

**PROTOTYPE OF BLOCKCHAIN-BASED WEB APPLICATION FOR
ELECTRONIC HEALTH RECORD (EHR) DATA MANAGEMENT
AMONG HOSPITALS**



اونيورسيٲى ٲكنيكل ماليسيا ملاك
CHEK SHIN JING

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

PROTOTYPE OF BLOCKCHAIN-BASED WEB APPLICATION FOR
ELECTRONIC HEALTH RECORD (EHR) DATA MANAGEMENT AMONG
HOSPITALS



CHEK SHIN JING

This report is submitted in partial fulfillment of the requirements for the
Bachelor of [Computer Science (Computer Security)] with Honours.

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TEKNIKAL MALAYSIA MELAKA

2023

DECLARATION

I hereby declare that this project report entitled
**PROTOTYPE OF BLOCKCHAIN-BASED WEB APPLICATION FOR
ELECTRONIC HEALTH RECORD (EHR) DATA MANAGEMENT AMONG
HOSPITALS**

is written by me and is my own effort and that no part has been plagiarized
without citations.

STUDENT : CHEK Date : 25/9/2023
(CHEK SHIN JING)

اونيورسيتي تيكنيكل مليسيا ملاك
UNIVERSITI TEKNIKAL MALAYSIA MELAKA

I hereby declare that I have read this project report and found
this project report is sufficient in term of the scope and quality for the award of
Bachelor of [Computer Science (Computer Security)] with Honours.

SUPERVISOR :  Date : 26/9/2023
(TS. DR. MOHD. FAIRUZ ISKANDAR OTHMAN)

DEDICATION

This project is dedicated to whom that has been part of my life.

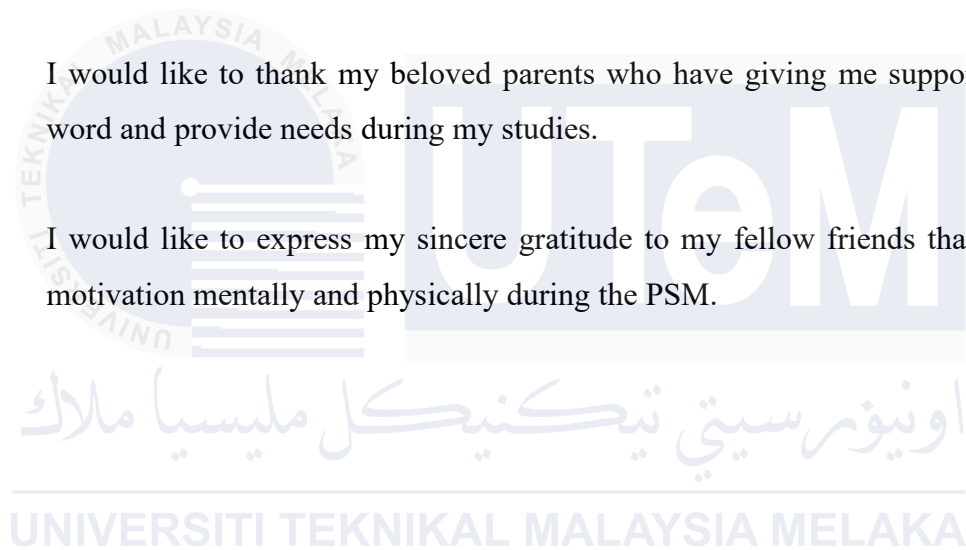


ACKNOWLEDGEMENTS

First of all, I would like to take this opportunity to express my gratitude to my supervisor TS. Dr. Mohd. Fairuz Iskandar Othman of Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka (UTeM) for his assistance, motivation and his patience to help me to complete this project successfully. I would also like to express my appreciation towards Dr. Zaheera Zainal Abidin for her guidance and evaluator GS. Dr. Othman Mohd for his advice.

I would like to thank my beloved parents who have given me support, motivation and provided needs during my studies.

I would like to express my sincere gratitude to my fellow friends that always give me motivation mentally and physically during the PSM.



ABSTRACT

Electronic Health Record (EHR) is considered as one of the most sensitive data in digital world. However, it is often being compromised due to obsolete of data management methods such as traditional data storage using local database, unpatched cloud storage that cannot overcome the latest cybersecurity attacks. Besides, it is also questioned that the current existing EHR data management solutions are mostly not transparent enough and causing interoperability challenges. Hence, the primary challenge addressed in this study is the secure sharing and management of sensitive patient health data among hospitals while ensuring data integrity, access control and auditability. The project utilizes Hyperledger Fabric's blockchain technology to provide a private and tamper-resistant ledger for storing and sharing EHR data. Through a rigorous research process involving system design, smart contract development and network configuration, the prototype establishes a more robust framework for EHR data management. The results demonstrate the potential implementation of a secure and scalable system that enables authorized medical authorities to access and update patient records efficiently, while ensuring data privacy and compliance with healthcare regulations. This project contributes to the advancement of healthcare data management by leveraging blockchain technology to enhance data security, accessibility and transparency among collaborating hospitals.

ABSTRAK

Rekod Kesihatan Elektronik (EHR) dianggap sebagai salah satu data yang paling sensitif dalam dunia digital. Walau bagaimanapun, ia sering terdedah kepada risiko keselamatan disebabkan oleh kaedah pengurusan data yang sudah ketinggalan zaman seperti penyimpanan data tradisional menggunakan pangkalan data tempatan, penyimpanan awan yang tidak dikemaskini yang tidak dapat mengatasi serangan siber terkini. Selain itu, juga dipertikaikan bahawa penyelesaian pengurusan data EHR yang sedia ada pada masa ini kebanyakannya tidak cukup telus dan menyebabkan cabaran interoperabiliti. Oleh itu, cabaran utama yang dibincangkan dalam kajian ini adalah berkongsi dengan selamat dan menguruskan data kesihatan pesakit yang sensitif di antara hospital-hospital sambil memastikan integriti data, kawalan akses dan boleh diperiksa. Projek ini menggunakan teknologi blok rantai Hyperledger Fabric untuk menyediakan lejar peribadi dan tidak boleh diubahsuai untuk penyimpanan dan perkongsian data EHR. Melalui proses penyelidikan yang teliti yang melibatkan reka bentuk sistem, pembangunan kontrak pintar dan konfigurasi rangkaian, prototaip ini membentuk rangka kerja yang lebih kukuh untuk pengurusan data EHR. Hasilnya menunjukkan pelaksanaan potensi sistem yang selamat dan boleh diubahsuai yang membolehkan pihak berkuasa perubatan yang diberi kuasa untuk mengakses dan mengemas kini rekod pesakit dengan cekap, sambil memastikan privasi data dan pematuhan terhadap peraturan kesihatan. Projek ini menyumbang kepada kemajuan pengurusan data kesihatan dengan menggunakan teknologi blok rantai untuk meningkatkan keselamatan data, aksesibiliti dan ketelusan data di antara hospital-hospital yang bekerjasama.

TABLE OF CONTENTS

	PAGE
DEDICATION	III
ACKNOWLEDGEMENTS	IV
ABSTRACT	V
ABSTRAK	VI
TABLE OF CONTENTS	VII
LIST OF TABLES	XII
LIST OF FIGURES	XIII
LIST OF ABBREVIATIONS	XVI
LIST OF ATTACHMENTS	XVII
CHAPTER 1: INTRODUCTION	1
1.1 Project Background	1
1.2 Problem Statement	2
1.3 Project Questions	3
1.4 Project Objective	3
1.5 Project Scope	4
1.6 Project Contribution	4
1.7 Report Organization	5
1.8 Conclusion	6

CHAPTER 2: LITERATURE REVIEW	7
2.1 Introduction	7
2.2 Blockchain Technology	7
2.2.1 Core of Blockchain Technology	8
2.2.1.1 Distributed Ledger Technology	9
2.2.1.2 Cryptography	9
2.2.1.3 Smart Contract	11
2.2.1.4 Consensus Mechanisms	11
2.2.2 Types of Blockchain	15
2.2.2.1 Public Blockchain	15
2.2.2.2 Private Blockchain	15
2.2.2.3 Consortium Blockchain	16
— 2.3 Blockchain Platform	17
2.3.1 Ethereum	17
2.3.2 Hyperledger	18
2.3.2.1 Hyperledger Fabric	19
2.3.2.2 Hyperledger Sawtooth Lake	21
2.3.2.3 Hyperledger Iroha	21
2.3.2.4 Hyperledger Burrow	21
2.3.2.5 Hyperledger Indy	22
2.3.3 MultiChain	22
2.3.4 Open Chain	23
2.4 Critical Review of Existing Works	23

2.5 Proposed Solution.....	30
2.6 Conclusion.....	34
CHAPTER 3: PROJECT METHODOLOGY	35
3.1 Introduction.....	35
3.2 Methodology.....	35
3.2.1 Planning.....	36
3.2.2 Design.....	36
3.2.3 Implementation.....	36
3.2.4 Testing.....	36
3.2.5 Evolution.....	36
3.3 Project Milestones.....	37
3.4 Conclusion.....	39
CHAPTER 4: DESIGN.....	40
4.1 Introduction.....	40
4.2 Problem Analysis.....	40
4.3 Requirement Analysis.....	41
4.3.1 Data Requirement.....	41
4.3.2 Functional Requirement.....	43
4.3.2.1 Smart Contract/Chaincode.....	47
4.3.3 Software Requirement.....	48
4.3.4 Hardware Requirement.....	50
4.4 High-Level Design.....	51
4.4.1 System Architecture.....	51
4.5 Conclusion.....	52

CHAPTER 5: IMPLEMENTATION	53
5.1 Introduction	53
5.2 Software Configuration Management	53
5.3 Prerequisite Base Software	54
5.4 Fabric and Fabric Samples	55
5.5 Contract APIs	55
5.6 Application SDK	58
5.6.1 Hyperledger Fabric Client SDK	58
5.6.2 Wallet	59
5.6.3 JSON Web Tokens	59
5.7 Application	59
5.7.1 State of Distributed Database	59
5.7.2 Identity (CA)	60
5.7.3 Membership Service Provider (MSP)	61
5.7.4 Endorsement Policies	61
5.7.5 Security Mechanism	63
5.8 Conclusion	64
CHAPTER 6: TESTING	65
6.1 Introduction	65
6.2 Test Strategy	65
6.2.1 Classes of Tests	65
6.3 Test Design	66
6.3.1 Test Description	66
6.3.2 Test Data	67

6.4 Test Results and Analysis	70
6.4.1 Network Testing	70
6.4.2 Functionality Testing	74
6.4.2.1 Admin	74
6.4.2.2 Patient	76
6.4.2.3 Doctor	80
6.5 Conclusion	84
CHAPTER 7: PROJECT CONCLUSION	85
7.1 Introduction	85
7.2 Project Summarization	85
7.3 Project Contribution	87
7.4 Project Limitation	88
7.5 Future Works	89
7.6 Conclusion	91
REFERENCES	92
APPENDICES	97

LIST OF TABLES

	PAGE
Table 1.1 : Problem Statement Table	3
Table 1.2 : Project Questions Table	3
Table 1.3 : Project Objective Table	4
Table 1.4 : Project Scope Table	4
Table 1.5 : Table for Report Organization	5
Table 2.1 : Type of blockchain	16
Table 2.2 : Comparisons among different Hyperledger frameworks	22
Table 2.3 : Comparison of blockchain platform	23
Table 2.4 : Comparison of existing work	28
Table 3.1 : Final year project 1 milestones	37
Table 3.2 : Final year project gantt chart	39
Table 4.1 : Participants permission in the network	45
Table 4.2 : Smart contracts in the system	47
Table 4.3 : Software requirement	48
Table 4.4 : Hardware requirement	50
Table 6.1 : Test Description	66
Table 6.2 : Login details for users	67

LIST OF FIGURES

	PAGE
Figure 2.1 : Structure of blockchain	8
Figure 2.2 : Taxonomy of cryptographic primitives	10
Figure 2.3 : Smart contract relationship with the blockchain	11
Figure 2.4 : Ledger on Hyperledger Fabric	20
Figure 2.5 : Architecture of proposed system from previous study	33
Figure 2.6 : Process flow of patient registration	34
Figure 2.7 : Process flow of records sharing and adding new record	34
Figure 3.1 : SDLC phases	35
Figure 4.1 : Comparison of present and future EHR system	41
Figure 4.2 : Data type of patient	42
Figure 4.3 : Data type of medical practitioner	43
Figure 4.4 : Use case of admin	43
Figure 4.5 : Use case of patient	44
Figure 4.6 : Use case of medical practitioner	44
Figure 4.7 : Sequence diagram of creation of a patient	46
Figure 4.8 : Sequence diagram of creation of a doctor	46
Figure 4.9 : System architecture of the blockchain.	51
Figure 5.1 : Application stack in Fabric application	54
Figure 5.2 : Smart contracts hierarchy	55
Figure 5.3 : Example of getPatientHistory() methods in doctor contract	56
Figure 5.4 : Example of revokeAccessFromDoctor() method	56
Figure 5.5 : Example of verifying if the doctor is granted access	57
Figure 5.6 : Example of utilizing fabric-network between backend server and blockchain network	58

Figure 5.7 : Endorsement policy for Hosp 1	62
Figure 5.8 : Chaincode-level endorsement policy	63
Figure 5.9 : TLS enabled in CA Hosp 1	63
Figure 5.10 : Patient’s passwords are hashed and stored in the blockchain (patient chaincode)	64
Figure 6.1 : EHR of patient 0	68
Figure 6.2 : EHR of patient 1	68
Figure 6.3 : EHR of patient 2	68
Figure 6.4 : EHR of patient 3	69
Figure 6.5 : EHR of patient 4	69
Figure 6.6 : EHR of patient 5	69
Figure 6.7 : Testing of bringing up network	70
Figure 6.8 : Testing of creating channel	71
Figure 6.9 : Testing of deploying smart contract	72
Figure 6.10 : Testing of running backend server	72
Figure 6.11 : Testing of running frontend server	73
Figure 6.12 : Main page of user interface	73
Figure 6.13 : Admin login page	74
Figure 6.14 : List of patients (admin)	75
Figure 6.15 : Create new patient	75
Figure 6.16 : Create new doctor	76
Figure 6.17 : Patient login page	77
Figure 6.18 : Mandatory of changing new password for first-time login patient	77
Figure 6.19 : View personal details & EHR (patient)	78
Figure 6.20 : View list of doctors & grant/revoke access	78
Figure 6.21 : Edit personal details	79
Figure 6.22 : Updated personal details	79
Figure 6.23 : View EHR history (patient)	80
Figure 6.24 : Doctor login page	81
Figure 6.25 : Doctor dashboard	81
Figure 6.26 : List of patients (doctor)	82
Figure 6.27 : View patient’s EHR	82
Figure 6.28 : Update patient’s EHR	83
Figure 6.29 : Updated patient’s EHR	83

Figure 6.30 : View patient's EHR history 84




LIST OF ABBREVIATIONS

FYP	-	Final Year Project
EHR	-	Electronic Health Record
PoW	-	Proof of Work
PBFT	-	Practical Byzantine Fault Tolerance
API	-	Application Programming Interface
SDK	-	Software Development Kit
CA	-	Certificate Authority
HLF	-	Hyperledger Fabric
JSON	-	JavaScript Object Notation

LIST OF ATTACHMENTS

	PAGE	
Appendix A	INSTALLATION OF PREREQUISITE	97
Appendix B	STEPS TO START NETWORK	102



The logo for Universiti Teknikal Malaysia Melaka (UTeM) is displayed. It features a circular emblem on the left with the text 'UNIVERSITI TEKNIKAL MALAYSIA MELAKA' around the perimeter and a stylized graphic of horizontal lines. To the right of the emblem is the acronym 'UTeM' in large, bold, white letters on a blue rectangular background.

اونيورسيتي تيكنيكل مليسيا ملاك

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

CHAPTER 1: INTRODUCTION

The purpose of this chapter is to provide project background, problem statement, project questions, project goals, project scope, project contribution and project development for the entire project.

1.1 Project Background

With the rapid development of internet technology, data sharing and storage have become more critical to many industries, including finance, healthcare, government, education and supply-chain as the needs of society to share and retrieve latest information among many different parties has increased. Most industries have adapted the cloud storage service or database service to store and transmit data where there will be a third party organization or company to provide the data sharing and storage services meanwhile the administrator is able to modify or delete any data from the database. It is also the services providers' responsibilities to maintain the operation of the database and ensure regularly security updates. As traditional database is centralized, it has often become a target for malicious party attempting to access sensitive information and gain benefit from it. It can be said that traditional methods of data sharing and storage are more often vulnerable to security breaches, cyber attacks and unauthorized access. Thus, losing guarantee of the integrity and reliability of user records and higher possibility of data loss or data misuse.

To eliminate the dependency for third party to develop a trust-based model, blockchain technology is introduced to provide secure, transparent and tamper-proof transactions. Blockchain is a distributed ledger technology which fully decentralized

peer-to-peer data storage where storage of data spread over all participants also known as nodes of the network in the form of a distributed ledger. Blockchain stores information in growing lists of records (blocks) that are securely linked together through cryptographic hashes meanwhile each block contains hashed information from the previous block, a timestamp and transaction data to provide cryptographic security.

While the most valuable asset within healthcare sector is information where accurate and complete patient records are important not to be compromised, there are some serious issues need to be concerned such as balancing easy accessibility between protecting the privacy of medical data, ensuring the integrity of patient data and managing authorization rules for data access. As healthcare management encompasses many different processes that will require patient related data, data management system that integrated with blockchain technology will optimize the complex medical processes and digital management of medical data. In the process, the blockchain plays an essential role as a distributed database structure for the electronic health record data management in which all transactions are checked and stored by all parties participating in the database. Meanwhile, the data management system which based on smart contracts for intelligent management would automated enable parties who allowed to access or modify patient's electronic health record based on patient's consent. This decentralization of the data would ensure higher security, reduce administrative costs and increase the authenticity of information due to transparency of blockchain technology. However, there is still a lot of improvement needed in blockchain technology in order to provide accountable, reliable and secure solutions to industries.

The researcher would identify the implementation of blockchain technology using web-based application for electronic health record data management among hospitals and point out the performance of the blockchain-based web application to identify whether it is better in data confidentiality and integrity.

1.2 Problem Statement

Existing centralized data storage services which might lead to single point of failure where such kind of systems usually rely solely on third party such as large

company who provides strong storage capacity to store and transmit data. Redundancy of different healthcare providers storing patients' health records in their own databases and problems in ensuring data integrity when retrieving and transmitting health records.

Table 1.1: Problem Statement Table

No.	Problem Statement
1	Existing centralized data storage services which might lead to single point of failure where such kind of systems usually rely solely on third party such as large company who provides strong storage capacity to store and transmit data.
2	Redundancy of different healthcare providers storing patients' health records in their own databases and problems in ensuring data integrity when retrieving and transmitting health records.

1.3 Project Questions

Table 1.2: Project Questions Table

No.	Project Questions
1	How can blockchain technology helps in securing and sharing data of electronic health records among hospitals?
2	Are blockchain-based web application for EHR data management more efficient than conventional storage method?
3	Does blockchain-based EHR data management system able to ensure data integrity and transparency?

1.4 Project Objective

This study's goal is to determine the application of blockchain technology for electronic health record among hospitals in healthcare sector. The second objective is to design and implement a prototype of blockchain-based web application for

electronic health record data management among hospitals. Finally, the third objective is to evaluate and analyze the performance of the blockchain-based prototype for electronic health record.

Table 1.3: Project Objective Table

No	Project Objective
1	To determine the application of blockchain technology for electronic health record among hospitals in healthcare sector.
2	To design and implement a prototype of blockchain-based web application for electronic health record data management among hospitals.
3	To evaluate and analyze the performance of the blockchain-based prototype for electronic health record.

1.5 Project Scope

This project would focus on developing a blockchain-based web application that is used for electronic health record data management among hospitals. The scope for this project is explained below:

Table 1.4: Project Scope Table

No	Project Scope
1	Understanding the architecture of blockchain technology and its application for electronic health record data management among hospitals.
2	To develop a web application that demonstrates the use of blockchain for electronic health record data management in healthcare sector.
3	Ensure data privacy and confidentiality for the blockchain-based web application for electronic health record data management among hospitals.

1.6 Project Contribution

The essential key in this project is to propose a more secure and efficient method using blockchain technology for electronic health record data management

that can ensure smooth operation among hospitals. Lastly, it will address the implementation of blockchain-based electronic health record data management system will be able to overcome the security issues of using conventional method of data storage and sharing.

1.7 Report Organization

Table 1.5: Table for Report Organization

Chapter	Detail
Chapter 1 Introduction	Research background, a problem statement, project questions, project objectives, project scope, and a project contribution are all included in this chapter.
Chapter 2 Literature Review	This part will be reviewing past research, journal and conference papers as well as related works for the project matter.
Chapter 3 Methodology	This chapter explains the approaches used in phasing out job activities to contextualize the process being introduced. For this work 4 steps are to be included. Using this phase encourages those tasks to be executed and managed. It involves simple framework and detailed framework.
Chapter 4 Design	This chapter discusses the requirement that will be used to run this project and a flowchart. The flowchart is discussed in detail about how the process works.
Chapter 5 Implementation	This chapter implements the new method and architecture that being used in controlled environment.

Chapter 6 Testing and Validation	The research and results of the suggested method were covered in this chapter. This chapter also determines whether the suggested approach works or not.
Chapter 7 Project Conclusion	This last chapter summarizes the entire process and explain further progress which could be implemented in the future.

1.8 Conclusion

To summarize, this chapter explains clearly about the general background of this project and highlight the potential implementation of blockchain technology in managing data for electronic health record among hospitals. The next chapter will explain about the related work.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter provides an overview of blockchain technology including its core technologies, types and algorithms. It also introduces the advantages and limitations of different blockchain platforms that are available to be used and then focus on private permissioned blockchain using Hyperledger Fabric to create and deploy decentralized applications. Furthermore, this chapter will highlight some critical reviews of the existing works and its proposed solution.

2.2 Blockchain Technology

Blockchain is a distributed ledger technology that records all the transactions that have occurred in the peer-to-peer network which the information is stored in growing lists of records also known as blocks where the blocks are securely linked together through cryptographic hashes to form a chain. A transaction represents a transfer of value from one address to another. When the number of transactions grow, the size of blockchain will grow as well. Blocks in the blockchain store the chronological order of transactions along with their respective timestamps (Usman & Qamar, 2020). Blockchain is called a secure, decentralized and immutable database as it eliminates the dependency on trusted third-party where there is no single node can control the entire network and all participants within the network possess an identical copy of the database (Wang et al., 2018).

The Genesis block, an initial block without any transactions, is the predefined start of the blockchain, then followed by subsequent blocks which constructed with transactions that are added to the blockchain. Each block contains a block header and

a series of transactions. Meanwhile each block header contains data such as link pointers to the previous block's hash, nonce which is a number generated and used only once for transaction replay protection and PoW consensus algorithms, timestamp of the block created and Merkle root which is the hash of all transactions in a block that can be used to verify all transactions present in the Merkle tree. As each block is linked to the preceding block by hashing the block's data along with the previous block's hash, this ensures the immutability and integrity of the data stored in the chain and prevents unauthorized modifications to the blockchain (Ismail & Materwala, 2020). A block is considered valid when there is more than 50% of the participants in the network reach an agreement on verifying the validity of transaction data using a consensus algorithm.

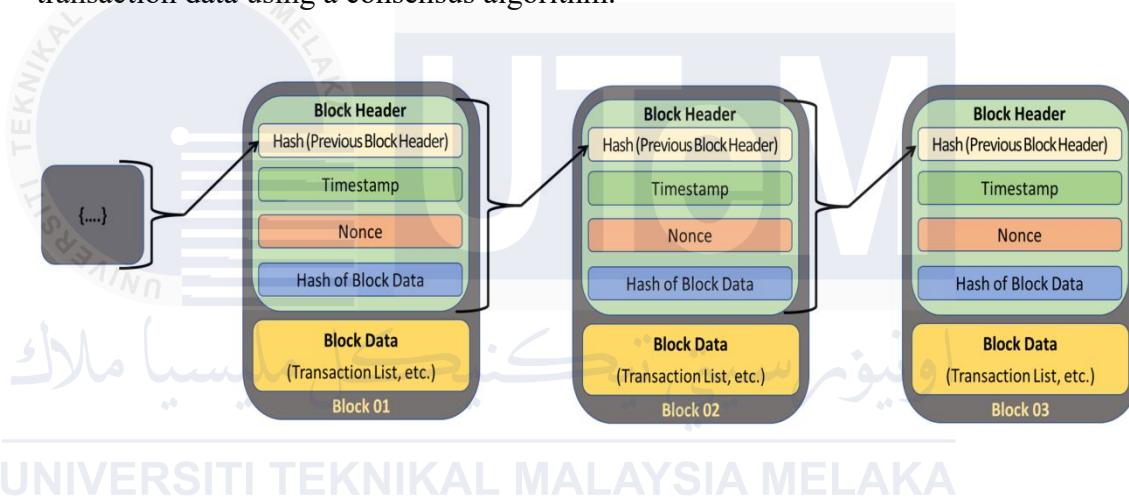


Figure 2.1: Structure of blockchain

(Odeh et al., 2022.)

2.2.1 Core of Blockchain Technology

According to Tanwar et al. (2020), the process of verifying and validating transactions in a blockchain is complex, as it involved techniques such as distributed ledger network, cryptography, smart contract and consensus protocol. There are various steps required to complete a blockchain transaction. In the first step, the network node requests the transaction by first creating and digitally signing it with its private key. The transaction is then broadcast using a flooding protocol, called Gossip protocol to peers that validate the transaction based on preset criteria including smart contract and consensus algorithm. Once the transaction is validated, it is included in a block, which is then propagated onto the network. At this point, the transaction is considered confirmed. The newly-created block is appended in the

ledger and the next block will link itself cryptographically back to this block through hash pointer. At this stage, the transaction gets its second confirmation and the block gets its first confirmation. Usually, transactions are reconfirmed every time when a new block is created. Lastly, the transaction is completed or committed to the network.

There are four core technologies that is essential in blockchain solution which are distributed ledger technology, cryptography, smart contract and consensus protocol or trust systems. These technologies are designed to work together to create a secure, transparent, and decentralized system for storing and processing data in blockchain.

2.2.1.1 Distributed Ledger Technology

A distributed ledger, also known as shared ledger or distributed ledger technology (DLT) is the consensus of replicated, shared, and synchronized digital data that is distributed among its network participants and spread across multiple sites, countries, or organizations (Bashir, 2018). Compared to a centralized database, transactions are appended in a distributed system on the network which did not required a central administrator and consequently eliminate a single point of failure. While all distributed ledgers are based on a decentralized structure, it is important to note that not all distributed ledgers can be classified as blockchains. The key distinction lies in the fact that a distributed ledger might not necessarily involved blocks of transactions to maintain the growth of the ledger. On the other hand, a blockchain is a specific form of shared database that specifically comprises blocks of transactions.

2.2.1.2 Cryptography

Cryptography involves the use of cryptographic algorithms and security protocols to secure transactions and protect sensitive data. It has been applied in blockchain to provide security services including non-repudiation, data integrity and data origin authentication which makes the ledger secure against tampering and misuse. The design of blockchain included this requirement due to the computational hardness assumption and making encryption harder to be break. Cryptography is

mainly divided into two categories which are symmetric cryptography and asymmetric cryptography.

(a) *Symmetric Cryptography*

According to Bashir (2018), symmetric cryptography also known as shared key cryptography refers to a type of cryptography where the key that is used to encrypt the data is the same for decrypting the data. Before the data exchange occurs between the communicating parties, the key must be established or agreed upon both parties. There are two types of symmetric ciphers which are stream ciphers and block ciphers. Typical example of stream ciphers are RC4 and A5 meanwhile Data Encryption Standard (DES) and Advanced Encryption Standard (AES) are examples of block ciphers.

(b) *Asymmetric Cryptography*

Asymmetric cryptography refers to a type of cryptography where two distinct keys are involved in the process which are a public key that mainly used for encryption or verifying key and a private key mainly used for decryption or signing key (Guo & Yu, 2022). This is also known as public key cryptography. Asymmetric key cryptography is well-suited for use in blockchain technology to authenticate identities for transactions and individuals. Various asymmetric cryptography schemes are introduced including RSA, DSA, and ElGammal.

