Design and Implementation of Blockchain Collaborated Cloud of Things: A Covid Vaccine Supply Chain Management System

Ranu Pandey,
Research Scholar, Shri rawatpurasarkar University Raipur Chhattisgarh, India
Dr. Rajesh Kumar Pathak ,
Vice Chancellor ,OPJSUniversity Rajasthan ,India

Abstract In recent times, there has been a significant focus on Blockchain and Cloud of Things (CoT), which combines blockchain, cloud computing and the Internet of Things (IoT) to offer valuable services in various technical applications. The integration of blockchain and CoT addresses challenges related to network security, data privacy, and decentralization in CoT, while CoT provides scalability and fault tolerance to enhance the effectiveness of blockchain functions. Hence, this paper presents a conceptual framework design of the BCoT system and outlines the overall structure and principles behind its integration. It provides a high-level view of how blockchain and CoT interact and collaborate to deliver enhanced functionalities andthe PoC architecture of the BCoT enabled Covid Vaccine Supply Chain Management System. It illustrates a practical implementation to validate its functionality.It demonstrates how the different components and layers of the system are interconnected and interact during real-world scenarios.It explores the possibilities and benefits of the proposed framework, utilizing diverse performance metrics. The results and test analysis findings underscore the significance of the proposed work and validate its potential for effectively addressing the challenges of existing system.

KeywordsBlockchain, BCoT integration, Cloud of Things, covid vaccine Supply Management system

1 Introduction

In recent years, blockchain technology has gained significant attention due to its decentralized and enhanced security features. It is being widely adopted in various commercial and financial service sectors, impacting different aspects of human life [1]. Blockchain is a distributed, public, and immutable database that secures transactions through a peer-to-peer (P2P) system architecture. Transaction data is decentralized and not controlled by any central entity. The data is stored in a series of blocks accessible publicly to all participants in the blockchain network. Cryptography and consensus mechanisms validate the legitimacy of transactions, ensuring the resistance of connected blocks to alterationsand modifications [1]. The transparency, immutability, and security provided by blockchain enhance service efficacy, leading to its widespread adoption across diverse sectors.

Alongside blockchain, cloud computing (CC) and the Internet of Things (IoT) are other prominent technologies ruling the world today. IoT is a network of physical things connected via the internet, enabling smart industrial services such as smart cities, smart manufacturing, and more. However, the limited resources of IoT devices have led to the utilization of cloud computing in the form of Cloud of Things (CoT) [2][3]. CoT leverages cloud environments to handle IoT services, offering improved efficiency and performance [4]. However, classical CoT infrastructures face several challenges. Firstly, they rely on centralized frameworks, making scalability difficult for massive IoT networks [5]. Secondly, many CoT systems require third-party trust for IoT data processing, posing privacy concerns. Lastly, the centralized system infrastructure can result in high power consumption and communication latency issues for IoT devices, limiting the practical deployment of CoT [6].

To address these challenges and foster the sustainable growth of CoT, the construction of a highly decentralized ecosystem has become crucial. Blockchain, as a robust candidate for achieving complete decentralization, can revolutionize CoT systems when combined in the form of Blockchain

and CoT (BCoT) [6]. BCoT offers a decentralized storage platform using virtual storage, enabling new and highly secure cloud storage operations that are resistant to information modifications. BCoT interconnects virtual machines and external computers on the cloud, constructing a fully decentralized storage network without relying on traditional data centers. This concept resolves many issues faced by standalone CoT or blockchain systems. BCoT also provides numerous potential benefits, including enhanced privacy, improved security, better cooperation, decentralization, fault tolerance, and scalable transaction support [6].

1.1 Related Studies

Several studies have focused on exploring the potential of Cloud of Things (CoT), blockchain, and related topics. Various survey works have been conducted to provide comprehensive overviews of the research efforts in these areas. For example, surveys conducted by researchers [7-9] have examined the utilization of blockchain platforms in different IoT applications and scenarios. Another study [10,24] discussed the integration of blockchain with IoT, specifically exploring its capabilities in heterogeneous applications ranging from intelligent manufacturing to smart vehicles, drones, and 5G communication networks. Furthermore, a study [11] focused on the utilization of blockchain for providing security services and discussed its technical characteristics in addressing challenges in diverse application domains, including cloud computing and IoT. Technical concepts related to blockchain, such as inherent principles, consensus mechanisms, and networking strategies, were elaborated in another study [12]. Additionally, the integration of edge computing and blockchain was explored in an integrated framework in a study [13]. A comprehensive examination of technical issues, opportunities, and challenges associated with the integration of cloud computing and blockchain was conducted in [14].

2. Background

This section provides background information on blockchain technologies and Cloud of Things (CoT). It explains the fundamental concepts of these technologies and highlights their significance in modern applications.

2.1 Blockchain

Blockchain is a decentralized, public, and trusted ledger based on a peer-to-peer (P2P) network. It allows nodes to verify and authenticate transactions, ensuring secure and robust operations with the advantages of tamper resistance and resilience against single-point failures. There are two main classes of blockchain: consented and consent-less. Private blockchains are governed by a central authority, requiring permission from participants for transaction submission, while public blockchains are open for all to participate in transactions and consensus processes. Popular blockchain platforms include Bitcoin, Ethereum, and Hyperledger [15].

The construction of a blockchain network involves key components such as data blocks, smart contracts, a distributed ledger, and consensus mechanisms. Data blocks contain multiple transactions and are connected to neighboring blocks through a hash function, ensuring the integrity of the data. The distributed ledger is a replicated database shared among participants in a peer-to-peer network. Consensus mechanisms enable agreement on individual blocks among multiple nodes, ensuring security within the blockchain system. Smart contracts, which are programmable applications, execute predefined contractual terms such as licenses, confidentiality agreements, and payment conditions [16].

The decentralized nature of blockchain brings several advantages, including cost savings, elimination of single-point failure risks, and improved trustworthiness. Transparency is another important characteristic of blockchain, as all transaction data is visible to all participants in the network. This allows blockchain users to access, validate, and trace transaction activities, enhancing transparency and accountability within the system.

Table 1 Prevalent blockchain Platforms

References	Blockchain platforms	Blockchain class	Machine-oriented language	Consensus algorithm	Transaction speed	Smart contract

[17]	Quorum	Private	Solidity	Istanbul BFT, Raft	~100 TPS	Yes
[17]	Ripple	Private	C++	Probabilistic voting	~1500 TPS	No
[17]	Hyperledger iroha	Private	C++	YAC algorithm	≤1000 TPS	Yes
[17]	Corda	Private	Java, Kotlin	Pluggable consensus	~170 TPS	Yes
[17]	Hyperledger sawtooth	Private/Public	C++, Javascript, Java, Go, Python, Rust	PoET, PBFT, Raft	>1000 TPS	Yes
[17]	Ethereum	Public	Solidity	PoW	~20 TPS	Yes
[17]	Hyperledger Fabric	Private	Javascript, Java, Go	Raft, Kafka, Solo	>2000 TPS	Yes

2.2 Cloud of Things (CoT)

Nowadays, Internet of Things (IoT) has become an integral part of various automated applications, gaining significant attention from both industry and academia. It enables the seamless interconnection of diverse objects and devices, creating a physical platform where processing, sensing, and interaction can take place intelligently without manual intervention. However, the massive amount of data generated by IoT devices has become a challenge in terms of storage resources and the limited power of IoT devices. To address this, the abundant computational power and storage capacity of Cloud Computing (CC) can provide efficient and robust services for IoT applications [17]. The integration of CC with IoT has led to the emergence of a new paradigm called Cloud of Things (CoT), which empowers both CC and IoT domains. By leveraging the resources available in the cloud, CoT can transform existing IoT service frameworks by enhancing service availability, improving system performance, and reducing management efforts. CoT

enables immediate service provisioning to users anytime and anywhere. Additionally, the virtual processing capabilities of CC enable CoT to optimize IoT computations by remotely executing data and facilitating data offloading, thereby addressing bandwidth and energy conservation challenges in IoT. CoT leverages virtual machines, resource infrastructure, and cloud servers to provide automated and simplified solutions. The management frameworks available in the cloud support seamless interconnections and communications between users, devices, and IoT components, enabling pervasive applications.

3Some prevalent CoT platforms and middlewares for CoT are listed in Table II and Table III.

Table 2 Prevalent CoT Platforms

CoT Platforms	**Features**	**Software**
ThingSpeak	Real-time information collection, information processing, apps, visualizations and plugins	Ruby

AWS IoT	Provides cloud services for connecting IoT devices with other devices	C++, python, java, node.js
CloudPlugs	Scalability, security, data and application management	Java, javascript, python, node.js, php
OpenIoT	Semantic interoperability, Integration of applications and IoT data within cloud infrastructures	Java, python
Evrythng	Provides a permanent and unique digital identity for every individual product and enables legitimate users and applications to access it.	Javascript, java
Nimbits	Offers data logging, M2M communication for devices and sensors.	HTML, javascript

Table 3 Application domain and architecture of top middlewares for CoT

Middlewares	**Application dom**	**Architecture**	**Cloud-based**	**Commercialized**
DropLock	Smart home	Service-based	Yes	No
Aura	Pervasive computing	Distributed	No	No
Capnet	Mobile multimedia	Node-based, distributed	No	No
Gaia	Handling ubiquitous computing environments and living areas	Service-based, distributed	No	No
Carriots	Smart energy, smart city	Service-based	Yes	Yes
OpenIoT	Crowdsourcing, smart cities	Service-based	Yes	No
Link smart	Smart networked embedded mechanisms	Service-based	No	Yes
Xively	Home appliances management and connectivity	Service-based	Yes	Yes
CHOReOS	Enabling QoS-aware, large-scale choreographies	Component-based, service-based	Yes	Yes
ThingWorx	Smart buildings, smart cities and agriculture	Service-based	Yes	Yes
VIRTUS	E-health	Distributed	No	No

Rimware	Smart lighting and healthcare	Service-based	Yes	No

3. Framework design of proposed BCoT system

3.1 Conceptual framework design of proposed BCoT system

Figure 1 presents the conceptual design of the proposed BCoT framework. IoT devices act as the data source, and the data is securely collected through APIs and stored in a cloud database. During the transaction process, two operations take place:

a. The transmitted data undergoes hash computation using the BLAKE2b algorithm.

b. Data encryption and decryption occur within the cloud database using the Salsa20 algorithm.

