**Abstract**

This research project is the merger of IoT based decentralized application with public blockchain based on Hyperledger fabric V2.4 and synchronization of NFTs in mobile app. Its multi-tier application. Main purpose of this decentralized application is to collect sensor values securely and sending those values to blockchain with user-based OAuth 2.0 security protocols, then fetching them in raspberry pi from blockchain as well as on mobile phone application by scanning QR code from raspberry pi. Further researchers and developers can use this decentralized blockchain network for storing their IoT devices (Sensor’s) values on network by using given network API’s. This project also provides detail information. Therefore, future scope of this project is to build more IoT based applications that can collaborate with network.

Contents

[Title: 6](#_Toc132843952)

[Introduction 6](#_Toc132843953)

[Chapter 1 6](#_Toc132843954)

[1.0 Project rationale 6](#_Toc132843955)

[1.1 Project Scope: 6](#_Toc132843956)

[1.2 Project Aim: 7](#_Toc132843957)

[1.3 Project Objectives 8](#_Toc132843958)

[1.4 Project Deliverables 8](#_Toc132843959)

[1.4.1 Raspberry Application Ubuntu based 8](#_Toc132843960)

[1.4.2 Android Application 8](#_Toc132843961)

[1.5 Summary of chapter 1 9](#_Toc132843962)

[Chapter 2 9](#_Toc132843963)

[2 Literature review: 9](#_Toc132843964)

[2. 1 Network Research for IoT: 9](#_Toc132843965)

[2.1.1 Cloud storage: 9](#_Toc132843966)

[2.1.2 Blockchain Storage 10](#_Toc132843967)

[2.2 Private vs Public blockchain data storage 11](#_Toc132843968)

[2.3 Security Measures 11](#_Toc132843969)

[2.3.1 Network Bearers Token 12](#_Toc132843970)

[2.3.2 JWT(JSon Web Token) 12](#_Toc132843971)

[2.3.3 ERC-721 smart contract 13](#_Toc132843972)

[2.3.4 Consensus mechanism 13](#_Toc132843973)

[2.3.5 Cryptography By network 14](#_Toc132843974)

[2.4 IoT System Designing: 15](#_Toc132843975)

[Resource constraints: 15](#_Toc132843976)

[Scalability: 15](#_Toc132843977)

[2.4.3 Interoperability 15](#_Toc132843978)

[2.4.4 Security 16](#_Toc132843979)

[2.4.5 Communication with Network 16](#_Toc132843980)

[2.5 IoT system and network scaling 17](#_Toc132843981)

[2.5.1 Load balancing 17](#_Toc132843982)

[2.5.2 Sharding 17](#_Toc132843983)

[2.5.3 Off-chain transactions 18](#_Toc132843984)

[2.5.4 Sidechain transactions 18](#_Toc132843985)

[2.6 Summary of chapter 2 19](#_Toc132843986)

[Chapter 3 19](#_Toc132843987)

[3.0 Research Design 19](#_Toc132843988)

[3.1 Hypothesis Formulation 20](#_Toc132843989)

[3.2 Formulation of sets to test hypothesis 21](#_Toc132843990)

[3.2.1 Rest API network communication with user authentication with security layer of JWT 21](#_Toc132843991)

[3.2.2 Capacitive Soil moisture sensor Relative percentage calculation 22](#_Toc132843992)

[3.2.3 Hardware Communication through Serial port & MQTT 25](#_Toc132843993)

[3.3 Testing Results & Analysis: 25](#_Toc132843994)

[3.3.1 Soil Moisture Relative percentage formulation Result 25](#_Toc132843995)

[3.3.2 Security layer: User authentication with API call 26](#_Toc132843996)

[3.3.3 Hardware communication MQTT 26](#_Toc132843997)

[3.4 Summary of chapter 3 27](#_Toc132843998)

[Chapter 4 27](#_Toc132843999)

[4.0 Private Blockchain overview w.r.t of Hyperledger fabric 27](#_Toc132844000)

[4.1 Why we selected Hyperledger fabric: 27](#_Toc132844001)

[4.1.1 Comparison between Hyperledger Fabric and Corda 28](#_Toc132844002)

[4.1.2 Comparison between Hyperledger Fabric vs Quorum 29](#_Toc132844003)

[4.2 Hyperledger fabric network designing 29](#_Toc132844004)

[4.3 Hyperledger Fabric Layers: 30](#_Toc132844005)

[4.3.1 Application layer: 30](#_Toc132844006)

[4.3.2 Chaincode layer 30](#_Toc132844007)

[4.3.3 Ledger layer 30](#_Toc132844008)

[4.3.4 Communication layer: 31](#_Toc132844009)

[4.3.5 Membership services layer: 31](#_Toc132844010)

[4.4 Hyperledger Fabric Channels 31](#_Toc132844011)

[4.5 Hyperledger fabric Organizations 32](#_Toc132844012)

[4.6 Hyperledger Fabric using Docker Swarm 33](#_Toc132844013)

[4.7 Hyperledger Fabric x509 cryptographic 33](#_Toc132844014)

[4.8 Hyperledger fabric Node 34](#_Toc132844015)

[4.9 Hyperledger fabric Peer 34](#_Toc132844016)

[4.10 Hyperledger fabric Orderer 34](#_Toc132844017)

[4.11 HLF Clients 34](#_Toc132844018)

[4.12 Summary of chapter 4 34](#_Toc132844019)

[Chapter 5 35](#_Toc132844020)

[5.0 System Architecture Development - Hyperledger Fabric with Soil Moisture Sensors(IoT) 35](#_Toc132844021)

[5.1 User Interface 35](#_Toc132844022)

[5.1.1 ElectonJs application consists of several screens 35](#_Toc132844023)

[5.1.2 Android application consists of screens 37](#_Toc132844024)

[5.2 IoT side application architecture 38](#_Toc132844025)

[5.2.2 Python Middleware 39](#_Toc132844026)

[5.2.3 ElectronJs Web Endpoint Application 39](#_Toc132844027)

[5.3 Server-Side Network architecture 39](#_Toc132844028)

[5.3.1 Hyperledger Fabric Network 39](#_Toc132844029)

[5.3.2 NodeJs server-side API’s: 40](#_Toc132844030)

[5.4 Blockchain side ERC-721 Smart Contract 41](#_Toc132844031)

[5.4.1 Minting Token Function: 41](#_Toc132844032)

[5.4.2 Client Minted NFT Function: 41](#_Toc132844033)

[5.4.3 Total Supply Function 41](#_Toc132844034)

[5.5 Mobile-Side Application Development 41](#_Toc132844035)

[5.5 System Architecture Diagram: 42](#_Toc132844036)

[5.7 System Information flow chart 43](#_Toc132844037)

[5.8 Summary of chapter 5 45](#_Toc132844038)

[Chapter 6 45](#_Toc132844039)

[6.1 System Hardware & Software Requirements 45](#_Toc132844040)

[6.2 Android Mobile: 45](#_Toc132844041)

[6.3 Raspberry Pi 45](#_Toc132844042)

[6.4 Arduino Board: 46](#_Toc132844043)

[6.5 Arduino Sensor Kit: 46](#_Toc132844044)

[6.6 Summary of chapter 6: 47](#_Toc132844045)

[Chapter 7 47](#_Toc132844046)

[7.1 Conclusion 47](#_Toc132844047)

[References 48](#_Toc132844048)

[Appendix: 56](#_Toc132844049)

[Appendix A - Research project plan 56](#_Toc132844050)

[Appendix D: ElectronJs client side user login API call code: 68](#_Toc132844051)

[Appendix E: ElectronJs(Client Side)User Registration API call code: 69](#_Toc132844052)

[Appendix F: Pushing sensor data to network API call code: 71](#_Toc132844053)

[Appendix G: ElectronJs Retrieving User NFTs(sensor data) API call code: 71](#_Toc132844054)

[Appendix H: Python Middleware Code: 73](#_Toc132844055)

[Appendix I: Arduino Kit Code: 75](#_Toc132844056)

[Appendix J: User Registration: 77](#_Toc132844057)

[Appendix K: User Login: 78](#_Toc132844058)

[Appendix L: Invoking Transaction On Chain code: 79](#_Toc132844059)

[Appendix M: Query Transaction On Chaincode: 81](#_Toc132844060)

[Appendix N: Server side User Authentication with JWT token: 82](#_Toc132844061)

[Appendix O: Mobile Application Code: 83](#_Toc132844062)

[Appendix P: Query chaincode on network 88](#_Toc132844063)

[Appendix Q: Smart contract ERC-721 minting NFT on blockchain 90](#_Toc132844064)

[Appendix R: Smart contract ERC-721 Client Minted 91](#_Toc132844065)

[Appendix S: Smart Contract ERC-721 Total supply 91](#_Toc132844066)

# Title:

**Securing Sensor Data Using Private Blockchain Hyperledger Fabric Network**

# Introduction

Decentralization of data produced by the soil moisture sensors in real time by using private blockchain based on Hyperledger fabric as a storage with extra layer of security through user authentication token for data transmission from sensors to blockchain network.

# Chapter 1

## 1.0 Project rationale

In modern times, Sensors all around the globe are performing actions and generating information which is getting stored on centralized cloud. IoT industry is gradually growing and started its journey towards blockchain as a data storage, To offer people services for data sharing and storage in the future, a decentralized storage strategy is required (Wang et al., 2018). It is necessity to create a prototype for IoT devices which can communicate with decentralized authority.

Blockchain technology has the potential to improve the security, scalability, and interoperability of IoT systems. Blockchain technology combines processing, and storage of data, the cryptography, consensus protocols, a P2P network, and smart contracts (Da Xu et al., 2021). While using blockchain as a storage, its necessary to build an application where sensors can communicate with blockchain by using extra security protocols where devices i.e. Soil Moisture sensors can independently communicate with the blockchain network, main purpose is to securely transmit a package over http protocol to blockchain network through REST Json API call, Blockchain end prototype will help whole IoT community to use blockchain network built on Hyperledger Fabric with security.

## 1.1 Project Scope:

Based on the idea of collaboration of IoT industry with blockchain where hardware sensor-based devices can communicate with disturbed ledger on blockchain network built on Hyperledger Fabric not being as blockchain node but as independent device with internet, with no third party dealer engagement and decision making through consensus base mechanism which handles all nodes on network.

Implementation of Soil Moisture sensor system and using blockchain technology as a storage with the authentication of user, reading and converting Soil Moisture Sensor (SMS) analogue values to digital values and further passing information to blockchain through API’s, then retrieval of information on android based mobile application and on raspberry pi.

## 1.2 Project Aim:

After taking into account the key areas of the project from the initial research question, the project's aims were as follows:

• Secure communication process between sensors, blockchain and mobile by using authentication bearer token.

• Every user will be able to join blockchain network and can register, login, fetch their digital NFT assets from distributed ledger.

• Building prototype of two applications, Linux ubuntu based application and android application.

• Linux based Electron JS application will read sensor values and automatically pass it on to blockchain after a specific interval of time.

• Real time processing, fetching sensor values in android application from blockchain with secure connectivity through QR Image.

• Setting up blockchain network on Hyperledger fabric, channel creation, organization setup, chaincode and APIs development and deployment.

Aim of this project is to provide smooth and secure flow of information through applications, by using this prototype IoT based applications can store their values as an NFT on a decentralized storage and can retrieve NFTs on mobile phone with the implementation of security protocols.

## 1.3 Project Objectives

Creation of a distributed ledger by using Hyperledger Fabric, by following network requirements i.e., dockers, channel, organization, chaincode, consensus mechanism and attaching network end with node.JS APIs.

A Linux-based application in which system collects sensor values over a set interval and sends them to a distributed ledger with the right user ownership. After a predefined time period, the system sends data collected to the blockchain for users who own that particular set of digital information.

Mobile phone application development for displaying NFTs (User sensor values) from distributed ledger, which can only be retrieve and viewed by an authorized user. In this process, there is no login or register, this can be achieved through QR scanning, which gets displayed on the arduino board LED, and after scanning with a mobile camera from mobile application, the user will be able to see his NFTs.

## 1.4 Project Deliverables

### 1.4.1 Raspberry Application Ubuntu based

ElectronJs decentralized application for collecting several soil moisture sensors data given below and securely transmitting it to distributed ledger on blockchain through APIs.

• Soil Moisture Sensor

• Soil Temperature & humidity Sensor

• Air Pressure Sensor

• Acceleration sensor

### 1.4.2 Android Application

Main purpose is to display user NFTs sensor values from Hyperledger Fabric distributed ledger through API calls with the security measurement check. User can login by scanning QR code which will get display on screen. QR code contains the bearer token, which will provide access to user to get his NFT’s from blockchain.

## 1.5 Summary of chapter 1

This chapter is defining what are our objectives and aim for producing this software prototype, in order to achieve the perspective of blockchain merger with Sensors and providing user interface in raspberry pi and mobile applications, private blockchain gets use as a storage memory for sensor values with user based security authentication.

# Chapter 2

## 2 Literature review:

For this project literature concern areas are development of a smooth process where architectures of several applications, network designing and consensus mechanism implementation along with communication between devices can happen with security protocols. Further Table [(1.0)](#LiteratureReviewTable) outlines the flow of architecture and justification of each area of research.

[Table (1.0)](#LiteratureReviewTable)

|  |  |
| --- | --- |
| Literature Area | Justification for Research |
| Network Research | Cloud storage vs Blockchain storage (Private/Public) |
| Security Measures for Network | Bearer Token, JWT, ERC 721 smart contract, Peers endorsements, consensus mechanism. |
| IoT System Designing | Sensor’s values bridge through IP with standard protocols.  IoT hardware challenges (Resource,constraints,Interoperability,Security  Regulations)  Communication with Network |
| System Testing | System and Network scaling standards and feasibilities. |

## 2. 1 Network Research for IoT:

### 2.1.1 Cloud storage:

Cloud storage refers to a model of storing and accessing data over the internet, rather than locally on a hard drive or other physical storage device. It involves storing data on servers that are maintained by a cloud storage provider, which can be accessed from anywhere with an internet connection. In cloud storage data is stored and maintained on third party servers, rather than on dedicated servers which gets used in traditional networked data storage(Wu et al., 2010). Cloud consists of a centralized server deal by an administrator, or a person. There is always a chance of record tempering by an individual so in coming era cloud computing will make security and privacy more of a problem. As well as on the IoT side, data security would be a problem. In a similar context, privacy would be a subject of greater concern too (Aazam et al., 2014).

