

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/354951836>

Blockchain Solution to Healthcare Record System using Hyperledger Fabric

Technical Report · March 2021

DOI: 10.13140/RG.2.2.27957.60640

CITATION

1

READS

1,769

3 authors:



[Varsha Kamath](#)

Frankfurt University of Applied Sciences

1 PUBLICATION 1 CITATION

[SEE PROFILE](#)



[Jathin Sreenivas](#)

Frankfurt University of Applied Sciences

6 PUBLICATIONS 3 CITATIONS

[SEE PROFILE](#)



[Kshitij Yelpale](#)

Frankfurt University of Applied Sciences

1 PUBLICATION 1 CITATION

[SEE PROFILE](#)



Frankfurt University of Applied Sciences

–High Integrity Systems–

Blockchain Solution to Healthcare Record System using Hyperledger Fabric

is a bonafide work done by

Jathin Sreenivas

Kshitij Yelpale

Varsha Vasudev Kamath

Under the guidance of : Prof. Dr. Martin Kappes

ABSTRACT

The research project explores the potential of Hyperledger Fabric in the health care industry. The current solution in healthcare for storing and sharing medical records is the Electronic Health Record (EHR) which is highly sensitive. The majority of EHR data sharing is still done through mail because shortage of trustable and reliable health data sharing mechanism. This leads to significant delays in the patient's treatment [1]. Blockchain has the potential to improve health care by putting the patient at the center of the system and enhancing health data protection and interoperability. Blockchain is a decentralized framework providing a cryptographic guarantee of data integrity, security, privacy, and smart contracts for data access. The paper introduces various concepts that are used in Hyperledger Fabric, proposes a permissioned blockchain system solution for storing and sharing health care records using the Hyperledger fabric framework. Patients have full control of their medical information and authorize the doctors that can view medical details by using grant and revoke access mechanisms and also presents the challenges faced, and issues to resolve, before the successful adoption of Hyperledger Fabric in healthcare systems.

CONTENTS

1	INTRODUCTION	1
1.1	Background	1
1.2	Existing Systems	1
1.3	Motivation	2
2	STATE OF THE ART	3
2.1	What is blockchain?	3
2.2	Evolution of blockchain	4
2.2.1	1991 - Blockchain Invention	4
2.2.2	2008-2013 - Bitcoin Emergence	4
2.2.3	2013-2015 - Ethereum Development	4
2.2.4	2015 - Hyperledger	4
2.2.5	2015 - Future	5
2.3	Types of blockchain	5
2.3.1	Public Blockchain	5
2.3.2	Private Blockchain	5
2.3.3	Consortium Blockchain	6
2.3.4	Hybrid Blockchain	6
2.4	Consensus algorithms	6
2.4.1	Byzantine generals problem	6
2.4.2	Consensus mechanisms	6
2.5	Comparison between Hyperledger fabric and Ethereum	8
3	THE SOLUTION	9
3.1	Scenario	9
3.2	Why blockchain and fabric?	9
3.3	Use cases	10
3.4	Architecture	10
3.5	Activity diagram	12
3.6	Applying Fabric Components	14
3.6.1	Smart Contract & Chaincode	14
3.6.2	Endorsement policies	15
3.6.3	Endorsing members of hospitals	16
3.6.4	Specifying the endorsing policy	16
3.6.5	Ledger	16
3.6.6	Identity (CA)	17
3.6.7	Membership Service Provider (MSP)	18
3.7	Security Mechanisms	18
3.7.1	Private Collections	19
3.7.2	Data re-encryption	20
4	IMPLEMENTATION	23
4.1	Fabric SDK	23
4.2	Steps to use the application	23
4.2.1	Prerequisites	23

4.2.2	Start network	24
4.2.3	Add additional hospital	24
4.2.4	Bring up the backend and frontend	24
4.3	Workflow	25
4.3.1	Login Admin	25
4.3.2	Admin Dashboard	26
4.3.3	Create Patient	26
4.3.4	Create Doctor	26
4.3.5	Login Patient	28
4.3.6	View Patient Details	28
4.3.7	Grant/Revoke Access	28
4.3.8	Edit Personal Details	30
4.3.9	View History	31
4.3.10	Login Doctor and View Doctor Details	32
4.3.11	View Patients	32
4.3.12	Edit Medical Details	33
4.3.13	View History	34
5	RESULTS	35
5.1	Pros and Cons of using hyperledger Fabric	35
5.1.1	Pros	35
5.1.2	Cons	35
5.2	Issues in hyperledger	35
5.2.1	getHistoryForKey - Private Data Collecion	35
5.2.2	Create a user defined role instead of client	36
5.2.3	Access user attributes using client	36
5.3	Challenges in developing application	36
6	CONCLUSION AND FUTURE WORK	37
6.1	Conclusion	37
6.2	Future Work	37
	BIBLIOGRAPHY	39

LIST OF FIGURES

Figure 2.1	Block Structure [14]	3
Figure 3.1	Admin use case diagram	10
Figure 3.2	Patient Use case diagram	11
Figure 3.3	Doctor Use case diagram	11
Figure 3.4	Architecture	13
Figure 3.5	Creation of a patient	13
Figure 3.6	Creation of a doctor	14
Figure 3.7	Smart Contracts Hierarchy	15
Figure 3.8	Ledger in Hyperledger Fabric [13]	17
Figure 3.9	Private Data Collections	20
Figure 3.10	Re-encryption	21
Figure 4.1	Admin Loogin Screen	25
Figure 4.2	Admin Dashboard	26
Figure 4.3	Create Patient	27
Figure 4.4	Admin Dashboard	27
Figure 4.5	Create Doctor	28
Figure 4.6	Login Patient	29
Figure 4.7	New Password	29
Figure 4.8	View Patient Details	30
Figure 4.9	Grant/Revoke access	30
Figure 4.10	Edit personal details	31
Figure 4.11	View patient's history	31
Figure 4.12	Login Doctor	32
Figure 4.13	View Doctor Details	32
Figure 4.14	List of patients	33
Figure 4.15	View Patient Details	33
Figure 4.16	Edit patient's medical details	34
Figure 4.17	View patient's history	34

LIST OF TABLES

Table 2.1	Hyperledger fabric and Ethereum comparison	8
-----------	--	---

ACRONYMS

EHR	Electronic Health Record
EMR	Electronic Medical Record
HL7	Health Level 7
FHIR	Fast Healthcare Interoperability Resources
PoW	Proof of Work
PoS	Proof of Stake
RSM	Replicated state machine
pBFT	Practical Byzantine Fault Tolerance
tps	Transactions Per Second
CA	Certificate Authority
MSP	Membership Service Provider
SDK	Software Development Kit
HLF	Hyprledger Fabric

INTRODUCTION

1.1 BACKGROUND

The health care services received from various hospitals and clinics have become prevalent, due to the predominant increase of specialty in the health care services and patient's mobility. Doctors with information about the patient's medical history can make precise decisions about the patient's medical condition and treatment. A major problem that the health care services are facing now, is how to share the clinical data with various health care facilities while ensuring confidentiality, data integrity, and patient privacy.

1.2 EXISTING SYSTEMS

An EHR is an electronic version of a patient's medical details. In current times, EHR is used to share patients' medical records across various hospitals. EHR consists of health information of the patient in the form of Electronic Medical Record (EMR) [1]. EMR contains a patient's medical diagnoses, allergies, history, treatment, and laboratory reports[22]. The large amount of data in EHR can be used for machine learning and data analysis.

The healthcare IT standards that are used in EHR are Fast Healthcare Interoperability Resources (FHIR) and Health Level 7 (HL7) to transfer clinical data between various applications that are used by different healthcare providers.

The other models for interchange of medical data between health care providers are push, pull, and view.

- Push - The medical information is sent from one health care provider to another and the transaction cannot be accessed by any other party. In the U.S, a protected email standard named Direct is used for the encrypted transfer between sender and receiver. From the data creation till the data use, there is no assurance of data integrity. It is accepted that the sender produced an exact payload and the receiver precisely ingested the payload. It is done without a standard audit trail [10].
- Pull - Medical information from one health care provider may be queried by another health care provider.
- View - One health care organization can look into the patient's medical data from another health care organization's record[10].

1.3 MOTIVATION

Blockchain-based systems are a decentralized technology that is used in several industries such as logistics, supply chain management, finance applications and Internet of Things (IoT)[2]. Blockchain provides a secure distributed database and queries to the database can be made without any intervention of unauthorized identities. It is highly efficient in the case when various participants want to access the same database. Thus, blockchain can minimize a lot of resources and costs to access the same database securely. HLF is a permissioned blockchain system that helps in preserving trust among participants in the network using CAs and MSPs. Medical data storing and sharing is an integral part of healthcare systems. However, sharing personal data among various participants through unsecured means can lead to leakage of critical information. Also, the lack of client control over their personal information leads to harmful consequences such as unauthorized identities can access/edit the personal medical details. And during sharing of the patient details can lead to even more risks[11]. The critical issue in the electronic health/medical records (EHR/EMR) is maintaining the interoperability among various involved identities. This issue may cause obstacles in the data transaction among each other. There is a lack of data management and sharing mechanism among the identities which leads to fragmentation of the healthcare information. Apart from interoperability, data security and privacy are also challenges in the current ways of data storing and sharing data through EHR/EMR systems[17]. Sharing and storing patient's data has a lot of liabilities due to data leakage and potential shortcoming in security mechanisms. Blockchain for healthcare can ensure the security of the personal and medical information of the patients and can make sure that only authorized identities can access/edit the data using smart contracts which enables specific features among various identities in the system [25]. Therefore, there is a clear need for a distributed way of sharing and store data where patients are more sure about their data security and privacy and in addition, all the involved identities can see the holistic view of overall transaction and interactions [12].

STATE OF THE ART

2.1 WHAT IS BLOCKCHAIN?

Blockchain is a decentralized, trustworthy distributed ledger on a peer to peer network that consists of a list of chronologically ordered blocks. Every block consists of a hash of a previous block and hence forming a chain. The first block in the blockchain is the genesis block and the block before a given block is called as its parent block [22].

A block contains a header and body. The block header consists of

- Version: Specifies the block validation rules.
- Previous block hash: This is to ensure that the previous block cannot be changed, without changing the current block header.
- Timestamp: The current block creation time.
- Merkle root hash: The Merkle root is obtained from the hash values of all the transactions present in the block. This is to ensure that the transactions cannot be modified without changing the header.
- Nonce: A nonce is a 4 byte unique number that is used only once in the communication.

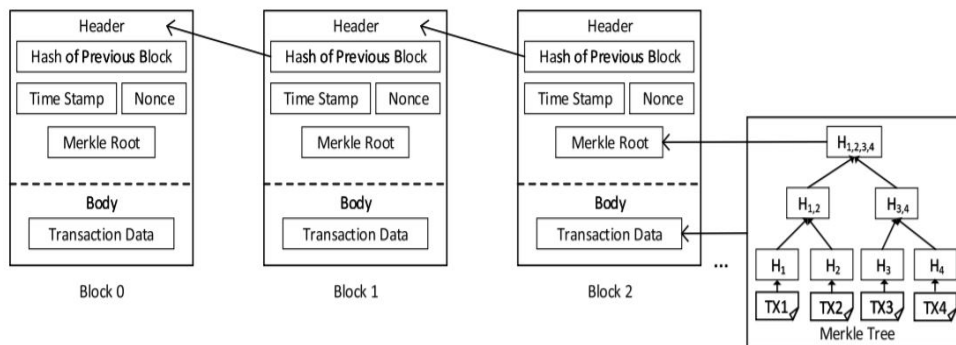


Figure 2.1: Block Structure [14]

The core components of blockchain architecture:

- Node/Peer: A device (like a computer) present inside a blockchain network that has a copy of all the transactions.
- Transaction: It is the exchange of information between the two blockchain addresses.

- **Block:** It is used to keep a group of transactions that is distributed among all the nodes in the network.
- **Chain:** A series of blocks arranged in chronological order.
- **Miners:** Before adding any transactions to a new block node, selected nodes carry out the block verification.
- **Consensus:** It is a mechanism through which all of the blockchain network's peers come to an agreement about any transaction in the blockchain.

2.2 EVOLUTION OF BLOCKCHAIN

The mainstream internet usage at the turn of the century aided the introduction of digital currency as an expansion of electronic cash systems. The following is a timeline of blockchain's growth.

2.2.1 1991 - Blockchain Invention

Scott Stornetta and Stuart Haber invented Blockchain. They created a cryptographically protected chain of blocks which would prevent anyone from tampering with document timestamps [3].

