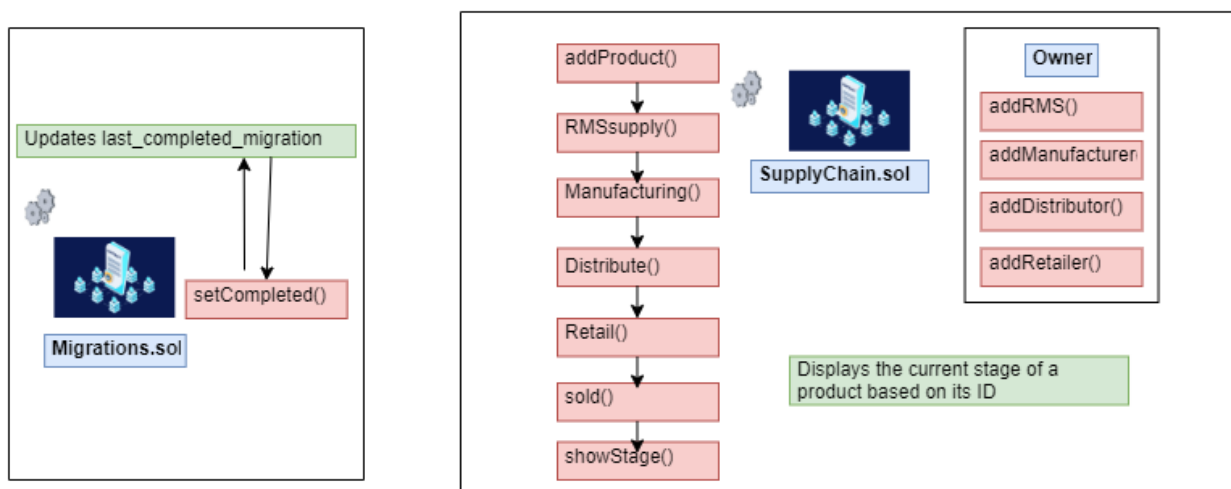


Fig. 3.3.2. Code Flow of supply chain management

The above figure 3.3.2 resembles the functions included in the smart contracts included in Supply chain Management system using Block chain.

product to the system and triggers the RMSsupply() function.

RMSsupply() then calls addRMS(), which possibly creates a record in the system for the newly added product in a database table named RMS.

From there, the flowchart splits into two paths. One path leads to Manufacturing(), and the other goes to SupplyChain.sol.

The Manufacturing() path includes addManufacturer(), which could be adding a manufacturer to the product record, and setCompleted(), which likely signifies the manufacturing stage being complete.

The SupplyChain.sol path involves addDistributor() and addRetailer(), presumably indicating adding a distributor and a retailer to the product record for the supply chain.

After either the manufacturing path or the supply chain path is completed, the function showStage() is called. This function likely retrieves and displays the current stage the product is in, based on its ID.

Finally, the product goes to the Retail() stage, which could indicate that the product is now available for purchase by the end consumer. There is also a sold() function here, which presumably means the product has been sold.

Installing Metamask:

1. Open Chrome Web Store : Open your Google Chrome or Brave browser and go to the Chrome Web Store.
2. Search for MetaMask : In the search bar of the Chrome Web Store, type "MetaMask" and press Enter [10].

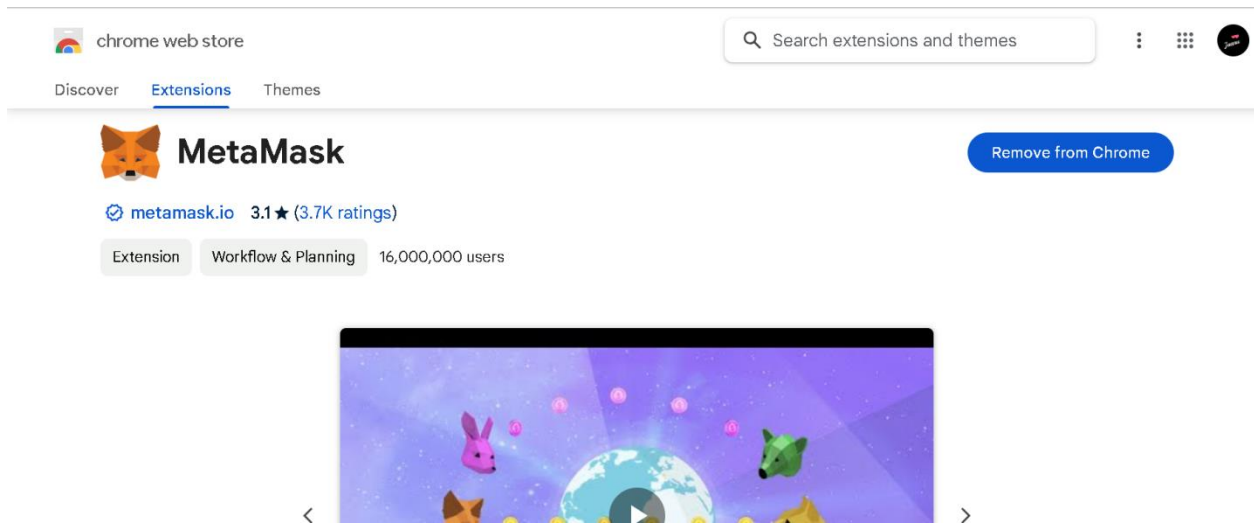


Fig. 3.3.3. Metamask interface

3. Add MetaMask Extension: Look for the MetaMask extension in the search results and click on it.
4. Add to Chrome: Click on the "Add to Chrome" button to install the MetaMask extension.
5. Confirm Installation: A confirmation dialog will appear. Click "Add Extension" to confirm and install MetaMask.
6. Open MetaMask: After installation, you'll see the MetaMask icon added to your browser's toolbar. Click on it to open MetaMask.

Installing Ganache:

1. To install Ganache on Windows, you can follow these steps:
2. Visit [ethereum official Ganache website](https://www.trufflesuite.com/ganache): [Ganache](<https://www.trufflesuite.com/ganache>), and navigate to the download section.

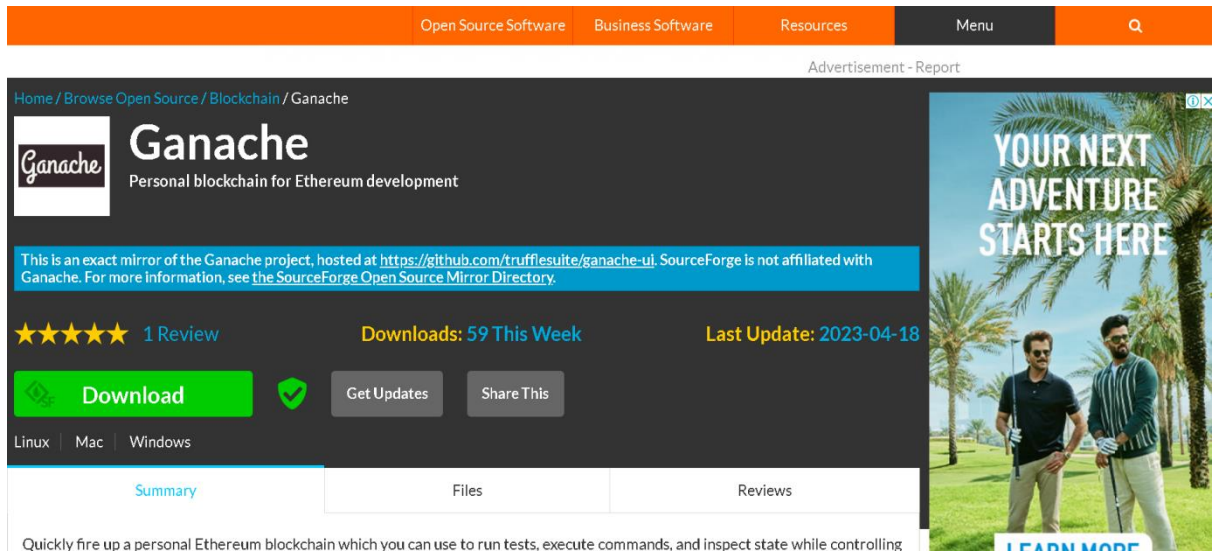


Fig. 3.3.4. Ganache interface

3. Click on the download button for the Windows version of Ganache. It typically comes as an executable installer file (.exe).
4. Once the download is complete, locate the downloaded .exe file (usually in your Downloads folder) and double-click on it to run the installer.
5. Follow the instructions provided by the Ganache installation wizard. This typically involves selecting the installation directory, agreeing to the terms of service, and clicking through the installation process.
6. After the installation is complete, you may be prompted to launch Ganache immediately. If not, you can find Ganache in your Start menu or desktop shortcut and launch it from there. Double-click on the Ganache shortcut to run the application. This will start Ganache, and you'll see a window with details about your local Ethereum blockchain, including accounts, private keys, and transaction history.

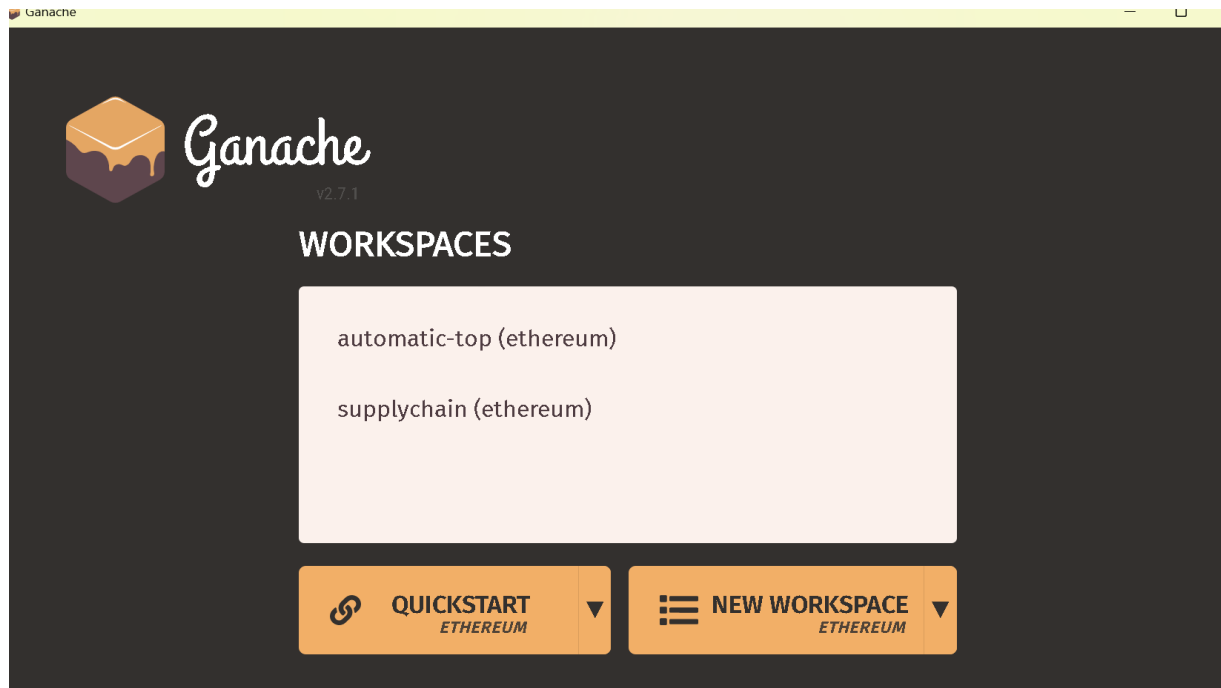


Fig. 3.3.5. Ganache interface(local blockchain)

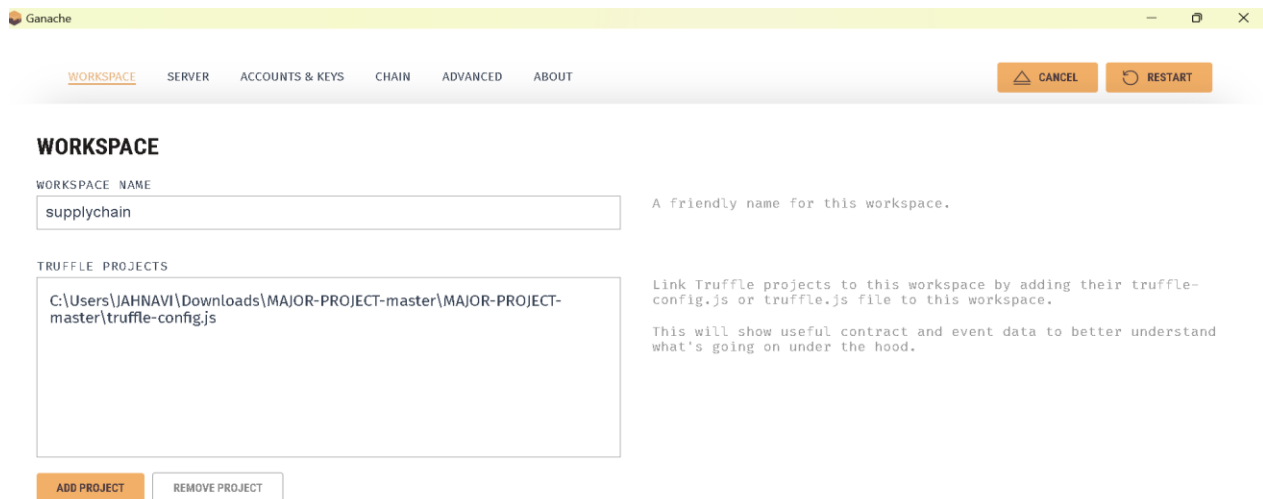


Fig. 3.3.6. Ganache blockchain creation

7. You may want to configure Ganache settings according to your requirements. This includes setting the number of accounts, the initial balance of each account, network ID, gas limit, and other advanced settings.
8. Once Ganache is running, you can connect it to your development environment, such as Remix, Truffle, or web3.js, to deploy and interact with smart contracts and develop Ethereum-based applications [9].
9. With Ganache installed and running, you now have a local Ethereum blockchain environment ready for development and testing. You can deploy smart contracts, interact with them, and debug your applications locally before deploying them to the Ethereum mainnet.

