Vietnamese - German University

DEPARTMENT OF COMPUTER SCIENCE

Frankfurt University of Applied Sciences

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

FINAL REPORT

Project name: SONA project

Bui Le Phi Long - 16619 : Team Leader
Huynh Cam Tu - 16610
Truong Canh Thanh Vinh - 15766
Le Duc Minh - 16669
Nguyen Quoc Trung - 15766

Project advisors:

Prof. Martin Kappes
TA. Lukas Atkinson
TA. Johannes Bouche

SONA

2022

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 Objectives

Today, the digitization of medical records has raised concerns about the privacy and security of health information. In Australia, the government and healthcare providers have gradually introduced e-health services and national electronic health records over the past two decades. In 2012, a national electronic health record platform, Personal Controlled Electronic Health Record (PCEHR) was launched. As of October 2021, 4.93 billion documents have been uploaded by healthcare providers and consumers. Health information platforms are set up, but nearly 10 percent of Australians opt out for privacy reasons [1]. When it comes to privacy, many features of this platform really need to be improved. B. Preventing Data Loss, Unauthorized Access, and Loss of Privacy. Additionally, having transparent data access and record sharing at your fingertips is critical to patient comfort.
When it comes to data processing, storing, reviewing, synchronizing, and sharing medical records has always been a daunting task. When healthcare providers and researchers need to access and share medical data, that data is subject to severe political and technological constraints. This means that significant time and resources must be expended in conducting approval reviews and data reviews. In most cases, each hospital's database is maintained separately. Each platform has its own standards, lacking incentives or incentives to share data between different medical institutions. These issues hinder the sharing of health information, which in turn impacts the large-scale delivery of more cost-effective health services and data-driven solutions.

To solve these problems, Hyperledger Fabric Framework is introduced for enterprise-level data processing capabilities and enhanced privacy protection for our project - Sona Healthcare System (SHS). The main contribution of this work is implementing a medical data management system for users, doctors and researchers.

1.1.1 Background

With smart contract support, the improved model has the potential to address privacy concerns and increase user engagement. Several simulation models were developed to evaluate the feasibility of the proposed framework. Together they demonstrate the scenarios and benefits of sharing medical data using blockchain technology. The main contribution of this work is

implementing a medical data management system and mechanism based on the Hyperledger Fabric blockchain environment. With support for smart contracts, the improved model has the potential to address privacy concerns and increase user engagement. Several simulation models were developed to evaluate the feasibility of the proposed framework. Together they demonstrate the scenarios and benefits of sharing medical data using blockchain technology.

The aim of this project is to propose a different approach compared with the traditional medical system . In addition, the project will provide all members of his Stuttgart team with an opportunity to immerse themselves in blockchain in general and his Hyperledger Fabric world in particular. Also, each individual is able to sharpening their coding skill before joining industry.

1.1.2 Related Work

This section describes the key differences between traditional database approaches and solutions that use other blockchain frameworks on the Hyperledger Fabric framework.
Traditional healthcare system databases use client-server network architecture. Here, a user (known as a client) can modify data, which is stored on a centralized server [13]. Control access of the traditional database remains with a designated authority that authenticates client credentials before granting access to the database. This institution is responsible for managing the database, so data may be altered or destroyed if the institution's security is cracked.
A blockchain database consists of multiple distributed nodes. Each node participates in management:
All nodes can validate new additions to the blockchain and enter new data into the database. A majority of nodes must reach consensus in order to add to the blockchain. This consensus mechanism ensures network security and makes it difficult to exploit.
After looking at several existing works, we decided that Hyperledger Fabric would be the best fit for our project due to our shortage of time and the built-in framework convenience.

1.2 Motivation

Medical record is one of the most important and sensitive data. However, according to The Department of Health and Human Services' Office for Civil Rights, in 2021, there was 686 healthcare data breaches and 44,993,618 healthcare records have been exposed or stolen [1]. Moreover, about 3 out of 4 of this breaches were hacking or other IT incidents. Also, most of the healthcare system is based on traditional centralized database which is vitally vulnerable. For those motivation, our team, Stuttgart, has implemented a basic healthcare system called Sona powered by Hyperledger Fabric, a blockchain powerful technology.

1.3 Project scope

This project focuses on creating a private and permissioned blockchain network leveraging an open source engine powered by Hyperledger Fabric. The main contribution of this project is that it proposed health data management. Frameworks and mechanisms based on the Hyperledger Fabric blockchain environment. With support for smart contracts, the improved model of healthcare system has arrived for privacy concerns and to improve user engagement. Finally, the deployment of the project is one of our attention to bring the project to essential users.

1.4 Limitation

Although the project is deploying quite well, it is still missing several functions for adapting to the real-world situation. For instance, the system is able to access by 10 clients as one time due to the weakness of the droplet in Digital Ocean cloud server. Also, this project user's interface is not really beautiful since our group just include 5 members.

1.5 Definition of terms

1.5.1 Blockchain

Although blockchain is widely accepted, it is not a new technology. Blockchain was first proposed by American cryptographer David Chaum in 1982 and has evolved over time. From 2008 to his 2009, Satoshi Nakamoto improved the design and implemented it as his core component of the currently popular cryptocurrency Bitcoin. Blockchain can be described as a distributed ledger made up of blocks. Each block contains a record of transactions, all distributed and maintained within a network of peers. In practice, peers can be various companies participating in the exchange, as well as individuals holding cryptocurrency assets. When a new transaction is proposed, it must first be approved by peers via a consensus protocol. It is then hashed, encoded into blocks, and ordered to ensure ledger consistency. In addition to transactions, each block contains a cryptographic hash of the previous block in the blockchain. This allows us to check the integrity of previous transactions.
Blockchain is an exciting and versatile technology, but it is not without its limitations. A big problem is the costly and painful process of implementing blockchain. Most blockchain projects, including Hyperledger Fabric, are open source, but investment needs are high for all interested organizations. Costs include hiring developers, managing a team of blockchain technology experts, and maintaining the system. Additionally, the knowledge and expertise required to implement and manage blockchain products is enormous. In addition to the difficulty and high cost of hiring blockchain specialists, companies need to train existing

developers to ensure optimal management and use of blockchain implementations.

1.5.2 Hyperledger Fabric

Hyperledger Fabric is a blockchain framework advanced through the Linux Foundation. By enabling a flexible approach to blockchain technology development, Hyperledger provides an extensible, modular architecture that can be used for different domains and built independently of any particular application.
What makes Hyperledger Fabric different from some other blockchain systems is that it is private and permissioned. This means that each member must prove their legitimate identity in order to join the network.
The ability to create channels provides the option to create a separate subscription for a group of participants. Because channels are partitioned, clients can only view messages and transactions related to the channels to which they are connected, while ignoring the existence of other channels. As a result, access to transactions is limited to the parties involved, to achieve consensus at the transaction level and not at the ledger level as with Ethereum.
The Hyperledger Fabric ledger consists of two components: the global state and the transaction log. At any given time, the world state displays the state of the general ledger, while the transaction log component maintains the information of all transactions, resulting in the current value of the world state. Non-blockchain applications invoke smart contracts to interact with the ledger. Smart contracts are used to defy the logic of transactions and they are encapsulated in blockchain code. Smart contract developers can use Go, Node.js, and Java to write chaincode.In this case, we use Node.js to implement our chaincode.
Nodes are distinguished according to their roles: client, orderer or peer. The client acts as an end user, having the ability to create and cancel transactions, and to communicate with the originator and other peers. Peers act as maintainers of the blockchain ledger, receiving messages from host companies in order and entering new transactions into the ledger. Validators are special peers tasked with confirming transactions by verifying the terms, status, and validity of the transaction (for example, by confirming the terms and signature of the request). transaction demand). Authorizers provide a communication channel between clients and peers to ensure that transaction messages are broadcast over this channel. Channels must ensure that all peers in the connection receive exactly the same message with the same logical order.
Every components in Hyperledger Fabric including orderers, peers, admins, etc have an identity. These components will use their identity to interact with the network, which is issued as an X.509 digital certificate. Identity is important because they also help the system determine what actions an agent can perform and their access to network resources.
Transaction Flow: Client submits transaction to auditor to initiate ledger update process. All validators must either agree to the proposed transaction or reach consensus based on the proposed ledger update. Clients will continue to seek approval from all validators. Approved transactions are sent to connected orderers to reach another consensus. This transaction is forwarded to the peer holding the ledger to execute the transaction.

2 METHOD

2.1 System design and implementation

The sona project is designed to overcome the limitations of existing healthcare systems by using permisioned blockchain network of hyperledger fabric framework which allow many organizations to interoperability while still ensure the privacy-preservation. The sona system will have following features:

- Many organizations including clinics and research labs can easily interoperate with each other.
- Electronic health records be kept track with patient consent.
- The usage of a patient's electronic health record is monitoring to allow the patient to know who is using their data and how it is used.

2.1.1 System overview

In the existing healthcare systems, the electronic health records (EHR) are scattered in heterogeneous databases which make it really hard for interoperability. The design of sona using a permissioned blockchain network solves this issue by allowing organizations to connect and share their resources while ensuring the fine grain control of their shared data. Organizations in the blockchain network can create several subnetworks and share data only to those in the subnetwork by leveraging the same underlying network created by all organizations using the channel feature of hyperledger fabric framework.
Sona system also focuses on the access control of the EHR to ensure the security of the patient's data. Organizations can define their access control mechanism in many levels:

1. In the channel level using policies by configuring the config file of the channel.
2. In the chaincode level by implementing access control in the definition of the chaincode.

With the traditional healthcare system, the work of generating usage records for a patient's health record is cumbersome as it requires lots of paperwork to keep track of the usage. The use of smart contracts and electronic health records in blockchain network eliminates those problems because the business logic of creating new usage records whenever the EHR is read or write is defined in the smart contract's transactions. The atomic nature of the transaction ensures that the usage record will be generated and stored whenever the EHR is used.