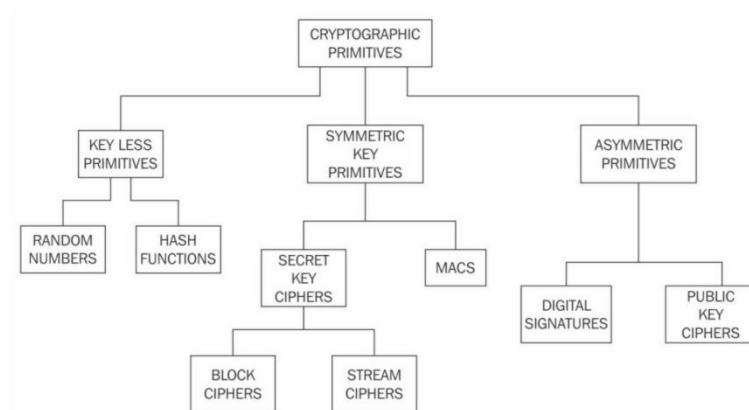


Figure 2.2: Taxonomy of cryptographic primitives

(Bashir, 2018.)

2.2.1.3 Smart Contract

A smart contract, introduced by Nick Szabo in 1994 is defined as a ‘set of promises, specified in a digital form, including protocols within which the parties perform on the other promises’ (Antonopoulos & Wood, 2018). The implementation of smart contracts in blockchain creates a platform for automatic, self-executing and self-verifying transactions based on specific rules and regulations where contracts are designed to execute when certain conditions and variables have been met. This allows for flexibility and traceability as smart contract can autonomously execute some or all of the operations related to a contract and provide verifiable evidences which allows an organization to keep a record of all the transactions without the need for human intervention. Moreover, the rules of a smart contract are acknowledged and enforced by multiple organizations that operate on a decentralized blockchain network for interaction among the users across the network. These smart contracts can automatically execute based on users conditions to provide a secure method and reduce the cost (Kumar et al., 2021).

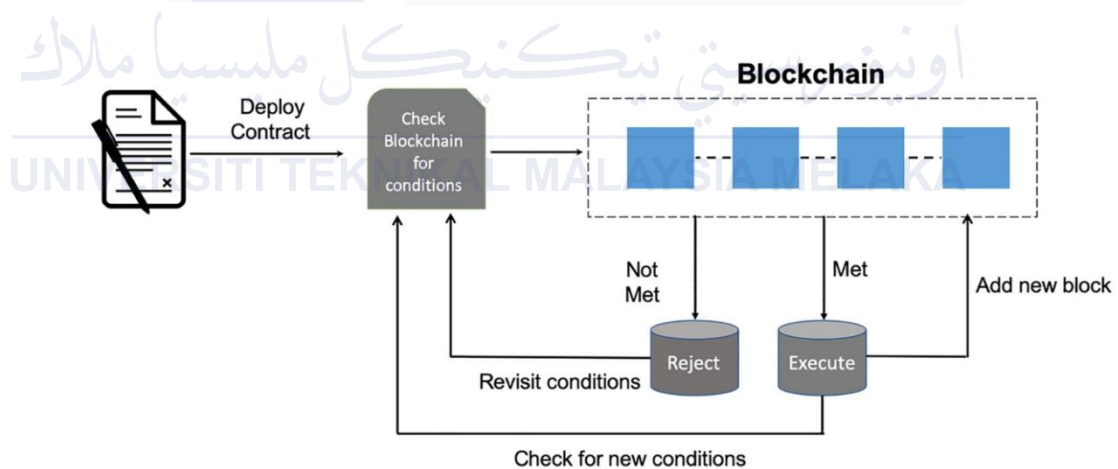


Figure 2.3: Smart contract relationship with the blockchain

(Jabbar & Dani, 2020.)

2.2.1.4 Consensus Mechanisms

According to Bashir (2018), a consensus mechanism refers to a series of steps followed by the majority or all nodes within a blockchain network to reach an

agreement on a proposed state or value. In client-server systems, it is easy to reach an agreement between two nodes, however in a distributed system where there are multiple participant nodes, it has become a challenge for all the nodes to agree on a single value to achieve consensus. The consensus protocol adopted by a blockchain ensures smooth execution of operation as it will determine which block to append and how it should be appended, validation of block's contents and construction of block in the process. Transactions are only updated when all verified users in the network agree to the condition of the transaction and then come to a conclusion of adding or dropping a block in the blockchain. It solves the problem of trust in blockchain, as assumed that all nodes in the network are untrusted and similar algorithm is needed to agree on the validity of the block. Thus, consensus mechanism can keep consistency in ledger synchronization and prevent malicious transactions (Muhammad & Soewito, 2022).

The characteristics of blocks, such as their structure and the time intervals between them, can be adjusted by modifying the parameters within the consensus protocol. Meanwhile the selection of the consensus algorithm is influenced by the specific type of blockchain being utilized as not all consensus mechanisms are appropriate for every type of blockchain. The following shown some of the common and popular consensus algorithms that have been introduced:

(a) Proof of Work (PoW)

Proof of Work (PoW) mechanism is based on mining which demands miners to contribute a great number of computing power to solve a very difficult cryptography puzzle by constantly trying to assemble blocks and new random numbers until the correct random number is found (Wang et al., 2018). Mining is a process of creating new blocks to the blockchain. To prevent the concentration of computational power, the difficulty, also called nonce, for generating the next block is adjusted dynamically on the basis of 10 minutes per block. While it is the only algorithm that has proven against any collusion attacks on a blockchain network, such as Sybil attack, PoW also results in low transaction throughput and excessive energy usage (Gadekallu et al., 2022). The popular implementation of this scheme are Bitcoin, Litecoin and other cryptocurrency blockchains.

(b) *Proof of Stake (PoS)*

Proof of Stake (PoS) addressed the problems posed by Proof of Work (PoW) mechanisms where a miner node is selected to invest computational power through pseudorandom methods weighted in relation to degree of ownership, credibility or reputation or etc rather than all miner nodes competing to create the next block depending on computational power. Another important concept in PoS is coin age which is determined by the duration and quantity of unspent cryptocurrencies. In this model, the probability of proposing and validating the next block is higher as the coin age increases. As stated by Dagher et al. (2018), such a miner selection approach leads to significantly reduced of resource consumption.

(c) *Delegated Proof of Stake (DPoS)*

Delegated Proof of Stake (DPoS) is an alternative to PoS where it is more efficient and scalable than PoS. In DPoS, each node that has a stake in the system can delegate the validation of a transaction to other nodes by voting. The delegates play a crucial role in maintaining consensus, mining, and validating new blocks. As rewards are earned, they are distributed proportionally among the stakeholders and their delegates. This algorithm relies on a democratic voting system, where the effectiveness and ethical behavior of delegates determine their reputation and continued participation in the network. Delegates who fail to operate efficiently or ethically may be expelled from the network to ensure its proper functioning. Examples of networks that use DPoS include BitShares blockchain, Ark and Lisk.

(d) *Byzantine Fault Tolerance (BFT)*

Byzantine fault tolerance (BFT) refers to the capability of a network or system to sustain its functionality even in the presence of faulty or failed components. In a BFT system, blockchain networks can continue to operate and execute intended actions as long as the majority of network participants are trustworthy and genuine. This implies that for a transaction to be validated and added to the blockchain, it requires agreement from more than half or two-thirds of the network nodes. However, BFT algorithms for asynchronous networks are only practical up to about 1000

participants due to the incurred overhead of the cryptographic algorithms (Knirsch et al., 2019).

(e) *PBFT*

Practical Byzantine Fault Tolerant (PBFT) consensus is an efficient consensus algorithm used in blockchain networks to achieve Byzantine fault tolerance. It involves participation of a set number of nodes, known as replicas that collectively agree on the validity and order of transactions. PBFT utilizes a leader that is elected among the replicas who proposes a block of transactions and initiates a voting process among the replicas. Once a sufficient number of replicas reach agreement, the proposed block is considered finalized and added to the blockchain. PBFT requires $3f+1$ nodes in order to keep the system stable, where f is the maximum count of defective nodes the system can handle. As a result, approval from $2f+1$ nodes is needed for the group of nodes to make any decision. PBFT provides high fault tolerance as long as the majority of replicas are trusted. However, it requires a predefined set of replicas, which can limit scalability. PBFT is commonly used in permissioned blockchain networks where efficiency and fault tolerance are essential.

(f) *Proof of Storage (PoS)*

Proof of Storage (PoS) is based on the concept that a particular piece of data is probably stored by a node which serves as a means to participate in the consensus mechanism rather than computational power (Bashir, 2018). In a PoS blockchain, nodes must prove that they are storing specific data or files to participate in the consensus process. This ensures that participants are actively contributing to the network's storage capacity. PoS promotes efficiency by reducing the need for computational resources and shifting the focus to storage capabilities which will enhance data integrity and availability through decentralization. Several variations based on PoS have been proposed such as Proof of Replication, Proof of Data Possession, Proof of Space and Proof of Space-Time.

(g) *Proof of Activity (PoA)*

Proof of Activity (PoA) is a combination of Proof of Work (PoW) and Proof of Stake (PoS) mechanisms, which participants are required to demonstrate both computational work and ownership of a certain amount of cryptocurrency to achieve consensus. It is more energy efficient as participants alternate between mining blocks through PoW and validating blocks through PoS. This consensus algorithm offers a compromise between security and energy efficiency while maintaining decentralized participation in the network.

2.2.2 Types of Blockchain

Blockchains can be categorized into three types which are public, private and consortium.

2.2.2.1 Public Blockchain

A public blockchain also known as permissionless blockchain, is a chain that anyone in the world can participate into the network to read, write, access or send a transaction to a valid user. According to Purwono et al. (2023), anyone can contribute in a consensus process to determine what blocks can be appended to the end of the chain as public blockchains are accessible by anyone. After validation of the newly created block that linked to the chain, other participants will either expand their individual copies of the chain with the newly generated block then broadcast the block to other participants in the network or else discarded the invalid block from the chain. This kind of blockchain is generally considered as decentralized network where there is no central authority controlling the system. Public blockchain is widely used in cryptocurrencies for example Bitcoin and Ethereum, are categorized as public chains without permission.

2.2.2.2 Private Blockchain

A private blockchain is a restricted type of blockchain network created by an entity or organization. It is also recognized as a centralized blockchain, where a central authority is granted the authorization to control transactions across the entire chain such as to add, delete or modify data (Dagher et al., 2018). This type of

blockchain is only accessible to those with access permissions where new users are required to obtain prior permission before they can join the network. It is mainly used to handle databases or private applications intended for sensitive data manipulation as only specific participants can be granted defined read access along with limitations on creating transactions. In private blockchain, the predefined nodes who participate in the consensus management will be responsible for validating transactions. Due to the nature of private blockchains, not all nodes are required to verify transactions. By implementing this approach, the transaction speed within the private blockchain can be significantly accelerated, making it the fastest blockchain-based solution available. Moreover, it will lead to a reduction in the overall workload required for processing transactions. Examples of private blockchains are Hyperledger, HydraChain and Quorum.

2.2.2.3 Consortium Blockchain

A consortium blockchain also known as semiprivate blockchain, consists of private part which is controlled by a group of individuals, while the public part is open for participation by anyone. It can be recognized as semi-decentralized model as only a select group of entities have the access right to view and participate in the consensus protocol meanwhile multiple users are allowed to join the network by following appropriate procedures, such as allowing mining. As the consortium blockchain is decentralized and is managed by multiple organizations, it can be used by banks, supply chain or food tracking companies where organization can manage and control data access while part of the data can be available to public (Antwi et al., 2021). Moreover, the consensus algorithm could be modified to use different ideas such as voting based concept. The consensus policy can be configured to require a specific number of nodes in the network to participate in voting or digitally signing the block before it is officially added to the blockchain. This modification ensures that consensus is reached through a democratic process and the blockchain can be secured using PoW, thus providing consistency and validity for both the private and public parts.

Table 2.1: Type of blockchain

Properties	Public blockchain	Consortium blockchain	Private blockchain
Read permission	Public	Public or restricted	Public or restricted
Consensus determination	All miners	Selected set of nodes	One organization
Consensus process	Permissionless	Permissioned	Permissioned
Centralized	No	Partial	Yes
Immutability	Nearly impossible	Could be tampered	Could be tampered
Efficiency	Low	High	High

(Muhammad & Soewito, 2022).

2.3 Blockchain Platform

Due to the diverse requirements of businesses and users, it is not feasible to have a one-size-fits-all blockchain network that can cater to all industries. This has led to the creation of different blockchain platforms which allow developers and users to create and execute applications on an existing blockchain network with a different set of protocols. Generally, blockchain platform supplies a collection of functionalities and tools that enable the management and creation of smart contracts, launching decentralized applications (dApps) and communication with the blockchain network. In this section, evaluation of some common blockchain platform and its applicability of different types of consensus algorithms and permissions are done as followed:

2.3.1 Ethereum

Ethereum is an example of public blockchain which was first proposed in 2013 by Vitalik Buterin. It is the first blockchain that introduced a Turing-complete language and the concept of a virtual machine. According to Wang et al. (2018), the Bitcoin system utilizes a scripting language that operates on a stack-based non-Turing complete model which can only support simple logic, thus limiting its application in many fields. Compared to the Bitcoin system, the availability of Turing-complete language called Solidity in Ethereum has brought wider possibilities for the development of decentralized applications using smart contracts. As the platform supports the application of Turing complete, it allows developers to

deploy systems with smart contracts where the logic of that system is transformed into the code. Once the contract is deployed, it can be automatically executed according to the agreed logic of smart contracts. Due to smart contracts are capable of representing practically arbitrarily complex transactions, it extends beyond financial transactions found in Bitcoin and enable the representation of states and their alterations.

On the Ethereum blockchain network, mining is a crucial process that includes new blocks to the blockchain using proof of work (PoW) algorithms. Meanwhile Ether (ETH) is the native cryptocurrency of the platform which the second most popular cryptocurrency after Bitcoin (BTC). It is used to pay for transaction fees and to incentivize network participants to maintain and update the blockchain. There are two types of Ethereum accounts can be created on Ethereum Virtual Machine (EVM) which are Externally Owned Accounts (EOAs) and contract accounts. EOAs are controlled by user's private key and are usually owned by devices or users while contract accounts are controlled by their contract code (smart contract) contained in them.

Although Ethereum is known for its versatility, prior studies have revealed that even moderately complex smart contracts can be quite costly. Additionally, the expense of executing operations and the unpredictability of costs are mostly influenced by the fluctuations in exchange rates to Euros. Meanwhile updating a modified Merkle Patricia Trie to store values in smart contracts is a time-consuming process, speed of execution is also tend to slow down as data volume grows.

2.3.2 Hyperledger

Hyperledger is an open source private blockchain project initiated by the Linux Foundation in December 2015 and has support from companies such as IBM and Intel to SAP Ariba. According to Hyperledger Foundation, Hyperledger has many frameworks and tools that can be used to build blockchain networks. While each of these frameworks and devices has a specific function, they can also collaborate during the implementation process of creating a blockchain network. There are five hyperledger frameworks which are Fabric, Sawtooth Lake, Burrow, Indy and Iroha meanwhile there are three modules that support these blockchains

which are Explorer, Cello and Composer. These frameworks will be discussed in later section.

According to Usman & Qamar (2020), Hyperledger employs a modular architecture that offers flexibility in terms of consensus and membership services. It utilizes container technology to host smart contracts, known as chaincode, which encapsulates the system's application logic. The modular architecture of Hyperledger enables the integration of various consensus algorithms, such as the Practical Byzantine Tolerance Algorithm (PBFT), which is computationally more efficient compared to Proof of Work (PoW). This design approach allows for easier customization and adaptability of the Hyperledger framework to meet diverse requirements in the blockchain ecosystem.

In Hyperledger blockchain, each node in the network possesses a unique identity. The Member Service Provider (MSP) plays a role in issuing cryptographic certificates to all participating nodes through a public key infrastructure. Network users are provided with a username and password combination, which is utilized to obtain an Enrollment certificate (Ecert). Transaction certificates (TCert) are issued to Ecert owners by the Transaction Certificate Authority. It is possible to derive multiple TCerts from a single Ecert. When conducting transactions, network participants employ Transaction Certificates (TCerts) as a means of authentication.

Besides, Hyperledger utilizes smart contracts, known as chaincode, to incorporate the business logic that governs transaction execution and changes to the World-State. The World-State represents a state database that stores the current values of various ledger states, along with their associated block numbers. Hyperledger offers a scalable, secure, and flexible blockchain platform with significant potential to revolutionize numerous industries.

2.3.2.1 Hyperledger Fabric

Hyperledger Fabric can be considered as an open-source platform for the permissioned blockchain proposed by IBM and DAH (Digital Asset Holdings). It has a modular design and architecture and therefore has a high degree of flexibility and extensibility (Zhang et al., 2020). It is based on a pluggable architecture where

various components, such as consensus engine and membership services, can be plugged into the system as required. Moreover, it is the first platform that supports smart contract through container technology using programming languages, such as Go and Java (Namasudra et al., 2020). Currently, its status is active and it's the first project to graduate from incubation to active state.

In Hyperledger Fabric, there is a database called world state that holds current values of a set of ledger states. This ease the program to directly access the current value of a state rather than having to calculate it by traversing the entire transaction log. Another record is transaction log, which it records all the changes that have resulted in the current world state. This enables transactions are collected inside blocks which are appended to the blockchain (Muhammad & Soewito, 2022).

Besides, the ability of Fabric to create trusted subnetworks, called channels, that can establish shared ledgers with a defined set of nodes and transact to the exclusion of the rest of the blockchain allows for the confidential execution of transactions. This approach ensures that information is not transmitted through a central authority, thereby enhancing security and privacy. Furthermore, it also provides secure interaction between different participants and organizations as the use of Crase Fault Tolerance (CFT) or Byzantine Fault Tolerance (BFT) consensus mechanism do not require more cost for mining. However, Hyperledger blockchains still have the challenge of the computational and storage cost of large block sizes.

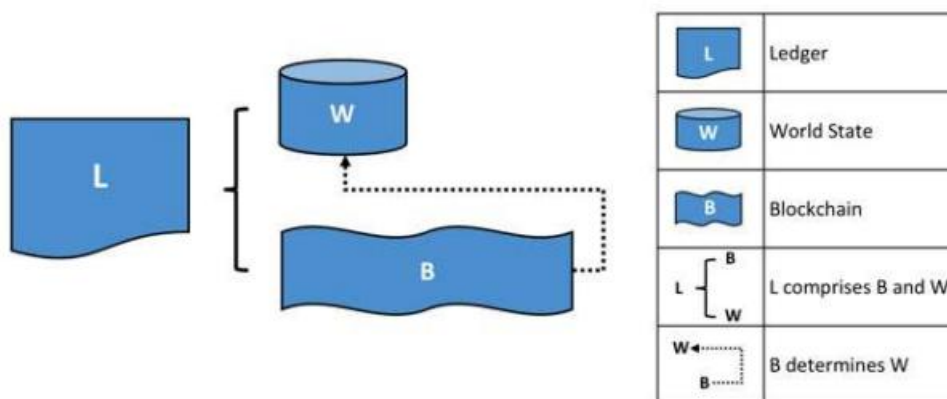


Figure 2.4: Ledger on Hyperledger Fabric

(Muhammad & Soewito, 2022.)

2.3.2.2 Hyperledger Sawtooth Lake

The Sawtooth Lake is a blockchain project proposed by Intel in April 2016 which network nodes are able to deploy with separate permission efficiently using transaction families and pluggable consensus. Based on the patterns and structures defined in the transaction families, Sawtooth separates the execution of every transaction from one another which allows for more flexibility, rich semantics and open design of business logic. It is mainly built for solving the challenges or problems arised in private networks. There are several components of Sawtooth including Sawtooth validators, Sawtooth applications, transaction processors, batch, network layer, global state and PoET. PoET, as known as Proof of Elapsed Time is a consensus algorithm which makes use of Trusted Execution Environment (TEE) provided by Intel Software Guard Extensions (Intel's SGX) to provide a safe and random leader election process (Bashir, 2018).

2.3.2.3 Hyperledger Iroha

Iroha was contributed by Soramitsu, Hitachi, NTT Data and Colu in September 2016. Its objective is to develop a collection of reusable components that users can select to operate on their own distributed ledgers based on Hyperledger. Iroha's primary goal is to complement other Hyperledger projects such as Sawtooth and Fabric by providing reusable components written in C++ with an emphasis on mobile development. This project has also proposed a novel consensus algorithm called Sumeragi, which is a chain-based Byzantine fault tolerant consensus algorithm. This feature makes it distinct from other Hyperledger frameworks (Namasudra et al., 2020).

2.3.2.4 Hyperledger Burrow

Hyperledger Burrow was contributed by Monax, who develop blockchain development and deployment platforms for business. Burrow introduces a modular blockchain platform and an Ethereum Virtual Machine (EVM) based smart contract execution environment. It employs a proof of stake consensus mechanism with

Byzantine fault tolerance called Tendermint. As a result, Burrow provides high throughput and transaction finality. However, Burrow is still in the incubation stage.

2.3.2.5 Hyperledger Indy

Indy is a distributed ledger developed for building a decentralized identity and to support participants to control and manage their identities rather than using a large amount of personal information. It was developed by Sovrin Foundation to provide tools, utility libraries and modules which can be used to build blockchain-based digital identities. These identities can be used across multiple blockchains, domains and applications where authentication is based on the attributes users shared. Indy has its own distributed ledger and uses Redundant Byzantine Fault Tolerance (RBFT) for consensus.

Table 2.2: Comparisons among different Hyperledger frameworks

Properties/ Type	Fabric	Sawtooth	Iroha	Burrow	Indy
Membership service	Yes	No	No	No	No
Modularity	High	High	Less	Less	Average
Flexibility	Average	Average	High	Average	High
Scalability	Less	High	Less	Average	Average
Decentralized identity	No	No	No	No	Yes

(Namasudra et al., 2020).

2.3.3 MultiChain

MultiChain is a private and permissioned blockchain implementation. As like Bitcoin, it is mainly be used within an organization for facilitating financial transactions and thus the consensus algorithm is PoW-based. Although it will be easier for management compared to Bitcoin, the less documentation of implementation of network using MultiChain and its supported platform are currently a major limitation.

2.3.4 Open Chain

OpenChain is a private blockchain that prioritizes energy efficiency, network communication, and block rate. To achieve these goals, it diverges from the peer-to-peer model and adopts a client-server architecture. Instead of using proof of work, OpenChain employs a consensus algorithm known as proof of authority. The selection of a specific authority to validate transactions would contradict with the desired security and trust properties within blockchain while this applied to other blockchains, such as Corda, that utilize Proof of Authority (PoA) to establish consensus (Knirsch et al., 2019).

Table 2.3: Comparison of blockchain platform

Properties/ Platforms	Ethereum	Hyperledger	MultiChain	Open Chain
Permissioned	✗	✓	✓	✓
Smart contract	✓	✓	✗	✗
Consensus	Proof of Work (PoW)	Pluggable framework	Proof of Work (PoW)	Proof of Authority (PoA)
Governance	Ethereum developers	Linux Foundation	Coin Sciences Ltd	Coinprism

2.4 Critical Review of Existing Works

As stated by Donawa et al. (2019), using Blockchain in electronic health records offers a convenient and symmetric health record storage service that promotes easy accessibility of such records through the web. The system is often designed to allow the patients full control of generating, managing, and consequently sharing their electronic health records with friends, family, healthcare providers, and other relevant data consumers.

While Dagher et al. (2018) proposed a approach uses Ethereum's public and private blockchains, smart contracts and a local database to provide patients with

secure access to their electronic health records, it allows a patient to send requests to its provider's system through the blockchain. The system stores data in the provider's local database such as file name, a secure link, patient's Ethereum address and public key of the patient, which it is a separate layer outside the blockchain. The database will only be used by the provider when adding a new file and when retrieving the secure link upon valid request from the patient. Smart contracts on the Ethereum blockchain are utilized to automate tasks and achieve access control. This solution however, does not support a hybrid system such as a Bitcoin blockchain and an Ethereum blockchain as these platforms employed different protocols.

According to Ismail & Materwala (2020), there is currently no existing research or study that directly compares client/server-based healthcare data management systems with blockchain-based counterparts. Therefore, the authors implemented a minimal blockchain-based healthcare platform and compared its execution time and amount of data transferred with the client/server system model for health records update and query by increasing the number of records and hospitals. A permissioned blockchain network is developed due to its advantages over the permissionless where the model consists of participants such as patients, allied health professionals and administrators. PBFT consensus protocol is used in the developed blockchain-based healthcare platform rather than Proof of Work (PoW) as PoW consumes more energy and less throughput than the former. It has been found that blockchain platform is 11.7 times faster compared with the client/server model in querying health records with increasing number of health records, but blockchain-based system model is more costly than the client/server system model as the longer execution time taken and larger amount of data transferred in updating health record. This is due to exchange of messages to update ledger involving PBFT consensus.

A proof-of-concept solution has been proposed by Newman & Thorpe, which is a private permissioned blockchain that structured with three layer which are application layer interface to patient EHR, blockchain application/transaction layer and a off-chain storage for large patient centric data. A hashing mechanism is used for linking the blockchain stored information such as patient demographic/metadata to the storage area information that stored EHR specific data. Hyperledger platform

is preferred as it can improve patient experience with its modular and extendable architecture that offers a broad understanding of consensus.

Besides, Purwono et al. (2023) has also suggested that the Hyperledger platform is suitable for healthcare applications as Hyperledger provides extensive control over smart contracts, allowing them to be executed using various programming languages such as Node.js and Javascript. Compared with Ethereum that can complete fifteen transactions per second, Hyperledger outperforms with transaction speeds of up to 3000 transactions per second.

Usman & Qamar (2020) have developed a permissioned blockchain-based system for efficient storage and sharing of electronic medical records (EMRs) which provides better security and privacy of data. They selected Hyperledger over Ethereum because Ethereum uses proof-of-work (PoW) algorithm that requires lot of resources for mining and transaction execution while permissioned feature in Hyperledger allows proper care of privacy requirements of medical data. Consensus algorithm used is PBFT consensus algorithm as to check whether a transaction is valid or not. The application focuses three type of users: Patients, Healthcare-Providers and Health Administration where Health Administration will be responsible for the registration of patients and doctors while patients can control of who can add new records and view their medical history. As storing large amount of data in blockchain might degrade the performance of whole blockchain system, the blockchain will only hold transaction information and the WorldSate database will hold data values (actual data), in this case CouchDB is used.

Other than that, a permissioned consortium blockchain network is created in which all participating healthcare stakeholders and their end users are identified and registered by health authorities through membership services component using certificate issuing authority (Muhammad & Soewito, 2022). The system involves users such as patient, hospitals and financial institutions where those institutions can only create and view data on the smart healthcare system with the patient's permission. This has comes with the use of multi-channel method where it allows organizations to use the same network while maintaining segregation between multiple blockchains. Only channel members (peers) are allowed to view

transactions made by any member in the channel. In this case, Hyperledger Fabric is used to create a blockchain with multi-channel so that access controls for respective channels can be configured by certificate authorities.

In 2020, a consortium blockchain platform, Hyperledger Fabric is used to develop a blockchain system based on purpose access control in order to build a secure channel in a network among participant healthcare organizations. All metadata of patient records, consents and data access are written immutably on the blockchain and shared among participant organizations where patient records are stored off-chain. Blockchain chaincode that performs business logic managing patient consent are also used so that patients can create, update and withdraw their consents in the blockchain.

Khatoon A. (2020) has proposed a decentralized application (DApp) that supports a private blockchain network with a back-end distributed file system (DFS). The main objective is to share the information through blockchain smart contracts by permitting labs, doctors, emergency clinics and different partners to effectively access and share a patient's therapeutic information among different stakeholders. In this solution, Ethereum has been used where PoW consensus is utilized. By comparing DFS content with ledger records, the DApp would have the ability to detect anomalies, unauthorized data insertions and missing entities. All of the medical record data is stored in local database storage to maintain the performance and hash of the data is the data element of the block committed to the chain. For announcing smart contracts to the blockchain, Ethereum Wallet has been utilized.