The computed hash and the encryption key are stored in the blockchain. Additionally, they are associated with the metadata stored in the blockchain.The user interface provides federated access to authorized users. These users can log in to the designated user interface and, upon successful authentication, send a request for the hash and key to the blockchain layer.

Upon receiving the request, the blockchain layer sends a data retrieval request to the cloud database. The cloud database then sends the encrypted data to the user application. In the user application, the data is decrypted using the key obtained from the blockchain, and the hash is computed.

If the newly computed hash matches the hash computed during the data transfer from the IoT devices to the cloud database, the data is displayed on the user interface. If the computed hash does not match, it indicates that the data has been tampered with. In such cases, the system notifies the requesting party about the tampering. Importantly, the BCoT framework is designed to be platform-agnostic,

meaning it can be implemented on different platforms without constraints.

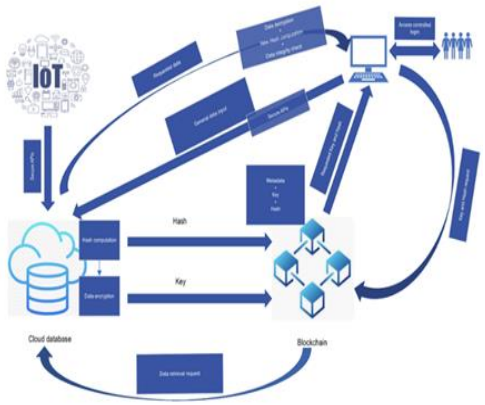

Figure1. Conceptual design of proposed BCoT framework

3.2 Layer-based architecture of proposed BCoT system

Figure 2 depicts the layer-based architecture of the proposed BCoT Framework. This architecture is designed with modularity and decoupling in mind, enabling developers to make independent changes or additions to modules without affecting the overall system.

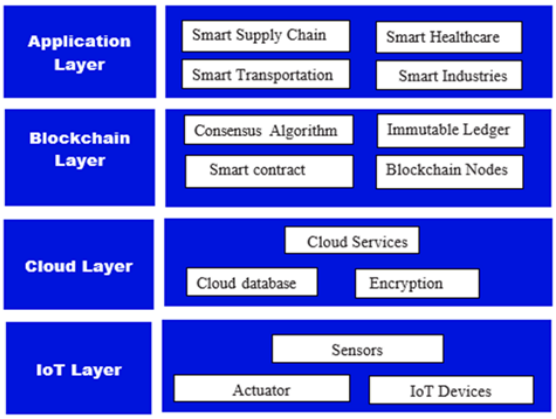

Figure 2. Layer based architecture of proposed BCoT System.

The layer-based architecture of the proposed BCoT Framework, as illustrated in Figure 5.2, consists of several layers that work together to provide different functionalities and advantages for the system. The IoT layer serves as the data source, encompassing sensors, actuators, and IoT devices interconnected in a complex network. It plays a crucial role in generating and collecting data.

The cloud layer acts as the hosting platform for the BCoT system and provides data storage capabilities . It

complements the framework by offering various cloud services, ensuring scalability, reliability, and security through encryption algorithms.

The Blockchain layer utilizes the Polygon blockchain, which operates using a distributed layered architecture to enhance scalability and interoperability. It employs a modified proof of stake (PoS) consensus mechanism called the PoS Chain, where validators stake their tokens as collateral and take turns proposing and validating blocks based on their stake and reputation. This mechanism ensures efficiency and security, as validators are incentivized to act honestly.

The main Ethereum network also plays a significant role within the Blockchain layer of the BCoT framework. Polygon leverages the Ethereum Virtual Machine (EVM) to maintain compatibility with Ethereum smart contracts, facilitating seamless migration of decentralized applications (dApps) and assets between Ethereum and Polygon.

Finally, the application layer offers numerous advantages for industrial applications in various domains, including smart healthcare, smart transportation, smart city, smart energy, and smart industry [21]. The BCoT framework not only improves network management and quality of service but also ensures security and privacy for these applied domains.

3.3 BCoT enabled Covid Vaccine Supply Chain Management System (PoC)

We have chosen to implement a proof-of-concept (Proof-of-Concept) of the proposed BCoT framework by developing a Covid vaccine supply chain management system. This use case is highly relevant, considering the global priority of efficiently manufacturing, distributing, and administering Covid vaccines to save lives. Through the BCoT framework, we aim to demonstrate how a complex supply chain management system can be made scalable, distributed, and secure at every stage of the transaction.

3.3.1 Architecture Design of Proof- of -Concept

In the Proof-of-Concept architecture design, the vaccine manufacturing process involves the use of the BCoT application interface to record manufacturing details in a cloud database. The IoT devices automatically add relevant data about the manufacturing process, eliminating the need for manual input.

For packaging, logistics, and distribution, IoT devices are integrated into the process to enable real-time tracking of environmental parameters, such as temperature, humidity, light exposure, O2 and CO2 levels, throughout packaging, transportation, and distribution. The recorded data is stored in a cloud database. Distributors utilize the BCoT system to track vaccine delivery and record logistics details, which are also stored in the cloud database.

Healthcare professionals can use the BCoT system to validate the authenticity and validity of vaccine batches received for administering to the general population. Administration details, including dose amount, patient information, and timestamps, are recorded in the cloud database and hashed for storage on the blockchain.

To ensure consumer access and transparency, patients or authorized parties can validate vaccines by scanning QR codes on the web interface of the BCoT system. This allows for greater transparency and accountability in the supply chain management of vaccines.

Overall, this Proof-of-Concept implementation of the BCoT framework in the Covid vaccine supply chain management system showcases how a comprehensive, secure, and scalable solution can be developed for critical processes like vaccine manufacturing, distribution, and administration.

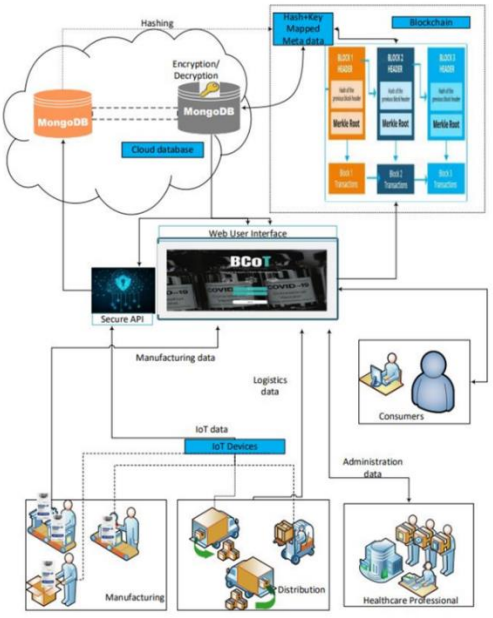

Figure 3. PoC architecture of proposed BCoT system

Figure 3 illustrates the implementation of a Proof-of-

Concept (PoC) for the BCoT framework, specifically focusing on a blockchain-enabled vaccine supply chain management system. This system incorporates end-to end monitoring and tracking mechanisms utilizing an IoT simulator, cloud database, and blockchain-enabled security features.

In this implementation, data inputs are streamed from the IoT simulator through secure APIs. Additionally, relevant data from manufacturers, distributors, and healthcare professionals is manually inputted and stored in an encrypted form in the cloud database, alongside the IoT data. To ensure security, all data streams pass through a secure API layer that prevents direct exposure of the backend. Before encryption, the data is hashed, and the resulting hash, along with the decryption key and unique metadata, is stored in the blockchain.

Users interact with the system through the blockchain layer, with access facilitated by a wallet system. Proper authorization is required for accessing the data, ensuring only authorized individuals can retrieve information from the system.

3.3.2 Transaction process of Proof -of -Concept

The transaction process of the BCoT-enabled Covid vaccine supply chain management system is depicted in Figure 4. The system aims to establish a resilient and secure workflow for every transaction occurring within its scope.In this system, each transaction is carefully managed to ensure its integrity and security. The process involves various stages, such as vaccine manufacturing, packaging, logistics, distribution, and administration. At each stage, data is recorded and stored in a secure manner, leveraging the BCoT framework.

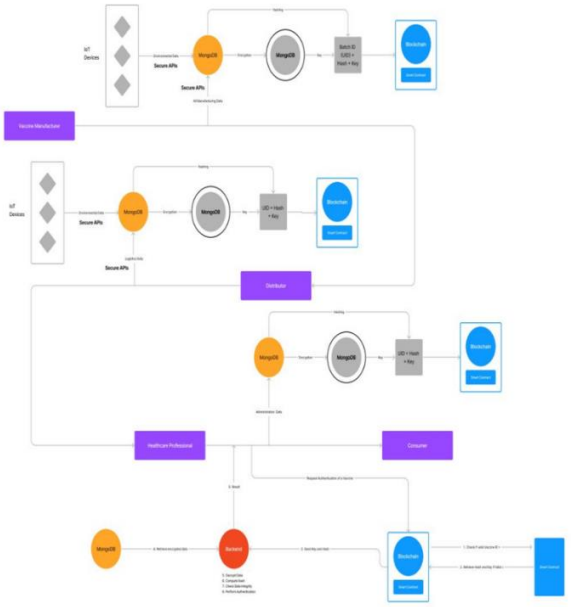

Figure 4. PoC Transaction Process

- **Signup process**

Any participant in the system, whether they are a manufacturer, distributor, or healthcare professional, can register by creating an account and providing their role. During the registration process, the user's wallet, such as MetaMask, prompts them to sign a unique piece of text using their private key.