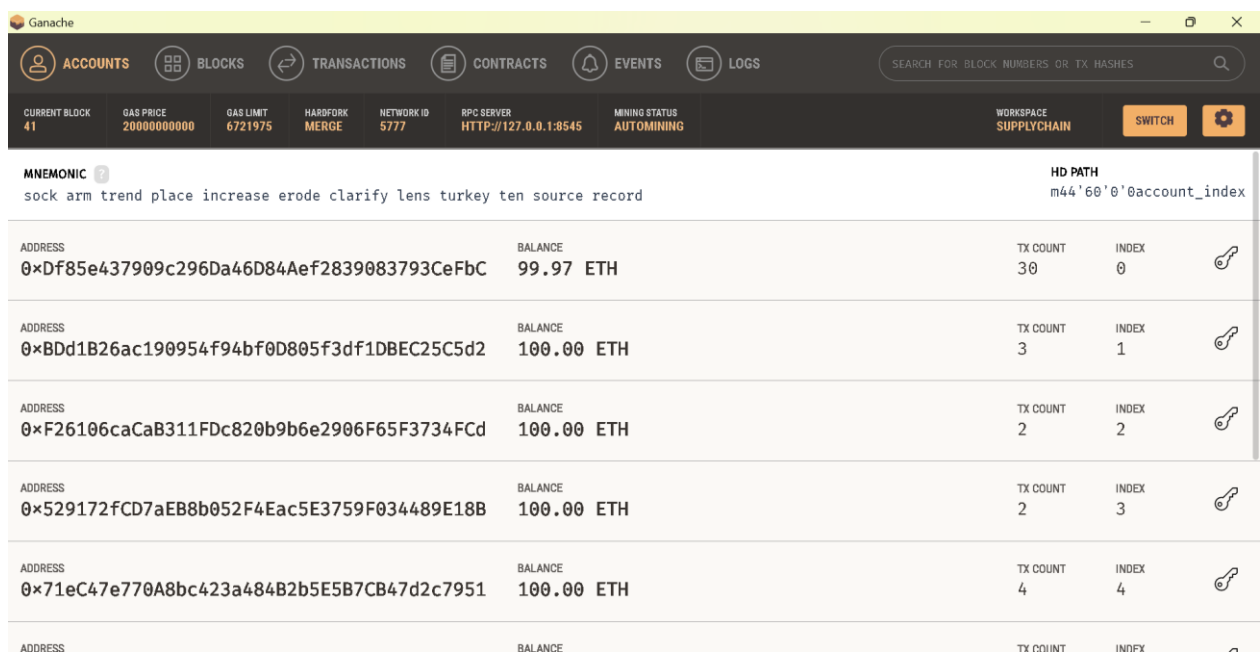


Fig. 3.3.7. Ganache local blockchain

Node Modules used for Blockchain project:

1. npm install truffle:

This command shown in Figure:3.3.8. installs Truffle globally on your system. Truffle is a development environment, testing framework, and asset pipeline for Ethereum. It allows you to write, compile, deploy, and test smart contracts on the Ethereum blockchain.

```

C:\Users\yelet\Downloads\Blockchain-SCM-main\Blockchain-SCM-main\client>npm install truffle
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: '@truffle/abi-utils@1.0.3',
npm WARN EBADENGINE   required: { node: '^16.20 || ^18.16 || >=20' },
npm WARN EBADENGINE   current: { node: 'v16.17.0', npm: '9.2.0' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: '@truffle/code-utils@3.0.4',
npm WARN EBADENGINE   required: { node: '^16.20 || ^18.16 || >=20' },
npm WARN EBADENGINE   current: { node: 'v16.17.0', npm: '9.2.0' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: '@truffle/codec@0.17.3',
npm WARN EBADENGINE   required: { node: '^16.20 || ^18.16 || >=20' },
npm WARN EBADENGINE   current: { node: 'v16.17.0', npm: '9.2.0' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: '@truffle/compile-common@0.9.8',
npm WARN EBADENGINE   required: { node: '^16.20 || ^18.16 || >=20' },
npm WARN EBADENGINE   current: { node: 'v16.17.0', npm: '9.2.0' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: '@truffle/db-loader@0.2.36',
npm WARN EBADENGINE   required: { node: '^16.20 || ^18.16 || >=20' },
npm WARN EBADENGINE   current: { node: 'v16.17.0', npm: '9.2.0' }
npm WARN EBADENGINE }
npm WARN EBADENGINE Unsupported engine {
npm WARN EBADENGINE   package: '@truffle/debugger@12.1.5',
npm WARN EBADENGINE   required: { node: '^16.20 || ^18.16 || >=20' },
npm WARN EBADENGINE   current: { node: 'v16.17.0', npm: '9.2.0' }

```

Fig. 3.3.8. truffle installation

```

npm WARN deprecated apollo-reporting-protobuf@3.4.0: The 'apollo-reporting-protobuf' package is part of Apollo Server v2 and v3, which are now deprecated (end-of-life October 22nd 2023 and October 22nd 2024, respectively). This package's functionality is now found in the '@apollo/usage-reporting-protobuf' package. See https://www.apollographql.com/docs/apollo-server/previous-versions/ for more details.
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated apollo-server-env@4.2.1: The 'apollo-server-env' package is part of Apollo Server v2 and v3, which are now deprecated (end-of-life October 22nd 2023 and October 22nd 2024, respectively). This package's functionality is now found in the '@apollo/usage-reporting-protobuf' package. See https://www.apollographql.com/docs/apollo-server/previous-versions/ for more details.
npm WARN deprecated multicodec@0.5.7: stable api reached
npm WARN deprecated @hapi/hoek@8.5.1: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated @hapi/joi@15.1.1: Switch to 'npm install joi'
npm WARN deprecated @ensdomains/ens@0.4.5: Please use @ensdomains/ens-contracts
npm WARN deprecated popper.js@1.16.1: You can find the new Popper v2 at @popperjs/core, this package is dedicated to the legacy v1
npm WARN deprecated @ensdomains/resolver@0.2.4: Please use @ensdomains/ens-contracts
npm WARN deprecated core-js@2.6.12: core-js@<3 is no longer maintained and not recommended for usage due to the number of issues. Please, upgrade your dependencies to the actual version of core-js@3.
added 2732 packages, and audited 2762 packages in 3m

217 packages are looking for funding
  run `npm fund` for details

84 vulnerabilities (2 low, 32 moderate, 34 high, 16 critical)

To address issues that do not require attention, run:
  npm audit fix

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

```

Fig. 3.3.9. truffle installation output

2. npx truffle migrate:

This command shown in Figure: 3.3.10. is used to migrate smart contracts to the Ethereum blockchain. Migrating contracts involves deploying them to a specified network (such as a local development network or a public testnet). Before running this command, ensure you have configured your Truffle project and specified the network settings in the `truffle-config.js` file.

```
C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client>npx truffle migrate

Compiling your contracts...
=====
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\SupplyChain.sol
> Artifacts written to C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client\src\artifacts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang
> Something went wrong while attempting to connect to the network at http://127.0.0.1:8545. Check your network configuration.
CONNECTION ERROR: Couldn't connect to node http://127.0.0.1:8545.
Truffle v5.11.5 (core: 5.11.5)
Node v16.17.0
npm notice
npm notice New major version of npm available! 9.2.0 -> 10.5.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.5.0
npm notice Run npm install -g npm@10.5.0 to update!
npm notice
```

Fig. 3.3.10. npx truffle migrate

3. npx truffle compile:

This command shown in Figure:3.3.11. compiles your smart contracts. Truffle automatically compiles contracts when needed (for example, before migration or testing), but you can explicitly compile them using this command.

```
C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client>npx truffle compile

Compiling your contracts...
=====
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\SupplyChain.sol
> Artifacts written to C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client\src\artifacts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang
```

Fig. 3.3.11. npx truffle compile

4. npm run start:

This command shown in Figure:3.3.12. is typically used to start a development server for a web application. If your project includes a front-end application (e.g., built with React, Vue.js, etc.) and you have configured a start script in your `package.json`, running `npm run start` will start the development server and allow you to view your application in a web browser.

```
C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client>npm run start

> client@0.1.0 start
> react-scripts start

(node:18800) [DEP0148] DeprecationWarning: Use of deprecated folder mapping "." in the "exports" field module resolution of the package at C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client\node_modules\postcss-safe-parser\node_modules\postcss\package.json.
Update this package.json to use a subpath pattern like "./*".
(Use 'node --trace-deprecation ...' to show where the warning was created)
i [wds]: Project is running at http://192.168.29.222/
i [wds]: webpack output is served from
i [wds]: Content not from webpack is served from C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client\public
i [wds]: 404s will fallback to /
Starting the development server...

Browserslist: caniuse-lite is outdated. Please run:
npx browserslist@latest --update-db

Why you should do it regularly:
https://github.com/browserslist/browserslist#browsers-data-updating
Compiled with warnings.

src\AssignRoles.js
  Line 84:19: Comparing to itself is potentially pointless      no-self-compare
  Line 102:9: 'redirect_to_home' is assigned a value but never used no-unused-vars

src\Homesec.js
  Line 28:9: 'backgroundImagePath' is assigned a value but never used no-unused-vars

src\Supply.js
  Line 18:10: 'MED' is assigned a value but never used      no-unused-vars
  Line 19:10: 'MedStage' is assigned a value but never used no-unused-vars
```

Fig. 3.3.12. npm run start

Before running these commands, ensure that you have initialized a Truffle project (`truffle init`), written your smart contracts, and configured your `truffle-config.js` file with the appropriate network settings. Additionally, make sure you have installed Node.js and npm on your system.

Libraries and Frameworks used in the project:

Web 3:

Web3.js, often referred to simply as Web3, is a JavaScript library designed to facilitate interaction between web applications and the Ethereum blockchain. As a crucial component of the Ethereum ecosystem, Web3.js acts as a bridge connecting decentralized applications (DApps) running on the Ethereum network with the frontend code written in JavaScript. Its primary purpose is to enable developers to build applications that can seamlessly interact with the Ethereum blockchain, accessing data, sending transactions, and executing smart contracts.

One of the central features of Web3.js is its ability to interact with Ethereum smart contracts. Smart contracts are self-executing contracts with predefined rules encoded on the Ethereum blockchain and Web3.js provides developers with methods to interact with these

contracts.

Developers can use Web3.js to deploy smart contracts, call their functions, send transaction to update contract state, and retrieve data stored within smart contracts.

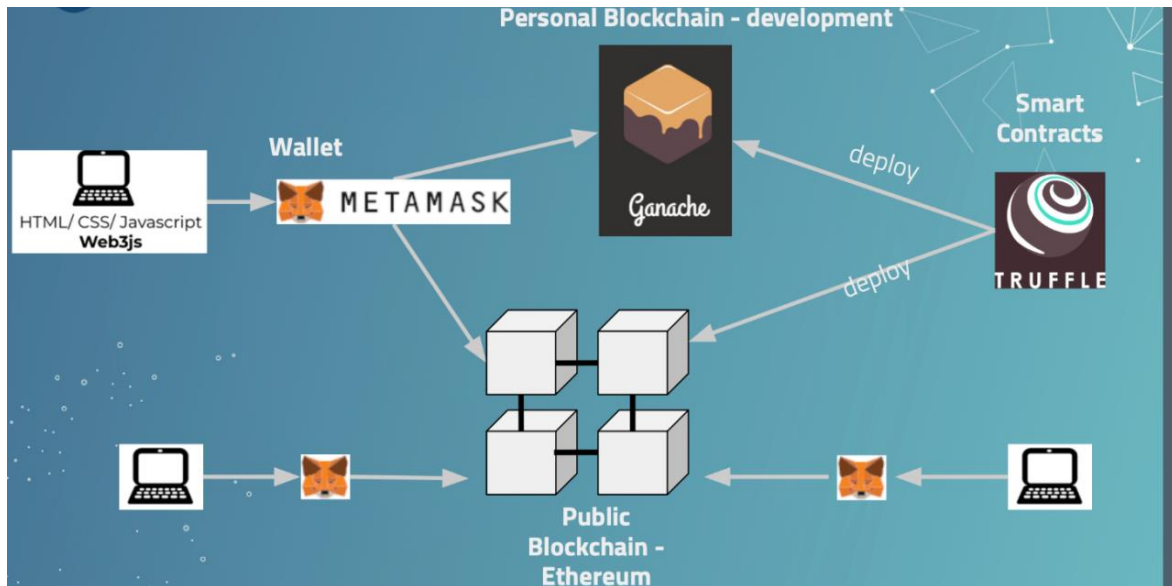


Fig. 3.3.13. Architecture of supply chain management

Additionally, Web3.js facilitates event listening, allowing developers to monitor and react to events emitted by smart contracts. This feature enables applications to respond dynamically to changes in the Ethereum blockchain, such as updates to contract state or the occurrence of specific conditions defined within smart contracts.

Web3.js also provides utilities for managing Ethereum accounts, including creating new accounts, importing existing accounts, and signing transactions with account private keys. This functionality is essential for handling account authentication and transaction signing securely within web applications.

Moreover, Web3.js supports interaction with different Ethereum networks, including the main Ethereum network (Mainnet), various test networks (e.g., Ropsten, Rinkeby, Kovan), and private Ethereum networks. This flexibility allows developers to test and deploy their applications on different networks according to their specific requirements.

Overall, Web3.js plays a vital role in the development of decentralized applications (DApps) by providing the necessary tools and functionalities for interacting with the

Ethereum blockchain. Its seamless integration with web applications, extensive features for smart contract interaction, and support for different Ethereum networks make it an indispensable tool for developers building applications on the Ethereum platform.

React:

Using React.js in a supply chain management system leveraging blockchain technology offers numerous advantages. React.js provides a robust framework for building the user interface (UI) of your application, allowing you to create a dynamic and intuitive front end that enhances user experience. With React.js's component-based architecture, you can easily develop modular UI components that efficiently represent various aspects of your supply chain management system, such as tracking shipments, managing inventory, and handling transactions.

Integrating React.js with blockchain technology enables seamless interaction between the front end and the underlying blockchain infrastructure. You can use React.js components to display relevant data from the blockchain, such as product information, transaction history, and supply chain events. Additionally, you can implement features like real-time updates and interactive visualization to provide users with insights into the status and progress of their supply chain operations.

Furthermore, React.js's flexibility and scalability make it well-suited for building complex applications like supply chain management systems. You can easily extend and customize your React.js components to meet the specific requirements of your project, whether it's integrating with different blockchain networks, implementing advanced analytics, or incorporating additional features based on user feedback.

Overall, by leveraging React.js in your supply chain management system using blockchain, you can create a modern and efficient application that streamlines operations, enhances transparency, and improves collaboration across the supply chain ecosystem.

Truffle:

Truffle is a development framework and asset pipeline for Ethereum-based decentralized applications (DApps) and smart contracts. It provides developers with a suite of tools and utilities to streamline the process of building, testing, and deploying smart contracts on the Ethereum blockchain. At its core, Truffle simplifies the development workflow by offering features such as automated contract compilation, testing, and deployment, along with built-in support for popular development languages like Solidity.

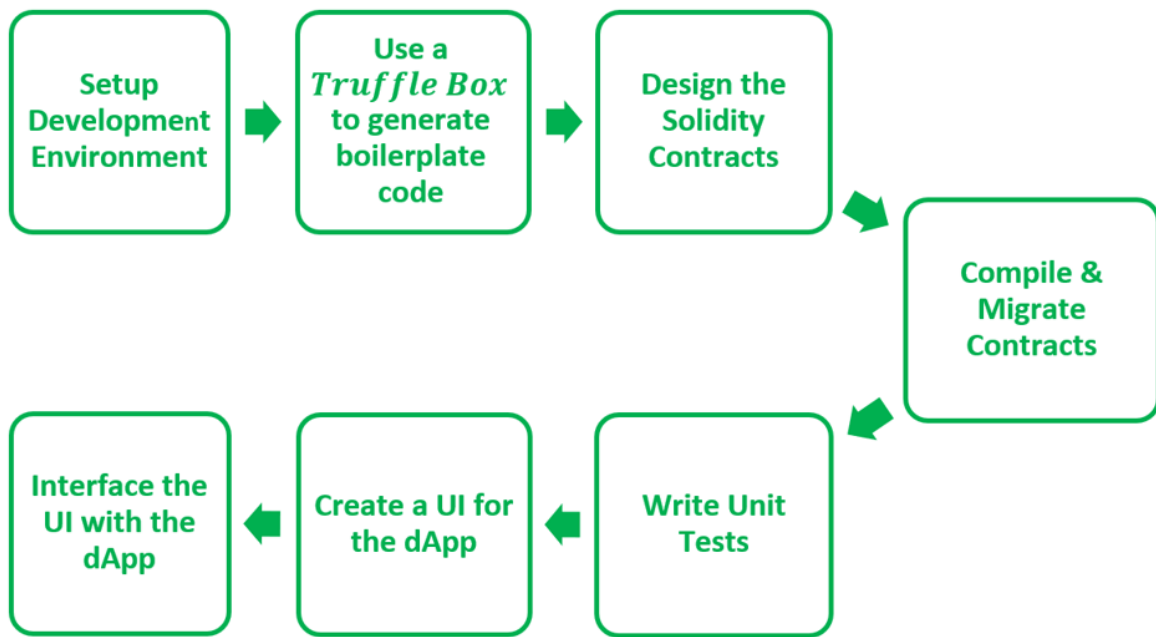


Fig. 3.3.14. Truffle Management Process

One of the key features of Truffle is its comprehensive development environment, which includes a command-line interface (CLI) for managing projects, compiling contracts, running tests, and deploying contracts to various Ethereum networks. Truffle's project structure and configuration files make it easy for developers to organize their code and dependencies, facilitating collaboration and code reuse across different projects.