According to Tanwar, Parekh & Evans (2020), a Hyperledger Fabric-based EHR sharing system and its related test environment that was based on Hyperledger composer has been proposed. This solution has used access control policy algorithm with symmetric key cryptography to improve data accessibility between healthcare providers and patient, also, the concept of chaincode is applied in assisting the simulation of environments to implement the Hyperledger-based electronic healthcare record (EHR) sharing system. A shared symmetric key enable the transaction and validation of EHR to be distributed to other participants in the blockchain network while a private key is required for user to login. Similar to the other previous studies

that have been stated, there are four types of participants in the EHR sharing system including admin, patients, clinicians and laboratory staff where participants have different roles in the system and can only access records that they have been granted access. While all transactions are committed into the blockchain network using patient public IDs that do not contain personal information, this means that the database of blocks stores only non-identifiable patient data such as gender, age and illnesses etc. In order to test the performance of the blockchain network, performance metrics such as latency, throughput and round trip time (RTT) have been considered to compare with traditional EHR systems which use client-server architecture. In this case, a benchmarking tool is used for the blockchain network, called Hyperledger caliper to verify and execute the performance of the system and its various parameters, including latency, throughput, CPU usage, memory consumption, disk write/read, network I/O, etc. It can be found that the increase in organizations and peers will increase the time needed to execute transactions thus higher latency. Plus, it is found that query a transaction on the blockchain network is much faster than writing a transaction, this is probably due to the nature of blockchain network where there are various ledger peers in each organization in the network which are used for carrying a copy of the ledger.

In addition, Antwi, M. et al. (2021) has proposed a permissioned blockchain solution for healthcare applications. Hyperledger Fabric is used as it allows a developer to manage user authentication and authorization while restricted messaging paths known as private channels are also utilized, which will provide both confidentiality and privacy for transactions. Besides, smart contract is also implemented to develop different access right such as a hospital account only can create practitioners' (doctors) accounts. The participants involved in the system are admin, member, medical institution, medical practitioner and patient. This give patients a full control of their EHR and they can decide who can access their health record and for what purpose, for example only permissioned medical practitioners are allowed to view or update patient's information. Although certain research suggested using a blockchain with a relational database to store the data, this could create new risks to personal/critical data security and privacy directly.

In the latest solution proposed by Ndzimakhwe, M., Telukdarie, A., Munien, I., Vermeulen, A., K., U., & Philbin, S. P. (2023), a framework for user-focused EHR system has been developed using Hyperledger Fabric. The orderer certificate authority (CA) is created and utilised by all the peers of hospital where the orderer acts as an admin that approves all organizations and validates the credentials of their peers. In this paper, there is a single channel created to connect two hospitals while it is possible to add additional hospitals into the channel. Similar to other studies, patient has full control of his/her own EHR and can decide to grant or revoke permission to access his/her data from a particular doctor. Meanwhile doctor can view or modify patient data and patient can view all the fields but edit only personal fields.

Table 2.4: Comparison of existing work

No.	Research	Blockchain Type	Platform	Consensus Algorithm	Difference
1	Usman & Qamar (2020). Secure Electronic Medical Records Storage and Sharing Using Blockchain Technology	Permissioned	Hyperledger Fabric	PBFT	World state database as actual data storage
2	Muhammad & Soewito (2022). A Blockchain For Secure Data Storing With Multi Chain On Smart Healthcare System	Consortium	Hyperledger Fabric	PBFT	Use of multi-channel between healthcare and financial industry
3	Tith et al.	Consortium	Hyperledger	PBFT	Storage of

	(2020). Patient Consent Management by a Purpose-Based Consent Model for Electronic Health Record Based on Blockchain Technology		Fabric		EHR in off-chain database linked with blockchain
4	Khatoun, A. (2020). A Blockchain-Based Smart Contract System for Healthcare Management	Private	Ethereum	PoW	Use of Gas to validate transaction
5	Antwi, M. et al. (2021). The case of HyperLedger Fabric as a blockchain solution for healthcare applications	Private	Hyperledger Fabric	PBFT	GDPR compliance
6	Tanwar, Parekh & Evans (2020). Blockchain-based electronic healthcare record system for healthcare 4.0	Permissioned	Hyperledger Fabric & Composer	PBFT	Design access control policy algorithm with smart contract

	applications				
7	Ismail & Materwala (2020). Blockchain Paradigm for Healthcare: Performance Evaluation	Permissioned	Not stated	PBFT	Validation of transaction done by doctors and pharmacists
8	Ndzimakhwe, M., Telukdarie, A., Munien, I., Vermeulen, A., K., U., & Philbin, S. P. (2023). A Framework for User-Focused Electronic Health Record System Leveraging Hyperledger Fabric	Private	Hyperledger Fabric	PBFT	Latest implementation where all hospitals are connected to a single channel

2.5 Proposed Solution

The Related Works have provided some limitation and gap that helps to propose some method that is able to improve the application of blockchain-based data management in electronic health record (EHR). Based on the proposed solution by Usman, M., & Qamar, U. (2020) and Ndzimakhwe, M., Telukdarie, A., Munien, I., Vermeulen, A., K., U., & Philbin, S. P. (2023), the private blockchain network will be organized by government department which is Ministry of Health. While the

department has full access to all users and system resources, there are also Health Administration which assigned to each hospital will be responsible for the registration of patients and doctors. Their responsibility is to monitor and allow only registered medical practitioners and public to enter the network. Within the application framework, there are components such as Membership Management, user interfaces facilitating user interactions, consensus nodes responsible for smart contract execution and consensus mechanisms, and databases for both the Chain (transaction history) and World-State (current state of the blockchain).

The application focuses on three types of users which are Patients, Healthcare Providers and Health Administration. Prior to granting access to the system, it is mandatory for all users to undergo a registration process which will be handled by health administration. In this case, the administrator will utilize the user interface to input patient details and sends the transaction to the Membership Service Provider (MSP), which hosts the Certificate Authority (CA). The CA issues a Certificate and Private Key for the user, which are then transmitted to the client application. The membership service also hosts a certification authority responsible for generating a key pair for signing and an encryption key pair for each user. The client application generates a private/public key pair and a symmetric patient key, which are then provided to the patient. Upon registration in the blockchain network, the user is furnished with login credentials, namely a User ID and Secret Key, which they can utilize for system access. While the process flow for Healthcare Providers follows a similar pattern, they are granted an ID and private key to gain entry to the system.

There will be different hospitals in a channel involved in the blockchain networks while the roles of patient is mainly the same as previous studies such as patient will be able to control their information and decide who can access or modify their information. Medical practitioners who have granted access can only view or modify specific patient's details while patient will have the right to revoke his access. These processes are done through a patient is provided with a symmetric encryption key, known as the Patient Key, which serves the purpose of encrypting and decrypting medical records. In situations where a patient intends to share their medical records with a Healthcare Provider, they have the ability to share their patient key by utilizing the public key of the specific Healthcare Provider.

Furthermore, the Healthcare Provider has the option to request the patient's key, and once it is provided, they can gain access to the patient's medical records and add new entries. The Healthcare Provider can log into the system to review patients' information. However, they can only view the patient's previous medical records after receiving authorization from the patient. Additionally, the Healthcare Provider has the ability to add new health records for the patient.

Besides, based on one of the function proposed by Khatoon A. (2020), our proposed solution has filling the gap by adding the role of emergency medical practitioners which they should have the right to access patient's medical records even when the patient is not available or conscious to give access right so that customized treatment can be provided as soon as possible. This process can be implemented using smart contract to streamline the transaction process.

As there is no cryptocurrency involved, the platform that will be used to develop the blockchain network is Hyperledger Fabric where PBFT consensus algorithm is utilized to ensure a secure, transparent and effective electronic health record data management. In the system, a network consists of three peer nodes that serve as both endorsing peers and committing peers is established. Additionally, there is one orderer node responsible for providing the ordering service. For a transaction to be successfully added to the blockchain, a consensus must be reached by at least two out of the three peers. Each peer node holds a ledger and the chaincode, which is written in JavaScript, along with its corresponding World-State database. Transactions submitted by users are received by the nodes through role-based APIs. When a user submits a transaction, the leader node organizes the transaction into a block and initiates the consensus mechanism. All nodes execute the transaction based on the implemented chaincode logic. Upon successful execution, the endorsing peers send their endorsement responses to the client. The client then sends the transaction, along with the endorsement responses, to the orderer node, which hosts the ordering service. The ordering service receives the endorsed transactions and arranges them into a block. Subsequently, it broadcasts the generated block to all peers in the network. Each peer verifies that the transactions within the received block are signed by the appropriate endorsers and that there are

sufficient endorsements. If the verification check passes, the peer proceeds to commit/save the block to its ledger.

The data within the system is stored in the distributed ledger of Hyperledger Fabric, utilizing two storage methods. Firstly, the blockchain maintains a chain of blocks, with each block containing transaction information in the form of key-value pairs. Secondly, the World-State database holds the values (assets) of all the most recent committed transactions, organized by specific keys. In our system, CouchDB is employed as the World-State database. Every peer within the network retains a copy of the ledger, encompassing both the blockchain and the World-State database.

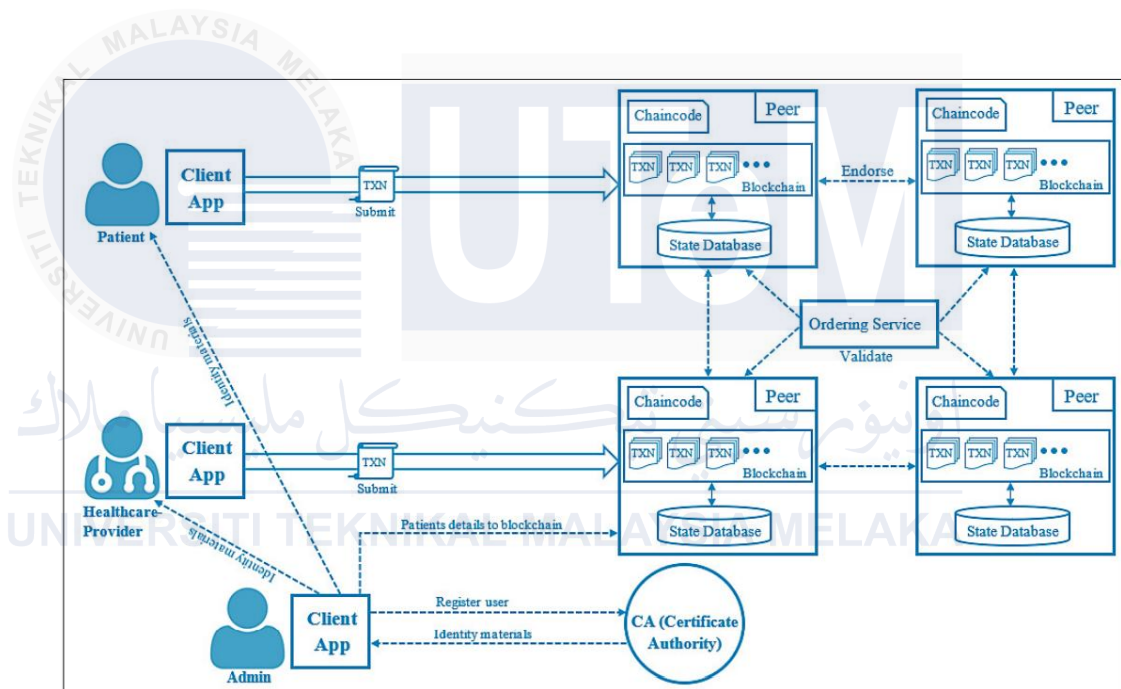


Figure 2.5: Architecture of proposed system from previous study

(Usman & Qamar, 2020.)

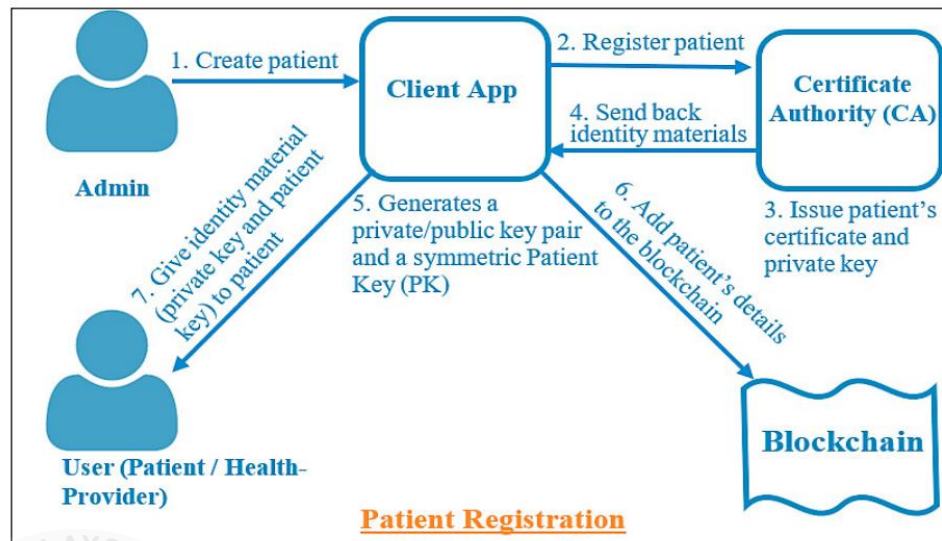


Figure 2.6: Process flow of patient registration

(Usman & Qamar, 2020.)

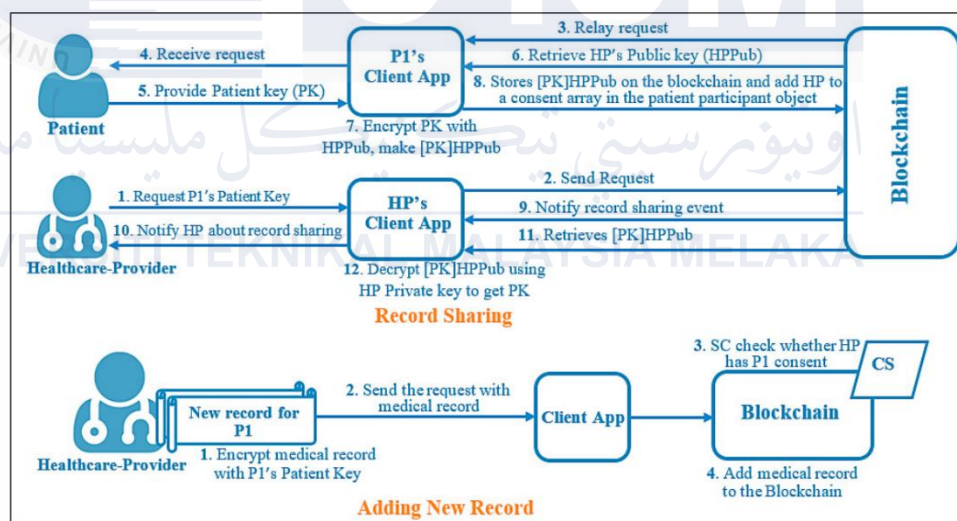


Figure 2.7: Process flow of records sharing and adding new record

(Usman & Qamar, 2020.)

2.6 Conclusion

To summarize, this chapter has covered introduction about blockchain, its platform and highlighted a variety of previous research on approaches or applications that have been proposed by different authors. The following chapter will discuss the methodology of the study used in this project.

CHAPTER 3: PROJECT METHODOLOGY

3.1 Introduction

This chapter will discuss the methodology that will be used to complete this project. This methodology will serve as a guide for creating the project so that it develops according to the time frame.

3.2 Methodology

The project methodology for creating a prototype of blockchain-based web application for electronic health record (EHR) data management among hospitals involves a series of stages including planning, design, implementation, testing, evolution and maintenance. This project methodology combines elements of the Software Development Life Cycle (SDLC) methodology with a focus on blockchain-specific perspectives and relevant phases necessary for development.

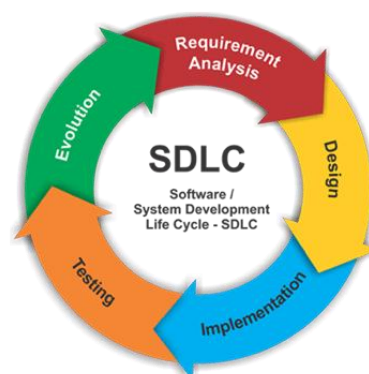


Figure 3.1: SDLC phases

(Sami, 2012.)

3.2.1 Planning

In this stage, the requirements and project scope for the development of blockchain-based web application for electronic health record (EHR) data management among hospitals are identified. This includes identifying the parties involved in the data management process, determining the features and functionalities of the smart contract and deciding on the data and transactions that need to be recorded.

3.2.2 Design

This stage involves outlining and specifying the protocol or set of rules for the blockchain application's architecture and interfaces. The mapping of the data flow and transaction are streamlined using smart contracts.

3.2.3 Implementation

Implementation is the stage where we develop and code the EHR data management system using the selected blockchain platform and programming language while adhering to best practices such as secure coding and testing methods.

Besides, this stage also implements proper data validation measures to prevent data tampering and unauthorized access.

3.2.4 Testing

This stage involves testing the blockchain-based web application for electronic health record (EHR) data management in various scenarios to ensure it works as intended. Besides, we will conduct proper validation of the data that comes into the system and check for any vulnerabilities.

3.2.5 Evolution

In this stage, maintenance and improvement to the blockchain network or web application code are done to ensure its performance is managed and the requirements are met. In addition, this stage also will address any issues that may arise and add updates to the contract as necessary.

3.3 Project Milestones

Project milestones is used to plan and track the progress of ongoing project. It is to ensure that the project progress follows the origins life cycle, so that it would not be contradictory with the project's objectives at the first place. Thus, the table shown below is the milestone and the Gantt chart for this project.

Table 3.1: Final year project 1 milestones

WEEK	ACTIVITY	NOTE/ACTION
W1 (20/03→24/03)	<ul style="list-style-type: none"> · Select a suitable project topic and potential supervisor. · Proposal PSM: Discussion with supervisor. · Proposal assessment and verification. 	<ul style="list-style-type: none"> · Title is chosen. · Proposal Form – Ulearn. · Deliverable – Draft Proposal Form – email PIC
W2 (27/03→31/03)	<ul style="list-style-type: none"> · List of students with project title versus supervisor and evaluator. · Proposal correction/improvement. · Proposal approval and submission 	<ul style="list-style-type: none"> · Email Committee for proposal approval. · Upload approved proposal at Ulearn
W3 (03/04→07/04)	Chapter 1 <ul style="list-style-type: none"> · Meeting 2 	
W4 (10/04→14/04)	Chapter 1 <ul style="list-style-type: none"> · Report Writing Progress 1 	<ul style="list-style-type: none"> · Log progress – ePSM. · Deliverable – Chapter 1 – ePSM.
W5 (17/04→21/04)	Chapter 2	

W6 (24/04→28/04)	MID-SEMESTER BREAK	
W7 (01/05→05/05)	Chapter 2 · Report Writing Progress · Project Progress 1	· Log progress – ePSM. · Deliverable – Chapter 2 – ePSM. · Progress Presentation 1 (KP1)
W8 (08/05→12/05)	Chapter 3	
W9 (15/05→19/05)	Chapter 3 · Report Writing Progress	· Log progress – ePSM · Deliverable – Chapter 2 – ePSM
W10 (22/05→26/05)	Chapter 4 · Project Progress 2	· Log progress – ePSM. · Progress Presentation 2 (KP2)
W11 (29/05→02/06)	Chapter 4 · Report Writing Progress 2	· Log progress – ePSM Deliverable – Chapter 2 – ePSM
W12 & W13 (05/06→16/06)	· PSM1 Draft Report preparation	
W14 (19/06→23/06)	· PSM1 Draft Report submission to SV & Evaluator · Report Evaluation	· Log Progress – ePSM · Deliverable – Complete PSM1 Draft Report – ePSM
W15 (26/06→30/06)	· PSM 1 Demo and Report Presentation to Supervisor & Evaluator · Presentation Skill · Submission of PSM 1 documents to PSM	· Log Record – ePSM · Submission of logbook in ePSM · Submission of Project Report PSM 1 to ePSM.

	supervisor, evaluator and committee in ePSM	
--	---	--

Table 3.2: Final year project gantt chart

Progress	Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FYP Proposal		■	■													
Project Progress 1				■	■	■	■	■								
Report Writing Progress 1				■	■	■	■	■								
Project Progress 2								■	■	■	■					
Report Writing Progress 2									■	■	■	■	■			
Report Evaluation														■	■	■
Demonstration															■	■
Presentation															■	■

3.4 Conclusion

This section offers a brief overview of the stage involved throughout the project and shows the project milestone to ensure the project management process schedules. The further explanation regarding the design of the project will be clarified in the next section.

CHAPTER 4: DESIGN

4.1 Introduction

This chapter begins with a comprehensive description of the requirements that have been gathered throughout the analytical process. The next part of the chapter focuses on high-level design. The high-level design emphasizes the overall system design, including the system architecture design. These designs will be extensively explained and documented in this chapter.

4.2 Problem Analysis

Currently, health service data is spread over various systems that have different architectures. There are also many problems of falsifying reports and withholding important information from patients, which are considered medical fraud. In traditional healthcare systems, patients often face cumbersome manual approval processes when sharing their data with other parties like hospitals or research institutes. These processes can be inefficient and challenging to coordinate, particularly in cases where patients relocate or seek treatment in different locations.

Therefore, the use of blockchain technology provides patients with comprehensive, immutable records and easier access to EHR free from service providers or treatment websites. By leveraging the capabilities of Hyperledger Fabric blockchain and various tools, such as Docker, npm and Ubuntu, the project aims to address the limitations of the current system, such as manual enforcement, lack of transparency, security vulnerabilities and challenges in interoperability among hospitals. The goal is to decentralized data management of electronic health record

among hospitals by automating the transaction process thus ensure transparency and integrity through blockchain technology.

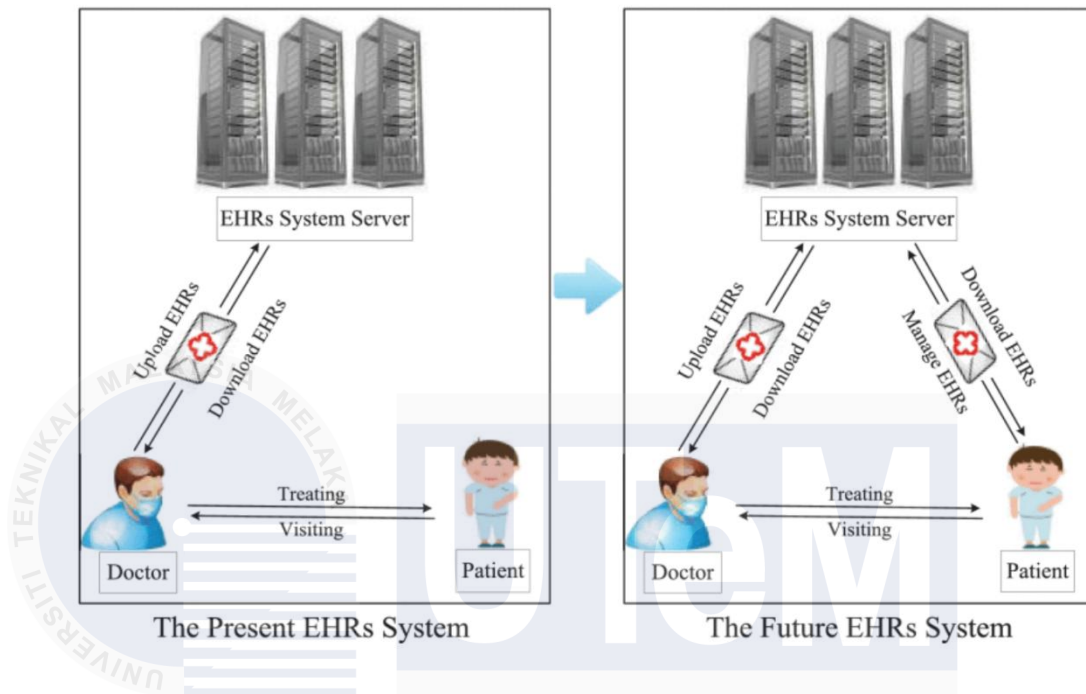


Figure 4.1: Comparison of present and future EHR system

(Guo, Shi, Zhao, & Zheng, 2018.)

4.3 Requirement Analysis

Requirement analysis is used to find and analyze any task or requirement that going to use to make this project run in successfully. This chapter will explain more about the functional requirement and other requirements.

4.3.1 Data Requirement

In this data model:

Inputs:

- Personal details (Name, age, weight, height, health record, diagnosis etc)

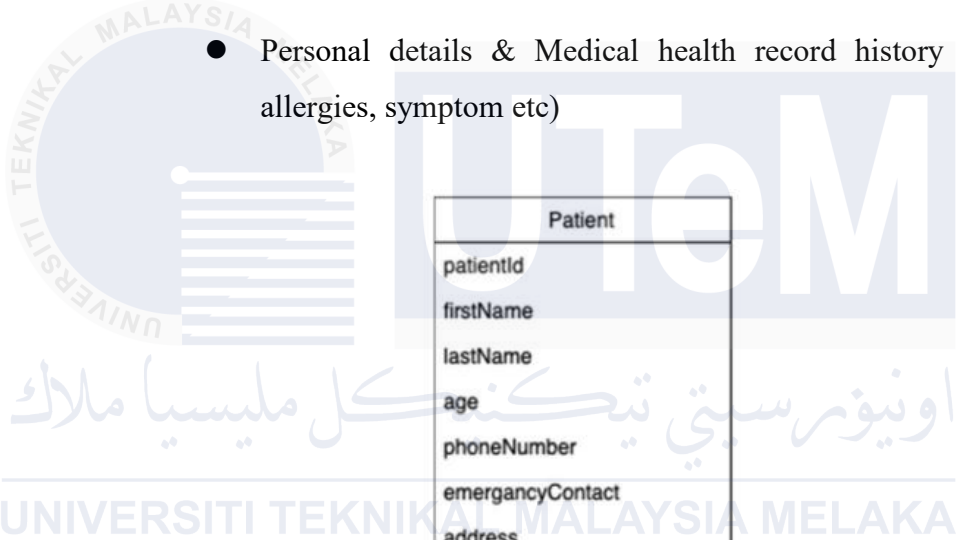
- Username, password, role
- Grant/Revoke permission

Outputs:

- Personal details (Name, age, weight, height, health record, diagnosis etc)

Data storage:

- Personal details & Medical health record history (Blood type, allergies, symptom etc)



Patient
patientId
firstName
lastName
age
phoneNumber
emergencyContact
address
bloodType
allergies
diagnosis
symptoms
treatment
followUp
permissionGranted
modifyBy
password

Figure 4.2: Data type of patient

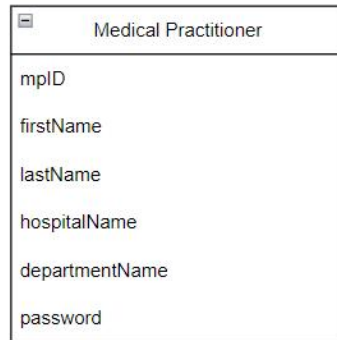


Figure 4.3: Data type of medical practitioner

4.3.2 Functional Requirement

Use case of actors within the system are shown as below:

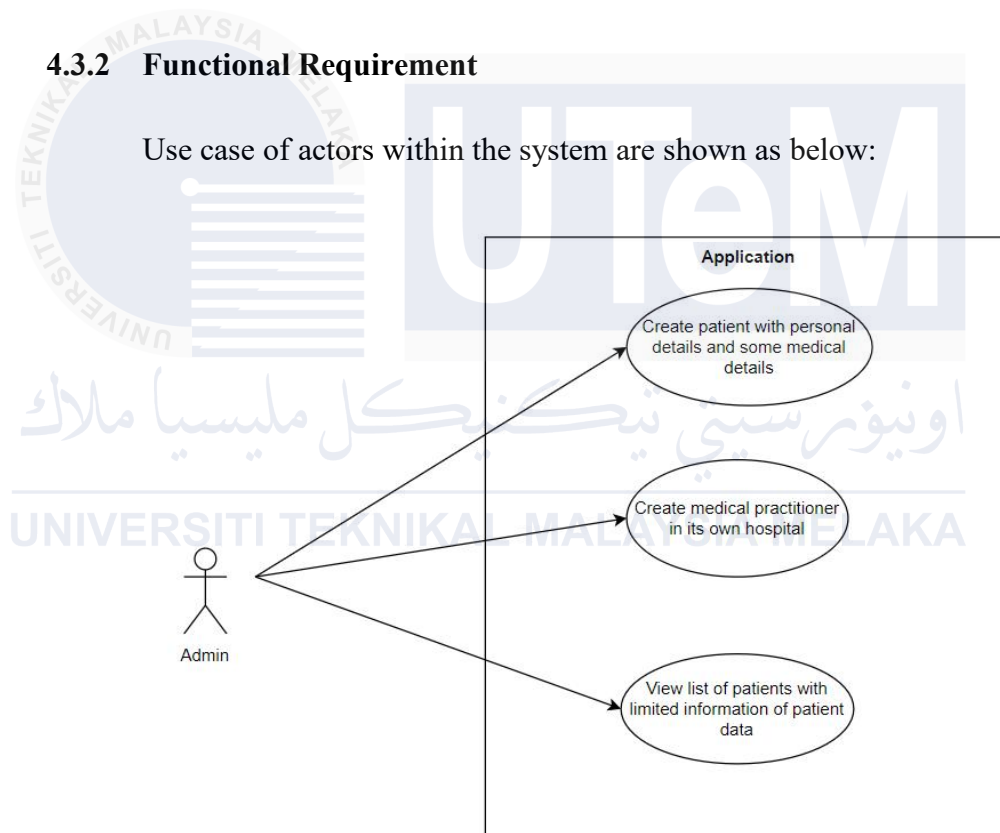


Figure 4.4: Use case of admin

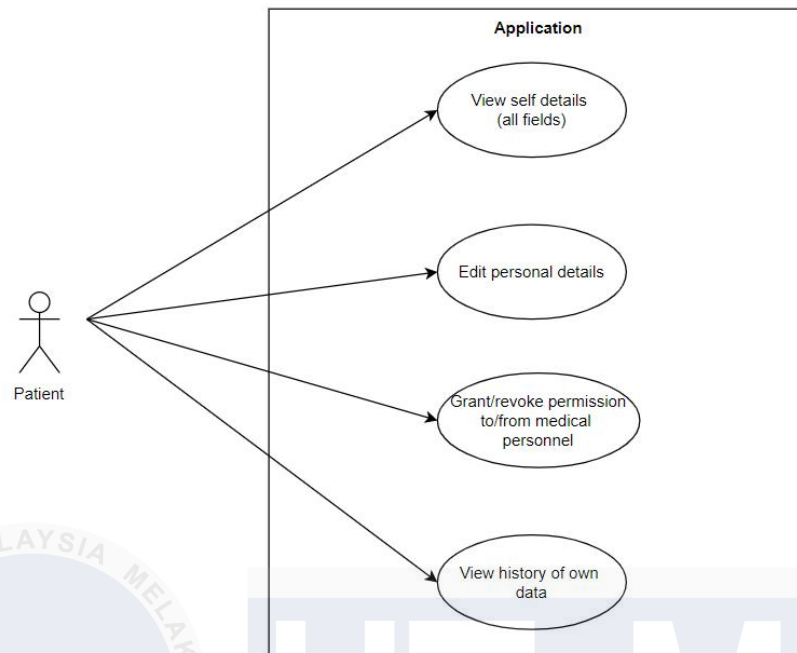


Figure 4.5: Use case of patient

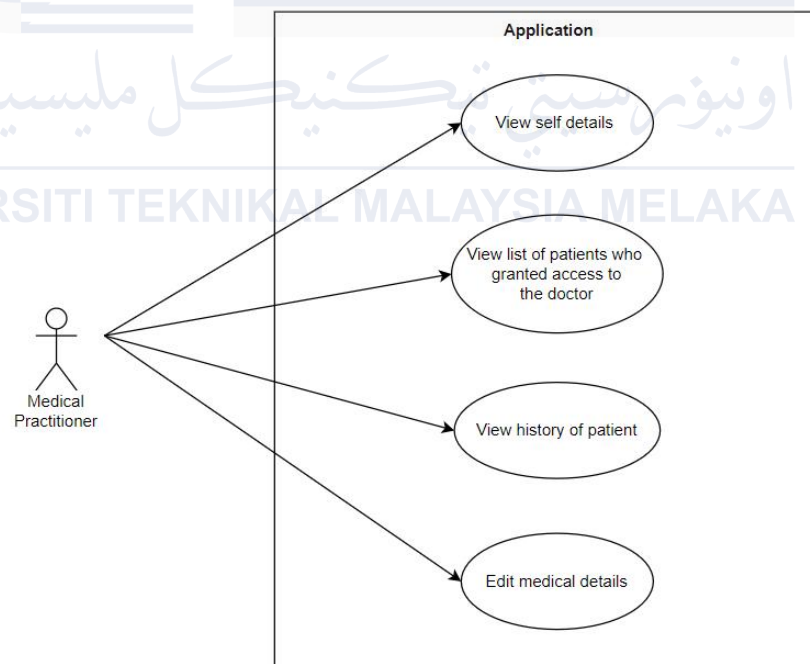


Figure 4.6: Use case of medical practitioner

Table below shows the participants involved in the blockchain network and their respective permission:

Table 4.1: Participants permission in the network

Role	Permission
Hospital admin	<ul style="list-style-type: none"> - Allow only registered medical practitioner to enter the network. - Allow public (patient) to enter the network. - Can view list of patients with limited information
Medical practitioner (Doctor)	<ul style="list-style-type: none"> - Can view self details - Can view list of patients who gave access permission to the doctor - Can read or update patient information with patient grant.
Patient	<ul style="list-style-type: none"> - Can create, read and update their own personal information - Grant access rights to medical practitioners - Remove permissions from medical practitioners - Can view history of own data

Activity diagram of actors within the system are shown as below:

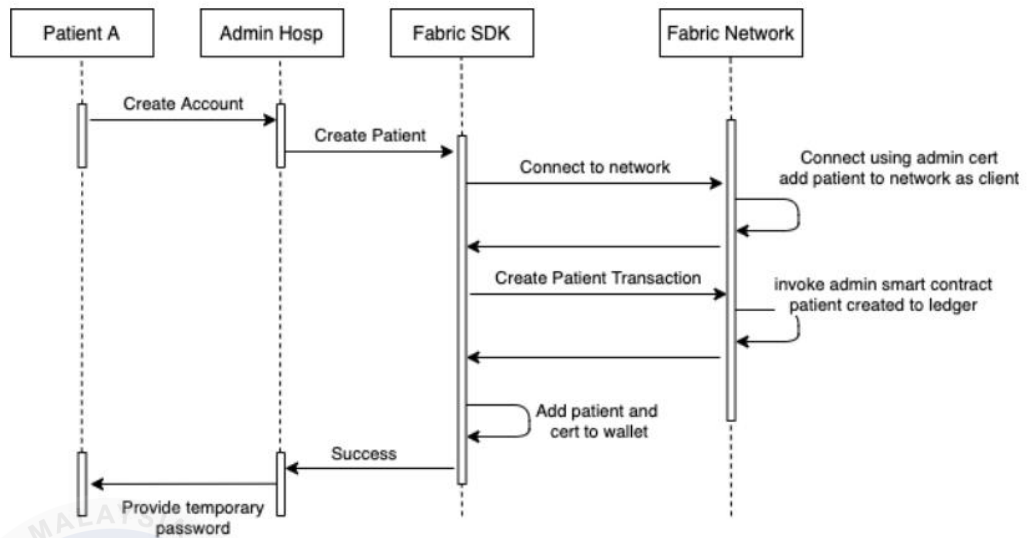


Figure 4.7: Sequence diagram of creation of a patient

The patient is required to establish an account only during their initial visit to any of the hospitals within the network. During this first visit, the patient provides their details to the administrator, who then invokes the AdminContract to create a patient. In the backend, the admin certificate is used to establish a connection to the network. A transaction is generated that incorporates the patient's information into the ledger and adds their identity to the blockchain network. Upon successful creation of the patient's account, the server generates a temporary password for the patient, which enables them to log into the network. The patient's credentials are recorded in the ledger, allowing them to visit any hospital within the network.

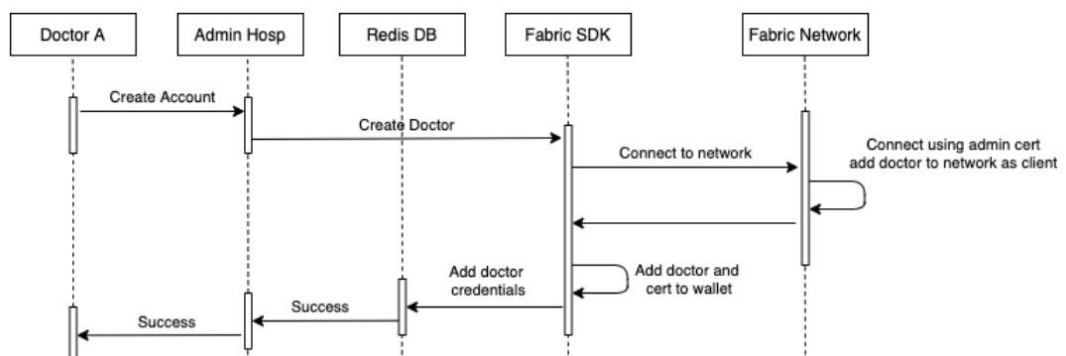


Figure 4.8: Sequence diagram of creation of a doctor

The doctor will provide their information to the admin, who proceeds to establish a connection with the network and adds the doctor as an identity within the blockchain network. The credentials of the doctor are stored in a Redis database specific to the hospital. Now that the patient and doctor have been added as identities in the blockchain network, they can subsequently connect to the network using their certificates for any future interactions.

4.3.2.1 Smart Contract/Chaincode

In Hyperledger Fabric, a smart contract is responsible for governing the transaction logic that manages the life cycle of a business object stored in the world state. Multiple smart contracts are bundled together into a chaincode, which is subsequently deployed onto a blockchain network. Smart contracts establish the rules and agreements between different organizations using executable code. The Fabric SDK is used to invoke smart contracts and initiate transactions, which then make modifications to the ledger. A chaincode encompasses one or more smart contracts, and when deployed, all the smart contracts within it become accessible to applications.

The proposed application includes mainly three smart contracts packaged into a single chaincode where each role (Patient/Medical Practitioner/Admin) invokes its very own smart contract:

Table 4.2: Smart contracts in the system

Smart Contracts	Descriptions
AdminContract	This contract is invoked by the administrator, who can utilize the contract's methods to create or delete patients or medical practitioner by adding or removing objects from the ledger. Additionally, the administrator has the capability to view all the patients across the entire network with limited information.
PatientContract	The patient actively engages with the ledger through this contract as it encompasses the necessary logic specifically tailored for the patient's operations. For

	example, the patient can exclusively update or view their personal details using the methods outlined within the contract. Moreover, the patient contract incorporates methods to grant or revoke access to doctors.
MPCContract	The doctor contract has methods that allows the medical practitioner to update or read the patients medical details based on the conditions set such as when normal condition there will need to have permission from patient while medical practitioner under emergency department will be able to view patient data directly.

4.3.3 Software Requirement

Software that is required for this project to complete the flow are as listed as follow:

Table 4.3: Software requirement

Software/Tool	Descriptions
Hyperledger Fabric	It is an open-source blockchain platform designed for developing decentralized applications. It provides a modular architecture, enabling customizable consensus protocols, privacy features and scalable network design, also, easy for building smart contract in the blockchain.
Hyperledger Fabric Client SDK	It provides APIs to interact with the Hyperledger Fabric blockchain, for instance, provides APIs to interact with smart contracts, submit transactions to a ledger and query the ledger.
Docker Compose	This tool is used to deliver software packages called containers such as docker-compose-ca.yaml, docker-compose-couch.yaml and docker-compose-net.yaml.
Couch DB	Couch DB is an open-source database which allows

	the storage of data in JSON format and is used as an external state of a database for Hyperledger fabric.
Node JS	This is an open-source cross-platform backend that runs scripts in the terminal, enabling the execution of JavaScript code outside of the browser. Node.js is employed to create APIs that interact with the Hyperledger Fabric blockchain. It handles the initial level of user authentication and serves as the gateway to the Fabric smart contracts.
Angular JavaScript	This framework is used to build the client application web interface.
Redis	Redis is an open source in-memory data structure store, used as a distributed, in-memory key–value database.
cURL	cURL is a software project that offers a command-line tool and library, allowing for data transfer across different network protocols.
Ubuntu	Ubuntu is a full Linux operating system, which is freely available for both technical and community support. In general, the software required can run smoothly in Ubuntu compare to windows.
VMware	VMware is a virtualization and cloud computing software that provides solution for building and managing virtualized environments. Its robust platform offers developers the ability to create and deploy blockchain projects with enhanced scalability, security and performance.
Visual Studio Code	Visual Studio Code is a code editor with support for development operations like debugging, task running and version control.

4.3.4 Hardware Requirement

To ensure smooth development and testing processes, it is recommended to have a system that meets the following minimum hardware (laptop/PC) specifications:

Table 4.4: Hardware requirement

Specification	Details
Processor	2 GHz dual core x86_64 CPU supporting SSE4.2 and above and virtualization
RAM	At least 4 GB or above
Storage	At least 50GB HDD or above
Operating System	Ubuntu Linux 14.04 / 16.04 LTS (both 64-bit), or Mac OS 10.12

4.4 High-Level Design

4.4.1 System Architecture

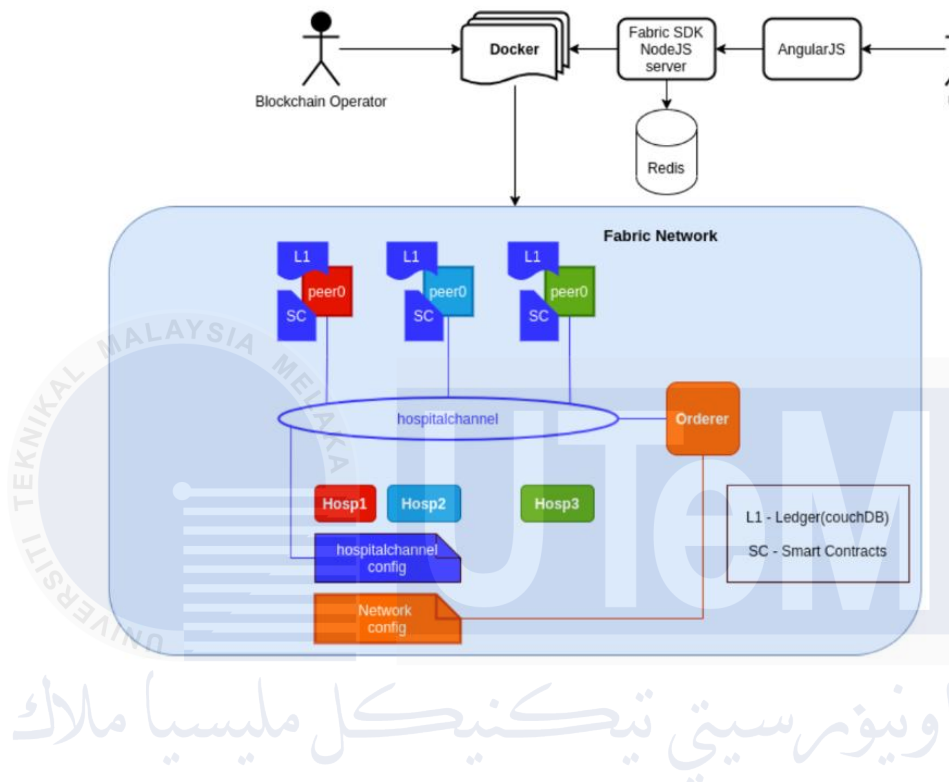


Figure 4.9: System architecture of the blockchain.

The system architecture begins with the blockchain operator bears the responsibility of initializing the network's configuration and granting access and credentials to users who manage the system. The orderer certificate authority (CA) is established within the docker fabric image and utilized by all hospital peers. The orderer assumes an administrative role by approving organizations and validating the credentials of their respective peers. All components of the application are interconnected to ensure seamless communication. The backend code and smart contract logic are implemented using JavaScript, with ExpressJS serving as the REST API server. The user interface is built with the Angular 11 framework, providing a user-friendly experience. Communication between the frontend and backend occurs through REST API calls, with authentication facilitated by JSON web tokens. Backend code can also be written in Java, Go, and Typescript which is officially supported languages by Hyperledger Fabric.

For the system design, a single channel named "hospitalChannel" is created with initial two hospital organizations. Plus, the network can seamlessly incorporate another new hospital organization even when the network is running as the new organization can join into the existing channel. In this scenario, third hospital organization should be able to join the network when the network is running and get to join with the same channel.

Hyperledger Fabric offers support for two databases, LevelDB and CouchDB. In this solution, CouchDB is chosen due to its flexibility and ability to handle image data. CouchDB also supports indexes, unlike LevelDB. Since all patient data is stored in CouchDB without maintaining a separate Electronic Health Record (EHR) store, CouchDB fits well in this context. On the other hand, LevelDB, developed by Google, is a powerful in-memory database designed to store key-value pairs. It can be faster than CouchDB in some cases. In an alternative use case scenario where the EHR database of hospitals is employed, LevelDB can be utilized to store references (such as APIs) to the records in the blockchain ledger. In the current simple use cases, CouchDB provides appropriate support. The ledger consists of the transaction log and the world state. CouchDB is used to store the world state, eliminating the need to query the entire transaction log for each transaction request. The transaction log retains all transactions starting from the first one stored in the genesis block. Additionally, the Redis key-value database is utilized to store doctor credentials, specifically usernames and passwords while other details of the doctor are stored as user attributes using the fabric SDK.

4.5 Conclusion

To summarize, project design plays a critical role in project development. It is essential to define and evaluate all software and hardware requirements before commencing the design implementation. This phase serves as a preparation stage for the application, enabling a comprehensive understanding of the entire system before implementation. Additionally, this chapter outlines the crucial design requirements that will be implemented and tested in the subsequent chapter. It also presents the planned architecture design for the project.

CHAPTER 5: IMPLEMENTATION

5.1 Introduction

This chapter explains about the implementation of prototype of blockchain-based web application for EHR data management among hospitals that has been described in previous chapter. The implementation phase including software development environment setup, configuration of smart contracts, application APIs and configuration on application.

5.2 Software Configuration Management

According to documentation in Hyperledger Fabric, the Fabric application stack has five layers which are:

- (a) *Prerequisite software*: The base layer needed to run the software, for example, Docker.
- (b) *Fabric and Fabric samples*: The Fabric executables to run a Fabric network along with sample code.
- (c) *Contract APIs*: To develop smart contracts executed on a Fabric Network.
- (d) *Application APIs*: To develop blockchain application.
- (e) *The Application*: Blockchain application that will utilize the Application SDKs to call smart contracts running on a Fabric network.

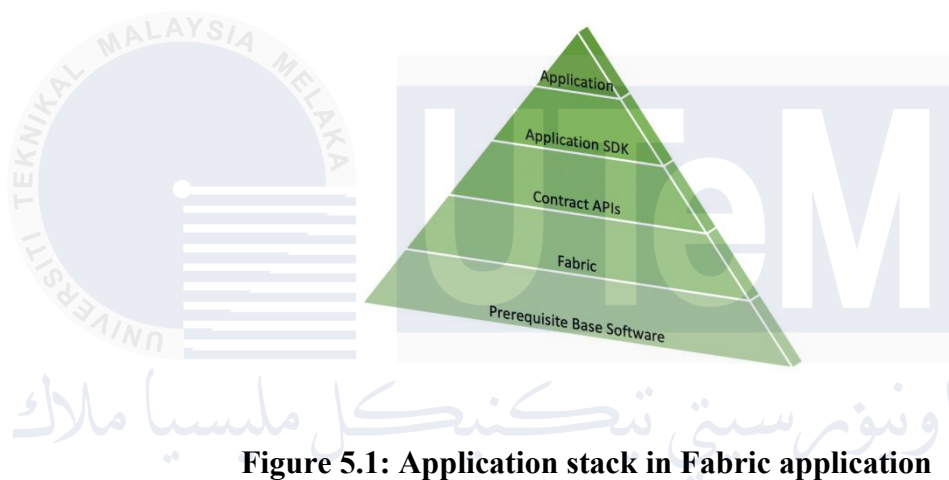


Figure 5.1: Application stack in Fabric application

Therefore, the implementation steps of this project will followed according to the stacks mentioned above from bottom-top in order to develop a blockchain-based web application for EHR data management among hospitals.

5.3 Prerequisite Base Software

As this project is planned to develop in a controlled environment, operating system, Ubuntu 22.04.3 LTS is used and setup in VMware. Within the system, prerequisite tools according to Hyperledger Fabric documentation are installed which are git, curl, docker, docker compose, go, python, nodejs, npm and angular. It is important to note that the version of node should be 10.15.3 which npm that included in it is 6.4.1 and the frontend framework is angular 11. Besides, Microsoft Visual Studio Code is also used to building and debugging application.

5.4 Fabric and Fabric Samples

After all prerequisite software are installed, Fabric basic framework is also downloaded which included Fabric Docker images, Fabric CLI tool binaries and Fabric samples which are test network created by HLF community using Docker compose. To test all components are working, test network from fabric samples can be run.

5.5 Contract APIs

Contracts serve as the implementation of all executable business logic within the application, with smart contracts executing actions like creating, reading, updating, or deleting assets on the distributed ledger. In this specific instance, smart contracts are written in JavaScript which one function has been written for each capability of the proposed system.

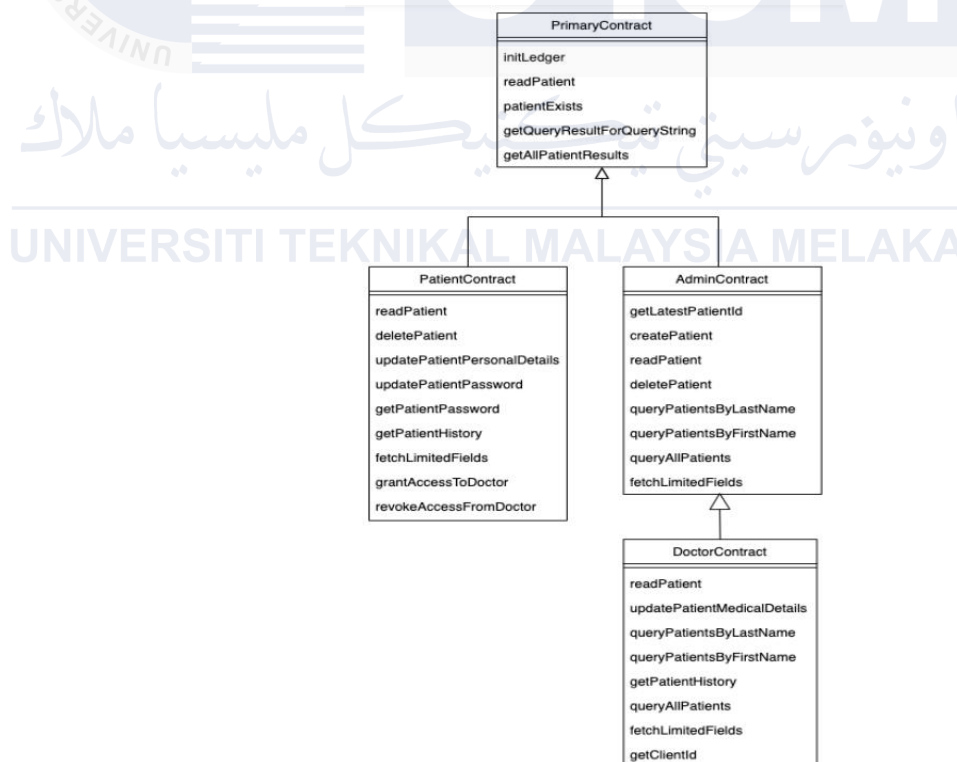


Figure 5.2: Smart contracts hierarchy

The primary focus of smart contract development is typically the entity on which network transactions are expected to occur. In this project, EHR take center

stage, and smart contracts are designed around them. For example, the `createPatient()` contract facilitates the creation of a new EHR when an admin registers a new patient, while contracts like `updatePatientPersonalDetails()` and `updatePatientMedicalDetails()` handle updates to patient personal and medical information. When access to a latest EHR is requested, `readPatient()` contracts are invoked. Retrieving an EHR's transaction history is accomplished using the `getPatientHistory()` contract. The HLF blockchain network offers a useful function which is `getHistory`, enabling users to access the transaction history of a particular entity. This is advantageous because the global state keeps track of the most recent record state while the history feature allows for tracing earlier transactions.

```

100 //Retrieves patient medical history based on patientId
101 async getPatientHistory(ctx, patientId) {
102     let resultsIterator = await ctx.stub.getHistoryForKey(patientId);
103     let asset = await this.getAllPatientResults(resultsIterator, true);
104
105     return this.fetchLimitedFields(asset, true);
106 }

```

Figure 5.3: Example of `getPatientHistory()` methods in doctor contract

A fundamental aspect of the Patient-centered Data Management System (PDMS) involves granting and revoking access. This is achieved through the `grantAccessToDoctor()` and `revokeAccessFromDoctor()` methods, enabling patients to manage a doctor's access to their electronic health record. Access control is implemented in smart contracts to regulate data access at the time of retrieval.

```

187 async revokeAccessFromDoctor(ctx, args) {
188     args = JSON.parse(args);
189     let patientId = args.patientId;
190     let doctorId = args.doctorId;
191
192     // Get the patient asset from world state
193     const patient = await this.readPatient(ctx, patientId);
194     // Remove the doctor if existing
195     if (patient.permissionGranted.includes(doctorId)) {
196         patient.permissionGranted = patient.permissionGranted.filter(doctor => doctor !== doctorId);
197         patient.changedBy = patientId;
198     }
199     const buffer = Buffer.from(JSON.stringify(patient));
200     // Update the ledger with updated permissionGranted
201     await ctx.stub.putState(patientId, buffer);
202 }

```

Figure 5.4: Example of `revokeAccessFromDoctor()` method

A `permissionGranted` array, which contains doctor IDs permitted to access the EHR, is maintained within the EHR. Patients can add or remove a doctor's ID from this list which is a process hidden from the doctor and only accessible to the patient. When a doctor interacts with a patient's EHR, the system checks the doctor's user ID against the `permissionGranted` array. If the ID is absent, access is denied, thus ensuring data privacy.

```

27     // Check if doctor has the permission to read the patient
28     const permissionArray = asset.permissionGranted;
29     if(!permissionArray.includes(doctorId)) {
30         throw new Error(`The doctor ${doctorId} does not have permission to patient ${patientId}`);
31     }
32     asset = ({
33         patientId: patientId,
34         firstName: asset.firstName,
35         lastName: asset.lastName,
36         age: asset.age,
37         bloodGroup: asset.bloodGroup,
38         allergies: asset.allergies,
39         symptoms: asset.symptoms,
40         diagnosis: asset.diagnosis,
41         treatment: asset.treatment,
42         followUp: asset.followUp
43     });
44     return asset;
45 }

```

Figure 5.5: Example of verifying if the doctor is granted access

There are four processes involved in deploying chaincode which are packaging, peer installation, approval for a channel and commitment. In Fabric, smart contracts are deployed on the network in packages referred to as chaincode. A chaincode is installed on the peers of an organization and then deployed to a channel, where it can then be used to endorse transactions and interact with the blockchain ledger. Before a chaincode can be deployed to a channel, the members of the channel need to agree on a chaincode definition that establishes chaincode governance. When the required number of organizations agree, the chaincode definition can be committed to the channel and then the chaincode is ready to be used. All of these processes can be streamlined using the `deployCC` command once the HLF network is operational.

5.6 Application SDK

5.6.1 Hyperledger Fabric Client SDK

The Hyperledger Fabric Client SDK offers APIs to interact with the HLF blockchain such as interact with smart contracts, submit transactions to a ledger and query the ledger. The Fabric SDK provides the following packages:

- (a) *fabric-ca-client*: The fabric-ca provides APIs to participants (admin, patient and doctor) to register and enroll for establishment of trusted identities on the blockchain network.
- (b) *fabric-common*: This component consolidates the common code utilized across all fabric-sdk-node packages, facilitating precise interactions with the Fabric network for transaction invocations. It offers APIs for monitoring events, logging and configuring settings, allowing customization via environment variables, program arguments and in-memory configuration.
- (c) *fabric-network*: This package contains the APIs required to connect to the Fabric network, submit transactions to query or edit the ledger. It also provides APIs to manage the wallet which is used for managing identities and create a connection profile based on the connection profile JSON generated when CA is created. In this package, the main class that allows the Fabric SDK to interact with the network is the Gateway class. Once instantiated, the object create a gateway (connection) to a peer (user) within the blockchain network and enables access to the chaincode and channels for which that peer is a member.

```

10  const {Wallets} = require('fabric-network');
11  const FabricCAServices = require('fabric-ca-client');
12  const path = require('path');
13  const {buildCAClient, enrollAdmin} = require('../patient-asset-transfer/application-javascript/CAUtil.js');
14  const {buildCCHosp2, buildWallet} = require('../patient-asset-transfer/application-javascript/AppUtil.js');
15  const adminHospital2 = 'hosp2admin';
16  const adminHospital2Passwd = 'hosp2lithium';

```

Figure 5.6: Example of utilizing fabric-network between backend server and blockchain network

5.6.2 Wallet

The wallet plays a crucial role in the Hyperledger Fabric SDK, serving as an identity repository. It stores essential Fabric metadata, including authorized private keys and corresponding public keys issued by a certificate authority. Wallets can be configured in multiple formats, such as file-based, in-memory and database-based wallets. In the context of developing a patient data management system, the file-based wallet is employed. When establishing a connection through the Gateway class, the wallet stores both the mspID and user type, which are then used to verify the user's access rights to the specific channel within the network.