2.1.2 User Scenarios

Because our sona project is just a prototype so we are using the test network provided by hyperledger fabric samples.
Our network contains of 3 organizations:

- Organization 1 is clinic1. It provides health services for patients so it is the party generating and controlling patients' identifiable health records. Clinic1 contributes 1 peer to the network.
- Organization 2 is a research lab named lab1. This lab wants to have access to health records for research purposes. Lab1 also contributes 1 peer to the network.
- Third organization is the orderer of the network.

For simplicity, we only create one channel containing all those organizations. Unidentifiable health records (reference to its definition) will be stored on channel ledger (which is stored on all peers in the channel). Identifiable health records of patients of clinic1 and usage records will be stored in clinic1's private data collection to ensure the security of those data (details in how it can secure the access of those data will be explained in the following sections).

2.1.3 System entities and Organizations

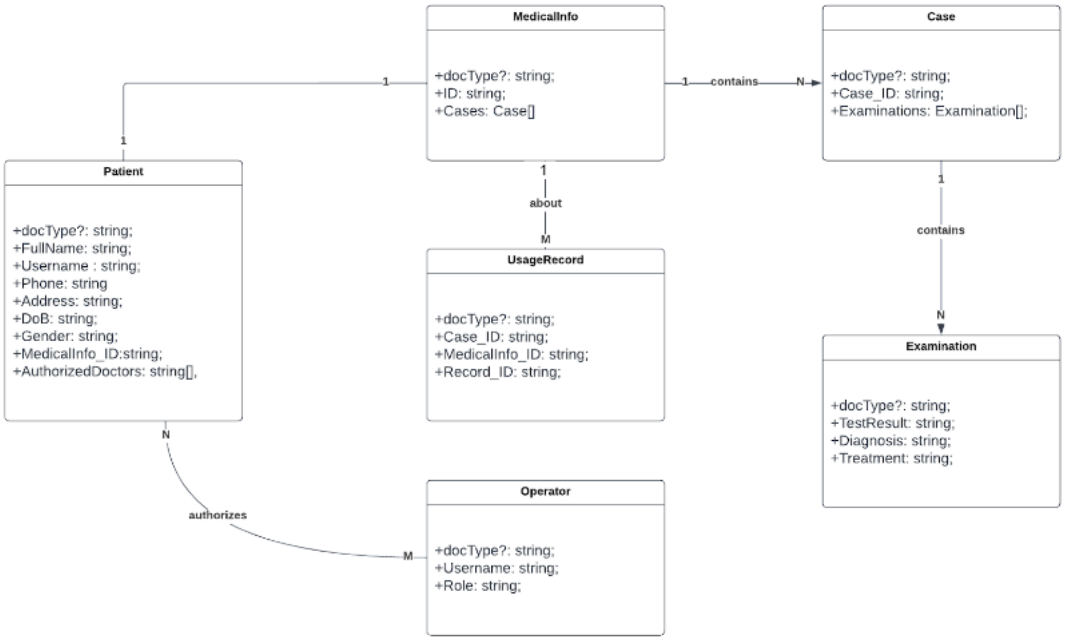


Figure 1: Entities in SONA System

This sections will introduce main entities and organizations in Sona healthcare system. Electronic Health Record: In existing healthcare system, medical records of patient are still paper based and required patients to keep and bring them to the doctor for next examinations. As the records are not digitally stored, they can easily be lost and very hard to keep track patient's medical history. Electronic health record can make the process of storing, accessing and sharing of the medical data much more efficient.

Usage Record. One of the main features of Sona system is that it allows patients to keep track of their health information usage. Usage Record will be generated when an operator of the system (it could be doctors, researchers or anyone accessing the medical information of a patient) manipulate a patient's medical information. For this simplified implementation, we only have two operations interacting with the medical record, read and write. Each usage record have information about which medical record are accessed, which operator access the record, what operation, time of the access. If a patient want to know about who interacting with their medical information, he/she can query the list of usage records of his/her medical information. Patient. In existing healthcare system, patients has no or limited access and control to their medical information stored in medical service provider databases. With our new system, patients is also an user of the system, their will have an interface to access to their medical information, have knowledge about who is using their medical information and manage the access control of their information.

Blockchain network. As introduced above, Sona will implement a permissioned blockchain network using Hyperledger Fabric framework. This network will comprises healthcare, research

organizations and other parties that need to have access to the medical health record as well as to inter-operate with each others. All organizations joining the blockchain network will contribute resources to create a network connected all the organizations. They can also create sub-networks that separated the business logic and shared data with different subsets of organizations in the network.
Medical Service Provider. Organizations that provide medical services, generate and processing medical data. In the traditional healthcare system, there are no linked between medical service providers which make the medical record are scattered in many heterogeneous database. Because the privacy and security of the medical records of patients is a very critical matter, medical service providers need lots of trust to share information with other services providers. The implements of permissioned blockchain network allow those organizations to connect together and sharing medical data with least amount of trust.
Research Lab. The entity that want to have access to the medical records for research and analysis. The existing healthcare system in which medical data is keep private and only accessible to the parties holding it, making it really hard for research organization to access to the medical record sources they need. With Sona system those organizations can join the blockchain network and get access to the shared medical records data in the system.

2.1.4 System Implementation

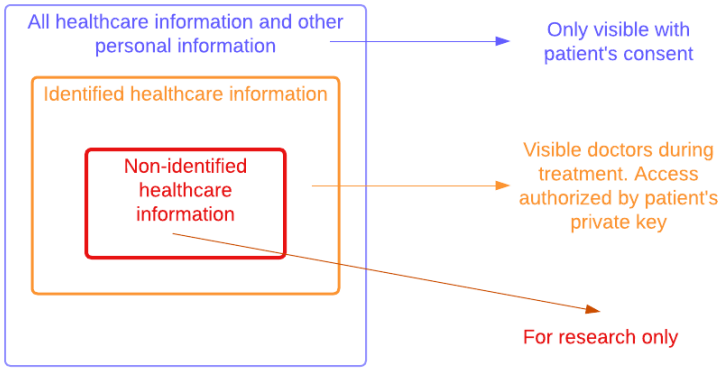


Figure 2: Access Control Hierarchical

Database implementation. Data in Hyperledger Fabric framework will be stored in the ledger which includes a world state and blockchain. The modular architecture of Hyperledger fabric allow us to choose between LevelDB and CouchDB for the world state but the database for block chain is LevelDB by default and Hyperledger fabric do not provide other options for that. Sona system using CouchDB for world state database as it is capable to store and query complex data object because the medical record contains all the medical history of patient so its structure can be very complex to includes all kinds of data formats.

Consensus mechanism is also a plugable component in Hyperledger Fabric Framework. Because HF is a permissioned blockchain network in which participants in the networks must have unambiguous identities, it do not need to implement Byzantine Fault Tolerant which is costly and inefficient. In our system design, we will use the default Raft based rash Fault Tolerant consensus protocol. System Users. Our implementations simplifies the users of the system to patients and operators (people working with medical records). Sona system allow patients to have control and understanding of their own medical data by provide them an interface to access and manage the access permission of their information. The operators in the Sona system can be anyone who have access and manipulate the medical record, they could be medical staffs or researchers.

Hierarchical access control for privacy issues. We only need to deal with data privacy concerns when the data is identifiable which means it can be used to trace back to an identity. In our Sona healthcare system, we will separate the identifiable information of patients from their medical health records. Which means that the medical health record is an unidentifiable data and can be public. With this design, we can ensure the privacy of patient's information while also allows the medical health record to be shared for researching and effective healthcare services.

2.1.4.1 System Functionalities

In this sections, we will introduce about main functions in Sona healthcare systems with respects of two main user types of the systems.

Product Backlog

Table + | Filter | Sort | New

User Story	Score	Priority	Time required	Status
As a patient, I want to see my information	5	5	1 week	Done
As a patient, I want to be asked for my permission if someone (not the doctor I currently work with) try to use my data	3	3	1 week	Done
As a patient, I can authorize or revoke the access (from some doctors) to my data	4	4	1 week	Done
As a patient, I want to see my authorization list	5	5	1 week	Done
As a doctor, I want to see medical information of patients	5	5	1 week	Done
As a doctor, I want to see the list of all patient that authorized me	5	5	1 week	Done
As a doctor, I want to create and update the patient medical information	5	5	1 week	Done
As a member in a research facility, I want to search for unidentified medical data	3	1	1 week	Done

Figure 3: Product Backlogs of Sona System

First we discuss patient's use cases in our implementation. As stated above, our system provides an interface for patient to have access to their health record, knowledge of who is using their health record and manage access control of their information. So basically, patients are provided the following functions:

- Query Personal Data: Patient using this function will providing their username and get their personal information associated with that username in world state. This function first check there's a patient with that username exists.
- Query Medical Information: This function receive the username of patient and check whether the username of that patient exists, then it will query the patient's personal information associated with the username to get the medical record's ID of that patient. Finally, it use the medical record's ID to query the medical record.
- Query Usage Record: This function receives ID of the medical record and return the list usage records associated with that ID. This function is only available for patients so that only patients can know how their medical record are used.
- Authorize Doctor: Because patient's personal information must be kept private and can only make accessible with patient's consent, patient will have a list of authorized doctors they allow to access their identifiable information. This function receives doctor and

patient's usernames. Then it first checks the doctor's username exists and not already contained in the list, then it adds the doctor's username to the patient's authorized doctors list.

- Revoke Doctor: This function used to revoke a doctor's right to access patient's personal information. It first check the doctor's username exists and contained in the patient's authorized doctors list, then it remove the doctor's username from the list.

Figure 4: Patient's Use Cases

Next we discuss about the interface for operators (doctors and researchers) in sona system. In our simplified implementation, we provides five functions for operators to manipulate the medical records:

- Query Personal Data. This function is used for operators to query their personal information on the system providing their username.
- Query Patient. This functions is for doctors authorized by patient to access to patient's personal information. It receives doctor's username, patient's username and checking if the doctor with that username is authorized to access the personal information associated with that patient's username.
- Query Medical Information. Both Doctors and Research have access to this function to query a specific Medical Information providing its ID.
- Create Patient. This function is called when a new patient register an account in sona system. User cannot create the account by themself, but the doctor will create it for them when they come to Medical Service Provider for the first time.
- Search Case by Keywords: This is function is used by researchers to query all the available medical records with the illnesses/conditions needed for their research. It

receives the list of keywords and return the list of medical record containing those keywords.