Once the user signs the text with their private key using MetaMask, the resulting signature, which is specific to their wallet address, is received and stored in the MongoDB database along with their address.

Simultaneously, a new user profile is created on the blockchain using a smart contract. This smart contract stores the user's address and role, ensuring the user's identity and role are recorded securely on the blockchain.

- **Login process**

During the login process, when a user wants to access their account, MetaMask prompts them to provide a signature again. This signature is then cross-checked with the signature stored in the database. If the two signatures match, the user profile is retrieved from the blockchain, and the user is successfully logged in.

It is important to note that only individuals who have access to the private key associated with the account can log in. Unauthorized individuals attempting to sign the text with a different wallet address will not produce matching signatures and, as a result, will be unable to log in. This ensures that only users with the

appropriate private key can access their respective accounts.

- Manufacturer workflow process

Manufacturing workflow Process is shown in figure 5.

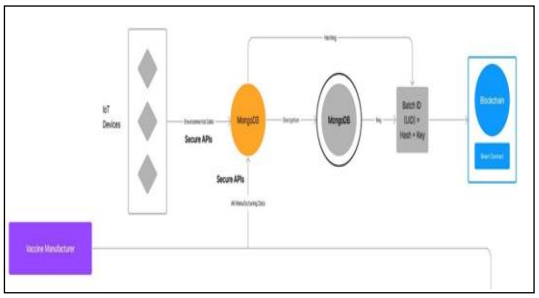

Figure 5. Manufacturer Workflow Process

When a user wants to add a new manufacturer, they provide various data such as batch ID, vaccine count, vaccine name, along with additional information like manufacturer ID, party address, quality control documents, and expiry date. This data is submitted through a secured API that requires an API key for authentication. The data is then encrypted and hashed in the MongoDB database.

To ensure transparency and traceability, the decryption key, data hash, sender's address, and transaction hash are sent to the blockchain by invoking a specific smart contract function. Access control measures are implemented in the smart contract, restricting the function to be called only by a manufacturer's address and preventing any other unauthorized access.

Furthermore, a QR code is dynamically generated in the frontend, pointing to a unique URL where the product can be tracked. This QR code serves as a convenient way to access real-time tracking information without the need for storing it explicitly.

The relevant IoT data associated with the batch, such as manufacturing details, is automatically added by the IoT device itself during the creation of a new batch. This eliminates the need for manual input from the creator, ensuring accurate and reliable data capture.

During the creation of a new batch, the manufacturer has the option to either move the batch to inventory or send it directly to the distributor. If the batch is moved to inventory, it is stored as stock, and later, the manufacturer can access the inventory and select the specific product to be transferred to the distributor, effectively managing the distribution process.

- **Distributor workflow process**

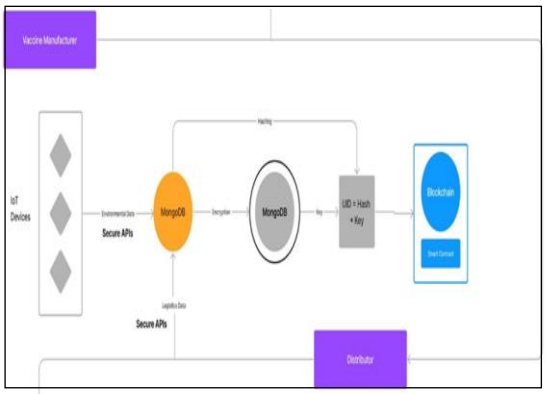

Figure 6. Distributor Workflow Process

Distributor workflow Process is shown in figure 6.In this distributor interacts with their dashboard by entering the batch ID to check the availability of the shipment for pickup. If the shipment is available, the distributor has the option to either move it to inventory or proceed with delivery. Similar to previous cases, a smart contract execution occurs, but this time it is restricted specifically to the distributor's role. If the batch is moved to inventory, the distributor can later select the product and send it to the healthcare professional. Once the product is selected for delivery to a healthcare professional, its status is updated as "In Transit."

The distributor can then track the shipment, including monitoring the environmental parameters. The dashboard simulates real-time data as if it were being collected by an IoT device, providing the distributor with up-to-date information about the shipment's location and environmental conditions.

- **Healthcare Professional workflow process**

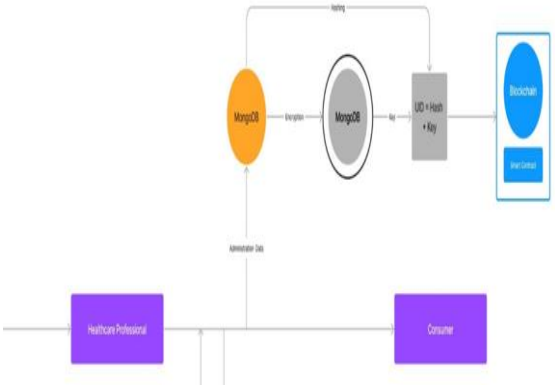

Figure7.Healthcare Professional Workflow Process

In this process**,** receiving the shipment, the healthcare professional marks it as received. Subsequently, all the environmental data captured by the IoT devices during transportation is sent to and stored in the MongoDB database.

Before administering the vaccine to a patient, the healthcare professional has the option to verify the data of the vaccine by entering its ID. The dashboard then displays information about the manufacturer and distributor associated with the vaccine. This allows the healthcare professional to validate the authenticity and traceability of the vaccine.

If the vaccine is verified and determined to be legitimate, the healthcare professional is permitted to administer it to the patient. Once administered, the vaccine is marked as consumed in the system, indicating that it has been successfully used and accounted for.

- **Consumer access process**

The consumer access process within the BCoT system. Consumers or recipients of the vaccine have the ability to retrieve details about the administered vaccine through the BCoT web interface. They provide the unique identifier (UID) of the vaccine within the web interface.

The request is then directed through the blockchain layer, which transfers the relevant hash key and decryption key to the backend system. The backend system retrieves the encrypted data from the database and decrypts it using the provided decryption key. It then calculates a new hash for the decrypted data and performs a data integrity check by comparing this new hash with the previous hash associated with the same data set.

Once the authentication process is successfully completed, and the data integrity check passes, the backend system renders the requested data to the web interface. This ensures that consumers can securely access and verify the information about the administered vaccine, guaranteeing the integrity and authenticity of the data.

4. Implementation of the Proposed BCoT system

Our research work introduces a groundbreaking approach by combining blockchain technology with the Cloud of Things (CoT) paradigm. The implementation of the BCoT framework focuses on creating a resilient architecture that seamlessly integrates blockchain networks, cloud platforms, and IoT data. This architecture facilitates secure and transparent communication as well as data sharing. To guarantee the integrity and confidentiality of transmitted and stored data within the BCoT system, we meticulously choose and configure suitable blockchain platforms, consensus mechanisms, cloud databases, and encryption techniques. Rigorous testing is conducted to validate the functionality, performance, and security of the integrated system.

4.1 Technology Selection

Choosing the appropriate technology stack is of utmost importance when implementing a collaborative Cloud of Things (CoT) system for covid vaccine manufacturing and supply chain management using blockchain technology. The following technology stack has been identified for our implementation process.

Table 4 Technology Stack

Frontend	**React JS**
CSS	**Tailwind CSS**
Backend	**Node JS**
Database	**MongoDB Atlas**
Blockchain	**Polygon Mumbai Testnet**
Libraries used	**Web3.js**
Smart Contracts	**Solidity**
Cloud platform	**Amazon Web Services**
IoT device Simulator	**AWS IoT hub**

4.2 Blockchain Network Setup

To set up our blockchain network, we need to perform several steps. Firstly, we need to set up a wallet. Once the wallet is set up, we proceed to set up the contract. This involves writing the complete code for the contract and then compiling it. During the compilation process, we obtain the Application Binary Interface (ABI) which describes the interface of the contract.

After obtaining the ABI, we need to convert it into JSON format. This JSON-formatted ABI is used to create a contract object, which allows us to interact with the functions defined in the contract.

In addition to the contract object, we also require the contract address. This address is obtained after deploying the contract on the blockchain. Once we have both the contract address and ABI, we can establish a connection to the contract within our application.

Figure 8 displays a screenshot of the code snippet used to integrate Web3.js into our application for this

setup process.

```
const setUpWeb3 = async () => {
  if (typeof window.ethereum !== "undefined") {
    try {
      await window.ethereum.request({ method: "eth_requestAccou
      const web3 = new Web3(window.ethereum);
      setAppState((prevState) => {
        return { ...prevState, web3: web3 };
      });
      console.log("<< Web3 Object Received  >>");

      window.ethereum
        .request({ method: "net_version" })
        .then(async (chainId) => {
          if (chainId !== "80001") {
            try {
              await window.ethereum.request({
                method: "wallet_switchEthereumChain",
                params: [{ chainId: "0x13881" }],
              });
              console.log("Polygon Mumbai Chain found.");
            } catch (switchError) {
              console.log("Error connecting to Polygon Mumbai C
            }
          }
        });

      const accounts = await web3.eth.getAccounts();
      console.log("<< Account Received  >>", accounts[0]);

      setAppState((prevState) => {
        return {
          ...prevState,
          account: accounts[0],
        };
      });
    } catch (error) {
      console.error(error);
      console.log("Error getting web3 object. Install Metamask.
    }
  } else {
    console.log("Please install MetaMask to connect your wallet
  }
};
```

Figure 8. Web3 js connector snippet

4.3 Smart Contracts Development

Smart contracts are digital contracts that operate automatically on blockchain technology. They are computer programs designed to execute predefined actions and enforce agreed-upon terms between involved parties. These contracts eliminate the need for intermediaries by facilitating, verifying, and enforcing contractual agreements in a secure and transparent manner. By leveraging smart contracts, trust can be established and the execution of agreements can be automated, enabling decentralized and secure transactions without intermediaries.