5.6.3 JSON Web Tokens

JSON Web Tokens (JWTs) are employed to manage API permissions and maintain user sessions. This approach ensures that the individual who initially logged in also has access to the API. Instead of the server retaining session information, a token is generated upon the user's login using their username and password. Subsequently, this token is signed, encrypted, transmitted to the client and stored there. When the user attempts to access the API with the same token, JWT verifies the token using a key (the user's password). If the token is unaltered and the user remains the same, access to the API is granted. To prioritize Hyperledger Fabric over application development, user login credentials are stored as JSON in a file and verified against it.

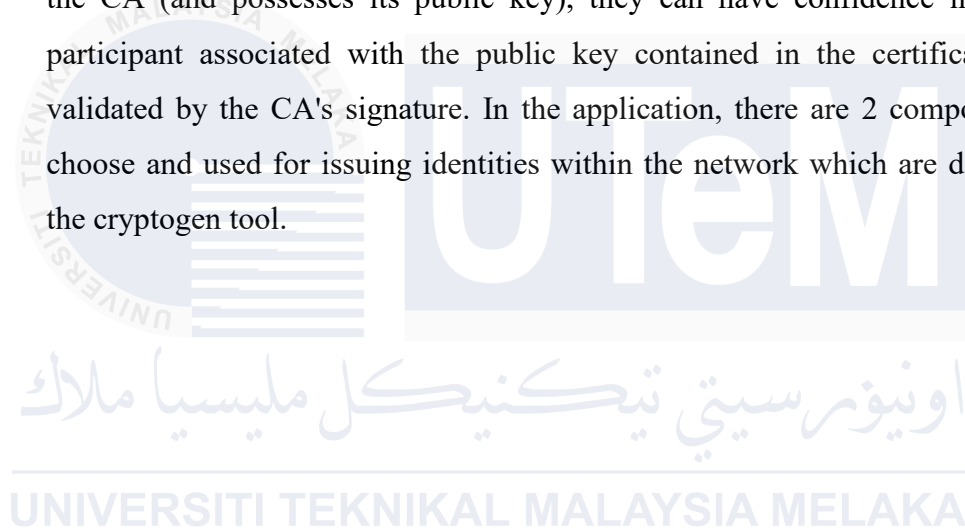
5.7 Application

5.7.1 State of Distributed Database

In this project, CouchDB is employed as a database on a peer node to represent each patient's health records in JSON data structures. The number of CouchDB Docker images depends on the number of peers within the network and it operates on the identical server as the respective peer. Since each peer has its ledger, an HLF network requires a single CouchDB image for each peer. In this case, 2 CouchDB images are required as there are 2 peer nodes in the network.

5.7.2 Identity (CA)

To become a recognized participant within the hospital network, individuals must first establish their trustworthiness to conduct transactions on the blockchain. This involves obtaining an identity issued by a trusted authority, which is Membership Service Providers (MSPs), as indicated in the `app/first-network/configtx/configtx.yaml` configuration file. Each hospital operates its Certificate Authority (CA), responsible for issuing certificates to its participants. These certificates are digitally signed by the CA and serve to bind a participant with their public key, along with a set of permissions. Consequently, if one places trust in the CA (and possesses its public key), they can have confidence in the specific participant associated with the public key contained in the certificate which all validated by the CA's signature. In the application, there are 2 components can be choose and used for issuing identities within the network which are default CAs or the cryptogen tool.



- (a) *Cryptogen*: Cryptogen is a tool that creates certificates and keys. This tool can be used during development and testing. The tool quickly just creates the required crypto material for the hospitals. The configuration files for all the hospitals and the orderer are defined in `app/first-network/organizations/cryptogen` directory, the YAML files contain the necessary information for the tool to create the crypto material for the hospitals and the Orderer.
- (b) *CAs*: Each hospital has its very own CA server that creates the identities using client of the server to prove that the identity belong to their hospital and can be trusted. All of the identities created by a CA run in the hospital share the same trust of the root CA. In this project, CAs are used to allow registration and enrollment of patients and doctors with Fabric SDK which cannot be done using Cryptogen. The configuration files of the CAs of the hospitals and orderer are defined in `app/first-network/organizations/fabric-ca`.

5.7.3 Membership Service Provider (MSP)

As stated in previous chapter, MSP is the one that verifies the private keys of the participants by matching them with the stored public key. While CAs are used to create trusted identities that are recognized by the network (peers/doctors/patients), MSPs are instrumental in defining the hospitals that enjoy trust among network members. MSPs allocate roles and permissions to participants from these hospitals, allowing them to engage within the network. In the application setup, all hospitals that become part of the network are empowered to carry out read and write transactions. These permissions are configured in the `app/first-network/configtx/configtx.yaml` file. However, to initiate a transaction within the network, participants must first obtain an identity issued by a CA that is acknowledged by the network. Furthermore, they must be the member in one of the hospitals.

5.7.4 Endorsement Policies

The chaincodes in Hyperledger fabric have an endorsement policy that specifies the peers on the channel that executes the chaincode functions and endorses

the results to the ledger and makes the transaction valid. The endorsement policies define the peers which will verify and approve/reject the execution of a transaction. During this process, the peers verify the transaction, and the committing peer ensures that the transaction contains the the required number of endorsements which is configured in the endorsement policy.

The Endorsement Policy is defined in the configtx.yaml. Each hospital contains its very own endorsing peers as specified in the YAML file in the path `&hosp1/Policies/Endorsement` similarly for hosp2. The rule specifies the roles who are endorsers. In the YAML file, all the peers of the hospitals are endorsers. In path `&hosp1/Policies/Endorsement`, the rule `OR('hosp1MSP.peer')` requests one signature from the peers of hosp1. A Chaincode-level endorsement policy is defined in the YAML file in the path `&ApplicationDefaults/Policies/Endorsement`, where “MAJORITY” specifies that only when a majority of channel members approve a chaincode definition then definition is committed to the channel.

```

- &hosp1
  # DefaultOrg defines the organization which is used in the sampleconfig
  # of the fabric.git development environment
  Name: hosp1MSP

  # ID to load the MSP definition as
  ID: hosp1MSP

  MSPDir: ../organizations/peerOrganizations/hosp1.lithium.com/msp

  # Policies defines the set of policies at this level of the config tree
  # For organization policies, their canonical path is usually
  # /Channel/<Application|Orderer>/<OrgName>/<PolicyName>
  Policies:
    Readers:
      Type: Signature
      Rule: "OR('hosp1MSP.admin', 'hosp1MSP.peer', 'hosp1MSP.client')"
    Writers:
      Type: Signature
      Rule: "OR('hosp1MSP.admin', 'hosp1MSP.client')"
    Admins:
      Type: Signature
      Rule: "OR('hosp1MSP.admin')"
    Endorsement:
      Type: Signature
      Rule: "OR('hosp1MSP.peer')"

```

Figure 5.7: Endorsement policy for Hosp 1


```

Application: &ApplicationDefaults

# Organizations is the list of orgs which are defined as participants on
# the application side of the network
Organizations:

# Policies defines the set of policies at this level of the config tree
# For Application policies, their canonical path is
# /Channel/Application/<PolicyName>
Policies:
  Readers:
    Type: ImplicitMeta
    Rule: "ANY Readers"
  Writers:
    Type: ImplicitMeta
    Rule: "ANY Writers"
  Admins:
    Type: ImplicitMeta
    Rule: "MAJORITY Admins"
  LifecycleEndorsement:
    Type: ImplicitMeta
    Rule: "MAJORITY Endorsement"
  Endorsement:
    Type: ImplicitMeta
    Rule: "MAJORITY Endorsement"

```

Figure 5.8: Chaincode-level endorsement policy

5.7.5 Security Mechanism

Security of the data is essential when question arises for patient data in healthcare sector. Therefore, encryption capabilities in HLF are enable which uses Transport Layer Security (TLS) 1.2/1.3 (RSA TLS) for certificates generation to securely work in the network and also SHA256 for hashing. These TLS certificates will be used by the peer when communicating with the network and hashes of passwords making it more difficult to be break, thus prevent any security risks.

```

61  tls:
62    # Enable TLS (default: false)
63    enabled: true
64    # TLS for the server's listening port
65    certfile:
66    keyfile:
67    clientauth:
68      type: noclientcert
69    certfiles:
70

```

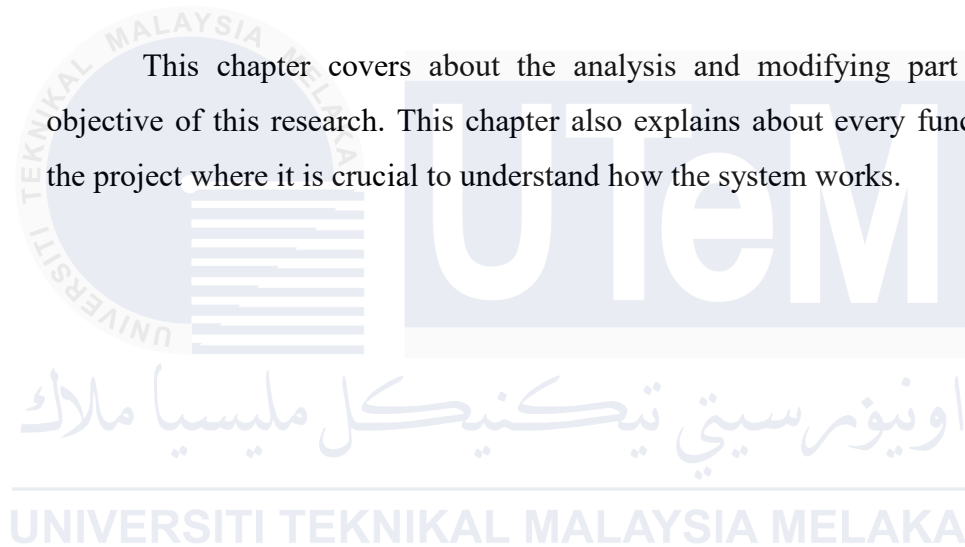
Figure 5.9: TLS enabled in CA Hosp 1

```
class Patient {  
  constructor(patientId, firstName, lastName, password, age, phoneNumber, emergPhoneNumber, address, bloodGroup,  
    changedBy = '', allergies = '', symptoms = '', diagnosis = '', treatment = '', followUp = '')  
  {  
    this.patientId = patientId;  
    this.firstName = firstName;  
    this.lastName = lastName;  
    this.password = crypto.createHash('sha256').update(password).digest('hex');  
    this.age = age;  
    this.phoneNumber = phoneNumber;  
    this.emergPhoneNumber = emergPhoneNumber;  
  }  
}
```

Figure 5.10: Patient's passwords are hashed and stored in the blockchain (patient chaincode)

5.8 Conclusion

This chapter covers about the analysis and modifying part to reach the objective of this research. This chapter also explains about every function used for the project where it is crucial to understand how the system works.



CHAPTER 6: TESTING

6.1 Introduction

This chapter will explain about the testing and evaluation after the blockchain-based web application for EHR data management system is created. This chapter is crucial to measure the performance of the system and discuss about the capability of blockchain-based EHR data management system in ensuring data confidentiality, integrity and transparency.

6.2 Test Strategy

The testing strategy of this project combines both top-down and bottom-up approaches. Top-down approach helps to define the overall system architecture and functionality of the blockchain-based EHR system while bottom-up approach helps to test individual components and smart contracts. In terms of testing types, both black-box and white-box tests are employed. Black-box tests involved functional tests while white-box tests includes security tests and data correctness checking.

6.2.1 Classes of Tests

As the evaluation of the prototype system aims to ensure data security in terms of confidentiality, integrity and availability, several tests can be conducted such as output correctness which provide data integrity, functionality test which evaluate system's functionality and security test which can analyze the system's internal logic and address any potential vulnerabilities.

6.3 Test Design

6.3.1 Test Description

Each test has objectives which outline the purpose of the test while a test description outlining the methodology. Expected result for each module in the prototype is designed and the actual result should be the same as the expected result in order to pass the test. There are two types of tests which are manual tests that are executed by an individual and automated tests that are executed via scripts. In this project, manual tests are conducted as it is a more appropriate approach to assess specific features within the prototype such as participant's access control.

Interactive tests will be used to check the validation, verification, permissions and the overall performance of the prototype of blockchain-based EHR system. All the test description listed below will be used to evaluate the fault tolerance and efficiency of the system.

Table 6.1: Test Description

No.	Test Description
1	Only admin in hospitals can login as admin.
2	Admin can only view list of patients with limited personal information.
3	Only admin can create patient and doctor to join the network.
4	Only registered patient can login as patient.
5	First-time login patient always need to change password.
6	Only patient himself can view all his personal information and EHR.
7	Only the EHR owner can view list of doctors and grant or revoke access rights to doctors.
8	Patient can only update his own personal information.
9	Patient can only view his own data history.
10	Only registered doctor in hospitals can login as doctor.
11	Only doctor can view his own details and specialty.
12	Only doctor can view list of patients who granted access and the patient's EHR.
13	Only doctor who granted access can update patient's EHR.

14	Only doctor who granted access can view patient's EHR history.
15	Data within the system is always correct.
16	HTTPS should be used to secure the channel between the client and blockchain.

6.3.2 Test Data

Table and figures below show the data involved and stored within the blockchain-based EHR system which can be used for testing. The EHR data of patients shown below are initial data which will directly created within blockchain when the network is up. The code is included in app/patient-asset-transfer/chaincode/lib/initLedger.json.

Table 6.2: Login details for users

Role	Username	Password
Admin	hosp1admin	hosp1lithium
Admin	hosp2admin	hosp2lithium
Patient	PID0	PID0
Patient	PID1	PID1
Patient	PID2	PID2
Patient	PID3	PID3
Patient	PID4	PID4
Patient	PID5	PID5
Doctor	HOSP1-DOC0	password
Doctor	HOSP1-DOC2	password
Doctor	HOSP2-DOC1	password
Doctor	HOSP2-DOC3	password

```

"firstName": "Monica",
"lastName": "Latte",
"age": "50",
"phoneNumber": "+4912345678",
"emergPhoneNumber": "+4912345678",
"address": "Albrechtstrasse 71, 86383 Stadtbergen",
"bloodGroup": "O+",
"allergies": "No",
"symptoms": "Cholesterol, Total 250 mg/dl",
"diagnosis": "High Cholesterol",
"treatment": "Vasolip 10 mg everyday",
"followUp": "6 Months",
"permissionGranted": ["hosp1admin", "hosp2admin"],
"changedBy": "initLedger",
"password": "ee6affdb08c3ef9c6444816329b56250ae42a2149b3e72136251a7d8340de43b",
"pwdTemp": true

```

Figure 6.1: EHR of patient 0

```

"firstName": "Max",
"lastName": "Mustermann",
"age": "60",
"phoneNumber": "+491764561111",
"emergPhoneNumber": "+491764561113",
"address": "Mainzer landstrasse 134, 60326 Frankfurt am Main",
"bloodGroup": "B+",
"allergies": "No",
"symptoms": "Heart Burn, shortness of breath, Acidity",
"diagnosis": "Esophagitis",
"treatment": "omeprazole 40 mg for 10 days before food",
"followUp": "2 Weeks",
"permissionGranted": ["hosp1admin", "hosp2admin"],
"changedBy": "initLedger",
"password": "fd1e5d642da134218f1cc5c260feb3e4042658a35d78c4eb2fe2b2f1c7e7795b",
"pwdTemp": true

```

Figure 6.2: EHR of patient 1

```

"firstName": "Johannes",
"lastName": "Schmidt",
"age": "63",
"phoneNumber": "+491764561111",
"emergPhoneNumber": "+491764561113",
"address": "Genslerstraße 19, 60326 Berlin",
"bloodGroup": "B+",
"allergies": "No",
"symptoms": "Dizziness, Nausea, systolic-150, diastolic-110",
"diagnosis": "Hypertension",
"treatment": "CORBIS 5 mg one per day",
"followUp": "2 Weeks",
"permissionGranted": ["hosp1admin", "hosp2admin"],
"changedBy": "initLedger",
"password": "8c90267256e654ee3f77196ee12e952dc45cd970d030bfa069614cc2d923313c",
"pwdTemp": true

```

Figure 6.3: EHR of patient 2

```

"firstName": "Torben",
"lastName": "Klaproth",
"age": "75",
"phoneNumber": "+491764561111",
"emergPhoneNumber": "+491764561113",
"address": "Genslerstraße 19, 60326 Berlin",
"bloodGroup": "B+",
"allergies": "No",
"symptoms": "Weight Loss, frequent urination, dizziness",
"diagnosis": "Diabetes Mellitus",
"treatment": "PRINIVIL TABS 20 MG (LISINOPRIL), HUMULIN INJ 70/30 20 units after breakfast",
"followUp": "4 Weeks",
"permissionGranted": ["hospladmin", "hosp2admin"],
"changedBy": "initLedger",
"password": "31bb1dcc31812dac9deb87c014fc8e5e46013a982c2a724447d5c444aac713a7",
"pwdTemp": false

```

Figure 6.4: EHR of patient 3



```

"firstName": "Lisa",
"lastName": "Eckel",
"age": "78",
"phoneNumber": "+491764561179",
"emergPhoneNumber": "+491764567913",
"address": "Genslerstraße 19, 60326 Berlin",
"bloodGroup": "B+",
"allergies": "No",
"symptoms": "Pain in the knee joints",
"diagnosis": "Osteoarthritis",
"treatment": "ULTRADAY 40 mg twice per day",
"followUp": "2 Weeks",
"permissionGranted": ["hospladmin", "hosp2admin"],
"changedBy": "initLedger",
"password": "eceb33ea781e143861a75ebc6462188a8428b8fab68d123cec31614f6b5a9b58",
"pwdTemp": false

```

Figure 6.5: EHR of patient 4

```

"firstName": "Harry",
"lastName": "Schumann",
"age": "72",
"phoneNumber": "+491764561156",
"emergPhoneNumber": "+491764589113",
"address": "Pappelallee 8, 98631 Behrungen",
"bloodGroup": "AB+",
"allergies": "No",
"symptoms": "Pain in the shoulder and difficulty in shoulder movement",
"diagnosis": "Periarthritis",
"treatment": "Hydrocortisone 20 ml injection",
"followUp": "4 Weeks",
"permissionGranted": ["hospladmin", "hosp2admin"],
"changedBy": "initLedger",
"password": "49bb4e28642c7eee1fdb6fc3d359d80c22b5a871c441be09c103538be167a281",
"pwdTemp": false

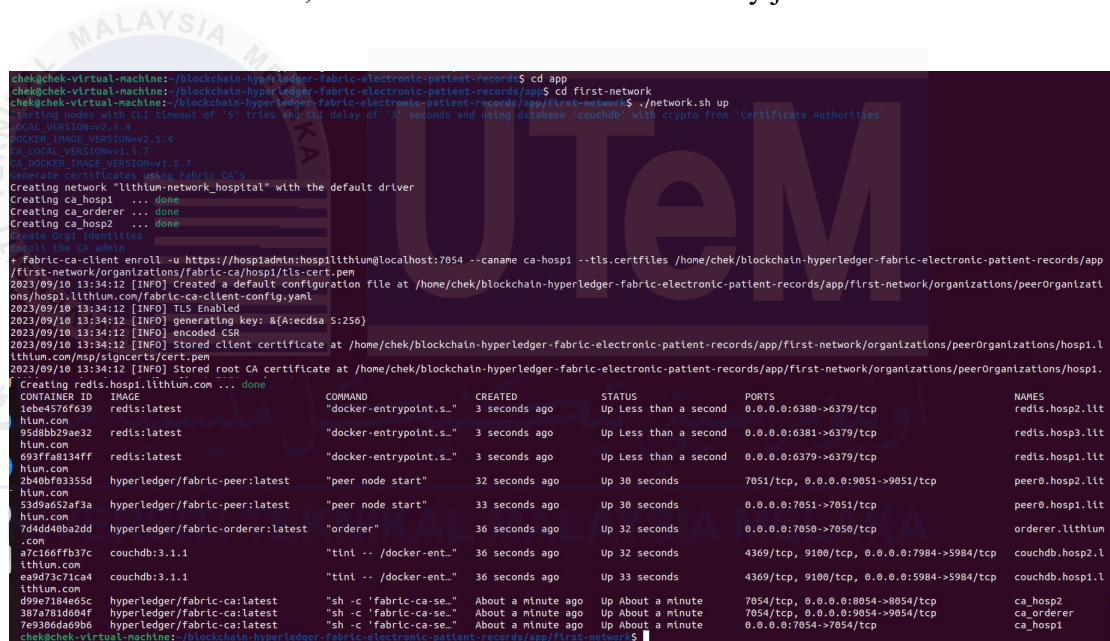
```

Figure 6.6: EHR of patient 5

6.4 Test Results and Analysis

6.4.1 Network Testing

In the `app/first-network` directory, command `./network.sh up` is executed to bring up the network. If there is no issue, a Fabric network that consists of two organizations which are Hosp 1 and Hosp 2 with one ordering node will be created. Docker containers are created too and will be running on the machine. With the command `./network.sh createChannel`, a channel called “hospitalChannel” will also be created between both organizations and join their peers to the channel. Once the command is successful, it will show “Channel successfully joined”.



```

che@cheek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records$ cd app
che@cheek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app$ cd first-network
che@cheek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app/first-network$ ./network.sh up
Starting nodes with CLI timeout of '5' tries and CLI delay of '5' seconds and using database 'couchdb' with crypto from 'Certificate Authorities'
DOCKER_IMAGE_VERSION=v2.5.4
CR_LOCAL_VERSION=v1.5.7
CA_DOCKER_IMAGE_VERSION=v1.5.7
Generating certificates using Fabric CA's
Creating network "lithium_network_hospital" with the default driver
Creating ca_hosp1 ... done
Creating ca_orderer ... done
Creating ca_hosp2 ... done
Start Org 1 as admin
+ fabric-ca-client enroll -u https://hosp1admin:hosp1lithium@localhost:7054 --caname ca-hosp1 --tls.certfiles /home/cheek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/fabric-ca/hosp1/tls-cert.pem
2023/09/10 13:34:12 [INFO] Created a default configuration file at /home/cheek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1.lithium.com/fabric-ca-client-config.yaml
2023/09/10 13:34:12 [INFO] TLS Enabled
2023/09/10 13:34:12 [INFO] generating key: &(A:ecdsa S:256)
2023/09/10 13:34:12 [INFO] encoded CSR
2023/09/10 13:34:12 [INFO] Stored client certificate at /home/cheek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1.lithium.com/msp/signcerts/cert.pem
2023/09/10 13:34:12 [INFO] Stored root CA certificate at /home/cheek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1.lithium.com/peers/hosp1.tls-cert.pem
Creating redis:hosp1.lithium.com ... done
CONTAINER ID        IMAGE               COMMAND                  CREATED              STATUS              PORTS              NAMES
1e8e4576f639       redis:latest       "docker-entrypoint.s..." 3 seconds ago        Up Less than a second    0.0.0.0:6380->6379/tcp    redis.hosp2.lit
hium.com
95d8bb29ae32       redis:latest       "docker-entrypoint.s..." 3 seconds ago        Up Less than a second    0.0.0.0:6381->6379/tcp    redis.hosp3.lit
hium.com
693ffab134ff       redis:latest       "docker-entrypoint.s..." 3 seconds ago        Up Less than a second    0.0.0.0:6379->6379/tcp    redis.hosp1.lit
hium.com
2b40bf03355d       hyperledger/fabric-peer:latest "peer node start"        32 seconds ago       Up 30 seconds        7051/tcp, 0.0.0.0:9051->9051/tcp    peer0.hosp2.lit
hium.com
53d9a652af3a       hyperledger/fabric-peer:latest "peer node start"        33 seconds ago       Up 30 seconds        0.0.0.0:7051->7051/tcp    peer0.hosp1.lit
hium.com
784dd40ba2dd       hyperledger/fabric-orderer:latest "orderer"                36 seconds ago       Up 32 seconds        0.0.0.0:7050->7050/tcp    orderer.lithium.com
a7c166ffb37c       couchdb:3.1.1      "tni -- /docker-ent..." 36 seconds ago       Up 32 seconds        4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp    couchdb.hosp2.1
lithium.com
e89873c71ca4       couchdb:3.1.1      "tni -- /docker-ent..." 36 seconds ago       Up 33 seconds        4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp    couchdb.hosp1.1
lithium.com
d99e7184e65c       hyperledger/fabric-ca:latest "sh -c 'fabric-ca-se..." About a minute ago    Up About a minute      7054/tcp, 0.0.0.0:8054->8054/tcp    ca_hosp2
387f783d084f       hyperledger/fabric-ca:latest "sh -c 'fabric-ca-se..." About a minute ago    Up About a minute      7054/tcp, 0.0.0.0:9054->9054/tcp    ca_orderer
7d9306da89b9       hyperledger/fabric-ca:latest "sh -c 'fabric-ca-se..." About a minute ago    Up About a minute      0.0.0.0:7054->7054/tcp    ca_hosp1
che@cheek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app/first-network$

```

Figure 6.7: Testing of bringing up network


```

chek@chek-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records$ cd app
chek@chek-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app$ cd first-network
chek@chek-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network$ ./network.sh up createChannel
Creating channel 'hospitalchannel'...
If network is not up, starting nodes with CLI timeout of '5' tries and CLI delay of '3' seconds and using database 'couchdb' with crypto from 'Certificate Authorities'
Bringing up network...
LOCAL_VERSION=2.2.2
DOCKER_IMAGE_VERSION=2.2.2
CA_LOCAL_VERSION=1.4.9
CA_DOCKER_IMAGE_VERSION=1.4.9
Generate certificates using Fabric CA's
Creating network 'lithium-network-hospital' with the default driver
Creating ca_hosp1 ... done
Creating ca_hosp2 ... done
Creating ca_orderer ... done
Create Org Identities
Done! The channel...
+ fabric-ca-client enroll -u https://hosp1admin:hosp1lithium@localhost:7054 --caname ca-hosp1 --tls.certfiles /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/fabric-ca/hosp1/tls-cert.pem
2023/09/10 17:22:59 [INFO] Created a default configuration file at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1lithium.com/fabric-ca-client-config.yaml
2023/09/10 17:22:59 [INFO] TLS Enabled
2023/09/10 17:22:59 [INFO] generating key: &{Aeccdsa S:256}
2023/09/10 17:22:59 [INFO] encoded CSR
2023/09/10 17:22:59 [INFO] Stored client certificate at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1lithium.com/msp/signcerts/cert.pem
2023/09/10 17:22:59 [INFO] Stored root CA certificate at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1lithium.com/msp/anchors/anchors.tls
2023-09-10 17:24:10.881 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:10.888 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:11.216 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:11.126 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:11.331 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:11.366 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:11.359 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:11.583 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:11.788 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:11.801 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:12.018 +08 [cli.common] readBlock -> INFO 010 Received block: 0
Channel 'hospitalchannel' created
Join Hospital 1 peers to the channel...
Using Hospital 1
+ peer channel join -b ./channel-artifacts/hospitalchannel.block
+ res=0
2023-09-10 17:24:15.226 +08 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2023-09-10 17:24:15.094 +08 [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
Join Hospital 2
Using Hospital 2
+ peer channel join -b ./channel-artifacts/hospitalchannel.block
+ res=0
2023-09-10 17:24:19.213 +08 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2023-09-10 17:24:19.068 +08 [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
Updating anchor peers for Hospital 1...
+ peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.lithium.com -c hospitalchannel -f ./channel-artifacts/hosp1MSpanchors.tx --tls --cafile /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/ordererOrganizations/lithium.com/orderers/orderer.lithium.com/msp/tlscacerts/tlsca.lithium.com-cert.pem
+ res=0
2023-09-10 17:24:22.755 +08 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2023-09-10 17:24:22.705 +08 [channelCmd] update -> INFO 002 Successfully submitted channel update
Anchor peers updated for org 'hosp1MSP' on channel 'hospitalchannel'
Updating anchor peers for Hospital 2...
Using Hospital 2
+ peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.lithium.com -c hospitalchannel -f ./channel-artifacts/hosp2MSpanchors.tx --tls --cafile /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/ordererOrganizations/lithium.com/orderers/orderer.lithium.com/msp/tlscacerts/tlsca.lithium.com-cert.pem
+ res=0
2023-09-10 17:24:28.075 +08 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2023-09-10 17:24:28.040 +08 [channelCmd] update -> INFO 002 Successfully submitted channel update
Anchor peers updated for org 'hosp2MSP' on channel 'hospitalchannel'
Channel successfully joined
chek@chek-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network$