2.2.2 2008-2013 - Bitcoin Emergence

Blockchain technology gained popularity due to Bitcoin in 2008, as it is the first application in the blockchain. Bitcoin was conceptualized by Satoshi Nakamoto. In 2009, Nakamoto published a white paper on Bitcoin. First Bitcoin was purchased for 10,000BTC in 2010. Bitcoin crosses \$1 billion in 2013.

2.2.3 2013-2015 - Ethereum Development

Ethereum was conceptualized by Vitalik Buterin. Ethereum has additional features like smart contracts compared to Bitcoin. The development of Ethereum has proven to be an important moment in the history of Blockchain. Ethereum is used for cryptocurrencies and in many other decentralized applications because of smartcontracts.

2.2.4 2015 - Hyperledger

Hyperledger was developed by Linux Foundation in 2015. It allows development of open-source blockchain. The different Hyperledger frameworks are Hyperledger Fabric, Hyperledger Iroha, Hyperledger Sawtooth, and Hyperledger Burrow [5].

2.2.5 2015 - Future

Many cryptocurrencies and applications have been developed after the emergence of Bitcoin and Ethereum. Nowadays, Blockchain technology is used by various companies and organizations.

2.3 TYPES OF BLOCKCHAIN

Blockchain is divided into four types, public, private, consortium and hybrid blockchains based on permission and assess.

2.3.1 Public Blockchain

It is a permissionless distributed technology. As the name suggests, anyone with the internet is allowed to access the public blockchain and participate in the transactions. Each peer in this network has their own copy of the ledger. Consensus mechanisms like Proof of Work (PoW), Proof of Stake (PoS), etc., are required to reach consensus while adding new blocks and during verification of transactions. Public blockchains are fully decentralized because all the peers have equal authority, which in terms makes it secure as no one can have full control over the blockchain network. In turn, ensuring data security helps in keeping the ledger immutable. Some examples are Bitcoin, Litecoin, and Ethereum.

Open and closed are used to tell who can read data in a blockchain. These two properties are used to describe blockchain.

- Public and Open: This blockchain is accessible for everyone and data can be read by everyone. For example, cryptocurrencies like Bitcoin and Ethereum.
- Public and Closed: The consumers can store confidential information and can restrict access for selected participants. In this blockchain, everyone can write in blockchain, but the reading rights are only held by selected entities. For example, Voting.

2.3.2 Private Blockchain

It is also known as permissioned blockchain. There are limitations on who can be part of the network and who can contribute to the transactions. A private blockchain is used by an organization or company for its internal usage. Private blockchains are centralized i.e, one organization has authority in the network. It provides transparency and security to the participants. For example, Hyperledger Fabric.

- Private and Open: Every contributor in the blockchain network can read the data. For example, Corporate income statements.

- Private and Closed: Only known participants can read and write in the blockchain network. For example, Tax returns, National Defense.

2.3.3 Consortium Blockchain

In this blockchain, more than one organization can manage the network, hence it is not fully centralized. It is used by organizations that need both public and private blockchain functionality. For example, Quorum and Hyperledger Fabric.

2.3.4 Hybrid Blockchain

It is a mix of the private and public. In the blockchain, the peers determine who has access to which data. Some processes are kept private, while others are made public. On an open ledger, businesses can protect background transactions with business partners while still providing the product details to customers. For example, Dragonchain.

2.4 CONSENSUS ALGORITHMS

All the peers have equal authority, hence it is difficult to reach consensus. Before Bitcoin, a lot of decentralized currency systems failed because when it came to reaching a consensus, they were unable to resolve the most important problem, i.e., Byzantine Generals Problem.[\[22\]](#)

2.4.1 Byzantine generals problem

The generals of various army battalions must choose whether to withdraw or assault. They must make a decision by sending messages by messengers and must reach an agreement. There's a possibility that some of the messengers and/or generals are traitors. These traitorous generals can change the message and send a malicious response, causing the loyal generals' plan to be disrupted.

2.4.2 Consensus mechanisms

The following are the few consensus mechanisms that can be used to address the issue of the Byzantine generals problem:

2.4.2.1 Proof of Work

Bitcoin uses PoW as the consensus mechanism. Miners "mine" a block to connect to the blockchain by solving cryptographic puzzles. This method requires a significant amount of energy and computation. The puzzles have been created in such a way that they are challenging and demanding on the system. When a miner completes the puzzle, they send their block for

verification to the network. The method of determining whether a block belongs in the chain or not is extremely easy. A 51% attack is a possible attack in the blockchain, in which a miner or a group of miners with more than 51% of the computing power will prevent new blocks from being produced and establish false transaction records that benefit the attackers [22].

2.4.2.2 *Proof of State*

It is a more energy-efficient version of PoW. Nodes with the most stakes (for example, currency) are thought to be less likely to strike the network. The disadvantage here is the monopoly, when it comes to both technological and economic aspects of the scheme, the main stakeholder has full influence and authority.

2.4.2.3 *RAFT*

RAFT is a distributed crash fault tolerance consensus algorithm that ensures that the system can make a decision and process client requests in the event of a failure [23]. It is a consensus algorithm for handling a replicated log. Consensus in RAFT is reached through an elected leader. The server within a raft cluster is either a follower or a leader or a candidate. The leader is in charge of replicating logs to the followers.

2.4.2.4 *Practical Byzantine Fault Tolerance (pBFT)*

Hyperledger fabric uses pBFT. It is a realistic algorithm that tolerates Byzantine faults. 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. Without performing complex mathematical computations (such as PoW), pBFT will achieve distributed consensus. Since they have been decided upon, the transactions do not need several confirmations. (For example, in Bitcoin's PoW, each node independently verifies all the transactions. Confirmations will take anywhere from 10 to 60 minutes depending on how many individuals have to confirm the new block).

2.5 COMPARISON BETWEEN HYPERLEDGER FABRIC AND ETHEREUM

	Hyperledger Fabric	Ethereum
Blockchain Type	Permissioned and Private	Permissionless and Private / Public
Objective	Suited for enterprises and B2B applications	Suited for B2C applications
Cryptocurrency	None	Ether
Smart contract language	JavaScript, GO and Java	Solidity
Consensus Mechanism	Pluggable consensus mechanism. Mining not required	PoW consensus mechanism. Mining is required to reach the consensus.
Confidentiality	Confidential transactions	Transparent
Transactions Per Second (tps)	> 2000 tps	≈ 20 tps

Table 2.1: Hyperledger fabric and Ethereum comparison

THE SOLUTION

3.1 SCENARIO

To manage the medical records of patients, need to map fabric components to requirements of the EHR systems. All hospitals act as organizations in fabric network. Patient data has been treated as assets which is stored in the ledger. It is also possible to store the reference of the EHR data in the ledger but since application is not managed by real data and for that it will be necessary to maintain the separate database which will have patients data. This can be a good solution when it is integrated with production or real hospitals EHR data. For now patient record has few fields like personal and medical details like age, address, allergies, symptoms, treatment, followup, etc. When doctor is medicating to a patient, patient history data will be available which helps doctors to assign appropriate treatment. To improve the privacy of records, it is designed to provide extra steps in application for patients. Patient can decide to have permission to access his/her data from a particular doctor. Doctor can view limited fields of assent means patient data like all medical fields along with age and allergies. Where as patient can view all the fields but edit only personal fields. A similar application approach also available in [7] which motivated to this solution further.

3.2 WHY BLOCKCHAIN AND FABRIC?

Blockchain has many advantages over many traditional database systems. One of the major advantages of blockchain is, it stores data cryptographically encrypted via distributed and decentralized way in peers which is nothing but a computing machine. This solves the availability of data within the network. It means for the outside world it is not possible to access data. The next benefit of blockchain is immutability. Since all the transactions are combined in a block and arranged in a chain as explained in Chapter 2.

On top of the blockchain, hyperledger fabric provides some additional features which perfectly fit the current scenario. It is a permissioned and closed blockchain, no person can get added into the network unlike permissionless blockchains such as bitcoin, ethereum. This solves the confidentiality problem of patients' data. The fabric has a concept of Certificate Authority for every organization or common CA called Fabric CA and MSP which provides identities and verifies when a transaction request is made. Moreover, all the components of the fabric network are scalable and pluggable. It means more organizations can be added connected through channels, any number of peers for organizations. Like ethereum, the fabric also provides the concept of smart contracts, and when it is packaged and deployed it is

called a chaincode. It is a business logic that is deployed on every endorsing peer. Initial organizations of the channel decide whether the new organizations should be added or not through Channel Configuration. It returns all the transaction information. All the components are explained further in the following section.

3.3 USE CASES

There are three users in the system with role admin, patient, and doctor. Each user has its own use cases which are described in Figures 3.1, 3.2 and 3.3. A user interfaces for admin from every hospital to make necessary changes about users.

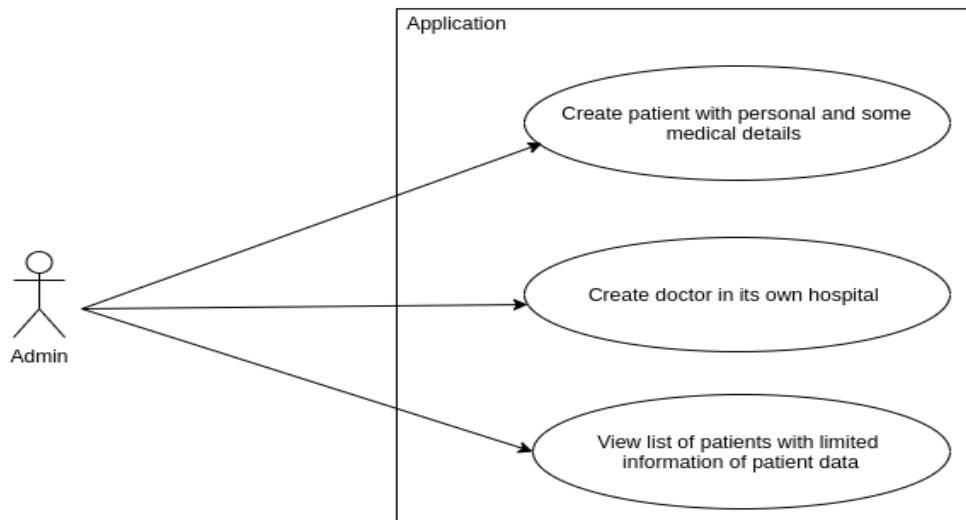


Figure 3.1: Admin use case diagram

3.4 ARCHITECTURE

A high-level architecture of a system is depicted in Figure 3.4. The architecture also shows the fabric network in detail with color schemes. The blockchain operator setups the initial configuration of the network and provides necessary access and credentials to the users who govern the system. All the peers from the hospitals are nothing but docker containers of image hyperledger/fabric-peer, orderer, and ca. As described, every component of the application is pluggable, so backend code and smart contract are written in JavaScript language with ExpressJS as a server to provide REST API. The user interface is developed using the Angular 11 framework. The communication between the front and back end is performed through REST calls with JSON web token for authentication. Backend code can also be written in Java, Go, and Typescript which is officially supported languages by Hyperledger Fabric. For the given scenario single channel named 'hospitalChannel' and two hospital organizations are sufficient. The third hospital can also be suc-

