Product Identification using Blockchain

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

Computer Science and Engineering

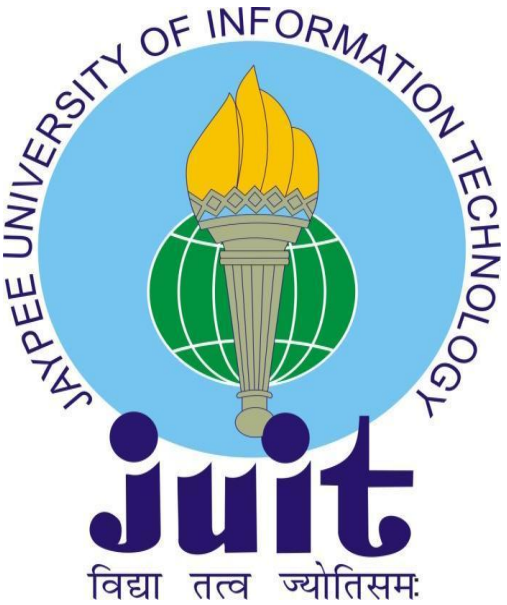
By

Parth Purwar (191281)

Under the supervision of

Dr. Amit Kumar

to



Department of Computer Science & Engineering and Information Technology
Jaypee University of Information Technology Waknaghat,

Solan-173234, Himachal Pradesh

CERTIFICATE

This is to certify that the work which is being presented in the project report titled "**Product Identification Using Blockchain**" in partial fulfillment of the requirements for the award of the degree of B.Tech in Computer Science And Engineering and submitted to the Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat is an authentic record of work carried out by "Parth Purwar, 191281" during the period from January 2023 to May 2023 under the supervision of Dr. Amit Kumar, Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat.

Parth Purwar
(191281)

The above statement made is correct to the best of my knowledge.

Dr. Amit Kuram
Associate Professor, Grade 2
Computer Science & Engineering and Information Technology
Jaypee University of Information Technology, Waknaghat

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

PLAGIARISM VERIFICATION REPORT

Date: ……

Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper

Name: _____ **Department:** _____ **Enrolment No** _____

Contact No. _____ **E-mail.** _____

Name of the Supervisor: _____

Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): _____

_____

_____

UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

Complete Thesis/Report Pages Detail:

- Total No. of Pages =
- Total No. of Preliminary pages =
- Total No. of pages accommodate bibliography/references =

(Signature of Student)

FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ……(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

(Signature of Guide/Supervisor) **Signature of HOD**

FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

<table>
<tr><th>Copy Received on</th><th>Excluded</th><th>Similarity Index (%)</th><th colspan="2">Generated Plagiarism Report Details (Title, Abstract & Chapters)</th></tr>
<tr><td rowspan="2"></td><td rowspan="4">• All Preliminary Pages
• Bibliography/Images/Quotes
• 14 Words String</td><td rowspan="2"></td><td>Word Counts</td><td></td></tr>
<tr><td rowspan="2">Report Generated on</td><td>Character Counts</td><td></td></tr>
<tr><td rowspan="2"></td><td>Submission ID</td><td>Total Pages Scanned</td><td></td></tr>
<tr><td></td><td>File Size</td><td></td></tr>
</table>

Checked by
Name & Signature **Librarian**

……

Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com

ORIGINALITY REPORT

16%	13%	4%	7%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	ethereum.org Internet Source	3%
2	arxiv.org Internet Source	2%
3	www.researchgate.net Internet Source	1%
4	Krutika Desai, Jinan Fiaidhi. "Developing a Social Platform using MERN Stack", Institute of Electrical and Electronics Engineers (IEEE), 2022 Publication	1%
5	Firdous Sadaf Mohammad Ismail, Sadaf Gauhar Mohammad Mushtaque, Dattatraya Adane. "chapter 7 Struggles, Potential, and Research Angles in the Amalgamation of Blockchain Technology With 6G Networks", IGI Global, 2023 Publication	1%
6	Submitted to Dr. S. P. Mukherjee International Institute of Information Technology (IIIT-NR) Student Paper	1%
7	Submitted to SSN COLLEGE OF ENGINEERING, Kalavakkam Student Paper	1%
8	github.com Internet Source	1%
9	Submitted to Westminster International University in Tashkent Student Paper	1%

ACKNOWLEDGEMENT

Firstly, we express our heartiest thanks and gratefulness to almighty God for His divine blessing makes it possible to complete the project work successfully.

We are really grateful and wish our profound indebtedness to Supervisor **Dr. Amit Kumar, Associate Professor(Grade 2)**, Department of CSE/IT Jaypee University of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of our supervisor in the field of "**Block Chain**" to carry out this project. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this project.

We would like to express our heartiest gratitude to **Dr. Amit Kumar,** Department of CSE/IT, for his kind help to finish our project.

We would also generously welcome each one of those individuals who have helped us straightforwardly or in a roundabout way in making this project a win. In this unique situation, We also want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking. Finally, We must acknowledge with due respect the constant support and patience of our parents.

Parth Purwar
(191281)

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Counterfeiting has become an increasingly prevalent issue in the context of globalization and the rapid advancement of technology. To combat this problem, various industrial producers and distributors have been working towards enhancing the transparency of their supply chain operations. In this paper, we propose a decentralized Blockchain-based application system (DApp) that can be used to detect counterfeit goods in the supply chain system.

We leverage the security and immutability of data stored on the Blockchain to facilitate the transfer of ownership of goods. Our proposed system issues a Quick Response (QR) code for each product, which is connected to the Blockchain and can be scanned by customers to verify the distribution and ownership information of the goods. This paper presents the design and implementation of our DApp system and evaluates its effectiveness in detecting and preventing counterfeit goods in the supply chain. The results of our experiments demonstrate the feasibility and potential of using Blockchain technology in combating counterfeiting.

Chapter 1: INTRODUCTION

1.1 Introduction

The inability of the consumer being able to authenticate their purchases has led to the growth of a market that thrives on selling copies/counterfeit of real products that are both.
This market has grown tremendously in recent years owing to the dominance of e-commerce and online outlets and the ease with which a product can be sold and bought there.

From apparel to electronics to medicines the counterfeit product market has been harming both consumers as well as the manufacturer of authentic products. From the product not being up to the mark in terms of quality and not working as the authentic product should to being outright life-threatening when it comes to medicines.
If the threat of encountering scams online and the bank details being stolen was already not enough, all of this leading to consumers being swayed away from making online purchases.

Here, we suggest a product verification system based on blockchain technology that is completely secure, safe, and dependable to combat this growing industry of counterfeit goods.
Barcodes, electronic product codes (EPCs), and RFID technologies are used by more advanced centralised systems to track products in the supply chain. However, these systems are fundamentally unsafe because they rely on centralised certification bodies and databases, which have single points of failure that leave them open to hacker attacks and insider fraud.

Blockchain solutions that are decentralized and irreversible allow for product authenticity and traceability at every stage of the supply chain. Based on this, several blockchain projects have already released decentralized applications (dApps), which use data from the supply chain to confirm that a commodity,

like luxury items or food, is actually authentic. The user of the dApp can authenticate and track the merchandise completely by scanning the QR code on the item.

1.2 Problem Statement

Producing and distributing fake goods is a pressing and crucial global problem, particularly in underdeveloped nations where there is less of a retail presence for major brands than in more affluent nations.

A reputable news source, The Guardian, said that the annual value of the trade in counterfeit goods is $600 billion. Up to 10% of all branded products sold could be fake. Whether knowingly or unknowingly, 80% of us have handled phony or counterfeit goods. The demand for luxury products has expanded dramatically in recent years, but the growth of fakes has been even faster: one estimate claims that fakes have grown by 10,000% in just two decades.

One of the major ways in which counterfeit products are sold is through the second-hand/resale market and the reason why it's so much easier is thanks to the not so perfect supply chain system. The ownership of a product changes from manufacturers to distributor and then to a wholesaler and retailer before it reaches the customer. Therefore, a product goes through many hoops and intermediate players, each presenting a new avenue to replace it with a counterfeit product before it reaches the ultimate player, the customer.

1.3 Tools/Technology used

a) Blockchain can be centralized or decentralized. However, it is important not to confuse decentralized with distributed. While blockchain is inherently distributed (meaning many parties own copies of the ledger), it is not inherently decentralized. Whether a blockchain is centralized or decentralized simply refers to the rights of participants in the ledger and is therefore a matter of design.

b) A digital signature verifies data integrity using asymmetric cryptography. A digital signature is nearly impossible to forge without access to the sender's private key. A digital signature is created by encrypting the hash of the document using a private key. The recipient can use the public key to decrypt the signature and ensure that the result actually matches the hash of the document. If the document is changed after signing, the digital signature is invalidated because the hash of the document will not match the decrypted signature. A digital signature not only verifies the identity of the signer but also verifies the content of the message itself.

c) Distributed Ledger Technology (DLT) refers to a technological infrastructure and protocol that allows simultaneous access, verification, and recording in an immutable manner in a network spread across multiple entities or locations. DLT, better known as Blockchain technology

d) Smart contracts, simply put, are blockchain-based programmes that execute when certain criteria are met. They are typically used to automate the implementation of an agreement so that all parties can be certain of the outcome right away, without the need for any intermediaries or additional time. They can also automate a workflow such that when certain criteria are met, the next action is initiated.

Benefits-

i) Speed, efficiency, and accuracy

The contract is immediately completed once the requirement is satisfied. There is no need to file any papers or spend time gathering errors that frequently occur when manually filling out documentation because smart contracts are digital and automated.

ii) Trust and transparency

There is no question that information has been altered for private gain because there is no external party engaged and participants share encrypted transaction logs.

iii) Safety

Since the records of blockchain transactions are encrypted, no one can easily hack them. Additionally, since every record in the distributed ledger is linked to all previous and subsequent records, hackers would need to alter the entire chain in order to alter just one record.

iv) Savings

Smart contracts eliminate the need for transactions to be processed by intermediaries, and thus the associated time delays and fees.

e) Node.js, an open-source, cross-platform, back-end JavaScript runtime running on the JavaScript engine.

f) Ganache, a private blockchain for Ethereum development that you can use to publish contracts, develop your application, and run tests.

g) Truffles, a world-class development environment, testbed, and active pipeline for blockchains using the Ethereum Virtual Machine (EVM), which aims to make life easier as a developer.

h) Remix IDE allows the development, deployment and management of smart contracts for Ethereum, such as blockchain. It can also be used as a learning platform.

i) MongoDB, the available platform for document-oriented database applications.

j) Solidity, a programming language for creating smart contracts on the Ethereum blockchain.

1.4 Objectives

The idea of this project arose from the need to counter the ever-growing market of counterfeit products.

The objectives of this project are:

1. Design an anti-counterfeit system.
2. Create an infrastructure for the supply chain, from manufacturer to consumer.
3. Make the system transparent and decentralized.
4. Make the system immutable to protect against attacks and hacks.
5. To achieve the above, build the system on Ethereum blockchain.
6. Secure product details using a QR code.
7. Protect the consumer from frauds by offering them data at every stage of the supply chain.

1.5 Methodology

The figure below illustrates the execution/working of a smart contract written in Solidity on Ethereum blockchain.

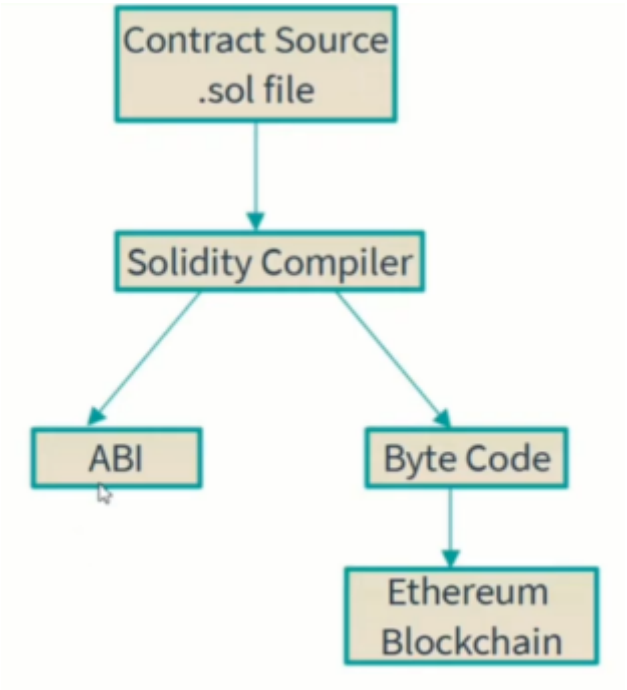


Figure 1. Working of an Ethereum smart contract

- ABI - The Contract Application Binary Interface (ABI) is the default way of dealing with contracts in the Ethereum ecosystem, both from outside the blockchain and for contract-to-contract communication. This standard describes how data is encoded based on its kind. A schema must be applied in order to decode the encoding because it is not self-descriptive.