Moreover, Truffle provides built-in support for smart contract testing using tools like Mocha and Chai, allowing developers to write automated tests to validate the functionality and behavior of their contracts. This testing framework enables developers to catch bugs and errors early in the development process, ensuring the reliability and security of their smart contracts before deployment [11].

Additionally, Truffle simplifies the process of deploying smart contracts to the Ethereum blockchain by providing deployment scripts and integration with Ethereum client software like Ganache and Geth. Developers can deploy contracts to various Ethereum networks, including the main Ethereum network (Mainnet), test networks (Ropsten, Rinkeby, Kovan), and private networks, with just a few simple commands.

Furthermore, Truffle offers a robust plugin system that allows developers to extend its functionality and integrate with other tools and services in the Ethereum ecosystem. This extensibility enables developers to customize their development workflow and integrate

additional features like code analysis, security auditing, and performance optimization seamlessly into their Truffle projects.

Overall, Truffle serves as an essential tool for Ethereum developers, providing a comprehensive and user-friendly framework for building, testing, and deploying smart contracts [11]. Its features, including automated compilation, testing, and deployment, along with support for various Ethereum networks and a vibrant plugin ecosystem, make it a preferred choice for developers looking to streamline their Ethereum development workflow and build robust and reliable decentralized applications.

Hashing Function:

In blockchain technology, hashing functions are essential cryptographic tools that ensure the integrity and security of the distributed ledger system. These functions employ mathematical algorithms to transform input data into fixed-size strings of characters, known as hash values or hash digests. The uniqueness of these hash values is paramount; even the smallest alteration in the input data produces an entirely different hash value. Hashing functions serve a pivotal role in blockchain by generating a unique identifier, or 'hash', for each block of data in the chain [12]. Each block typically comprises a list of transactions and a reference to the previous block's hash value. Through hashing, the data within each block is condensed into a hash value, effectively creating a digital fingerprint for that block. This fingerprint is crucial for maintaining data integrity within the block chain network, as any attempt to tamper with the data would result in a change in the hash value, alerting network participants to potential fraud or manipulation.

Furthermore, the use of cryptographic hashing algorithms, such as SHA-256, ensures that hash values are collision-resistant, meaning it is highly improbable for two different sets of data to produce the same hash value. This property reinforces the security of the block chain network, making it exceedingly difficult for malicious actors to compromise the integrity of the distributed ledger [12]. Overall, hashing functions are foundational to blockchain technology, providing a robust mechanism for verifying and securing transactions, maintaining an immutable record of data, and fostering trust among network participants.

The blockchain has a number of different uses for hash functions. Some of the most common uses of the hash function in blockchain are:

Merkle Tree: This uses hash functions to make sure that it is infeasible to find two Merkle trees with the same root hash. This helps to protect the integrity of the block header by

storing the root hash within the block header and thus protecting the integrity of the transactions.

Proof of Work Consensus: This algorithm defines a valid block as the one whose block header has a hash value less than the threshold value.

Digital signatures: Hash functions are the vital part of digital signatures that ensures data integrity and are used for authentication for blockchain transactions.

The chain of blocks: Each block header in a block in the blockchain contains the hash of the previous block header. This ensures that it is not possible to change even a single block in a blockchain without being detected.

As modifying one block requires generating new versions of every following block, thus increasing the difficulty.

Types of Cryptographic Hash Functions

There are several different classes of hash functions.

Some of the popular classes are:

1. **RACE Integrity Primitives Evaluation Message Digest (RIPEMD):** This set includes RIPEMD, RIPEMD-128, RIPEMD-160, RIPEMD-256, RIPEMD-320.

Out of these RIPEMD-160 is the most common.

The original RIPEMD-128 is based on the design principles used in MD4.

The RIPEMD-256 and 320 have fewer chances of the accidental collision but do not have higher levels of security as compared to RIPEMD-128 and RIPEMD-160.

2. **Message-Digest Algorithm:** This family comprises hash functions MD2, MD4, MD5, and MD6.

MD5 is the most widely used cryptographic hash function.

It is used to generate a 128-bit digest from a 512-bit string broken down into 16 words composed of 32 bits each [12].

Ronald Rivest designed this algorithm in 1991 to use for digital signature verification.

These are no longer considered cryptographically secure methods and should not be used for cryptographic authentication.

3. **BLAKE2:** It was announced on December 21, 2012. BLAKE2 is a cryptographic hash function based on BLAKE, designed with the aim to replace MD5 and SHA-1 algorithms in applications requiring high performance in software. It provides better security than SHA-2 and is similar to that of SHA-3. It provides the following features:

It provides immunity to length extensions.

It removes the addition of constants to message words.

It simplifies padding and reduces the number of rounds from 16 to 12.

4. **BLAKE3:** It was announced on January 9, 2020. BLAKE3 is a cryptographic function based on Bao and BLAKE2. It is a few times faster than BLAKE2. This algorithm provides many features like parallelism, XOF, KDF, etc.

5. **Whirlpool:** It is a cryptographic hash function, first described in 2000. It is a modified version of the Advanced Encryption Standard (AES). Whirlpool produces a hash of 512 bits.

6. **Secure Hashing Algorithm:** The family of SHA comprises four SHA algorithms: SHA-0, SHA-1, SHA-2, and SHA-3.

SHA-0 is a 160-bit hash function that was published by the National Institute of Standards and Technology in 1993.

SHA-1 was designed in 1995 to correct the weaknesses of SHA-0. In 2005, a method was found to uncover collisions in the SHA-1 algorithm due to which long-term employability became doubtful.

SHA-2 has the following SHA variants, SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256. It is a stronger hash function and it still follows the design of SHA-1.

In 2012, the Keccak algorithm was chosen as the new SHA-3 standard.

SHA-256 is the most famous of all cryptographic hash functions because it's used extensively in blockchain technology. The SHA-256 Hashing algorithm was developed by the National Security Agency (NSA) in 2001.

SHA 256:

SHA-256, or Secure Hash Algorithm 256-bit, stands as one of the most widely used cryptographic hashing functions, revered for its robustness and security in various applications, including blockchain technology. At its core, SHA-256 takes an input message of any size and produces a fixed-size output of 256 bits, regardless of the size or complexity of the input. This fixed-size output, known as a hash value or digest, serves as a unique digital fingerprint for the input data [13].

Secure Hashing Algorithm, or SHA. Data and certificates are hashed with SHA, a modified version of MD5. By using bitwise operations, modular additions, and compression

functions, a hashing algorithm reduces the input data into a smaller form that is impossible to comprehend [13]. Can hashing be cracked or decrypted, you may wonder? The main distinction between hashing and encryption is that hashing is one-way; once data has been hashed, the resultant hash digest cannot be decrypted unless a brute force assault is applied. See the illustration below to see how the SHA algorithm functions.

SHA is designed to provide a different hash even if only one character in the message changes. As an illustration, consider combining the themes Heaven and Heaven Is Different. The only difference between a capital and tiny letter, though, is size.

The strength of SHA-256 lies in its ability to generate a unique hash value for each input message, ensuring that even a minute change in the input data results in a drastically different hash value. This property, known as collision resistance, is fundamental in blockchain technology, where data integrity and security are paramount. In the context of blockchain, SHA-256 is utilized to create a unique identifier, or hash, for each block of data in the chain.

Every block in the blockchain contains a list of transactions and a reference to the previous block's hash value. By applying SHA-256 to the block's data, a unique hash value is generated, effectively creating a digital fingerprint for that block. This hash value serves as a cryptographic link to the previous block, forming an immutable chain of blocks. Any attempt to alter the data within a block would result in a change to its hash value, thus disrupting the integrity of the entire blockchain.

Furthermore, SHA-256's collision resistance ensures the security of the blockchain network, making it exceedingly difficult for adversaries to forge or manipulate transactions [13]. The robustness of SHA-256 has made it a cornerstone of blockchain technology, providing a reliable mechanism for verifying and securing transactions, maintaining an immutable record of data, and fostering trust among network participants.

Block chain:

Blockchain technology stands as a revolutionary innovation that has transformed various industries by providing a decentralized, secure, and transparent method for recording and verifying transactions. At its core, a blockchain is a distributed ledger that consists of a chain of blocks, with each block containing a list of transactions. These transactions are grouped together, cryptographically linked, and sequentially added to the blockchain, forming an immutable and tamper-proof record of data.

A simple analogy for how blockchain technology operates can be compared to how a Google Docs document works. When you create a Google Doc and share it with a group of people, the document is simply distributed instead of copied or transferred. This creates a decentralized distribution chain that gives everyone access to the base document at the same time. No one is locked out awaiting changes from another party, while all modifications to the document are being recorded in real-time, making changes completely transparent. A significant gap to note however is that unlike Google Docs, original content and data on the blockchain cannot be modified once written, adding to its level of security.



Fig. 3.3.15. Objectives of Blockchain

One of the defining features of blockchain is its decentralized nature, which eliminates the need for intermediaries such as banks or financial institutions to facilitate transactions. Instead, transactions are validated and recorded by a network of nodes, each maintaining a copy of the blockchain [14]. This decentralization ensures that no single entity has control over the entire network, promoting transparency and reducing the risk of censorship or fraud.

Furthermore, blockchain technology relies on cryptographic techniques, such as hashing and digital signatures, to secure the integrity and authenticity of transactions [14]. Each block in the blockchain contains a unique hash value that is generated based on the block's data, making it virtually impossible to alter the data within a block without affecting the entire chain. Additionally, digital signatures ensure that transactions are authenticated and cannot be forged.

Blockchain technology has a wide range of applications across various industries, including finance, supply chain management, healthcare, and real estate. In the financial sector, blockchain enables fast, secure, and cost-effective peer-to-peer transactions, as well as the issuance and trading of digital assets such as cryptocurrencies. In supply chain management, blockchain facilitates transparent and traceable tracking of goods from production to delivery, reducing fraud and ensuring product authenticity.

Moreover, blockchain technology has the potential to revolutionize the healthcare industry by securely storing and sharing patient data, streamlining processes such as medical record management and insurance claims processing. In the real estate sector, blockchain enables efficient and transparent property transactions, reducing the need for intermediaries and minimizing the risk of fraud.

Blockchain is a shared, immutable ledger that facilitates the process of recording transactions and tracking assets in a business network.

The first concept of blockchain dates back to 1991, when the idea of a cryptographically secured chain of records, or blocks, was introduced by Stuart Haber and Wakefield Scott Stornetta. Two decades later the technology gained traction and widespread use. The year 2008 marked a pivotal point for blockchain, as Satoshi Nakamoto gave the technology an established model and planned application [15]. The first blockchain and cryptocurrency officially launched in 2009, beginning the path of blockchain's impact across the tech sphere.

A blockchain network can track orders, payments, accounts, production and much more. And because members share a single view of the truth, you can see all details of a transaction end to end, giving you greater confidence, and new efficiencies and opportunities.

An *asset* can be tangible (a house, car, cash, land) or intangible (intellectual property, patents, copyrights, branding). Virtually anything of value can be tracked and traded on a blockchain network, reducing risk and cutting costs for all involved. Business runs on information [15]. The faster information is received and the more accurate it is, the better. Blockchain is ideal for delivering that information because it provides immediate, shared, and observable information that is stored on an immutable ledger that only permissioned network members can access.

Overall, blockchain technology holds immense promise for transforming various industries by providing a decentralized, secure, and transparent method for recording and verifying transactions. As the technology continues to evolve and mature, its potential to drive innovation and disrupt traditional business models is becoming increasingly apparent,

paving the way for a more decentralized and inclusive global economy.

In a blockchain every block has its own unique nonce and hash, but also references the hash of the previous block in the chain, so mining a block isn't easy, especially on large chains.

Miners use special software to solve the incredibly complex math problem of finding a nonce that generates an accepted hash. Because the nonce is only 32 bits and the hash is 256, there are roughly four billion possible nonce-hash combinations that must be mined before the right one is found. When that happens miners are said to have found the "golden nonce" and their block is added to the chain.

Making a change to any block earlier in the chain requires re-mining not just the block with the change, but all of the blocks that come after. This is why it's extremely difficult to manipulate blockchain technology. Think of it as "safety in math" since finding golden nonces requires an enormous amount of time and computing power.

When a block is successfully mined, the change is accepted by all of the nodes on the network and the miner is rewarded financially.

These are digital, programmed contracts that automatically enact or document relevant events when specific terms of agreement are met. Each contract is directly controlled through lines of code stored across a blockchain network. So once a contract is executed, agreement transactions become trackable and unchangeable. Though fundamental to the Ethereum platform, smart contracts can also be created and used on blockchain platforms like Bitcoin, Cardano, EOS.IO and Tezos.

Node:

In the context of blockchain technology, a node refers to a participant in the decentralized network that maintains a copy of the entire blockchain ledger and actively participates in the validation and propagation of transactions. Nodes play a crucial role in ensuring the integrity, security, and decentralization of the blockchain network by collectively verifying and recording transactions.

Each node in the blockchain network operates autonomously and independently, communicating with other nodes to reach a consensus on the state of the blockchain.

Consensus mechanisms, such as Proof of Work (PoW) or Proof of Stake (PoS), govern

how nodes agree on the validity of transactions and the order in which they are added to the blockchain. By participating in the consensus process, nodes collectively secure the network against malicious actors and ensure that only valid transactions are accepted and recorded.

Nodes in a blockchain network can be categorized into different types based on their roles and responsibilities. Full nodes, also known as network nodes, maintain a complete copy of the blockchain ledger and independently validate and propagate transactions to other nodes in the network. These nodes play a critical role in maintaining the integrity and decentralization of the blockchain network by ensuring that all transactions adhere to the network's consensus rules.

In addition to full nodes, there are also lightweight nodes, or SPV (Simplified Payment Verification) nodes, which do not maintain a complete copy of the blockchain ledger. Instead, lightweight nodes rely on full nodes to provide them with relevant transaction information and verify the validity of transactions using simplified verification methods. While lightweight nodes consume fewer resources and require less storage space compared to full nodes, they rely on the security and trustworthiness of the full nodes they are connected to.

There are different types of nodes in a Blockchain network, including full nodes, light nodes, and miner nodes.

1. Full nodes store a complete copy of the Blockchain ledger, while light nodes only store the necessary data to verify transactions.
2. Nodes communicate with each other through a peer-to-peer network, which allows them to exchange information and maintain consensus on the state of the Blockchain.
3. Node operators can earn rewards for their participation in the network, either through block rewards or transaction fees.
4. Node operators have an important role in maintaining the security of the network. They can help prevent attacks like double-spending by verifying transactions and rejecting any that are invalid.
5. Running a node requires technical knowledge and resources, such as computing power and storage space. However, there are many tools and services available that make it easier for users to set up and maintain their own nodes.
6. Blockchain nodes are a critical component of the Blockchain ecosystem, as they help to ensure the network's security and reliability [16].
7. As Blockchain technology continues to evolve and gain adoption, nodes will play an increasingly important role in the success of Blockchain-based applications and services.

Overall, nodes are fundamental building blocks of blockchain networks, responsible for maintaining the integrity, security, and decentralization of the network. By participating in the consensus process and validating transactions, nodes ensure the trustworthiness and reliability of the blockchain ledger, enabling secure and decentralized transactions without the need for central authorities or intermediaries.

As more individuals get interested in cryptocurrencies like Bitcoin, there is a greater need for them to understand how the system works [16]. Of course, this is true in any sector, but the uniqueness of cryptocurrency heightens its appeal. While you don't need to comprehend Blockchain to profit from an increase in Bitcoin's price in India, having a rudimentary understanding of the concepts that are bandied around might be beneficial.

4. System Design

4.1. Introduction:

System design for a project involves the process of defining the architecture, components, modules, interfaces, and data flow of a software system to meet the specified requirements. It is a crucial phase in the software development lifecycle where high-level requirements are translated into a detailed design that serves as a blueprint for implementation.

System design for a project involves the process of defining and organizing the components, modules, and interactions of a system to fulfill specific requirements effectively and efficiently. It encompasses various aspects such as architecture, data flow, functionality, scalability, reliability, and security. The design phase typically begins with understanding the project requirements, followed by breaking down the system into smaller components and defining their functionalities.