### 2.1.2 Blockchain Storage

Blockchain is a decentralized P2P network connected via a consensus protocol, in contrast to traditional transactions that depend on the central authentication of high credit units like banks and governments. As a result, transactions on the network are validated and shared among all participating nodes (computers) in the network (Yang et al., 2020).

Blockchain technology is a distributed database that allows multiple parties to record transactions on a shared ledger in a secure and transparent way, It involves storing data on a decentralized network of computers as a nodes that are connected through a distributed ledger. Blockchain has the potential to dominate almost any domain through its structure, which makes it very difficult to create fraudulent records, and architecture, it also encourages security features. Despite the fact that blockchain-based tools and applications may vary across different industries, generally blockchain approaches have a similar organizational structure (Berdik et al., 2021).

Nodes on the network stores a copy of the data, which helps to ensure that it is secure and can't be altered without the consensus of all network nodes. “The idea behind the blockchain is to store authorized transactions in a way that cannot be altered and remained tamper-proof (Samaniego, M., Jamsrandorj, U., & Deters, R., 2016). The decentralized nature of a blockchain also makes it difficult for a single entity to alter the data stored on it, since the network is distributed across multiple nodes. This increases the authentication and security of information. The blockchain is a decentralized, transparent, secure, validated, immutable, and pseudo-anonymous database (Bambara et al., 2018)

## 2.2 Private vs Public blockchain data storage

Public blockchains are not well-suited for data storage of sensors due to their decentralized nature and the fact that they are open to anyone. Because anyone can access and participate in a public blockchain, it is not a secure place to store sensitive or private data, as the data is stored on a decentralized network of nodes. This means that there is no central point of control or access. To access data on a public blockchain, one can use a blockchain explorer, which is a tool that allows users to view and search for transactions and other data on the blockchain. Additionally, there are a variety of software development kits (SDKs) and application programming interfaces (APIs) that can be used to interact with the blockchain and access data.

With a growing number of enterprises who wish to adopt blockchain technology in order to support their operations and maximize their use of resources like IoT devices, supply chains and distribution networks, private blockchain has become essential (Assaqty et al., 2020). A private blockchain can be used as a storage solution because it offers a secure and decentralized way to store data. Transactions on a private blockchain are validated by a pre-selected group of nodes, rather than the entire network, which allows for faster transaction processing and increased privacy. The data stored on a private blockchain is also immutable, meaning it cannot be altered or tampered with, providing an additional layer of security. Additionally, private blockchains can be configured to meet the specific needs and requirements of the organization or industry using it, along with that it also requires permission to read, write, or participate. A private blockchain is usually permissioned, meaning that only authorized users are allowed to access it. This makes private blockchains more secure, as they are not exposed to attacks by malicious actors. Private blockchains are often used in enterprise settings, where they can be used to securely and efficiently track data and assets of an organization.

## 2.3 Security Measures

For securing a network built over Hyperledger fabric its necessary to ensure the confidentiality and integrity of data transmitted between network nodes, you should use secure communication channels such as TLS/SSL. As well as enabling access control features, such as identity-based policies and certificate authorities, to control which users and applications are allowed to access the network and perform specific actions and implementation of security protocols. Advanced encryption or other security measures cannot be supported by sensors that measure humidity or temperature (Srivastava et al., 2020), due to the limitation of the hardware its necessity to develop an independent approach in which device can communicate with blockchain. A typical node in an IoT ecosystem doesn't typically need to have the processing power that a blockchain does i.e., Raspberry Pi, smartwatch, embedded system (Pešić et al., 2019) For communication with private blockchain we have to develop REST API communication, In between the end-user layer and the micro-service layer, the RESTful APIs serve as a manager. It will take data from users and send it in json format to a microservice (Mukherjee et al., 2020).

### 2.3.1 Network Bearers Token

A type of security token that represents a set of permissions or privileges that are granted to the holder of the token. In a Hyperledger Fabric network, bearer tokens can be used to authenticate clients and authorize them to perform certain actions, such as invoking chaincode or querying the ledger on private blockchain. Bearer tokens are typically issued by an identity provider and can be passed around between different parties, allowing the privileges they represent to be transferred along with the token. Bearer tokens give their owners access to the protected resource. OAuth 2.0 uses bearer tokens by default and TLS (Transport Layer Security) is used to ensure secure communication between the various parties (Siris et al., 2020).

When a client wants to access a protected resource, it can send a request to the API call along with a bearer token in an Authorization header. The API can then verify the token and, if it is valid, allow the client to access the requested resource. Any party in possession of a "bearer token" can use it without proving they have a cryptographic key (Zheng & Jiang, 2014).

### 2.3.2 JWT(JSon Web Token)

JSON Web Tokens (JWTs) are a standardized way to represent claims securely between two parties. They are often used to authenticate API requests and can be easily transmitted using a JSON object. After the user credentials have been validated, the authentication service generates a JWT from the retrieved user authorization data. This token is then encrypted using the JWT secret before being sent back to the client and is not stored on the server (Jánoky et al., 2018)

JWTs consist of three parts: a header, a payload, and a signature. The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA. JWTs can be used independently with a wide variety of technologies because they are flexible and platform independent (Akanksha & Chaturvedi, 2022). JWTs serve as the authentication of the authorized application because they are based on JSON web signing (JWS) and JSON web encryption (JWE)(Solapurkar, 2016). The payload contains the claims. Claims are statements about an entity (typically, the user) and additional data. The signature is used to verify that the sender of the JWT to ensure that the message wasn't changed along the way.

### 2.3.3 ERC-721 smart contract

ERC-721 is a standard for non-fungible tokens on the Ethereum blockchain. Non-fungible tokens (NFTs) represent unique assets, such as collectibles, art, or other items of value. Each NFT is unique and cannot be replaced or exchanged for another token. The ERC-721 standard defines a minimum interface to allow unique tokens to be managed, owned, and traded (Pirker et al., 2021).

In our scenario it is required to convert IoT sensor value to an NFT. HLF does not natively support ERC-721, but we can use ERC-721 tokens with HLF by building a custom smart contract (also known as chaincode in Hyperledger Fabric). Its necessary to build a chaincode that implements the ERC-721 functions, such as ‘transferFrom’, ‘approve’, and ‘getApproved’. Further can client application that can interact with the chaincode on HLF to create, manage, and transfer the ERC-721 tokens. The management of IoT devices as singular, indivisible entities is proposed to use Non Fungible Tokens (NFTs) based on the ERC-721 standard (Arcenegui et al., 2020).

One of the key security considerations for ERC-721 contracts is the proper handling of user-provided data. As with any smart contract, it is important to validate any input data to ensure that it is in the expected format and within the allowed bounds. Non-fungible tokens (NFTs) are a unique class of token that have the ability to uniquely identify objects (Chirtoaca et al., 2020).

### 2.3.4 Consensus mechanism

Consensus mechanism is a process used by blockchain networks to ensure that all participants agree on the contents of the shared ledger. It is an essential component of blockchain technology, as it allows the network to reach agreement on the state of the ledger without the need for a central authority.

Few of the mechanisms are shared below

There are several different consensus mechanisms that have been developed for use in blockchain networks, including:

* Proof of Work (PoW): This is the most well-known consensus mechanism, and it is used by networks like Bitcoin and Ethereum. In a PoW system, participants called miners. “Proof-of-work is essentially one-CPU-one-vote” (Nakamoto, 2008) or in other words on network one node can make one vote.
* Proof of Stake (PoS): PoS is an consensus mechanisms in which participants act as validators are chosen to create the next block in the chain based on how many stake (or coins) they hold in the network. PoS can be represented as a probabilistic process in which the probability of a validator being chosen to create the next block is proportional to their stake (or ownership) in the network.
* Delegated Proof of Stake (DPoS): This is a variant of PoS in which participants known as delegates are voted in by the community to create the next block in the chain.

### 2.3.5 Cryptography By network

Cryptography is a key component of blockchain technology that helps to make it secure. It is used to secure the communication channels between different components of the blockchain system, as well as to secure the data stored on the ledger. For data privacy, integrity, and authentication on the blockchain, public-key cryptography and hash functions are used (Fernandez-Carames & Fraga-Lamas, 2020). One of the main ways that cryptography makes blockchain secure is by protecting the privacy of participants. In a blockchain system, each participant is identified by a unique cryptographic identity, which is managed using public-key cryptography. This allows participants to prove their identity without revealing their private key, ensuring that their transactions and activities on the network are kept private, trust can be built by using cryptography and a large amount of users (Kaushik et al., 2017)

Cryptography is also used to ensure the integrity and authenticity of transactions on the blockchain. Digital signatures, which are created using cryptographic algorithms, are used to prove that a transaction was created by the sender and has not been tampered with in transit. This helps to prevent fraud and ensures that the blockchain remains a tamper-evident record of all transactions. Cryptography is also used to secure the communication channels between different components of the blockchain system. Protocols like SSL (Secure Sockets Layer) and TLS (Transport Layer Security) use a combination of public-key and symmetric-key cryptography to encrypt the data transmitted between components and to authenticate the identity of the communicating parties.

## 2.4 IoT System Designing:

There are plenty of challenges that IoT devices face when it comes to using blockchain technology:

### Resource constraints:

IoT devices often have limited resources such as processing power, storage, and battery life, which can make it difficult to run blockchain software on these devices. “IoT devices have limited resources and these devices are connected to the internet.” (Mustafa et al., 2016) Because of the limited memory and processing speed of these devices are very low to and makes it difficult to integrate Hyperledger Fabric software and use IoT device as a node connected to network.

### Scalability:

The number of IoT devices is expected to grow significantly in the coming years, which could put a strain on existing blockchain networks and its necessary to manage sensors producing information to secure it. Devices must establish secure connections to their corresponding backend systems because the data they transmit is frequently confidential or sensitive. Before the data leaves the device, it must be encrypted (Gupta et al., 2017)

### 2.4.3 Interoperability

IoT devices use a variety of protocols and standards, which can make it difficult to integrate them with a blockchain network. Interoperability issues with IoT in connection with network can include:

Protocol incompatibility: Different devices and networks may use different communication protocols, such as Zigbee, Z-Wave, or Bluetooth, which can make it difficult for them to communicate with each other.

Network fragmentation: IoT devices may be connected to different networks, such as cellular networks, Wi-Fi networks, or low-power wide-area networks (LPWANs), which can lead to fragmentation and make it difficult for devices to connect and communicate with each other.

Security and privacy concerns: Different devices and networks may have different security and privacy protocols and standards, which can make it difficult to ensure that data is secure and protected when it is transmitted between devices and networks.

Limited scalability: Many IoT devices and networks have limited capacity, which can make it difficult to scale up the number of devices and networks that can connect and communicate with each other.

To overcome these issues, IoT networks and devices need to be designed with interoperability in mind and standardization is essential to make it easier to connect and communicate between different systems and devices.

### 2.4.4 Security

IoT devices are often targets for cyber-attacks, and securing them against these threats is a major challenge. The default software configuration, inconsistent software updates, and a long time between patch release and patch installation make it simple for cybercriminals to attack IoT devices (Gurunath et al., 2018).

### 2.4.5 Communication with Network

We looked at recent work done in the creation of interesting and useful applications using low-cost development boards like the Raspberry Pi and Arduino. Home automation, patient monitoring programs, and systems for monitoring the environment and the weather are some of the popular applications created using these boards (Zafar et al., 2018). To overcome the challenges while communicating with blockchain network there are several approaches and protocols to send sensor values from an Arduino to an API (Application Programming Interface):

1.REST API: This allows the Arduino to send HTTP requests to the API with the sensor data as part of the request. This can be done using the Arduino's and communication of sensors data with a serial port connection to a separate device “Raspberry Pi” which will read data from serial port and further pass it to API call. REST API even makes communication much easier with third party servers. Its light weight and human readable nature is another factor. It aims to make the API more readable and understandable for humans by working with the description key value of controllers, devices, services, actions, and state variables. (Ferreira et al., 2013)

2.HTTP POST: One of the option is to use the Arduino's built-in Ethernet or WiFi capabilities to send an HTTP POST request to the API with the sensor data as the payload. This can be done using the Arduino's Ethernet library or the WiFi library. The HTTP Secure Protocol (HTTP using Secure Sockets Layer encryption) is used by websites like Google, the Raspbian OS enables the Raspberry Pi to connect HTTP Post. Occasionally, micro-controller-based devices might not be able to do that. So, microcontrollers must connect via proxy servers running the straightforward HTTP application layer protocol (Kulkarni et al., 2016). But this approach is not much feasible.

3.MQTT: Another option is to use the MQTT protocol, which is designed for lightweight communication between devices. The Arduino can publish sensor data to an MQTT broker, which can then forward the data to the Raspberry Pi though communication with server based broker which can further communicate with API. It is Originally developed by Andy Stanford-Clark and Arlen Nipper, MQ Telemetry Transport (MQTT) is a messaging protocol that uses a publish/subscribe mechanism. OASIS (Organization for the Advancement of Structured Information Standards) currently using it as a standard. (Andy et al., 2017)

## 2.5 IoT system and network scaling

Scaling an IoT (Internet of Things) system and a blockchain network is challenging due to the large number of devices and the amount of data being transmitted. We can use some approaches to scale these types of systems:

### 2.5.1 Load balancing

One way to scale an IoT system is to use load balancing techniques to distribute the workload across multiple servers or devices. This can help to prevent any one device from being overwhelmed by the volume of data being transmitted. Without load balancing, a coordinator with an excessive number of wireless sensor nodes might consume more energy (Kim & Kim, 2016)

### 2.5.2 Sharding

The scalability problem with blockchain protocols is being addressed by the use of sharding, a concept that was first used as a classical database design principle (Chow et al., 2018). Blockchain networks can be scaled using sharding, which involves dividing the network into smaller pieces called shards and distributing the workload across these shards. This allows the network to process more transactions in parallel and can improve the overall scalability of the system.