- Byte Code - Bytecode is the "translated" data from our Solidity code. Two-way computer guidance is available. Bytecode is usually a small number of codes, sequences, and other pieces of data. Each instruction step is an operation called "opcodes" which is usually one byte (eight bits) in length. That is why they are called "bytecode" - single-byte code.

- EVM - After each new block is added to the chain, the Ethereum Virtual Machine, also known as the EVM, computes the state of the Ethereum network and executes smart contracts. The Ethereum Virtual Machine is built on top of Ethereum's hardware and node network layer.

Note:-

1. Contract bytecode is public in readable form
2. Contract doesn’t have to be public.
3. Bytecode is immutable
4. ABI acts as a bridge between applications and smart contracts
5. ABI and Bytecode cannot be generated without source code

The diagram below shows the basic structure or idea behind how the system needs to be built.

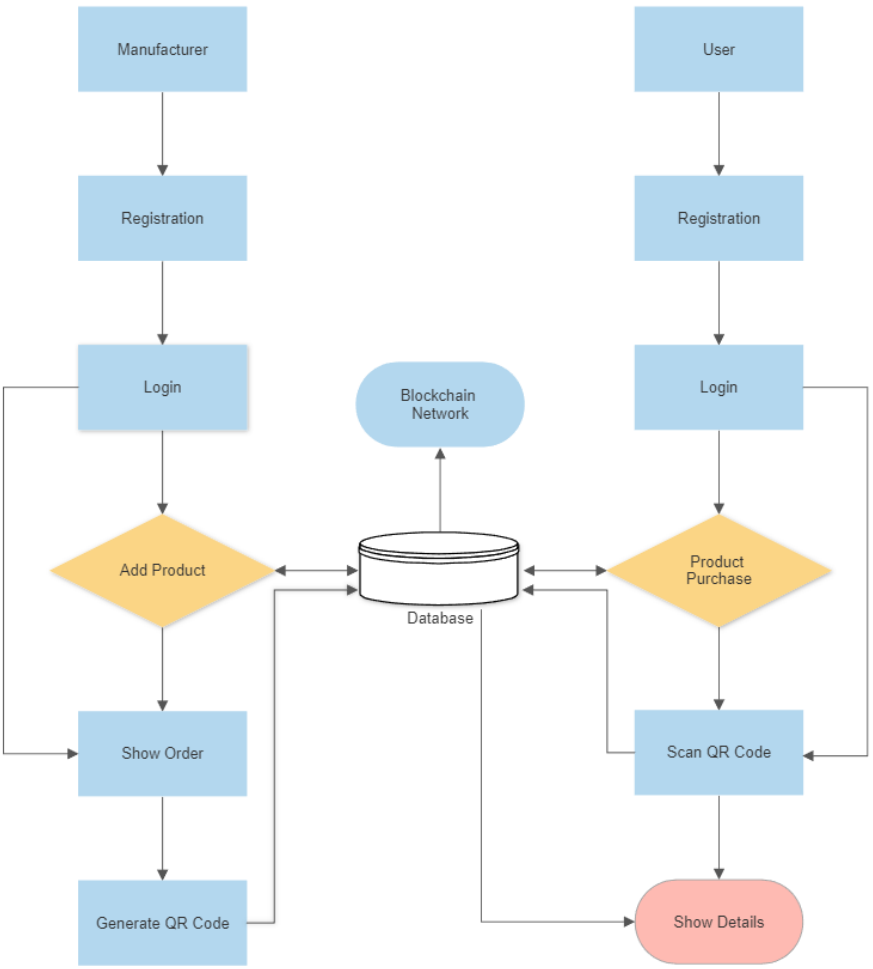


Figure 2. Flow chart of the system design

The diagram below illustrates a more detailed working structure of the supply chain pipeline and how everyone (participants of the supply chain) interact with the blockchain and the powers they hold.

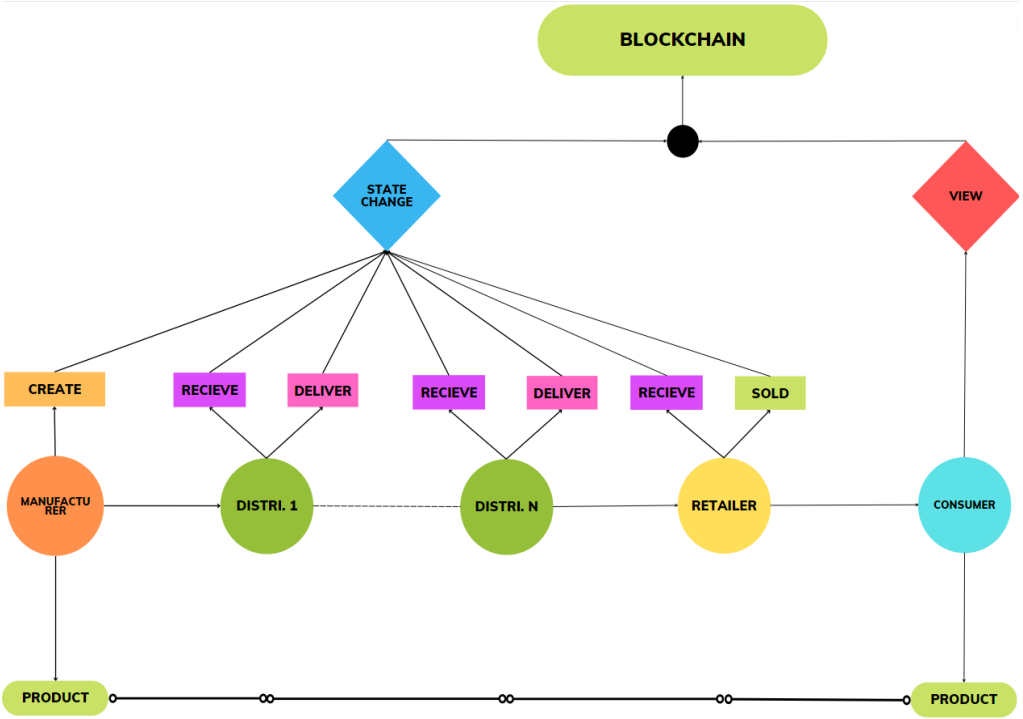


Figure 3. Flow chart showing different participants

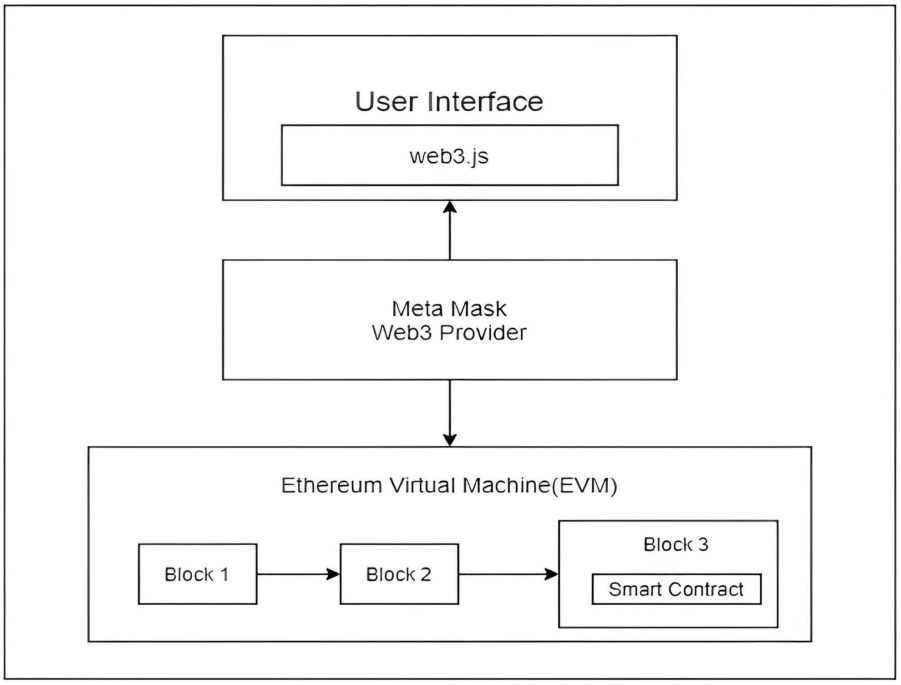


Figure 4. System architecture of an Ethereum Virtual Machine

1.6 Organization

A. System Diagram

The proposed DApp's system diagram is shown in Fig. 1.2. Before logging in, every user of the DApp must first be authorized, regardless of whether they are a producer, distributor, retailer, or customer. Utilizing MongoDB, this authentication mechanism was put into place. The manufacturer can join their business to the DApp and enroll their items after a successful authentication. The manufacturer is given the company's contract address, and the entire company's data as well as the account address of the manufacturer are recorded in the blockchain network. A QR code is given to a product after it has been added to the blockchain for verification. After registering, the distributor/sellers can purchase goods from the manufacturer. The QR code allows for tracking of the product's ownership transfer.

B. Manufacturer

The manufacturer's duties include adding the business to the blockchain, naming it, and establishing the minimal registration cost for third parties that wish to sell or buy from the business. Only the manufacturer retains the authority to add products to the network. After a merchant purchases product stock, the manufacturer may transfer ownership. The manufacturer's two main tasks in this system are adding and distributing products. A product is added using Algorithm 1.

Algorithm 1 Create Product

Input: Product Name, Product Price, Product Stock
Output: Added Product
if *msg.sender* is not manufacturer **then**
    throw;
    end
else

insert product in product array

end if

For distribution of product Algorithm 2 is used. The product and order status in the blockchain is changed through this.

Algorithm 2 Distribute Product

Input: Product ID

Output: Changed Product Status

if *msg.sender* is not manufacturer **then**

throw;

end

else

change product status to ‘shipped’ and set order status as ‘complete’

end if

C. Seller

A vendor can register for the business by paying the minimal fee imposed by the manufacturer. The vendor can purchase any goods and follow the distribution of it after registering just once. When a product is delivered to the seller by the manufacturer, its status is changed from "Ready To Go" to "Shipped."

Algorithm 3 here is used to make sure a seller pays the minimum registration fee set by the manufacturer.

Algorithm 3 Seller(Distributor/Retailer) Registration

Input: Min. registration fee set by manufacturer

Output: Registered Seller

if *msg.sender* is already registered or fee not appropriate **then**

throw;

end

else

map msg.sender is **true**

end if

Algorithm 4 Complete Purchase

Input: : ProdID, Seller Name, Quantity

Output: Set Current Owner of product as msg.Sender

if *msg.sender* is not registered seller **then**

throw;

end

else if *msg.value* is less that required amount **then**

throw;

end

else

set product owner name to seller name and store account address of seller

end if

Here, the seller purchases or books products from the manufacturer using algorithm 4. It stores information about the seller in the blockchain.

D. Customer

Customers can confirm the ownership transfer from manufacturer to distributor to seller to the customer by scanning the QR code included with each product. Additionally, the buyer has the option of checking the product's distribution status and the name of the product's current owner.

Chapter 2: LITERATURE SURVEY

In a paper titled “Identifying Counterfeit Products using Blockchain Technology in Supply Chain System ” published in IEEE by Nafisa Anjum and Pramit Dutta, the system proposed here used the MetaMask cryptocurrency wallet for transactions and the smart contract here was deployed in Rinkeby; Ethereum Blockchain Testnet. The dApp is based on three main stakeholders, producers, sellers, and consumers.

However, the disadvantage of this paper was that there was no system in place to track the detail of the discontinuation of a product or return of a product.
In another paper published in the same journal by Yasmeen Dabbagh, Reem Khoja, Leena Al Zahrani, Ghada Al Showaier, and Nidal Nasser, titled “A Blockchain-Based Fake Product Identification System” an implementation of smart contract on solidity was proposed, but the paper failed to propose details on how the actual system should be implemented.

In the journal IJARIIE a paper titled “Fake Product Detection Using Blockchain Technology " was published by Tejaswini Tambe, Sonali Chitalkar, Manali Khurud, Madhavi Varpe, S. Y. Raut where they were able to create an application using Android Studio and Firebase as the central database, but it went against the principle of decentralization which is the fundamental base for any blockchain-based application. Also, there was no mention of cryptocurrency used or the system design in place and its implementation.

The survey "A Survey on Counterfeit Product Detection" by Prabhu Shankar and R. Jayavadivel. Because there are so many products on the black market and online, the market for counterfeit goods is growing rapidly. Therefore, it is imperative to find a solution to the problems associated with identifying fake goods and to create the necessary technology to increase detection precision. This is one of the active study areas being studied in the current era. This essay examines many techniques for identifying counterfeit items.