Figure 5: Operator's Use Cases

2.1.4.2 Data Types

```
@Object()
export class Patient {

    @Property()
    public docType?: string;

    @Property()
    public FullName: string;

    @Property()
    public Username: string;

    @Property()
    public Phone: string;

    @Property()
    public Address: string;

    @Property()
    public DoB: string;

    @Property()
    public Gender: string;

    @Property()
    public MedicalInfo_ID: string;

    @Property()
    public AuthorizedDoctors: string[];
}
```

Figure 6: Data type of Patient

The patient objects contains basic personal information of a patient. Each patient object contains an ID of medical record associated with his/her and an authorizedDoctors array which is used to store all usernames of doctors authorized to access patient's personal information.

```
@Object()
export class Operator {

    @Property()
    public docType?: string;

    @Property()
    public Username: string;

    @Property()
    public Role: string;

}
```

Figure 7: Data type of Operator

The Operator object in this simplified implementation only contains the Username and role of an operator. We need this the role of the operator to know whether the access of a user to patient's information is valid or not.

MedicalInfo contains the list of cases of a patient. It has an ID to uniquely identify it and an case array contains all cases belongs to the patient associated with this MedicalInfo ID. We will store the entire case object in the case array instead of IDs of cases.

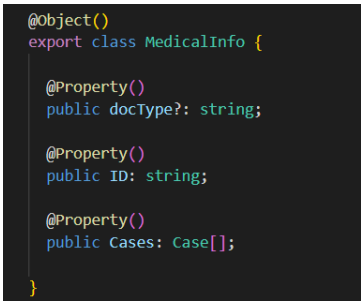


Figure 8: Data type of Medical Information

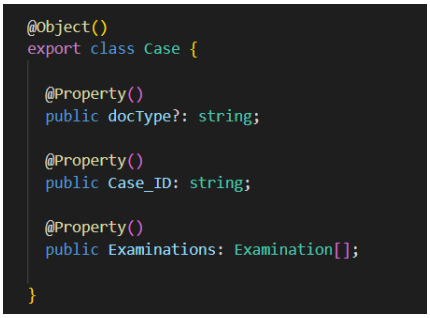


Figure 9: Data type of Case

Case represent a list of patient's examinations in a particular period of time. When a patient go to doctor, this is considered one examination. If the the gap between examinations is less than six months, they will be added to the same case. Each case contains an uuid to uniquely identify it and an array of type Examination.

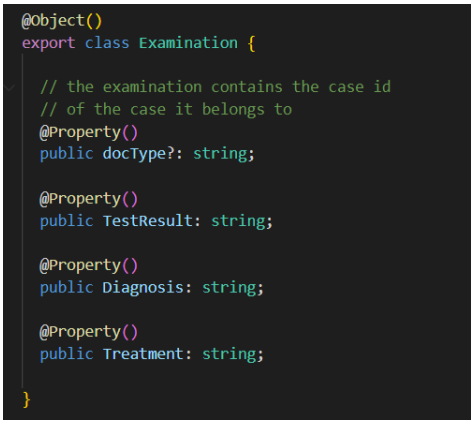


Figure 10: Data type of Medical Examination

Examination represent a medical record patient receive from doctor for each examination. This is simplified to include only test result, diagnosis and treatment. In real cases, the examination may contains many complex data types such as images, videos but this implementation simplified to only text data.

```
@Object()
export class UsageRecord {

  @Property()
  public docType?: string;

  // case_id will be undefined if the operation is query
  @Property()
  public Case_ID?: string;

  @Property()
  public MedicalInfo_ID: string;

  @Property()
  public Record_ID: string;

  // operation can be:
  //    append (append to an existing case - by calling updateCase())
  //    add (add a new case to medicalinfo - by calling createCase())
  //    read (query medical record - by calling QueryMedicalInfo())

  @Property()
  public Operation: string;

  // get this from Operator info
  @Property()
  public Roles: string;

  // get this from Operator info
  @Property()
  public OperatorName: string;

  @Property()
  public Time: string;

}
```

Figure 11: Data type of Usage Record

The UsageRecord contains all information of one usage of a MedicalInfo. It includes the ID of MedicalInfo in used, the ID of the case involved in the operation, type of operation (read/write), operator username and his/her role and the time when the operation occured. Each UsageRecord have an ID to uniquely identify it.

2.1.4.3 Smart Contract Implementation

In the previous section, we have introduced all the data types in our implementations. In those six data types, only four of them are the assets of our system, the other two only used for structuring main asset data types. For each asset, we create a smart contract encapsulating all business logic to manipulate that asset. The interaction of those smart contracts and the utility class are illustrated in the figure 12.

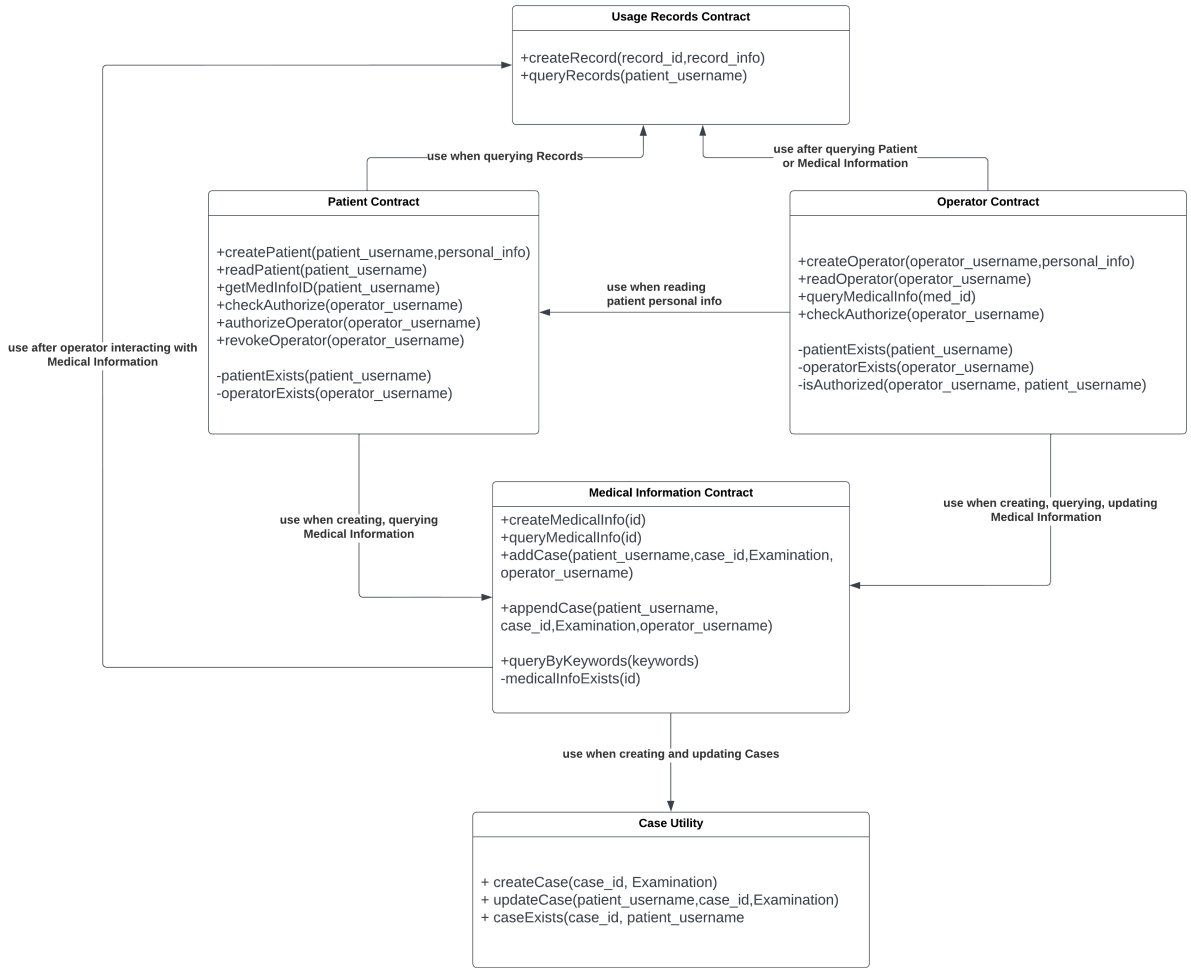


Figure 12: Contracts Structure

2.1.4.3.1 Patient Contract

This section discuss the functions in patient smart contract in more details

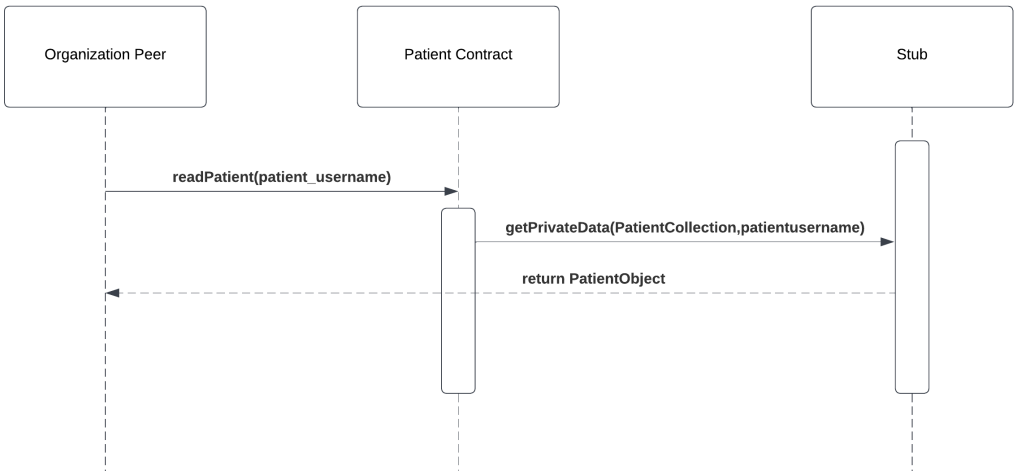


Figure 13: PatientContract Sequence diagram: Read Patient

Because we implement the access control mechanism in the chaincode level, with the same functions, we will have different kind of authorization. ReadPatient is a private function called by other functions to query patient object, this function just receive the username and return the patient object associate with that username without any authentication check.