```

Figure 6.8: Testing of creating channel

To test whether the chaincode is applicable, command `./network.sh deployCC` is executed. This command will package the smart contract code inside `app/patient-asset-transfer`, then install package to peer nodes, query installed chaincode, approve chaincode for both organizations and finally commit and invoke the chaincode. Once successful, the initial data of six patients are created in a ledger.

```

chek@chek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app/first-network$ ./network.sh deployCC
Deploying chaincode on channel 'hospitalchannel'
executing with the following
- CHANNEL_NAME: hospitalchannel
- CC_NAME: patient
- CC_SRC_PATH: NA
- CC_SRC_LANGUAGE: javascript
- CC_VERSION: 1.0
- CC_SEQUENCE: 1
- CC_END_POLICY: NA
- CC_COLL_CONFIG: -/private-collections/private-collections.json
- CC_INIT_FCN: InitLedger
- DELAY: 3
- MAX_RETRY: 5
- VERBOSE: false
- HOSPS: false
Determining the path to the chaincode
patient-asset-transfer
+ peer lfecycle chaincode package patient.tar.gz --path ../patient-asset-transfer/chaincode/ --lang node --label patient_1.0
+ res=0
Chaincode is packaged
Installing chaincode on peer0.hospi...
Using Hospital
+ peer lfecycle chaincode install patient.tar.gz
+ res=0
2023-09-11 02:04:27.207 +08 [ctl.lifecycle.chaincode] submitInstallProposal -> INFO 001 Installed remotely: response:status:200 payload:"\npatient_1.0:c7e485e1f90be749d9ed1e63eda48196936200c28e3dc27da6ff4efdf84c7f022f013patient_1.0" >
2023-09-11 02:04:27.207 +08 [ctl.lifecycle.chaincode] submitInstallProposal -> INFO 002 Chaincode code package identifier: patient_1.0:c7e485e1f90be749d9ed1e63eda48196936200c28e3dc
Chaincode is installed on peer0.hospi
Install chaincode on peer0.hosp2...
Installing chaincode on peer0.hosp2...
+ peer chaincode invoke -o localhost:7050 --ordererTLSHostnameOverride orderer.lithium.com --tls --cafile /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/ordererOrganizations/lithium.com/orderers/orderer.lithium.com/tlscaerts/tlsca.lithium.com-cert.pem -C hospitalchannel -n patient --peerAddresses localhost:7051 --tlsrootcertfiles /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hospi.lithium.com/peers/peer0.hospi.lithium.com/tls/ca.crt --peerAddresses localhost:9051 --tlsrootcertfiles /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp2.lithium.com/peers/peer0.hosp2.lithium.com/tls/ca.crt --isinit -c '{"Function":"InitLedger","Args":[]}'
+ res=0
2023-09-11 02:06:03.356 +08 [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001 Chaincode invoke successful. result: status:200
Invoke transaction successful on peer0.hospi peer0.hosp2 on channel 'hospitalchannel'
chek@chek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app/first-network$

```

Figure 6.9: Testing of deploying smart contract

In directory app/server, command `npm start` is executed to start the ExpressJS backend server. If it is successful, users from the network including admin from hospital organizations, initial six patients data and four doctors will be enrolled and registered which are then created inside the wallet and the backend server will be running.

```

chek@chek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/chaincode$ cd ../server
bash: cd: ../server: No such file or directory
chek@chek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/chaincode$ cd ..
chek@chek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer$ cd ..
chek@chek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app$ cd server
chek@chek-virtual-machine: /blockchain-hyperledger-fabric-electronic-patient-records/app/server$ npm start

> patient-application@1.0.0 prestart /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/server
> node initServer.js

Built a CA Client named ca-hospi
Successfully registered and enrolled user PID5 and imported it into the wallet
msg: Successfully enrolled user PID5 and imported it into the wallet
Done
Built a file system wallet at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/application-javascript/wallet
Loaded the network configuration located at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hospi.lithium.com/connection-hospi.json
Built a CA Client named ca-hospi
Successfully registered and enrolled user HOSPI-DOC0 and imported it into the wallet
msg: Successfully enrolled user HOSPI-DOC0 and imported it into the wallet
Built a file system wallet at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/application-javascript/wallet
Loaded the network configuration located at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp2.lithium.com/connection-hosp2.json
Built a CA Client named ca-hosp2
Successfully registered and enrolled user HOSPI-DOC1 and imported it into the wallet
msg: Successfully enrolled user HOSPI-DOC1 and imported it into the wallet
Built a file system wallet at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/application-javascript/wallet
Loaded the network configuration located at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hospi.lithium.com/connection-hospi.json
Built a CA Client named ca-hospi
Successfully registered and enrolled user HOSPI-DOC2 and imported it into the wallet
msg: Successfully enrolled user HOSPI-DOC2 and imported it into the wallet
Built a file system wallet at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/application-javascript/wallet
Loaded the network configuration located at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp2.lithium.com/connection-hosp2.json
Built a CA Client named ca-hosp2
Successfully registered and enrolled user HOSPI-DOC3 and imported it into the wallet
msg: Successfully enrolled user HOSPI-DOC3 and imported it into the wallet

> patient-application@1.0.0 start /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/server
> ./node_modules/nodemon/bin/nodemon.js src/server.js

[nodemon] 1.19.4
[nodemon] to restart at any time, enter 'rs'
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting node src/server.js
Backend server running on 3001

```

Figure 6.10: Testing of running backend server

To test frontend server, command `ng serve -o` is used to bring up the angular server. If there is no issue, a user interface will be automatically opened up in the default browser.

```

chek@chek-virtual-machine: ~/blockchain-hyperledger-fabric-electronic-patient-records/app/client$ ng serve -o
Your global Angular CLI version (11.2.19) is greater than your local version (11.0.5). The local Angular CLI version is used.

To disable this warning use "ng config -g cli.warnings.versionMismatch false".

Compiling @angular/core : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
Compiling @angular/common/http : es2015 as esm2015
Compiling @angular/forms : es2015 as esm2015
Compiling @ng-select/ng-select : es2015 as esm2015
Compiling @ng-bootstrap/ng-bootstrap : es2015 as esm2015
Compiling @angular/router : es2015 as esm2015
✓ Browser application bundle generation complete.

Initial Chunk Files | Names | Size
vendor.js | vendor | 3.86 MB
main.js | main | 255.59 kB
styles.css | styles | 218.71 kB
polyfills.js | polyfills | 141.34 kB
runtime.js | runtime | 6.15 kB
| Initial Total | 4.46 MB

Build at: 2023-09-11T04:37:30.269Z - Hash: 24876a4c809fe00937cf - Time: 20061ms

** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

✓ Compiled successfully.
✓ Browser application bundle generation complete.

Initial Chunk Files | Names | Size
styles.css | styles | 218.71 kB

4 unchanged chunks

Build at: 2023-09-11T04:37:38.176Z - Hash: c69d95f4032435ff3ae2 - Time: 7263ms

✓ Compiled successfully.

```

Figure 6.11: Testing of running frontend server

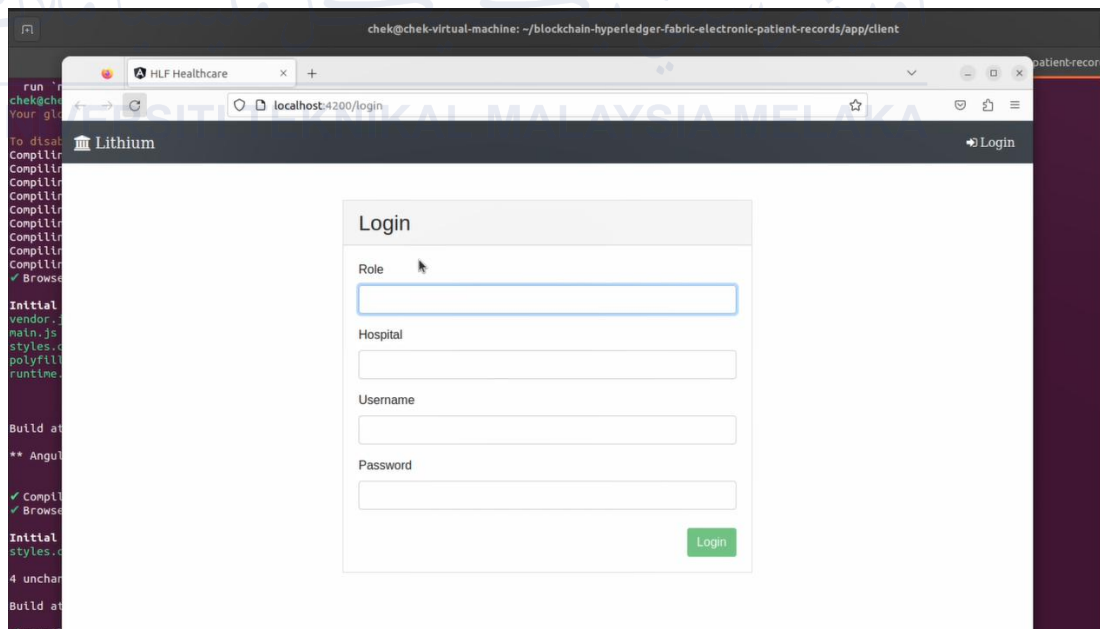


Figure 6.12: Main page of user interface

6.4.2 Functionality Testing

6.4.2.1 Admin

(a) Login

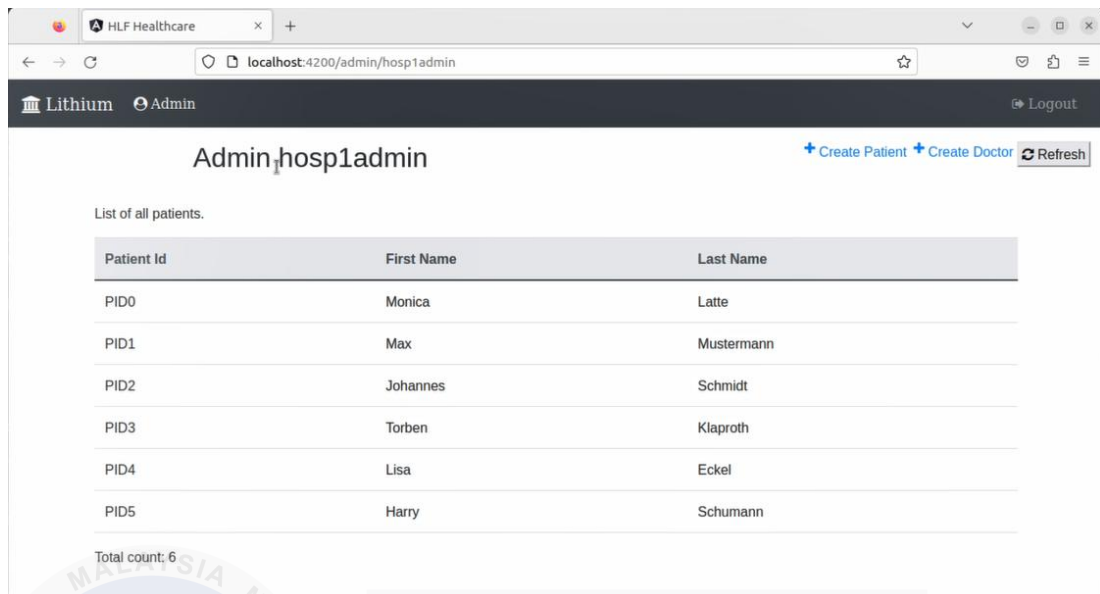
An admin is created for each hospital that joins the network. During the hospital's network enrollment process, the details of the administrator must be present. The administrator's credentials which are username and password, are specified in the fabric-ca-server-config.yaml file within the configuration of the respective hospital's CA (app/first-network/organizations/fabric-ca). For admin login, specific role and hospital should be chosen and the credentials entered are checked against the credentials stored in the blockchain network where only valid admin can log into the system.

The screenshot shows a web interface for an admin login. At the top, there is a dark navigation bar with the word 'Lithium' on the left and a 'Login' link on the right. Below this is a white login form with the title 'Login'. The form contains four input fields: 'Role' (a dropdown menu with 'Admin' selected), 'Hospital' (a dropdown menu with 'Hospital 1' selected), 'Username' (a text input field containing 'hosp1admin'), and 'Password' (a text input field with masked characters). A green 'Login' button is located at the bottom right of the form. The background of the page features a large, semi-transparent watermark of the 'UTEM' logo and the text 'UNIVERSITI TEKNIKAL MALAYSIA MELAKA'.

Figure 6.13: Admin login page

(b) View List of Patients

Once the admin login successfully, a list of patients with limited information (patient ID, first name and last name) within the blockchain network are displayed. The patient list is retrieved by invoking the admin contract which a new transaction that retrieves all patient objects in the world state is recorded in the ledger. It has also shown that the admin chaincode is functioning as it limits the access of admin only to the names while EHR of the patients are restricted.



The screenshot shows a web browser window with the URL `localhost:4200/admin/hosp1admin`. The page title is "Admin_hosp1admin" and it includes navigation links for "Create Patient", "Create Doctor", and "Refresh". Below the header, there is a section titled "List of all patients." containing a table with the following data:

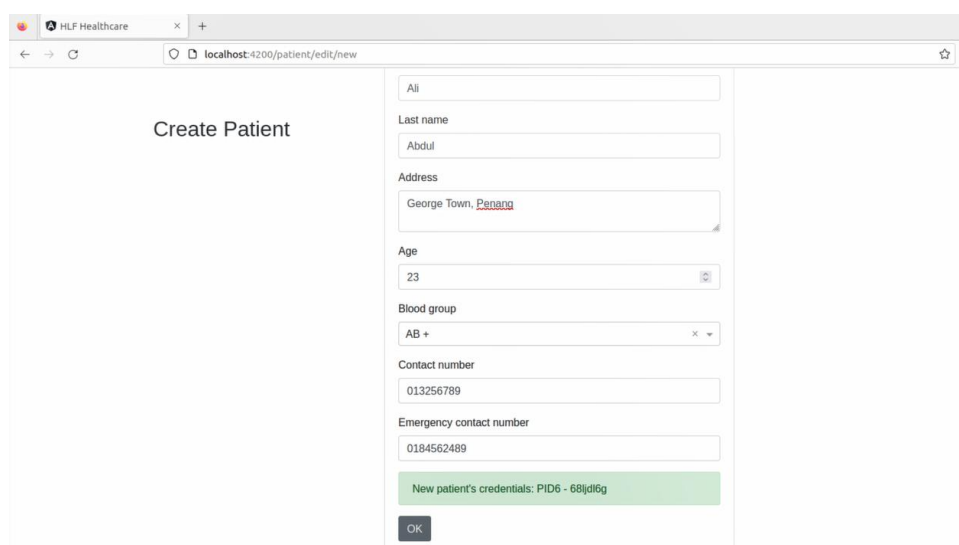
Patient Id	First Name	Last Name
PID0	Monica	Latte
PID1	Max	Mustermann
PID2	Johannes	Schmidt
PID3	Torben	Klaproth
PID4	Lisa	Eckel
PID5	Harry	Schumann

Below the table, it indicates "Total count: 6".

Figure 6.14: List of patients (admin)

(c) Create Patient

Admin has the capability to create a new patient. When creating a new patient, a transaction is generated in the ledger which is to create an object in world state then it will be sent to all network peers for verification, endorsed and stored in the ledger if valid. The patient is also registered within the network as a client so that interaction between ledger can be done. In addition, temporary passwords are created which the patient must change on the first-time login for security reasons.



The screenshot shows a web browser window with the URL `localhost:4200/patient/edit/new`. The page title is "Create Patient" and it contains a form with the following fields:

- First name:
- Last name:
- Address:
- Age:
- Blood group:
- Contact number:
- Emergency contact number:

Below the form, there is a green box displaying "New patient's credentials: PID6 - 68ljdi6g" and an "OK" button.

Figure 6.15: Create new patient

(d) Create Doctor

Admin can also create a new doctor, however, doctor is not an object stored in the ledger due to problem of HLF. Therefore, there is no relationship with the ledger upon registration of a doctor while doctor's credentials are stored in Redis database. A client is created in the network for the doctor so that he can communicate with the ledger.

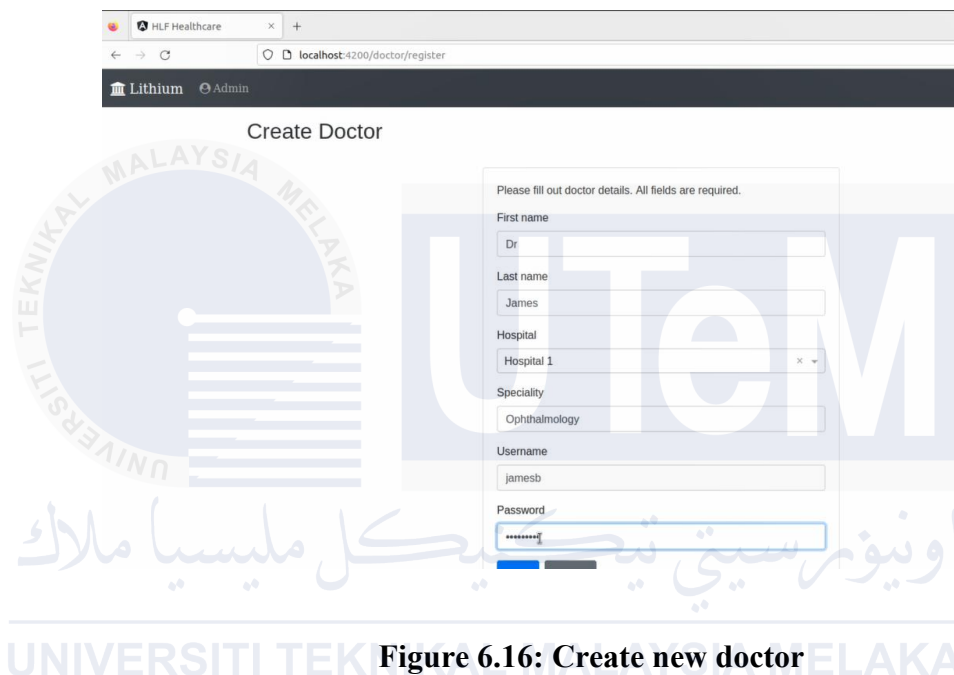


Figure 6.16: Create new doctor

6.4.2.2 Patient

(a) Login

If a patient is logged into the system using temporary password given for the first time, a message of requesting for changing new password will be appeared. The temporary password and new password are hashed and stored in the ledger. During every login, the entered password is hashed and compared with the hash value of password stored in ledger.

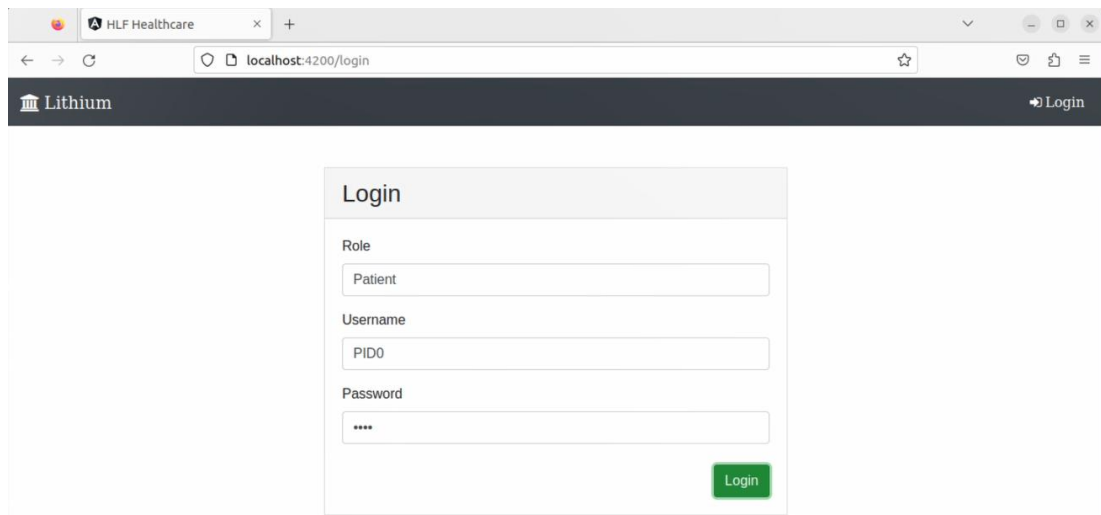


Figure 6.17: Patient login page

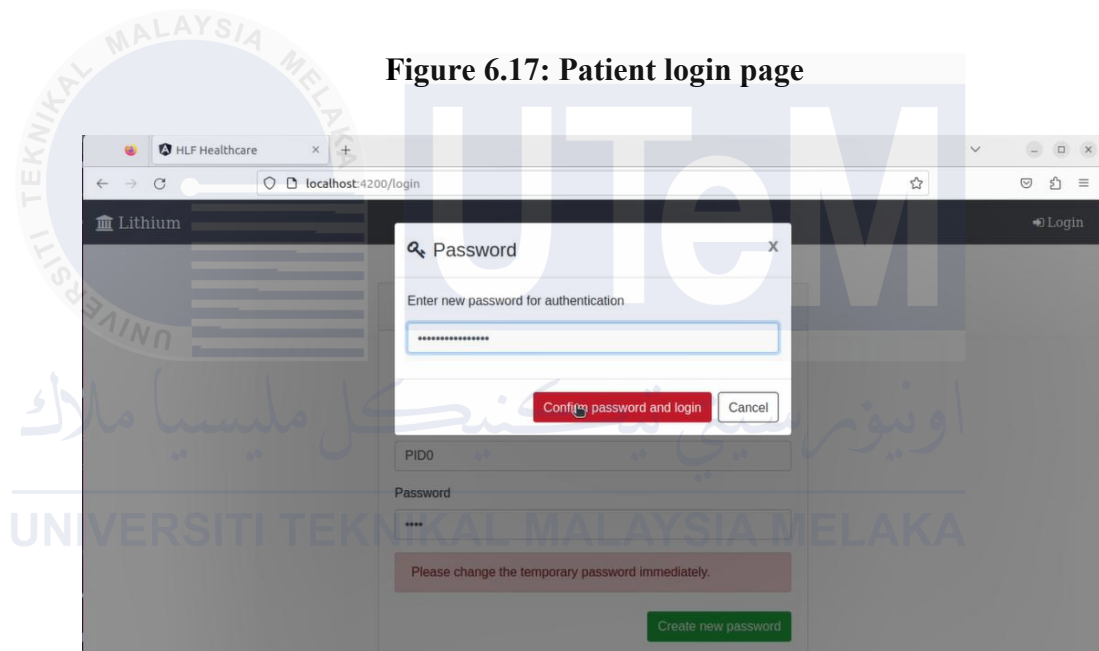


Figure 6.18: Mandatory of changing new password for first-time login patient

(b) View Personal Details

After successful login, the patient can view all his personal data as well as his current medical details. These data are retrieved from the world state by invoking the patient contract.

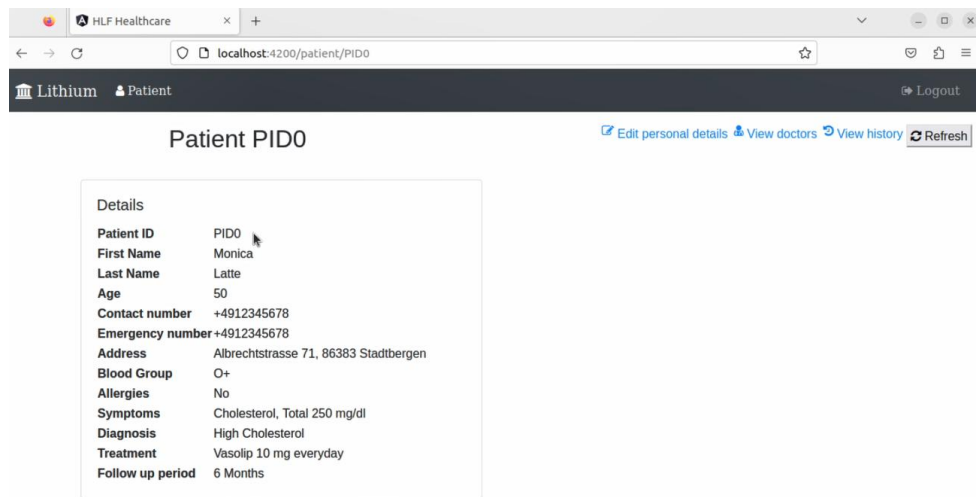


Figure 6.19: View personal details & EHR (patient)

(c) View List of Doctors & Grant/Revoke Access

From the patient dashboard selecting “View Doctors”, a list of available doctors from both hospitals are shown. The patient has the right to grant or revoke access to/from specific doctor. Doctors who are granted access only can view and update the patient’s EHR else unable to do so if access revoked.

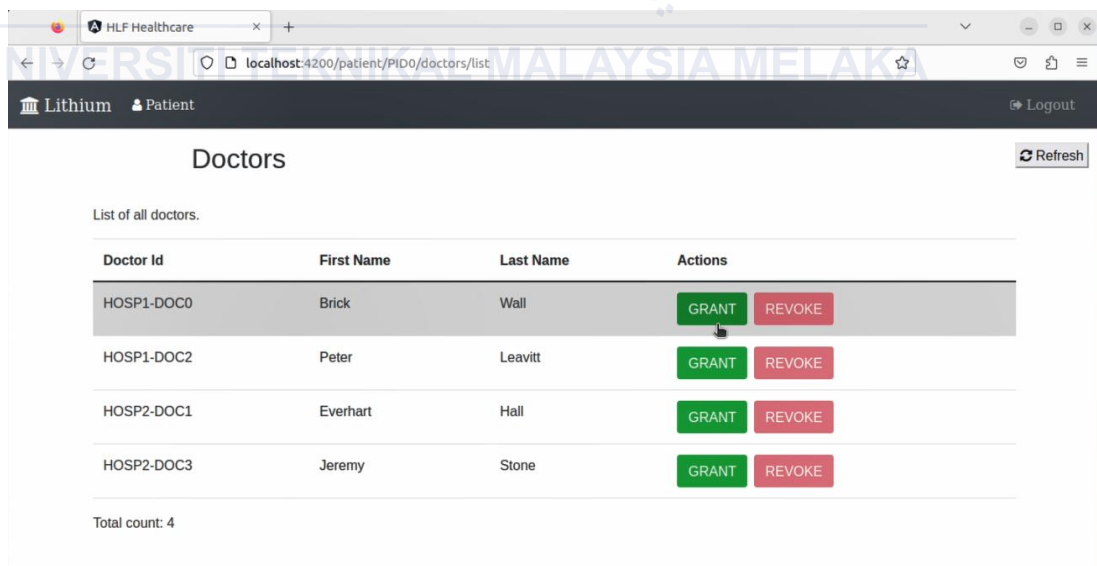


Figure 6.20: View list of doctors & grant/revoke access

(d) Edit Personal Details

The patient can edit his personal details. If any changes are made, the new details will be updated in the ledger using patient contract through new transaction.

Please fill out patient details. All fields are required.

Edit Patient

First name
Jacky

Last name
Cheung

Address
Ayer Keroh, Melaka

Age
60

Contact number
+6012345678

Emergency contact number
+6032456765

Save Cancel Refresh

Figure 6.21: Edit personal details

HLF Healthcare

localhost:4200/patient/PID0

Lithium Patient Logout

Patient PID0 Edit personal details View doctors View history Refresh

Details	
Patient ID	PID0
First Name	Jacky
Last Name	Cheung
Age	60
Contact number	+6012345678
Emergency number	+6032456765
Address	Ayer Keroh, Melaka
Blood Group	O+
Allergies	No
Symptoms	Cholesterol, Total 250 mg/dl
Diagnosis	High Cholesterol
Treatment	Vasolip 10 mg everyday
Follow up period	6 Months

Figure 6.22: Updated personal details

(e) *View EHR History*

The patient can view all his history records starting from the first visit to the latest visit where any modification in the records can be known. This functionality is possible due to the presence of `getHistoryForKey` API for public data in Hyperledger Fabric which returns the history of patient. This provides complete transparency over all the transactions made to the patient's data as all records are traceable including who modified the data and timestamp.