The developed smart contracts consist of several components. First, there are structs that define the data structures used in the contracts. The "VaccineBatch" struct represents a batch of vaccines and includes fields such as batchId, decryptionKey, dataHash, manufacturerId, distributorId, and deliverId. The "User" struct represents a user and includes fields such as userId, role, userAddress, decryptionKey, and dataHash.

Mappings are used to store and retrieve data efficiently. The "users" mapping maps userId to the User struct, allowing user details to be stored and accessed based on their userId. The "usersByAddress" mapping maps Ethereum addresses to the User struct, enabling users to be looked up based on their Ethereum address. The "batches" mapping maps batchId to the VaccineBatch struct, facilitating storage and retrieval of vaccine batch details based on their batchId.

A state variable called "userCount" keeps track of the number of registered users.

Modifiers are used to restrict access to certain functions based on user roles or ownership of Ethereum addresses. The "onlyManufacturer" modifier restricts access to functions to users with the "Manufacturer" role, while the "onlyDistributor" modifier restricts access to users with the "Distributor" role. The "onlyHProf" modifier allows access only to users with the "HProf" role, and the "onlyUser" modifier restricts access to functions based on ownership of the Ethereum address associated with the function call.

Several functions are implemented in the smart contracts. The "signup" function allows users to sign up by providing their role, decryption key, data hash, and userId. It adds the user to the "users" mapping using their userId as the key and increments the "userCount". The "signin" function enables users to retrieve their details by verifying ownership of the Ethereum address associated with the function call. The "updateUser" function allows users to update their decryption key and data hash by providing their userId, new key, and new hash. This function updates the corresponding fields in the "users" and "usersByAddress" mappings.

For manufacturers, the "newManufacture" function enables the creation of a new vaccine batch by providing batch details such as batchId, decryption key, data hash, manufacturerId, and distributorId. The function adds the batch to the "batches" mapping using the batchId as the key. The "updateManufacture" function allows manufacturers to update the decryption key and data hash of a specific vaccine batch, as well as the data hash of the associated user. It modifies the corresponding fields in the "batches," "users," and "usersByAddress" mappings. The "getManufactureData" function permits manufacturers to retrieve the details of a specific vaccine batch by providing the batchId and their userId.

```
function signup(    infinite gas
    string memory _role,
    string memory _key,
    string memory _hash,
    string memory _userId
) public {
    userCount++;
    users[_userId] = User(_userId, _role, msg.sender, _key, _hash);
    usersByAddress[msg.sender] = User(
        _userId,
        _role,
        msg.sender,
        _key,
        _hash
    );
}

function signin() public view onlyUser(msg.sender) returns (User memory) {
    return usersByAddress[msg.sender];
}

function updateUser(    infinite gas
    string memory _userId,
    string memory _key,
    string memory _hash
) public onlyUser(msg.sender) {
    User storage user = users[_userId];
    usersByAddress[msg.sender].decryptionKey = _key;
    usersByAddress[msg.sender].dataHash = _hash;
    user.decryptionKey = _key;
    user.dataHash = _hash;
}

function newManufacture(    infinite gas
    string memory _batchId,
    string memory _decryptionKey,
    string memory _dataHash,
    string memory _manufacturerId,
    string memory _disId
) public onlyManufacturer(_manufacturerId) {
    batches[_batchId] = VaccineBatch(
        _batchId,
        _decryptionKey,
        _dataHash,
        _manufacturerId,
        _disId,
        ""
    );
}

function updateManufacture(    infinite gas
    string memory _batchId,
    string memory _key,
    string memory _hash,
    string memory _userId,
    string memory _userHash
) public onlyManufacturer(_userId) {
    batches[_batchId].decryptionKey = _key;
    batches[_batchId].dataHash = _hash;
    users[_userId].dataHash = _userHash;
    usersByAddress[msg.sender].dataHash = _userHash;
}

function getManufactureData(string memory _batchId, string memory _userId)
    public
    view
    onlyManufacturer(_userId)
    returns (VaccineBatch memory)
{
    return batches[_batchId];
}
```

Figure 9. Screenshot of functions of smart contract

4.4 Integration with AWS IoT device simulator service

The AWS IoT Device Simulator is a service offered by Amazon Web Services (AWS) that allows users to mimic the behavior of IoT devices within a virtual environment. This service is designed to assist developers and IoT solution architects in testing and validating their IoT applications and systems without the need for physical devices. By utilizing the AWS IoT Device Simulator, users can create and configure virtual IoT devices through a user-friendly graphical interface or an API.

The simulator enables users to define the characteristics, behavior, and data patterns of their virtual IoT devices, allowing them to simulate a wide range of scenarios. This capability enables thorough testing of IoT applications, including functionality, scalability, and performance, using simulated datasets.

One of the advantages of the AWS IoT Device Simulator is its ease of use. Users can effortlessly create and simulate a large number of connected devices using the web-based GUI console. This eliminates the need to configure and manage physical devices or spend time developing complex scripts.

Furthermore, the AWS IoT Device Simulator can seamlessly integrate with any custom application through private and public endpoints. This allows users to connect their simulated IoT devices with their desired external applications, facilitating the testing and validation of IoT sensor integration.

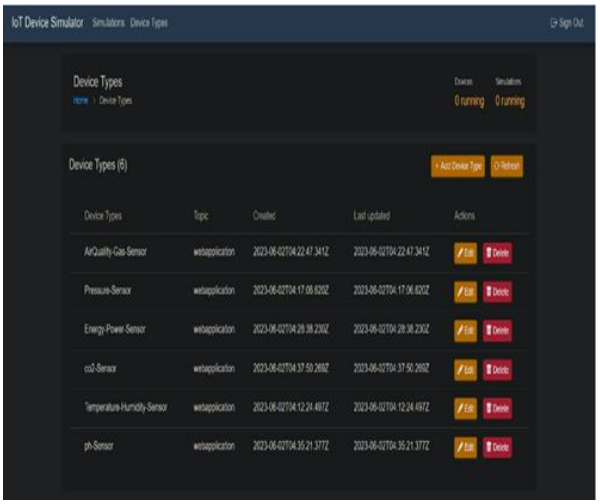

Figure 10.IoT device setup for PoC application

4.5 Data Storage and Management

In this implementation, MongoDB Atlas, a cloud database solution, is utilized. MongoDB Atlas is a platform-agnostic database that can be integrated with various cloud platforms. It offers flexibility in terms of scalability and performance as it is managed by the service provider.

Using MongoDB Atlas also provides the benefit of user-friendly subscription options, such as Serverless, Dedicated, and Shared, which are reasonably priced.

In this project, both the data received from IoT devices (AWS IoT simulator) and the data manually input by manufacturers, distributors, or healthcare professionals are stored in encrypted form within MongoDB. The project architecture ensures that the database is not exposed to the UI layer or any other interfaces. All communication takes place through secure APIs, which transmit the data to MongoDB for hash calculation and encryption.

The calculated hash of the data and the decryption key are then passed to the blockchain, where they are associated with a unique identifier (UID) of a vaccine batch.

When there is a request from the blockchain layer to display data based on the provided input, the

decryption key is passed to MongoDB. Using this key, the encrypted data is decrypted. Subsequently, the hash of the decrypted data is calculated and compared with the previous hash. Only when the hashes match, the data is rendered to the UI.

The figure provided illustrates the MongoDB collections created to store data from various sources:

The "Batches" collection stores data related to vaccine batches, including manufacturing data, IoT data from monitoring vaccine vials at different stages, and transportation and logistics details.

The "Tokens" collection stores user tokens for individuals who sign up on the BCoT application using their Blockchain wallet.

The "Userprofiles" collection stores data associated with different roles created for users, with varying levels of access. These roles determine the authority of users to view or edit data.

The "Users" collection stores user data, such as names, demographic details, company information, and place of origin.

To enhance security, the data stored in MongoDB collections are encrypted and can only be decrypted by authorized users who undergo a rigorous authentication and authorization process.

The provided figure below illustrates the encrypted form of the data.

Figure 11. Encrypted form of the data

4.6 Security and Identity Management

To ensure security and identity management in a blockchain-enabled Cloud of Things (CoT) system, robust access control mechanisms and authentication methods are implemented.

A. Access control mechanisms

Access control mechanisms are in place to restrict unauthorized access to the BCoT system. These mechanisms include authentication and authorization processes to verify the identity of devices, users, and applications before granting access to system resources. The specific authentication mechanism implemented considers factors such as user experience, security requirements, scalability, and the needs of the BCoT network.

In our application, the authentication mechanism follows the following steps for signup:

User connects their MetaMask wallet: The user links their MetaMask wallet to the application, allowing the application to interact with their Ethereum address and sign messages.

User clicks on signup: The user initiates the signup process by clicking the "signup" button.

Nonce generation: A nonce (a unique number) is generated based on the user's wallet address. This nonce helps prevent replay attacks and ensures the integrity of the authentication process.

User signs a message + nonce: The user is prompted to sign a message that includes the nonce. The message typically follows a specific format defined by the application.

Signature generation: The user's MetaMask wallet uses their private key to generate a signature for the message + nonce.

Signature verification: The generated signature is verified to ensure that it corresponds to the actual address holder. This verification process uses the user's public key (wallet address) and the provided signature. Successful verification confirms that the signature matches the expected wallet address.

Storage in MongoDB: If the signature verification is successful, the signature and the associated wallet address are stored in the MongoDB database. This information serves as a record of the user's authentication.

For signin, a similar process is followed:

User connects their MetaMask wallet: The user links their MetaMask wallet to the application.

User clicks on signin: The user initiates the signin process by clicking the "signin" button.

Nonce generation: A new nonce is generated based on the user's wallet address to ensure a unique value for each signin attempt.

User signs a message + nonce: The user signs a message that includes the newly generated nonce.

Signature generation: The user's MetaMask wallet uses their private key to generate a signature for the message + nonce.

Signature verification: The generated signature is verified to ensure it corresponds to the actual address holder. This verification process compares the user's public key (wallet address) and the provided signature. If the verification is successful, indicating a matching signature, the user is considered authenticated and allowed to log in. If the signatures do not match, the login attempt is rejected, indicating invalid credentials.