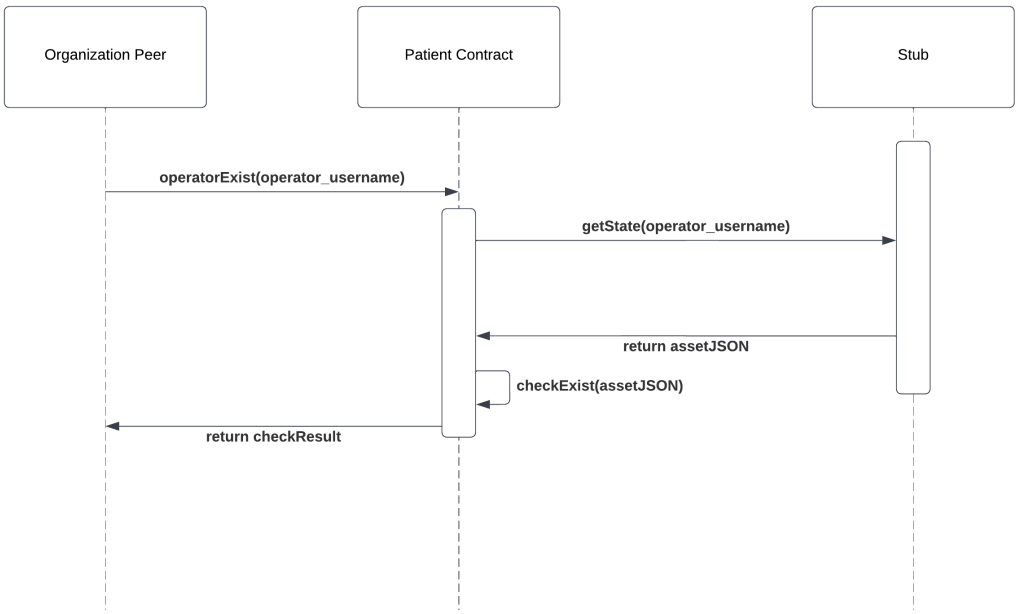


Figure 14: PatientContract Sequence diagram: Checking if an Operator exists

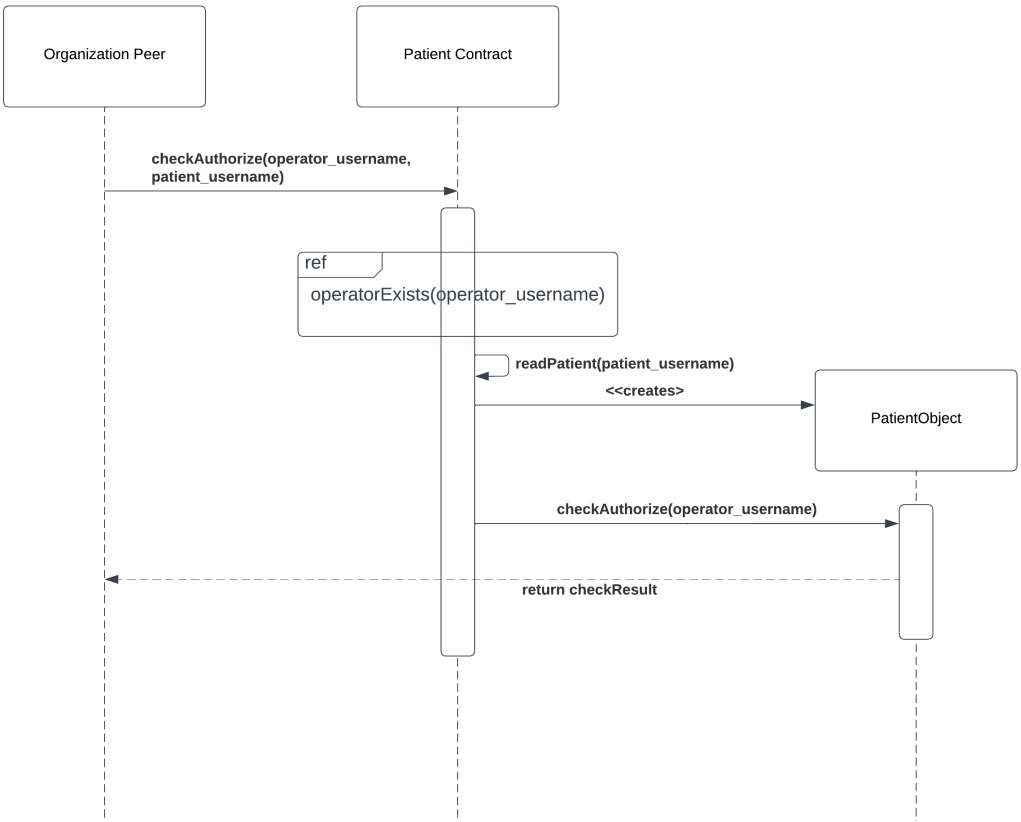


Figure 15: PatientContract Sequence diagram: Checking if an Operator is Authorized

CheckAuthorized is a private function, used to check if operator have access to a patient personal information. It receives operator and patient usernames and return a boolean value indicates whether the operator is authorized to access identifiable information of that patient. This function first check if the operator exists, it returns error if one of those usernames do not exist in the blockchain network.

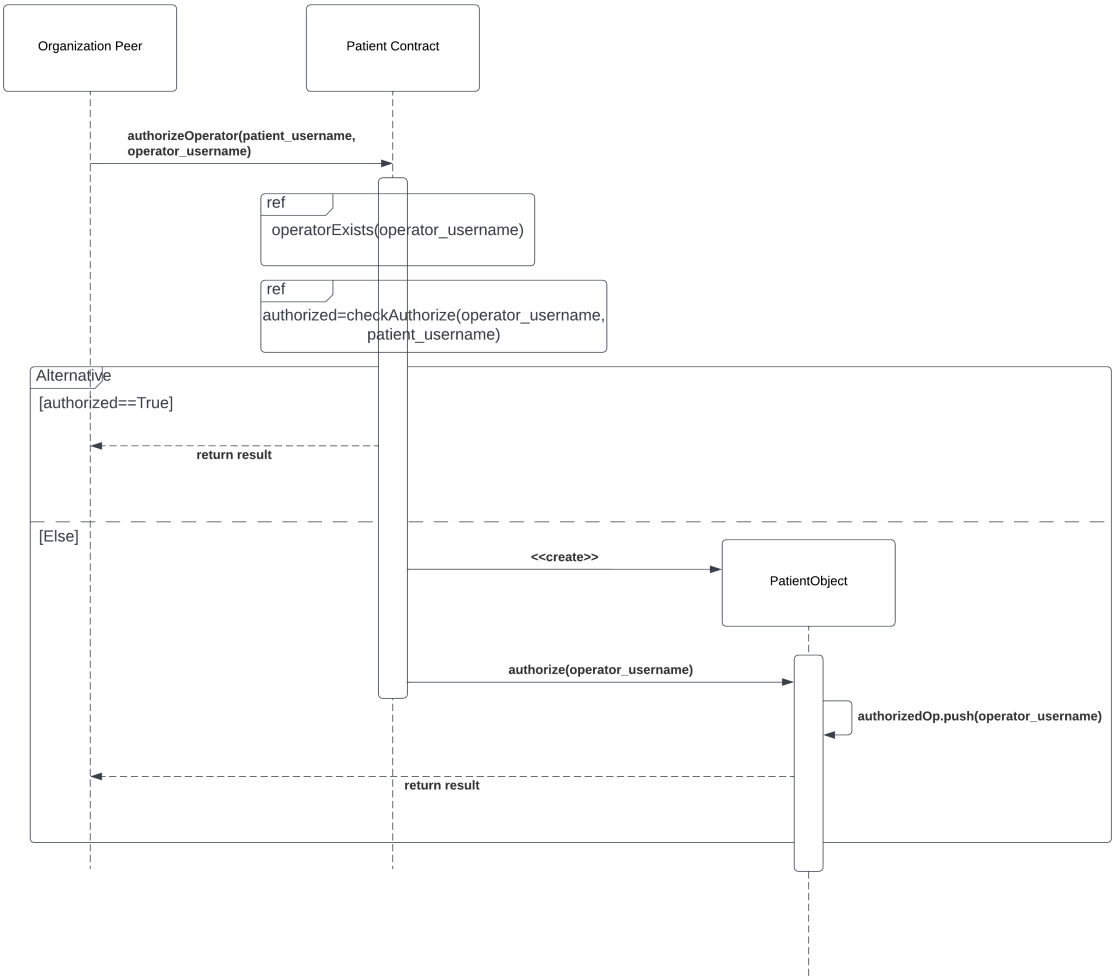


Figure 16: PatientContract Sequence diagram: Authorize an Operator

AuthorizeOperator function used by patient to authorize a doctor to access their personal information. Basically what it will do is to add a doctor username to a corresponding patient's authorized doctor list. This function receives the doctor and patient usernames and return void. We first check if the operator exists and have not already authorized. If the doctor username do not exist in the authorized doctor list, we query the patient object associated with the patient username, add the doctor username to the that patient's object list and save the new patient object to the private data collection. If the username is already there, the function will raise an error.