Design decisions are made considering factors like technology stack, database design, communication protocols, and third-party integrations. Throughout the process, emphasis is placed on ensuring the system's ability to scale, adapt to future changes, and maintain reliability and security. System design documentation often includes diagrams, flowcharts, and detailed descriptions to communicate the design decisions effectively to stakeholders and development teams.

Input Design:

In an information system, input is the raw data that is processed to produce output. During the input design, the developers must consider the input devices such as PC, MICR, OMR, etc.

Therefore, the quality of system input determines the quality of system output. Well-designed input forms and screens have following properties –

- It should serve a specific purpose effectively such as storing, recording, and retrieving the information.

- It ensures proper completion with accuracy.
- It should be easy to fill and straightforward.
- It should focus on the user's attention, consistency, and simplicity.
- All these objectives are obtained using the knowledge of basic design principles regarding
 - What are the inputs needed for the system?
 - How end users respond to different elements of forms and screens.

Objectives for Input Design:

The objectives of input design are –

- To design data entry and input procedures
- To reduce input volume
- To design source documents for data capture or devise other data capture methods
- To design input data records, data entry screens, user interface screens, etc.
- To use validation checks and develop effective input controls.

Output Design:

The design of output is the most important task of any system. During output design, developers identify the type of outputs needed and consider the necessary output controls and prototype report layouts.

Objectives of Output Design:

The objectives of input design are:

- To develop output design that serves the intended purpose and eliminates the

production of unwanted output.

- To develop the output design that meets the end user's requirements.
- To deliver the appropriate quantity of output.
- To form the output in appropriate format and direct it to the right person.
- To make the output available on time for making good decision.

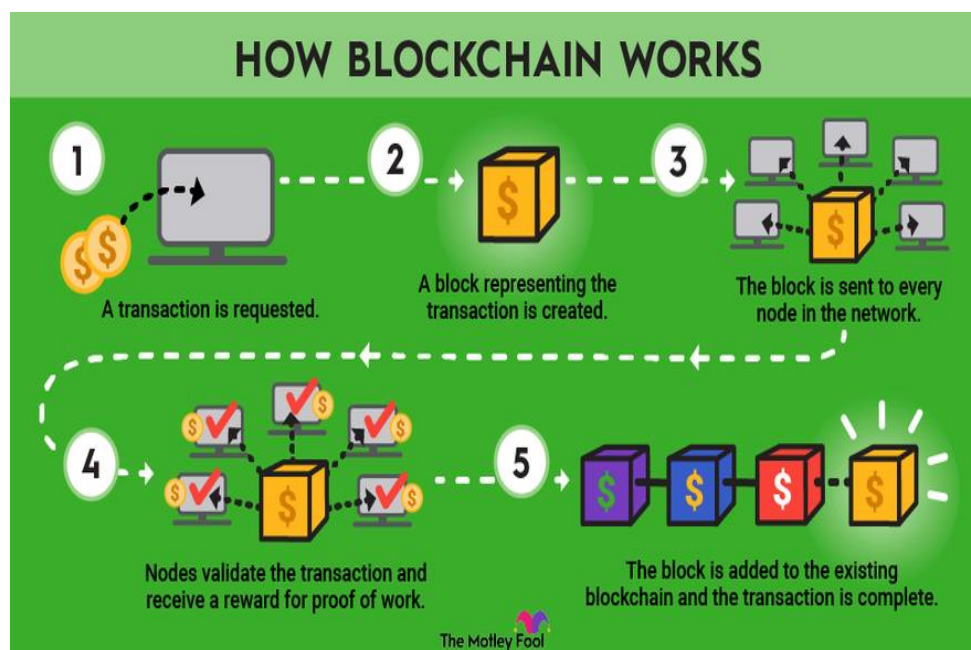


Fig. 4.1.1. How Blockchain works

4.2. Data Flow Diagrams:

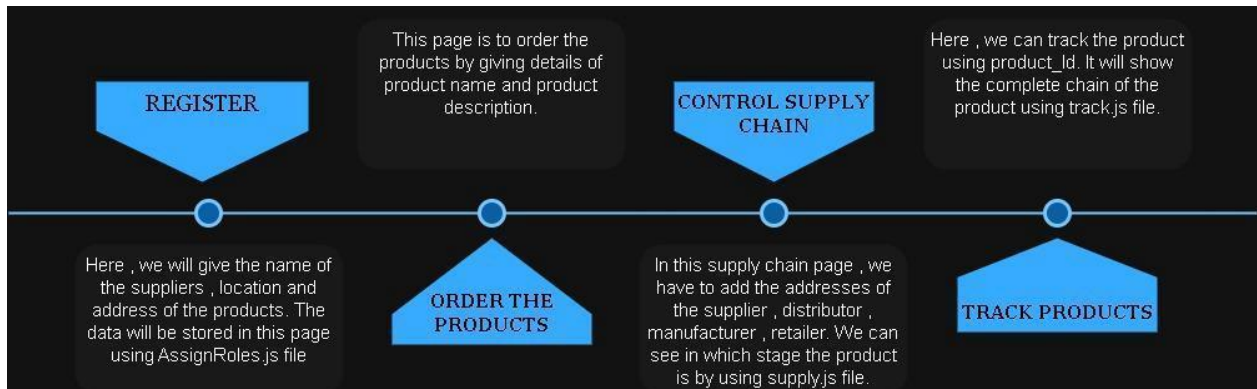


Fig. 4.2.1. Data Flow Diagram

UML Diagrams:

UML stands for Unified Modelling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML comprises two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

Use case Diagram:

- ▶ A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis.
- ▶ The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

Key Components of a Use Case Diagram:

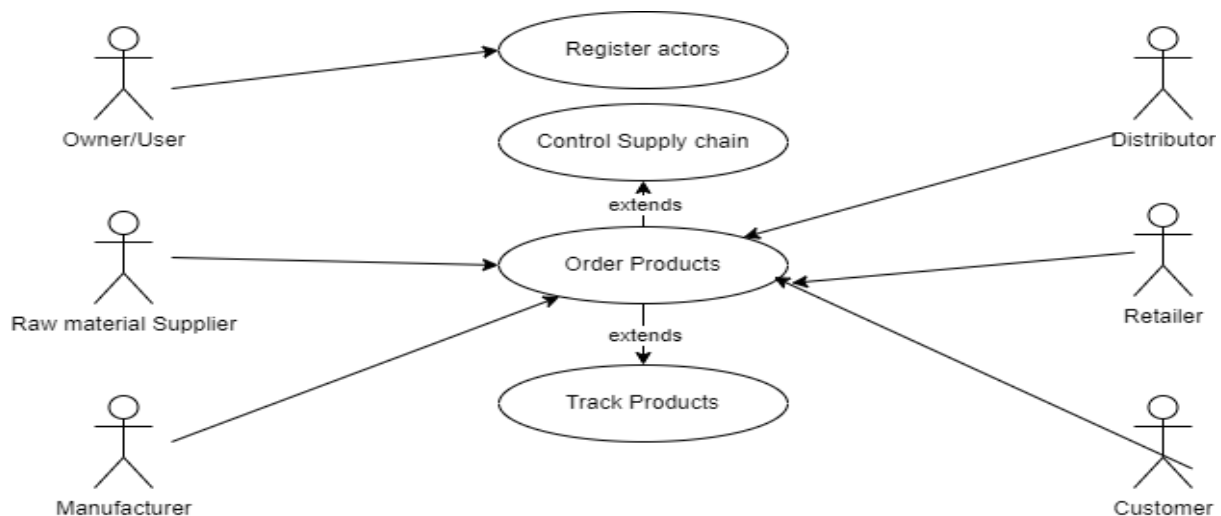


Fig. 4.2.2. Use Case Diagram

Actors: Actors represent the external entities (users or systems) that interact with the system. Each actor has specific roles or responsibilities within the system and initiates one or more use cases.

Use Cases: Use cases represent the specific tasks or functionalities that the system provides to its users. They describe the interactions between actors and the system to achieve a particular goal or objective.

Relationships: Relationships between actors and use cases depict the interactions or associations between them. These relationships can include associations, generalizations, and dependencies, which illustrate how actors and use cases are related to each other.

The use case diagram for the Supply Chain Management System using blockchain illustrates the interactions between actors and the system functionalities. It provides a clear and visual representation of the tasks performed by actors within the system, helping to understand the system's behavior and requirements from a high-level perspective.

Class Diagram:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, attributes, operations (or methods), and the relationships among the classes. It explains which class contains information

Key Components of a Class Diagram:

Classes: Classes represent the building blocks of the system and encapsulate the data and behavior associated with a specific entity or concept. Each class is depicted as a rectangle with three compartments: the class name at the top, attributes in the middle, and operations at the bottom.

Attributes: Attributes are the properties or characteristics of a class that describe its state. They represent the data members or variables associated with the class and are typically shown in the middle compartment of the class rectangle.

Operations (Methods): Operations define the behavior or functionality of a class and represent

the actions that the class can perform. They encapsulate the algorithms or procedures that manipulate the class's data and are depicted in the bottom compartment of the class rectangle.

Relationships: Relationships between classes depict the associations, dependencies, and interactions among them. There are various types of relationships, including associations, generalizations (inheritance), aggregations, and compositions, which illustrate how classes are connected to each other.

The class diagram for the Supply Chain Management System using blockchain provides a visual representation of the system's structure, illustrating the classes, their attributes, operations, and relationships. It helps in understanding the organization of the system and the interactions among its components, aiding in system design, implementation, and maintenance.

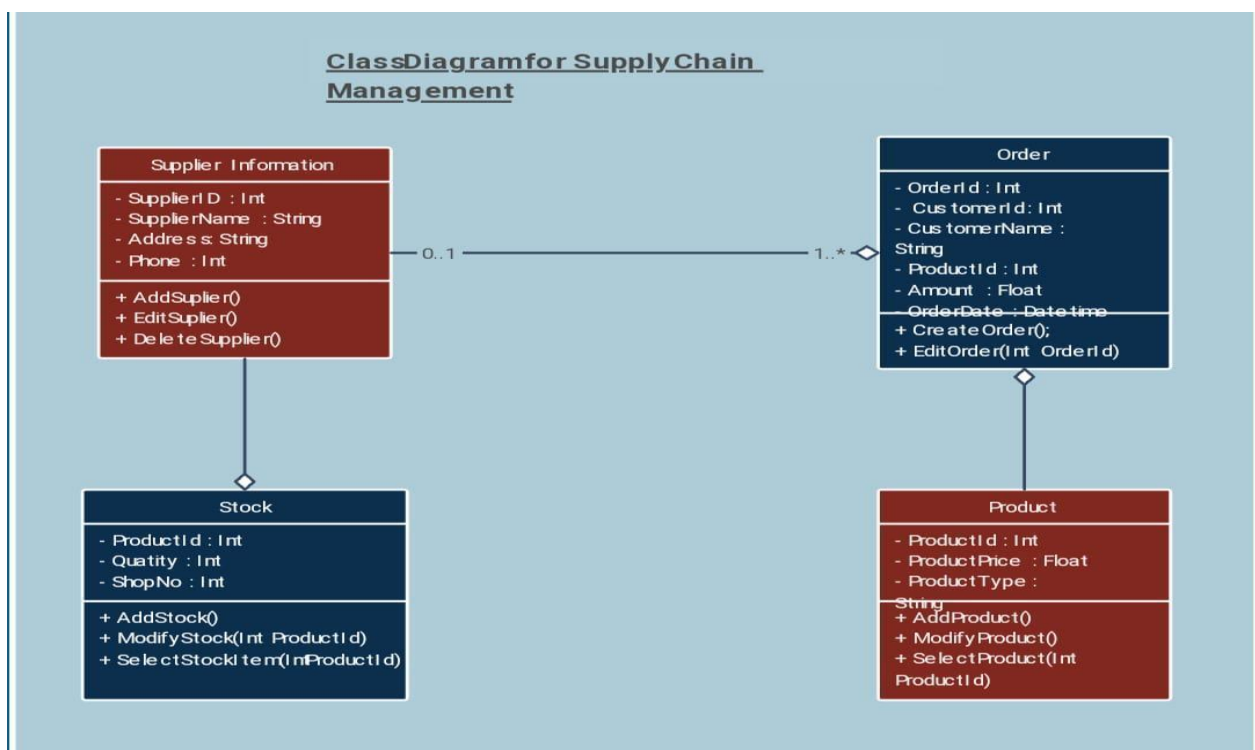


Fig. 4.2.3. Class Diagram

Activation Boxes: Activation boxes (also known as activation bars or execution occurrences) represent the duration of time during which an object is actively processing a message. They show when an object is executing an operation and provide a visual indication of the sequence Of operations performed by the objects.

Collaboration Diagram:

In the collaboration diagram, the method call sequence is indicated by some numbering technique as shown below. The number indicates how the methods are called one after another. We have taken the same order management system to describe the collaboration diagram. The method calls are similar to that of a sequence diagram. But the difference is that the sequence diagram does not describe the object organization whereas the collaboration diagram shows the object organization.

Key Components of a Collaboration Diagram:

Objects: Objects represent the instances of classes or entities participating in the collaboration. Each object is depicted as a named box or rectangle, representing its identity and role in the system.

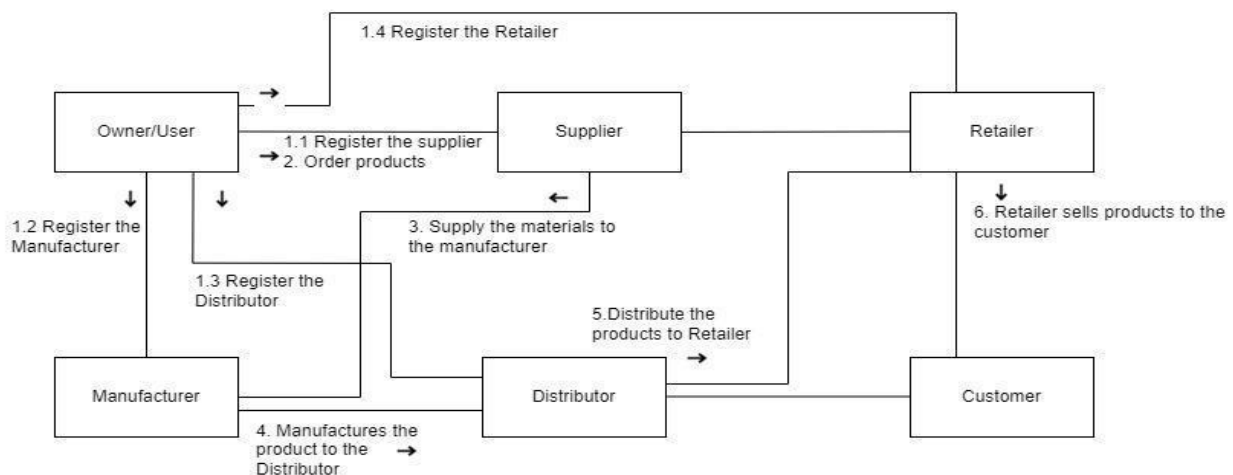


Fig. 4.2.5. Collaboration Diagram

Messages: Messages represent the interactions or communications between objects in the collaboration diagram. They indicate the flow of control or data between objects and are depicted

as labeled arrows connecting the objects. Messages may include method calls, requests, or responses exchanged between objects.

Method Call Sequence: The method call sequence in a collaboration diagram is indicated by

numbering techniques, showing the order in which methods are called or executed by the objects. The numbers next to the messages indicate the sequence of method calls, illustrating how objects collaborate to achieve the desired functionality.

Deployment Diagram:

The deployment diagram represents the deployment view of a system. It is related to the component diagram. Because the components are deployed using the deployment diagrams. A deployment diagram consists of nodes. Nodes are nothing but physical hardware used to deploy the application

Key Components of a Deployment Diagram:

Nodes: Nodes represent the physical hardware devices or execution environments where software components are deployed. Each node typically corresponds to a physical server, computer, or device in the network. Nodes are depicted as rectangular boxes, often labeled with the name of the device or its role in the deployment architecture.