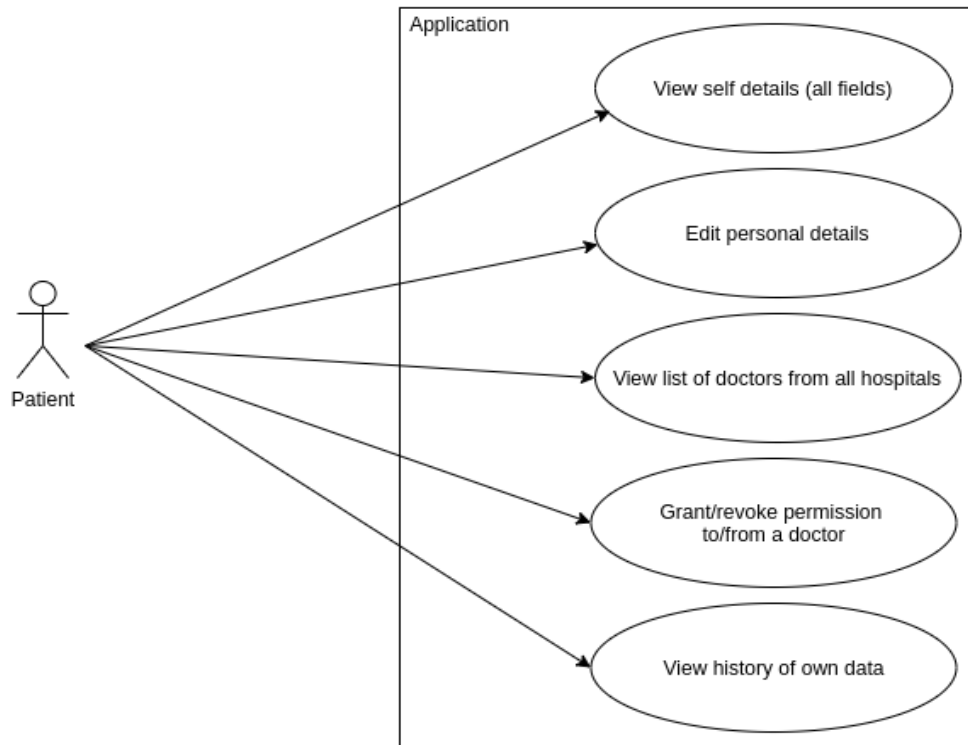


Figure 3.2: Patient Use case diagram

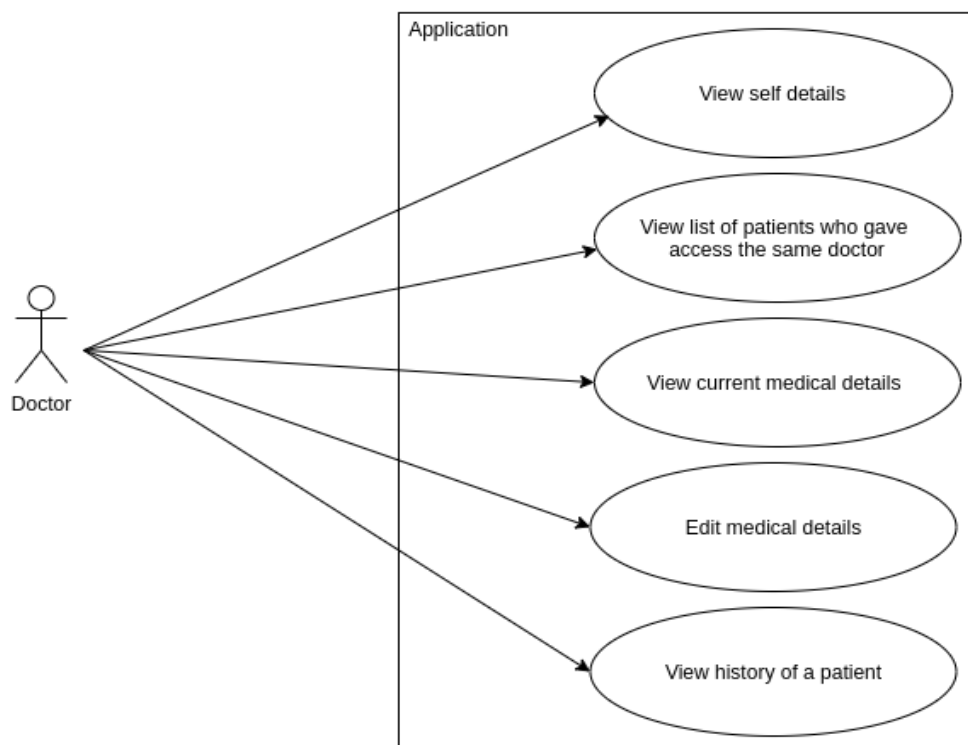


Figure 3.3: Doctor Use case diagram

cessfully added in between when the network is running and gets joined with the same channel. Fabric provides support for two databases, LevelDB and CouchDB. The CouchDB is used for the solution because it is more flexible than LevelDB. Images handling is possible. It supports indexes, unlike LevelDB. Since all patient data stored in CouchDB and not maintaining any separate EHR store, CouchDB fits well. Whereas LevelDB is a powerful in-memory database developed by Google to store key-value pairs and in some cases, it is faster than CouchDB. The other use case scenario where the EHR database of hospitals is used and only stores references (maybe in the form of API) to that record in LevelDB as a blockchain ledger. Since use cases are simple here so CouchDB supports here properly. The ledger is a combination of the transaction log and world state. CouchDB is used to store the world state. Because of which no need to query the whole transaction log for every transaction request. Transaction log stores all transactions start from the first one which is stored in the genesis block. It can be found on local system at `/var/lib/docker/volumes/net_peer0.org2.example.com/_data/ledgersData/chains/chains/mychannel/blockfile.000000` (depends upon the installation path). In Couchdb docker containers it is by default available at `/var/hyperledger/production/ledgersData/chains/chains/mychannel/`. It is configurable in `docker-compose-hospital-net.yaml`. The Redis key-value DB is also used to store doctor credentials - username and password for doctor's credentials. Whereas other details of the doctor are stored as user attributes using fabric SDK. The patient record has some personal and medical fields describe here in `initLedger` file. This is the initial data when the network gets up. Similarly, two doctors from every two hospitals are created.

3.5 ACTIVITY DIAGRAM

The Figure 3.5 shows the interaction between components for creation of a patient. The patient needs to create an account only the first time the patient visits any one of the hospitals in the network, during the first visit the patient provides details to the admin, admin invokes the AdminContract to create a patient. In the backend, firstly the admin certificate is used to connect to the network, and a transaction is created which adds the patient object to the ledger and also adds the patient identity to the blockchain network. After the creation of the patient is successful, the server creates a temporary password for the patient using which the patient can log in to the network. The patient credentials are added to the ledger as the patient can go to any of the hospitals in the network. The Figure 3.6 shows the interaction to create a doctor. The doctor provides details to the admin, who then firstly connects to the network and then adds the doctor as an identity to the blockchain network. And the credentials of the doctor are stored in the hospital-specific Redis database. Now as patient and doctor are added as identities in the blockchain network, for the next interaction the patient/doctor can connect to the network using their certificates.

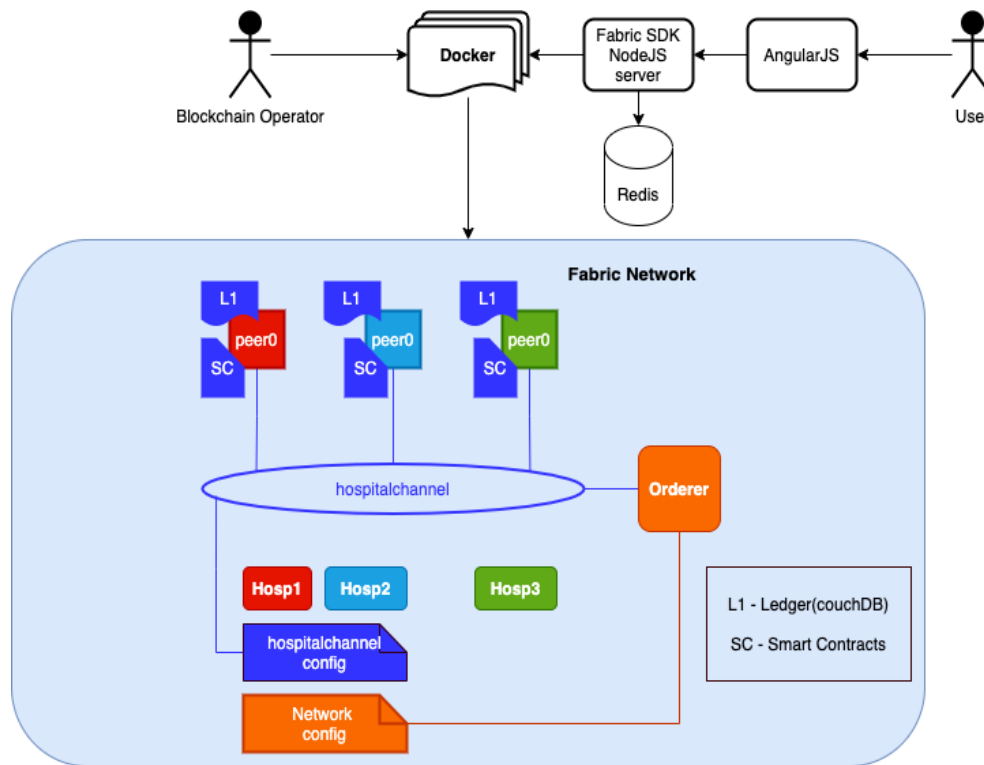


Figure 3.4: Architecture

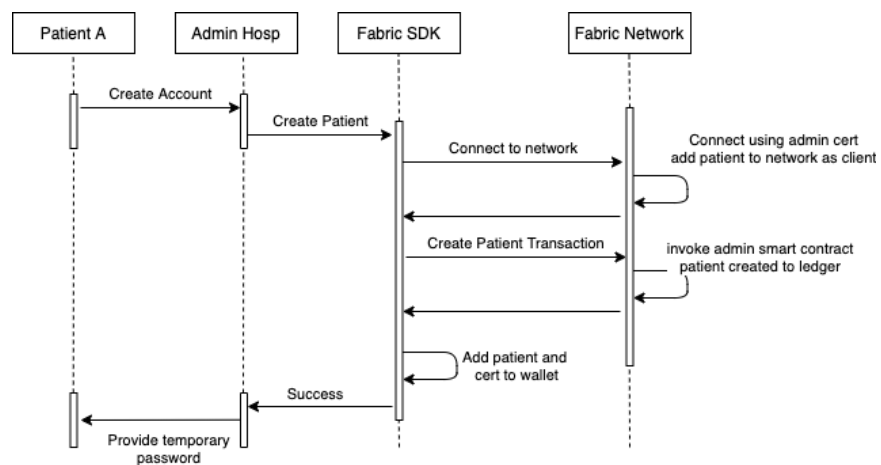


Figure 3.5: Creation of a patient

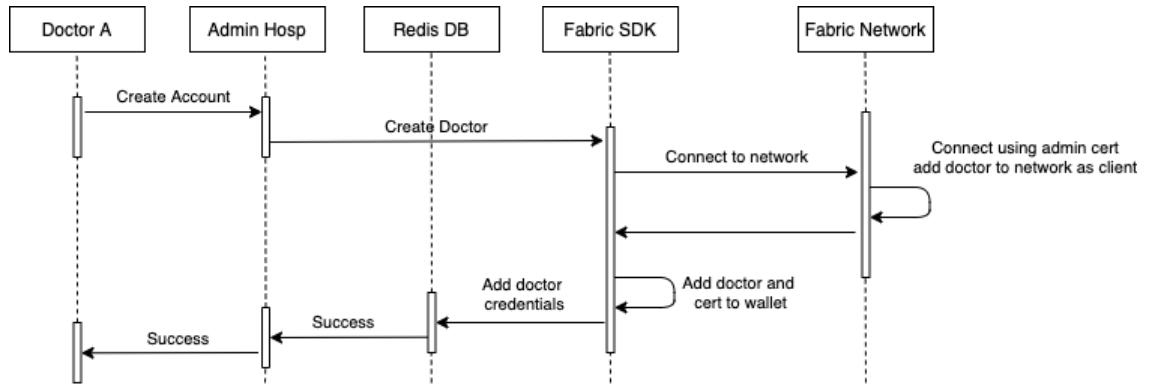


Figure 3.6: Creation of a doctor

3.6 APPLYING FABRIC COMPONENTS

The HLF comprises many components, has a complex architecture. And these components can be applied and used in many various ways. The following describes how HLF components are applied.

3.6.1 Smart Contract & Chaincode

A smart contract in hyperledger fabric defines the transaction logic that controls the lifecycle of a business object contained in the world state [24]. Multiple Smart contracts are packaged into a single chaincode which is then deployed to a blockchain network. A smart contract defines the rules between different organizations in executable code. Fabric SDK invokes a smart contract to create transactions which then performs changes to the ledger. A smart contract is defined within a chaincode. Multiple smart contracts can be defined within the same chaincode. When a chaincode is deployed, all smart contracts within it are made available to applications. As shown in Figure 3.7 the application comprises mainly three smart contracts packaged into a single chaincode - each role (Patient/Doctor/Admin) invokes its very own smart contract.

- **AdminContract** - This contract is invoked by the admin. The methods in the contract give the admin the ability to create/delete patients by adding/deleting patient object to from the ledger. Admin can also view all the patients available throughout the network.
- **PatientContract** - The patient interacts with the ledger by invoking this contract. The contract contains the logic that is required for the patient. For instance, only the patient can update/view the personal details and password via the methods defined in the contract. And also patient contract contains methods to grant and revoke access to from a doctor.
- **DoctorContract** - The doctor contract has methods that allows the doctor to update/read the patients medical details.