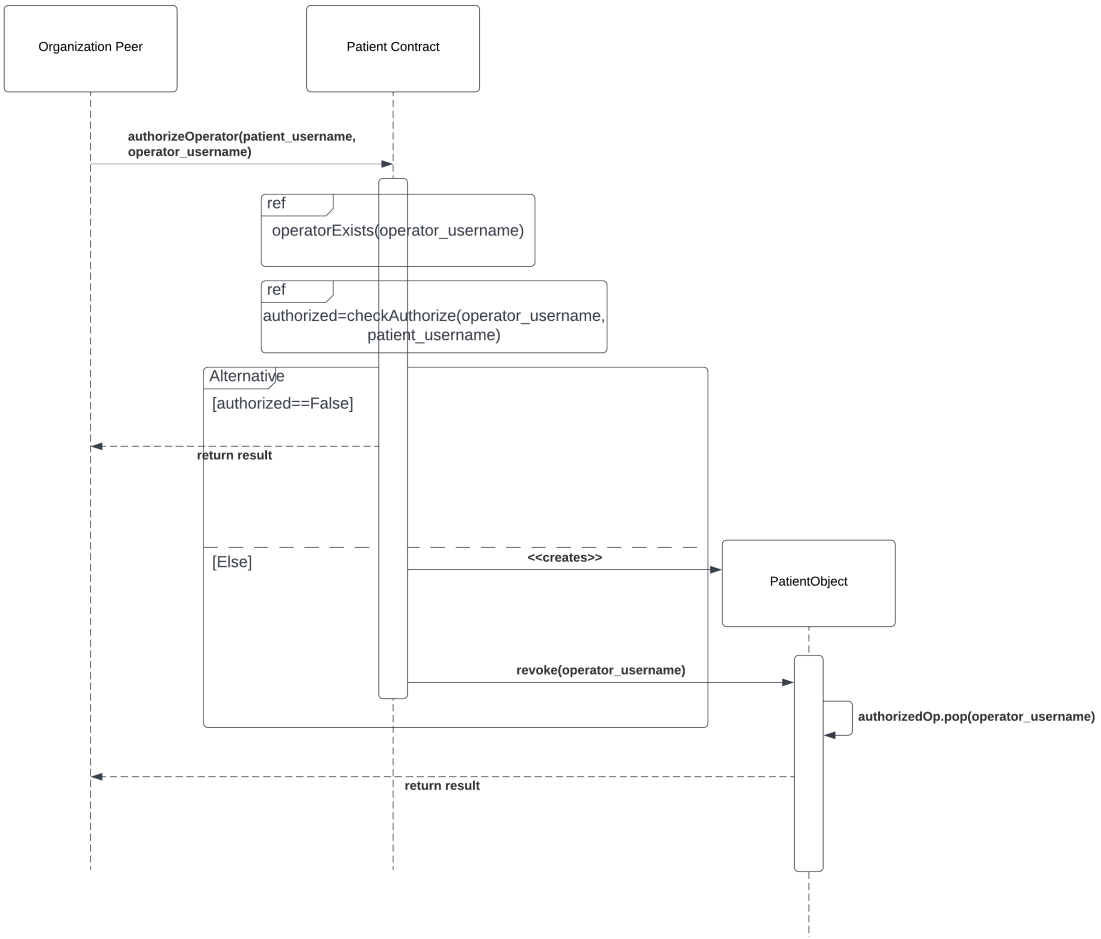


Figure 17: PatientContract Sequence diagram: Revoke an Operator

RevokeOperator function used by patient to revoke the access permission of a doctor to a patient personal information. It is opposite to the previous function because what it do is remove the doctor username from the authorized doctor list. It receives the doctor and patient usernames and return void. We first check if the operator exists and contained in the list. If the doctor username do exist in the authorized doctor list, we query the patient object associated with the patient username, remove the doctor username to the that patient's object list and save the new patient object to the private data collection. If the username is not there, the function will raise an error.

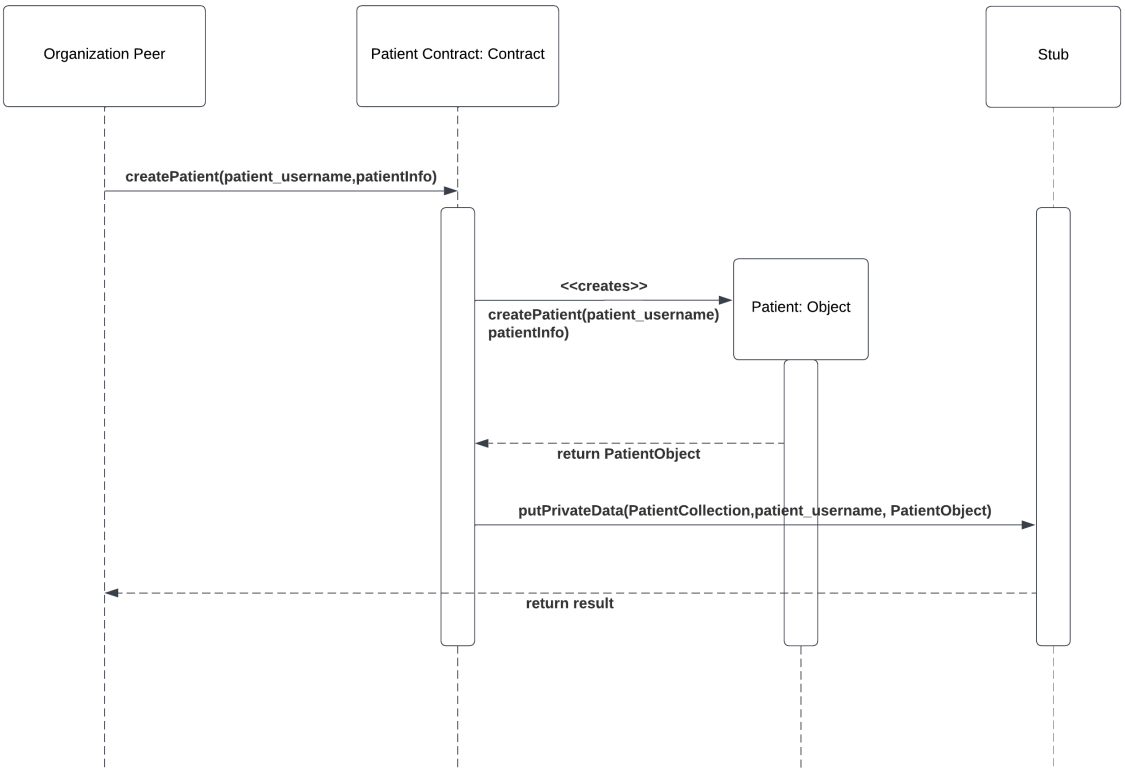


Figure 18: PatientContract Sequence diagram: Create a new Patient

CreatePatient function used to create a new patient asset in the network. It is called when a new patient is registered to the Sona system. The input of this function is patient username and personal information, and operator username. Because Sona system do not allow patient to register account by themselves, they need to first come to a medical service provider of the Sona network and open an account there. Operator username required is username of the doctor in charge of this patient, so that this doctor can be authorized to access the patient personal information.

2.1.4.3.2 Operator Contract

This section discusses the main functions in the operator smart contracts. This smart contract encapsulate all the business logic of manipulating the operator asset in the network.

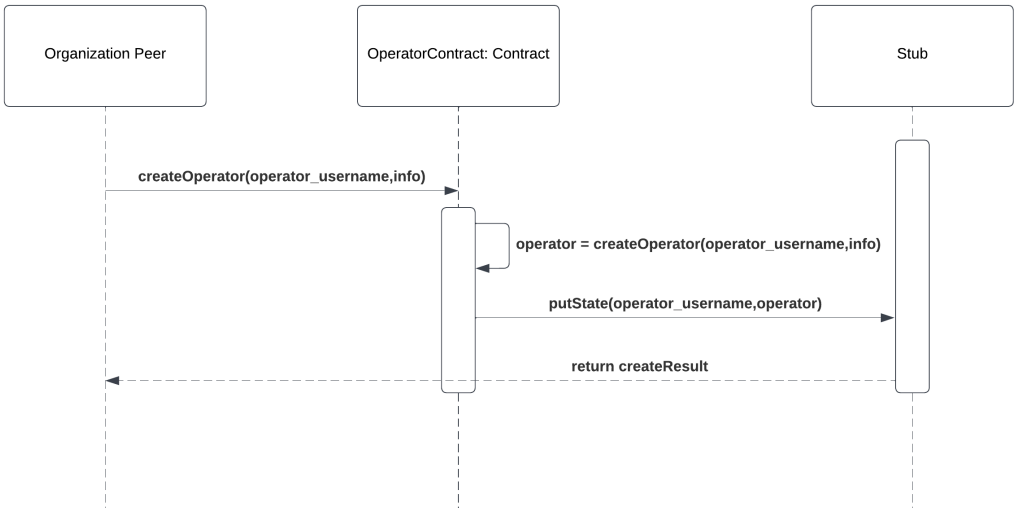


Figure 19: OperatorContract Sequence diagram: Create a new Operator

createOperator. This is the function used to create new operator asset in the network. The function receives operator username, his/her role and username, create a new operator object save in the blockchain network. This is called whenever a new operator register a new account.

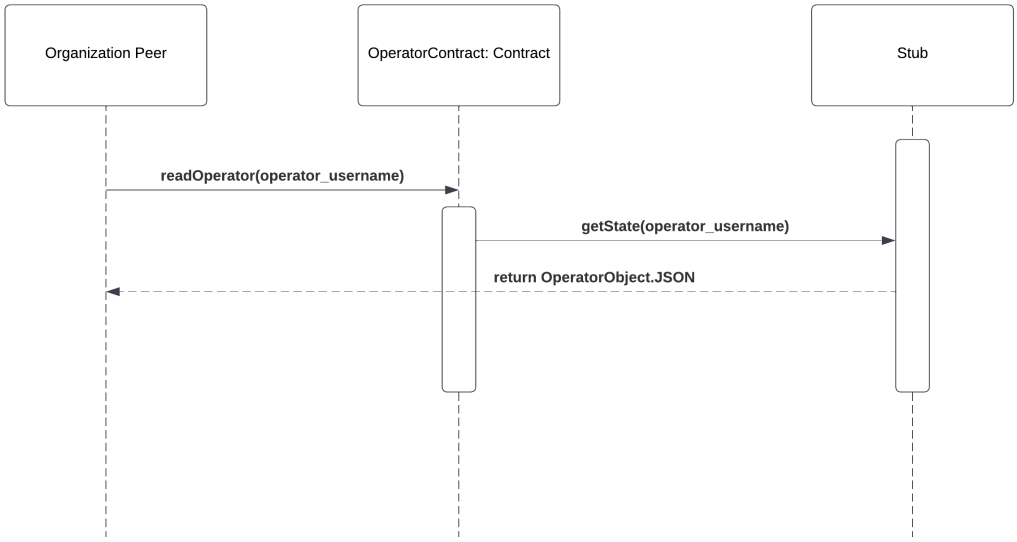


Figure 20: OperatorContract Sequence diagram: Read an Operator

queryOperator function is used to query the operator object store in the world state database providing the username of the operator.