Components: Components represent the software modules, applications, or services that are deployed onto the hardware nodes. Components encapsulate the functionality and behavior of the system and interact with each other to perform specific tasks. Components are depicted as rounded rectangles connected to the nodes where they are deployed.

Relationships: Relationships in a deployment diagram illustrate the associations and dependencies between nodes and components. These relationships indicate how components are deployed onto nodes and how they interact with each other across the deployment architecture.

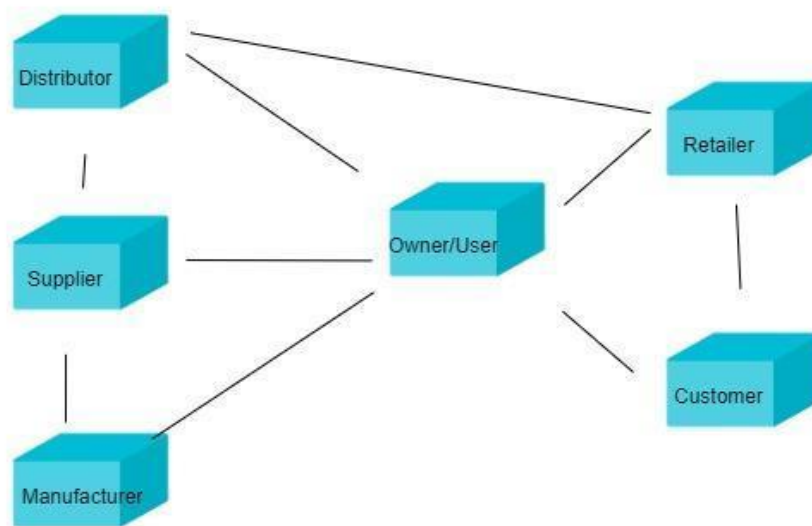


Fig. 4.2.6. Deployment Diagram

Activity Diagram:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration, and concurrency.

In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

Key Components of an Activity Diagram:

Initial Node: The initial node represents the starting point of the activity diagram, indicating where the process begins. It is typically depicted as a solid circle or ellipse.

Activities: Activities represent the tasks or actions that are performed within the process. They can be simple actions, such as calculations or data processing, or more complex procedures involving multiple steps. Activities are depicted as rounded rectangles with text inside describing the action.

Decisions: Decisions, also known as branches or choices, represent points in the process where different paths or outcomes are possible based on certain conditions or criteria. They are depicted as diamonds, with lines representing different paths leading from them based on the outcome of the decision.

Merge Nodes: Merge nodes are used to combine multiple paths back into a single path after diverging at a decision point. They are depicted as diamond shapes with multiple incoming and outgoing flows.

Fork and Join Nodes: Fork nodes split the flow of control into multiple concurrent paths, allowing activities to be performed in parallel. Join nodes synchronize the flow of control, combining multiple concurrent paths back into a single path.

Final Node: The final node represents the endpoint of the activity diagram, indicating where the process concludes. It is typically depicted as a solid circle or ellipse with a border.

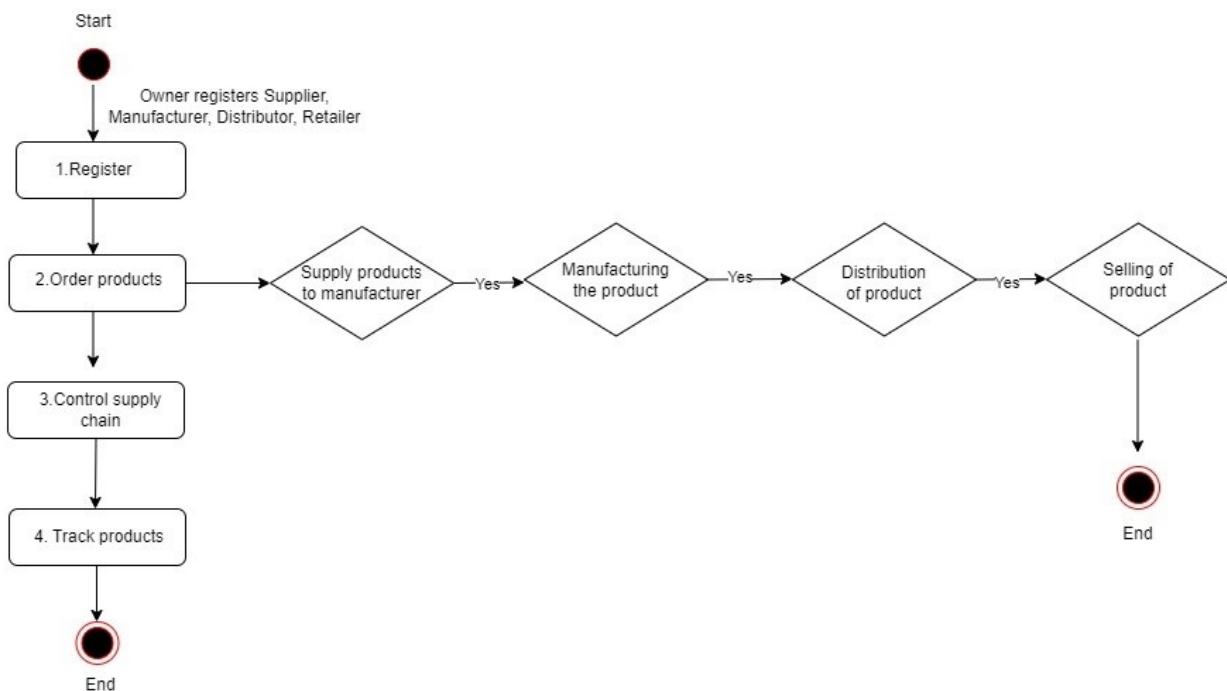


Fig. 4.2.7. Activity Diagram

Component Diagram:

A component diagram, also known as a UML component diagram, describes the organization and wiring of the physical components in a system. Component diagrams are often drawn to help model implementation details and double-check that every aspect of the system's required function is covered by planned development.

Key Components of a Component Diagram:

Components: Components represent the physical or modular units of the system, encapsulating functionality and behavior. They can be software modules, libraries, executables, hardware devices, or other tangible entities. Each component is depicted as a rectangle with the name of the component written inside.

Interfaces: Interfaces define the interactions or contracts between components, specifying the methods, operations, or services that a component offers or requires. They represent the points of connection between components and are depicted as small squares on the boundaries of components, labeled with the interface name.

Dependencies: Dependencies represent the relationships between components, indicating how one component relies on or is related to another component. Dependencies can be directed or undirected and may include associations, dependencies, and dependencies. They are depicted as dashed lines with arrows pointing from the dependent component to the component it depends on.

Ports: Ports represent specific connection points within a component where interactions occur. They provide a mechanism for components to communicate with each other and are often associated with interfaces. Ports are depicted as small squares or circles on the boundaries of components, typically connected to interfaces.

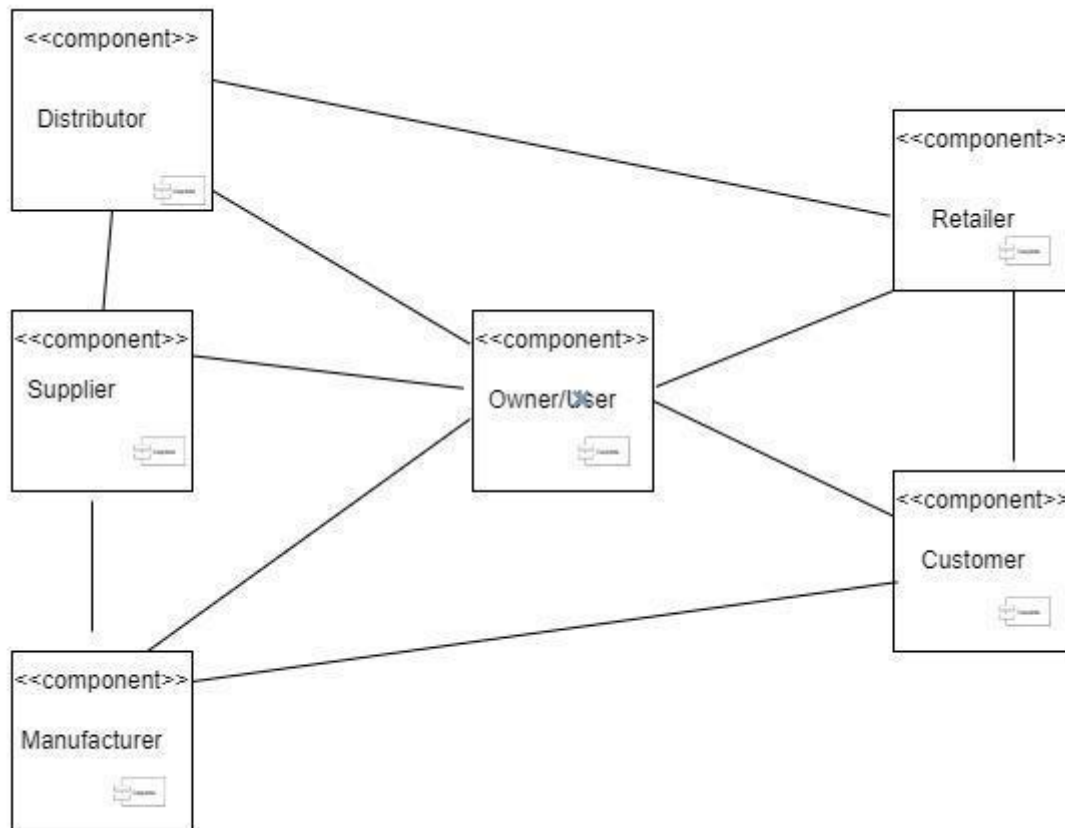


Fig. 4.2.8. Component Diagram

4.3. Database Design:

Database design for a project involves structuring and organizing data in a way that efficiently stores, retrieves, and manages information. Designing a database for a Supply Chain Management System using blockchain involves defining the structure and relationships of the data entities involved in the system. Since blockchain technology is utilized, the database design will primarily focus on storing and managing the necessary data related to product ids and addresses of the actors involved in the project.

4.3.1.ER DIAGRAM:

An Entity-relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as an Entity Relationship Diagram (ER Diagram). An ER model is

a design or blueprint of a database that can later be implemented as a database. The main components of the E-R model are the entity set and relationship set.

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in a database, so by showing the relationship among tables and their attributes, the ER diagram shows the complete logical structure of a database. Let's have a look at a simple ER diagram to understand this concept.

Main Components of an ER Model:

Entity Set: An entity set represents a collection of similar entities, where each entity is a distinct object or concept within the system. Entities can be tangible (e.g., a person, a book) or intangible (e.g., an event, a transaction).

Relationship Set: A relationship set defines the associations and interactions between entity sets. It represents the connections or associations between entities, indicating how they are related to each other.

ER Diagram Structure:

An ER diagram visually represents the structure of a database, showcasing the entities, attributes, and relationships involved. Here's an overview of the components of an ER diagram:

Entities: Entities are represented as rectangles with rounded corners, labeled with their names. Each entity typically corresponds to a table in the database schema, representing a collection of related data.

Attributes: Attributes are the properties or characteristics of entities, defining the data elements associated with each entity. They are depicted as ovals connected to their respective entities, indicating their relationship. Attributes provide detailed information about the entities.

Relationships: Relationships between entities are represented by lines connecting the related entities. The lines are labeled with the type of relationship (e.g., one-to-one, one-to-many, many-to-many) and indicate how entities are connected or associated with each other.

An ER diagram serves as a visual representation of the logical structure of a database,

showcasing the entities, attributes, and relationships involved in the system. It helps in understanding the database schema, designing the database effectively, and communicating the database structure to stakeholders. The ER diagram for the Supply Chain Management System using blockchain provides insights into how actors(owner, supplier, manufacturer, retailer, distributor, customer) are related to each other within the system.

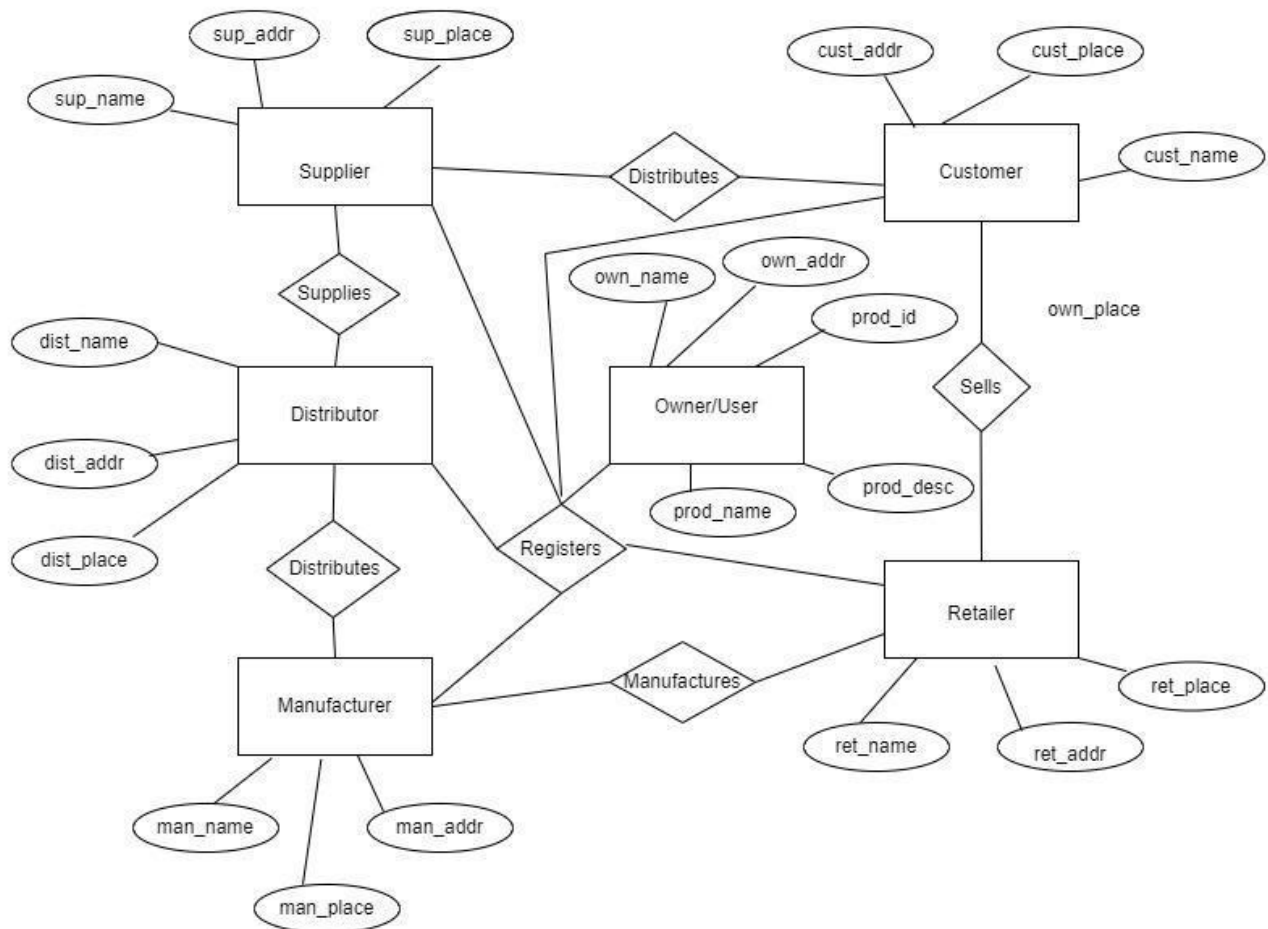


Fig. 4.3.1.1. ER Diagram

5.System Implementation

5.1. Introduction:

A supply chain consists of all parties involved, directly or indirectly, in fulfilling a customer request. The supply chain includes not only the manufacturer and suppliers, but also transporters, warehouses, retailers, and even customers themselves. Within each organization, such as a manufacturer, the supply chain includes all functions involved in receiving and filling a customer request. These functions include, but are not limited to, new product development, marketing, operations, distribution, finance, and customer service. Through the implementation of blockchain, our project seeks to address critical pain points such as counterfeit prevention, real-time tracking, and streamlined transactions. By immutably recording every transaction and transfer of ownership on the blockchain, participants can achieve unprecedented levels of transparency and trust, thereby reducing fraud, errors, and delays.