## 2.5.3 Off-chain transactions

Off-chain transactions refer to transactions that take place outside of the blockchain but are still related to it. They can be used to improve the scalability and performance of a blockchain by moving some of the transaction processing off of the blockchain's main network. One example of off-chain transactions is the use of payment channels, such as the Lightning Network for Bitcoin, which allows for multiple transactions to occur between two parties without each one being recorded on the blockchain. This reduces the number of transactions that need to be processed on the blockchain and can help to speed up the overall transaction process. Off-chain transactions can help to increase the efficiency and speed of a blockchain, but they also introduce new security risks and challenges as they move away from the security guarantees provided by the blockchain's decentralized network. Through distributed software architecture that integrates off-chain resources with the blockchain network, Off-Chain Blockchain Systems (OCBS) make it possible to process and manage information (KenMiyachi et al., 2021).

## 2.5.4 Sidechain transactions

A sidechain is a separate blockchain that is attached to a main blockchain, We call the first blockchain the parent chain and the second just the sidechain (Back et al., 2014), allowing for the transfer of assets or information between the two. According to a research analyses, sidechain is a secondary blockchain that is linked to the primary blockchain. Unlike the mainchain's protocol, sidechains may have their own consensus protocols that are entirely distinct from it. (Singh et al., 2020)

The main advantage of sidechains is that they allow for more flexibility and scalability in the blockchain network. By moving some of the transactions and data processing to the sidechain, the main blockchain can focus on processing the most critical and secure transactions. Sidechains can have different characteristics and rules than the main blockchain, for example, a sidechain can have a different consensus mechanism, or can be used for a specific use case like smart contracts or micropayments. Some popular examples of sidechains include Liquid, a sidechain for Bitcoin, and Ethereum's Plasma, which is a proposed sidechain for the Ethereum blockchain.

Sidechains can improve the scalability and flexibility of a blockchain network, they also introduce new security risks and challenges. Because sidechains are connected to the main blockchain, any vulnerability or attack on a sidechain could potentially affect the security and integrity of the main blockchain. Therefore, it's important to thoroughly evaluate the security of any proposed sidechain before implementing it.

## 2.6 Summary of chapter 2

In this chapter we are focusing on cloud storage and blockchain storage and the network research and security parameters, also essential measures like interoperability, scalability, security and communication of IoT devices with network and approach recommendation for communications protocol.

# Chapter 3

## 3.0 Research Design

Based on the system design, the purpose of this project is to merge IoT devices real time data with blockchain network with enhanced security protocols. Maintaining the devices linked to sensors in an IoT context and mining the blockchain will always be a technical problem (Shyamala Devi et al., 2019). IoT hardware devices are not much progressive in processing and memory due to which we have to look for an independent solution that can communicate with sensors and store sensor values. We determined that the internet of things will not become a complete member of a blockchain network because of the high-end hardware requirements. However, the internet of things will gain benefits from the functionality given by blockchain technology via the APIs provided by blockchain network nodes (Singh et al., 2018). So, there is still need of the development of the system where sensors can communicate with blockchain without installing a network node on hardware, from other point of view regards to author (Goudos et al., 2017). One of the most often used application layers is HTTP. HTTP is verbose and quite complex, making it unsuitable for use on IoT nodes, solution to resolve this issue as proposed by author for the data transmission between clients and servers over HTTP is made easier by REST API’s. In the light of IoT industry rise privacy and security concerns are becoming extremely important with the perspective of Internet-of-Everything (IoE) movement, which involves application-specific IoTs too (Sadawi et al., 2021).

We have adopted deductive approach for this research with scientific investigations from multiple researches. The hypothesis is fundamental to determining if the aspects of it are correct or incorrect. The hypothesis's testability is crucial for determining its validity (Toledo et al., 2011). We have generated hypothesis on the base of several key variables.

1. Capacitive Soil Moisture Sensor percentage calculation.
2. Hardware communication through serial port and MQTT protocol.
3. Rest API network communication with security.

## 3.1 Hypothesis Formulation

Soil moisture sensors of both the capacitance and conductance types are available. Due to their capacity to reduce the impact of ionic activities, which are frequently observed in cultivated soil, out of which capacitance type sensors are advantageous among them (Radi et al., 2018). Making selection for capacitance type sensor having voltage 1.0, its analogue signal value ranges between 0 - 1023, due to which implementation of formulation is needed to exactly find out the soil moisture. To resolve this we need to calculate the sensors values we have to implement relative range percentage formulation.

Percentage = ((CRV – MN)\* 100)/(MX - MN)

* Minimum Value Recorded during specific interval of time = MN
* Maximum Value Recorded during specific interval of time = MX
* Current Raw Value = CRV

There has been a critical need of protocol for information sharing and transmission between multiple devices with low processing memory (Moses et al., 2007). Communication of sensors from Arduino board to raspberry pi based electron.js application is achievable through serial communication; serial communication configurations are usually used for reading data values which gets send from the module sensors.

Serial connectivity between the Arduino and Raspberry is disrupted while communicating with NodeJs after an interval of time (Basic, 2018). By using ElectronJs derived from the NodeJS architecture the serial port is only readable when Arduino IDE serial monitor display is in execution state, otherwise it often returns undefinable and garbage values. One of the solution is by python programming, the values received at the Raspberry's pin via serial connection should always be readable from the port (Princy & Nigel, 2015). In context of serial communication from port is also addressed as, Python is a robust programming language.

Engagement of devices after specific period of time and reloading of new sensors values, we also needs some service that can communicate with Arduino, raspberry Pi and mobile. An IoT device's do have the ability to transmit messages in order to communicate with other devices. In particular, the MQTT protocol, which has been implemented in numerous IoT devices and instant messaging systems, it is also a lightweight protocol that works well on low-power devices (Hwang et al., 2016). MQTT protocol engage devices on network and can broadcast message to the authentic users.

Security is a critical essence while communication between devices and network. The research also outlined four strategies for enhancing IoT security, including configuring IoT, finding valid IoT, authenticating users, and adopting blockchain technology for secure communication (Singh et al., 2018). As this research shows that one of the major implementations for security is OAuth on the end points of communication i.e. REST API, for providing enhanced user authentication mechanism while communicating with blockchain. As proposed by the researchers while development of car registration system with blockchain, The backend uses SDK of Hyperledger Fabric for node to make connection and invoking the chaincodes on blockchain and securing communication between the browser and NodeJS server by using JSON Web Tokens (Tran et al., 2021). JSON Web Tokens, sometimes known as JWT for short, are an RFC Open Standard (RFC 7519) for securely exchanging claims—a type of data—between two parties, generally a server and a client. Digital signature and encryption methods allows safe transmission and verification of data(Vid Visočnik,2018). Derived by these research our current prototype proposed JWT authentication for users for secure communication.

## 3.2 Formulation of sets to test hypothesis

### 3.2.1 Rest API network communication with user authentication with security layer of JWT

#### Server-Side API

In order to test the hypothesis for JWT authenticity. We have Implemented NodeJs API’s which is hosted and working as an end point for communication with our own blockchain network based on Hyperledger fabric, and written functionality of encoding and decoding of JWT(JSon Web Token) for authorization of the user, described in server side NodeJs code. [Appendix](#ElectronJsRetrievingUserNFTs).

#### Client-Side API

In order to get data response from server where Hyperledger fabric is deployed, we have written device side(client side) API call code along with user authentication to get the response. [Appendix.](#ElectronJsRetrievingUserNFTs)

### 3.2.2 Capacitive Soil moisture sensor Relative percentage calculation

We have derived a set of instructions in arduino for testing of soil moisture sensor values reading, during initial test we placed soil sensor dry place, then in second round we have placed sensor inside the water jar.

|  |  |
| --- | --- |
| A picture containing text  Description automatically generated |  |

Executed Code:

Void setup(){

Serial.begin()

}

Void loop(){

Serial.Print(“Moisture Sensor Raw Value: ”);

Serial.Println(analog.read(A1));

Delay(2000);

}

|  |  |
| --- | --- |
| Graphical user interface, text  Description automatically generated with medium confidence | Graphical user interface, text  Description automatically generated with medium confidence |

|  |  |
| --- | --- |
| Dry Soil Moisture Sensor Raw Values | Wet Soil Moisture Sensor Raw Values |
| 501 | 719 |
| 499 | 720 |
| 489 | 720 |
| 495 | 717 |
| 490 | 723 |
| 495 | 721 |

We have generated the dataset of sensor-based values for calculation of soil moisture in percentage, we have implemented the formulation in code to find desired result.

**Python Based Code:**

int soil\_sensor = A1;

const int air\_value = 720; // Maximum reading value fetched

const int water\_value = 490; // Minimum reading value fetched

int Current\_raw\_moisture,moisturepercent;

void setup() {

// initialize serial communication at 9600 bits per second:

Serial.begin(9600);

Pressure.begin();

}

// the loop routine runs over and over again:

void loop() {

// Fetching Moisture value

Current\_raw\_moisture = analogRead(soil\_sensor);

//Calculation of Soil Moisture Percentage

moisturepercent = ((Current\_raw\_moisture - water\_value) \* 100)/(air\_value - water\_value);

if(moisturepercent<=0) // Conditional Statement

{

doc["moisture"] = 0;

}else if(moisturepercent>=100)

{

doc["moisture"] = 100;

}else{

doc["moisture"] = moisturepercent;

}

Serial.println(data\_out);

delay(10000);

}

### 3.2.3 Hardware Communication through Serial port & MQTT

We have tested the serial communication through port with baud rate 9600 which means it will transfer 9600 bits per second, but unfortunately, we failed to retrieve data from serial port “ttyACM0” in ElectronJs application which is deployed on raspberry pi.

So, there we faced a need to change approach as per hypothesis generated, first we were reading sensors values from serial communication port then we initialized a middleware of communication with python as this issue is also addressed in a way, python can communicate with arduino via serial communication from standard USB-COM port by object-oriented Programming language (Fatehsingh et al., 2020), and further python is communicating with Electronjs through MQTT protocol. So, we have installed MQTT broker on server and initialized in python, whenever python receives value from USB serial communication it broadcasts message to ElectronJs and mobile app.

## 3.3 Testing Results & Analysis:

### 3.3.1 Soil Moisture Relative percentage formulation Result

When we put soil moisture sensor in water and executed the code that is implementing the calculation of the soil moisture sensor given in [appendix](#ArduinoKitCode), It sends the response to ElectronJs application from where it gets transfer to distributed ledger, Here is the output, which is correct.

Graphical user interface, text, application

Description automatically generated

### 3.3.2 Security layer: User authentication with API call

Implemented JWT(Json Web Token) user security authentication on server side Hyperledger fabric end NodeJs API, For testing it we have made 2 calls to network by using postman platform, One is with correct authentication token and the other one is with wrong authenication and we have achieved desired results successfully, given in picture below:

**Picture with correct authentication JWT entry**

Graphical user interface, text, application, email

Description automatically generated

**Picture with wrong authentication JWT entry**

Graphical user interface, text, application, email

Description automatically generated

### 3.3.3 Hardware communication MQTT

Result sample by using the MQTT, on the right side of the screen we have generated a message as a broadcaster, on the left side we have received MQTT broadcasted message to the user. Hence it proves that we can communicate through MQTT.

Graphical user interface, text

Description automatically generated

## 3.4 Summary of chapter 3

In this chapter we have adopted deductive and qualitative approach for generation of our hypothesis, then made scenarios where we have tested hypothesis and successfully derived the results for our research.

# Chapter 4

## 4.0 Private Blockchain overview w.r.t of Hyperledger fabric

A private blockchain is a distributed ledger that is permissioned, meaning that only authorized parties are able to read, write, or validate transactions on the network. It is typically set up within an organization or consortium, and is used for various purposes such as record keeping, supply chain management, and asset tracking and IoT data storage is taking a rise.

Examples:

Some examples of private blockchains include Hyperledger Fabric, Corda, and Quorum.

### 4.1 Why we selected Hyperledger fabric:

It is an open source blockchain framework hosted by Linux foundation. One of the key features of Hyperledger Fabric is its modular architecture, which allows you to plug in different components such as consensus algorithms, membership services, and privacy enhancements. This allows you to customize your blockchain to fit your specific needs. Major reasons for the selection of Hyperledger fabric because of its:

Scalability: Hyperledger Fabric is designed to support a high number of transactions, making it suitable for use in IoT environments where there may be a large number of devices sending and receiving data.

Privacy: Hyperledger Fabric supports private transactions, allowing only authorized parties to view the data being transmitted on the blockchain. This is important for IoT applications where sensitive data may be involved.

Modularity: It also allows developers to choose from a range of modular components, such as consensus algorithms and membership services, to build a blockchain solution that meets their specific needs.

Interoperability: It is designed to be interoperable with other technologies, making it easy to integrate with existing systems and devices in an IoT environment.

### 4.1.1 Comparison between Hyperledger Fabric and Corda

Hyperledger Fabric and Corda are both private blockchain platforms that are designed for enterprise use cases. However, they have some key differences in terms of their architecture and design.

One of the main differences between Hyperledger Fabric and Corda is the way they handle transactions. Hyperledger Fabric uses a traditional blockchain architecture, where transactions are recorded in blocks and added to the chain. Corda, on the other hand, uses a different approach called a "distributed ledger" where transactions are recorded on a shared ledger, but are not necessarily grouped into blocks.

Another difference is the way they handle smart contracts. Hyperledger Fabric uses chaincode which is written in Go or JS, to define the business logic for transactions. Corda uses smart contracts written in Java or Kotlin.

In terms of performance, Hyperledger Fabric is generally more scalable than Corda, due to its modular architecture and support for parallel processing of transactions. Corda is more focused on data privacy and interoperability with other systems.