2.1.4.3.3 Medical Information Contract

This section discuss all the main functions of the Medical Information smart contract which is used to manage the medical information record of patients.

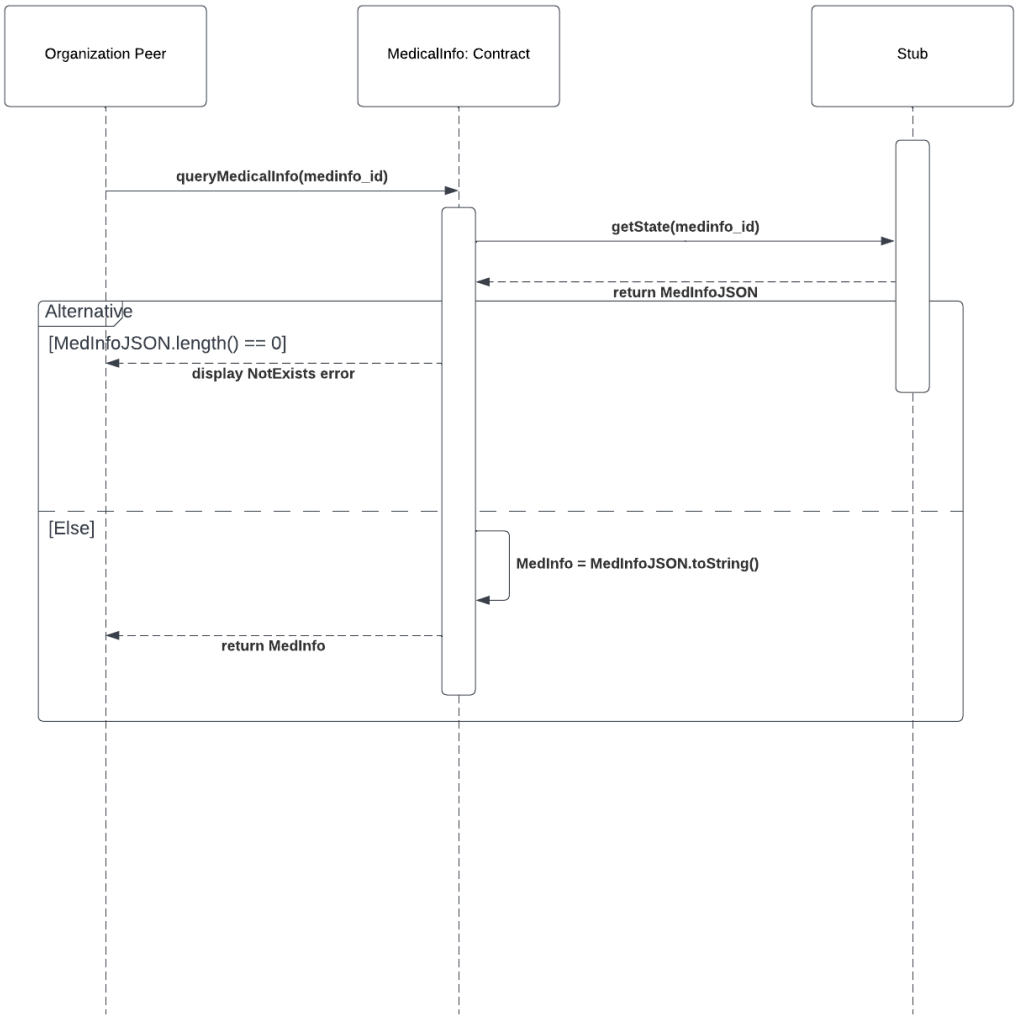


Figure 21: MedicalInformationContract Sequence diagram: Querying Medical Information

queryMedicalInfo. This is the private function used by other functions to query the medical information record by providing uuid of that record. It will raise error if the medical record associated with that uuid do not exist. If the asset exists, it will return a JSON string of that corresponding object. This function has no access control and authentication checking.

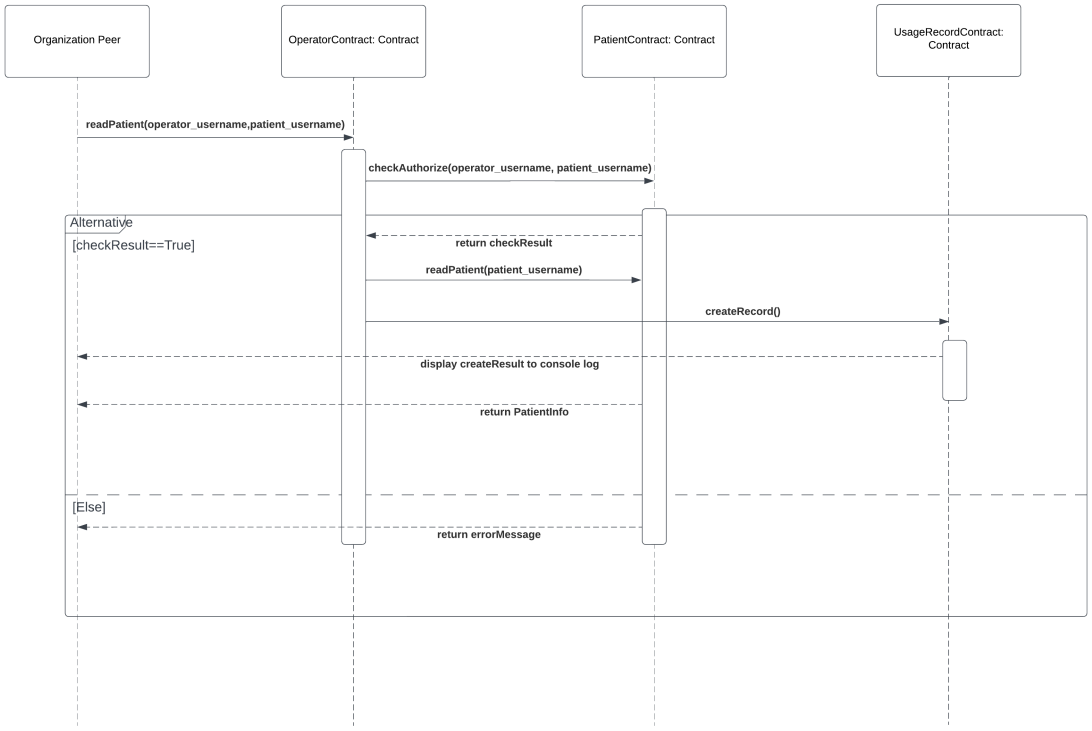


Figure 22: OperatorContract Sequence diagram: Operator querying a Identified Patient Information

operatorQueryPatient function is used by authorized operators to access the patient personal information. It receives the operator's username to check for authorization and create usage record, the patient's username is used to query the patient object associated with that username.

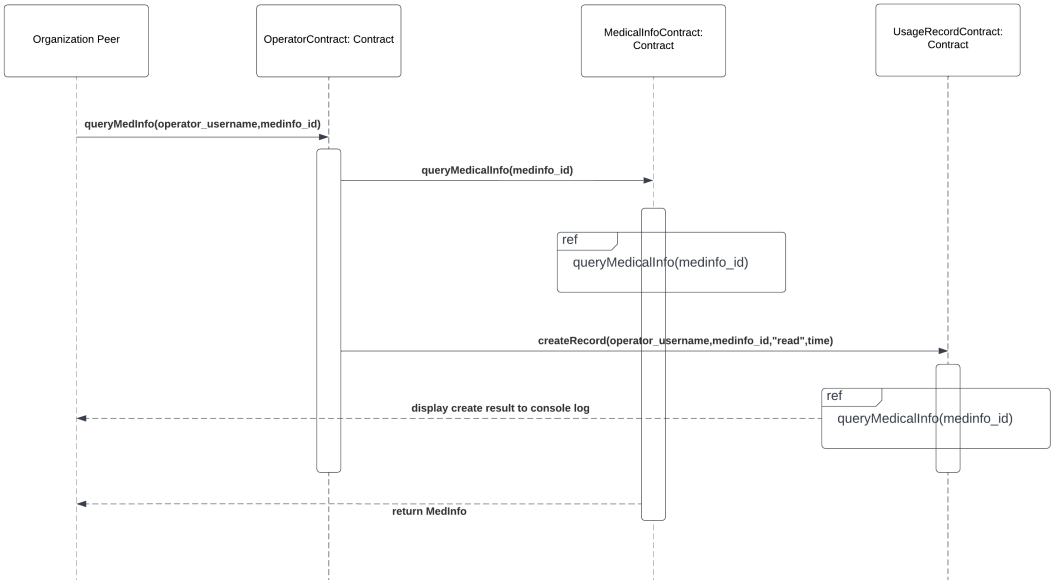


Figure 23: OperatorContract Sequence diagram: Operator querying a Unidentified Medical Information

operatorQueryMedicalInfo function is used for operator to access the medical information of patients. Operators can only use this function to query medical record as we added the access control for operators. It receives the operator username to get information to create new usage record for this access and the medical information ID is used to query the assets. This function uses the queryMedicalInfo function in the smart contract to get the necessary medical information, then it call the createUsageRecord function from the smart contract to create new usage record.