The ideology for the three smart contracts is to make sure that the right role has the right access to the data on the ledger. Only the patient has the right to update the personal details, grant/revoke access, and update the password. These methods can be invoked by neither the admin nor the doctor. And similarly, the methods of the doctor cannot be accessed by the patient or admin. The read patient method is common for all, but the data which is retrieved by the contract is different for each role admin just has the access to patient name, the doctor has the access to the medical details and the patient has the access to the whole object.

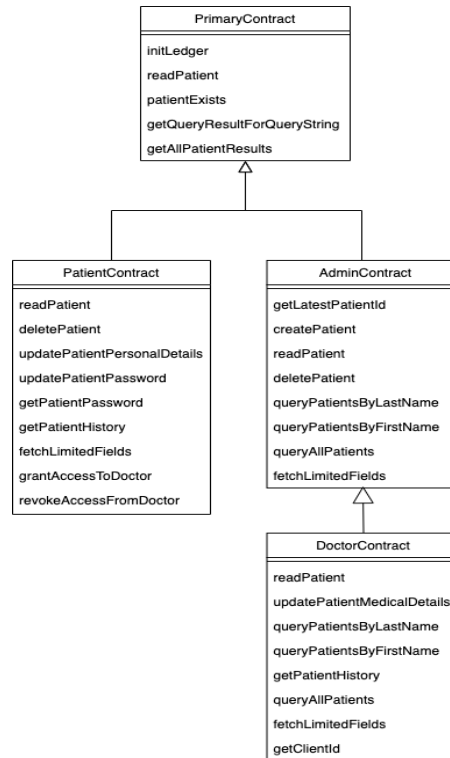


Figure 3.7: Smart Contracts Hierarchy

3.6.2 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 sure that the transaction contains the the required number of endorsements which is configured in the endorsement policy.[18].

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. A Chaincode-level endorsement policy is defined in the YAML file in the PATH Application: `&ApplicationDefaults/Policies/Endorsement 'MAJORITY'` specifies that only when a majority of channel members approve a chaincode definition then definition is committed to the channel.

3.6.3 Endorsing members of hospitals

In path `&hosp1/Policies/Endorsement` the rule `OR('hosp1MSP.peer')` requests one signature from the peers of `hosp1`. Other values can be :

- Requests one signature from each role
`AND('hosp1MSP.peer', 'hosp1MSP.admin')`
- Requests one signature from either one of the roles
`OR('hosp1MSP.peer', 'hosp1MSP.admin')`

3.6.4 Specifying the endorsing policy

In path `&ApplicationDefaults/Policies/Endorsement` other types can be a signature with the one of the following rules:

- A transaction is committed only if admin of the both the hospital approve.
`AND('hosp1MSP.admin', 'hosp2MSP.admin')`
- A transaction is committed only if either one of the admin of the both the hospital approve.
`OR('hosp1MSP.admin', 'hosp2MSP.admin')`
- `OutOf(1, 'hosp1MSP.admin', 'hosp2MSP.admin')` which is equivalent to `OR('hosp1MSP.admin', 'hosp2MSP.admin')`

3.6.5 Ledger

A Distributed ledger in Hyperledger Fabric as shown in Figure 3.8 - one that is logically singular, but has many consistent copies distributed throughout a network comprises of two parts - the world state which holds the current values of the objects, the other part is the blockchain which is the history of the transactions which resulted to the current world state[13].

In the application, the ideology of a ledger can be thought of as one logical database in a Hyperledger Fabric network. In reality, the network contains multiple copies of a ledger for each hospital's peers – which are kept consistent with every other copy through consensus. The ledger in the application is being utilized in the following ways

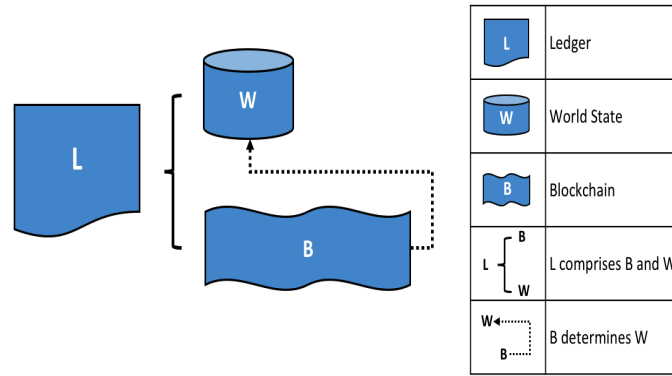


Figure 3.8: Ledger in Hyperledger Fabric [13]

- Firstly, the world state is the database where all the patient data is stored. All patients that are queried and updated are stored in the world state. All the query transactions are retrieved from the world state.
- Secondly, the transaction log which is a sequential structured inter-linked blocks of log files, where each block contains a sequence of transactions, and a transaction represents a query or an update to the world state. The `getHistory` API specifically uses the transaction log to retrieve the history of an asset.

3.6.6 Identity (CA)

For a participant of a hospital, the participants first need to prove that can be are trusted to make transactions in the blockchain network. An identity for each participant issued by a trusted authority is required to be recognized in this network [9]. The MSPs are the trusted authorities - which can be found in `configtx.yaml`. An Hospital Certificate Authority(CA) dispenses certificates to each of its participants. These certificates are digitally signed by the CA and bind together the participant with the participant's public key with a set of permissions. As a result, if one trusts the CA (and knows its public key), it can trust that the specific participant bound to the public key included in the certificate by validating the CA's signature on the participant's certificate. In the application either CAs(default) or cryptogen can be used to issue identities in the network:

- *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[4]. The configuration files for all the hospitals and the orderer are defined in `cryptogen` directory, the YAML files contain the necessary information for the tool to create the crypto material for the hospitals and the Orderer.

- **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 of the hospital share the same trust of the root CA [9]. The application uses CAs to allows registration and enrolling patients/doctors with the Fabric SDK which cannot be done using Cryptogen. The configuration files of the CAs of the hospitals and orderer defined in **Hosp1CA**, **Hosp2CA** and **OrdererCA** respectively.

3.6.7 Membership Service Provider (MSP)

Hyperledger Fabric is a permissioned blockchain network, all the participants of the network, in our application the peers/doctors/patients need to prove their identity to join and perform transactions in the blockchain network to prove that these participants can be trusted. Hyperledger Fabric uses a Public Key Infrastructure (PKI) [16] to verify identities in a chain of trust. The fabric uses Certificate Authorities(CA) to prove identity, the CAs generate a public and a private key for each identity which forms a key-pair using which the participants can prove their identity and be recognized in the network. The MSP is the one that verifies the private keys of the participants by matching them with the stored public key. For instance, in the application the peers of the hospital endorse a transaction using its private key, the MSP on the ordering service is where the public key is stored which is used to verify that the private key attached to the transaction is valid. CAs are used to create trusted identities that are recognized by the network(peers/doctors/patients). MSPs are used for defining the hospitals that are trusted by the network members. MSPs are allocated the participants of the hospitals with a set of roles and permissions within the network. In the applications, all the hospitals that join the network can perform read/write transactions in the network. These permissions are configured in **configtx.yaml** and for participants of these hospitals to perform a transaction in the network they first have to have an identity issued by a CA that is recognized in the network. And be a member of any of the hospitals - MSP is how the participants are linked as a member to the hospital, this is when the public key of the participant is stored in the hospital MSP.

3.7 SECURITY MECHANISMS

Security of the data is essential when question arises for patient data in medical data. Blockchain concept is very safe and secure itself and provides a mechanism of chaining in which if any transaction changes all successive blocks needs to change which is a very heavy task. In addition to that in hyperledger fabric when new peer and or organizations joins the channel approved by in channel configuration, still it should not be exposed, as it is available for all kind of peers. To avoid that data can be encrypted or hidden based on patient's grant/revoke actions for doctors. The proof of

concepts of two mechanisms are described further. Unfortunately it is not yet implemented due to some unavoidable reasons.

3.7.1 *Private Collections*

The data used and stored in Healthcare is highly confidential and must be secure. The data of the patient must be private to only the hospitals or doctors that the patient has allowed access to. Hyperledger Fabric contains functionality to store private data using private data collections[20]. Private data is the data that must be kept private from other hospitals or doctors. A private data collection contains two elements:

- *The actual private data:* Only the data of the private collections that are authorized for particular hospitals can be accessed.
- *A hash of that data:* Patient data for which the hospital does not have the authorization, this data is hashed and stored and cannot be accessed by the member of the hospital.

In this application, there are $n! + 1$ number of private data collections where n is the number of hospitals. For instance, if there are 3 hospitals, there will be 7 private data collections. There will be a private data collection for each combination such as there will be a private collection for each hospital and a private collection which will be shared by any two hospitals and one private data collection shared by all the hospitals. The patient data stored on the collection depends on the doctors to which patient has authorized. If the patient has authorized to only one hospital, this will be stored in the private collection of that hospital. Now if the patient grants access to a doctor of another hospital, the private data is moved from the current collections to the collection which is private to the two hospitals. There will be no duplicate of the data. The patient data will always be in any one of the private data collection. The same mechanisms are applied when the patient revokes access from a hospital.

3.7.1.1 *Implementation Example*

Let us consider if there are two hospitals named Hospital 1 and Hospital 2. There will be a total of 3 private data collections named `Hosp1PrivateCollection` which contains patient data which is authorized only for Hospital 1 similar `Hosp2PrivateCollection` and a collection named `Hosp12PrivateCollection` which is private to both Hospital 1 and Hospital 2, in this collection the patient would have granted access to both the hospitals. The channel world state will contain the `assetId` and the collection in which the asset is stored. This makes the reading of the patient data faster.

3.7.1.2 Scenarios

- *Read/Update Patient*: First the patientId is read on the world state which will give the private data collection name, and then the patient data is read/updated from that private collection.
- *Grant/Revoke access to a new Hospital*: For instance if a patient is present in Hosp1PrivateCollection and this patient grants access to Hospital 2, The data is first copied into Hosp12PrivateCollection and then deleted in Hosp1PrivateCollection.
- *Get History*: The private data collections has not been implemented yet in the application due to the work in progress. Read the section for more details in Chapter [5.2.1](#)

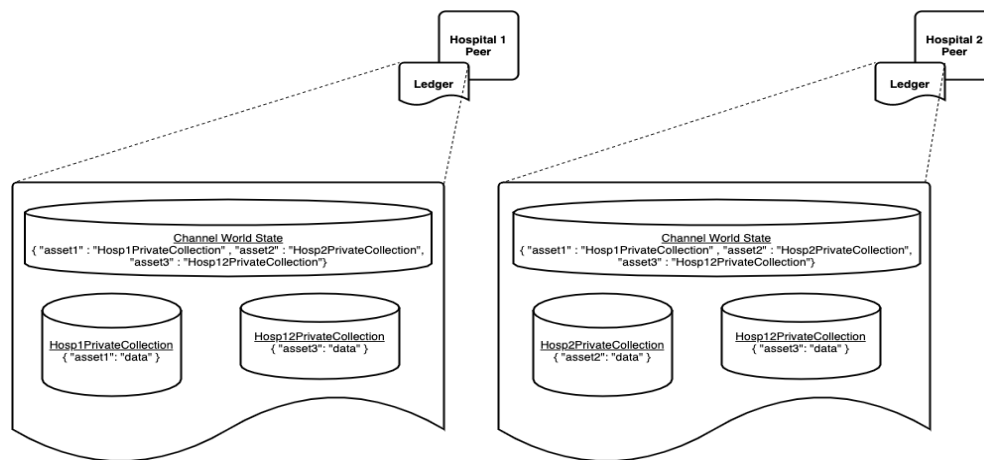


Figure 3.9: Private Data Collections

3.7.2 Data re-encryption

It is a very common technique to store data in an encrypted format and decrypt it whenever is required. There are two types of encryption - Symmetric and asymmetric encryption. In the symmetric encryption technique, a common key is used to encrypt and decrypt the data. Whereas two different keys are used in asymmetric cryptographic encryption methods which are called private and public keys. It is also called public-key cryptography. RSA (Rivest–Shamir–Adleman) is a common algorithm used by modern computers to encrypt and decrypt data. typical use case, user A sends data to user B by encrypting data using user B's public key. Public keys are publically available for usage. To prove the authenticity of a public key, it is signed by some authority using a digital signature. Certificate Authority is an entity that issues a digital certificate. A digital certificate certifies the ownership of a public key of a user or an organization. So when user B receives data, it is decrypted by its private key. In the reverse mechanism, i.e., data en-

rypted by a private key and decrypted by the public key used in case of authentication a user.