Moreover, blockchain enables smart contracts and programmable agreements that automatically execute and enforce terms when predefined conditions are met. This feature empowers supply chain stakeholders to automate processes such as payments, compliance verification, and contract management, leading to significant cost savings and operational efficiencies. There are other additional methods for achieving supply chain management. Transactions made via techniques other than blockchain are not safe, nor are the systems that use them secure.

Thus, we can conclude that supply chain management using blockchain technology may be the optimal procedure for agreements and transactions that provide the necessary advantages for the client. People are becoming interested in projects that use new technology these days. The blockchain method for supply chain management also requires the use of numerous other technologies, such as Ganache, Metamask, Visual Studio, etc.

Ganache plays a pivotal role in the development and testing phases of our blockchain-based supply chain management project. Its multifaceted capabilities contribute significantly to enhancing the security, efficiency, and robustness of our system. Ganache serves as a comprehensive development environment tailored specifically for blockchain projects. By simulating various network scenarios and testing different use cases, developers can validate

the functionality and reliability of smart contracts and decentralized applications (DApps) within a controlled environment. This iterative testing process ensures that the project meets its functional requirements and performs optimally under different conditions before deployment.

Overall, Ganache empowers our team to build, test, and deploy a robust and secure supply chain management solution powered by blockchain technology. Its versatile features and seamless integration with other development tools make it an indispensable asset in our quest to revolutionize supply chain management practices. This tool is also used for debugging to check the created test cases, one of the key components of this tool is to implement advanced encryption algorithms to ensure the confidentiality and integrity of stored data. Our project contains a separate truffle config file, which is mainly used for the ganache. Based on this file, the ganache will provide the addresses and keys for the people who are involved in developing the product or goods. So, this is the main tool for the application.

MetaMask is an extension of the blockchain wallet that manages transactions and contracts. Supply chain managers depend on MetaMask because it makes it possible for consumers to interact with blockchain applications, especially those created on the Ethereum network. We must include the metamask addon for Chrome and other search engines. In the metamask, we must construct an Ethereum project.

To link the project, the addresses and private keys generated in the ganache are added to the metamask. Transactions between the various sources will take place based on the private keys that have been added. After going through all of these phases, the items or products will finally reach the buyer. We have access to more tools that function similarly to these. Since these contracts and transactions may only be accessed by those who are utilizing the project, they will be safe and secure. They can't be changed or altered. For verification, every transaction is kept in a database or the cloud.

Popular Ethereum wallet Metamask functions as a browser plugin and offers users a safe and practical interface to manage their Ethereum funds and engage with decentralized apps. MetaMask makes it simple to incorporate Ethereum-based smart contracts into supply chain processes. Intelligent contracts have the potential to automate several supply chain activities, including order fulfillment, record-keeping, and payment settlement. MetaMask acts as a bridge, enabling users to interact with these smart contracts through a user-friendly interface.

This guarantees the transparency and immutability of the recorded data on the Ethereum blockchain in addition to improving supply chain activities' efficiency.

Solidity is an object-oriented programming language that may be used to create EVM-compatible programs like smart contracts. It is a brand-new programming language that combines web development, assembly language, and networking conventions. One of the most well-known blockchain platforms is Ethereum, and Solidity is a high-level programming language designed specifically for use in using these networks to create smart contracts. Supply chain management depends on solidity because it allows smart contracts to automate and execute business logic. As self-executing agreements that enact terms directly into code, smart contracts promote transparency, immutability, and confidence among supply chain actors.

Solidity makes it possible to integrate supply chain elements like transparency and traceability. Every event and transaction in the supply chain may be transparently and irrevocably recorded using the blockchain, making it accessible to all stakeholders with the appropriate authorization. This openness helps track the origin, movement, and status of products by increasing accountability and reducing the possibility of fraud or counterfeiting. All things considered, Solidity is an effective tool for raising the effectiveness and accountability of supply chain management on blockchain platforms because of its capacity to build reliable, decentralized, and automated smart contracts. Industries, factories, railroads, agriculture, and other sectors can all use this blockchain-based supply chain management system.

Supply chain management (SCM) involves the coordination and integration of various activities within the supply chain, including procurement, production, inventory management, logistics, and distribution, to ensure the smooth flow of goods and services from the point of origin to the point of consumption. Blockchain technology has gained attention as a potential solution to enhance transparency, traceability, security, and efficiency in supply chain management.

Here's how a supply chain management system using blockchain can include the owner, supplier, manufacturer, distributor, retailer, and consumer:

Owner/Brand/User: The owner or brand initiates the supply chain process by sourcing raw materials or finished goods from suppliers. They can use blockchain to record the origin and characteristics of the materials, ensuring transparency and traceability throughout the supply chain.

Supplier: Suppliers provide raw materials or components to the manufacturer. By leveraging blockchain, suppliers can record the details of their products, including quality certificates, production processes, and shipment information. This data is securely stored on the blockchain, providing an immutable record of the product's journey.

Manufacturer: Manufacturers transform raw materials or components into finished products. Through blockchain, manufacturers can track the production process, monitor quality control measures, and record product specifications. Smart contracts can automate payment processes based on predefined conditions, streamlining transactions between manufacturers and suppliers.

Distributor: Distributors are responsible for transporting products from manufacturers to retailers or directly to consumers. Blockchain enables distributors to track the movement of goods in real-time, ensuring visibility into the supply chain's logistics. Smart contracts can automate inventory management and replenishment, triggering orders when stock levels reach predefined thresholds.

Retailer: Retailers sell products to end consumers through physical stores or online platforms. With blockchain, retailers can verify the authenticity and origin of products, reducing the risk of counterfeit goods entering the supply chain. Additionally, blockchain-based loyalty programs can incentivize consumer engagement and reward loyalty points transparently.

Consumer: Consumers are the final recipients of products in the supply chain. Blockchain empowers consumers to access detailed information about the products they purchase, such as origin, production methods, and sustainability practices. By scanning QR codes or accessing blockchain-based platforms, consumers can make informed purchasing decisions and trust the authenticity of the products they buy.

Overall, a supply chain management system using blockchain enhances transparency, traceability, and trust among stakeholders, from owners and suppliers to manufacturers, distributors, retailers, and consumers. By leveraging blockchain technology, organizations can streamline operations, mitigate risks, and deliver value-added services throughout the supply chain ecosystem.

5.2. Project Modules

- Owner Module:

The Owner Module serves as the central control hub within our blockchain-based supply chain management system, empowering the owner with comprehensive oversight and management capabilities over the entire supply chain ecosystem.

The owner has the authority to register various stakeholders involved in the supply chain, including retailers, manufacturers, raw material suppliers, distributors, and any other relevant parties. Through this registration process, the owner establishes a trusted network of participants within the blockchain system.

The owner possesses the privilege to monitor and modify the details of products stored within the blockchain. This includes essential information such as product specifications, production dates, batch numbers, quality certifications, and any other relevant attributes. By maintaining accurate and up-to-date product details, the owner ensures transparency and traceability throughout the supply chain.

As the primary decision-maker within the system, the owner has the exclusive authority to place orders for products or raw materials. Once an order is placed, it triggers a series of events within the supply chain, prompting manufacturers, suppliers, and distributors to initiate the necessary processes to fulfill the order requirements. The owner is equipped with robust tracking capabilities that enable real-time monitoring of products as they progress through various stages of the supply chain. Leveraging the immutable nature of blockchain technology, the owner can trace the journey of each product from its origin to its final destination, ensuring visibility and accountability at every step.

- Raw Materials Supplier Module:

The Raw Material Supplier Module is designed to streamline the interaction between raw material suppliers and the broader supply chain network, offering a range of functionalities and features to enhance efficiency, transparency, and collaboration.

The module facilitates a seamless registration process for raw material suppliers, allowing them to be added by the owner or administrator of the supply chain system. Through a user-friendly interface, suppliers can submit necessary information and credentials, enabling quick onboarding into the system. Upon successful registration, raw material suppliers gain access to the platform's order management system. Here, they can view and process orders placed by the owner or other stakeholders within the supply chain network. Using unique identifiers

associated with each order, suppliers can accurately identify the required materials, quantities, and delivery specifications.

Through immutable and transparent transaction records, suppliers can track the status of each order, verify delivery milestones, and address any discrepancies or delays in transit. The module fosters seamless communication and collaboration between raw material suppliers and other stakeholders within the supply chain ecosystem.

Integrated messaging systems and notification features enable suppliers to stay informed about order updates, communicate with customers or partners, and address any queries or concerns on time. Raw material suppliers can utilize the module to maintain stringent quality control measures throughout the supply chain process. By recording quality assurance data and certifications on the blockchain ledger, suppliers can ensure compliance with regulatory standards, mitigate quality-related risks, and uphold the reputation of their products and services.

- **Manufacturer Module:**

The Manufacturer Module constitutes a pivotal element within our supply chain management framework, offering manufacturers a comprehensive suite of tools and functionalities to optimize production processes and enhance collaboration within the supply chain network.

Manufacturers can be seamlessly onboarded into the system by the owner or administrator. Through a streamlined registration process, manufacturers submit relevant information and credentials, gaining access to the platform's features and functionalities. Using unique identifiers associated with each supplier or material, manufacturers can efficiently source the required materials and quantities, ensuring timely availability for production processes. The module facilitates efficient production management by providing manufacturers with tools to schedule, monitor, and optimize manufacturing processes. Manufacturers can track the progress of production orders, allocate resources effectively, and manage production workflows to meet demand and delivery deadlines.

Through blockchain technology, manufacturers gain real-time visibility into the movement and status of products throughout the production lifecycle. By recording production milestones, inventory movements, and product transfers on the blockchain ledger, manufacturers can track the provenance of each product, verify authenticity, and address any issues or discrepancies in transit. The module fosters seamless communication and collaboration between manufacturers

and other stakeholders within the supply chain ecosystem. Integrated messaging systems and collaboration tools enable manufacturers to communicate with suppliers, distributors, and customers, facilitating coordination, resolving issues, and addressing inquiries in a timely manner.

- Distributor Module:

The Distributor Module is a crucial component within our supply chain management system, facilitating efficient distribution processes and fostering seamless collaboration between distributors and other stakeholders.

Distributors are onboarded into the system by the owner or administrator through a streamlined registration process. Once added, distributors gain access to the platform's features and functionalities, enabling them to participate in the distribution network. Using unique identifiers associated with each product or order, distributors procure products from registered manufacturers within the supply chain network. Leveraging the module's capabilities, distributors can efficiently obtain the required products and quantities, ensuring timely availability for distribution.

Distributors can track incoming and outgoing product shipments, monitor stock levels, and optimize inventory storage and distribution processes to meet customer demand and minimize stockouts or overstock situations. Distributors play a key role in fulfilling customer orders by supplying products to retailers within the supply chain network. Using order details and product identifiers, distributors ensure accurate and timely delivery of products to retailers, facilitating seamless transactions and customer satisfaction. Through blockchain technology, distributors gain visibility into the movement and status of products throughout the distribution process. By recording product transfers, delivery milestones, and inventory movements on the blockchain ledger, distributors can track the provenance of each product, verify authenticity, and address any issues or discrepancies in transit.

The module fosters communication and collaboration between distributors and other stakeholders within the supply chain ecosystem. Integrated messaging systems and collaboration tools enable distributors to communicate with manufacturers, retailers, and customers, facilitating coordination, resolving issues, and addressing inquiries promptly.

- Retailer Module :

The Retailer Module is an essential component within our supply chain management system, empowering retailers with the tools and capabilities needed to efficiently manage sales operations and deliver exceptional customer experiences.

Retailers are onboarded into the system by the owner or administrator through a seamless registration process. Once added, retailers gain access to the platform's features and functionalities, enabling them to participate in the retail network. Using unique identifiers associated with each product or order, retailers procure products from registered distributors within the supply chain network. Leveraging the module's capabilities, retailers can efficiently obtain the required products and quantities, ensuring adequate stock levels to meet customer demand. The module provides tools for retailers to manage their inventory effectively. Retailers can track incoming product shipments, monitor stock levels, and optimize inventory storage and replenishment processes to ensure product availability and minimize stock outs or overstock situations.

Retailers utilize the module to facilitate point-of-sale operations and conduct transactions with customers. The module offers (Customer Relationship Management) CRM features that enable retailers to manage customer relationships and enhance customer satisfaction. Retailers can capture customer information, track purchase histories, and personalize marketing efforts to engage customers effectively and drive repeat business. Through blockchain technology, retailers gain visibility into the origin and authenticity of products within their inventory. By accessing product information recorded on the blockchain ledger, retailers can verify product provenance, ensure product quality, and address any concerns or inquiries from customers.

5.3. Screens:

- **Execution Steps**

```
C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client>npx truffle migrate

Compiling your contracts...
=====
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\SupplyChain.sol
> Artifacts written to C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client\src\artifacts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang
> Something went wrong while attempting to connect to the network at http://127.0.0.1:8545. Check your network configuration.
CONNECTION ERROR: Couldn't connect to node http://127.0.0.1:8545.
Truffle v5.11.5 (core: 5.11.5)
Node v16.17.0
npm notice
npm notice New major version of npm available! 9.2.0 -> 10.5.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.5.0
npm notice Run npm install -g npm@10.5.0 to update!
npm notice
```

Fig. 5.3.1. npx truffle migrate command output

First, we have to go to the new terminal and open the command prompt and copy the path where scripts are present and paste in the command prompt as shown in the Figure: 5.3.1..

```
C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client>npx truffle compile

Compiling your contracts...
=====
> Compiling .\contracts\Migrations.sol
> Compiling .\contracts\SupplyChain.sol
> Artifacts written to C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client\src\artifacts
> Compiled successfully using:
   - solc: 0.5.16+commit.9c3226ce.Emscripten.clang
```

Fig. 5.3.2. npx truffle compile command output

These are the parameters that are stored in the blockchain when truffle migration starts.

```

C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client>npm run start

> client@0.1.0 start
> react-scripts start

(node:18800) [DEP0148] DeprecationWarning: Use of deprecated folder mapping "." in the "exports" field module resolution of the package at C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client\node_modules\postcss-safe-parser\node_modules\postcss\package.json.
Update this package.json to use a subpath pattern like ".*/*".
(Use 'node --trace-deprecation ...' to show where the warning was created)
[wds]: Project is running at http://192.168.29.222/
[wds]: webpack output is served from
[wds]: Content not from webpack is served from C:\Users\yelet\OneDrive\Desktop\Blockchain-SCM\client\public
[wds]: 404s will fallback to /
Starting the development server...

Browserslist: caniuse-lite is outdated. Please run:
npx browserslist@latest --update-db

Why you should do it regularly:
https://github.com/browserslist/browserslist#browsers-data-updating
Compiled with warnings.

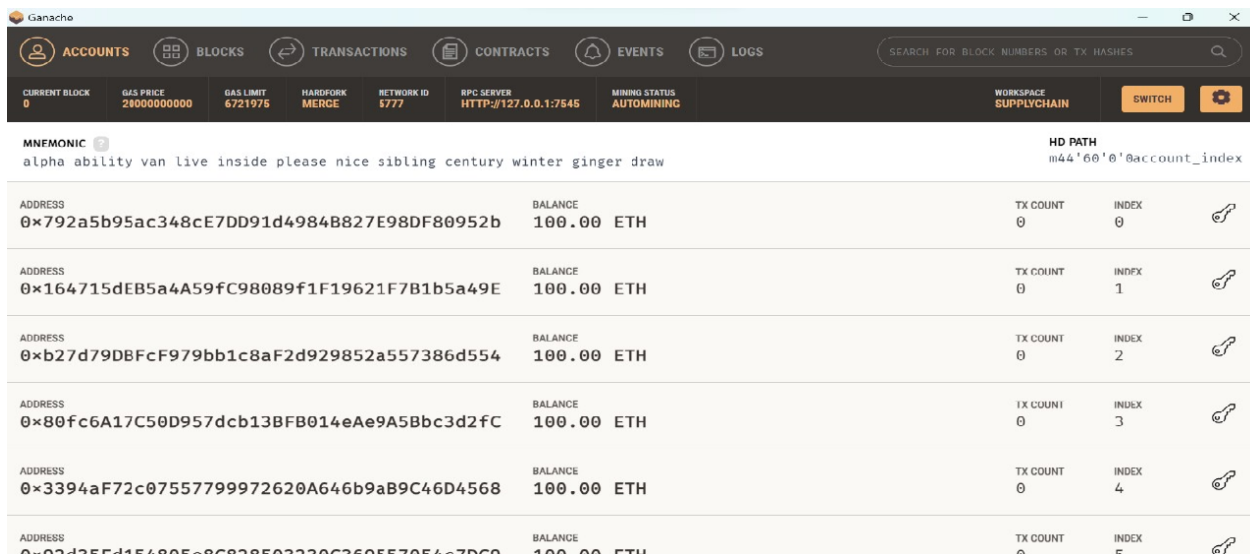
src\AssignRoles.js
  Line 84:19: Comparing to itself is potentially pointless      no-self-compare
  Line 102:9: 'redirect_to_home' is assigned a value but never used  no-unused-vars

src\Homesec.js
  Line 28:9: 'backgroundImagePath' is assigned a value but never used  no-unused-vars

src\Supply.js
  Line 18:10: 'MED' is assigned a value but never used      no-unused-vars
  Line 19:10: 'MedStage' is assigned a value but never used  no-unused-vars