Table 1. Literature Survey

Author(s)	Research Paper	Publisher	Methodology	Disadvantage
Nafisa Anjum Pramit Dutta	Identifying Counterfeit Products using Blockchain Technology in Supply Chain System [2]	IEEE	The system proposed here uses the MetaMask cryptocurrency wallet for transactions and smart contract has here were deployed in the Rinkeby Test Network of the Ethereum Blockchain. DApp is based on three main stakeholders, producer, seller and consumer.	No detail about discontinuation of a product or return of a product
Yasmeen Dabbagh, Reem Khoja, Leena AlZahrani, Ghada AlShowaier, Nidal Nasser	A Blockchain-Based Fake Product Identification System [3]	IEEE	Smart Contract implemented using Solidity.	Not much detail about implementation.
Tejaswini Tambe, Sonali Chitalkar, Manali Khurud, Madhavi Varpe, S. Y. Raut	Fake Product Detection Using Blockchain Technology [4]	IJARIIE	Used Android Studio for interface and Firestore as centralized database	No mention of cryptocurrency used. No mention of system design in place and its implementation.
Roshan Jadhav; Altaf Shaikh; M. A. Jawale; A.B. Pawar; P. William	System for Identifying Fake Product using Blockchain Technology[5]	IEEE	develop the necessary technology to enhance detection accuracy	

Chapter 3: SYSTEM DEVELOPMENT

3.1 Introduction

This section would describe the system development of the product in detail along with the system design, the data flows of the project.

3.2 Authentication

Authentication of the system is an important part of the problem to tackle first. Since there are many types of users in the system namely the customer, the Original Equipment Manufacturer(OEM), the distributor and the retailer, authentication of each of the players needs to be handled by the backend.

The JSON Web Token (JWT) standard was used to achieve system authentication. A concise, URL-safe method of encoding claims that need to be exchanged between two parties is the JSON Web Token (JWT). A JWT's claims are encoded as a JSON object, which is used as the payload of a JSON Web Signature (JWS) structure or as the plain-text of a JSON Web Encryption (JWE) structure, allowing the claims to be digitally signed or integrity protected with a Message Authentication Code (MAC) and/or encrypted..

JSON Web Tokens(JWT) can be used mainly in two scenarios namely:

- Authorization: JWT can be used for authorization of a user in a system which is the use case for the project. For each protected route, the JWT token would also be sent in the request body itself. The received token would be validated at the server-side allowing a user to access resources, protected routes and services that are permitted within the scope of that user.

- Exchange of information: For exchanging information between two parties, JSON Web Token is a good way of ensuring that:

    (1) Senders are who they are claiming to be since JWTs are signed using either secret key or public/private key encryption. Taking example of public-private key encryption, at the receiver side, the receiver would have to use the sender's public key in order to decrypt the received message. Since, no key other than that of the sender can decrypt the message, it can be ensured that the message can be sent by the sender itself and not a third party masquerading as the sender.

    (2) Payload or the message has not been tampered with since at the server side, the signature is recalculated and is checked with the original hash. Checking both of them ensures that the information has been tampered with and is the same as what was sent.

Structure of a JSON Web Token(JWT) consists of three parts namely the header, the payload and the signature.

Header is nothing but a Javascript Object and consists of two parts: the type of the token(denoted by key "typ", can be JWT, JWE and so on) and the hashing algorithm used for signing the payload(denoted by "alg", can be SHA 256, SHA 384, RSA and so on). An example of a header is denoted below;

```
{
   "alg": "HS256",
   "typ": "JWT"
}
```

This header is then encoded using the Base64Url encoding to form the first part of a JWT token. For example, the base64url encode of the above header would be:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Similar to the header, payload is also a Javascript Object and contains claims. In JWT, the statement about a user along with additional related data is called a claim. There are three types of claims namely Registered Claims, Public Claims and Private Claims. Registered claims are predetermined claims. Some of the examples of registered claims are subject(sub), issuer(iss), expiration time(exp) and so on. Public claims are generally custom claims defined by the user. Private claims are also custom claims and are used in exchanging information when two parties agree on them. An example of payload in JSON format is given below:

```
{
  "username": "John Doe",
  "pass": "test@password"
  "admin": true
}
```

This payload is then encoded using the Base64Url encoding just like the header before it to form the second part of a JWT token. For example, the base64url encode of the above payload would be:

eyJ1c2VybmFtZSI6IkpvaG4gRG9lIiwicGFzcyI6InRlc3RAcGFzc3dvcmQiLCJhZG1pbiI6dHJ1ZX0

Finally, using the algorithm described in the header part of the token, the base64url encoded headed, the base64url encoded payload and a secret key is hashed together to for the third and final part of the JSON Web Token(JWT), the signature. This process can be denoted as:

```
HashingAlgorithm(
   base64UrlEncode(header) + "." +
   base64UrlEncode(payload),
   secret_key)
```

For the given payload and header and using “aGVsbG8sIGkgYW0gc29sY” as the secret key for hashing, the above information can be encoded into a JWT token as:

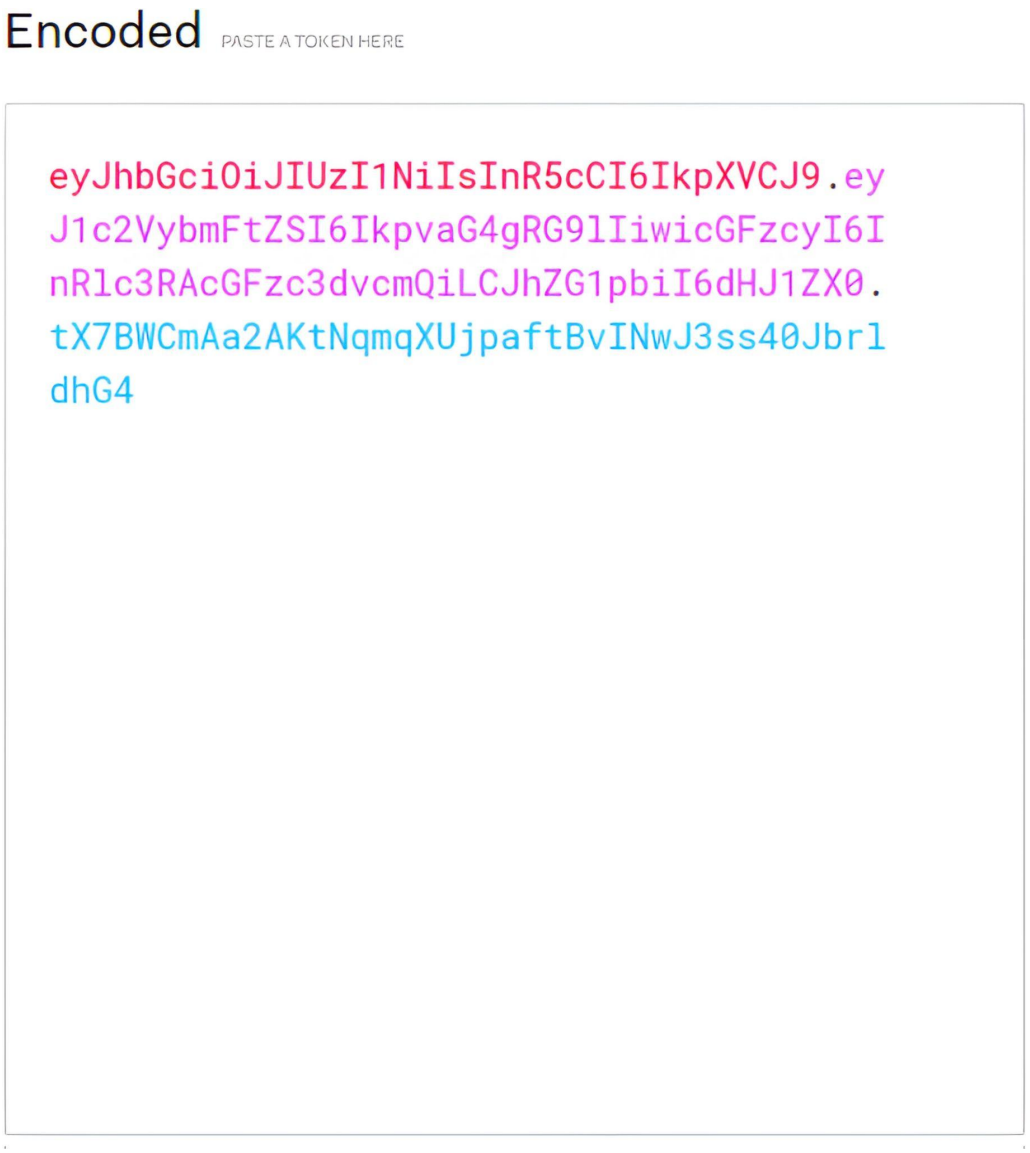


Figure 5. Encoding a JSON Web Token

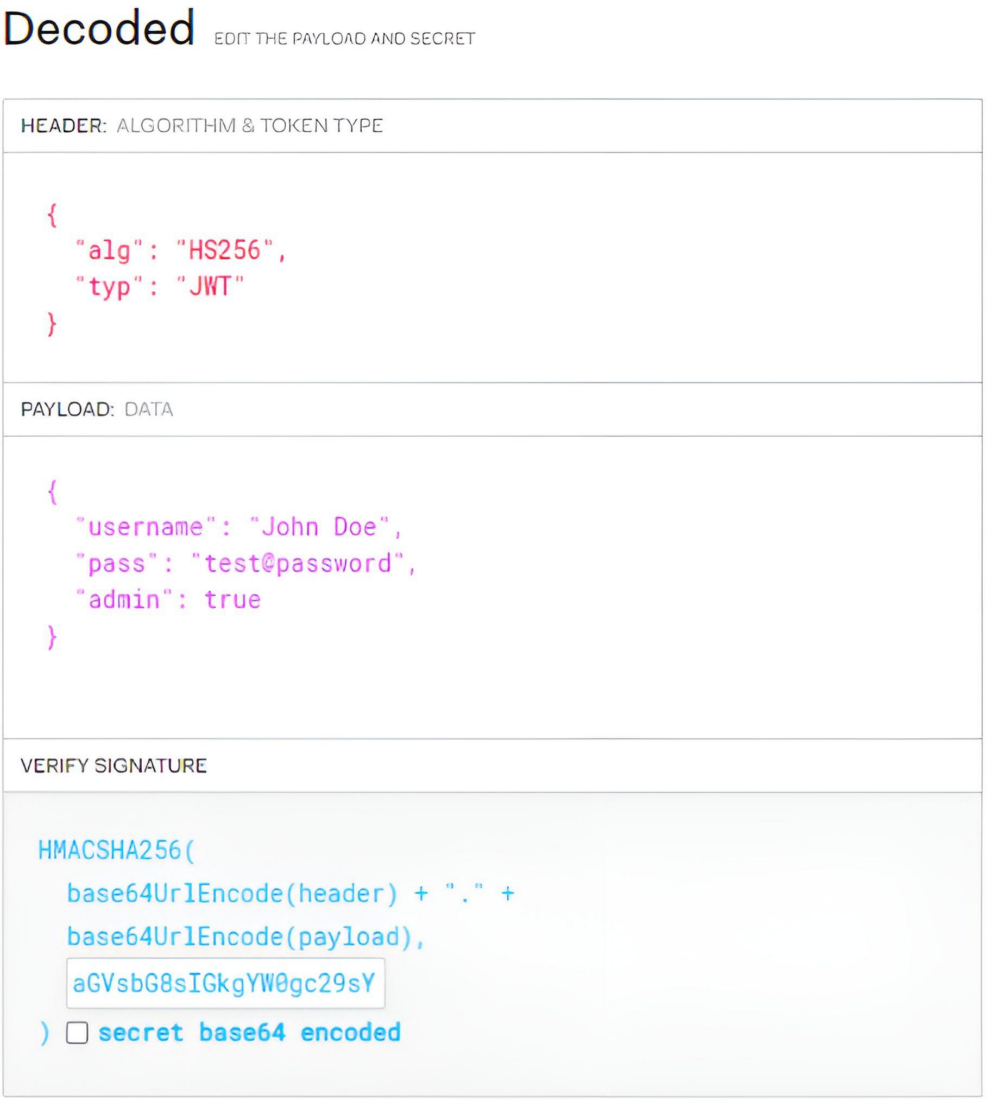


Figure 6. Decoding a JSON Web Token

Similarly, the verification of a JWT token works in reverse on the server side. First, the header part of the JWT token is fetched(generally from the Bearer token in the Authorization schema in a request payload). The header part is then decoded using base64url decoder.

After the header has been decoded, the payload is fetched and depending on the hashing algorithm defined in the “alg” section of the decoded header of the received JWT token, the header and the payload is hashed again using the secret key as used in the encoding process..

Finally, the hashes of the token and the newly calculated are matched together. Along with any claims made in the payload itself. If the hashes do not match, the JWT is rejected. If the claims made in the payload are not checked out (like time exceeds expiry as defined in "expiry" in claims part), the JWT is again rejected.