### 4.1.2 Comparison between Hyperledger Fabric vs Quorum

Hyperledger Fabric and Quorum are both private blockchain platforms that are designed for enterprise use cases. However, they have some key differences in terms of their architecture and design.

One of the main differences between Hyperledger Fabric and Quorum is the consensus algorithm they use. Hyperledger Fabric supports a variety of consensus algorithms, including Practical Byzantine Fault Tolerance (PBFT) and Kafka-based consensus. Quorum, on the other hand, uses a variant of the PBFT algorithm called Istanbul Byzantine Fault Tolerance (IBFT).

Another difference is the way they handle smart contracts. Hyperledger Fabric uses chaincode, which is written in Go, to define the business logic for transactions. Quorum uses smart contracts written in Solidity, the same language used by Ethereum.

In terms of performance, Hyperledger Fabric is generally more scalable than Quorum, due to its modular architecture and support for parallel processing of transactions. Quorum is more focused on enabling private and confidential transactions.

### 4.2 Hyperledger fabric network designing

Hyperledger Fabric is an open source framework for developing blockchain applications and networks. there are several key factors which we must understand before network designing:

#### Business requirements:

It's important to start by understanding the business needs that the network will be addressing. This will help determine the type of network (public, private, consortium) and the level of security and permissions required. In our scenario its private blockchain with the purpose of minting tokens in string format with the authentication of user and retrieval of NFTs.

#### Network architecture:

There are several different architectural patterns that can be used when designing a Hyperledger Fabric network. These include a single-organization network, a multi-organization network, and a network with channels. In our project we are using single organization.

#### Network topology

The topology of the network will depend on the number of organizations and nodes involved, as well as the geographic distribution of the nodes.

#### Consensus algorithm:

Hyperledger Fabric supports several different consensus algorithms, including solo, Kafka, and PBFT. The choice of algorithm will depend on the requirements of the network and the trade-offs between performance and fault tolerance.

#### Identity and access management:

It's important to carefully design the identity and access management system for a Hyperledger Fabric network to ensure that only authorized parties can participate in the network and access its data, In our scenario we are using Private SSH key as a passcode generated by the fabric and username as a login for security protocol which gets set by the user.

#### Smart contracts

The smart contracts that run on a Hyperledger Fabric network will depend on the business requirements and the specific use case being addressed. It's important to carefully design and test these contracts to ensure that they function as intended.

## 4.3 Hyperledger Fabric Layers:

### 4.3.1 Application layer:

This is the topmost layer of the Hyperledger Fabric network and consists of the applications that interact with the blockchain. Applications can be used to submit transactions, query the ledger, and invoke smart contracts.

### 4.3.2 Chaincode layer

The most important layer which executes smart contracts, which are programs that encode business logic and define the rules for interacting with the ledger. Chaincode can be written in a variety of programming languages, such as Go, Java, and javascript.

### 4.3.3 Ledger layer

Its main purpose is to maintaining the state of the blockchain and storing the transaction history. It consists of a distributed database that is replicated across all the peers in the network.

### 4.3.4 Communication layer:

Main responsibility of this layer is to enabling communication between the various components of the Hyperledger Fabric network, such as peers, orderers, and clients. It uses a gossip protocol to propagate transactions and ledger updates across the network.

### 4.3.5 Membership services layer:

It is responsible for managing the membership of the Hyperledger Fabric network, including identity verification and access control. It consists of a membership service provider (MSP) that defines the rules for membership in the network.

## 4.4 Hyperledger Fabric Channels

In Hyperledger Fabric, a channel is a private "subnet" within a network that allows a group of specific organizations to transact with each other and access a shared ledger. Channels provide a way to isolate the transactions of one group from those of another within a single network, and can be used to enable privacy and confidentiality within the network. A smaller group of transacting parties within a large consortium can establish channels in Fabric for confidentiality and privacy of transactions (Baliga et al., 2018) A channel consists of two or multiple participating organizations, which runs a peer node. These peer nodes communicate with one another and maintain a copy of the channel's shared ledger. The ledger contains a record of all the transactions that have occurred on the channel, as well as the current state of the channel.

Each channel has its own unique set of chaincode (smart contracts) and policies governing access to the channel, which can be customized by the channel's participants. This allows organizations to tailor the channel to their specific business needs and privacy requirements.

Channels are useful for scenarios where there is a need for private, confidential transactions between specific parties. For example, a channel could be used by a group of companies which facilitate secure, private transactions between themselves, without involving the larger network.

## 4.5 Hyperledger fabric Organizations

In Hyperledger Fabric, an organization refers to a business or other entity that is participating in a blockchain network. Organizations can be thought of as the "members" of the network and are responsible for maintaining their own nodes and endorsing transactions. Clarity and insight into business processes can be gained by implementing blockchain in an organization (Li et al., 2020).

Each organization in a Hyperledger Fabric network will typically have one or more "peers," which are the blockchain nodes that are managed by the organization. Peers can be divided into two categories: endorsing peers and committing peers. Endorsing peers are responsible for evaluating transactions and "endorsing" them as valid, while committing peers are responsible for adding endorsed transactions to the ledger. Fabric demands that all participating organizations be provisioned into the network during blockchain setup and that they be known beforehand (Baliga et al., 2018).

Organizations are also responsible for managing their own member services, which are used to manage the identities of users within the organization and control access to the network. Additionally, Fabric gives an organization the freedom to create its own Access-Control, Membership, Validation, and Consensus policies (Dabholkar & Saraswat, 2019).

To create an organization in Hyperledger Fabric, you will need to perform the following steps:

1. Choose the membership service provider (MSP) type: Hyperledger Fabric supports two types of membership service providers (MSPs): fabric-ca-server based MSPs and third-party certificate authority (CA) based MSPs. You will need to choose which type of MSP you want to use for your organization.
2. Set up the MSP: If you are using a fabric-ca-server based MSP, you will need to set up a fabric-ca-server and register it with the network. If you are using a third-party CA, you will need to obtain the necessary certificates and configure them for use with Hyperledger Fabric.
3. Create the organization's crypto material: You will need to generate the necessary cryptographic material for the organization, including keys and certificates. This can be done using the cryptogen tool that is included with Hyperledger Fabric.
4. Configure the organization's peers: You will need to set up one or more peers for the organization and configure them to use the crypto material that you generated in step 3.
5. Register the organization's users: You will need to register the users who will be authorized to participate in the network on behalf of the organization. This can be done using the fabric-ca-client tool.
6. Join the organization to the network: Once the organization's peers and users are configured, the organization can join the network by creating a channel and installing and instantiating any necessary chaincode (smart contracts).

## 4.6 Hyperledger Fabric using Docker Swarm

One way to deploy Hyperledger Fabric is using Docker Swarm, a native clustering tool for Docker with Docker Swarm, we can create a cluster of Docker nodes and deploy our Fabric network to the cluster. This allows us to easily scale your network and manage the availability of your distributed applications. A group of machines running Docker and combined to form a swarm (Marathe et al., 2019).

To deploy Hyperledger Fabric using Docker Swarm, we need to create a Docker Swarm cluster and then deploy the Fabric components, such as the peer nodes and orderers, to the cluster. There are a few different approaches you can take to do this, but one way is to use Docker Compose to define and deploy your Fabric network.

Using Docker Compose, you can define your Fabric network in a YAML file and then use the ‘docker stack deploy’ command to deploy the network to your Swarm cluster. This makes it easy to manage and update your Fabric network, as you can simply modify the Compose file and redeploy the network to apply the changes.

## 4.7 Hyperledger Fabric x509 cryptographic

X.509 is a public key infrastructure (PKI) standard that is extensively used to secure internet communication. X.509 certificates are used for authentication and authorization in Hyperledger Fabric. X.509 certificates are used in Hyperledger Fabric to identify the network's users, peers, and orderers. Each entity on the network is identified by a unique X.509 certificate that includes the name, public key, and digital signature of the entity. The security aims of confidentiality, integrity, and authenticity are attained with the aid of X.509 certificates, which are used in a number of cryptographic communication protocols (Kinkelin et al., 2020).

X.509 certificates are also for authentication of the identity of entities on the network and to establish trust between them. For example, when a client submits a transaction to the network, the transaction is signed with the client's private key and verified with the client's X.509 certificate. This ensures that only authorized clients can submit transactions to the network. The validity of a server's claim to its public key is currently largely verified by trusted third parties, according to SSL/TLS. Trusted third parties known as certificate authorities (CAs) certify servers' public keys by running audits and issuing X.510 certificates (Madala et al., 2018).

## 4.8 Hyperledger fabric Node

Nodes refer to the physical or virtual machines that run the peer and orderer processes in a Hyperledger Fabric network. Each node can host one or more peers or orderers.

## 4.9 Hyperledger fabric Peer

Peers are the main participants in the Hyperledger Fabric network and are responsible for executing transactions and maintaining the state of the ledger. Peers can be divided into two types: endorser peers and committing peers. Endorser peers simulate the execution of transactions and return the results to the client, while committing peers are responsible for validating transactions and committing them to the ledger.

## 4.10 Hyperledger fabric Orderer

Orderers are responsible for ordering transactions and creating blocks that are committed to the ledger. They ensure that transactions are delivered in a consistent order across the network.

## 4.11 HLF Clients

Clients are the applications or users that interact with the Hyperledger Fabric network by submitting transactions and querying the ledger. Clients can be divided into two types: user clients and system clients. User clients represent the users of the system, while system clients are used by the system to perform administrative tasks, such as deploying and upgrading smart contracts.

## 4.12 Summary of chapter 4

This provides a detail background knowledge of Hyperledger fabric and defines its structure and how it helps us to deploy our own network on server, along with the comparison with other private blockchains.

# Chapter 5

## 5.0 System Architecture Development - Hyperledger Fabric with Soil Moisture Sensors(IoT)

We have developed a multi-tier application interface for agricultural industry users to view their soil moisture, light, air-pressure sensor values on their mobile application as an NFT.

Selecting a suitable application protocol depends on the devices, applications, and environment of the network. (Yassein et al., 2016), before starting project we have analyse hypothesis results and figured that depending on the nature of our project we have to adapt several programming technologies, which includes Arduino code, python, electronJs and on the server side we have used JavaScript for ERC-721 smart contract and nodeJs for server end APIs. Further all details are given below.

## 5.1 User Interface

The concept of web application composition at the presentation layer, i.e., developing web applications from reusable components in user interfaces, is a relatively new research area. (Pietschmann et al., 2010). As per user interface we have designed ElectronJs and android app with essential user inputs and display of sensor outputs.

### 5.1.1 ElectonJs application consists of several screens

#### User Login:

**Graphical user interface, text, application, email

Description automatically generated**

#### User Registration

**Table

Description automatically generated with low confidence**

#### User Password Return Screen:

**Graphical user interface, text, application

Description automatically generated**

#### NFTs result display

Graphical user interface, table

Description automatically generated

#### QR Scanning

**Qr code

Description automatically generated**

### 5.1.2 Android application consists of screens

QR Reading & NFT sensor values display result

|  |  |
| --- | --- |
| Qr code  Description automatically generated | Text  Description automatically generated |

### 5.2 IoT side application architecture

#### 5.2.1 Arduino Sketch architecture

In our scenario we are using Arduino IDE sketch for fetching sensors values from Arduino sensor kit. We are collecting data of several sensors from Arduino sensor kit by implementing [code for Arduino kit](#ArduinoKitCode).

1. Soil Moisture Level
2. Air pressure
3. Temperature
4. Acceleration
5. Light
6. Sound

### 5.2.2 Python Middleware

At second stage, after fetching sensors generated data we are further sending it to serial port after an interval of 60 seconds, and reading it in python code through serial communication on “ttyUSB0” port, at this point we are wrapping sensors information in JSON code format and further passing it to MQTT protocol. Main purpose to engage MQTT protocol is to broadcast message to all the connected devices of the user i.e., Raspberry based ElectronJs application and android application, So we can make communication with devices and engage them that new sensor value is generated and needs to be get display on screens. Implemented code provided in [appendix](#PythonMiddlewareCode).

### 5.2.3 ElectronJs Web Endpoint Application

We have supported raspberry pi user front end screens with electronJs technology environment. At first stage in our application we have initialized MQTT protocol broker for capturing sensor values which middleware is broadcasting and further sending that information to our electronJs application where our end point for API’s is initialized, these APIs are communicating blockchain network. We have developed multiple API calls to support our operations, mentioned below:

1. User login API call in [appendix](#ElectronJsClientsideUserlogin)
2. User registration API call in [appendix](#ElectronJsClientSideUserRegistration)
3. Pushing sensor data to network API call in [appendix](#PushingSensorDatatoNetworkAPIcall)
4. Retrieving User NFTs(sensor data) API call in [appendix](#ElectronJsRetrievingUserNFTs)

## 5.3 Server-Side Network architecture

We have implemented a small scale of network with 8 GB Memory,2 Core Processor, 160 GB SSD Disk with 5 TB Transfer rate, and deployed Hyperledger fabric network on it on AWS(Amazon Web Services) with the endpoint written in NodeJs application.

### 5.3.1 Hyperledger Fabric Network

We have setup the Hyperledger fabric version 2.0 SDK for achieving decentralized storage and implemented 1 channel and 1 organization as per our scenario for robust transactions.

### 5.3.2 NodeJs server-side API’s:

We have deployed NodeJs API’s on server end for communication with Hyperledger fabric. This API is performing several operations.

#### User Registration in network:

This is a POST type API call, which accepts username and pass it to Hyperledger fabric and in response it returns Private SSH key. This is a special unique key which gets generated by Hyperledger fabric upon user registration. In response of this API, we are providing that key to the user along with the authentication bearer token. [Code Attached in appendix](#ServerSideUserRegistration).

#### User login on network:

This is a POST type API call, User have to provide username and Private SSH key, In response API provides user a authentication bearer token again which user can use for fetching NFTs from network. [Code attached in appendix](#ServerSideUserLogin).