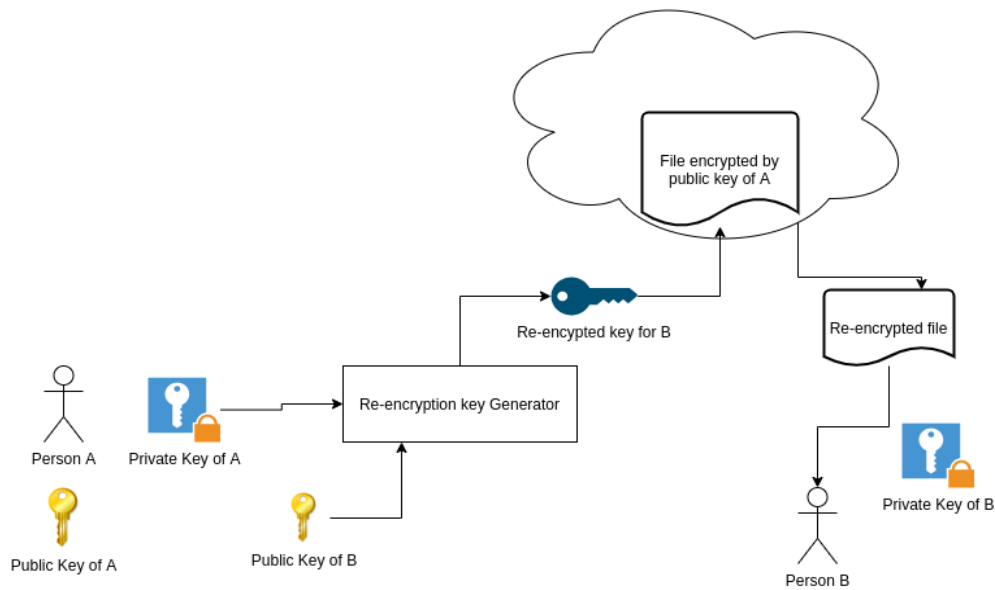


Figure 3.10: Re-encryption

Consider the scenario depicted in Figure 3.10. Person A has a data file on the cloud and person B wants to access the data. So typically, person A has to download the file, encrypt it with person B's public key, and then person B can decrypt it with its private key. These steps have some limitations. Person A needs to download the file and do encryption which is not feasible as the file size can be large and after encryption person needs to upload. So in the re-encryption approach, person A generates a re-encryption key from its private key and person B's public key, so it is a combination. This key sends to person B and the person re-encrypts the cloud file. The file is already encrypted by person A's public key but it encrypts again with the re-encryption key so the name is that way. After re-encryption, person B decrypts the file using its private key [26]. Now even if the re-encryption key is lost or someones else found it, still at the end it needs to be decrypted by person B's private key. So it is secure. The same approach can be applied in this scenario. The patient record would look as follows:

```

1 {
2   "patientId": "p1",
3   "password": hash(pwd),
4   "pwdTemp": true
5   "firstName": "abc",
6   "lastName": "xyz",
7   "data": encrypted patient data using symmetric key,
8   "changedBy": "doctorId XX",
9   "permissionGranted": [doctorId1: re-encrypted key for doctor 1,
                          doctorId2: re-encrypted key for doctor 2, ...],

```

```
10 "encryptedSymmetricKey" : "#####"  
11 }
```

All personal and medical fields are encrypted by some randomly generated key which is used as a symmetric key and stored in data as shown above. The symmetric key is also encrypted by the patient's public key and stored in the ledger under 'encryptedSymmetricKey' field. The reason behind using both encryption methods is, that a symmetric key is mostly recommended to encrypt large data. So when a patient provides access to a doctor then the re-encrypted key is calculated and stored with doctorId in the 'permissionGranted' field. So that when the doctor reads the ledger, if that doctorId is present then that re-encrypted key can be used to re-encrypt the 'encryptedSymmetricKey' and then decrypt using the doctor's private key. Once a doctor gets a symmetric key then it can easily be used to decrypt data. So basically data is encrypted only by symmetric but a re-encryption mechanism is applied to hide the symmetric key so it makes the mechanism quite robust.

IMPLEMENTATION

4.1 FABRIC SDK

The Hyperledger Fabric Client SDK 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 [8]. The Fabric SDK provides the following packages:

- `fabric-ca-client` - The `fabric-ca` provides APIs to register and to enroll participants(admin/patient/doctor) to establish trusted identities on the blockchain network. The package creates a new CA client for interacting with the CA server of the hospital to register and enroll the participants.
- `fabric-common` - encapsulates the common code used by all `fabric-sdk-node` packages supporting fine-grain interactions with the Fabric network to send transaction invocations. Provides APIs to monitoring events, logging and configuration settings of the environment variable, program arguments, in-memory settings
- `fabric-network` - This package contains the APIs required to connect to the Fabric network, submit transactions to query or edit the ledger. 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 package `fabric-network` - 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/user is a member. [8].

4.2 STEPS TO USE THE APPLICATION

This section describes very detailed steps to run the application start from the prerequisites. The project is available on **GitHub**

4.2.1 Prerequisites

The system should have installed some tools such as docker, docker-compose, node, npm, go, curl. Follow the detailed steps on the official website of fabric for the specific version and commands [19].

Download the fabric basic framework which generates platform-specific binaries and Docker images [6]. Execute the following curl command.

```
curl -sSL https://bit.ly/2ysb0FE | bash -s -- 2.2.2 1.4.9
```

This generates all test network code in folder 'fabric samples'. Now clone this project's git repository. Copy bin directory from 'fabric samples' to blockchain-hyperledger-fabric-electronic-patient-records/app.

4.2.2 *Start network*

Go to the first network directory and execute the network.sh file with argument up.

```
./network up
```

This command brings up all docker containers required for two hospital organizations.

```
./network createChannel
```

This command creates a channel 'hospitalChannel' on which both organizations are connected.

```
./network deployCC
```

This command packages the smart contract code inside 'patient-asset-transfer'. The initial data of six patients are created in a ledger. The folder structure looks similar to fabric samples which helps a developer who is already worked with the test network.

4.2.3 *Add additional hospital*

In addition to the existing two hospital organizations, one can also add a new hospital organization called hosp3. Go inside 'addHosp3' of the first-network directory.

```
./addHosp3 up
```

This command brings up the containers required for hospital3 to work and join 'hospitalChannel'. If there are multiple channels in the network, it is mandatory to specify which channel to connect to.

```
./addHosp3 deployCC
```

This command installs chaincode on hosp3 nodes. Hosp3 added separately to show that framework provides pluggable architecture. On scaling the network, new organizations can be easily added without down the existing network. Still, this solution is needed to be improved to support this pluggable feature more mature. Few things are hardcoded in the backend and front end.

4.2.4 *Bring up the backend and frontend*

Go to app/server.

```
npm install
```

This command installs all dependency files required for the backend server to run.


```
npm start
```

This command starts the express js backend server. It also enrolls and registers few users from the network. It includes both admin users from hospital organizations, initial six patient data, and four doctors. The users are created inside the wallet. Now go to app/client

```
npm install
```

```
ng serve -o
```

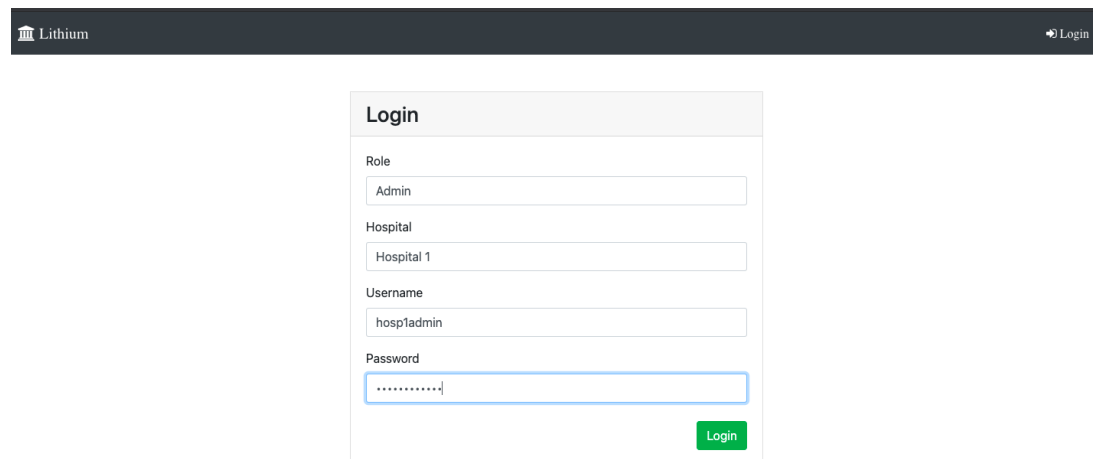
This brings up the angular server and automatically opened up in the default browser.

4.3 WORKFLOW

This section describes the working of the application through the perspective of all users - admin, patient, and doctor.

4.3.1 *Login Admin*

An admin is created for each hospital that joins the network. The details of the admin must be present during the addition of the hospital to the network. The admin details (username and password) are present in the fabric-ca-server-config.yaml of the respective hospital CA configuration. The login page for the admin is as shown in Figure 4.1 the role admin has to be chosen, as the web application is the same for all the participants, the admin must choose the hospital and enter the credentials. These credentials are checked against the credentials stored in the blockchain network which was configured in the Hospital CA config YAML file.

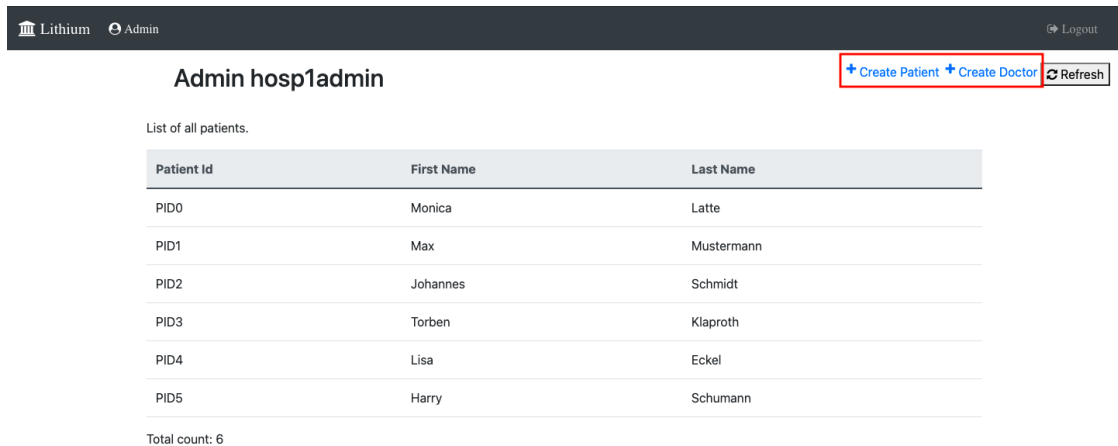


The screenshot shows the 'Login' page of the application. At the top, there is a dark navigation bar with the 'Lithium' logo on the left and a 'Login' link on the right. Below the navigation bar, the 'Login' form is displayed. The form has a title 'Login' and four input fields: 'Role' (with 'Admin' selected), 'Hospital' (with 'Hospital 1' selected), 'Username' (with 'hosp1admin' entered), and 'Password' (with masked characters). A green 'Login' button is located at the bottom right of the form.

Figure 4.1: Admin Loogin Screen

4.3.2 Admin Dashboard

As soon the admin log in successfully, a dashboard is displayed which contains the list of all the patients in the network and the functions highlighted in Figure 4.2. The patient list is retrieved by invoking the admin contract through which a transaction is created in the ledger to retrieve all the patient objects in the world state. The admin has access only to the names of the patient other details are restricted by the contract.



The screenshot shows the 'Admin hosp1admin' dashboard. At the top, there is a header bar with 'Lithium Admin' on the left and a 'Logout' button on the right. Below the header, the title 'Admin hosp1admin' is displayed. To the right of the title are two buttons: '+ Create Patient' and '+ Create Doctor', both highlighted with a red box. Next to them is a 'Refresh' button. Below the buttons, the text 'List of all patients.' is shown. A table follows with three columns: 'Patient Id', 'First Name', and 'Last Name'. The table contains six rows of patient data. At the bottom of the table, it says 'Total count: 6'.

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

Total count: 6

Figure 4.2: Admin Dashboard

4.3.3 Create Patient

The admin has the capability to create a patient that is creating a transaction in the ledger to create an object to the world state. The admin must enter the basic info of the patient as shown in Figure 4.3 on save a transaction is created in the ledger and sent to all the peers of the network. The patient data is validated and if valid endorsed and stored in the ledger. The patient is also created as a client in the network using this client the patient can interact with the ledger. And on save, temporary patient credentials are created, which the patient must change on the first login for security reasons. Figure 4.4 shows the list of all patients, the created patient was added to the ledger and hence is retrieved when all the patients are queried again.