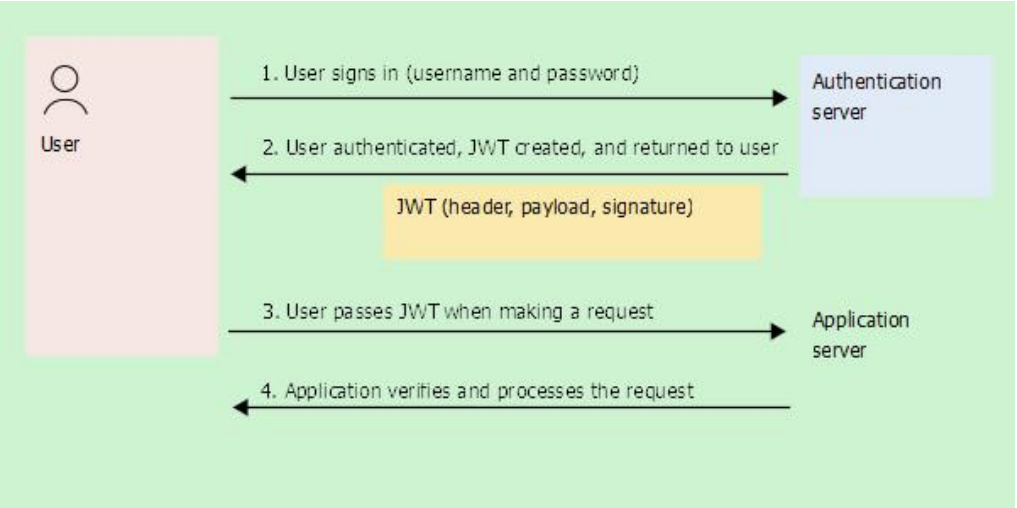


Figure 7. Working of JWT for authentication

3.3 Ethereum Blockchain

Ethereum Blockchain is a transaction based state machine and is one of the major implementations of the blockchain technology. It is an open-sourced, decentralized blockchain with the support of Smart Contracts. Only second to bitcoin in market capitalization, Ether is the official cryptocurrency supported by the Ethereum Blockchain.

Any blockchain implementation like Ethereum is characterized by the following properties:

- Transactional Singleton Machine: There is only a single canonical instance of the blockchain at any given instance of time. This means that there is defined global truth or state that every node(or machine) in the blockchain network partakes in.

- Shared State: All the different nodes in the blockchain network share only a single canonical global state. Any machine that wants to hold the blockchain is open to do so and they can be a partial node or a full node.
- Cryptographically Secure:

Aforementioned, Ethereum is a translation based state machine. Therefore, it receives a series of inputs and goes to the next state which is globally shared with all the nodes in the blockchain network.

With Ethereum's state machine, the state starts with a state analogous to a blank state with no transitions called the "genesis block". Whenever a translation occurs in the blockchain network, the state of the blockchain changes to a different state. At any point in time, the final state is the current state of the blockchain.

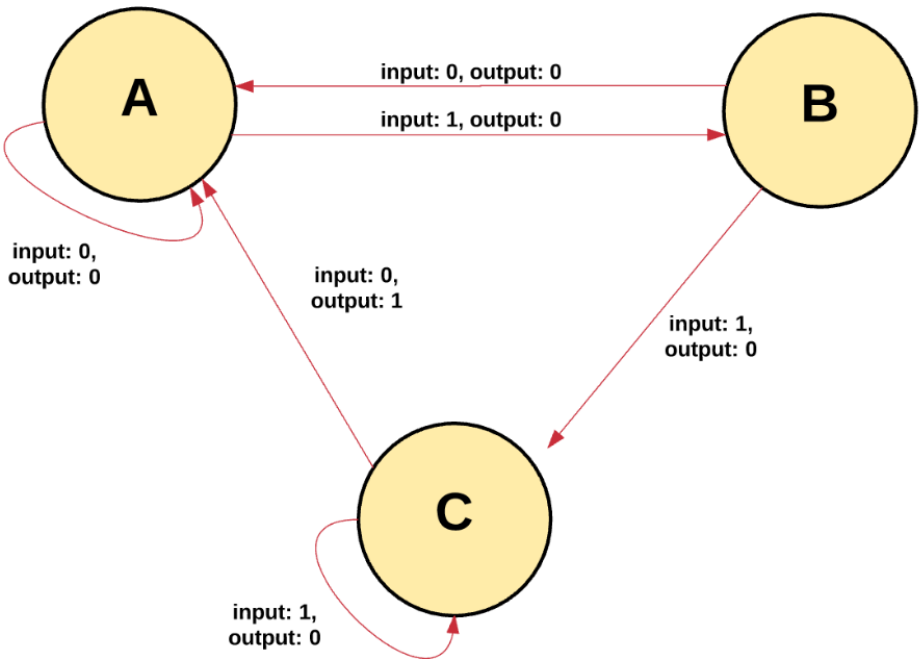


Figure 8. An example of Finite State Machine

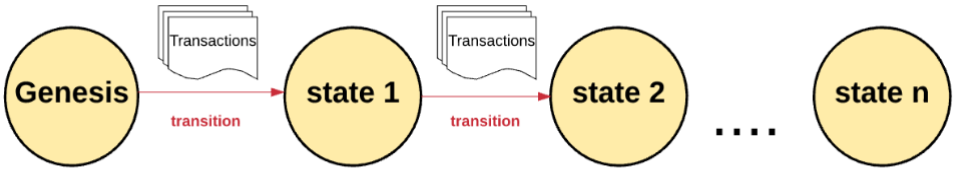


Figure 9. State change from Genesis state to current state

State change in the Ethereum blockchain does not necessarily have to be a single transaction. Thousands of transactions can be bunched together to form blocks. Therefore a block may contain many transactions and each block is chained together with its previous block.

The nature of blocks is immutability. A blockchain ledger's immutability is its capacity to stay unaltered, unchangeable, and irrevocable. A cryptographic principle or a hash value is used to carry out each block of data, such as facts or transaction details. Now, each block independently generates an alphanumeric string for this hash value.

Each block has a hash value or digital signature for both it and the block before it. In turn, this ensures that the blocks are implacably tied together in the past. This feature of blockchain technology ensures that no one can tamper with the system or change the data that has already been saved into the block. A blockchain is therefore referred regarded as being cryptographically secure.

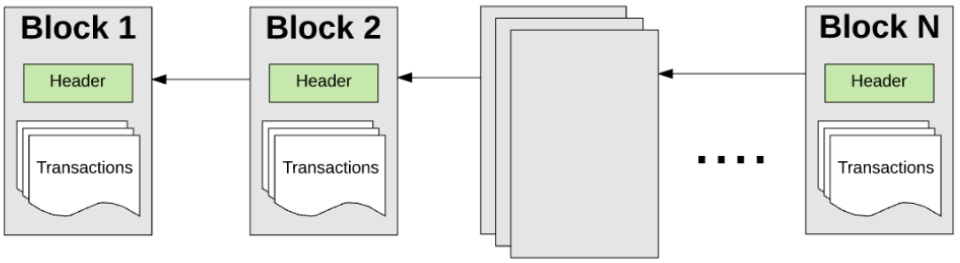


Figure 10. State Change in Blocks in Ethereum

For the block to be added to the blockchain and the state to advance, the transactions in the block must be legitimate. Mining is the process of confirming a block's validity. In mining, a number of blockchain nodes pool their processing power to solve a cryptography hashing puzzle. The miner must demonstrate a block is valid faster than any rival miner in order for it to be added to the main network. "Proof of work" refers to the procedure of validating each block by requesting a mathematical justification from the miner.

Any node that is part of the blockchain has the ability to mine. The miner's responsibility is to build and verify each block in the network. When submitting a block to the blockchain, each miner includes a mathematical "proof" that serves as a guarantee that the block is valid.

The miner who successfully validates the block first by solving the cryptography problem is rewarded with Ether, the official cryptocurrency supported by the Ethereum blockchain.

However, it can be possible that more than one miner validates the block at exactly the same instance. This can result in conflict as a blockchain can only have one global state. This problem is known as Forking and is a common problem in a fast blockchain.

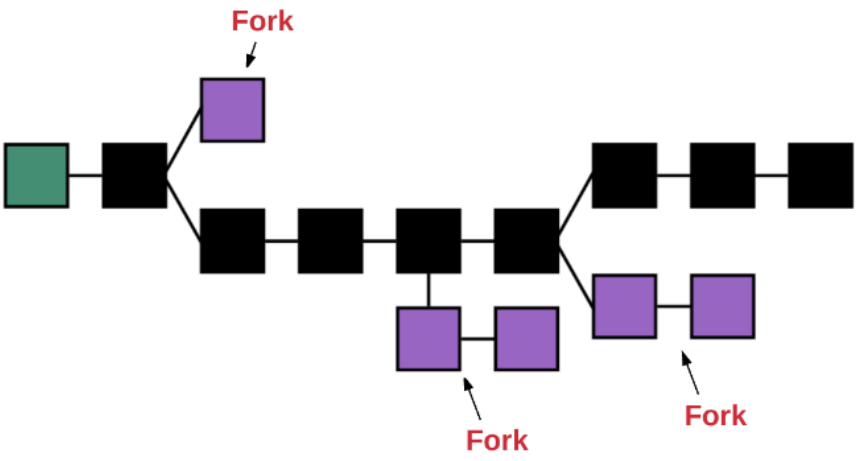


Figure 11. Fork problem in a blockchain

To prevent forking, a blockchain should come down to having only one branch in its global state. Different algorithms can be used for selecting the branch as the global state. For example, Ethereum uses the Greediest Heaviest Observed Subtree abbreviated as the GHOST Protocol. The principle behind the GHOST protocol is to select the branch on which most work has been done. In general, the longest branch has the most computational work done on it and is the one generally selected for included in the canonical global state.

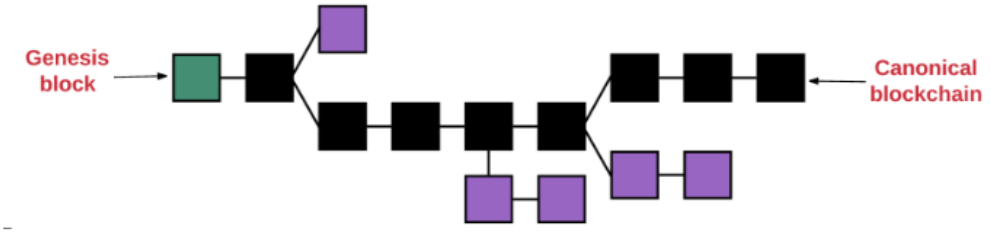


Figure 12. GHOST Protocol

3.4 Ethereum Smart Contract

The Smart Contract was written on the Solidity Programming Language version 0.8.7.

The design pattern for the smart contract was inspired by the openZeppelin library. In the smart contract, a ***Ownable*** contract is defined first. This contract would have functionality related to the admin control of the smart control that includes transferring ownership of the smart control, relinquishing control of the smart contract and so on. This would play an important role in the overall design of the smart contract implementation. Since there are many kinds of users in the project, this would enable access control and ownership of those accesses with transfer and renouncing rights.

```
1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity ^0.8.7;
4
5  contract Ownable{
6      address private origOwner;
7      event TransferOwnership(address indexed oldOwner, address indexed newOwner);
8
9      constructor ()  {
10         origOwner = msg.sender;
11         emit TransferOwnership(address(0), origOwner);
12     }
13
14     function owner() public view returns (address) {
15         return origOwner;
16     }
17
18     modifier onlyOwner(){
19         require(isOwner(), "Only the contract owner can perform this functionality");
20         _;
21     }
22
23     function isOwner() public view returns (bool) {
24         return msg.sender == origOwner;
25     }
26
27     function renounceOwnership() public onlyOwner(){
28         emit TransferOwnership(origOwner, address(0));
29         origOwner = address(0);
30     }
31
32     function transferOwnership(address newOwner) public onlyOwner {
33         _transferOwnership(newOwner);
34     }
35
36     function _transferOwnership(address newOwner) internal {
37         require(newOwner != address(0));
38         emit TransferOwnership(origOwner, newOwner);
39         origOwner = newOwner;
40     }
41 }
```

Figure 13. Implementation of Ownable smart contract

The first owner of the smart contract would be the address that would make the contract deployment call. This is defined in the constructor() of the smart contract Line 9. This first call or contract creation call would also emit a user-defined event call ***TransferOwnership(address, address).***

Modifier functionality of the solidity programming language is used here. Function modifiers are used to change the behavior of a function like adding a prerequisite clause or clean up. They are either called before the function is called or just after the function has been called in this implementation. In the implementation, ***onlyOwner()*** modifier is used in functionality where the

actions can be performed only by the owner itself like in the case of ***transferOwnership(address)*** and ***renounceOwnership()*** functions.

After the ownership part of the smart contract was implemented, different Roles need to be defined. Different actions associated with a role also need to be defined and implemented. Aforementioned, the roles are Original Equipment Manufacturer(OEM) or the manufacturer, the distributor, the retailer and the consumer.