#### Mint NFT to blockchain:

This is also a POST type API call request, which accepts sensors information provided by user along with bearer authentication token, and passes it to smart contract for minting, In case of approval from smart contract API return response “values successfully minted.” store information on the distributed ledger which is deployed on Hyperledger fabric, Otherwise if bearer token is faulty or corrupt. It will fail the request.

[Code attached in appendix](#InvokingTransactionOnChaincode).

#### Get Assets By owner:

This is a GET type API call request, in which bearer token is mandatory for the authentication of the user, once it receives request from the user API communicates with the Hyperledger fabric, in response it returns all NFT’s information included “TokenID, Owner Information and TokenURI”. TokenURI is our sensor data generated NFT. Once nodeJs receives this information it passes it as response to the user.

[Code attached in appendix](#Querychaincodeonnetwork).

#### Total Supply:

This is a GET type API call request. For monitoring our network we have developed this API call. It returns us the circulation supply of our network, or in other words total number of NFT’s minted over our network by all users.

[Code attached in appendix](#SmartContractERC721TotalSupply).

## 5.4 Blockchain side ERC-721 Smart Contract

Hyperledger fabric provides its own official ERC-721 smart contract for the minting, which we have used in our network and modified it as per our needs of sensor values.

### 5.4.1 Minting Token Function:

This function gets executed by NodeJs Api Server side. Before minting a sensor value to an NFT into a network it checks several parameters.

1. Authorization: Check the authorization of the user. If client is authorized to mint NFT then its sensor value will get store on distributed ledger.
2. Token\_ID: As per blockchain base architecture every NFT which gets generated on the network have some unique identity(Token\_ID). We handled that process in NodeJs by providing auto increment id to every new sensor value to achieve uniqueness.
3. TokenURI: This is the sensor value in our scenario. It will get stored on distributed ledger against owner.

[Code is attached in appendix](#SmartContractMinting).

### 5.4.2 Client Minted NFT Function:

This function of smart contract communicates with Hyperledger Fabric and checks the authentication of owner, if its valid then it fetches the list of Non-Fungible Tokens which are listed against owner.

[Code is attached in appendix](#SmartContractERC721ClientMinted).

### 5.4.3 Total Supply Function

This function of ERC-721 smart contract calculates the total number of NFTs generated and handled by this contract in the network and returns the total supply of all non-fungible tokens on network.

[Code is attached in appendix](#SmartContractERC721TotalSupply).

## 5.5 Mobile-Side Application Development

We have developed the mobile application in react native platform for users to view the details of sensor generated values from the distributed ledger. To achieve this functionality we have developed a code can scan the QR code, which returns the user security token and on the base of that token we make further call to the REST API on network which returns user its sensors generated values (NFTs).

API connectivity and display of NFT values code:

This code first check the user token value, then further we developed an API call where we are binding the security bearer token with API URL as a security parameter, once we get response from the API we are processing the NFT sensor values on the front of screen.

[Code is attached in appendix](#MobileApplicationCode).

## 5.5 System Architecture Diagram:

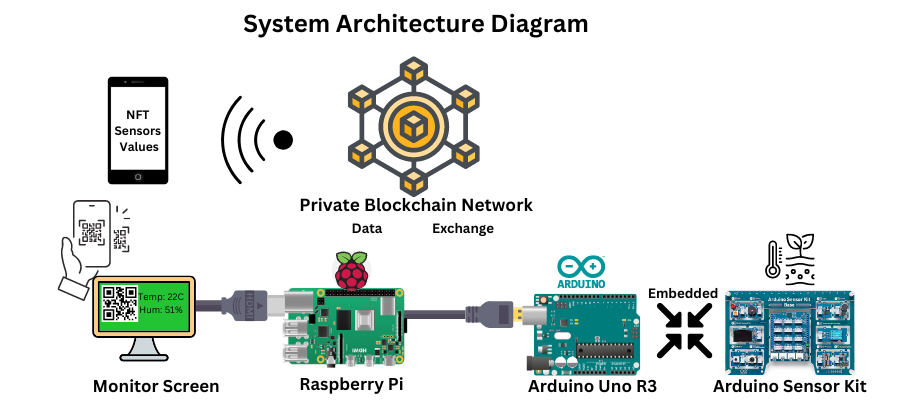


Diagram above explains the architect of our achieved prototype model, in which multiple hardware’s are engaged together to develop an ecosystem for IoT to communicate with private blockchain distributed ledger.

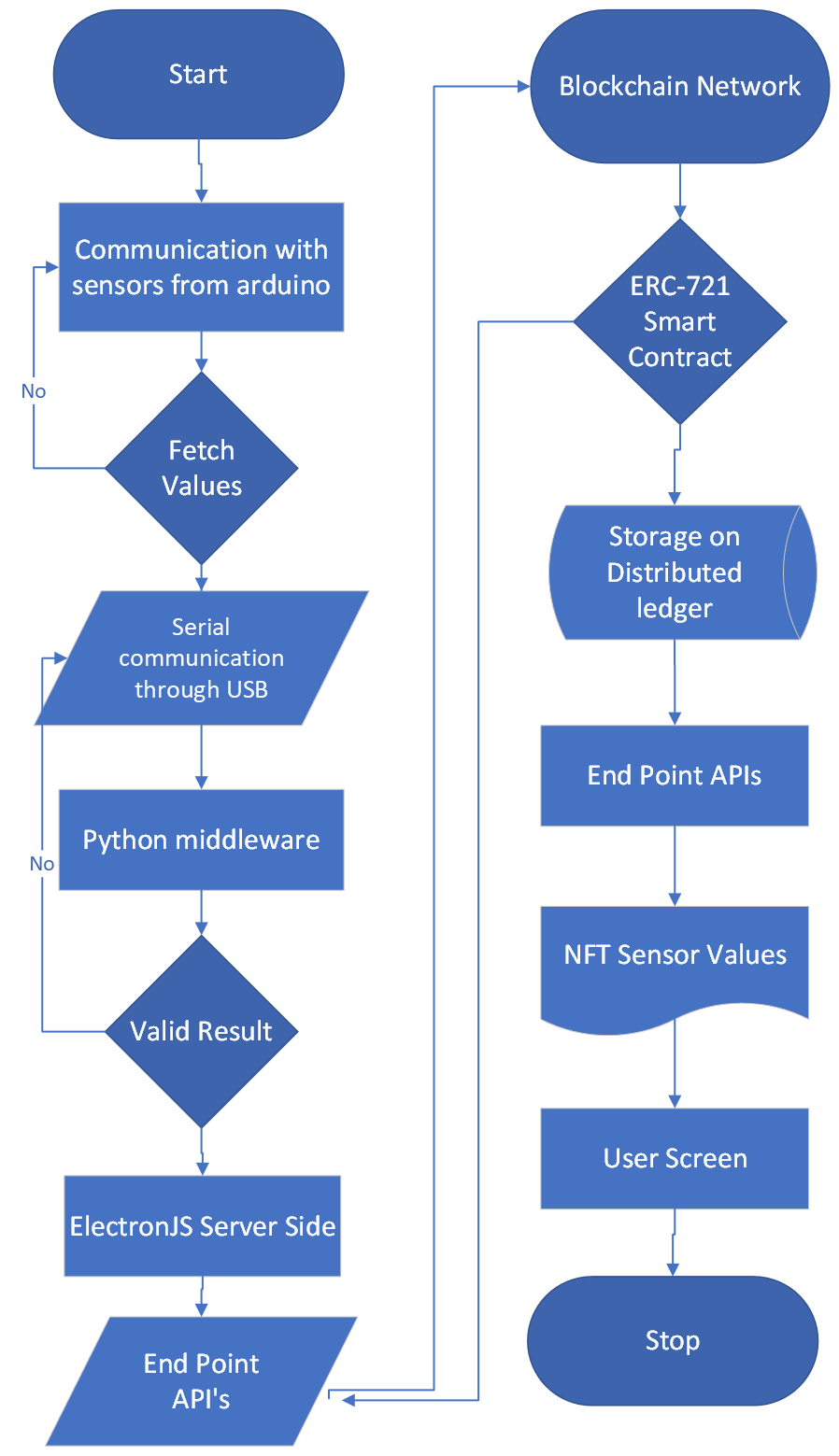
#### 5.1 IoT side Image:

In this image RaspberryPi unit is attached with Arduino Uno R3 with USB serial cable and further on top of it we have embedded Arduino sensor kit



## 5.7 System Information flow chart

This flow chart explains they flow of information of all the processes which we have executed step by step for developing entire system.



## 5.8 Summary of chapter 5

This chapter describes the whole system architecture which we have built by using several technologies i.e., NodeJs, ElectronJs, Python, Arduino IDE, React Native, Network deployment and security perspectives and how the information flows from IoT to blockchain and from blockchain to mobile.

# Chapter 6

## 6.1 System Hardware & Software Requirements

As we have used several environments for development for different devices, If any user wants to use it he/she might need to get several hardware’s i.e., mobile, raspberry pi, Arduino uno board, Arduino sensor kit.

## 6.2 Android Mobile:

5Mega Pixel Camera with Supported android version.

|  |  |
| --- | --- |
| Android Version Name | Version Serial |
| Marshmallow | 6.0 - 6.0.1 |
| Nougat | 7.0 |
| Nougat | 7.1.0 - 7.1.2 |
| Oreo | 8.0 |
| Oreo | 8.1 |
| Pie | 9.0 |
| Android 10 | 10.0 |
| Android 11 | 11 |

## 6.3 Raspberry Pi

Raspberry Pi with raspbian operating system

|  |  |  |
| --- | --- | --- |
| Hardware | Memory | Processor |
| Raspberry Pi 3 A+ | 1GB | 1.4GHz 64-bit quad-core processor |
| Raspberry Pi 3 B+ | 1GB | 1.4GHz 64-bit SoC ,Cortex-A53 (ARMv8) |
| Raspberry Pi 4 B | 2GB, 4GB, 8GB | 1.5GHz 64-bit SoC, Quad core Cortex-A72 (ARM v8) |

## 6.4 Arduino Board:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| hardware | Micro Controller | Clock Speed | EEPROM | SRAM | Flash Memory |
| Arduino Uno Rev3 | ATmega328P | 16 MHz | 1 KB | 2 KB | 32 KB |
| Arduino Uno | ATmega328P | 16 MHz | 1 KB | 2 KB | 32 KB |

## 6.5 Arduino Sensor Kit:

Arduino sensor kit that have integrated all sensors on one board

The light sensor - a photoresistor that reads light intensity.

The sound sensor - a tiny microphone that measures sound vibrations.

The air pressure sensor - reads air pressure, using I2C protocol.

The temperature sensor - reads temperature and humidity at the same time.

The OLED screen - a screen that values or messages can be printed to

From Arduino official merchandise[[1]](#footnote-1)

## 6.6 Summary of chapter 6:

This context in this chapter defines the need of hardware and software which we have used in the development of this project, along with the other feasibilities of other hardware devices.

# Chapter 7

## 7.1 Conclusion

We have developed a prototype of a system and named it as “PlanterAI” which merges IoT with blockchain, both are two cutting-edge technologies that have the potential to revolutionize various industries by enabling secure, decentralized, and automated systems, further development of this multi-tier architecture will convince industry to take revolutionary step towards blockchain to use it as a storage memory unit instead of cloud.

With the help of key findings and hypothesis generation I have solved the flow errors and developed python middleware in order to resolve communication issues between Arduino and Raspberry Pi along with that we have taken security measures too for secure data transmission between device and network through user authentication by implementation of JWT(Json Web Token) and finally, user can access his NFT’s (sensor values) on his mobile screen. We have used multiple programming languages for accomplishing these applications, i.e., Arduino code, Python, ElectronJs on Raspberry Pi side, and NodeJs on server side along with smart contract development in Javascript.