4.3.4 Create Doctor

The admin can also create a doctor. As the doctor is not an object that is stored in the ledger, a client is created in the network using which the doctor now can interact with the ledger. There is no interaction with the ledger on

Create Patient

Please fill out patient details. All fields are required.

First name
Jake

Last name
Crews

Address
Frankfurt

Age
25

Blood group
AB +

Contact number
123456789

Emergency contact number
234567890

New patient's credentials: PID6 - 0h14z8ew

OK

Refresh

Figure 4.3: Create Patient

Lithium Admin Logout

Admin hosp1admin

Create Patient Create Doctor Refresh

List of all patients.

Patient Id	First Name	Last Name
PID0	Monica	Latte
PID1	Max	Mustermann
PID2	Johannes	Schmidt
PID3	Torben	Klaproth
PID4	Lisa	Eckel
PID5	Harry	Schumann
PID6	Jake	Crews

Total count: 7

Figure 4.4: Admin Dashboard

the creation of the doctor. The admin must enter the details of the doctor as shown in Figure 4.5 and on save a client is created in the network.

The screenshot shows the 'Create Doctor' form within the 'Lithium Admin' interface. The form is titled 'Please fill out doctor details. All fields are required.' and contains the following fields: 'First name' with the value 'Adam', 'Last name' with the value 'Jones', 'Hospital' with a dropdown menu showing 'Hospital 2', 'Speciality' with the value 'Neurosurgeon', 'Username' with the value 'HOSP2-DOC002', and 'Password' which is masked with dots. At the bottom of the form are 'Save' and 'Cancel' buttons. The top of the page features the 'Lithium Admin' header and a 'Logout' link. A 'Refresh' button is located in the top right corner of the form area.

Figure 4.5: Create Doctor

4.3.5 Login Patient

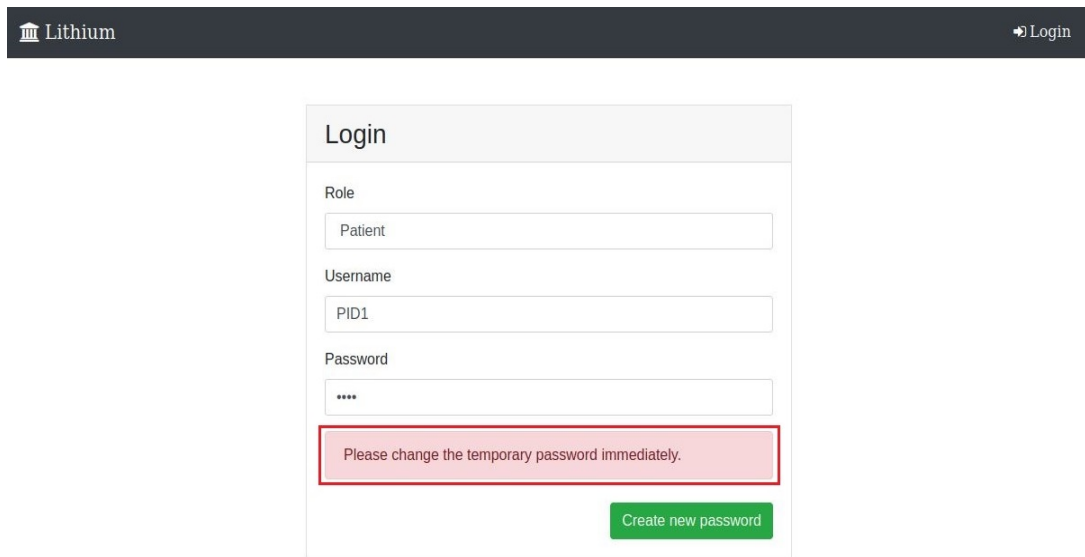
The hospital provides patient his/her username and temporary password during the first visit to the hospital. If the patient is logging in for the first time, then after selecting role as patient and entering username and password provided by hospital, gets the message requesting to change the temporary password immediately as shown in Figure 4.6. The patient has to enter new password as per his/her choice as shown in Figure 4.7. The new password is then hashed and stored in ledger. So, from next time, whenever the patient wants to login, just has to select role, and enter username and new password. Both the temporary 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.

4.3.6 View Patient Details

After successful login, the patient can view his/her personal as well as medical details as shown in Figure 4.8. The patient details are retrieved from the world state by invoking the patient contract.

4.3.7 Grant/Revoke Access

On selecting *View doctors*, it displays a list of doctors from both the hospitals as shown in Figure 4.9. The patient has the right to grant or revoke the access



The image shows the 'Login' form for a 'Patient' role in the Lithium system. The form is titled 'Login' and has a dark header bar with the 'Lithium' logo and a 'Login' button. The form fields are: 'Role' (set to 'Patient'), 'Username' (set to 'PID1'), and 'Password' (masked with '****'). Below the password field is a red-bordered box with the text 'Please change the temporary password immediately.' and a green button labeled 'Create new password'.

Role

Patient

Username

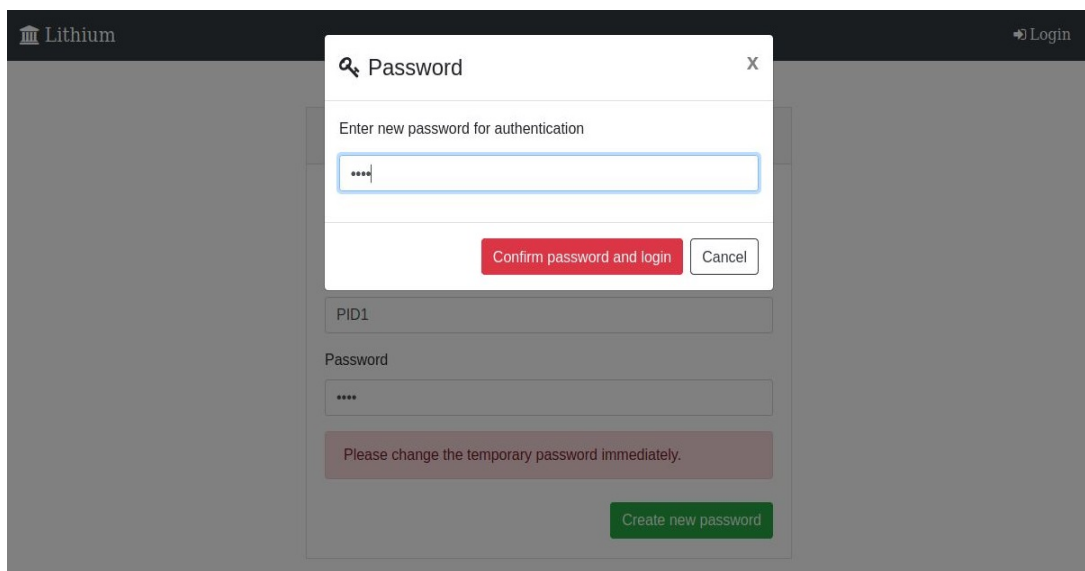
PID1

Password

Please change the temporary password immediately.

Create new password

Figure 4.6: Login Patient



The image shows a 'New Password' dialog box overlaid on the 'Login Patient' form. The dialog is titled 'Password' and has a close button (X). It contains the text 'Enter new password for authentication' and a password input field (masked with '****'). Below the input field are two buttons: 'Confirm password and login' (red) and 'Cancel' (white). The background form is dimmed, showing the 'PID1' username, 'Password' field, and the 'Please change the temporary password immediately.' message.

Password

Enter new password for authentication

Confirm password and login Cancel


PID1


Password

Please change the temporary password immediately.

Create new password

Figure 4.7: New Password


Lithium


Patient

Logout

Patient PID1

[Edit personal details](#)
[View doctors](#)
[View history](#)
[Refresh](#)

Details

Patient ID

PID1

First Name

Max

Last Name

Mustermann

Age

60

Contact number

+491764561111

Emergency number

+491764561113

Address

Mainzer landstrasse 134, 60326 Frankfurt am Main

Blood Group

B+

Allergies

No

Symptoms

Heart Burn, shortness of breath, Acidity

Diagnosis

Esophagitis

Treatment


omeprazole 40 mg for 10 days before food


Follow up period

2 Weeks

Figure 4.8: View Patient Details

to/from the doctor. When patient grants access to the doctor only then the doctor can view patient's medical details and medical history, When patient revokes the access, then doctor is unable to access patient's medical records,


Lithium


Patient

Logout

Doctors

Refresh

List of all doctors.

Total count: 4

Figure 4.9: Grant/Revoke access

4.3.8 Edit Personal Details

The patient can edit his/her personal details as shown in Figure 4.10. If any changes are made, then the new details are updated in the ledger using patient contract.

Patient

Logout

Edit Patient

Refresh

Please fill out patient details. All fields are required.

First name

Last name

Address

Age

Contact number

Emergency contact number

Save Cancel

Figure 4.10: Edit personal details

4.3.9 View History

The patient can view all the records, personal and medical starting from the first visit to the latest visit to all the doctors as shown in Figure 4.10 . This functionality is possible because of the presence of `getHistoryForKey` API in hyperledger fabric which returns the history of the patient. This provides complete transparency of the transactions to the patient.

Patient

Logout

Patient History

Refresh

History of a patient.

Date	Last changed by	First Name	Last Name	Age	Blood Group	Address	Contact number	Emergency number	Allergies	Diagnosis	Symptoms	Treatment	Followup duration
Thu Mar 25 2021	PID1	Max	Mustermann	60	B+	Rebstockbad 27, 60326 Frankfurt am Main	+491764561111	+491764561113	No	Esophagitis	Heart Burn, shortness of breath, Acidity	omeprazole 40 mg for 10 days before food	2 Weeks
Thu Mar 25 2021	PID1	Max	Mustermann	60	B+	Mainzer landstrasse 134, 60326 Frankfurt am Main	+491764561111	+491764561113	No	Esophagitis	Heart Burn, shortness of breath, Acidity	omeprazole 40 mg for 10 days before food	2 Weeks
Thu Mar 25 2021	PID1	Max	Mustermann	60	B+	Mainzer landstrasse 134, 60326 Frankfurt am Main	+491764561111	+491764561113	No	Esophagitis	Heart Burn, shortness of breath, Acidity	omeprazole 40 mg for 10 days before food	2 Weeks
Thu Mar 25 2021	initLedger	Max	Mustermann	60	B+	Mainzer landstrasse 134, 60326 Frankfurt am Main	+491764561111	+491764561113	No	Esophagitis	Heart Burn, shortness of breath, Acidity	omeprazole 40 mg for 10 days before food	2 Weeks

Total count: 4

Figure 4.11: View patient's history

4.3.10 Login Doctor and View Doctor Details

To login as doctor, user need to select 'Doctor' as a role, hospital which he/she belong and right credentials on login page as mentioned in the Figure 4.12.