Roles were first defined as a library. Any different roles and their functionalities were implemented after importing from the ***roles*** smart contract.

```
1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity ^0.8.7;
4  library Roles {
5    struct Role {
6      mapping (address => bool) bearer;
7    }
8    function add(Role storage role, address account) internal {
9      require(account != address(0));
10     require(!has(role, account));
11
12     role.bearer[account] = true;
13   }
14   function remove(Role storage role, address account) internal {
15     require(account != address(0));
16     require(has(role, account));
17
18     role.bearer[account] = false;
19   }
20   function has(Role storage role, address account)
21     internal
22     view
23     returns (bool)
24   {
25     require(account != address(0));
26     return role.bearer[account];
27   }
28 }
```

Figure 14. Implementation of roles

In this smart contract, ***Roles*** library was defined first. A library in solidity programming language which provides code reusability while being similar to a smart contract.

Every role defined later would have a mapping from address to boolean along with some functionalities like assigning an address a role, checking if an address has the role and removing an account from its role. Mapping in solidity works like any dictionary implementation where data is stored as a key-value pair and search is generally done on the basis of key values. A boolean value false of a given key for a role would mean that the account does not currently have the role. In the implementation of the Roles smart contract, ***add(Role, address)*** function is used to assign a role to a given address. Similarly, ***remove(Role, address)*** works to remove an account from a given role. ***has(Role, address)*** is used to check if a given address belongs to a given role.

After that, different smart contracts for different roles were defined. Every smart contract thus would inherit the ***ownable*** smart contract already defined and would import from the ***roles*** smart contract to use the functionalities defined in that smart contract.

Starting with the ***OEMRoles*** smart contract, the roles, action and access control of an OEM was defined. Only a manufacturer is allowed to create a listing of products, remove a product and dispatch the product to the next stage of the supply chain. For some functions, modifier need to be used to make sure that only the addresses who are OEM can perform functionalities. This was achieved using the onlyOEM modifier.

```
   modifier onlyOEM(){
        require(isOEM(msg.sender), "Only an OEM can
perform this functionality");
       _;
```

```
}
```

Apart from this, different functions for adding an OEM, removing an OEM and checking if an address is an OEM. All of these functions would make function calls to the ***roles*** smart contract defined earlier.

```
function isOEM(address account)
 public
 view
 returns (
             bool
         )
 {
     return Roles.has(oems, account);
 }

 function addOEM(address account)
 public
 payable
 onlyOEM
 {
     _addOEM(account);
 }

 function _addOEM(address account)
 internal
 {
     Roles.add(oems, account);
     emit OEMAdded(account);
 }
```

```
    function renounceOEM()
    public
    {
        _removeOEM(msg.sender);
    }
    function _removeOEM(address account)
    internal
    {
        Roles.remove(oems, account);
        emit OEMRemoved(account);
    }

}
```

Similarly, the same was implemented for different roles in the ***ConsumerRoles***, ***RetailerRoles*** and ***DistributorRoles*** smart contracts respectively.

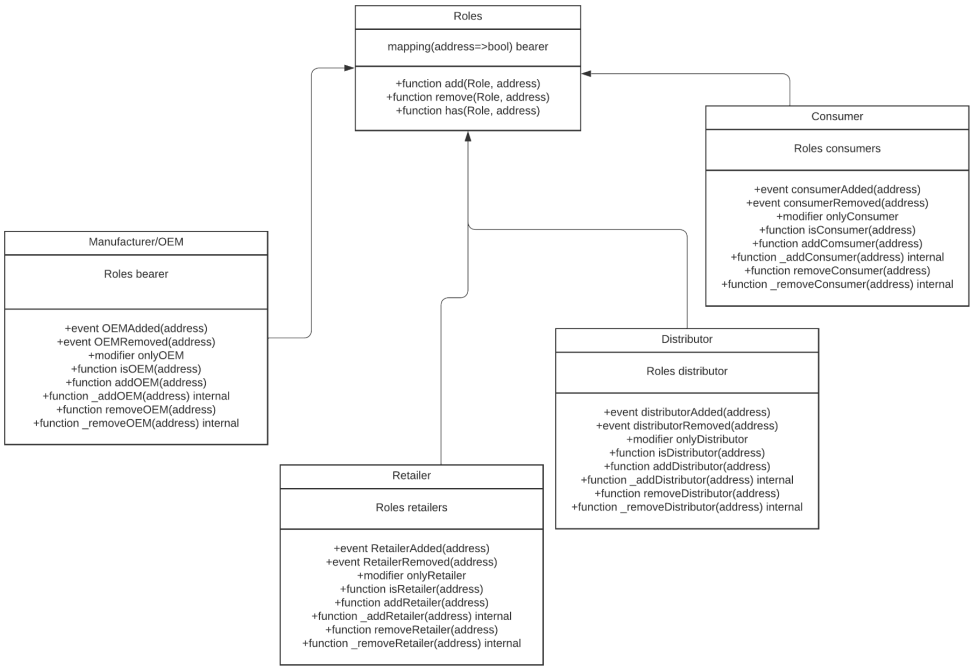


Figure 15. UML Class Diagram for different roles

After that the main smart contract ***SupplyChain*** was implemented. As mentioned, a product in a supply chain goes through different states like Manufactured, Packed, Dispatched to distributor, Dispatched to retailer, Sold to consumer and so on. Their states were represented using an enum in solidity.

```
enum State
{
    MANUFACTURED,           //0
    PACKED,                 //1
    DISPATCHED_DISTRIBUTOR, //2
    RECEIVED_DISTRIBUTOR,   //3
    DISPATCHED_RETAILER,    //4
    RECEIVED_RETAILER,      //5
    SOLD                    //6
}
```

Figure 16. State Structure

Data related to a product and its history also needs to be stored. This was achieved mainly by using the struct and mapping functionality as provided by solidity. An item can be uniquely identified by its UPA or unique product code. UPC is unique for different items for the same product. SKU stands for Stock Keeping Unit and serves the same purpose as UPC. However, SKU is generally used internally by the manufacturer and the distributor whereas UPC is used at the retailer in the supply chain.

The implementation of Item is done using struct as:

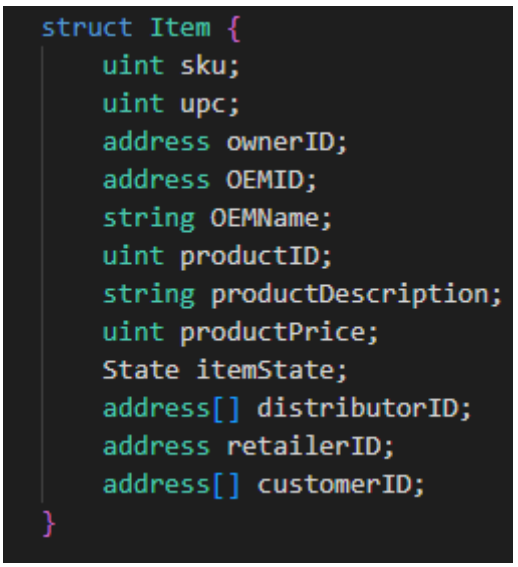


Figure 17. Item Structure

- SKU: Stands for Stock Keeping Unit. Used internally by manufacturer to identify a product
- UPC: Stands for Unique Product Code. Used externally by retailers to uniquely identify a product
- ownerID: The address of the user in the supply chain which holds the ownership to the product and can perform different functionalities to the product in accordance to its role defined.
- OEMName: Name of the product Original Equipment Manufacturer(OEM).
- productID: An integer used to identify a product in the project.
- productDescription: A string containing the description of the product.
- itemState: The current state of the product item in the supply chain.
- distributorID: An array of addresses of different distributors which were in the supply chain cycle of the product.
- retailerID: The address of the retailer of the product.
- customerID: An array of addresses of different customers.

Similarly, the history of a product as it moves through the supply chain also needs to be maintained. This was also done by the use of struct. The implementation is:

```
struct ItemHistory{
    uint status;
    string stateDescription;
}
```

Figure 18 Item History Structure

- status: Denotes the status of the product.
- stateDescription: The description associated with the associated status of the product. This would include the timestamp, location and other data of the state change.

The product item data and the product history data were stored as mapping from uint to Item object and from uint to ItemHistory array which is demonstrated as code snippet below:

```
struct ItemHistory{
    uint status;
    string stateDescription;
}
```

Figure 19. Item History Structure 2

The ER diagram of the above relation is given below:

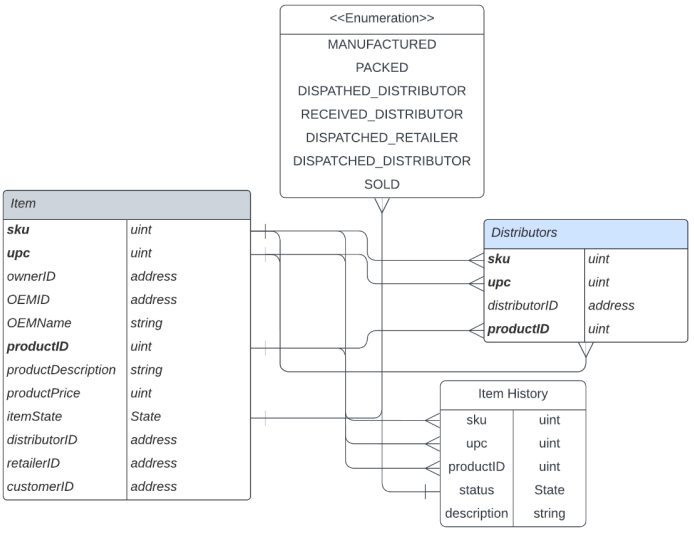


Figure 20. ER Diagram of Objects and Entities

To make sure that the access control of all functions are well-defined depending on roles of a user and to check for the correct state of the product before performing actions on it like changing state and triggering events associated with state change, function modifiers are used whose implementation is given below:

```
92      modifier manufactured(uint _upc){
93          require(items[_upc].itemState == State.MANUFACTURED, "Item has not been manufactured yet");
94          _;
95      }
96      modifier packed(uint _upc){
97      require(items[_upc].itemState == State.PACKED, "Item has not been manufactured yet");
98      _;
99      }
100
101     modifier dispatchedDistributor(uint _upc){
102         require(items[_upc].itemState == State.DISPATCHED_DISTRIBUTOR, "Item has not been sent to distributor yet");
103         _;
104     }
105     modifier receivedDistributor(uint _upc){
106         require(items[_upc].itemState == State.RECEIVED_DISTRIBUTOR, "Item has not been received by distributor yet");
107         _;
108     }
109     modifier dispatchedRetailer(uint _upc){
110         require(items[_upc].itemState == State.DISPATCHED_RETAILER, "Item has not been sent to retailer yet");
111         _;
112     }
113     modifier receivedRetailer(uint _upc){
114         require(items[_upc].itemState == State.RECEIVED_RETAILER, "Item has not been received by retailer yet");
115         _;
116     }
117     modifier soldCustomer(uint _upc){
118         require(items[_upc].itemState == State.SOLD, "Item has not been sold to a retail customer yet");
119         _;
120     }
```

Figure 21. Access Control Implementation

- ***modifier manufactured(uint)*** is used to check where the listing of a product has been made or not before performing subsequent actions related to it.
- ***modifier packed(uint)*** is used to check whether a product has been packed by the manufacturer before doing actions on it like dispatching to a distributor.
- ***modifier dispatchDistributor(uint)*** is used to check if the product has been dispatched to a distributor before performing any related actions on it.
- ***modifier receivedDistributor(uint)*** is used to check if the product has been received by the distributor before performing any subsequent actions like dispatching to a retailer or dispatching to another distributor.
- ***modifier dispatchRetailer(uint)*** is used to assert before performing any subsequent action that the product has been dispatched to the retailer.

- ***modifier receivedRetailer(uint)*** is used to assert before performing any subsequent action that the product has been received by the retailer.
- ***modifier soldCustomer(uint)*** is used to check before running a function that the product has been sold to the customer.

Apart from this, another modifier called ***verifyCaller(address)*** is defined as:

```
75      modifier verifyCaller(address _address){
76          require(msg.sender == _address, "Caller not verified");
77          _;
78      }
```

Figure 22. Verify Caller

The role of the verifyCaller(address) modifier is to make sure that the owner of the product at any given instance of time is the one calling a function that can change the state of the product. This binds the state and actions associated with the product with a address which is the owner of the product at a given time and makes sure that no address other than the owner can call function that changes the state and ownership of the product.