# References

1. Application layer protocols for the Internet of Things: A survey. 2016 International Conference on Engineering & MIS (ICEMIS). <https://ieeexplore.ieee.org/abstract/document/7745303>
2. Application composition at the presentation layer. Proceedings of the 12th International Conference on Information Integration and Web-Based Applications & Services - IiWAS ’10. <https://dl.acm.org/doi/abs/10.1145/1967486.1967558>
3. Design and Implementation of a Reliable Message Transmission System Based on MQTT Protocol in IoT. Wireless Personal Communications, 91(4), 1765–1777. <https://link.springer.com/article/10.1007/s11277-016-3398-2>
4. OPTICAL DATA COMMUNICATION USING PYTHON AND ARDUINO µC [Review of OPTICAL DATA COMMUNICATION USING PYTHON AND ARDUINO µC] <http://ijeast.com/papers/400-405,Tesma411,IJEAST.pdf>
5. Implementation of cloud server for real time data storage using Raspberry Pi. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/7453790>
6. Survey of IoT Key Enabling and Future Technologies: 5G, Mobile IoT, Sematic Web and Applications. Wireless Personal Communications, 97(2), 1645–1675. <https://link.springer.com/article/10.1007/s11277-017-4647-8>
7. Blockchain: A game changer for securing IoT data. 2018 IEEE 4th World Forum on Internet of Things (WF-IoT). <https://ieeexplore.ieee.org/abstract/document/8355182>
8. Design of IoT Blockchain Based Smart Agriculture for Enlightening Safety and Security. Emerging Technologies in Computer Engineering: Microservices in Big Data Analytics, 7–19. <https://link.springer.com/chapter/10.1007/978-981-13-8300-7_2>
9. Calibration of Capacitive Soil Moisture Sensor (SKU:SEN0193). IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/8528624>
10. Cloud Storage as the Infrastructure of Cloud Computing - IEEE Conference Publication.(2010).Ieee.org. <https://ieeexplore.ieee.org/abstract/document/5565955>
11. Blockchain as a Service for IoT. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/7917130>
12. Blockchain A Practical Guide to Developing Business, Law, and Technology Solutions. <https://pdfuni.com/sample/IT/IT1-100/IT055/sample%EF%BC%8DBlockchain%201st%201E%20Joseph%20Bambara.pdf>
13. A secure and efficient authentication and data sharing scheme for Internet of Things based on blockchain <https://www.sciencedirect.com/science/article/pii/S1383762121000862>
14. Public and private blockchain in construction business process and information integration. Automation in Construction, 118, 103276. <https://www.sciencedirect.com/science/article/pii/S0926580520301886>
15. Private-Blockchain-Based Industrial IoT for Material and Product Tracking in Smart Manufacturing. IEEE Network, 34(5), 91–97 <https://ieeexplore.ieee.org/abstract/document/9199799>
16. A Blockchain-Based Framework for Data Sharing With Fine-Grained Access Control in Decentralized Storage Systems. IEEE Access, 6, 38437–38450. <https://ieeexplore.ieee.org/abstract/document/8400511>
17. Embedding Blockchain Technology into IoT for Security: A Survey. IEEE Internet of Things Journal, 1–1.<https://ieeexplore.ieee.org/abstract/document/9358215>
18. The Future of Blockchain Technology in Healthcare Internet of Things Security. Advances in Information Security, 161–184.<https://link.springer.com/chapter/10.1007/978-3-030-38181-3_9>
19. Hyperledger Fabric Blockchain as a Service for the IoT: Proof of Concept. Model and Data Engineering, 172–183 <https://link.springer.com/chapter/10.1007/978-3-030-32065-2_12>
20. A Hyper-ledger Fabric Framework as a Service for Improved Quality E-voting System. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/9230820>
21. Decentralized authorization in constrained IoT environments exploiting interledger mechanisms. Computer Communications, 152, 243–251. <https://www.sciencedirect.com/science/article/pii/S0140366419312174>
22. A token authentication solution for hadoop based on kerberos pre-authentication. IEEE Xplore.<https://ieeexplore.ieee.org/abstract/document/7058096>
23. Building secure healthcare services using OAuth 2.0 and JSON web token in IOT cloud scenario. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/7917942>
24. An analysis on the revoking mechanisms for JSON Web Tokens. International Journal of Distributed Sensor Networks, 14(9), 155014771880153. <https://journals.sagepub.com/doi/full/10.1177/1550147718801535>
25. Comparison of Different Authentication Techniques and Steps to Implement Robust JWT Authentication. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/9835796>
26. A Blockchain-based Shared Mobility Platform for Private and Commercial Vehicles utilizing ERC-721 Tokens. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/9357605>
27. Secure Management of IoT Devices Based on Blockchain Non-fungible Tokens and Physical Unclonable Functions. Lecture Notes in Computer Science, 24–40. <https://link.springer.com/chapter/10.1007/978-3-030-61638-0_2>
28. Hardening X.509 Certificate Issuance using Distributed Ledger Technology. NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. <https://ieeexplore.ieee.org/abstract/document/9110311>
29. Certificate Transparency Using Blockchain. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/8637448>
30. Docker Swarm and Kubernetes in Cloud Computing Environment. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/8862654>
31. Ripping the Fabric: Attacks and Mitigations on Hyperledger Fabric. Applications and Techniques in Information Security, 300–311. <https://link.springer.com/chapter/10.1007/978-981-15-0871-4_24>
32. Performance Characterization of Hyperledger Fabric. 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). <https://ieeexplore.ieee.org/abstract/document/8525394>
33. A Survey on Blockchain for Enterprise Using Hyperledger Fabric and Composer. 2019 6th International Conference on Dependable Systems and Their Applications (DSA). <https://ieeexplore.ieee.org/abstract/document/9045815>
34. A Framework for Creating Deployable Smart Contracts for Non-fungible Tokens on the Ethereum Blockchain. 2020 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS). <https://ieeexplore.ieee.org/abstract/document/9126027>
35. A Survey on the Integration of Blockchain With IoT to Enhance Performance and Eliminate Challenges. IEEE Access, 9, 54478–54497. <https://ieeexplore.ieee.org/abstract/document/9393912>
36. Towards Post-Quantum Blockchain: A Review on Blockchain Cryptography Resistant to Quantum Computing Attacks. IEEE Access, 8, 21091–21116. <https://ieeexplore.ieee.org/abstract/document/8967098>
37. Blockchain — Literature survey. IEEE Xplore <https://ieeexplore.ieee.org/abstract/document/8256979>
38. Performance Characterization of Hyperledger Fabric. 2018 Crypto Valley Conference on Blockchain Technology (CVCBT). <https://ieeexplore.ieee.org/abstract/document/8525394>
39. A Review on Internet of Things (IoT) in Healthcare. <https://www.researchgate.net/profile/Mustafa-Azzawi/publication/309718253_A_Review_on_Internet_of_Things_IoT_in_Healthcare/links/581e46e008aea429b295c6eb/A-Review-on-Internet-of-Things-IoT-in-Healthcare.pdf>
40. Scalability in Internet of Things: Features, Techniques and Research Challenges. International Journal of Computational Intelligence Research, 13(7), 1617–1627. <http://www.ripublication.com/ijcir17/ijcirv13n7_06.pdf>
41. An Overview: Security Issue in IoT Network. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/8653728>
42. Sharding Blockchain. 2018 IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). <https://ieeexplore.ieee.org/abstract/document/8726639>
43. A load balancing scheme based on deep-learning in IoT. Cluster Computing, 20(1), 873–878. <https://link.springer.com/article/10.1007/s10586-016-0667-5>
44. A privacy-preserving blockchain framework for healthcare data leveraging an on-chain and off-chain system design. (2021). Information Processing & Management, 58(3), 102535. <https://www.sciencedirect.com/science/article/pii/S0306457321000431>
45. Developing the Research Hypothesis. Journal of Investigative Surgery, 24(5), 191–194. <https://www.tandfonline.com/doi/full/10.3109/08941939.2011.609449>
46. The 8 th Annual Basic Science International Conference Convergence of Basic Sciences, Toward the World’s Sustainability Challenges. <https://basic.ub.ac.id/files/proceeding/BASIC%202018%20Proceedings%20Book_Revised5.pdf>
47. Singh, M., Singh, A., & Kim, S. (2018). Blockchain: A game changer for securing IoT data. 2018 IEEE 4th World Forum on Internet of Things (WF-IoT). <https://ieeexplore.ieee.org/document/8355182>
48. Comparison of JWT and OAuth 2.0 authorisation and authentication techniques in REST services. <https://core.ac.uk/download/pdf/159753962.pdf>
49. Bitcoin: A Peer-to-Peer Electronic Cash System. <https://www.debr.io/article/21260.pdf>
50. Model for Car Registration System based on Blockchain Technology. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/9668245>
51. IoT based data processing for automated industrial meter reader using Raspberry Pi. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/7562704>
52. Attack scenarios and security analysis of MQTT communication protocol in IoT system. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/8239179>
53. IoT architecture to enable intercommunication through REST API and UPnP using IP, ZigBee and arduino. IEEE Xplore. <https://ieeexplore.ieee.org/abstract/document/6673340>
54. An IoT Based Real-Time Environmental Monitoring System Using Arduino and Cloud Service. Engineering, Technology & Applied Science Research, 8(4), 3238–3242. <https://pdfs.semanticscholar.org/ccef/9364d35331adb759fc476cf25f33c842a0df.pdf>
55. Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. Proceedings of 2014 11th International Bhurban Conference on Applied Sciences & Technology (IBCAST) Islamabad, Pakistan, 14th - 18th January, 2014. <https://ieeexplore.ieee.org/abstract/document/6778179>
56. A Survey on Blockchain for Information Systems Management and Security <https://www.sciencedirect.com/science/article/pii/S030645732030892X>
57. Enabling Blockchain Innovations with Pegged Sidechains. <http://kevinriggen.com/files/sidechains.pdf>

# Appendix:

## Appendix A - Research project plan

**TITLE**

Soil Moisture System (IoT) Integration with Blockchain-Hyperledger Fabric

**Introduction and justification:**

Upcoming era is evolving to decentralization. Blockchain is based on decentralized and distributed ledger/Shared ledger network of blocks which are chained together. Currently IoT sensor-based devices are storing their values on cloud database which is not more secure as compared to private blockchain due to its “Immutability, Auditability, Autonomy”. Distributed ledger technology protocol permits for storage of all digitalized information in accurate and secure manner using cryptography. This shift of the industry will tremendously change the trend of information storing all around the globe. This project will be a prototype of “Soil Moisture System” in which soil moisture system will communicate with private blockchain based on distributed ledger for storing information and will retrieve data as an NFT(Non-Fungible Token) on mobile phone application and will discuss the complexities of communication between devices and private blockchain as well as security aspects of data in which I will use special authentication security barriers. This idea of app will further help industry and education sector to enhance smart devices data storage on distributed ledger over permissioned blockchain network.

**Research question, Aims, Objectives and Deliverable**.

The aim of this project is to create an application for IoT industry which is even more beneficial with blockchain. This prototype of application will transmit data on blockchain from hardware “Arduino board, Raspberry Pi, Sensor Kit” with security. This may help the IoT industry for storing their sensor values on blockchain and future researchers and developers will be able to use this prototype for education purpose. This app will collect and store information on distributed ledger as an NFT and retrieval of information as an NFT in mobile phone application, so user can view their sensors reading.

**Research question**

How to get soil moisture sensors values and securely transmit values by using security authentication tokens to distributed ledger over private blockchain as well as retrieval of digital assets “NFT – Sensor values” in mobile application?

How only authentic users can access their own sensors values in mobile application as an NFT?

**Objectives**

Conduct research on soil moisture systems Linux based.

Development of Linux based application in which system will collect sensors values from sensors after a specific time interval and will send to distributed ledger with right ownership.

Setup of distributed ledger by using Hyperledger Fabric over AWS or Azure which will include dockers setting, node.js setup, network permission settings which will include smart contracts.

Secure API’s building for communication between raspberry pi, distributed ledger and mobile application.

Mobile phone application development for display of NFTs from distributed ledger. Only authorize user/owner can view NFTs. This will get accomplish through QR scanning in which QR code will get display on arduino board LED, after scanning from mobile camera user will be able to view NFTs.

**Deliveries:**

**1) Development of 2 applications**

1. Linux base application for collection of soil moisture several sensors values mentioned below and securely transmission of data to distributed ledger.

* Soil Moisture Sensor.
* Temperature & Humidity Sensor.
* Air Pressure Sensor.

1. Mobile application for display of NFT’s (Sensors values) only to verified owner after QR code scanning from Arduino LED.

**2) Server side operations:**

1. Installation of Hyperledger Fabric(distributed ledger) on AWS or Azure.
2. Smart contract development.
3. API’s development for communication with hardware and mobile application.

**Literature Review**

**Why selection of blockchain as a storage medium?**

**Decentralized:** There is not a single point of failure or centralised component. In other words, it has a peer-to-peer (P2P) design as its foundation.

*“Blockchain is the mechanism that allows transactions to be verified by a group of unreliable actors. It provides a distributed, immutable, transparent, secure and auditable ledger.” (Reyna et al., 2018)*

**Fault-tolerance:** With the help of consensus protocol and each peer or node verification on the permissioned private blockchain network. It is specifically designed to tolerate malicious behaviour of the users until or unless they got majority votes they cant perform malicious activity within a network.

**Immutability:** Blockchain main idea is depending upon immutable blocks which are linked to each other like a link list. Every next block knows about the information of their neighbouring blocks, because of this architecture it becomes tamper resistant. If the information in any block get tampered all other nodes acts and don’t allow block to change its state or information. For attacker its really challenging because he needs to compromise majority of the nodes within a network which is not quite possible.

Diagram

Description automatically generated

(Iftekhar et al., 2021)

“*Blockchain immutability and auditability is necessary for IoT infrastructures to keep user data and provides transparency.” (Tseng et al., 2020)*

**Auditability:** All participants have access to all transaction records since they each possess a copy of the Blockchain (also known as a local chain). Participants can look up any desired block and get the information which is inside because of the transparency of the records.

“I*t is important to understand the consistency model achieved by Blockchain if we want to use it as a database, since consistency specifies how the system orders the blocks, and which version of data may be legally observed by the participants.*” *(Tseng et al., 2020)*

**Autonomy:** Consensus-based protocol autonomy permission all the nodes in the blockchain network to securely update or send data.

**Why not centralized cloud database?**

Centralization is the process of providing authority of the system in the hands of top management who can temper or update record from the system, whereas decentralization, on the other hand, refers to the delegation of power and authority from the top to functional-level management.

Centralized organisations must wait for decisions to be approved, whereas decentralised organisations are fully independent and can make decisions quickly.

**General findings of the review**

“A major function of IoT is to collect data to produce better decision making or controlling (systems)” *(Tseng et al., 2020)*

Instead of storing data from IoT devices on a centralised cloud where anyone with administrative rights could change or alter records, we can store the data on a private blockchain distributed ledger to achieve transparency and security.

“*Private blockchain has been successfully applied in industrial Internet of Things (IoT) because of its advantages and industrial characteristics*” (Jo et al., 2020)

*“Private blockchain is suitable for industrial IoT because most industrial IoT organizations are exclusive and private (less public) and focus on money, assets, time. The blockchain-IoT combination is powerful and can cause significant transformations across several industries*”(Christidis & Devetsikiotis, 2016)

**Availability of sources:**

The well-known examples of private blockchains that enable consensus and membership services for organisations are Hyperledger Fabric and R3 Corda.

We need Raspberry Pi, Arduino UNO R3 for reading data values from sensors and passing values to network via web API calls to connect IoT devices with distributed ledger.

Data communication and transfer of data from IoT device end points to distributed ledger is bit complex in terms of security and access control over data ownership, as well as populating data from distributed ledger to DApp (Decentralized Application).