```

Fig. 5.3.3. npm run start command output



ACCOUNTS	BLOCKS	TRANSACTIONS	CONTRACTS	EVENTS	LOGS
<p>CURRENT BLOCK: 0 GAS PRICE: 20000000000 GAS LIMIT: 6721976 HARDFORK: MERGE NETWORK ID: 5777 RPC SERVER: HTTP://127.0.0.1:7545 MINING STATUS: AUTOMINING WORKSPACE: SUPPLYCHAIN SWITCH </p> <p>MNEMONIC: alpha ability van live inside please nice sibling century winter ginger draw HD PATH: m/44'/60'/0'/0'/account_index</p>					
ADDRESS	BALANCE	TX COUNT	INDEX		
0x792a5b95ac348cE7DD91d4984B827E98DF80952b	100.00 ETH	0	0		
0x164715dEB5a4A59fC98089f1F19621F7B1b5a49E	100.00 ETH	0	1		
0xb27d79DBFcF979bb1c8aF2d929852a557386d554	100.00 ETH	0	2		
0x80fc6A17C50D957dcb13BFB014eAe9A5Bbc3d2fC	100.00 ETH	0	3		
0x3394aF72c07557799972620A646b9aB9C46D4568	100.00 ETH	0	4		
0x02d25F5d156885e8C838503230C2605F7054c7DC0	100.00 ETH	0	5		

Fig. 5.3.4. This is the Ganache interface which is the local blockchain where we have different ethereum addresses for actors included in the project.

• User Interface

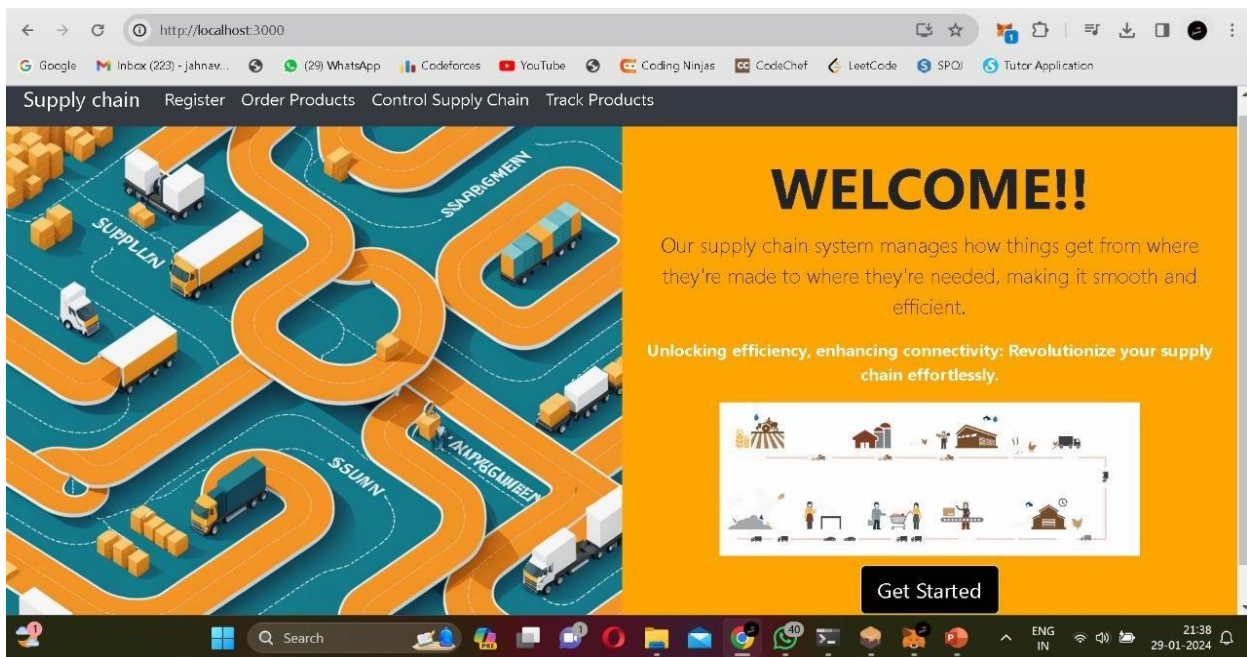


Fig. 5.3.5. This is the Home page of our project which consists of navigation bar with Register, Order Products, Control Supply Chain and Track Products Options.

The above Figure 5.3.5 is the Home page of the project, where we have a navbar which consists of SupplyChain, Register, Order Products, Control Supply chain, Track products options. These options are responsible for ordering the product from one user to another user. It resembles registration of Supplier, Manufacturer, Distributor, Retailer and Consumer in the blockchain with their name, location and ethereum address from the ganache local blockchain. It includes a carousel where by pressing arrow keys we can see the remaining actors included in the project such as Manufacturer, Distributor, Retailer, Consumer.

The screenshot shows a web browser at <http://localhost:3000/roles>. The page has a navigation bar with links: Supply chain, Register, Order Products, Control Supply Chain, and Track Products. The main content area has an orange background with the text "Current Account Address :". In the center is a white registration form titled "Raw Material Suppliers". The form contains three input fields: "Name of the Supplier :" (with placeholder "Raw Material Supplier Name"), "Enter the Location :" (with placeholder "Place"), and "Ethereum Address :". Below these fields is a black "Register" button. At the bottom of the form is a table with the following structure:

ID	Name	Place	Ethereum Address
----	------	-------	------------------

The Windows taskbar at the bottom shows the date as 29-01-2024 and time as 21:38.

Fig. 5.3.6. This is the Registration page of the project where the actors need to be registered.

The above Figure 5.3.6 resembles registration of Supplier, Manufacturer, Distributor, Retailer and Consumer in the blockchain with their name, location and ethereum address from the ganache local blockchain. It includes a carousel where by pressing arrow keys we can see the remaining actors included in the project such as Manufacturer, Distributor, Retailer, Consumer.

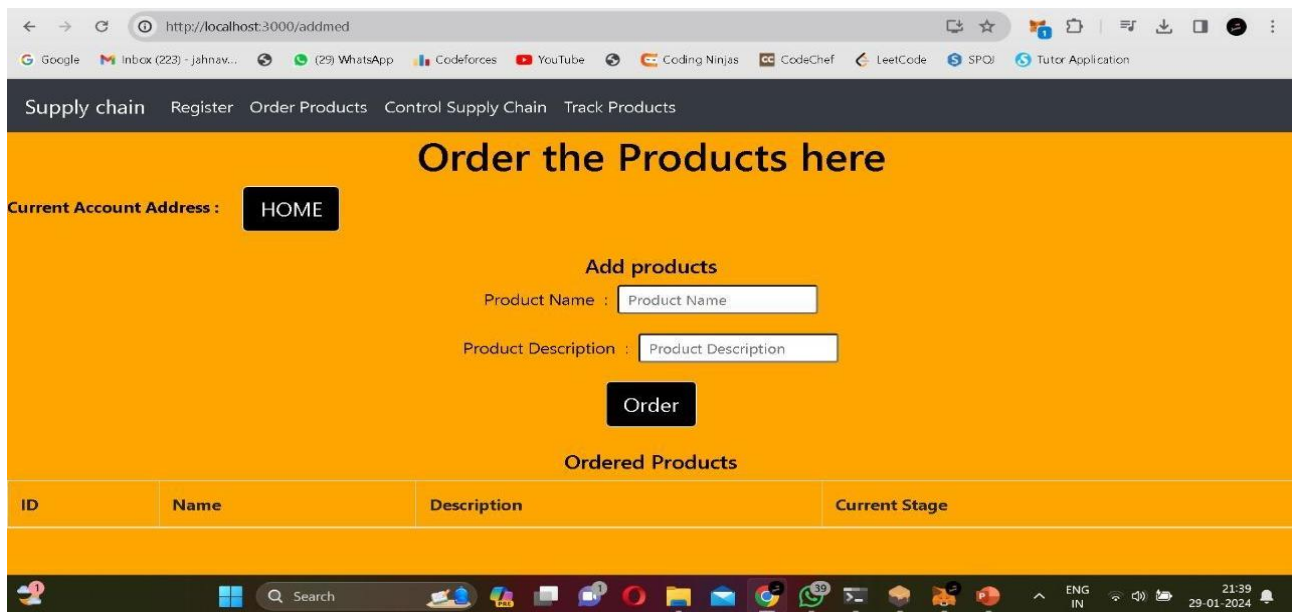


Fig. 5.3.7. This is the Order Products Page where the products are ordered by using Product Name and Product Description.

The Figure 5.3.7 is the Order products page where one can add the products in the blockchain so that they will transform from one actor to another actor. Here we have to enter Product name and Product Description.

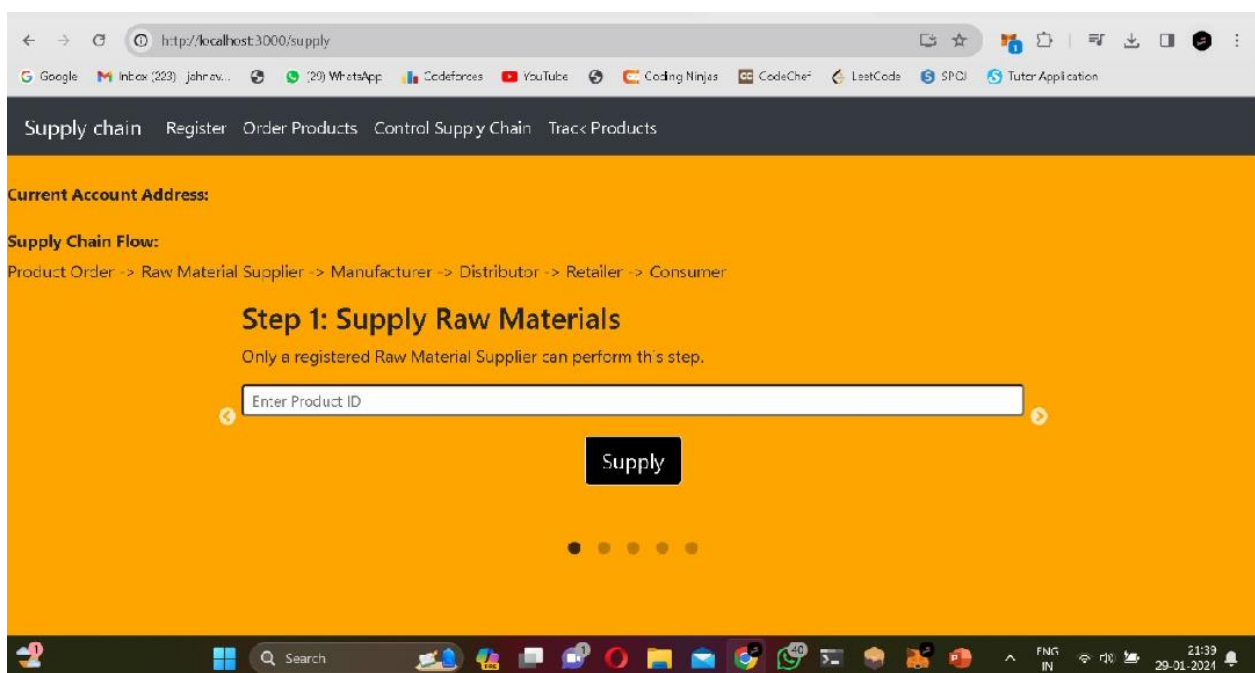


Fig. 5.3.8. This is the Control Supply chain Page where the product transferring happens.

The above Figure 5.3.8 is responsible to control the supply chain by entering the product ID. In this interface we have carousel where by pressing arrow keys we are able to send the

product to the next stage. We have to change the Current Account Address, according to the actor involved in the Supply chain at that particular time.

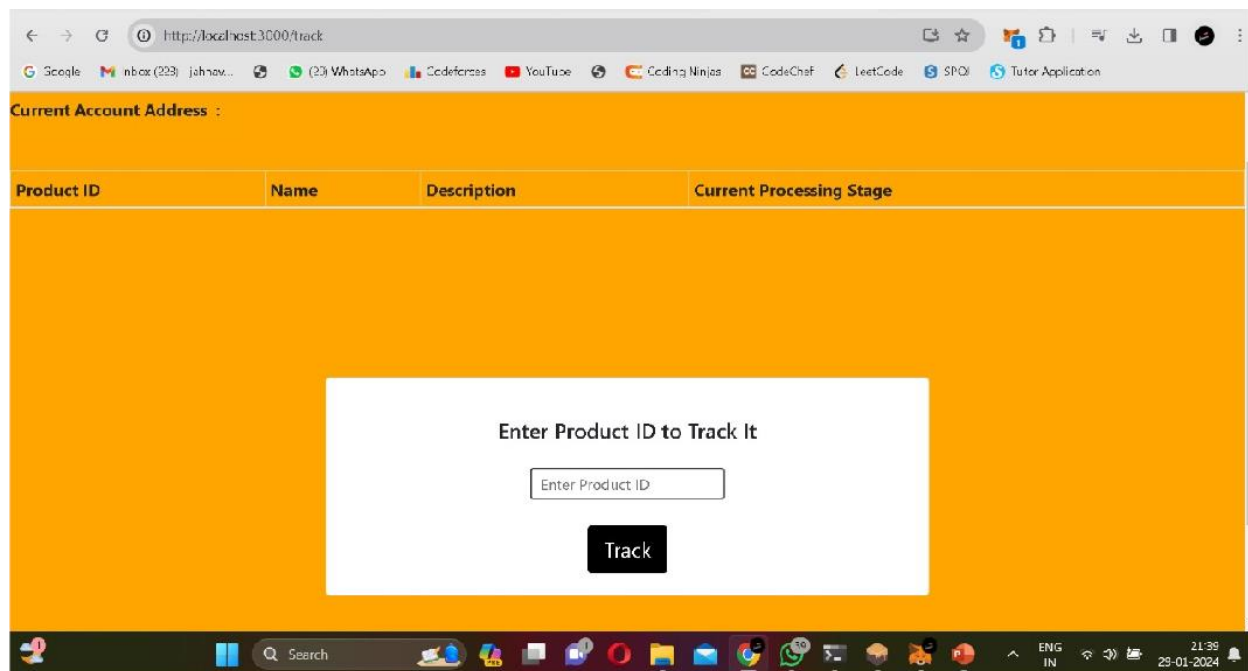


Fig. 5.3.9. This is the tracking page of the project where the ordered product is tracked by using its ID.