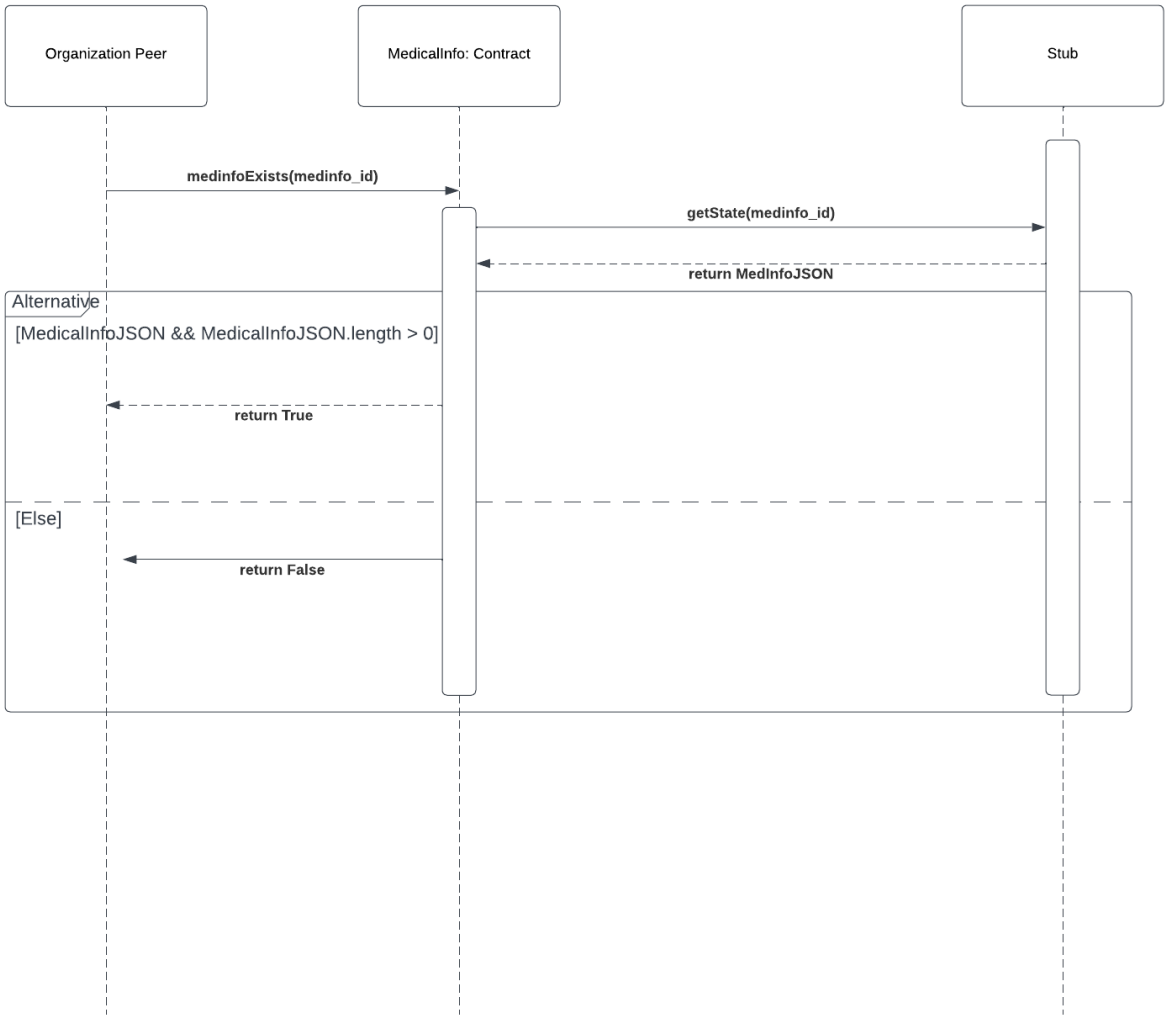


Figure 24: MedicalInformationContract Sequence diagram: Check If a Medical Information exists

medicalExists is a private function used by other functions to check if a medical information of a patient exists providing its ID.

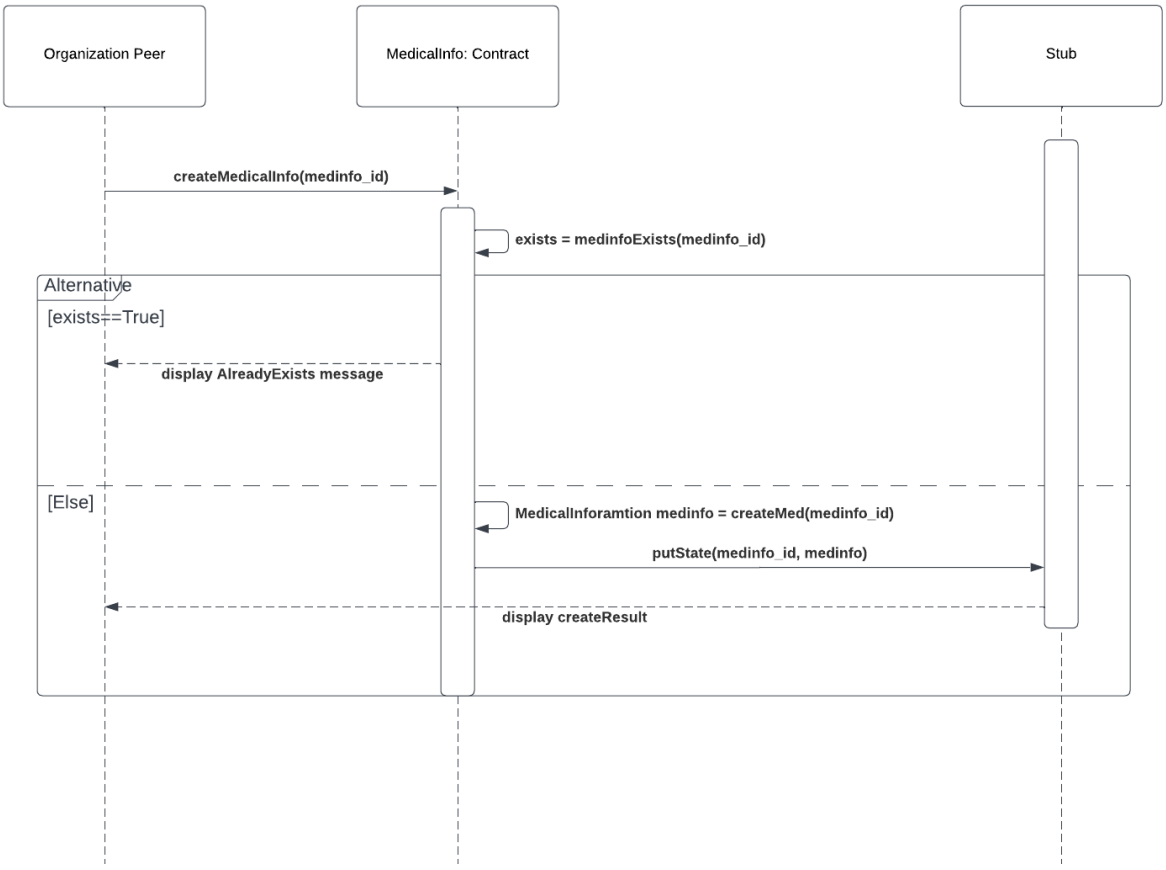


Figure 25: MedicalInformationContract Sequence diagram: Create new Medical Information

createMedicalInfo function is used to create new medical information for new patient in the network. It is not called directly by any user but called by the createPatient functions in the Patient smart contract. We need to provide the function an uuid for this function to used as ID for new medical information cause we cannot generate uuid in smart contract.

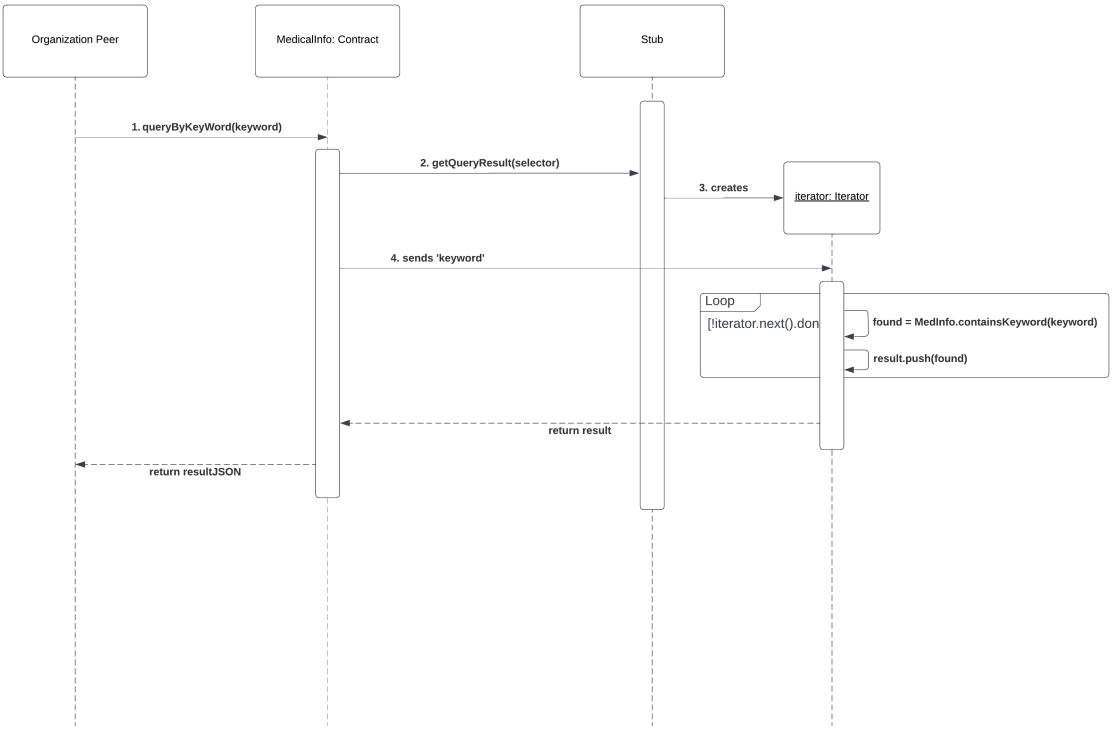


Figure 26: MedicalInformationContract Sequence diagram: Query all Medical Information by Keywords

queryByKeywords function is used by researchers to query all the public medical records containing information they interested in. It receives a list of keywords and using the getQueryResult which is used to query complex queries to query all the medical information containing those keywords. The function return back the JSON string of the array of those MedicalInfo objects.