“*The other benefit of Blockchain-based database is to provide an easy-to-manage and accessible mechanism to share data among multiple clients due to immutable data and decentralization. However, “Blockchain as a database” is a young field filled with many challenges, especially in IoT environment.*” *(Tseng et al., 2020)*

**IOT ADOPTION:**

Blockchain technology is now getting into industry, few of the systems which are using private permission blockchain are mentioned below:

1. Intrusion Detection Systems (IDSs)
2. Crowd-sensing Applications
3. Internet of vehicles (IoV)
4. IoT in the 5G
5. Healthcare Applications
6. Cloud and Fog computing

As we've seen, the combination of IoT and blockchain is extremely powerful. In a trusted environment, blockchain provides a resilient distributed peer-to-peer mechanism. Smart contracts, on the other hand, allow peers to automate advanced multi-step methodologies. The IoT system's devices are the points of interaction with the physical world, allowing for the automation of long workflows in a distinct way, enabling encryption verifiability, and saving significant process costs and time.

**RESEARCH DESIGN**

In this section, we will discuss methodology of implementation as well as brief description on the software analysis and discussion of appropriate hardware for IoT application along with security and reliability concepts.

**Project Management Methodology:**

As this application idea is innovative in terms of using soil moisture sensors values and transmission of data on private blockchain distributed ledger so there might be some changes required while implementation of the project after research, as it will be based on 2 different hardware application and server side management too. For such scenarios we need to stick with “Agile Methodology” for project management as it is an iterative approach and will be more beneficial in future as for modification and creation of an app with IoT idea.

**Reason for selection of Agile Methodology :**

As this approach is used for rapid application development and there might be architectural errors during testing phase so might need to apply different technique frequently to accomplish goal within time.

This is a big project with 4 main sides(Mobile App, Server Side APIs, Raspberry App, Hyperledger Fabric integrations) so might need to handle raspberry Linux-based application and server side API development all together. Scenario: If there will be any error in API’s while development of mobile application, will be needed to fix on first priority because of dependency, so agile is suitable for this project.

**Hardware Requirements:**

* Raspberry Pi
* Arduino Uno
* Soil Moisture sensor board
* O-LED
* Moisture sensors
* Android Operating system phone

**Server Requirements and Software Installations:**

* AWS or Azure
* Hyperledger Fabric Integration on server
* Smart contracts
* Node.js API’s
* URI allocation for API’s
* Google play store

**Hyperledger Fabric Selection for distributed ledger:**

It is an blockchain open source framework which is hosted by Linux foundation. It is also know as permissioned framework where all participants are authenticated and known.

There are also several frameworks like Ethereum, Ripple, Corda but so far Hyperledger Fabric outruns because of its performance in terms of execution time and latency rate.

“*The performance analyses of both Hyperledger Fabric and Ethereum were presented. Findings indicate that the performance of Hyperledger Fabric outperform that of Ethereum in terms of latency, throughput and execution time*.” (Kuzlu et al., 2019)

**API’s Requirement Analysis:**

1. This is very important part of project which will act like a bridge between hyperledger fabric distributed ledger and sensor application and mobile application.
2. Node-API toolkit is preferable as it act as intermediary between backend code and node java script engine. Node-API is built into node version 8.0.
3. Security – “API’s needs to get secure by barrier tokens or authentication tokens for avoiding hackers to getting into network”. This is also a solution of our research question discussed in section above.

Major functionality of API includes:

* User Registration.
* Send data through soil moisture system.
* Receive NFT’s(digital assets) on mobile application.

**Raspberry Pi Linux based Sensor App Requirement Analysis**

1. This app will get values of Soil Moisture Sensor, Temperature & Humidity Sensor, Air Pressure Sensor and will send them to private distributed ledger through APIs through web call.
2. Soil moisture sensors usually communicate in analogue signal medium, it needs to be converted into digital reading and threshold also needs to calculate with formulation before sending sensor value to distributed ledger.
3. O-LED on sensor board will display a QR code for security purpose, will discuss more briefly in next section

**Android mobile app requirement Analysis:**

1. This mobile app will show the values of all sensors on screen with specific time stamp
2. Verified owner will only be able to access those values after scanning the QR code which will be displayed on Sensor board O-LED.

**ETHICS, RISKS AND ISSUES**

**Ethical Issues:**

Up till now there are no ethical problems in the lightening of the ethics checklist. As research is not involving any human and this research and project is using open source resources.

As per hyperledger fabric framework we have to write a chaincode for our network and it is mandatory for every peer or node to execute chaincode for successful transaction.

**Endorsement policy of Hyperledger fabric:**

“*Every chaincode has an endorsement policy which specifies the set of peers on a channel that must execute chaincode and endorse the execution results in order for the transaction to be considered valid. These endorsement policies define the organizations (through their peers) who must “endorse” (i.e., approve of) the execution of a proposal*.”*(HyperLedger-Fabric Official, 2022)*

**Risk Assessment:**

**Distributed-Ledger Potential Risk Assessment:**

“*the risk does exist, especially in blockchains with small networks*” (Swan 2015).

As blockchain network takes decision after approval from majority of nodes, In other words if 51% of the network peers get malicious they could change or tamper record, but this never happened since 2009, but if hacker gets 51% of the entire network he can cause harm to whole network.

“*Although no 51% attacks have occurred in the bitcoin network since January 2009, when the first genesis block was created and added to the blockchain*” (Crosby et al. 2016; Sundararajan 2016)

**Technical risks assessment:**

This project also involves high technical risks, because Linux based raspberry application will gather the sensor data and will send to distributed ledger through web API’s call. If sensors will stop working application could send zero values to API’s after a specific time interval. Once the wrong data is submitted it is difficult to remove that record or NFT from ledger. To resolve that issue it needs to be monitor that if sensors working perfectly.

For getting NFT’s or values of sensors on mobile application for the first time users needs to be scan LED attached on the board for authentication, If LED(Hardware) will be not of good pixel quality it will break the QR image, so it couldn’t be scannable.

**Security Risk Assessment:**

Security is the first priority while developing this system. API’s which will get written in “Node” technology needs to be more secure. So the hacker couldn’t trace the encrypted data. To overcome this risk it is mandatory to use barrier token or authentication token approach.

**Time Plan**

**Timeline

Description automatically generated**

**REFERENCES**

1) On blockchain and its integration with IoT. Challenges and opportunities. Future Generation Computer Systems, 88, 173–190.

https://doi.org/10.1016/j.future.2018.05.046

2) Hyperledger Fabric Access Control System for Internet of Things Layer in Blockchain-Based Applications. Entropy, 23(8), p.1054. doi:10.3390/e23081054.

3) Tseng, L., Yao, X., Otoum, S., Aloqaily, M., & Jararweh, Y. (2020). Blockchain-based database in an IoT environment: challenges, opportunities, and analysis. Cluster Computing. https://doi.org/10.1007/s10586-020-03138-7

4)Jo, M., Hu, K., Yu, R., Sun, L., Conti, M., & Du, Q. (2020). Private Blockchain in Industrial IoT. IEEE Network, 34(5), 76–77. https://doi.org/10.1109/mnet.2020.9199796

5) Christidis, K., & Devetsikiotis, M. (2016). Blockchains and Smart Contracts for the Internet of Things. IEEE Access, 4(4), 2292–2303. https://doi.org/10.1109/access.2016.2566339

6) https://hyperledger-fabric.readthedocs.io/en/release-2.2/endorsement-policies.html

7) Crosby M, Nachiappan Pattanayak P, Verma S, Kalyanaraman V (2016) Blockchain technology: Beyond bitcoin. Appl Innov Rev 2:6–19

https://j2-capital.com/wp-content/uploads/2017/11/AIR-2016-Blockchain.pdf

8) Swan M (2015) Blockchain: Blueprint for a New Economy. In Google Books. “O’Reilly Media, Inc.”

## Appendix D: ElectronJs client side user login API call code:

request({

body: payload,

followAllRedirects: true,

headers: {

'Content-Type': 'application/json',

},

method: 'POST',

url: 'http://planterai.com:4000/users/login'}, callback);

function callback(error, response, body) {

if (!error && response.statusCode == 200) {

console.log("body:"+body);

const obj = JSON.parse(body);

var succ = obj.success;

var mess = obj.message;

token = obj.token;

tokenSend = token;

if(succ == false) {

win.webContents.send('recievedPassword', body);

console.log(succ);

console.log(mess);

} else {

createReadingWindow();

win.webContents.send('sendQRCode', token);

client.publish('token', token);

update\_listener.subscribe(username+'/update', function (err) {

if (!err) {

console.log("Subscribed for update");

}else{

console.log("Update subscribe error");

}

})

}

} else {

console.log("Error: \n"+body);

win.webContents.send('recievedPassword', body);

}

};

// const randomPassword = data;

}

})

## Appendix E: ElectronJs(Client Side)User Registration API call code:

ipcMain.on('registerUserRegister', (event, data) => {

const myArr = data.split("+++++");

var username = myArr[0];

var email = myArr[1];

var firstname = myArr[2];

var lastname = myArr[3];

var payload = JSON.stringify({ "username": username, "email": email, "first\_name": firstname, "lastname": lastname });

// const randomPassword = data;

// win.webContents.send('recieveRegisterUserData', payload);

request({

body: payload,

followAllRedirects: true,

headers: {

'Content-Type': 'application/json',

},

method: 'POST',

url: 'http://planterai.com:4000/register'

}, callback);

function callback(error, response, body) {

if (!error && response.statusCode == 200) {

console.log(body);

const obj = JSON.parse(body);

var succ = obj.success;

var mess = obj.message;

var secret = obj.secret

var token = obj.token;

// var res = succ + "=====" + mess + "=====" + secret + "=====" + token;

var res = "Save Your Password: " + secret;

if (succ == false) {

win.webContents.send('recieveRegisterUserData', res);

console.log(res);

} else {

win.webContents.send('recieveRegisterUserData', res);

console.log(res);

}

} else {

console.log("Error: \n" + error);

win.webContents.send('recievedPassword', body);

}

}

// const randomPassword = data;

// win.webContents.send('recieveRegisterUserData', payload);

})

## Appendix F: Pushing sensor data to network API call code:

client.on('message', function (topic, message) {

var data = message.toString();

console.log(data);

if (token != "none") {

data\_obj = JSON.parse(data);

data\_obj["time"] = new Date().toISOString();

var payload = JSON.stringify(

{

"fcn": "MintWithTokenURI",

"args": [JSON.stringify(data\_obj)]

}

);

request({

body: payload,

followAllRedirects: true,

headers: {

'Content-Type': 'application/json',

'Authorization': 'Bearer ' + token,

},

method: 'POST',

url: 'http://planterai.com:4000/channels/mychannel/chaincodes/token\_erc721'

}, push\_cb);

function push\_cb(error, response, body) {

console.log(body);

update\_listener.publish(username + '/update', 'true');

};

}

})

## Appendix G: ElectronJs Retrieving User NFTs(sensor data) API call code:

function update\_readings() {

request({

followAllRedirects: true,

headers: {

'Content-Type': 'application/json',

'Authorization': 'Bearer ' + token,

},

method: 'GET',

url: "http://planterai.com:4000/channels/mychannel/chaincodes/token\_erc721?fcn=ClientMintedNFTs&args='[]'"

}, fetch\_cb);

function fetch\_cb(error, response, body) {

try {

var data = JSON.parse(body);

var results = data.result;

new\_html = "";

count = 1;

for (let i in results) {

uri = results[i].Record.tokenURI;

var uriObj = JSON.parse(uri);

console.log(JSON.stringify(uriObj, null, 2));

new\_html += `<tr>

<th scope="row">${count}</th>

<td>

Moisture:${uriObj["moisture"]}%

<br />

Temperature:${uriObj["temperature"]}

<br />

Humidity:${uriObj["humidity"]}

</td>

<td>

Pressure:${uriObj["pressure"]}

<br />

Temp:${uriObj["temp"]}C

<br />

acceleration:${uriObj["acceleration"]}m

</td>

<td>Light Level:${uriObj["light"]}</td>

<td>Sound Value:${uriObj["sound"]}</td>

<td>${uriObj["time"]}</td>

</tr>`;

count = count + 1;

}

//console.log(new\_html);

win.webContents.send('readingsTableData', new\_html);

} catch (err) {

console.log(err);

}

}

}

## Appendix H: Python Middleware Code:

from urllib.request import urlopen

import paho.mqtt.client as mqtt

import requests

import \_thread

import serial

import json

import time

import os

broker\_address="localhost"

port=1883

delay = 1

ser = serial.Serial ("/dev/ttyACM0", 9600)

updating = False

def wait\_for\_internet(): #ping google until it is succesful (else keep waiting)

print("Waiting for internet...")

while True:

try:

response = urlopen('http://google.com',timeout=1)

print("Internet Available!")

return

except:

pass

time.sleep(1)

def on\_connect(client, userdata, flags, rc):

print("Connected with rc=",rc)

client.subscribe("token")

def on\_message(client, userdata, msg):

try:

global updating

topic = msg.topic

update = str(msg.payload.decode("utf-8"))

token\_file = open("token.txt","w")

token\_file.write(update)

token\_file.close()

print("token:",update)

except:

print("Error")

def send\_ping(mqtt\_client):

while True:

mqtt\_client.publish("ping","online")

time.sleep(1)

client = mqtt.Client("planter\_pyt")

client.on\_message=on\_message

client.on\_connect=on\_connect

client.connect(broker\_address,port)

client.loop\_start()

\_thread.start\_new\_thread(send\_ping, (client, ))

while True:

try:

received\_data = ser.readline()

data = received\_data.decode('utf-8')

try:

json.loads(data)

client.publish("data",data)

except ValueError as e:

print("garbage data")

print("REC:",data)

#time.sleep(delay)

except:

print("Error")

time.sleep(1)

## Appendix I: Arduino Kit Code:

#include <ArduinoJson.h>

#include <Arduino\_SensorKit.h>

#include "DHT.h"

#define DHTTYPE DHT20

DHT dht(DHTTYPE);

StaticJsonDocument<250> doc;

char data\_out[250];

int time\_delay = 60;

int soil\_sensor = A1;

int sound\_sensor = A2;

int light\_sensor = A3;

const int air\_value = 700;

const int water\_value = 450;

int raw\_moisture,moisturepercent;

float temp\_hum\_val[2] = {0};

void setup() {

// initialize serial communication at 9600 bits per second:

Serial.begin(9600);

Pressure.begin();

dht.begin();