These authentication mechanisms, along with the secure storage of signatures and wallet addresses in MongoDB, help ensure the security and integrity of the BCoT system.

```
const signupUser = async (email, password, name, role) => {
  try {
    await axios
      .post(backendURL + "/v1/auth/register", {
        address: address,
        signature: signature,
      })
      .then(async (response) => {
        let rand = (Math.floor(Math.random() * 900) + 100).toString();
        await axios
          .post(backendURL + "/create-user-profile", {
            userId: response.data.user.id,
            role: role,
            uid: "MAN" + rand,
          })
          .then(async (response) => {
            const result = await signupSmartContract(
              role,
              response.data.credentials.key,
              response.data.credentials.hash,
              "MAN" + rand.toString()
            );
            if (result === "success") {
              toast.success("Signup successful.", response);
              navigate("/signin");
            } else {
              toast.error("An error occurred while signing up.");
              disconnectWallet();
            }
          })
          .catch((error) => {
            console.log(
              "<< User Profile Creation Response Received  >>",
              error
            );
          });
      })
      .catch((error) => {
        setLoading((prevState) => {
          return {
            loading: false,
            message: "Connecting your wallet...",
          };
        });
        if (error.response.status === 400) {
          toast.error("Account already exists. Please sign in.");
        } else {
          toast.error("An error occurred while signing up.");
        }
        disconnectWallet();
      });
  } catch (error) {
    console.error("Error in signupUser:", error);
    toast.error("An error occurred while signing up.");
  }
};
```

Figure 12.User sign up implementation using authmechanism

```
const signinUser = async (email, password) => {
  try {
    const result = await signinSmartContract();
    if (result !== "error") {
      await axios
        .post(backendURL + "/v1/auth/login", {
          address: address,
          signature: signature,
        })
        .then(async (response) => {
          const userId = response.data.user.id;
          await axios
            .get(backendURL + "/v1/users/" + response.data.user.id, {
              headers: {
                Authorization: `Bearer ${response.data.tokens.access.
                token}`,
              },
            })
            .then(async (response) => {
              await axios
                .get(backendURL + "/user-profile", {
                  params: {
                    userId: userId,
                    key: result.decryptionKey,
                    hash: result.dataHash,
                  },
                })
                .then((response) => {
                  if (response.data.status === "success") {
                    navigate("/dashboard");
                  } else {
                    if (
                      response.data.message ===
                      "Error decrypting user profile"
                    ) {
                      toast.error("Decryption Failure. Invalid key. ");
                    } else if (response.data.message === "Hash mismatch")
                    {
                      toast.error(
                        "Data Integrity Failure. Invalid Hash or Data
                        might have been tampered."
                      );
                    }
                  }
                })
                .catch((error) => {
                  toast.error(
                    "An error occurred while getting the user profile."
                  );
                });
            })
            .catch((error) => {
              toast.error("An error occurred while getting the user.");
            });
        })
        .catch((error) => {
          if (error.response.data.code === 401) {
            toast.error("Invalid Credentials.");
          } else {
            toast.error("An error occurred while signing in.");
          }
          disconnectWallet();
        });
    } else {
      toast.error("An error occurred while signing in the smart
      contract");
    }
  } catch (error) {
    toast.error("An error occurred while signing in.");
  }
};
```

Figure 13. User sign in implementation using auth mechanism

B. Data Encryption/Decryption

We have implemented encryption and decryption techniques to safeguard sensitive data stored in the blockchain or transmitted between devices. To ensure data confidentiality and integrity, we have utilized the Salsa20 encryption algorithm along with key management practices.

The Salsa20 algorithm is a symmetric key stream cipher that operates on 64-byte blocks and supports key sizes of 128, 192, or 256 bits. Here is a simplified explanation of the Salsa20 algorithm:

Initialization: The algorithm takes a secret key and a 64-bit nonce as inputs. These are used to generate the initial state of 16 32-bit words, known as the Salsa state.

Key Setup: The secret key is divided into several words based on the chosen key size. Constant words specific to Salsa20 are also set.

Generate Stream: The Salsa state is set based on the constant words, key words, and the nonce. The state undergoes 20 rounds of mixing the columns and rows, resulting in a keystream.

Encryption/Decryption: To encrypt or decrypt data, the keystream is XORed with the plaintext or ciphertext. If additional keystream blocks are required, the nonce is incremented, and the stream generation process is repeated.

The Salsa20 algorithm utilizes a quarter-round function that operates on four 32-bit words. Eachround involves XOR operations and bit rotations to mix the words.

In our application, we have implemented the Salsa20 algorithm to ensure data security. We have incorporated the necessary code for encryption and decryption using Salsa20. Screenshots of the implemented code can be provided for reference.

```
const encryptUserProfile = async (userProfile, encodedString) => {
  const decodedString = encodedString.split('xxxxx');
  const key = util.decodeBase64(decodedString[0]);
  const nonce = util.decodeBase64(decodedString[1]);
  let encryptedUserProfile = {};
  try {
    encryptedUserProfile = {
      userId: userProfile.userProfile.userId,
      role: util.encodeBase64(tweetnacl.secretbox(util.decodeUTF8
      (userProfile.userProfile.role), nonce, key)),
      uid: util.encodeBase64(tweetnacl.secretbox(util.decodeUTF8
      (userProfile.userProfile.uid), nonce, key)),
      vaccineBatches: util.encodeBase64(
        tweetnacl.secretbox(util.decodeUTF8(JSON.stringify(userProfile.
        userProfile.vaccineBatches)), nonce, key)
      ),
    };
    return {
      status: 'success',
      userProfile: encryptedUserProfile,
    };
  } catch (error) {
    console.error(error);
    return {
      status: 'error',
      message: 'Error encrypting user profile',
    };
  }
};
```

Figure 14.Encrypting user data snippet

```
const decryptUserProfile = async (userProfile, key, nonce) => {
  let decryptedUserProfile = {};
  try {
    decryptedUserProfile = {
      userId: userProfile.userId,
      role: util.encodeUTF8(tweetnacl.secretbox.open(util.decodeBase64
      (userProfile.role), nonce, key)),
      uid: util.encodeUTF8(tweetnacl.secretbox.open(util.decodeBase64
      (userProfile.uid), nonce, key)),
      vaccineBatches: JSON.parse(
        util.encodeUTF8(tweetnacl.secretbox.open(util.decodeBase64
        (userProfile.vaccineBatches), nonce, key))
      ),
    };
    return {
      status: 'success',
      userProfile: decryptedUserProfile,
    };
  } catch (error) {
    console.error(error);
    return {
      status: 'failed',
      message: 'Error decrypting user profile',
    };
  }
};
```

Figure 15. Decrypting user data snippet

C. Data Hashing

Hashing is a process of converting input data into a fixed-length string of characters, known as a hash value or hash code. This transformation is performed by a hash function, which is designed to efficiently generate unique hash values for different inputs. The hash value is considerably shorter in length compared to the original input.

One key characteristic of hash functions is that they are one-way, meaning it is computationally difficult to derive the original input data from the hash value. This property makes hashing useful for scenarios like password storage, where the actual passwords are not stored but rather their hash values. When a user enters a password, it is hashed and compared to the stored hash value for authentication.

In our implementation, we have utilized the BLAKE2b hashing algorithm. Here is a simplified explanation of the algorithm:

Initialization: Set the initial state vector, counter, buffer variables, and message schedule variables based on the BLAKE2b constants.

Padding: Add padding to the input message to make it a multiple of the block size (128 bytes). Append a 1-bit followed by zeros to mark the end of the input.

Compress: Divide the padded message into blocks and perform the following operations for each block:

XOR the block with the current state vector.

Apply the BLAKE2b round function, which operates on 16 64-bit words, to permute the block.

XOR the permuted block with the state vector.

Mix the state vector using a mixing function.

Finalization: XOR the final state vector with the last block of the message. XOR the final state vector with the length of the message. Permute the final state vector. The desired hash value is typically the first n bytes of the state vector.

The BLAKE2b algorithm employs multiple rounds, consisting of mixing operations and round function applications, to ensure strong diffusion and cryptographic strength. It also supports optional features such as keying and personalization. The specific details of the mixing functions, round functions, and constants used in BLAKE2b are more complex and detailed.

It's important to note that the provided code snippet showcases an implementation of the BLAKE2b algorithm and demonstrates how hashing is performed in our system.

```
const hashUserProfile = async (userProfile) => {
  let calculatedHash = '';

  const stringForHash = userProfile.userId + userProfile.role + userProfile.
  uid + JSON.stringify(userProfile.vaccineBatches);


  calculatedHash = blake.blake2bHex(stringForHash);

  return calculatedHash;
};
```

Figure 16. Hashing a ue3ser profile data

4.7 Integration with Cloud Services

- **Integration with Mongodb Atlas**

To integrate MongoDB Atlas as the cloud storage solution, certain steps need to be followed. Initially, the IP address of the backend where the system is deployed is configured for IP whitelisting. This ensures that only connections from the specified IP address are allowed to access the storage, while others are denied access.

Once the IP address is whitelisted, MongoDB Atlas provides a URL string that contains the necessary credentials for establishing a connection to the cloud storage. This URL string includes a username and password, which are crucial for establishing a connection to the MongoDB database. Internally, the application utilizes the Mongoose library to connect to the MongoDB database using the provided URL string and associated credentials.

In summary, IP whitelisting is employed to restrict access to the MongoDB Atlas cloud storage to a specific IP address. The URL string withcredentials is utilized in conjunction with the Mongoose library to establish a connection to the MongoDB database.

- **Integration with AWS IoT device simulator**

To connect with the AWS IoT device simulator AWS IoT Core Rules Engine is used. Rules are defined in the AWS IoT Core Rules Engine to process and transform the incoming device messages. We can specify actions such as invoking AWS Lambda functions, storing data in AWS services, or forwarding the messages to other endpoints.