The functionalities associated with a manufacturer are to create a listing for a product, pack a product and dispatch the product. All of these functionalities are implemented in the ***SupplyChain*** smart contract itself. For manufacturing a product, the implemented function is given below:

```
    function manufactureItem(
                        uint _upc,
                        address _OEMID,
                        string memory _OEMName,
                        uint _productID,
                                        string  memory
_productDescription,
                        uint _productPrice,
                        string memory _stateDescription
    )
    public
```

```
    payable
    onlyOEM
    {
        items[_upc].sku = sku;
        items[_upc].upc = _upc;
        items[_upc].ownerID = _OEMID;
        items[_upc].OEMName = _OEMName;
        items[_upc].OEMID =  _OEMID;
        items[_upc].productID = _productID;
                    items[_upc].productDescription  =
_productDescription;
        items[_upc].productPrice = _productPrice;
        items[_upc].itemState = State.MANUFACTURED;
        items[_upc].distributorID.push(address(0));
        items[_upc].retailerID = address(0);
        items[_upc].customerID.push(address(0));
        sku = sku + 1;

        ItemHistory memory hist;
        hist.status = 0;
        hist.stateDescription = _stateDescription;
        itemHistory[_upc].push(hist);
        emit Manufactured(_upc);
    }
```

---

manufactureItem function is defined as payable. This is because the function is adding data to the blockchain and hence it needs to be a payable function. onlyOEM modifier is used to make sure that only the manufacturer is able to call the function. An error is thrown if any address other than that of an OEM tries to create a product listing. As seen in the function implementation, the item data and item history data is initialized. This is followed by an ***Manufactured(uint)*** event.

Similarly, a manufacturer's job is to pack the item before sending it to the next player in the supply chain. This was also implemented as a payable function

```
    function packItem(uint _upc)
        manufactured(_upc)
        onlyOEM()
        verifyCaller(items[_upc].OEMID)
        public
        payable
    {
        items[_upc].itemState = State.PACKED;
        emit Packed(_upc);
    }

         function dispatchToDistributor(uint _upc,
address _distributorAddress)
        packed(_upc)
        onlyOEM()
        verifyCaller(items[_upc].OEMID)
        public
        payable
    {
                              items[_upc].itemState =
State.DISPATCHED_DISTRIBUTOR;
        items[_upc].ownerID = _distributorAddress;
        emit Shipped_To_Distributor(_upc);
    }
```

Similar to the ***manufactureItem()*** function, the ***packItem(uint)*** function is also implemented as a payable function since data is modified in the blockchain. In this function, three function modifiers are used namely

manufactured(uint) to make sure that the product has been listed and has been manufactured by the OEM, ***onlyOEM()*** to make sure before calling the function that only a manufacturer can call the function and change its state and ***verifyCaller(address)*** to make sure that only the manufacturer which is the current owner of the product can change its state. The function call does only change the state key of the item struct to PACKED which is followed by a trigger of the ***Packed(uint)*** event.

Another functionality provided by a manufacturer is to dispatch the product to a distributor. The implementation is given below in the code snippet:

```
  function dispatchToDistributor(uint _upc, address
_distributorAddress)
        packed(_upc)
        onlyOEM()
        verifyCaller(items[_upc].OEMID)
        public
        payable
    {
                            items[_upc].itemState    =
State.DISPATCHED_DISTRIBUTOR;
        items[_upc].ownerID = _distributorAddress;
        emit Shipped_To_Distributor(_upc);
    }
```

The function is, again, payable as it is modifying the contents of data stored on the blockchain. ***packed(uint)*** modifier is used to make sure that the product has been already packed by the manufacturer before calling this function. ***onlyOEM()*** modifier to make sure that only those addresses can call this function that are in the manufacturer role. ***verifyCaller()*** function to make sure that only those manufacturers can make a call to this function who are the original manufacturer of the product and made the product listing.

The function is responsible for changing the state of the product item to DISPATCHED_DISTRIBUTOR and for changing the current owner of the product to the distributor the product has been dispatched to. This means that the ownership of the product changes to the distributor and only a distributor can call any subsequent function that may change the state of the product. The function ends with a call to event ***Shipped_To_Distributor(uint).***

The next player in the supply chain is the distributor. A distributor role in the supply chain is to receive products from a manufacturer in bulk and then redirect the products to another distributor and/or to a retailer. Before doing so, a distributor would receive the product from a manufacturer or another distributor. This is implemented in ***receivedByRetailer(uint)*** function as:

```
    function receivedByDistributor(uint _upc)
        dispatchedDistributor(_upc)
        onlyDistributor()
        verifyCaller(items[_upc].ownerID)
        public
        payable
    {
                            items[_upc].itemState    =
State.RECEIVED_DISTRIBUTOR;
        emit Received_By_Distributor(_upc);
    }
```

This function is also payable as it is changing the state of the data stored in the blockchain. This requires function modifier dispatchedDistributor(uint) to make sure that only after the product has been dispatched that the function can be called, onlyDistributor() modifier to make sure that only a distributor would be able to call this function verifyCaller(uint) to make sure that only the distributor to which the product was dispatched by the manufacturer would be able to call this function. This function changes the state of the product to

RECEIVED_DISTRIBUTOR and does not change the ownership of the product. The execution of the function is followed by triggering of an event called ***Received_By_Distributor(uint)***.

After receiving the product, a distributor may either dispatch the product to another distributor or to a retailer itself. Functions for both of the actions have been implemented. The function for dispatching to a retailer is defined as ***dispatchToRetailer(uint, address)*** and in ***dispatchToDistributor2(uint, address)*** respectively the implementations to which are given below:

```
      function dispatchToRetailer(uint _upc, address
_retailerAddress)
        receivedDistributor(_upc)
        onlyDistributor()
        verifyCaller(items[_upc].ownerID)
        public
        payable
    {
                              items[_upc].itemState  =
State.DISPATCHED_RETAILER;
            items[_upc].ownerID = _retailerAddress;
            emit Shipped_To_Retailer(_upc);
    }
```

```
     function    dispatchToDistributor2(uint    _upc,
address _distributorAddress)
        receivedDistributor(_upc)
        onlyDistributor()
        verifyCaller(items[_upc].ownerID)
        public
        payable
```

```
    {
                                items[_upc].itemState    =
State.DISPATCHED_DISTRIBUTOR;
        items[_upc].ownerID = _distributorAddress;
        emit Shipped_To_Distributor(_upc);
    }
```

These two functions are also payable since they are changing the data stored in the blockchain. The function modifiers used are ***receivedDistributor(uint)*** to make sure that the product has been received by the distributor before it has been dispatched to a retailer or a distributor. onlyDistributor() to make sure that only a distributor can call this function and ***verifyCaller(address)*** to make sure that only the distributor with which the current ownership of the product lies can call this function. This function would change the ownership of the product to a retailer or a distributor and the ownerID of the product to that of the retailer or a distributor followed by a trigger to ***Shipped_To_Retailer(uint)*** or ***Shipped_To_Distributor(uint)*** event depending which of the two functions were called.

From a retailer, a product can be sold in the primary market i.e. to a customer. This was facilitated in the project. The code snippet for the same is given below:

```
     function   soldByRetailer(uint   _upc,   address
_customerAddress)
        receivedRetailer(_upc)
        onlyRetailer()
        verifyCaller(items[_upc].ownerID)
        public
        payable
    {
        items[_upc].itemState = State.SOLD;
        items[_upc].ownerID = _customerAddress;
```

```
        emit SoldPrimary(_upc);
    }
```

Again this function would take the upc of the product to be sold and the Ethereum address of the customer to which the product is being sold to. Along with this, function modifiers are used for access control. ***receivedRetailer(uint)*** is used to assert that the product has been sold only after it has been received by the retailer. ***onlyRetailer()*** is to make sure that only a retailer can perform this functionality and ***verifyCaller(address)*** is to make sure that no ethererum address other than that of the retailer which holds the current ownership of the product can change its state and ownership. This function changes the ownership of the product to that of the customer followed by a trigger to ***SoldPrinary(uint)*** event.

3.5 NodeJS Backend

NodeJS is used as a backend for the project. A backend framework is necessary to facilitate a couple of this. For one, the NodeJS backend is responsible for making web3 calls to the different smart contract functions that have been implemented so far. In this case, therefore, the NodeJS backend would act as an intermediary between the user of the program and the smart contract. All of the functionality implemented in the smart contract has been wrapped by different REST API calls.

Apart from providing an interface between the smart contract functions and the user, the NodeJS backend is also used to communicate with a centralized server. MongoDB is used as a centralized server. The need for a centralized server arises because of the fact that there can be different kinds of users(manufacturer, distributor, retailer, customer) present in the system. Therefore authentication and authorization of those users needs to be done. The user details would have to be stored in a centralized database that would be used later on in logging a user in the system and fetching information

related to the logged-in user. Any data related to product like description and product history are stored in the blockchain.

The mongoDB schemas of different users is given below:

```
const mongoose = require('mongoose');

const customerSchema = new mongoose.Schema({
    mail: {
        type: String,
        unique: true
    },
    username: {
        type: String,
        unique: true,
    },
    password: {
        type: String,
    },
    address: {
        type: String,
    },
    ownedProducts: [{
        type: String,
    }]
});

module.exports = mongoose.model(
    "customers",
    customerSchema
);
```

Figure 23. Schema of Customer

```
const mongoose = require('mongoose');

const distributorSchema = new mongoose.Schema({
    mail: {
        type: String,
        unique: true
    },
    username: {
        type: String,
        unique: true,
    },
    password: {
        type: String,
    },
    address: {
        type: String,
    },
    productDistributed: [{
        type: String,
    }]
});

module.exports = mongoose.model(
    "distributor",
    distributorSchema
    );
```

Figure 24 Distributor

```
const mongoose = require('mongoose');

const manufacturerSchema = new mongoose.Schema({
    mail: {
        type: String,
        unique: true
    },
    username: {
        type: String,
        unique: true,
    },
    password: {
        type: String,
    },
    address: {
        type: String,
    },
    manufacturedProducts: [{
        type: String,
    }]
});

module.exports = mongoose.model(
    "manufacturer",
    manufacturerSchema
);
```

Figure 25 Schema of Manufacturer

```
const mongoose = require('mongoose');

const retailerSchema = new mongoose.Schema({
    mail: {
        type: String,
        unique: true
    },
    username: {
        type: String,
        unique: true,
    },
    password: {
        type: String,
    },
    address: {
        type: String,
    },
    productsSold: [{
        type: String,
    }]
});

module.exports = mongoose.model(
    "retailer",
    retailerSchema);
```

Figure 26 Retailer

The UML class diagram of the implemented API is given below:

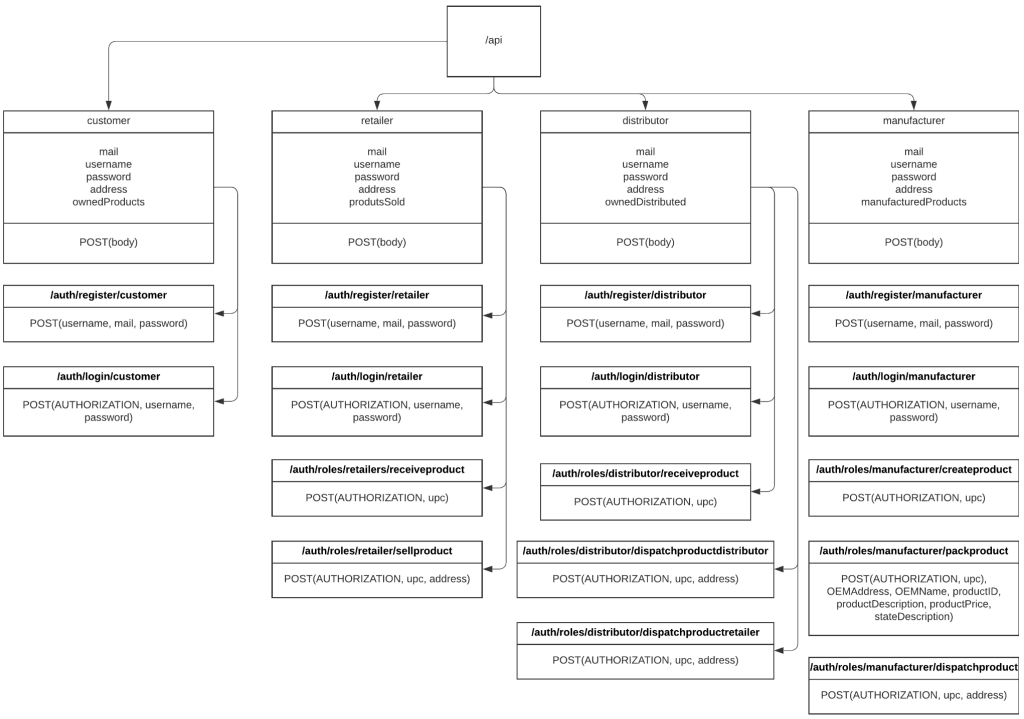


Figure 27. UML Diagram of API Endpoints

Chapter 4: PERFORMANCE ANALYSIS

To evaluate the performance of our system we first need to understand the term “gas”.

The term "gas" refers to the metric used to express the amount of computational power necessary to carry out particular activities on the Ethereum network.

Each Ethereum transaction has a cost since they all need computing resources to complete. The charge needed to complete a transaction on Ethereum is referred to as "gas."