}

// the loop routine runs over and over again forever:

void loop() {

raw\_moisture = analogRead(soil\_sensor);

moisturepercent = map(raw\_moisture, air\_value, water\_value, 0, 100);

if(moisturepercent<=0)

{

doc["moisture"] = 0;

}else if(moisturepercent>=100)

{

doc["moisture"] = 100;

}else{

doc["moisture"] = moisturepercent;

}

if(!dht.readTempAndHumidity(temp\_hum\_val)){

doc["temperature"] = temp\_hum\_val[1];

doc["humidity"] = temp\_hum\_val[0];

}else{

doc["temperature"] = 0;

doc["humidity"] = 0;

}

doc["pressure"] = Pressure.readPressure();

doc["temp"] = Pressure.readTemperature();

doc["acceleration"] = Pressure.readAltitude();

int raw\_light = analogRead(light\_sensor);

int light = map(raw\_light, 0, 1023, 0, 100);

doc["light"] = light;

int sound = 0; //create variable to store many different readings

for (int i = 0; i < 32; i++) //create a for loop to read

{ sound += analogRead(sound\_sensor); } //read the sound sensor

sound >>= 5; //bitshift operation

doc["sound"] = sound;

serializeJson(doc, data\_out);

Serial.println(data\_out);

for(int i=0; i<time\_delay; i++){

delay(1000); // delay in between reads for stability

}

}

**NodeJs Server Side API Code:**

## Appendix J: User Registration:

app.post('/register', async function (req, res) {

// Get user input

const { first\_name, last\_name, username, email } = req.body;

logger.debug('End point : /register');

logger.debug('End point : /register');

logger.debug('first\_name : ' + first\_name);

logger.debug('last\_name : ' + last\_name);

logger.debug('username : ' + username);

logger.debug('email : ' + email);

if (!username) {

res.json(getErrorMessage('\'username\''));

return;

} else if (!email) {

res.json(getErrorMessage('\'email\''));

return;

}

// check if user already exist

// Validate if user exist in our database

const oldUser = await User.findOne({ username }).exec();

const oldEmail = await User.findOne({ email }).exec();

if (oldUser || oldEmail) {

return res.json(getErrorMessage('\'User already exit\''));

}

## Appendix K: User Login:

app.post('/users/login', async function (req, res) {

var username = req.body.username;

var password = req.body.password;

logger.debug('End point : /users');

logger.debug('User name : ' + username);

if (!username) {

res.json(getErrorMessage('\'username\''));

return;

}

if (!password) {

res.json(getErrorMessage('\'password\''));

return;

}

// var encryptedPassword = await bcrypt.hash(password, 10);

// const oldUser = await User.find({ 'username': username, 'password': encryptedPassword }).exec();

// if(!oldUser) {

// res.json({ success: false, message: `User not found.` });

// return;

// }

// Validate if user exist in our database

const user = await User.findOne({ username });

if (user && (await bcrypt.compare(password, user.password))) {

// Create token

// const token = jwt.sign(

// {user\_id: user.\_id, email},

// process.env.TOKEN\_KEY,

// {

// expiresIn: "2h",

// }

// );

var token = jwt.sign({

exp: Math.floor(Date.now() / 1000) + parseInt(constants.jwt\_expiretime),

username: username,

password: password

}, app.get('secret'));

res.json({success: true, message: 'User login successfully', token: token});

return;

}

res.json({ success: false, message: `User with username ${username} is not registered with ${password}, Please register first.` });

res.json({ success: false, message: `User with username ${username} is not registered with ${password}, Please register first.` });

return;

});

## Appendix L: Invoking Transaction On Chain code:

app.post('/channels/:channelName/chaincodes/:chaincodeName', async function (req, res) {

try {

logger.debug('==================== INVOKE ON CHAINCODE ==================');

var peers = req.body.peers;

var chaincodeName = req.params.chaincodeName;

var channelName = req.params.channelName;

var fcn = req.body.fcn;

var args = req.body.args;

var transient = req.body.transient;

console.log(`Transient data is ;${transient}`)

logger.debug('channelName : ' + channelName);

logger.debug('chaincodeName : ' + chaincodeName);

logger.debug('fcn : ' + fcn);

logger.debug('args : ' + args);

if (!chaincodeName) {

res.json(getErrorMessage('\'chaincodeName\''));

return;

}

if (!channelName) {

res.json(getErrorMessage('\'channelName\''));

return;

}

if (!fcn) {

res.json(getErrorMessage('\'fcn\''));

return;

}

if (!args) {

res.json(getErrorMessage('\'args\''));

return;

}

let message = await invoke.invokeTransaction(channelName, chaincodeName, fcn, args, req.username, 'org1', transient);

const response\_payload = {

result: message,

error: null,

errorData: null

}

res.send(response\_payload);

} catch (error) {

const response\_payload = {

result: null,

error: error.name,

errorData: error.message

}

res.send(response\_payload)

}

});

## Appendix M: Query Transaction On Chaincode:

app.get('/channels/:channelName/chaincodes/:chaincodeName', async function (req, res) {

try {

logger.debug('==================== QUERY BY CHAINCODE ==================');

var channelName = req.params.channelName;

var chaincodeName = req.params.chaincodeName;

console.log(`chaincode name is :${chaincodeName}`)

let args = req.query.args;

let fcn = req.query.fcn;

let peer = req.query.peer;

logger.debug('channelName : ' + channelName);

logger.debug('chaincodeName : ' + chaincodeName);

logger.debug('fcn : ' + fcn);

logger.debug('args : ' + args);

if (!chaincodeName) {

res.json(getErrorMessage('\'chaincodeName\''));

return;

}

if (!channelName) {

res.json(getErrorMessage('\'channelName\''));

return;

}

if (!fcn) {

res.json(getErrorMessage('\'fcn\''));

return;

}

if (!args) {

res.json(getErrorMessage('\'args\''));

return;

}

// console.log('args=======d===', args);

// args = args.replace(/'/g, '"');

// args = JSON.parse(args);

// logger.debug(args);

// console.log('args======a====', args);

let message = await query.query(channelName, chaincodeName, args, fcn, req.username, "org1");

const response\_payload = {

result: message,

error: null,

errorData: null

}

res.send(response\_payload);

} catch (error) {

const response\_payload = {

result: null,

error: error.name,

errorData: error.message

}

res.send(response\_payload)

}

});

## Appendix N: Server side User Authentication with JWT token:

app.use((req, res, next) => {

logger.debug('New req for %s', req.originalUrl);

if (req.originalUrl.indexOf('/users') >= 0 || req.originalUrl.indexOf('/users/login') >= 0 || req.originalUrl.indexOf('/register') >= 0 || req.originalUrl.indexOf('/test') >= 0) {

if (req.originalUrl.indexOf('/users') >= 0 || req.originalUrl.indexOf('/users/login') >= 0 || req.originalUrl.indexOf('/register') >= 0 || req.originalUrl.indexOf('/test') >= 0) {

return next();

}

var token = req.token;

jwt.verify(token, app.get('secret'), (err, decoded) => {

if (err) {

console.log(`Error ================:${err}`)

res.send({

success: false,

message: 'Failed to authenticate token. Make sure to include the ' +

'token returned from /users call in the authorization header ' +

' as a Bearer token'

});

return;

} else {

req.username = decoded.username;

req.orgname = decoded.orgName;

logger.debug(util.format('Decoded from JWT token: username - %s, password - %s', decoded.username, decoded.orgname));

return next();

}

});

});

## Appendix O: Mobile Application Code:

import React, { useEffect, useState } from 'react';

import {

StyleSheet,

View,

Text,

ScrollView,

Image,

TouchableOpacity,

ActivityIndicator,

ToastAndroid

} from 'react-native';

import axios from 'axios';

import { useNavigation, useRoute } from '@react-navigation/native';

import moment from 'moment';

import { dummyArray } from './res';

import { Entypo } from '@expo/vector-icons';

const NftValue = () => {

const route = useRoute()

const authToken = route?.params?.state?.token;

const navigation = useNavigation();

const [data, setData] = useState([])

const [loader, setLoader] = useState(false);

const [auth, setAuth] = useState(true)

const handelApiCall = async () => {

try {

setLoader(true);

const response = await axios.get("http://54.234.23.69:4000/channels/mychannel/chaincodes/token\_erc721?fcn=ClientMintedNFTs&args='[]'", {

headers: {

"Authorization": `Bearer ${authToken}`

}

})

if (response?.data && response.status === 200) {

let filteredArray = response?.data.result.map((item) => {

return JSON.parse(item?.Record?.tokenURI?.replace(/\\/g, ""))

})

setData(filteredArray)

setLoader(false);

}

} catch (error) {

setAuth(false)

setLoader(false);

console.log(error)

}

}

useEffect(() => {

handelApiCall();

}, [authToken])

return (

<>

{auth ? (<>

{loader ? (

<ActivityIndicator

size={'large'}

color={'black'}

style={{ marginTop: '100%' }}

/>

) : (

<ScrollView style={styles.container}

stickyHeaderIndices={[0]}

showsVerticalScrollIndicator={false}

>

<TouchableOpacity style={styles.homeLogoView} onPress={() => { navigation.navigate('Main') }}>

<Entypo name="home" size={30} style={styles.homeLogo} />

</TouchableOpacity>

<View style={styles.innercontainer}>

<Text style={styles.nftHeadingText} >Your NFT's</Text>

{data?.map((item, index) => (

<View key={index}>

<View style={styles.headingView}>

<Text style={styles.nftHeadingDateText}>{moment(item?.time).format('YYYY-MM-DD h:mm:ss')}</Text>

</View>

<View style={{ alignItems: 'center', justifyContent: 'center', alignContent: 'center', marginBottom: 20, }}>

<Text style={styles.textGlobal}>Moisture: {item?.moisture}%</Text>

<Text style={styles.textGlobal}>Temperature: {Math.trunc(item?.temperature)}C</Text>

<Text style={styles.textGlobal}>Humidity: {Math.trunc(item?.humidity)}%</Text>

</View>

<View style={{ alignItems: 'center', justifyContent: 'center', alignContent: 'center', marginBottom: 20 }}>

<Text style={styles.textGlobal}>Air pressure: {item?.pressure}</Text>

<Text style={styles.textGlobal}>Acceleration: {item?.acceleration}</Text>

</View>

<View style={{ alignItems: 'center', justifyContent: 'center', alignContent: 'center' }}>

<Text style={styles.textGlobal}>Light Level: {item?.light}</Text>

<Text style={styles.textGlobal}>Sound Value: {item?.sound}</Text>

</View>

<View style={{ borderWidth: 6, borderRadius: 20, width: 300, marginVertical: 20 }}>

</View>

</View>

))}

</View>

</ScrollView>

)}

</>) : (<View style={styles.innercontainernotauthorized}>

<Text style={styles.innercontainernotauthorizedText}>Not authorized</Text>

<TouchableOpacity onPress={() => { navigation.navigate('Main') }} style={styles.goToHomeView}><Text style={styles.goToHomeText}>go to Home</Text></TouchableOpacity>

</View>)}

</>

);

};

export default NftValue;

const styles = StyleSheet.create({

container: {

flex: 1,

backgroundColor: '#fff',

paddingTop: 30,

},

homeLogoView: { position: 'absolute', left: 30, top: 60 },

homeLogo: {},

innercontainer: {

flex: 1,

backgroundColor: '#fff',

alignItems: 'center',

paddingTop: 100,

paddingBottom: 80,

justifyContent: 'space-between',

},

innercontainernotauthorized: {

flex: 1,

backgroundColor: '#fff',

justifyContent: 'center',

alignItems: 'center',

alignContent: 'center'

},

innercontainernotauthorizedText: {

},

goToHomeView: { marginTop: 30 },

goToHomeText: { color: '#8C53F7' },

headingView: {

alignItems: 'center',

marginBottom: 40,

},

nftHeadingText: {

fontWeight: 'bold',

marginBottom: 40,

fontSize: 25

},

nftHeadingDateText: {

fontWeight: '600',

fontSize: 18,

alignContent: 'center',

color: '#8C53F7'

},

textGlobal: {

fontWeight: '600',

fontSize: 18

}

});

## Appendix P: Query chaincode on network

app.get('/qscc/channels/:channelName/chaincodes/:chaincodeName', async function (req, res) {

try {

logger.debug('==================== QUERY BY CHAINCODE ==================');

var channelName = req.params.channelName;

var chaincodeName = req.params.chaincodeName;

console.log(`chaincode name is :${chaincodeName}`)

let args = req.query.args;

let fcn = req.query.fcn;

// let peer = req.query.peer;

logger.debug('channelName : ' + channelName);

logger.debug('chaincodeName : ' + chaincodeName);

logger.debug('fcn : ' + fcn);

logger.debug('args : ' + args);

if (!chaincodeName) {

res.json(getErrorMessage('\'chaincodeName\''));

return;

}

if (!channelName) {

res.json(getErrorMessage('\'channelName\''));

return;

}

if (!fcn) {

res.json(getErrorMessage('\'fcn\''));

return;

}

if (!args) {

res.json(getErrorMessage('\'args\''));

return;

}

console.log('args==========', args);

args = args.replace(/'/g, '"');

args = JSON.parse(args);

logger.debug(args);

let response\_payload = await qscc.qscc(channelName, chaincodeName, args, fcn, req.username, req.password);

// const response\_payload = {

// result: message,

// error: null,

// errorData: null

// }

res.send(response\_payload);

} catch (error) {

const response\_payload = {

result: null,

error: error.name,

errorData: error.message

}

res.send(response\_payload)

}

});

## Appendix Q: Smart contract ERC-721 minting NFT on blockchain

Graphical user interface, text, application, email

Description automatically generated

## Appendix R: Smart contract ERC-721 Client Minted

Graphical user interface, text, application, email

Description automatically generated

## Appendix S: Smart Contract ERC-721 Total supply

Graphical user interface, text, application, email

Description automatically generated

1. <https://store.arduino.cc/products/arduino-sensor-kit-base> [↑](#footnote-ref-1)