5. Implementation Outcome

This section assesses the outcomes attained through the proof-of-concept implementation of the proposed BCoT (Blockchain-enabled Cloud of Things) system. The functional model successfully integrated the envisioned principles and objectives of the BCoT framework, including security, scalability, distribution, decentralization, interoperability and immutability. The system effectively addresses a prominent use case where the BCoT system is particularly relevant: the secure management of the vaccine supply chain, crucial for saving lives**.**

Figure17. Login screen

There are 3 types of user roles who can register on the BCoT application.

i. Manufacturer
ii. Distributor and
iii. Healthcare Professional

When a user registers, a unique id is assigned which is later also used in authorizing the user.

A manufacturer can use the system to register relevant data like batch id, manufacturing date, expiry date etc. about the new batch of vaccines which have been manufactured and, are ready to be shipped to distributor or added to the manufacturer's inventory.

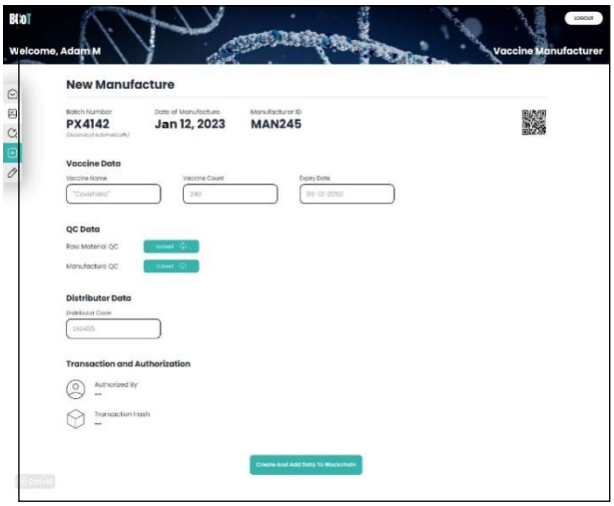

Figure 18.Newmanufacturer screen details

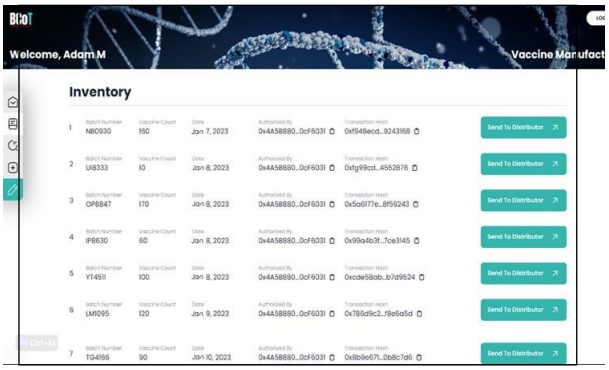

Figure 19. Inventory listing

A distributor can confirm whether the vaccine batch is received and can also validate the details as well as environmental parameters to ascertain the authenticity and validity of the batch.

He/She can then move the batch to healthcare professional or add the batch of vaccines to the inventory.

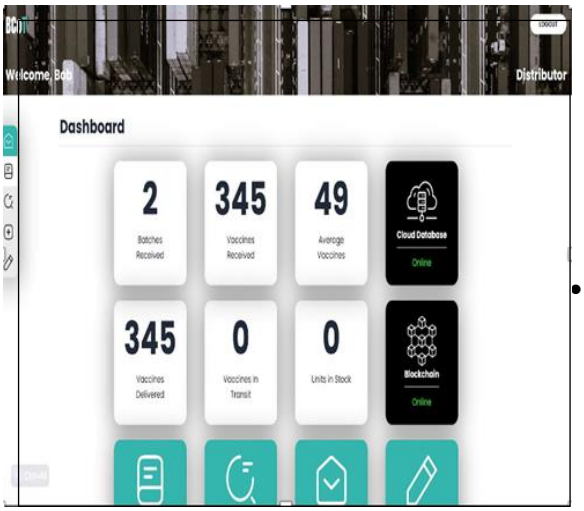

Figure 20. Distributor Dashboard

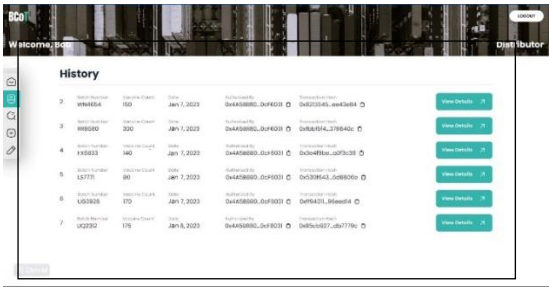

Figure21. Distributorbatch history details

A healthcare professional must be sure that the vaccine is authentic, unspoiled and untampered before administering to patients.

He/she have the ability to check the authenticity in the system and also check the environmental details. Only after all the necessary confirmation, the healthcare professional administers the Vaccine.

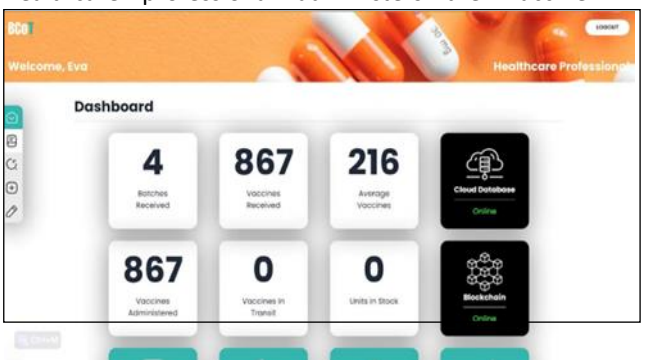

Figure 22.Healthcare professional dashboard

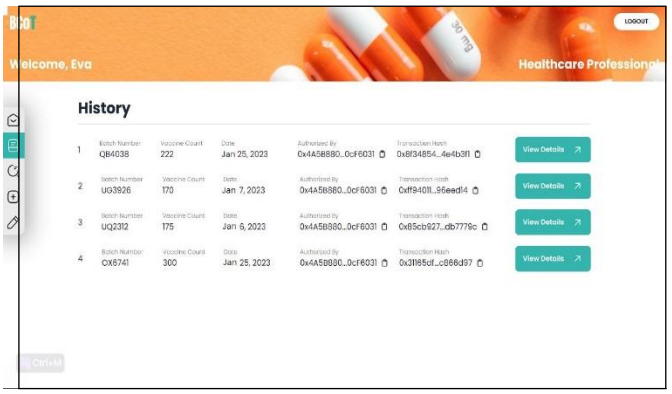

Figure 23. Batch history details

6. Testing Results

Smart Contract Testing Result:

In the process of testing smart contracts using Hardhat, we utilize various functions and techniques. One important function is the "expect" function, which is employed for performing assertions and verifying the expected behavior of the contract. Additionally, the "describe" function is used to group and organize test cases within a test suite, providing a descriptive name for the test suite.

In our specific case, we have created a test suite for a vaccine contract. The first test case within this suite focuses on verifying that users can successfully sign up for the vaccine after the contract has been deployed. By using the "describe" function, we can group related test cases together and assign a meaningful name to the test suite, such as "Vaccine Contract Test Suite." This aids in organizing and categorizing the tests.

The test results, including the executed test cases and their outcomes, are displayed in the provided figure. Apart from the "vaccine contract allows a user to sign up after deployment" test case, there are other test cases that have been executed. These include tests

for signing in, updating user information, creating and updating vaccine batches, as well as retrieving batch data for a given batch ID.

The test results are depicted below the description, showcasing the outcomes of the executed test cases.

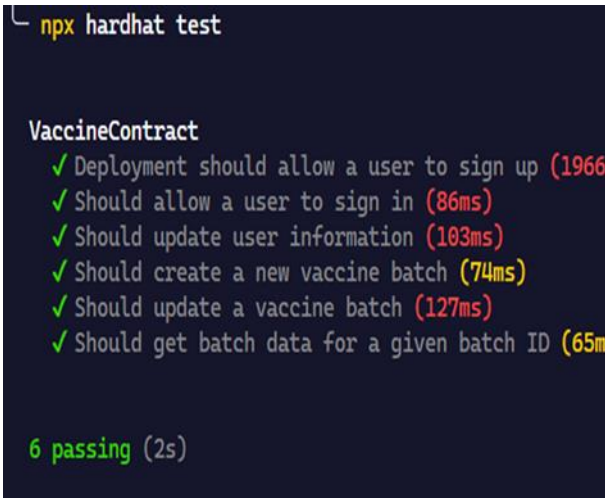

Figure 24.Smart contract testing result using hardhat test

- **API functional testing result**

We conducted API functional testing using Postman. In the request builder window, we provided the necessary details for our API request, including the request URL, headers, request method (e.g., GET, POST, PUT, DELETE), and request body (if applicable). We utilized test scripts to check various values relevant to our API.

Postman displayed the response received from the API, and we carefully examined the response to ensure it aligned with our expectations. We validated the APIs and performed assertions on the responses to verify the proper functioning of our API endpoints.

Below, the test scripts we utilized and the corresponding results.