Last changed by	Date	First Name	Last Name	Age	Blood Group	Address	Contact number	Emergency number	Allergies	Diagnosis	Symptoms	Treatment
PID0	Mon Sep 11 2023	Jacky	Cheung	60	O+	Ayer Keroh, Melaka	+6012345678	+6032456765	No	High Cholesterol	Cholesterol, Total 250 mg/dl	Vasodilator every
PID0	Mon Sep 11 2023	Monica	Latte	50	O+	Albrechtstrasse 71, 86383 Stadtbergen	+4912345678	+4912345678	No	High Cholesterol	Cholesterol, Total 250 mg/dl	Vasodilator every
initLedger	Mon Sep 11 2023	Monica	Latte	50	O+	Albrechtstrasse 71, 86383 Stadtbergen	+4912345678	+4912345678	No	High Cholesterol	Cholesterol, Total 250 mg/dl	Vasodilator every

Total count: 3

Figure 6.23: View EHR history (patient)

6.4.2.3 Doctor

(a) *Login*

To login as doctor, user need to select role as “Doctor”, hospital his belong and correct credentials on login page. Upon successful login, details of the doctor will be shown.

Figure 6.24: Doctor login page

Figure 6.25: Doctor dashboard

(b) View List of Patients & Patient's EHR

If there is any patient who has granted access to the doctor, his information will be displayed otherwise the list will be empty. Through “View More”, doctor can view the latest condition and information of the patient.

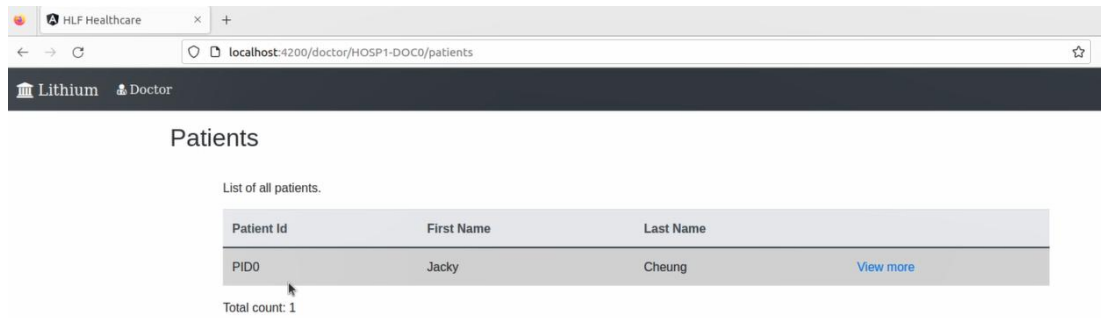


Figure 6.26: List of patients (doctor)



Figure 6.27: View patient's EHR

(c) Update EHR

In “Edit medical details”, doctor will be able to provide latest consultation and treatment by updating patient’s EHR. Upon successful update, it will redirected to the patient’s information page that displayed latest patient’s data.

HLF Healthcare

localhost:4200/patient/edit/PID0

Lithium Doctor

Edit Patient

Please fill out patient details. All fields are required.

Allergies
Yes

Symptoms
Lose weight, urinate alot

Diagnosis
Type 1 diabetes mellitus with diabetic autonomic

Treatment
Insulin shot every day

Follow Up Duration
5 Months

Save Cancel

Figure 6.28: Update patient's EHR

HLF Healthcare

localhost:4200/patient/PID0

Lithium Doctor

Patient PID0

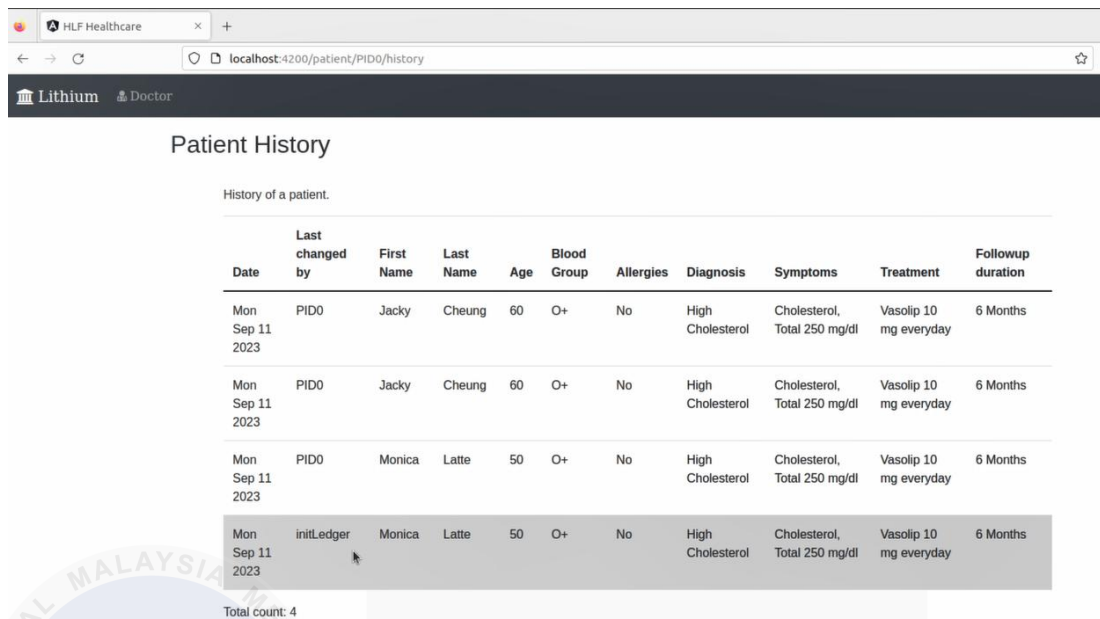
[Edit medical details](#)

Details	
Patient ID	PID0
First Name	Jacky
Last Name	Cheung
Age	60
Blood Group	O+
Allergies	No
Symptoms	Lose weight, urinate alot
Diagnosis	Type 1 diabetes mellitus with diabetic autonomic
Treatment	Insulin shot every day
Follow up period	5 Months

Figure 6.29: Updated patient's EHR

(d) View Patient's EHR history

As the patient's whole medical history is available to the doctor, it can ease the effort of doctor to retrieve patient's EHR and avoid any healthcare fraud, therefore improved the doctor's understanding on patient's condition so that better and appropriate medication can be conducted.



Date	Last changed by	First Name	Last Name	Age	Blood Group	Allergies	Diagnosis	Symptoms	Treatment	Followup duration
Mon Sep 11 2023	PID0	Jacky	Cheung	60	O+	No	High Cholesterol	Cholesterol, Total 250 mg/dl	Vasolip 10 mg everyday	6 Months
Mon Sep 11 2023	PID0	Jacky	Cheung	60	O+	No	High Cholesterol	Cholesterol, Total 250 mg/dl	Vasolip 10 mg everyday	6 Months
Mon Sep 11 2023	PID0	Monica	Latte	50	O+	No	High Cholesterol	Cholesterol, Total 250 mg/dl	Vasolip 10 mg everyday	6 Months
Mon Sep 11 2023	initLedge	Monica	Latte	50	O+	No	High Cholesterol	Cholesterol, Total 250 mg/dl	Vasolip 10 mg everyday	6 Months

Total count: 4

Figure 6.30: View patient's EHR history

6.5 Conclusion

This chapter has demonstrated the functionality of the prototype of blockchain-based web application for EHR data management among hospitals and proven that it is an applicable approach with different users permissions but increased data security compared with conventional EHR database system. Data validation testing has also been done which it has passed the test. For improvement, there should have an error message indicates that the request is not being processed. Although other performance parameters and tests for the blockchain such as transaction throughput, latency and resource consumption are not evaluated yet, it is planned that Hyperledger Caliper which is a blockchain performance benchmark framework can be used to test the performance of blockchain solution using custom use cases. This chapter has also helps to point out the advantages and weaknesses of the prototype.

CHAPTER 7: PROJECT CONCLUSION

7.1 Introduction

This chapter addresses the concluding phases of the project, which are Project Summarization, Project Contribution, Project Limitation, Future Works and Summary. This chapter holds significant importance within the project as it helps fellow researchers in enhancing our model by consolidating all its constraints and offering recommendations for enhancement.

7.2 Project Summarization

This project has been designed to determine the application of blockchain technology for EHR among hospitals in healthcare sector. There are 7 chapters in this project which starts with Chapter 1 that introduced about research background, problem statement, project questions, project objectives, project scope and project contribution while Chapter 2 emphasizes on introduction for blockchain technology, critical review on existing papers and researches then proposal of a better solution. Chapter 3 covers about methodology and approach used in this project to achieve project objectives. Next is Chapter 4 and Chapter 5 that can relate with second objective of this project which are design of the proposed blockchain-based web application for EHR data management including data involved, requirements and system architecture and implementation of the architecture using Hyperledger Fabric that aims to tackle the identified issues in current existing conventional healthcare data management systems. The following chapter which is Chapter 6 has analyzed

the results of the developed blockchain-based web application for EHR data management among hospitals in terms of functionality, evaluation and comparison between conventional EHR data management systems have also done to know whether the proposed blockchain-based web application for EHR data management is a more secure and efficient method. Lastly, Chapter 7 concludes the entire project and explain further progress that could implement in the future.

The system's components are adaptable to meet specific design requirements such as creation of new patient and doctor, different rights of the participants in the network to modify or view their own or other person's data, which allow for scenario simulations mirroring real-world application behavior. This has offered an insight on complexity in managing patients' health records where many roles are involved and the challenges in system implementation, encompassing usability, security, scalability and maintainability. Beyond the prototype's accomplishments, further theoretical ideas have also been explored that could enhance the system, capitalizing the advantages of a decentralized blockchain approach, such as re-encryption algorithm which is a more robust security approach to protect public and private keys, feature of private data collection in Hyperledger Fabric that can prevent complexity of deploying a large number of channels when scaling different organizations with various business needs to join the network. Additionally, it has been proved that adopting a blockchain-based EHR data management system is feasible as patient's health records can be more secure in blockchain network where data integrity and transparency are ensured as any access to patient's medical history is preserved in the ledger through transaction history and a patient-centric system can provide greater control to individuals. Due to the nature of blockchain, also offered by Hyperledger Fabric, which data are distributed in blocks, immutable as hashes are stored and compared consistently, blockchain system can also mitigate the risk of single point failures in conventional EHR databases as copies of valid data are appended in the blockchain.

In Hyperledger Fabric, the pluggable feature for consensus method and identity management has made business organization easier to integrate their requirements with blockchain system. While data confidentiality is one of the primary goal in most businesses, permissioned-based blockchains provided by

Hyperledger Fabric have ensure data security as its membership service provider (MSP) component offers user authentication by issuing and validating certificates and also creation of private channel where transactions are only can be seen by specific peers within the channel has improved data privacy. However, there are some weaknesses that can be pointed out and could be enhanced in future works. For instance, the current version of Hyperledger Fabric only supports three roles in HLF Registrar which are peer, client and admin. The limitation of having just three roles is that it results in all clients having identical permissions despite there could have different role types and corresponding permissions for each custom-defined role in a blockchain system. Issues might arise as patient may not have access to view attributes that are relevant to doctors even though that information is stored in the blockchain, but only admin user can read the attributes. In this case, doctor and patient can be the user-defined roles rather than just the client where they could have their own set of rules. By creating user-defined role, it can overcome the issue of separate permissions cannot be applied as if both doctor and patient are client but yet this is a progress need to be improved by HLF. Whereas doctor is not an asset to the ledger, the name and specialty of the doctor are stored in the attributes of the identity. Even though there are some proposed techniques not able to finish on given time such as function of removing invalid doctor or patient and role of medical practitioner under emergency department, yet the blockchain-based EHR system still able to complete with base model.

7.3 Project Contribution

The Project Contribution that can gain from this Project is the study of blockchain technology and development of application using blockchain for EHR data management among hospitals. This project has proposed a more secure and efficient method using blockchain technology for EHR data management that can ensure smooth operation among hospitals. As the current client/server-based system stores patients' data differently in terms of system architecture, data format and structure resulting in interoperability challenges, the features of synchronized and duplicated ledger within the blockchain-based EHR system have resolved this problem and simplified the data sharing process among all hospitals within the network. It has also contributed the possibility of developing a national or even

international data sharing program involving healthcare data with researchers, partner facilities or other interested parties such as insurance providers.

In addition, the project's contribution includes demonstrating how a blockchain-based EHR data management system can address the security issues associated with conventional methods of data storage and sharing. These issues include inconvenient accessibility, potential for a single point of failure, data breaches and data alteration. The implementation of blockchain-based EHR system ensures data security, enhances flexibility, efficiency, transparency and reduces medical errors, thereby improving fault tolerance.

7.4 Project Limitation

The developed project contains few limitations. As the architecture of Hyperledger Fabric is quite complex and it offers limited database support including CouchDB and LevelDb only, it is a challenge to host documents or big data in the blockchain database and the limitation of Fabric to store data in JSON format causing the current prototype of the blockchain-based EHR data management system proposed is only accepting text-based data. The capabilities of the system could be expand to accommodate various data types, including medical images such as results of CT scan, ultrasound, X-ray etc to enable healthcare providers to store and analyze a broader range of patient information, facilitating a more comprehensive blockchain-based EHR system. While implementation of an application layer that will convert any data into text using Base64 is possible, many researchers have recommended an alternative way which is to store data in an external database and integrate with blockchain system, for example InterPlanetary File System (IPFS) which is a peer-to-peer file sharing network in a distributed file system that provides lookups and storage for the mapping of keys to values using distributed hash tables (DHT). However, it is believed that this approach will add another layer of complexity and security risks.

Another limitation faced in this project is the security implementation on patient data in the ledger on peer level using the feature of private data collection in Hyperledger Fabric. The utilization of private data collection is driven by security concerns when a client initiates a transaction, it is sent to the orderer before being

included in a block. Once added to a block, that block is broadcast to all participating organizations in the network meanwhile this process could potentially expose sensitive data included in the transaction, such as patients' data. To ensure the confidentiality of such sensitive information, private data collections are employed, allowing organizations to restrict access to specific data, ensuring that only authorized parties can view or transact with it. Although this feature can help to address the drawbacks of channel when it comes to scalability, the private data collection approach is unable to apply due to the version of Hyperledger Fabric used which is 2.2.2, does not support the API `getHistoryForKey` for a private data collection. However, it is believed that the Hyperledger Fabric community will progress to add a history index and chaincode API similar to public data history to enable the query of history of a private data key.

Besides, the project also faced challenges in scaling peers of hospital organizations and adding new organization into the network. Due to certain unforeseen reason, one fabric-peer container always stops whenever the network gets up. While adding third hospital organization to join the network, it has also failed to join the channel. Moreover, error handling in the proposed system should be enhanced to define error with error codes and messages. This is a common practice to help the user and developer to distinguish whether a request or response is successful or not. Until the time being, these problems have not been solved yet and required more research works.

7.5 Future Works

There are many improvements can be done to elevate the application to a product that can fits real-world scenario and different business requirements. As this project has only discussed about basic participant entities including patient, doctor and hospital, other stakeholders such as commercial insurance companies, researchers, laboratory, pharmacy and transnational health care industries might also have the potential to participate into the network through the blockchain-based EHR system. While scalability is allowed within Hyperledger Fabric framework, further studies are required to address this feature as there might involved different sub-channels and endorsement policies. Besides, the source code can also be improved to

make the solution more pluggable when adding more hospitals and its peers. Once the network scales up, it will need to handle higher amount of transaction requests and approvals, and this required more ordering peers to speed up the process. As suggestion, Apaches Kafka, an open-source distributed event streaming platform which can provide high-throughput, low-latency for real-time data handling should be considered to manage multiple ordering nodes. In addition, Kubernetes, an open-source container orchestration system for automating software deployment, scaling and management can also be used for production environment when there are getting more hospitals with their peers and channels exist in the network.

One notable trend in the healthcare industry is the increasing integration of artificial intelligence (AI) into medical systems. Implementing AI algorithms into EHR system where AI can aid in analyzing ledger data that includes patients' health data from all hospital in the network could revolutionize patient care in better prognosis and diagnosis support. AI can assist healthcare professionals in detecting patterns and anomalies within patient records, leading to quicker and more accurate diagnoses. Additionally, the current metadata concept keeps the actual patient data on the blockchain ledger as developed. However, it is not ideal when the blockchain network grows bigger as it might degrade the transaction speed, plus many healthcare entities have already invested significantly in conventional databases or cloud-based storage solutions, making it impractical to migrate all data into a blockchain network. By adopting a hybrid approach, where the blockchain stores metadata and access records while the actual patient data remains in conventional databases or cloud storage, healthcare organizations can leverage the benefits of both worlds. This integration not only preserves existing investments but also allows for a gradual transition towards decentralized systems and minimizing disruption which benefits from the blockchain technology.

Another important thing to note is the network communication within the system should be conducted using HTTPS to ensure transport-level security through TLS encryption. Currently, passwords are transmitted in plain text from the front end to the back end, and this security vulnerability can be addressed by implementing HTTPS. Besides, it is recommended to integrate temporary password generated when new user created through emails. Additionally, adaption of "forgot password"

feature is advisable. To enhance user experience and manage extensive patient data, UI/UX strategies and a search functionality can be implemented. However, it is essential to research whether Hyperledger Fabric supports wildcard searches. Given that the data originates from multiple databases, it is crucial to prevent frequent search queries so that system performance can be maintained.

These future developments align with the ongoing evolution of healthcare technology, ensuring that the blockchain-based EHR system remains relevant and adaptable to the changing landscape of EHR data management.

7.6 Conclusion

As a summary, this project has achieved its goals successfully and able to contribute for future research to develop a more sophisticated blockchain-based EHR data management system. It can be identified from this study that the personalized medical system is a practical and useful application. By utilizing Hyperledger Fabric, which is an encouraging blockchain framework that comes with the concepts of smart contracts, endorsement policies and provision of secure identities which make the records secure and controlled, it can offer a reliable and secure solution in managing medical health record. This technology enables interoperability among multiple hospital organizations, simplifying doctors' access to patient histories, eliminating the need for patients to carry physical medical records, and substantially enhancing the digital record-keeping process. Nonetheless, there is room for improvement within Hyperledger Fabric before a finalized product can be developed. Moreover, additional components or technologies beyond Hyperledger Fabric may be necessary to further enhance its effectiveness and satisfy real-world scenario requirements.

REFERENCES

- Antonopoulos, A., & Wood, G. (2018). *Mastering Ethereum: Building Smart Contracts and Dapps*. O'Reilly Media.
- Antwi, M., Adnane, A., Ahmad, F., Hussain, R., Muhammad Habib Rehman, & Kerrache, C. A. (2021). The Case of HyperLedger Fabric as a blockchain Solution for Healthcare Applications. *Blockchain: Research and Applications*, 2(1). <https://doi.org/10.1016/j.bcra.2021.100012>
- Bashir, I. (2017). *Mastering Blockchain*. Packt Publishing. <https://shorturl.at/pxGLP>
- Dagher, G. G., Adhikari, C. L., & Enderson, T. (2018). *Towards Secure Interoperability between Heterogeneous Blockchains using Smart Contracts*. Future Technologies Conference (FTC). https://saiconference.com/Downloads/FTC2017/Proceedings/8_Paper_491-Towards_Secure_Interoperability.pdf
- Dimitrov, D. V. (2019). Blockchain Applications for Healthcare Data Management. *Healthcare Informatics Research*, 25(1), 51. <http://dx.doi.org/10.4258/hir.2019.25.1.51>
- Donawa, A., Orukari, I., & Baker, C. E. (2019). Scaling Blockchains to Support Electronic Health Records for Hospital Systems. *2019 IEEE 10th Annual Ubiquitous Computing, Electronics and Mobile Communication Conference, UEMCON 2019*, 550-556, 8993101. <https://doi.org/10.1109/UEMCON47517.2019.8993101>
- Guo, H., & Yu, X. (2022). A Survey on Blockchain Technology and Its Security. *Blockchain: Research and Applications*, 3(2), 100067. <https://doi.org/10.1016/j.bcra.2022.100067>

Guo, R., Shi, H., Zhao, Q., & Zheng, D. (2018). Secure Attribute-Based Signature Scheme with Multiple Authorities for Blockchain in Electronic Health Records Systems. *IEEE Access*, 6, 11676-11686. doi: 10.1109/ACCESS.2018.2801266.

Huynh-The, T., Gadekallu, T. R., Wang, W., Yenduri, G., Ranaweera, P., Pham, Q., da Costa, D. B., & Liyanage, M. (2023). Blockchain for the metaverse: A Review. *Future Generation Computer Systems*, 143, 401-419. <https://doi.org/10.1016/j.future.2023.02.008>

Ismail, L. & Materwala, H. (2020). Blockchain Paradigm for Healthcare: Performance Evaluation. *Symmetry*, 12(8), 1200. DOI:10.3390/sym12081200

Jabbar, A., & Dani, S. (2020). Investigating the Link Between Transaction and Computational Costs in A Blockchain Environment. *International Journal of Production Research*, 58(11), 3423-3436. <https://doi.org/10.1080/00207543.2020.1754487>

Khatoun, A. (2020). A Blockchain-Based Smart Contract System for Healthcare Management. *Electronics*, 9(1), 94. <https://doi.org/10.3390/electronics9010094>

Knirsch, F., Unterweger, A., & Engel, D. (2019). Implementing a Blockchain from Scratch: Why, How, and What We Learned. *EURASIP Journal on Information Security*, 2. <https://doi.org/10.1186/s13635-019-0085-3>

Kumar, R., Wang, W., Kumar, J., Yang, T., Khan, A., Ali, W., & Ali, I. (2021). An Integration of Blockchain and Ai for Secure Data Sharing and Detection Of CT Images for the Hospitals. *Computerized Medical Imaging and Graphics*, 87. <https://doi.org/10.1016/j.compmedimag.2020.101812>

Lu, N., Zhang, Y., Shi, W., Kumari, S., & Choo, K. K. R. (2020). A secure and Scalable Data Integrity Auditing Scheme Based on Hyperledger Fabric.

Computers & Security, 92, 101741. doi:
<https://doi.org/10.1016/j.cose.2020.101741>.

Mitra, A., Bera, B., Das, A. K., Jamal, S. S., & You, I. (2023). Impact on Blockchain-based AI/ML-enabled Big Data Analytics for Cognitive Internet of Things environment. *Computer Communications*, 197, 173-185.
<https://doi.org/10.1016/j.comcom.2022.10.010>

Muhammad, S. & Soewito, B. (2022). A Blockchain for Secure Data Storing with Multi Chain on Smart Healthcare System. *Journal of Theoretical and Applied Information Technology*, 100, 13.
<http://www.jatit.org/volumes/Vol100No13/4Vol100No13.pdf>

Naz, M., A., F., Khalid, R., Javaid, N., Qamar, A. M., Afzal, M. K., & Shafiq, M. (2019). A Secure Data Sharing Platform Using Blockchain and Interplanetary File System. *Sustainability*, 11(24), 7054.
<https://doi.org/10.3390/su11247054>

Ndzimakhwe, M., Telukdarie, A., Munien, I., Vermeulen, A., K., U., & Philbin, S. P. (2023). A Framework for User-Focused Electronic Health Record System Leveraging Hyperledger Fabric. *Information*, 14(1), 51.
<https://doi.org/10.3390/info14010051>

Negi, L., & Bhatt, S. (2022). A Review on security schemes for Electronic Health Records. 2022 Fifth International Conference on Computational Intelligence and Communication Technologies (CCICT), Sonapat, India, 37-41. doi: 10.1109/CCICT56684.2022.00019.

Newman, O., & Thorpe, S. Towards a Privately Permissioned Blockchain for Electronic Health Care Environments – A Proof of Concept case study.

Odeh, A., Keshta, I., & Al-Haija, Q. A. (2022). Analysis of Blockchain in the Healthcare Sector: Application and Issues. *Symmetry*, 14(9), 1760.
<https://doi.org/10.3390/sym14091760>

- Purwono, Nisa, K., Wibisono, S. K., & Dewa, B. P. (2023). Private Blockchain in the Field of Health Services. *Journal of Advanced Health Informatics Research (JAHIR)*, 1, 1, 10-15. DOI: <https://doi.org/10.59247/jahir.v1i1.14>
- Rouhani, S., Butterworth, L., Simmons, A. D., Humphery, D. G., & Deters, R. (2019). MediChainTM: A Secure Decentralized Medical Data Asset Management System. *ArXiv*. https://doi.org/10.1109/Cybermatics_2018.2018.00258
- Tanwar, S., Parekh, K., & Evans, R. (2020). Blockchain-based electronic healthcare record system for healthcare 4.0 applications. *Journal of Information Security and Applications*, 50, 102407. <https://doi.org/10.1016/j.jisa.2019.102407>
- Tith, D., Lee, J., Suzuki, H., Wijesundara, W. M. A. B., Taira, N., Obi, T., & Ohyama, N. (2020). Patient Consent Management by a Purpose-Based Consent Model for Electronic Health Record Based on Blockchain Technology. *Healthcare Informatics Research* 2020, 26(4), 265-273. <https://doi.org/10.4258/hir.2020.26.4.265>
- Usman, M., & Qamar, U. (2020). Secure Electronic Medical Records Storage and Sharing Using Blockchain Technology. *Procedia Computer Science*, 174, 321-327. <https://doi.org/10.1016/j.procs.2020.06.093>
- Wang, Q., & Qin, S. (2021). A Hyperledger Fabric-Based System Framework for Healthcare Data Management. *Applied Sciences*, 11(24), 11693. <https://doi.org/10.3390/app112411693>
- Wang, S., Zhang, Y., & Zhang, Y. (2018). A Blockchain-Based Framework for Data Sharing with Fine-Grained Access Control in Decentralized Storage Systems. *IEEE Access*, 6, 38437-38450. doi: 10.1109/ACCESS.2018.2851611.