The above Figure 5.3.9 is related to tracking of the product which is ordered using its ID.

When we press Track, we can able to get the details of the product(in which stage the product is...)

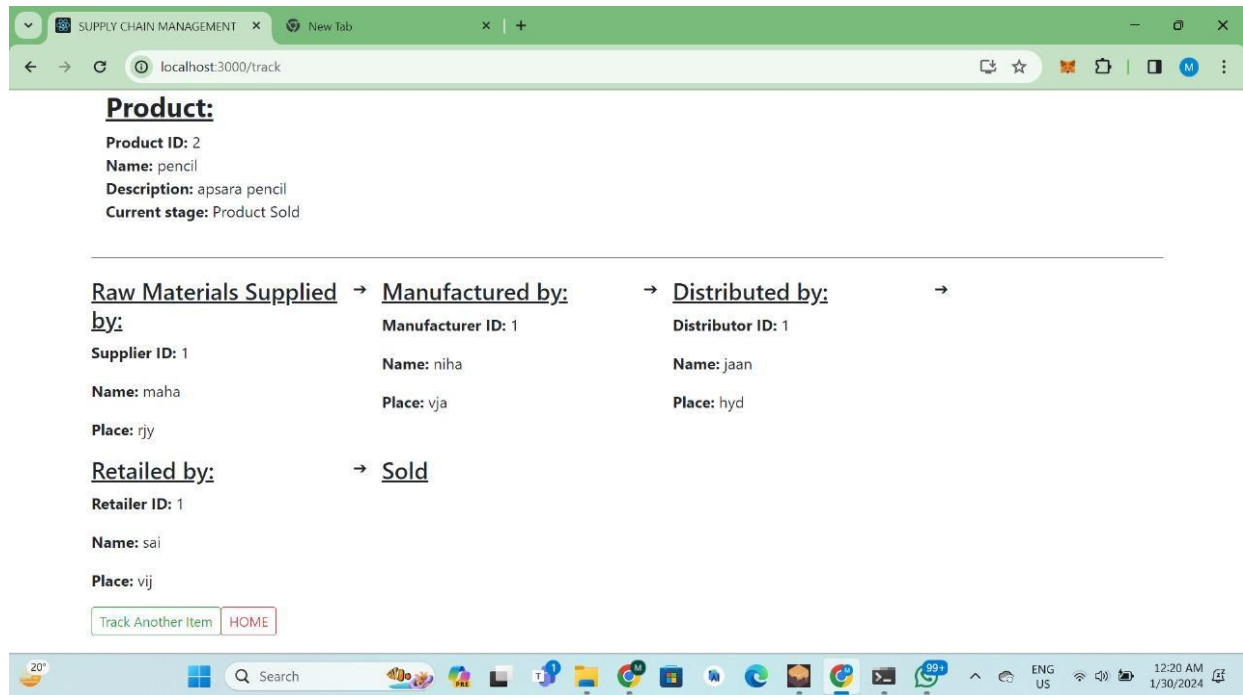


Fig. 5.3.10. This is the final output screen where the product details along with actors is displayed.

The above Figure:5.3.10 is the final output of the project when we press the Track button by entering the product ID. As shown in the above interface, we can able to see the Supplier, Manufacturer, Distributor, Retailer, Consumer involved in that particular product ordering.

6. System Testing

6.1. Introduction:

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies, and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail unacceptably. There are various types of tests. Each test type addresses a specific testing requirement.

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration-oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at the component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration tests are designed to test integrated software components to determine if they run as one program. Testing is event-driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfactory, as shown by successful unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional tests provide systematic demonstrations that functions tested are available as

specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input: Identified classes of valid input must be accepted. Invalid

Input: Identified classes of invalid input must be rejected. Functions:

Identified functions must be exercised.

Output: Identified classes of application outputs must be exercised.

Systems/Procedures: Interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identifying Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System testing is a critical phase in the software development life cycle (SDLC) where the entire software system is tested as a whole to ensure that it meets its specified requirements and functions correctly in its intended environment. Unlike unit testing and integration testing, which focus on testing individual components or the integration of those components, system testing evaluates the system as a whole, including its interactions with external systems and dependencies.

The primary objectives of system testing are to validate the system's functionality, performance, reliability, scalability, and security. It aims to uncover defects or discrepancies between the actual system behavior and the expected behavior defined in the system requirements. System testing also assesses the system's compliance with non-functional requirements such as usability, accessibility, and regulatory standards.

System testing typically involves the following key activities:

Test Planning: This phase involves defining the test objectives, scope, approach, resources, and schedule for system testing. Test plans are developed based on the system requirements and risk analysis to ensure comprehensive test coverage.

Test Case Design: Test cases are designed to validate the system's functionality,

performance, and other attributes. Test cases are derived from requirements, use cases, user stories, and other relevant documents. They cover both positive and negative scenarios to verify different aspects of the system behavior.

Test Execution: Test cases are executed against the system to identify defects and verify that the system behaves as expected. Test execution involves running test scripts, entering test data, and analyzing test results. Automated testing tools may be used to streamline and accelerate the test execution process.

Defect Tracking and Management: Defects identified during system testing are logged, prioritized, and tracked to resolution. A defect management system is used to document, assign, and monitor the status of defects throughout the testing process. Defects are retested after resolution to ensure they have been adequately addressed.

Regression Testing: Regression testing is performed to ensure that changes made to the system, either due to defect fixes or system enhancements, do not adversely impact existing functionality. Regression test suites are executed to validate the system's stability and integrity after modifications.

Performance Testing: Performance testing evaluates the system's responsiveness, throughput, and scalability under various load conditions. Performance tests simulate real-world usage scenarios to identify performance bottlenecks and optimize system performance.

Security Testing: Security testing assesses the system's ability to protect sensitive data, prevent unauthorized access, and withstand security threats and vulnerabilities. It includes testing for authentication, authorization, encryption, input validation, and other security controls.

Acceptance Testing: Acceptance testing involves validating the system against user acceptance criteria to determine whether it meets stakeholder expectations and business requirements. User acceptance testing (UAT) is typically performed by end-users or stakeholders in a production-like environment.

System testing is a crucial phase in the software development lifecycle that focuses on validating the entire integrated system to ensure that it meets the specified requirements and functions correctly. For the Supply chain Management System using blockchain, system testing plays a vital role in verifying the system's functionalities, performance, security, and reliability.

This phase involves testing the system as a whole, including its user interface, backend processes, data integrity, and interaction with the blockchain network.

System testing for the Supply chain Management System using blockchain is essential to ensure the system meets the desired quality standards, functionality requirements, and user expectations. By thoroughly testing the system's functionalities, performance, security, and reliability, organizations can mitigate risks, enhance user trust, and deliver a robust and reliable solution for Supply chain Management in the marketing sector.

6.2.Testing Methods:

White Box Testing:

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It has a purpose. It is used to test areas that cannot be reached from a black box level. White box testing, also known as clear box testing, glass box testing, or structural testing, is a software testing technique that involves examining the internal structure, code, and logic of a software application to ensure its correctness. Unlike black box testing, where the tester focuses solely on the external behavior of the software without considering its internal implementation, white box testing requires knowledge of the internal workings of the application's codebase.

The primary objectives of white box testing are to verify the correctness of individual code units, ensure that all paths and branches within the code are exercised, and validate the accuracy of algorithms and logic implemented in the software. White box testing is typically performed at the unit, integration, and system levels of software testing.

Black Box Testing:

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a test in which the software under test is treated as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

Black box testing is a software testing technique that focuses on assessing the functionality

of a software application without considering its internal code structure, implementation details, or logic. In black box testing, the tester treats the software as a black box, where only the input and output behaviors are visible, and the internal workings are not known or considered during testing.

The primary objective of black box testing is to validate the software against its specified requirements and expected behavior from an end-user perspective. It aims to identify defects, errors, and discrepancies between the actual behavior of the software and its intended functionality. Black box testing is typically performed at the system, integration, and acceptance testing levels.

Unit Testing:

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Unit testing is a software testing technique where individual units or components of a software application are tested in isolation to verify their correctness and functionality. A unit typically refers to the smallest testable part of the software, such as a function, method, class, or module. Unit testing is performed by developers during the development process to ensure that each unit of code behaves as expected and meets its specified requirements.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.
- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing:

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

Integration testing is a software testing technique that focuses on evaluating the interactions and interfaces between individual modules or components of a software application when integrated together. The primary goal of integration testing is to verify that the integrated modules work as expected and interact correctly with each other, ensuring that the software functions as a cohesive unit.

Integration testing is performed after unit testing and before system testing, typically during the middle stages of the software development life cycle (SDLC). It helps identify defects and issues that arise from the integration of different components, such as incompatible interfaces, data flow errors, communication failures, and module dependencies.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing:

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

6.3. Test Cases:

Creating test cases for the Supply chain Management System using blockchain involves validating various aspects of the system to ensure its functionality, security, and reliability.

Below are some example test cases covering different scenarios:

Smart Contract Deployment and Integration tests:

Test Case 1: Verify that the smart contract is deployed successfully on the blockchain.

Test Case 2: Confirm that the smart contract's address matches the expected address.

Test Case 3: Test interactions with the smart contract (e.g., adding products, updating status) and ensure the transactions are recorded on the blockchain.

Test Case 4: Execute truffle tests to check the integration of smart contracts with the application.

Ganache and Truffle Integration:

Test Case 5: Ensure that Ganache is connected to the blockchain network.

Test Case 6: Test that relevant events are emitted during different contract interactions.

Test Case 7: Confirm that events are logged accurately in the Truffle test output.

Test Case 8: Confirm that Ganache is configured with the correct network parameters (e.g., port, gas limit) for seamless Truffle integration.

Metamask and React Integration:

Test Case 9: Confirm that MetaMask is integrated with the React application.

Test Case 10: Test connecting MetaMask to the blockchain network and ensure the wallet is correctly linked.

Test Case 11: Validate that MetaMask prompts users to approve transactions initiated from the React application.

React Website:

Test Case 12: Confirm that the React website renders correctly with all necessary components.

Test Case 13: Test user authentication using MetaMask and verify access to appropriate features.

Test Case 14: Check if products are listed accurately on the website.

Test Case 15: Place an order through the website and verify the order details.

Test Case 16: Confirm that the order status updates in real-time when changed on the blockchain.

Product Ordering:

Test Case 17: Initiate an order for a specific product through the React website.

Test Case 18: Verify that the order is successfully recorded on the blockchain.

Product Tracking:

Test Case 19: Simulate the product's journey from creation to shipping by updating its status at each stage (e.g., "Manufacturing," "In Transit").

Test Case 18: Verify that each status change is recorded on the blockchain.

Test Case 19: Check the product's status after it is marked as "Sold" on the blockchain.

Usability Testing:

Test Case 19: Evaluate how clearly the order status is presented to users throughout the supply chain journey.

Test Case 20: Test a scenario where a transaction runs out of gas, and verify that it fails gracefully with an appropriate error message.

Network Connectivity:

Test Case 21: Simulate a temporary loss of connection to the Ganache blockchain network during Truffle tests.

Test Case 22: Test the handling of errors and exceptions within Truffle tests, ensuring that error messages are meaningful.

These test cases cover a range of scenarios to thoroughly validate the functionality, security, and performance of the Student Certificate Verification System using blockchain. Additional test cases may be required based on the specific requirements and features of the system.

7. Conclusion

The incorporation of blockchain technology into supply chain management (SCM) has resulted in a revolutionary change, enabled by platforms such as Ganache and MetaMask. This convergence improves transparency, traceability, and overall efficiency by addressing important industry concerns. Blockchain, as a decentralized and immutable ledger, ensures a single version of truth across the supply chain. This is especially important in supply chain management (SCM), as there are numerous parties involved, each with their own data and set of procedures. Every link in the supply chain can obtain a transparent and safe transaction record by utilizing blockchain technology. This fosters trust and minimizes the risk of discrepancies or fraudulent activities.

The use of Ganache, a personal blockchain for Ethereum development, provides a practical and controlled environment for testing and simulating blockchain applications. It allows SCM professionals and developers to experiment with various scenarios, ensuring the robustness and reliability of the implemented solutions before they are deployed in a production environment. A wider spectrum of enterprises can now access blockchain development with Ganache's cost-effective and scalable approach. Because of Ganache's versatility and ease of use, developers can design customized solutions to solve certain supply chain problems. Its characteristics support the creation of safe, open, and effective systems, building stakeholder confidence and setting the stage for an ecosystem of the supply chain that is more responsive and resilient.

MetaMask, a cryptocurrency wallet and gateway to blockchain applications adds another layer of usability to the SCM ecosystem. With MetaMask, users can seamlessly interact with blockchain-based supply chain applications, making the technology more accessible and user-friendly. For broad adoption by all parties involved in the supply chain—from producers and distributors to retailers and customers—this inclusion is crucial.

8. References

- [1] Chopra, S., & Meindl, P. (2019). Supply chain management: Strategy, planning, and operation. Pearson.
- [2] Seuring, S., & Müller, M. (2008). From a literature review to a conceptual framework for sustainable supply chain management. *Journal of Cleaner Production*, 16(15), 1699-1710.
- [3] Amulya Gurtu: supply chain management with blockchain technology's potential.
- [4] Shuchih Ernest Chang's book, *When Blockchain Meets Supply Chain: A Systematic Literature Review on Current Development and Possible Applications*, was published in 2017.
- [5] "Blockchain technology in supply chain management: An application perspective" by Trung Thanh Nguyen, Phuc Doan Nguyen, and Hong Va Le. Published in 2018
- [6] Lee, H. L., & Tang, C. S. (2018). Reshaping the future of supply chain management. *Journal of Operations Management*, 60, 1-10.
- [7] Sarkis, J. (2019). Supply chain sustainability: Learning from the past and mapping a path forward. *Decision Sciences*, 50(2), 231-256.
- [8] Ivanov, D. (2020). Viable supply chain model: Integrating agility, resilience, and sustainability perspectives—Lessons from and thinking beyond the COVID-19 pandemic. *Annals of Operations Research*, 1-21.
- [9] Garima Mathur.(2023). GANACHE : A Robust Framework for Efficient and Secure Storage of Data on Private Ethereum Blockchains.
- [10] Dr. Kumud Saxena , vaibhav kushwaha, Umang Gupta(2023) Ethereum transaction using metamask wallet.
- [11] Trivedi, A., Patel, V., & Prajapati, B. (2020). Blockchain technology in supply chain management: A review. In *2020 International Conference on Smart Electronics and Communication (ICOSEC)* (pp. 548-553). IEEE.
- [12] Sarkis, J., & Cohen, M. J. (2019). Blockchain applications and research opportunities in supply chain management. *Transportation Research Part E: Logistics and Transportation Review*, 125, 423-441.
- [13] Katina Michael, Keith W. Miller, Jeremy Pitt (2019). Blockchain technology in the supply chain: An integrated theoretical perspective of organizational adoption.
- [14] Xiaxia Niu, Zeping Li. Research on Supply Chain Management Based on Blockchain Technology.
- [15] Casey and Wong, Babich, and Hilary,(2017). Blockchain technology in the supply chain.
- [16] Jianting Xia, Haohua Li, Zhou He.The Effect of Blockchain Technology on Supply Chain Collaboration: A Case Study of Lenovo.

- [17] Shivani Bhalerao, Siya Agarwal, Shruthi Borkar, Shruti Anekar, Nikita Kulkarni, Sumedha Bhagwat. (IEEE) Supply Chain Management using Blockchain.
- [18] Tan Guerpinar, Gilberto Guadiana, Philipp Asterios Ioannidis, Natalia Straub, Michael Henke (2021). The Current State of Blockchain Applications in Supply Chain Management
- [19] Sara Saberi, Mahtab Kouhizadeh, Joseph Sarkis & Lejia Shen (2018). Blockchain technology and its relationships to sustainable supply chain management
- [20] Rita Azzi a, Rima Kilany Chamoun a, Maria Sokhn ba (2019). The power of a blockchain-based supply chain