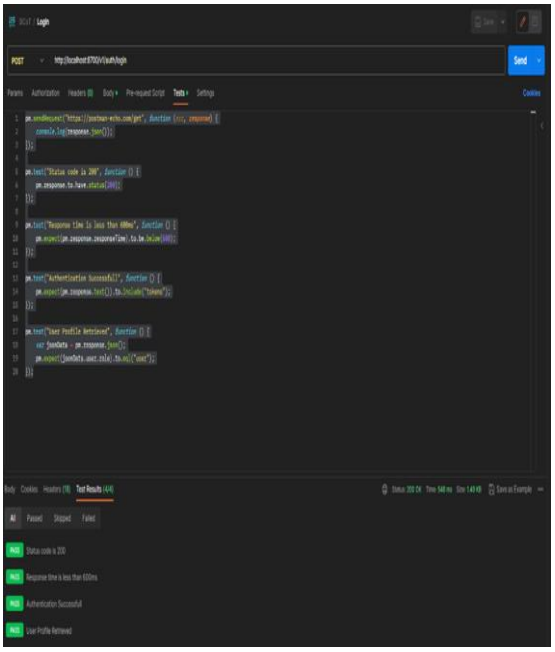

Figure25.Postman testing results for login API

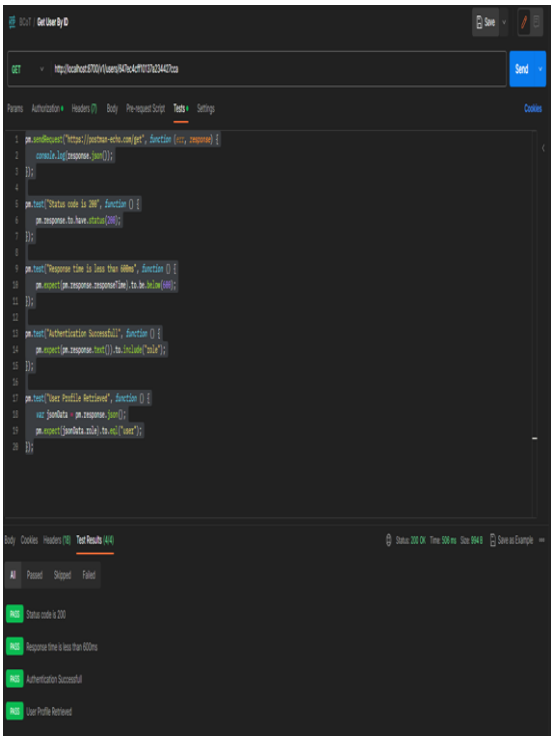

Figure 26. Postman testing result for User profile API

7. Findings

We have achieved successful design and development of a collaborative framework called Blockchain Collaborated Cloud of Things (BCoT). This framework embodies essential qualities such as security, decentralization, distribution, scalability, affordability, immutability, and interoperability. Our comprehensive research and development process

enabled us to address all identified research problems and achieve our research objectives effectively.

To ensure the security of our framework, we have implemented strong security measures including authentication mechanisms, encryption algorithms, and consensus protocols. By harnessing the power of blockchain technology, we have realized a decentralized and distributed system, enhancing resilience and fault tolerance within the BCoT environment.

Scalability has been a primary focus throughout the framework's development. Innovative solutions, such as leveraging Polygon blockchain and cloud databases, have been implemented to ensure the system can efficiently handle the increasing number of IoT devices, data transactions, and user interactions without compromising performance.

Affordability has also been prioritized in our framework. By utilizing cost-effective technologies like cloud databases and Polygon blockchain, we have minimized infrastructure and licensing costs, making the framework more accessible to organizations and users.

We have conducted rigorous testing procedures to validate the reliability and effectiveness of our implementation. Through extensive testing, we have successfully passed each test, demonstrating the robustness and functionality of our framework.

Conclusion This research paper highlights the significant impact that can be made by combining blockchain technology with the cloud of Things (CoT). Our findings demonstrate that integrating blockchain technology into CoT enhances security, data integrity, and trustworthiness. By leveraging the decentralized and immutable nature of blockchain, a robust and tamper-resistant framework can be established for managing IoT devices, data, and transactions. The practical applications of blockchain-enabled CoT are showcased across various industries, including supply chain management, healthcare, logistics, and energy, offering improved efficiency and transparency. Advancements such as smart contracts, decentralized data sharing, and auditability have the potential to revolutionize these sectors. Looking ahead, this research holds great promise. It is crucial for academia, industry, and regulatory bodies to continue collaborating to establish standards, ensure interoperability, and develop best practices. By refining the existing framework, we can unlock the full potential of the proposed collaborative approach.

References

[1] Tschorsch, F., & Scheuermann, B. (2016). Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys & Tutorials, 18(3), 2084-2123. https://doi.org/10.1109/COMST.2016.2535718

[2] Karunarathne, M. S., Jones, S. A., Ekanayake, S. W., & Pathirana, P. N. (2014, December). Remote monitoring system enabling cloud technology upon smart phones and inertial sensors for human kinematics. In 2014 IEEE Fourth International Conference on Big Data and Cloud Computing (pp. 137-142). IEEE.

[3] Aazam, M., Khan, I., Alsaffar, A. A., & Huh, E. N. (2014, January). Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. In Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan, 14th-18th January, 2014 (pp. 414-419). IEEE.

[4] Kantarci, B., &Mouftah, H. T. (2015, June). Sensing services in cloud-centric Internet of Things: A survey, taxonomy and challenges. In 2015 IEEE International Conference on Communication Workshop (ICCW) (pp. 1865-1870). IEEE.

[5] Atlam, H. F., Alenezi, A., Alharthi, A., Walters, R. J., & Wills, G. B. (2017, June). Integration of cloud computing with internet of things: challenges and open issues. In 2017 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) (pp. 670-675). IEEE.4

[6] Zhou, J., Cao, Z., Dong, X., &Vasilakos, A. V. (2017). Security and privacy for cloud-based IoT: Challenges. IEEE Communications Magazine, 55(1), 26-33. https://doi.org/10.1109/MCOM.2017.1600363CM

[7] Ali, M. S., Vecchio, M., Pincheira, M., Dolui, K., Antonelli, F., &Rehmani, M. H. (2018). Applications of blockchains in the Internet of Things: A comprehensive survey. IEEE

Communications Surveys & Tutorials, 21(2), 1676-1717. https://doi.org/10.1109/COMST.2018.2886932

[8] Ferrag, M. A., Derdour, M., Mukherjee, M., Derhab, A., Maglaras, L., &Janicke, H. (2018). Blockchain technologies for the internet of things: Research issues and challenges. IEEE Internet of Things Journal, 6(2), 2188-2204. https://doi.org/10.1109/JIOT.2018.2882794

[9] Fernández-Caramés, T. M., & Fraga-Lamas, P. (2018). A Review on the Use of Blockchain for the Internet of Things. Ieee Access, 6, 32979-33001. https://doi.org/10.1109/ACCESS.2018.2842685

[10] Dai, H. N., Zheng, Z., & Zhang, Y. (2019). Blockchain for Internet of Things: A survey. IEEE Internet of Things Journal, 6(5), 8076-8094. https://doi.org/10.1109/JIOT.2019.2920987

[11] Uriarte, R. B., & DeNicola, R. (2018). Blockchain-based decentralized cloud/fog solutions: Challenges, opportunities, and standards. IEEE Communications Standards Magazine, 2(3), 22-28. https://doi.org/10.1109/MCOMSTD.2018.1800020

[12] Wu, M., Wang, K., Cai, X., Guo, S., Guo, M., & Rong, C. (2019). A comprehensive survey of blockchain: From theory to IoT applications and beyond. IEEE Internet of Things Journal, 6(5), 8114-8154. https://doi.org/10.1109/JIOT.2019.2922538

[13] Yang, R., Yu, F. R., Si, P., Yang, Z., & Zhang, Y. (2019). Integrated blockchain and edge computing systems: A survey, some research issues and challenges. IEEE Communications Surveys & Tutorials, 21(2), 1508-1532. https://doi.org/10.1109/COMST.2019.2894727

[14] Park, J. H., & Park, J. H. (2017). Blockchain security in cloud computing: Use cases, challenges, and solutions. Symmetry, 9(8), 164. https://doi.org/10.3390/sym9080164

[15] Panwar, A., & Bhatnagar, V. (2020). Analyzing the performance of data processing in private blockchain based distributed ledger. Journal of Information and Optimization Sciences, 41(6), 1407-1418. https://doi.org/10.1080/02522667.2020.1809095

[16] Liu, J., & Liu, Z. (2019). A survey on security verification of blockchain smart contracts. IEEE Access, 7, 77894-77904. https://doi.org/10.1109/ACCESS.2019.2921624

[17] Li, Z., Yang, Z., &Xie, S. (2019). Computing resource trading for edge-cloud-assisted Internet of Things. IEEE Transactions on Industrial Informatics, 15(6), 3661-3669.https://doi.org/10.1109/TII.2019.2897364

[18] Zhang, Y., Xu, C., Lin, X., & Shen, X. S. (2019). Blockchain-based public integrity verification for cloud storage against procrastinating auditors. IEEE Transactions on Cloud Computing, 9(3), 923-937.https://doi.org/10.1109/TCC.2019.2908400

[19] Nayak, S., Narendra, N. C., Shukla, A., &Kempf, J. (2018, July). Saranyu: Using smart contracts and blockchain for cloud tenant management. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD) (pp. 857-861). IEEE.

[20] Alammary, A., Alhazmi, S., Almasri, M., & Gillani, S. (2019). Blockchain-based applications in education: A systematic review. Applied Sciences, 9(12), 2400.https://doi.org/10.3390/app9122400

[21] Pandey, R., & Pathak, R. K. (2023). BLOCKCHAIN EMPOWERED CLOUD OF THINGS: A REVIEW ON ITS APPLICATIONS, MERITS AND DEMERITS. Journal of Data Acquisition and Processing, 38(2), 634

[22] Hori, M., & Ohashi, M. (2018, June). Adaptive Identity authentication of blockchain system-the collaborative cloud educational system. In EdMedia+ Innovate Learning (pp. 1339-1346). Association for the Advancement of Computing in Education (AACE).

[23] Purdon, I., &Erturk, E. (2017). To the cloud and its potential role in computer science education. Engineering, Technology & Applied Science Research, 7(6), 2340-2344.https://doi.org/10.48084/etasr.1629

[24] Pandey, R., &kumar Pathak, R. (2022). Design and Implementation of Collaborative Framework of Blockchain and Cloud of Things. Mathematical Statistician and Engineering Applications, 71(4), 3709-3720.