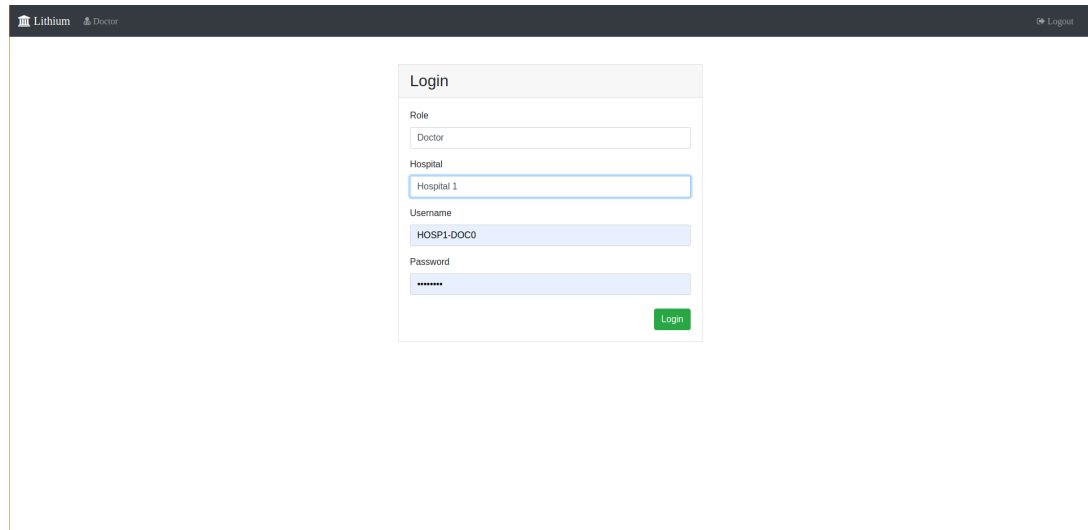


Figure 4.12: Login Doctor

After successfully logged in as a doctor, the doctor details showed just like patient details. Here only few fields of the doctor shown in Figure 4.13

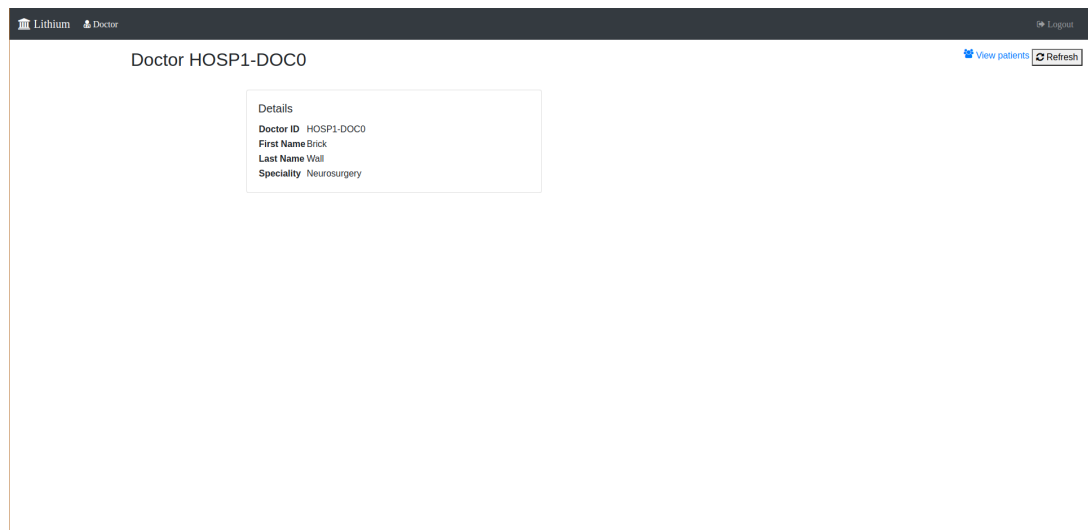
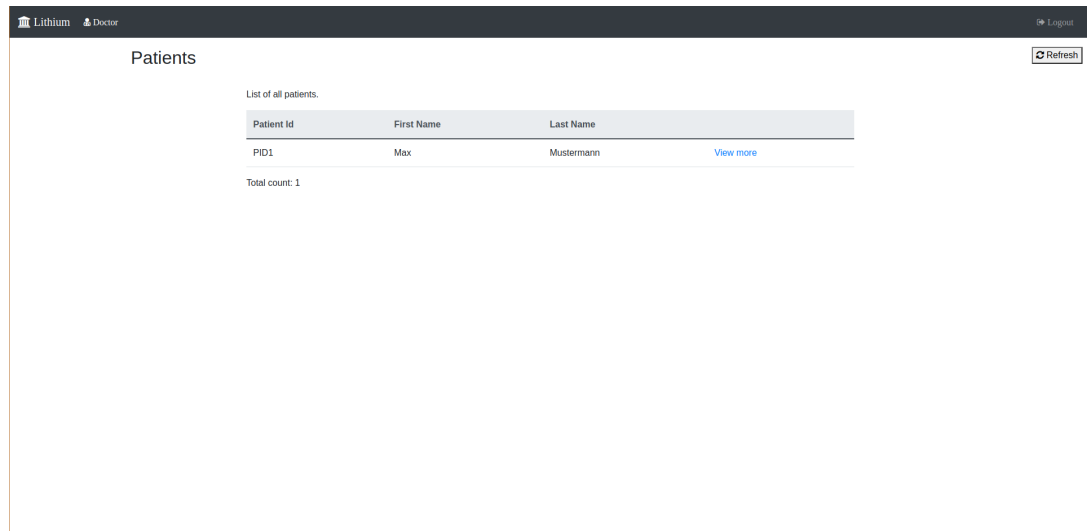


Figure 4.13: View Doctor Details

4.3.11 View Patients

Doctor can view list of patients who provides access to him/her, otherwise list will be empty. In this case, the logged user doctor is 'HOSP1-DOC0' and

patient PID1 provided access to this doctor. So in the list PID1 patient is visible in Figure 4.14.

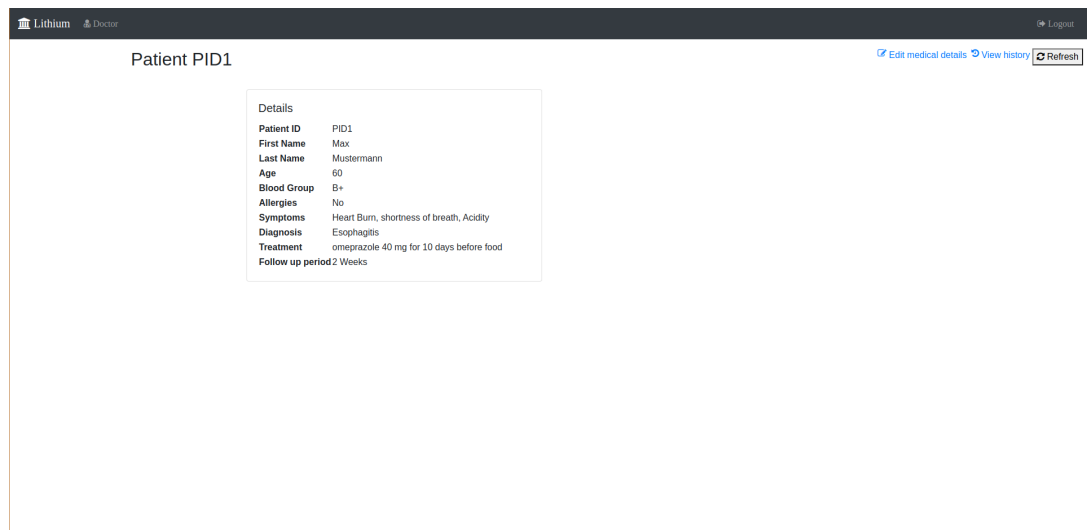


Patient Id	First Name	Last Name
PID1	Max	Mustermann

Total count: 1

Figure 4.14: List of patients

The list looks exactly same as to admin's dashboard, but doctor can see details of patient so in every entry there is a view more button as shown in Figure 4.14. This redirects to patient details page, but only relevant fields are visible to doctor in Figure 4.15. Here doctor can see the latest condition and information of the patient.

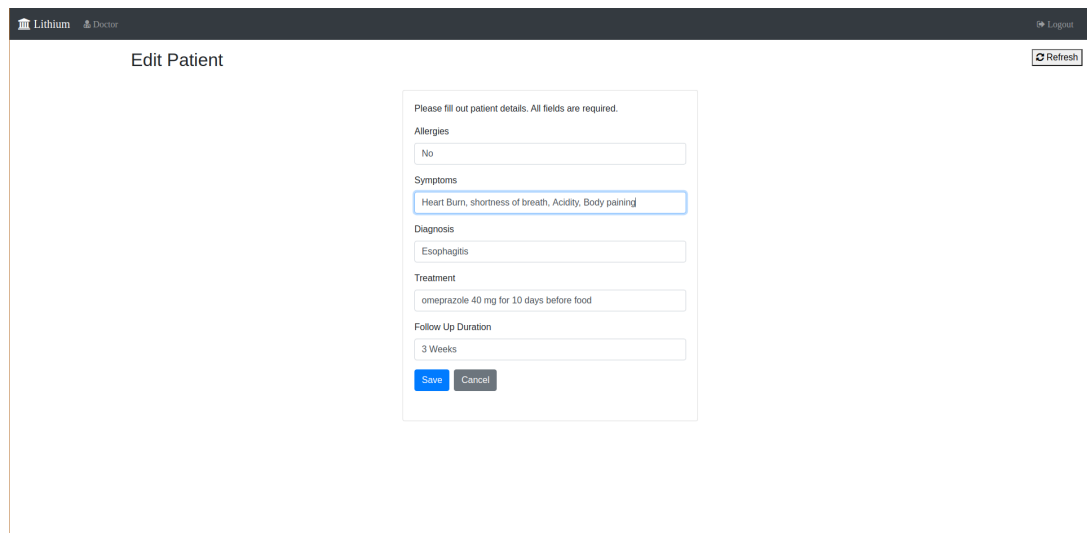


Details	
Patient ID	PID1
First Name	Max
Last Name	Mustermann
Age	60
Blood Group	B+
Allergies	No
Symptoms	Heart Burn, shortness of breath, Acidity
Diagnosis	Esophagitis
Treatment	omeprazole 40 mg for 10 days before food
Follow up period	2 Weeks

Figure 4.15: View Patient Details

4.3.12 Edit Medical Details

Doctor can provide treatment by changing medical details of the patient in Figure 4.16. On save button page redirected to patient details page.



Please fill out patient details. All fields are required.

Allergies
No

Symptoms
Heart Burn, shortness of breath, Acidity, Body painning

Diagnosis
Esophagitis

Treatment
omeprazole 40 mg for 10 days before food

Follow Up Duration
3 Weeks

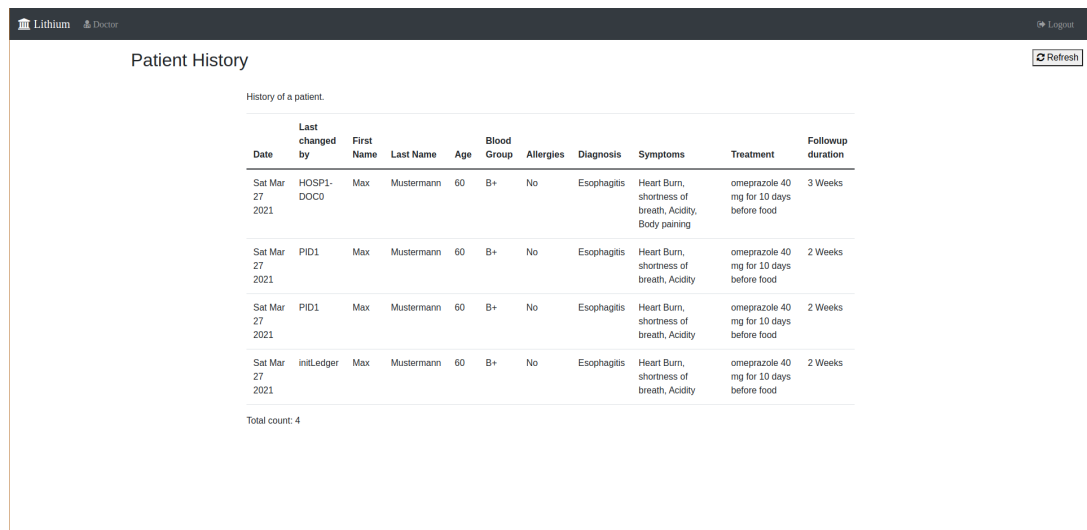
Save Cancel

Refresh

Figure 4.16: Edit patient's medical details

4.3.13 View History

Doctor can view all history details of the patient which help doctor to understand patient's condition and can decide how the medication should be done. In Figure 4.17. it shows same limited fields, as visible in patient details. In addition to those date and changed by info is also helpful to understand the treatment.



History of a patient.

Date	Last changed by	First Name	Last Name	Age	Blood Group	Allergies	Diagnosis	Symptoms	Treatment	Followup duration
Sat Mar 27 2021	HOSP1-DOCO	Max	Mustermann	60	B+	No	Esophagitis	Heart Burn, shortness of breath, Acidity, Body painning	omeprazole 40 mg for 10 days before food	3 Weeks
Sat Mar 27 2021	PID1	Max	Mustermann	60	B+	No	Esophagitis	Heart Burn, shortness of breath, Acidity	omeprazole 40 mg for 10 days before food	2 Weeks
Sat Mar 27 2021	PID1	Max	Mustermann	60	B+	No	Esophagitis	Heart Burn, shortness of breath, Acidity	omeprazole 40 mg for 10 days before food	2 Weeks
Sat Mar 27 2021	initLedger	Max	Mustermann	60	B+	No	Esophagitis	Heart Burn, shortness of breath, Acidity	omeprazole 40 mg for 10 days before food	2 Weeks

Total count: 4

Refresh

Figure 4.17: View patient's history

RESULTS

5.1 PROS AND CONS OF USING HYPERLEDGER FABRIC

This framework has some pros and cons.