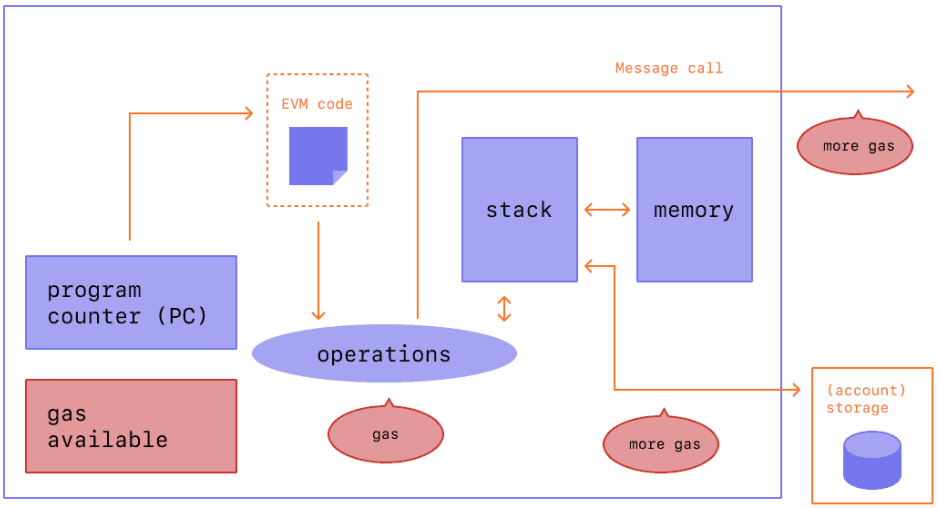


Figure 28. Ethereum Transactions

Gas (ETH) is purchased using ether, the cryptocurrency that runs Ethereum. Each gwei, which serves as a unit of currency for ETH and is used to represent petrol prices, is equal to 0.000000001 ETH (10-9 ETH). You may say, for instance, that your petrol costs 1 gwei as opposed to 0.000000001 ether. A giga-wei, or 1,000,000,000 wei, is denoted by the word "gwei" itself. Wei itself, the lowest ETH unit, is named after Wei Dai, the creator of b-money.

Block Size: Each block's intended size is 15 million gas, however actual block sizes may vary from that up to the block limit of 30 million gas depending on

network demand. The protocol is able to achieve an equilibrium block size of 15 million on average by using tâtonnement. This means that if the block size exceeds the intended block size, the protocol will increase the base price for the following block. Similar to this, the protocol will reduce the base fee if the block size is smaller than the intended block size. Depending on how far off target the current block size is, the basic fee is adjusted.

Base Fee: Each block has a base cost that acts as a reserve price. To be included in a block, the offered price per petrol must at least match the base fee. Because the base cost is set independently of the current block and is determined by the blocks before it, transaction fees are more predictable for customers. During block mining, this base fee is "burned," or removed from circulation.

The basic charge is calculated using a formula that contrasts the size of the preceding block (the amount of petrol utilised for all transactions) with the desired size. The base charge will increase by a maximum of 12.5% per block if the target block size is surpassed. Due to this exponential growth, maintaining a large block size is no longer commercially viable.

Table 2. Working transactions in an Ethereum blockchain

Block Number	Included Gas	Fee Increase	Current Base Fee
1	15M	0%	100 gwei
2	30M	0%	100 gwei
3	30M	12.5%	112.5 gwei
4	30M	12.5%	126.6 gwei
5	30M	12.5%	142.4 gwei
6	30M	12.5%	160.2 gwei
7	30M	12.5%	180.2 gwei
8	30M	12.5%	202.7 gwei

According to the table above, a wallet will advise the user that the maximum base cost that may be raised to the following block in order to create a transaction on block number 9 is the "current base fee * 112.5% or 202.8 gwei * 112.5% = 228.1 gwei."

Max Fee: Users have the option of setting a maximum amount they are willing to pay for a network transaction. This opportunistic parameter is known as maxFeePerGas. For a transaction to be processed, the maximum fee must be greater than the sum of the base cost and the tip. The transaction sender is given a refund for the discrepancy between the maximum fee and the sum of the base fee and tip.

The performance of the project can be measured in two ways. One is to calculate the cost(in gas) of every payable function in the smart contract. Another is to calculate the latency of the different endpoints of the REST API. Both have been done. It should be noted that both NodeJS server and the ethereum blockchain are running on the local machine. The Ethereum smart contract was compiled using truffle and deployed on Ganache providing local hosting of ethereum blockchain.

The table below shows the cost in gas of different functionalities:

Table 3. Cost of different smart contract function calls

Actions	**Cost(in gas)**	**Cost(in ETH)**
Creating Manufacturer	46306	78.49
Creating Customer	46307	78.49
Creating Distributor	46240	78.39
Creating Retailer	46239	78.39
Creating Product Listing	293117	495.95

Pack Product	46869	79.28
Dispatching to Distributor	38368	64.72
Received By Distributor	31892	53.8
Dispatch to retailer	38368	64.72
Received by retailer	32626	55.02
Sold to customer	38390	64.85

Another performance measure for testing the project was to check the latency for each API call of the NodeJS backend. Calls were made using an open-source tool called Postman

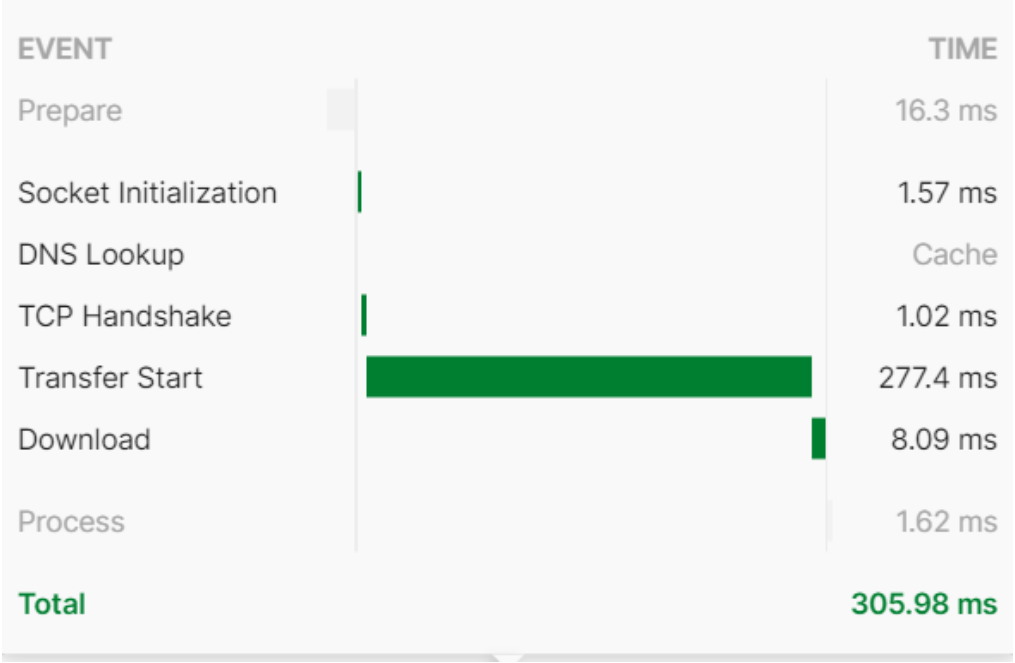


Figure 29. Latency for logging in a customer

Figure 30. Latency for registering a customer

Figure 31. Latency for registering a manufacturer

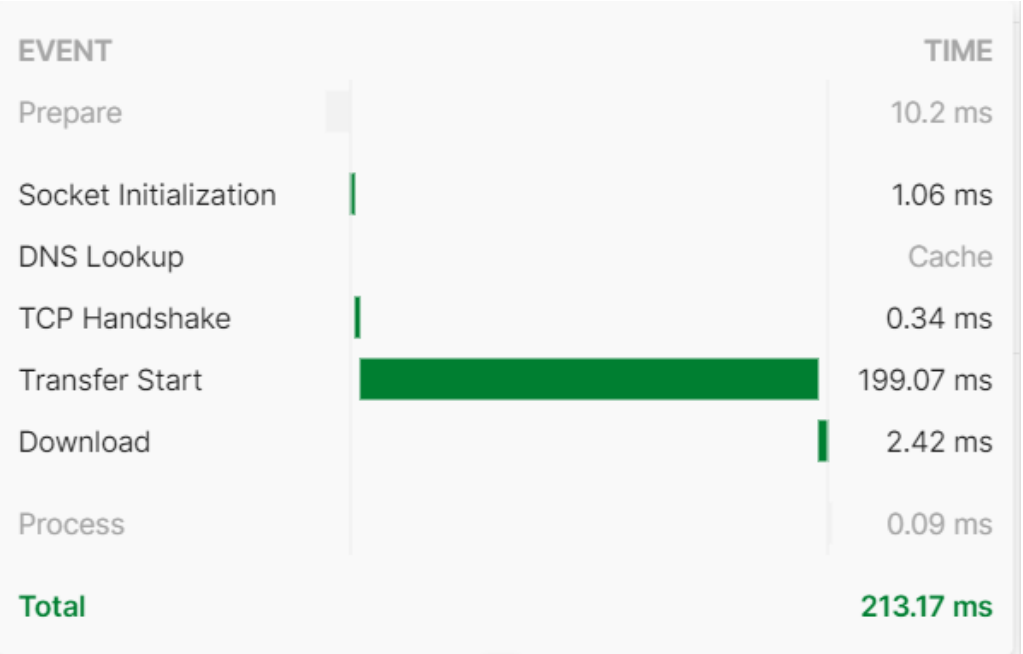


Figure 32. Latency for logging in a manufacturer

Figure 33. Latency for registering a distributor

Figure 34. Latency for /api/roles/manufacturer/createproduct API call

Figure 35. Latency for /api/roles/manufacturer/packproduct API call

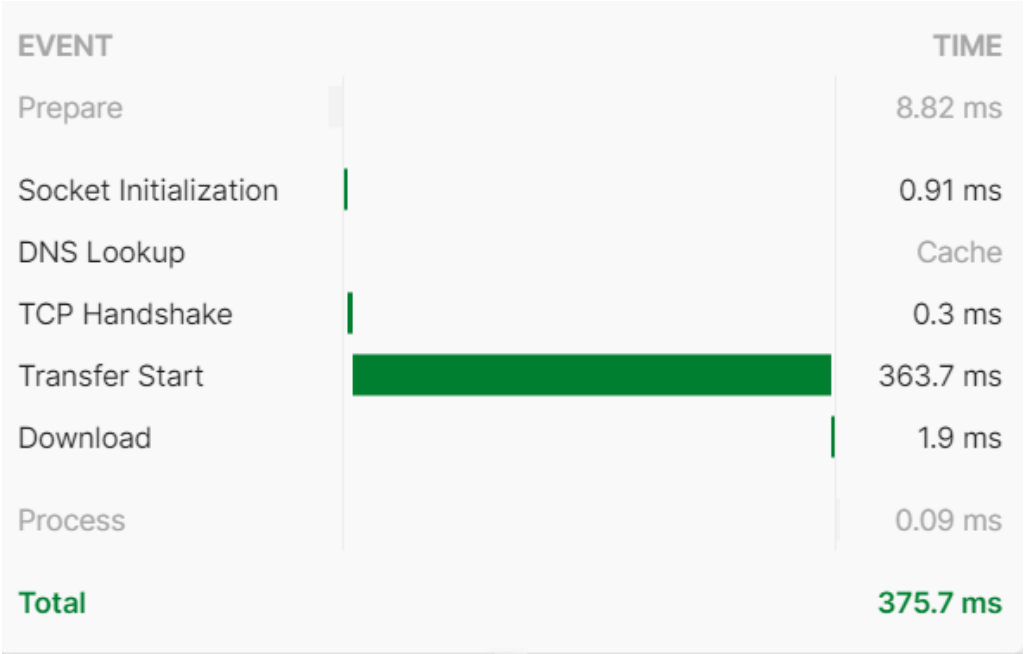


Figure 36. Latency for /api/roles/manufacturer/dispatchproduct API call

Figure 37. Latency for /api/roles/distributor/receiveproduct API call

Figure 38. Latency for /api/roles/distributor/dispatchproductretailer API call

Figure 39. Latency for /api/roles/retailer/receiveproduct API call

Figure 40. Latency for /api/roles/retailer/sellproduct API call

Chapter 5: CONCLUSIONS

5.1 Conclusions

With the increasing market size of counterfeit products the need to counter the supply and distribution of such products is the need of the hour. Giving the consumer the ability to verify the product authenticity can majorly help reduce the circulation of counterfeit products. Such systems, although already in place for certain products are still very vulnerable due the nature of them being centralized, this is where our proposed system for tracking the products by storing and updating the details of origin, transfer and sale on the blockchain is a new step in creating a safe, secure and transparent ecosystem for both manufacturers and consumers.