[25] Rahman, M. A., Rashid, M. M., Hossain, M. S., Hassanain, E., Alhamid, M. F., &Guizani, M. (2019). Blockchain and IoT-based cognitive edge framework for sharing economy services in a smart city. IEEE Access, 7, 18611-18621.https://doi.org/10.1109/ACCESS.2019.2896065

[26] 24. Yu, H., Yang, Z., & Sinnott, R. O. (2018). Decentralized big data auditing for smart city environments leveraging blockchain technology. IEEE Access, 7, 6288-6296.https://doi.org/10.1109/ACCESS.2018.2888940

[27] Singh, S., Ra, I. H., Meng, W., Kaur, M., & Cho, G. H. (2019). SH-BlockCC: A secure and efficient Internet of things smart home architecture based on cloud computing and blockchain technology. International Journal of Distributed Sensor Networks, 15(4).https://doi.org/10.1177%2F1550147719844159

[28] Dorri, A., Steger, M., Kanhere, S. S., &Jurdak, R. (2017). Blockchain: A distributed solution to automotive security and privacy. IEEE Communications Magazine, 55(12), 119-125.https://doi.org/10.1109/MCOM.2017.1700879

[29] Sookhak, M., Tang, H., He, Y., & Yu, F. R. (2018). Security and privacy of smart cities: a survey, research issues and challenges. IEEE Communications Surveys & Tutorials, 21(2), 1718-1743.https://doi.org/10.1109/COMST.2018.2867288

[30] Tapas, N., Merlino, G., & Longo, F. (2018, June). Blockchain-based IoT-cloud authorization and delegation. In 2018 IEEE International Conference on Smart Computing (SMARTCOMP) (pp. 411-416). IEEE.

[31] Paul, R., Ghosh, N., Sau, S., Chakrabarti, A., & Mohapatra, P. (2021). Blockchain based secure smart city architecture using low resource IoTs. Computer Networks, 108234.https://doi.org/10.1016/j.comnet.2021.108234

[32] 30.Khezr, S., Moniruzzaman, M., Yassine, A., & Benlamri, R. (2019). Blockchain technology in healthcare: A comprehensive review and directions for future research. Applied sciences, 9(9), 1736.https://doi.org/10.3390/app9091736

[33] Hölbl, M., Kompara, M., Kamišalić, A., & Nemec Zlatolas, L. (2018). A systematic review of the use of blockchain in healthcare. Symmetry, 10(10), 470.https://doi.org/10.3390/sym10100470

[34] 3Li, S., & Pathirana, P. N. (2014, December). Cloud-based non-invasive tele-rehabilitation exercise monitoring. In 2014 IEEE Conference on Biomedical Engineering and Sciences (IECBES) (pp. 385-390). IEEE.

[35] Pham, H. T., & Pathirana, P. N. (2015, June). Measurement and assessment of hand functionality via a cloud-based implementation. In International Conference on Smart Homes and Health Telematics (pp. 289-294). Springer, Cham.

[36] Al Omar, A., Bhuiyan, M. Z. A., Basu, A., Kiyomoto, S., & Rahman, M. S. (2019). Privacy-friendly platform for healthcare data in cloud based on blockchain environment. Future generation computer systems, 95, 511-521.https://doi.org/10.1016/j.future.2018.12.044

[37] Wang, H., & Song, Y. (2018). Secure cloud-based EHR system using attribute-based cryptosystem and blockchain. Journal of medical systems, 42(8), 1-9. https://doi.org/10.1007/s10916-018-0994-6

[38] Nguyen, D. C., Pathirana, P. N., Ding, M., & Seneviratne, A. (2019). Blockchain for secure ehrs sharing of mobile cloud based e-health systems. IEEE access, 7, 66792-66806. https://doi.org/10.1109/ACCESS.2019.2917555

[39] Dwivedi, A. D., Srivastava, G., Dhar, S., & Singh, R. (2019). A decentralized privacy-preserving healthcare blockchain for IoT. Sensors, 19(2), 326.https://doi.org/10.3390/s19020326

[40] Cao, S., Zhang, G., Liu, P., Zhang, X., &Neri, F. (2019). Cloud-assisted secure eHealth systems for tamper-proofing EHR via blockchain. Information Sciences, 485, 427-440.https://doi.org/10.1016/j.ins.2019.02.038

[41] Kaur, H., Alam, M. A., Jameel, R., Mourya, A. K., & Chang, V. (2018). A proposed solution and future direction for blockchain-based heterogeneous medicare data in cloud environment. Journal of medical systems, 42(8), 1-11.https://doi.org/10.1007/s10916-018-1007-5

[42] Jin, H., Luo, Y., Li, P., & Mathew, J. (2019). A review of secure and privacy-preserving medical data sharing. IEEE Access, 7, 61656-61669.https://doi.org/10.1109/ACCESS.2019.2916503

[43] Yang, T., Guo, Q., Tai, X., Sun, H., Zhang, B., Zhao, W., & Lin, C. (2017, November). Applying blockchain technology to decentralized operation in future energy internet. In 2017 IEEE

Conference on Energy Internet and Energy System Integration (EI2) (pp. 1-5). IEEE.

[44] Liang, X., Zhao, J., Shetty, S., Liu, J., & Li, D. (2017, October). Integrating blockchain for data sharing and collaboration in mobile healthcare applications. In 2017 IEEE 28th annual international symposium on personal, indoor, and mobile radio communications (PIMRC) (pp. 1-5). IEEE.

[45] Wang, S., Zhang, Y., & Zhang, Y. (2018). A blockchain-based framework for data sharing with fine-grained access control in decentralized storage systems. IEEE Access, 6, 38437-38450.https://doi.org/10.1109/ACCESS.2018.2851611

[46] Zheng, X., Mukkamala, R. R., Vatrapu, R., &Ordieres-Mere, J. (2018, September). Blockchain-based personal health data sharing system using cloud storage. In 2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom) (pp. 1-6). IEEE.

[47] Xia, Q. I., Sifah, E. B., Asamoah, K. O., Gao, J., Du, X., &Guizani, M. (2017). MeDShare: Trust-less medical data sharing among cloud service providers via blockchain. IEEE Access, 5, 14757-14767.https://doi.org/10.1109/ACCESS.2017.2730843

[48] Xia, Q., Sifah, E. B., Smahi, A., Amofa, S., & Zhang, X. (2017). BBDS: Blockchain-based data sharing for electronic medical records in cloud environments. Information, 8(2), 44.https://doi.org/10.3390/info8020044

[49] Liu, H., Zhang, Y., & Yang, T. (2018). Blockchain-enabled security in electric vehicles cloud and edge computing. IEEE Network, 32(3), 78-83.https://doi.org/10.1109/MNET.2018.1700344

[50] Du, Y., Liu, J., Guan, Z., & Feng, H. (2018, September). A medical information service platform based on distributed cloud and blockchain. In 2018 IEEE International Conference on Smart Cloud (SmartCloud) (pp. 34-39). IEEE.

[51] Celiz, R. C., De La Cruz, Y. E., & Sanchez, D. M. (2018, November). Cloud model for purchase management in health sector of peru based on IoT and blockchain. In 2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON) (pp. 328-334). IEEE.

[52] Park, J., Park, S., Kim, K., & Lee, D. (2018, December). CORUS: Blockchain-based trustworthy evaluation system for efficacy of healthcare remedies. In 2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom) (pp. 181-184). IEEE.

[53] Perboli, G., Musso, S., &Rosano, M. (2018). Blockchain in logistics and supply chain: A lean approach for designing real-world use cases. IEEE Access, 6, 62018-62028.https://doi.org/10.1109/ACCESS.2018.2875782

[54] O'Leary, D. E. (2017). Configuring blockchain architectures for transaction information in blockchain consortiums: The case of accounting and supply chain systems. Intelligent Systems in Accounting, Finance and Management, 24(4), 138-147.http://dx.doi.org/10.1002/isaf.1417

[55] Rejeb, A., Keogh, J. G., &Treiblmaier, H. (2019). Leveraging the internet of things and blockchain technology in supply chain management. Future Internet, 11(7), 161.https://doi.org/10.3390/fi11070161

[56] Mohamed, N., Al-Jaroodi, J., &Lazarova-Molnar, S. (2019). Leveraging the capabilities of industry 4.0 for improving energy efficiency in smart factories. IEEE Access, 7, 18008-18020.https://doi.org/10.1109/ACCESS.2019.2897045

[57] Bahga, A., &Madisetti, V. K. (2016). Blockchain platform for industrial internet of things. Journal of Software Engineering and Applications, 9(10), 533-546.http://dx.doi.org/10.4236/jsea.2016.910036

[58] Lee, C. K., Huo, Y. Z., Zhang, S. Z., & Ng, K. K. H. (2020). Design of a smart manufacturing system with the application of multi-access edge computing and blockchain technology. IEEE Access, 8, 28659-28667.https://doi.org/10.1109/ACCESS.2020.2972284

[59] Wang, S., Taha, A. F., Wang, J., Kvaternik, K., & Hahn, A. (2019). Energy crowdsourcing and peer-to-peer energy trading in blockchain-enabled smart grids. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 49(8), 1612-1623.https://doi.org/10.1109/TSMC.2019.2916565

[60] Xu, C., Wang, K., & Guo, M. (2017). Intelligent resource management in blockchain-based cloud datacenters. IEEE Cloud Computing, 4(6), 50-59.https://doi.org/10.1109/MCC.2018.1081060

[61] Nadeem, S., Rizwan, M., Ahmad, F., & Manzoor, J. (2019). Securing cognitive radio vehicular ad hoc network with fog node based distributed blockchain cloud architecture. International Journal of Advanced Computer Science and Applications, 10(1), 288-295.https://dx.doi.org/10.14569/IJACSA.2019.0100138

[62] Li, M., Zhu, L., & Lin, X. (2018). Efficient and privacy-preserving carpooling using blockchain-assisted vehicular fog computing. IEEE Internet of Things Journal, 6(3), 4573-4584.https://doi.org/10.1016/j.vehcom.2020.100250