5.1.1 *Pros*

Fabric architecture allows the option to add plugins for the identity management and consensus algorithm. This helps companies having their own identity management system to integrate with fabric. The main aim of companies using permissioned blockchain is the confidentiality of data. MSP component in fabric issues and validates certificates and provides user authentication. Data is secure since all the participants in the blockchain network are known. The PoW algorithm is not used and hence mining is not required. This significantly optimizes the performance of the fabric. Fabric allows the creation of a private channel for only a few participants among a large blockchain network. There might be few transactions that should be viewed only by limited participants.

5.1.2 *Cons*

The architecture of hyperledger fabric is quite complex. It is not a fault-tolerant network. As of today fabric supports LevelDB and CouchDb. In the health care scenario, MRI reports and high resolution images need to be stored, in such cases fabric provides limited database support.

5.2 ISSUES IN HYPERLEDGER

Hyperledger Fabric project is an open-source blockchain platform that is still under development. New features and fixes are being developed every day. The following are some of the issues that were faced during the development of the application.

5.2.1 *getHistoryForKey - Private Data Collection*

In the current version of Hyperledger Fabric, the fabric doesn't support the API `getHistoryForKey` for a private data collection [21], however, this is a work in progress by the Hyperledger Fabric community which adds a history index and chaincode API similar to public data history to enable the query of history of a private data key.

5.2.2 *Create a user defined role instead of client*

The current version of hyperledger fabric the only possible roles for `hf.Registrar.Roles` are Peer, client, and admin. The flaw with only 3 roles is, now all the clients have the same set of permission, but in a blockchain, there can be any number of types of roles and a set of permissions for each user-defined role. In the scenario, doctor and patient can be the user-defined roles rather than just the client. Doctor and patients can have their own set of rules to accomplish the issue mentioned in 5.2.3. The fabric community is currently in progress to implement a way to add new `hf.Registrar.(Delegate)Roles` [27].

5.2.3 *Access user attributes using client*

In this scenario, as the doctor is not an asset to the ledger, the name and specialty of the doctor are stored in the attributes of the identity. However, the patients in the blockchain network cannot retrieve the attributes, as the patient does not have the access to read the doctor's attributes, but can be read using the admin user. This issue is due to the reason that a separate set of permissions cannot be applied as all the clients are considered to be the same.

5.3 CHALLENGES IN DEVELOPING APPLICATION

Major challenges are faced in implementing security on patient data in the ledger on a peer level. As there are two mechanisms described in 3.7. The private data collection approach is unable to apply due to fabric issues described in the above section. For data re-encryption, the approach fails to implement due to two major reasons. The first problem is, nodejs is lacking a decent library for re-encryption. Few libraries are available here [15]. But none of them worked here. One has a working approach but does not fit regular system generated private and public keys. The format is not understandable by that library. Node-RSA is a very library for public-key cryptography and also accepts keys with the regular format, but does not have re-encryption functionality. There is an opportunity here to understand implement a re-encryption algorithm which is a time consuming task but a nice step as a separate small project and contributing to the community. The other problem is when a user gets created certificate and private key are generated. But not able to find a public key in the fabric framework. More research is required to diagnose this issue and make re-encryption mechanism work. Next challenge faced in scaling peers of hospital organizations. For some reason, one fabric-peer container always stops when the network gets up. In addition to that when generating ccp files, not all peer configurations are included. The work is in progress and **draft pull request is opened**.

CONCLUSION AND FUTURE WORK

6.1 CONCLUSION

Hyperledger fabric is a promising blockchain framework that has some concepts of policies, smart contracts, and provision of secure identities which make the records secure and controlled. It enables the EHR systems to inter-operable among multiple hospital organizations. Doctors can track history easily. Patients do not need to carry medical history files and will be significant improvements in digital records.

An EHR scenario comes under the private and closed blockchain category and this solution can successfully conclude that it is an encouraging framework of this kind of blockchain. It provides a reliable and secure solution in managing medical field record

6.2 FUTURE WORK

Many improvements can be done to improve the solution, make a production-grade application. Even though blockchain and fabric provide tons of security by default in their framework and concept, still need to overcome the security challenges discussed above and implement successful security for the patient records. Though the fabric framework is quite pluggable by itself, the source code can be improved to make the solution more pluggable when adding more hospitals and its peers. As the network scales up, more organizations and peers are connected to the channel, and to handle many transaction requests and approvals, many ordering peers are needed to speed up the process. Apaches Kafka, an open-source distributed event streaming platform is a very promising tool to manage multiple ordering nodes.

Design consortium policy in such a way that minimum criteria of consensus algorithm can be satisfied. The fabric uses pBFT in which all peers' approval is needed to approve transactions, such as 75% approving criteria it means 3 out of 4 peer's approval is sufficient to make a transaction. Distribute the wallet as per the organization's peers. The users created at the wallet should be stored to their respective organization peers. The wallet can also be stored in the no-SQL database and replicate to multiple nodes to avoid data loss.

The network calls in the network should be done via HTTPS which provides transport-level security (TLS). Currently, from the front end to the back end, passwords are sent in plain text which can be improved by this. For temporary password integration of email, functionality is the best approach. Moreover adaption of forgetting password functionality is good to have. UI/UX strategies can be applied to make enhance the experience. Search

functionality can be useful when patients' data go beyond the limit. But need to research that if fabric supports wildcard searches or not. Again the data is not coming from a single database so it is important that frequent search queries can not be possible.

The application should have the test cases to run the application in a production environment. Few test cases are available but it should have a test case for unit, integration, public REST API, and end-to-end cases. The existing test case was written using the Cypress framework which is an E2E testing framework. An application can be deployed using Kubernetes for the production environment. As the network grows the more hospitals with their peers and channels exist in the network. So Kubernetes is the best orchestration tool to manage all these containers.

BIBLIOGRAPHY

- [1] Zhigang Xu Rohit Shukla Pratik Sushil Zambani Arun Swaminathan Md Majid Jahangir Khadija Chowdhry Rahul Lachhani Nitesh Idnani Michael Schumacher Karl Aberer Scott D Stoller Samuel Ryu Alevtina Dubovitskaya Furqan Baig and Fusheng Wang. "ACTION-EHR: Patient-Centric Blockchain-Based Electronic Health Record Data Management for Cancer Care." In: NCBI. 2019. DOI: [10.2196/13598](https://doi.org/10.2196/13598).
- [2] Roman Beck. "Beyond Bitcoin: The Rise of Blockchain World." In: *Computer* 51 (Feb. 2018), pp. 54–58. DOI: [10.1109/MC.2018.1451660](https://doi.org/10.1109/MC.2018.1451660).
- [3] Dr. Stefan Beyer. *Blockchain Before Bitcoin: A History*. <https://blocktelegraph.io/blockchain-before-bitcoin-history/>. [accessed 27.03.21].
- [4] *Cryptogen*. <https://hyperledger-fabric.readthedocs.io/en/release-2.2/commands/cryptogen.html?highlight=cryptogen>. [Accessed 27.03.2021].
- [5] *Different Types of Hyperledger Technologies*. <https://medium.com/@vinshublockcluster/different-types-of-hyperledger-technologies-929a67c98c32>. [Accessed 27.03.2021].
- [6] "Download fabric samples procedure." In: URL: https://hyperledger-fabric.readthedocs.io/en/release-2.2/test_network.html#before-you-begin.
- [7] Alevtina Dubovitskaya et al. "ACTION-EHR: Patient-Centric Blockchain-Based Electronic Health Record Data Management for Cancer Care." In: *Journal of Medical Internet Research* 22.8 (Aug. 2020), e13598. DOI: [10.2196/13598](https://doi.org/10.2196/13598). URL: <https://doi.org/10.2196/13598>.
- [8] *Hyperledger Fabric SDK for Node.js*. <https://hyperledger.github.io/fabric-sdk-node/release-2.2/module-fabric-network.html>. [Accessed 27.03.2021].
- [9] *Identity*. <https://hyperledger-fabric.readthedocs.io/en/release-2.2/identity/identity.html>. [Accessed 27.03.2021].
- [10] Andrew Lippman John D. Halamka and Ariel Ekblaw. *The Potential for Blockchain to Transform Electronic Health Records*. <https://hbr.org/2017/03/the-potential-for-blockchain-to-transform-electronic-health-records>. [accessed 27.03.21].
- [11] T. Kumar, A. Braeken, M. Liyanage, and M. Ylianttila. "Identity privacy preserving biometric based authentication scheme for Naked health-care environment." In: *2017 IEEE International Conference on Communications (ICC)*. 2017, pp. 1–7. DOI: [10.1109/ICC.2017.7996966](https://doi.org/10.1109/ICC.2017.7996966).

- [12] T. Kumar, V. Ramani, I. Ahmad, A. Braeken, E. Harjula, and M. Ylianttila. "Blockchain Utilization in Healthcare: Key Requirements and Challenges." In: *2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom)*. 2018, pp. 1–7. DOI: [10.1109/HealthCom.2018.8531136](https://doi.org/10.1109/HealthCom.2018.8531136).
- [13] *Ledger*. <https://hyperledger-fabric.readthedocs.io/en/release-2.2/ledger/ledger.html>. [Accessed 27.03.2021].
- [14] Ying-Chang Liang. "Blockchain for Dynamic Spectrum Management." In: *Dynamic Spectrum Management, Signals and Communication*. Research Gate. 2020. DOI: [10.1007/978-981-15-0776-2_5](https://doi.org/10.1007/978-981-15-0776-2_5).
- [15] "Libraries in nodejs for re-encryption." In: URL: <https://github.com/topics/proxy-re-encryption?l=javascript>.
- [16] *Membership Service Provider (MSP)*. <https://hyperledger-fabric.readthedocs.io/en/release-2.2/membership/membership.html>. [Accessed 27.03.2021].
- [17] Kevin J. Peterson, Rammohan Deeduvanu, Pradip Kanjamala, and K. Mayo. "A Blockchain-Based Approach to Health Information Exchange Networks." In: 2016.
- [18] *Policies*. <https://hyperledger-fabric.readthedocs.io/en/release-2.2/policies/policies.html?highlight=endorsement>. [Accessed 27.03.2021].
- [19] "Prerequisites before start test network of hyperledger fabric." In: URL: <https://hyperledger-fabric.readthedocs.io/en/release-2.2/prereqs.html#prerequisites>.
- [20] *Private data*. <https://hyperledger-fabric.readthedocs.io/en/release-2.2/private-data/private-data.html>. [Accessed 27.03.2021].
- [21] *Query history of private data - by key, block, or transaction id*. <https://jira.hyperledger.org/browse/FABC-5094>. [Accessed 28/03/2020].
- [22] Neeraj Kumar Muhammad Khurram Khan Shuyun Shi Debiao He and Kim-Kwang Raymond Choof. "Applications of blockchain in ensuring the security and privacy of electronic health record systems: A survey." In: NCBI. 2020. DOI: [10.1016/j.cose.2020.101966](https://doi.org/10.1016/j.cose.2020.101966).
- [23] Shyam Pratap Singh. *Detail Analysis of Raft its implementation in Hyperledger Fabric*. <https://medium.com/coinmonks/detail-analysis-of-raft-its-implementation-in-hyperledger-fabric-d269367a79c0>. [accessed 27.03.21].
- [24] *Smart Contracts and Chaincode*. <https://hyperledger-fabric.readthedocs.io/en/release-2.2/smartcontract/smartcontract.html>. [Accessed 27.03.2021].

- [25] Sudeep Tanwar, Karan Parekh, and Richard Evans. "Blockchain-based electronic healthcare record system for healthcare 4.0 applications." In: *Journal of Information Security and Applications* 50 (Feb. 2020), p. 102407. DOI: [10.1016/j.jisa.2019.102407](https://doi.org/10.1016/j.jisa.2019.102407). URL: <https://doi.org/10.1016/j.jisa.2019.102407>.
- [26] Dara Tith, Joong-Sun Lee, Hiroyuki Suzuki, W. M. A. B. Wijesundara, Naoko Taira, Takashi Obi, and Nagaaki Ohyama. "Application of Blockchain to Maintaining Patient Records in Electronic Health Record for Enhanced Privacy, Scalability, and Availability." In: *Healthcare Informatics Research* 26.1 (2020), p. 3. DOI: [10.4258/hir.2020.26.1.3](https://doi.org/10.4258/hir.2020.26.1.3). URL: <https://doi.org/10.4258/hir.2020.26.1.3>.
- [27] *fabric-ca should need a way to add new hf.Registrar.(Delegate)Roles*. <https://jira.hyperledger.org/browse/FABC-548>. [Accessed 28/03/2020].