A system for monitoring ownership that is based on distributed ledger technology using blockchain. The DApp created here guarantees increased security and transparency in the supply chain and can be relied upon for use in both local markets and e-commerce.

The fact that the system makes use of QR codes and user authentication at every step in the supply chain hence helping verify each transaction and eliminate the risk of counterfeits being put in at any instance of the product delivery. Besides, the cost for enrolling each product in the proposed model is suggested to be kept minimal so as to encourage more and more companies to make use of it.

5.2 Future Scope

This project can further be expanded making use of the database built upon and stored on the blockchain of all the registered manufacturers, distributors, sellers as well as customers which can help the market in many ways such as,

- Grading the manufactures
- Trust rating for distributors as well as sellers

- Help the customers know about the product better before making the purchase
- Help recommend products to the customers based on their previous brand interactions
- Increasing the complexity of the smart contract by adding functionalities like returning a product.
- Create a front-end for the NodeJS backend and to add different automation techniques like the use of QR to automate different functionalities.

5.3 Applications Contribution

The following contributions can be drawn form the conclusion,

- We aim to counter the supply and distribution of counterfeit products
- Protect the rights of the consumer by empowering them with the knowledge of authenticity of their purchase
- Creation of an ownership tracking system DApp on Ethereum blockchain
- Promoting transparency and security in the product supply chain

REFERENCES

[1]https://www.theguardian.com/fashion/2022/may/10/spot-the-difference-the-invincible-business-of-counterfeit-goods#:~:text=According%20to%20some,in%20two%20decades.

[2] Anjum, N. and Dutta, P., 2022, January. Identifying Counterfeit Products using Blockchain Technology in Supply Chain System. In 2022 16th International Conference on Ubiquitous Information Management and Communication (IMCOM) (pp. 1-5). IEEE.

[3] Y. Dabbagh, R. Khoja, L. AlZahrani, G. AlShowaier and N. Nasser, "A Blockchain-Based Fake Product Identification System," 2022 5th Conference on Cloud and Internet of Things (CIoT), 2022, pp. 48-52, doi: 10.1109/CIoT53061.2022.9766493.

[4] Sonali P. Chitalkar, Manali B. Khurud, Tejaswini L. Tambe, Madhavi G. Varpe, and S. Y. Raut, "Fake Product Detection Using Blockchain Technology," International Journal Of Advance Research And Innovative Ideas In Education, vol. 7, no. 4, pp. 314-319, Jul-Aug 2021. [Online]. Available: http://ijariie.com/AdminUploadPdf/Fake_Product_Detection_Using_Blockchain_Technology_ijariie14881.pdf [Accessed : 03 July 2022].

[5] Jadhav, R., Shaikh, A., Jawale, M.A., Pawar, A.B. and William, P., 2022, June. System for Identifying Fake Product using Blockchain Technology. In 2022 7th International Conference on Communication and Electronics Systems (ICCES) (pp. 851-854). IEEE.

[6] https://phys.org/news/2019-03-counterfeit-pirated-goods-global.html

[7]https://www.statista.com/statistics/1117921/sales-losses-due-to-fakegood-by-industry-worldwide

[8] T. J. Sayyad, ”Fake Product Identification Using Blockchain Technology,” in International Journal of Future Generation Communication and Networking, vol. 14, pp. 780-785, 2021, ISSN: 2233-7857 IJFGCN

[9] T. Tambe, S. Chitalkar, M. Khurud, M. Varpe, S. Y. Raut, ”Fake Product Detection Using Blockchain Technology,” in International Journal of Advance Research, Ideas and INNOVATIONS in Technology, vol. 7, pp. 314-319, 2021, IJARIIE-ISSN(O)-2395-4396

[10] J. Ma, S. Lin, X. Chen, H. Sun, Y. Chen and H. Wang, ”A Blockchain-Based Application System for Product Anti-Counterfeiting,” in IEEE Access, vol. 8, pp. 77642-77652, 2020, doi: 10.1109/ACCESS.2020.2972026.

[11] K. Toyoda, P. T. Mathiopoulos, I. Sasase and T. Ohtsuki, ”A Novel Blockchain-Based Product Ownership Management System (POMS) for Anti-Counterfeits in the Post Supply Chain,” in IEEE Access, vol. 5, pp. 17465-17477, 2017, doi: 10.1109/ACCESS.2017.2720760.

[12] Y. P. Tsang, K. L. Choy, C. H. Wu, G. T. S. Ho and H. Y. Lam, ”Blockchain-Driven IoT for Food Traceability With an Integrated Consensus Mechanism,” in IEEE Access, vol. 7, pp. 129000-129017, 2019, doi: 10.1109/ACCESS.2019.2940227.

[13] S. Anandhi, R. Anitha and S. Venkatasamy, ”RFID Based Verifiable Ownership Transfer Protocol Using Blockchain Technology,” 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2018, pp. 1616-1621, doi: 10.1109/Cybermatics 2018.2018.00270.

[14] S. Rahmadika, B. J. Kweka, C. N. Z. Latt and K. Rhee, ”A Preliminary Approach of Blockchain Technology in Supply Chain System,” 2018 IEEE International Conference on Data Mining Workshops (ICDMW), 2018, pp. 156-160, doi: 10.1109/ICDMW.2018.00029. [10] https://coinmarketcap.com/converter/eth/usd/

APPENDICES

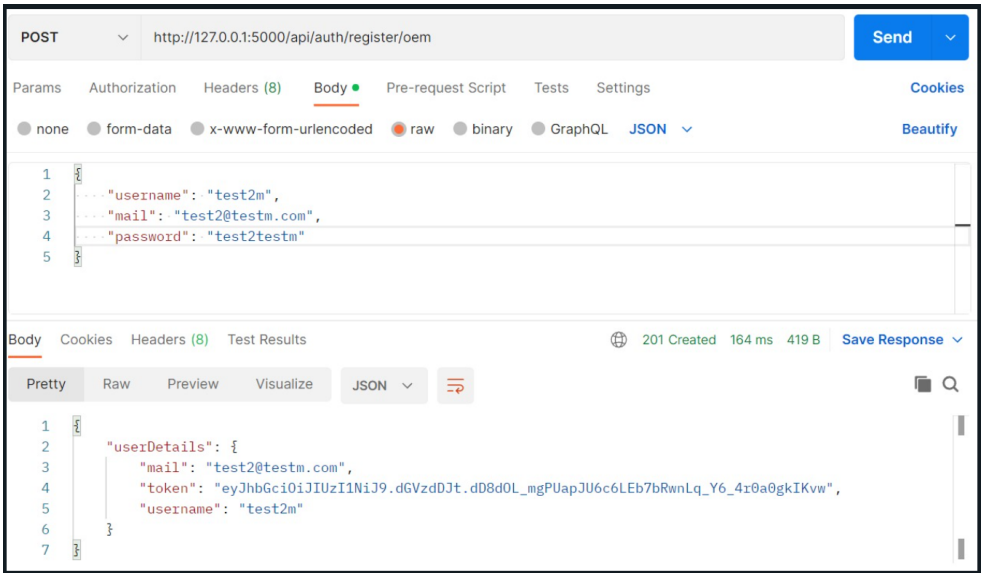


Figure 41. Implementation of registration

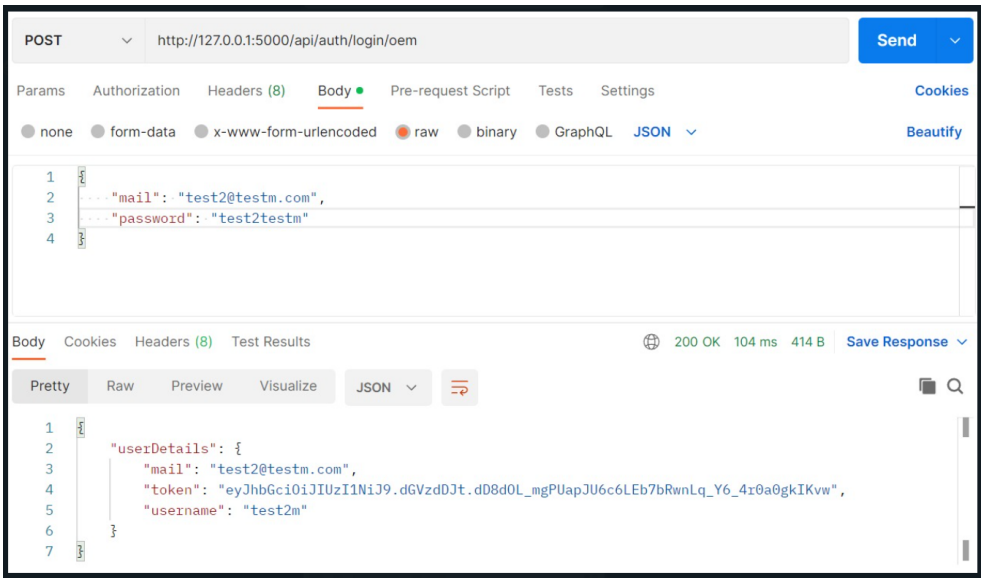


Figure 42. Implementation of login

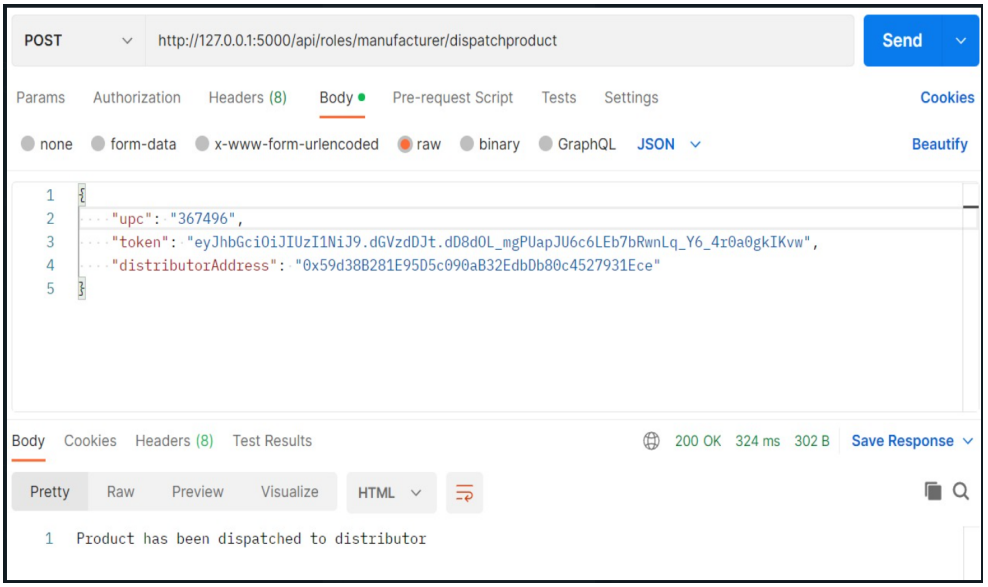


Figure 43; Implementation of Ownership Transfer

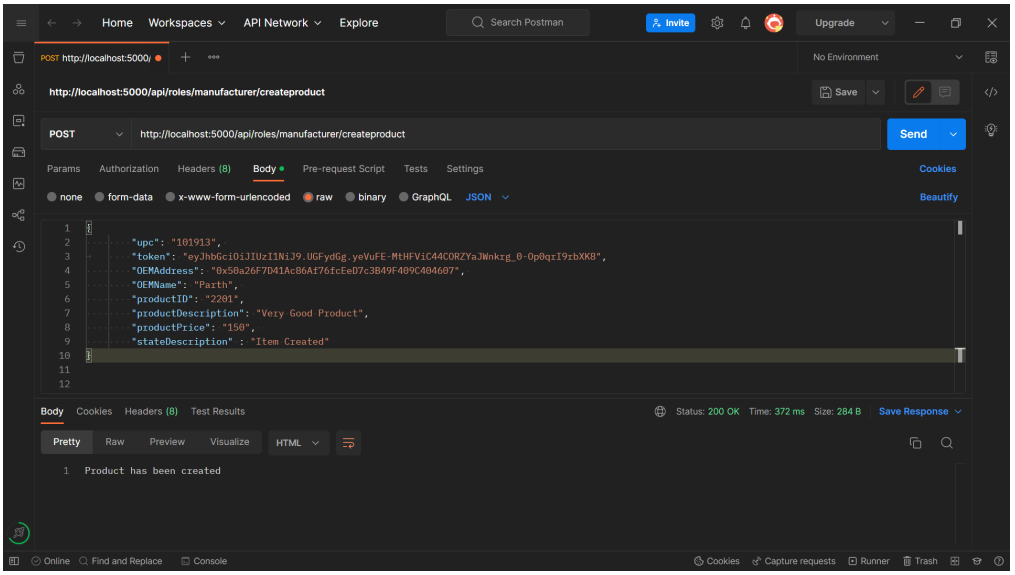


Figure 44; Implementation of Product Creation