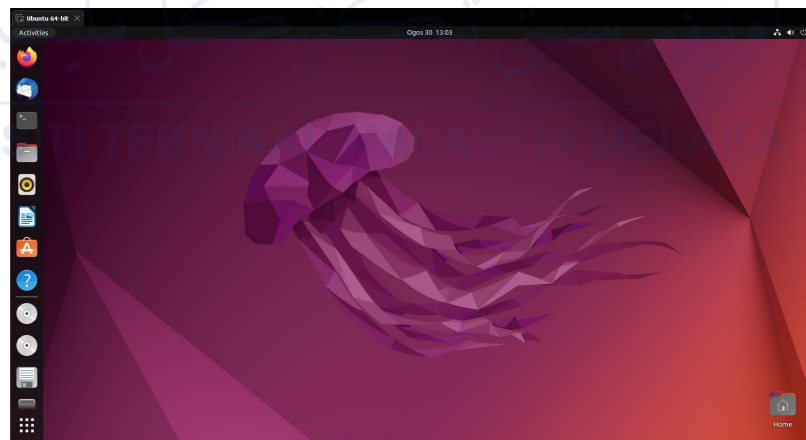
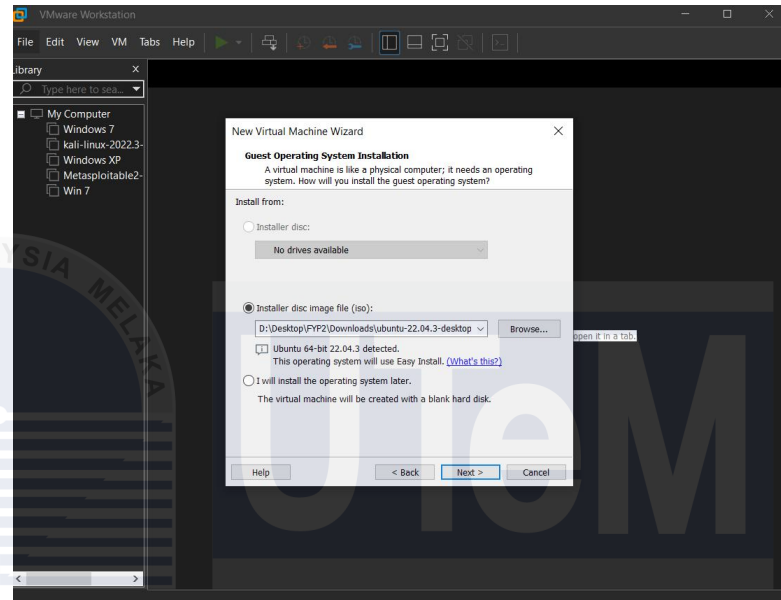
Westphal, E., & Seitz, H. (2021). Digital and Decentralized Management of Patient Data in Healthcare Using Blockchain Implementations. *Frontiers in Blockchain*, 4, 732



APPENDICES

APPENDIX A - INSTALLATION OF PREREQUISITE

1. Setting up Ubuntu 22.04.3 LTS in VMware.



2. Install git.

```

chek@chek-virtual-machine: ~
chek@chek-virtual-machine:~$ git --version
Command 'git' not found, but can be installed with:
sudo apt install git
chek@chek-virtual-machine:~$ sudo apt-get install git
[sudo] password for chek:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  git-man liberror-perl
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-email git-gui gitk gitweb
  git-cvs git-mediawiki git-svn
The following NEW packages will be installed:
  git git-man liberror-perl
0 upgraded, 3 newly installed, 0 to remove and 2 not upgraded.
Need to get 4,147 kB of archives.
After this operation, 21.0 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://my.archive.ubuntu.com/ubuntu jammy/main amd64 liberror-perl all 0.17029-1 [26.5 kB]
Get:2 http://my.archive.ubuntu.com/ubuntu jammy-updates/main amd64 git-man all 1:2.34.1-1ubuntu1.10 [954 kB]
Get:3 http://my.archive.ubuntu.com/ubuntu jammy-updates/main amd64 git amd64 1:2.34.1-1ubuntu1.10 [3,166 kB]
Fetched 4,147 kB in 2s (1,716 kB/s)
Selecting previously unselected package liberror-perl.
(Reading database ... 198900 files and directories currently installed.)
Preparing to unpack .../liberror-perl_0.17029-1_all.deb ...
Unpacking liberror-perl (0.17029-1) ...
Selecting previously unselected package git-man.
Preparing to unpack .../git-man_1%3a2.34.1-1ubuntu1.10_all.deb ...
Unpacking git-man (1:2.34.1-1ubuntu1.10) ...
Selecting previously unselected package git.
Preparing to unpack .../git_1%3a2.34.1-1ubuntu1.10_amd64.deb ...
Unpacking git (1:2.34.1-1ubuntu1.10) ...
Setting up liberror-perl (0.17029-1) ...
Setting up git-man (1:2.34.1-1ubuntu1.10) ...
Setting up git (1:2.34.1-1ubuntu1.10) ...
Processing triggers for man-db (2.10.2-1) ...
chek@chek-virtual-machine:~$ git --version
git version 2.34.1
chek@chek-virtual-machine:~$

```

3. Install curl.

```

chek@chek-virtual-machine:~$ sudo apt install curl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  curl
0 upgraded, 1 newly installed, 0 to remove and 2 not upgraded.
Need to get 194 kB of archives.
After this operation, 454 kB of additional disk space will be used.
Get:1 http://my.archive.ubuntu.com/ubuntu jammy-updates/main amd64 curl amd64 7.81.0-1ubuntu1.13 [194 kB]
Fetched 194 kB in 1s (206 kB/s)
Selecting previously unselected package curl.
(Reading database ... 199885 files and directories currently installed.)
Preparing to unpack .../curl_7.81.0-1ubuntu1.13_amd64.deb ...
Unpacking curl (7.81.0-1ubuntu1.13) ...
Setting up curl (7.81.0-1ubuntu1.13) ...
Processing triggers for man-db (2.10.2-1) ...
chek@chek-virtual-machine:~$ curl --help
curl: try 'curl --help' or 'curl --manual' for more information
chek@chek-virtual-machine:~$ curl --version
curl 7.81.0 (x86_64-pc-linux-gnu) libcurl/7.81.0 OpenSSL/3.0.2 zlib/1.2.11 brotli/1.0.9 zstd/1.4.8 libidn2/2.3.2 libpsl/0.21.0 (+libidn2/2.3.2) libssh/0.9.6/openssl/zlib nghttp2/1.43.0 librtmp/2.3 OpenLDAP/2.5.16
Release-Dates: 2022-01-05
Protocols: dict file ftp ftps gopher gophers http https imap imaps ldap ldaps mqtt pop3 pop3s rtsp rtsp scp sftp smb smbs smtp smtps telnet tftp
Features: alt-svc AsynchDNS brotli GSS-API HSTS HTTP2 HTTPS-proxy IDN IPv6 Kerberos Largefile libz NTLM NTLM_WB PSL SPNEGO SSL TLS-SRP UnixSockets zstd
chek@chek-virtual-machine:~$

```

4. Install docker desktop.

```

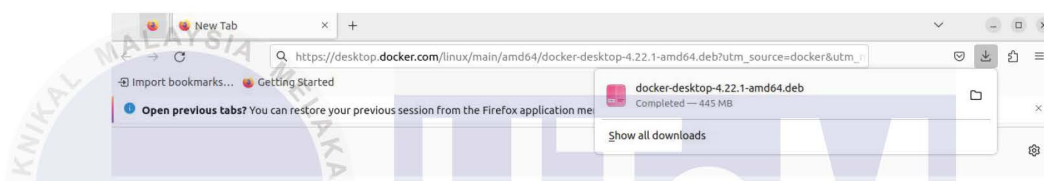
chek@chek-virtual-machine:~$ sudo usermod -aG kvm chek
[sudo] password for chek:
chek@chek-virtual-machine:~$ ls -al /dev/kvm
crw-rw----+ 1 root kvm 10, 232 Ogos 30 13:44 /dev/kvm
chek@chek-virtual-machine:~$ sudo apt install gnome-terminal
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
gnome-terminal is already the newest version (3.44.0-1ubuntu1).
gnome-terminal set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
chek@chek-virtual-machine:~$ sudo apt remove docker-desktop
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package docker-desktop

```

```

chek@chek-virtual-machine:~$ sudo apt-get update
Hit:1 http://my.archive.ubuntu.com/ubuntu jammy InRelease
Hit:2 http://my.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:3 http://my.archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:4 http://security.ubuntu.com/ubuntu jammy-security InRelease
Reading package lists... Done
chek@chek-virtual-machine:~$ sudo apt-get install ca-certificates curl gnupg
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
ca-certificates is already the newest version (20230311ubuntu0.22.04.1).
ca-certificates set to manually installed.
curl is already the newest version (7.81.0-1ubuntu1.13).
gnupg is already the newest version (2.2.27-3ubuntu2.1).
gnupg set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 2 not upgraded.
chek@chek-virtual-machine:~$ sudo install -m 0755 -d /etc/apt/keyrings
chek@chek-virtual-machine:~$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
chek@chek-virtual-machine:~$ sudo chmod a+r /etc/apt/keyrings/docker.gpg
chek@chek-virtual-machine:~$ echo \
"deb [arch=$(dpkg --print-architecture)] signed-by=/etc/apt/keyrings/docker.gpg https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "SVERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
chek@chek-virtual-machine:~$ sudo apt-get update
Get:1 https://download.docker.com/linux/ubuntu jammy InRelease [48.9 kB]
Get:2 https://download.docker.com/linux/ubuntu jammy/stable amd64 Packages [21.4 kB]
Hit:3 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:4 http://my.archive.ubuntu.com/ubuntu jammy InRelease
Hit:5 http://my.archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:6 http://my.archive.ubuntu.com/ubuntu jammy-backports InRelease
Fetched 70.3 kB in 1s (64.5 kB/s)
Reading package lists... Done
chek@chek-virtual-machine:~$

```



```

chek@chek-virtual-machine:~$ cd Desktop
chek@chek-virtual-machine:~/Desktop$ sudo apt-get install ./docker-desktop-4.22.1-amd64.deb
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Note, selecting 'docker-desktop' instead of './docker-desktop-4.22.1-amd64.deb'
The following additional packages will be installed:
docker-buildx-plugin docker-ce-cli docker-compose-plugin ibverbs-providers ipxe-qemu
ipxe-qemu-256k-compat-efi-roms libaio1 libcacard0 libdaxctl1 libdecor-0-0
libdecor-0-plugin-1-cairo libfdt1 libgfp10 libgfrpc0 libgfxdr0 libglusterfs0
libibverbs1 libiscsi17 libnbdctl0 libnmem1 libnmemobj1 libqrencode4 librados2 librbdl1
librdmacm1 libsd2-2.0-0 libslirp0 libspice-server1 liburing2 libusbredirparser1
libvirtglrenderer1 ovmf pass qemu-block-extra qemu-system-common qemu-system-data
qemu-system-gui qemu-system-x86 qemu-utils qrencode seabios tree uidmap xclip
Suggested packages:
gststreamer1.0-libav gststreamer1.0-plugins-ugly libxml-simple-perl python ruby samba vde2
debootstrap
The following NEW packages will be installed:
docker-buildx-plugin docker-ce-cli docker-compose-plugin docker-desktop
ibverbs-providers ipxe-qemu ipxe-qemu-256k-compat-efi-roms libaio1 libcacard0 libdaxctl1
libdecor-0-0 libdecor-0-plugin-1-cairo libfdt1 libgfp10 libgfrpc0 libgfxdr0
libglusterfs0 libibverbs1 libiscsi17 libnbdctl0 libnmem1 libnmemobj1 libqrencode4 librados2 librbdl1
librdmacm1 libsd2-2.0-0 libslirp0 libspice-server1 liburing2 libusbredirparser1
libvirtglrenderer1 ovmf pass qemu-block-extra qemu-system-common qemu-system-data
qemu-system-gui qemu-system-x86 qemu-utils qrencode seabios tree uidmap xclip

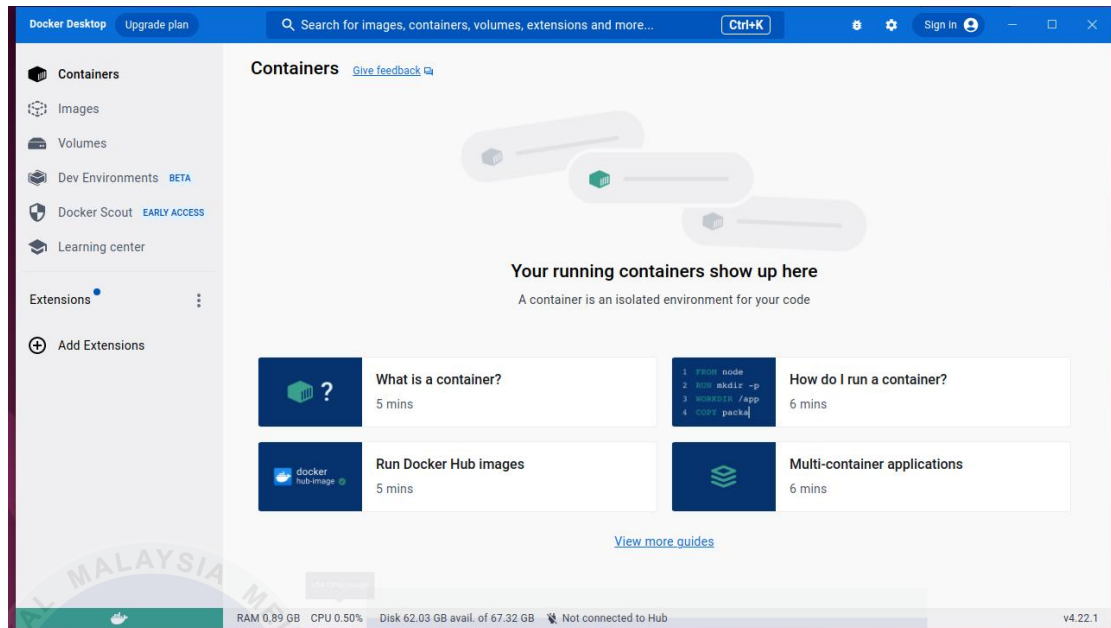
```

```

chek@chek-virtual-machine:~$ systemctl --user start docker-desktop
chek@chek-virtual-machine:~$ docker compose version
Docker compose version v2.20.2-desktop.1
chek@chek-virtual-machine:~$ docker --version
Docker version 24.0.5, build ced0996
chek@chek-virtual-machine:~$ docker version
Client: Docker Engine - Community
Cloud integration: v1.0.35-desktop+001
Version: 24.0.5
API version: 1.43
Go version: go1.20.6
Git commit: ced0996
Built: Fri Jul 21 20:35:18 2023
OS/Arch: linux/amd64
Context: desktop-linux

Server: Docker Desktop 4.22.1 (118664)
Engine:
Version: 24.0.5
API version: 1.43 (minimum version 1.12)
Go version: go1.20.6
Git commit: a61e2b4
Built: Fri Jul 21 20:35:45 2023
OS/Arch: linux/amd64
Experimental: false
containerd:
Version: 1.6.21
GitCommit: 3dce8eb055cbb6872793272b4f20ed16117344f8
runc:
Version: 1.1.7
GitCommit: v1.1.7-0-g860f061
docker-init:
Version: 0.19.0
GitCommit: de40ad0
chek@chek-virtual-machine:~$ systemctl --user enable docker-desktop
Created symlink /home/chek/.config/systemd/user/docker-desktop.service → /usr/lib/systemd/user/docker-desktop.service.
Created symlink /home/chek/.config/systemd/user/graphical-session.target.wants/docker-desktop.service → /usr/lib/systemd/user/docker-desktop.service.

```



5. Install docker compose.

```
chek@chek-virtual-machine:~$ sudo apt-get install git curl docker-compose -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
curl is already the newest version (7.81.0-1ubuntu1.13)
```

```
chek@chek-virtual-machine:~$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
docker-ce-cli is already the newest version (5:24.0.5-1-ubuntu.22.04-jammy).
docker-ce-cli set to manually installed.
docker-compose-plugin is already the newest version (2.20.2-1-ubuntu.22.04-jammy)
```

```
chek@chek-virtual-machine:~$ sudo systemctl start docker
chek@chek-virtual-machine:~$
```

```
chek@chek-virtual-machine:~$ sudo usermod -a -G docker chek
chek@chek-virtual-machine:~$ docker --version
Docker version 24.0.5, build ced0996
chek@chek-virtual-machine:~$ docker-compose --version
docker-compose version 1.29.2, build unknown
chek@chek-virtual-machine:~$
```

6. Install golang.

```
chek@chek-virtual-machine:~$ sudo apt install golang-go
Reading package lists... Done
Building dependency tree... Done
```

```
chek@chek-virtual-machine:~$ go version
go version go1.18.1 linux/amd64
chek@chek-virtual-machine:~$
```

7. Install Hyperledger Fabric and download binary, images.

```

chek@chek-virtual-machine:~$ curl -sSL https://bit.ly/2ysb0FE | bash -s -- 2.2.2 1.4.9

Clone hyperledger/fabric-samples repo

===> Cloning hyperledger/fabric-samples repo
Cloning into 'fabric-samples'...
remote: Enumerating objects: 12544, done.
remote: Counting objects: 100% (577/577), done.
remote: Compressing objects: 100% (323/323), done.
remote: Total 12544 (delta 274), reused 426 (delta 224), pack-reused 11967
Receiving objects: 100% (12544/12544), 22.53 MiB | 1.40 MiB/s, done.
Resolving deltas: 100% (6728/6728), done.
===> Checking out v2.2.2 of hyperledger/fabric-samples

Pull Hyperledger Fabric binaries

===> Downloading version 2.2.2 platform specific fabric binaries
===> Downloading: https://github.com/hyperledger/fabric/releases/download/v2.2.2/hyperledger-fabric-linux-amd64-2.2.2.tar.gz
% Total % Received % Xferd Average Speed Time Time Time Current
         Dload Upload Total Spent Left Speed
 0     0     0     0     0     0     0 0 0:00:00 0:00:00 0:00:00 0
100 72.8M 100 72.8M 0 0 9.9M 0 0:00:07 0:00:07 0:00:00 11.4M
==> Done.
===> Downloading version 1.4.9 platform specific fabric-ca-client binary
===> Downloading: https://github.com/hyperledger/fabric-ca/releases/download/v1.4.9/hyperledger-fabric-ca-linux-amd64-1.4.9.tar.gz
% Total % Received % Xferd Average Speed Time Time Time Current
         Dload Upload Total Spent Left Speed
 0     0     0     0     0     0     0 0 0:00:00 0:00:00 0:00:00 0
100 23.6M 100 23.6M 0 0 9186k 0 0:00:02 0:00:02 0:00:00 16.4M
==> Done.

```

8. Install node, npm and angular.

```

chek@chek-virtual-machine:~$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
% Total % Received % Xferd Average Speed Time Time Time Current
         Dload Upload Total Spent Left Speed
100 14926 100 14926 0 0 25194 0 0:00:00 0:00:00 0:00:00 25127
=> nvm is already installed in /home/chek/.nvm, trying to update using git
=> Compressing and cleaning up git repository

=> nvm source string already in /home/chek/.bashrc
=> bash_completion source string already in /home/chek/.bashrc
=> Close and reopen your terminal to start using nvm or run the following to use it now:

export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh" # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion" # This loads nvm bash_completion
chek@chek-virtual-machine:~$ source ~/.bashrc
bash: /usr/local/bin/ng: No such file or directory
chek@chek-virtual-machine:~$ nvm install 10.15.3
Downloading and installing node v10.15.3...
Downloading https://nodejs.org/dist/v10.15.3/node-v10.15.3-linux-x64.tar.xz...
##### 100.0%
Computing checksum with sha256sum
Checksums matched!
Now using node v10.15.3 (npm v6.4.1)
Creating default alias: default -> 10.15.3 (-> v10.15.3)
chek@chek-virtual-machine:~$ npm install -g @angular/cli@11
npm WARN deprecated @schematics/update@0.1102.19: This was an internal-only Angular package up through Angular v11 which is no longer used or maintained. Upgrade Angular to v12+ to remove this dependency.
npm WARN deprecated source-map-codec@1.4.8: Please use gajdewell/source-map-codec instead
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/request/request/issues/3142
npm WARN deprecated har-validator@5.1.5: this library is no longer supported
npm WARN deprecated node-gyp@4.0.0: Please upgrade to version 7 or higher. Older versions may use Math.random() in certain circumstances, which is known to be problematic. See https://www.devblog.com/math-random for details.
npm WARN deprecated @npmcli/move-file@1.2.0: This functionality has been moved to @npmcli/fs
npm WARN deprecated @npmcli/cli-detect@1.4.0: this package has been deprecated, use 'cli-info' instead
/home/chek/.nvm/versions/node/v10.15.3/bin/ng -> /home/chek/.nvm/versions/node/v10.15.3/lib/node_modules/@angular/cli/bin/ng
> @angular/cli@11.2.19 postinstall /home/chek/.nvm/versions/node/v10.15.3/lib/node_modules/@angular/cli
> node ./bin/postinstall/script.js
+ @angular/cli@11.2.19
added 247 packages from 183 contributors in 23.264s
chek@chek-virtual-machine:~$ node -v

```

```

chek@chek-virtual-machine:~$ node -v
v10.15.3
chek@chek-virtual-machine:~$ npm -v
6.4.1
chek@chek-virtual-machine:~$ ng --version

Angular CLI 11.2.19
Node: 10.15.3
OS: linux x64

Angular:
...
Ivy Workspace:

Package          Version
-----
@angular-devkit/architect    0.1102.19 (cli-only)
@angular-devkit/core        11.2.19 (cli-only)
@angular-devkit/schematics  11.2.19 (cli-only)
@schematics/angular        11.2.19 (cli-only)
@schematics/update         0.1102.19 (cli-only)

chek@chek-virtual-machine:~$

```

9. Install python.

```

chek@chek-virtual-machine:~$ sudo apt install python2
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libpython2-stdlib python2-minimal
Suggested packages:
  python2-doc python-tk
The following NEW packages will be installed:
  libpython2-stdlib python2 python2-minimal
0 upgraded, 3 newly installed, 0 to remove and 2 not upgraded.
Need to get 37.4 kB of archives.

```

APPENDIX B - STEPS TO START NETWORK

1. Copy bin directory from 'fabric samples' to blockchain-hyperledger-fabric-electronic-patient-records/app directory.

2. At first network directory, execute the network.sh file with argument up.

```

chek@chek-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records$ cd app
chek@chek-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app$ cd first-network
chek@chek-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network$ ./network.sh up createChannel
Creating channel: hosp1
If network is not up, starting nodes with cli timeout of '5' tries and cli delay of '3' seconds and using database 'couchdb' with crypto from 'Certificate Authorities'
Bringing up network
LOCAL_VERSION=2.0
ORDERER_IMAGE_VERSION=2.0
CL_LOCAL_VERSION=1.4.9
CA_ORDERER_IMAGE_VERSION=1.0.0
CA_CLIENT_IMAGE_VERSION=1.0.0
Creating network "lithium-network_hospital" with the default driver
Creating ca_hosp1 ... done
Creating ca_hosp2 ... done
Creating ca_orderer ... done
Enroll the CA admin
+ fabric-ca-client enroll -u https://hospiadmin:hosp1lithium@localhost:7054 --caname ca-hosp1 --tls.certfiles /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/fabric-ca/hosp1/tls-cert.pem
2023/09/10 17:22:59 [INFO] Created a default configuration file at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1.lithium.com/fabric-ca-client-config.yaml
2023/09/10 17:22:59 [INFO] TLS Enabled
2023/09/10 17:22:59 [INFO] generating key: &{Aecdsa S:256}
2023/09/10 17:22:59 [INFO] encoded CSR
2023/09/10 17:22:59 [INFO] Stored client certificate at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1.lithium.com/msp/signcerts/cert.pem
2023/09/10 17:22:59 [INFO] Stored root CA certificate at /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1.lithium.com/msp/roots/tlsca.pem

2023-09-10 17:24:10.001 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:10.009 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:10.909 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:11.210 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:11.226 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:11.331 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:11.368 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:11.569 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:11.588 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:11.698 +08 [cli.common] readBlock -> INFO 000 Expect block, but got status: &{SERVICE_UNAVAILABLE}
2023-09-10 17:24:11.801 +08 [channelCmd] InitCmdFactory -> INFO 000 Endorser and orderer connections initialized
2023-09-10 17:24:12.018 +08 [cli.common] readBlock -> INFO 010 Received block: 0
Channel 'hospitalchannel' created
Join Hospital 1 peers to the channel...
Using Hospital 1
+ peer channel join -b ./channel-artifacts/hospitalchannel.block
+ res=0
2023-09-10 17:24:15.226 +08 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2023-09-10 17:24:16.041 +08 [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
Join Hospital 2 peers to the channel...
Using Hospital 2
+ peer channel join -b ./channel-artifacts/hospitalchannel.block
+ res=0
2023-09-10 17:24:19.213 +08 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2023-09-10 17:24:19.608 +08 [channelCmd] executeJoin -> INFO 002 Successfully submitted proposal to join channel
Updating anchor peers for Hospital 1...
Using Hospital 1
+ peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.lithium.com -c hospitalchannel -f ./channel-artifacts/hosp1MSpanchors.tx --tls --cafile /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/ordererOrganizations/lithium.com/orderers/orderer.lithium.com/msp/tlsacerts/tlsca.lithium.com-cert.pem
+ res=0
2023-09-10 17:24:22.755 +08 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2023-09-10 17:24:22.795 +08 [channelCmd] update -> INFO 002 Successfully submitted channel update
Anchor peers updated for org 'hosp1MSP' on channel 'hospitalchannel'
Updating anchor peers for Hospital 2...
Using Hospital 2
+ peer channel update -o localhost:7050 --ordererTLSHostnameOverride orderer.lithium.com -c hospitalchannel -f ./channel-artifacts/hosp2MSpanchors.tx --tls --cafile /home/chek/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/ordererOrganizations/lithium.com/orderers/orderer.lithium.com/msp/tlsacerts/tlsca.lithium.com-cert.pem
+ res=0
2023-09-10 17:24:28.875 +08 [channelCmd] InitCmdFactory -> INFO 001 Endorser and orderer connections initialized
2023-09-10 17:24:29.008 +08 [channelCmd] update -> INFO 002 Successfully submitted channel update
Anchor peers updated for org 'hosp2MSP' on channel 'hospitalchannel'
Channel successfully joined
chek@chek-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network$

```

3. Package smart contract codes inside 'patient-asset-transfer' using deployCC command.


```

connection-hosp1.json
Built a CA Client named ca-hosp1
Successfully registered and enrolled user PID5 and imported it into the wallet
msg: Successfully enrolled user PID5 and imported it into the wallet
Done
Built a file system wallet at /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/application-javascript/wallet
Loaded the network configuration located at /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1.lithium.com/connection-hosp1.json
Built a CA Client named ca-hosp1
Successfully registered and enrolled user HOSP1-DOC0 and imported it into the wallet
msg: Successfully enrolled user HOSP1-DOC0 and imported it into the wallet
Built a file system wallet at /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/application-javascript/wallet
Loaded the network configuration located at /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp2.lithium.com/connection-hosp2.json
Built a CA Client named ca-hosp2
Successfully registered and enrolled user HOSP2-DOC1 and imported it into the wallet
msg: Successfully enrolled user HOSP2-DOC1 and imported it into the wallet
Built a file system wallet at /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/application-javascript/wallet
Loaded the network configuration located at /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp1.lithium.com/connection-hosp1.json
Built a CA Client named ca-hosp1
Successfully registered and enrolled user HOSP1-DOC2 and imported it into the wallet
msg: Successfully enrolled user HOSP1-DOC2 and imported it into the wallet
Built a file system wallet at /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/patient-asset-transfer/application-javascript/wallet
Loaded the network configuration located at /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/first-network/organizations/peerOrganizations/hosp2.lithium.com/connection-hosp2.json
Built a CA Client named ca-hosp2
Successfully registered and enrolled user HOSP2-DOC3 and imported it into the wallet
msg: Successfully enrolled user HOSP2-DOC3 and imported it into the wallet

> patient-application@1.0.0 start /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/server
> ./node_modules/nodemon/bin/nodemon.js src/server.js

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/server.js`
Backend server running on 3001

```

```

[nodemon] 1.19.4
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/server.js`
Backend server running on 3001

```

8. Go to app/client, install node project by npm install.

```

cheq@cheq-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app/server$ cd ../client
cheq@cheq-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app/client$ npm install
npm WARN read-shrinkwrap This version of npm is compatible with lockfileVersion@0, but package-lock.json was generated for lockfileVersion@2. I'll try to do my best with it!
[... ] \ extract:qbabel/plugin-transform-reserved-words: http fetch GET 200 https://registry.npmjs.org/qbabel/plugin-transform-block-scoping/-/plugin-transform-block-

> cypress@6.8.0 postinstall /home/cheq/blockchain-hyperledger-fabric-electronic-patient-records/app/client/node_modules/cypress
> node index.js --exec install

Installing Cypress (version: 6.8.0)
  ✓ Downloaded Cypress
  ✓ Unzipped Cypress
  ✓ Finished Installation /home/cheq/.cache/Cypress/6.8.0

You can now open Cypress by running: node_modules/.bin/cypress open
https://on.cypress.io/installing-cypress

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules/webpack-dev-server/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.13 (node_modules/watchpack-chokidar2/node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@2.1.3 (node_modules/fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"linux","arch":"x64"})

added 1587 packages from 1305 contributors and audited 1593 packages in 91.104s
found 265 vulnerabilities (2 low, 107 moderate, 131 high, 25 critical)
  run `npm audit fix` to fix them, or `npm audit` for details
cheq@cheq-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app/client$

```

9. Brings up angular server using command ng serve -o.

```

cheq@cheq-virtual-machine:~/blockchain-hyperledger-fabric-electronic-patient-records/app/client$ ng serve -o
Your global Angular CLI version (11.2.19) is greater than your local version (11.0.5). The local Angular CLI version is used.

To disable this warning use "ng config -o cli.warnings.versionMismatch false".
Compiling @angular/core : es2015 as esm2015
Compiling @angular/common : es2015 as esm2015
Compiling @angular/platform-browser : es2015 as esm2015
Compiling @angular/platform-browser-dynamic : es2015 as esm2015
Compiling @angular/common/http : es2015 as esm2015
Compiling @angular/forms : es2015 as esm2015
Compiling @ng-select/ng-select : es2015 as esm2015
Compiling @ng-bootstrap/ng-bootstrap : es2015 as esm2015
Compiling @angular/router : es2015 as esm2015
  ✓ Browser application bundle generation complete.

Initial Chunk Files | Names | Size
vendor.js           | vendor | 3.86 MB
main.js             | main   | 255.59 KB
styles.css          | styles | 218.71 KB
polyfills.js        | polyfills | 141.34 KB
runtime.js          | runtime | 6.15 KB
                    | Initial Total | 4.46 MB

Build at: 2023-09-11T04:37:30.269Z - Hash: 248764c809fe0937cf - Time: 2006ms
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/ **

  ✓ Compiled successfully.
  ✓ Browser application bundle generation complete.

Initial Chunk Files | Names | Size
styles.css          | styles | 218.71 KB

4 unchanged chunks

Build at: 2023-09-11T04:37:30.176Z - Hash: c69d95f4032435ff3ae2 - Time: 7263ms
  ✓ Compiled successfully.

```