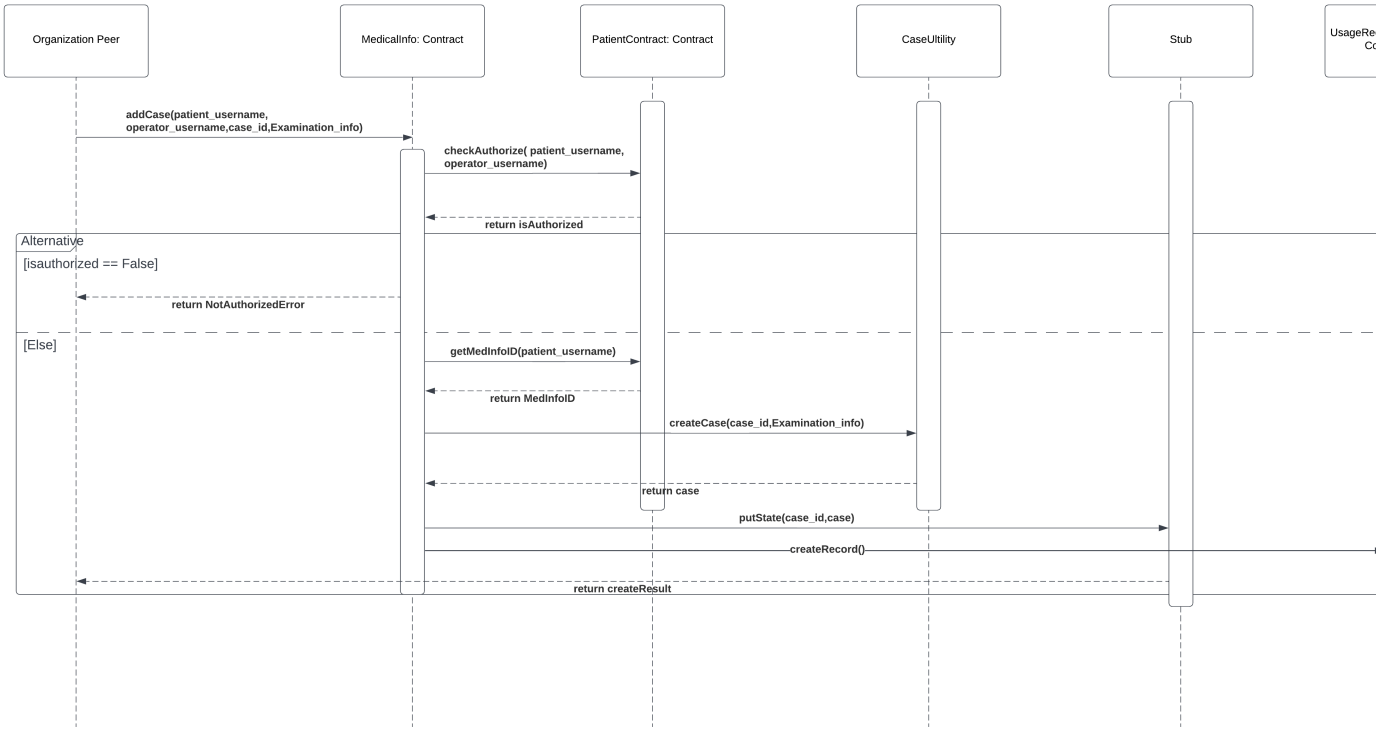


Figure 27: MedicalInformationContract Sequence diagram: Create a case and add it into a Medical Information

addCase is the function used by doctors to add new case to the patient's medical information. The input of this function is patient username, operator username, case's id and examination information. It first using the operator username to check if the operator is authorized to modify the patient's medical information. Then it get the MedicalInfo object associated with the patient username, create a new case with the examination in the input then add that new case to the MedicalInfo object.

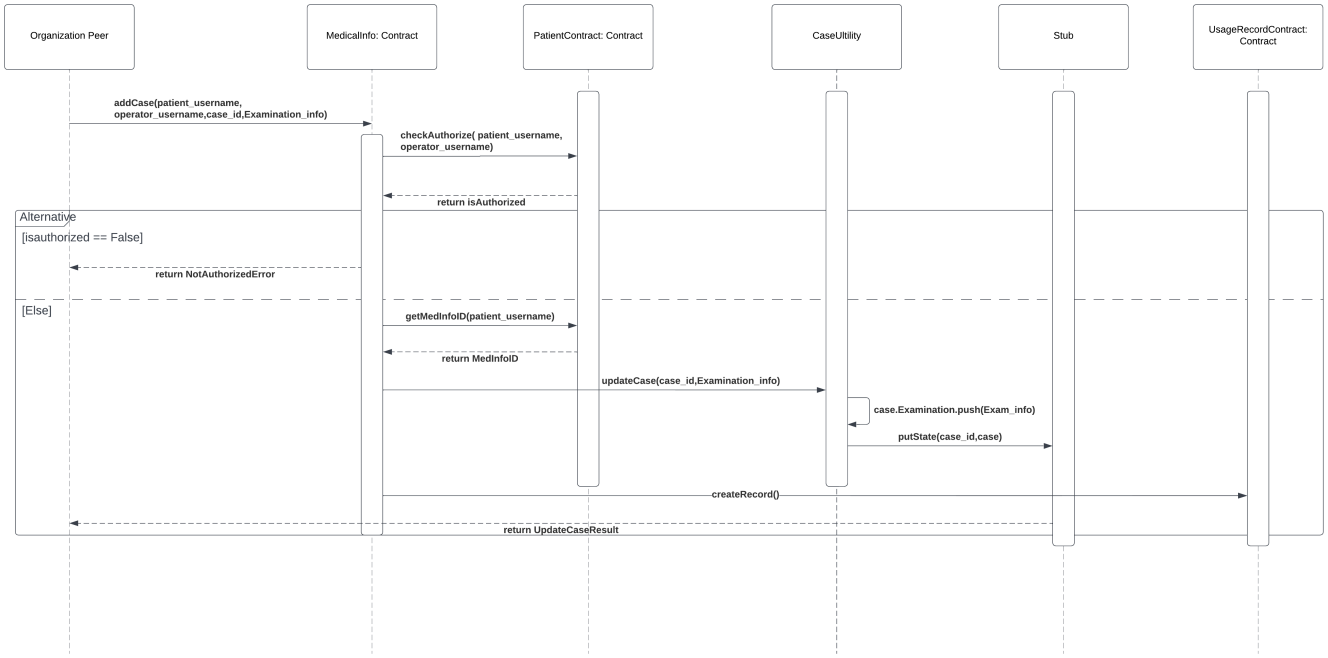


Figure 28: MedicalInformationContract Sequence diagram: Append an medical Examination into an existed case

appendCase function is used by doctors to append new Examination records to existing medical information of patients. The inputs of this function are receives patient's username, operator's username, case's id, examination information. It first used the operator username to check if the doctor is authorized, if not it will raise an error. Then it use the patient's username to query the medical information of that patient, create a new examination object with the examination information in the inputs and add it to the MedicalInfo object.

2.1.4.3.4 Usage Records Contract

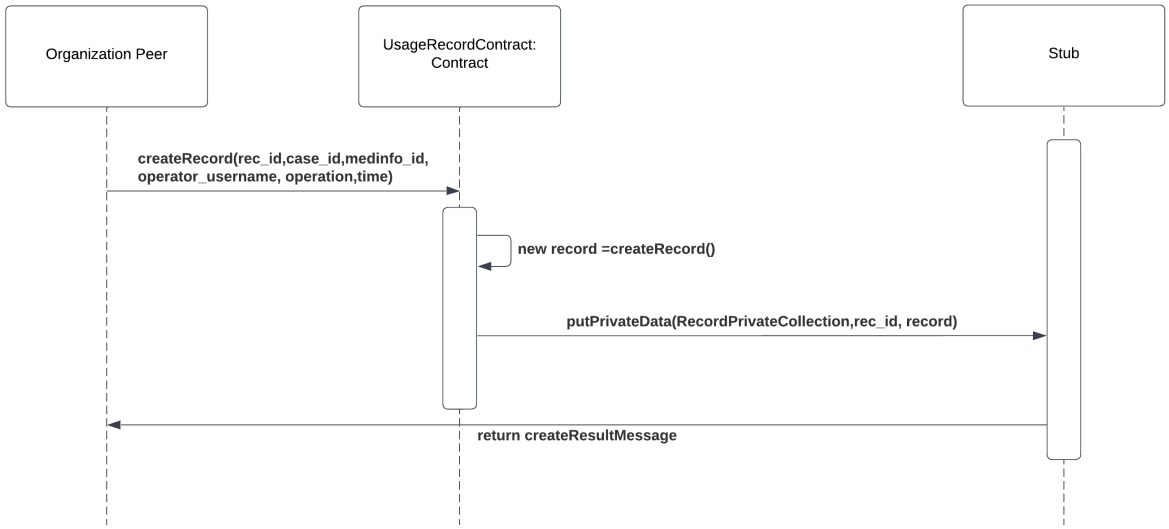


Figure 29: UsageRecordContract Sequence diagram: Create an Usage Record

CreateRecord function is used to create new usage record and store it in the private data collection of usage record. It is not directly called by operator but call by other smart contract functions which manipulate the MedicalInfo. This function receives case's id, MedicalInfo's id, username of the operator who manipulate the MedicalInfo, time, and a uuid used as ID for this new record. We need to pass the uuid from the server as the consensus mechanism in Hyperledger Fabric do not allow us to generate uuid in the smart contract. We first create a new Record object then put it to the private data collection for usage record.

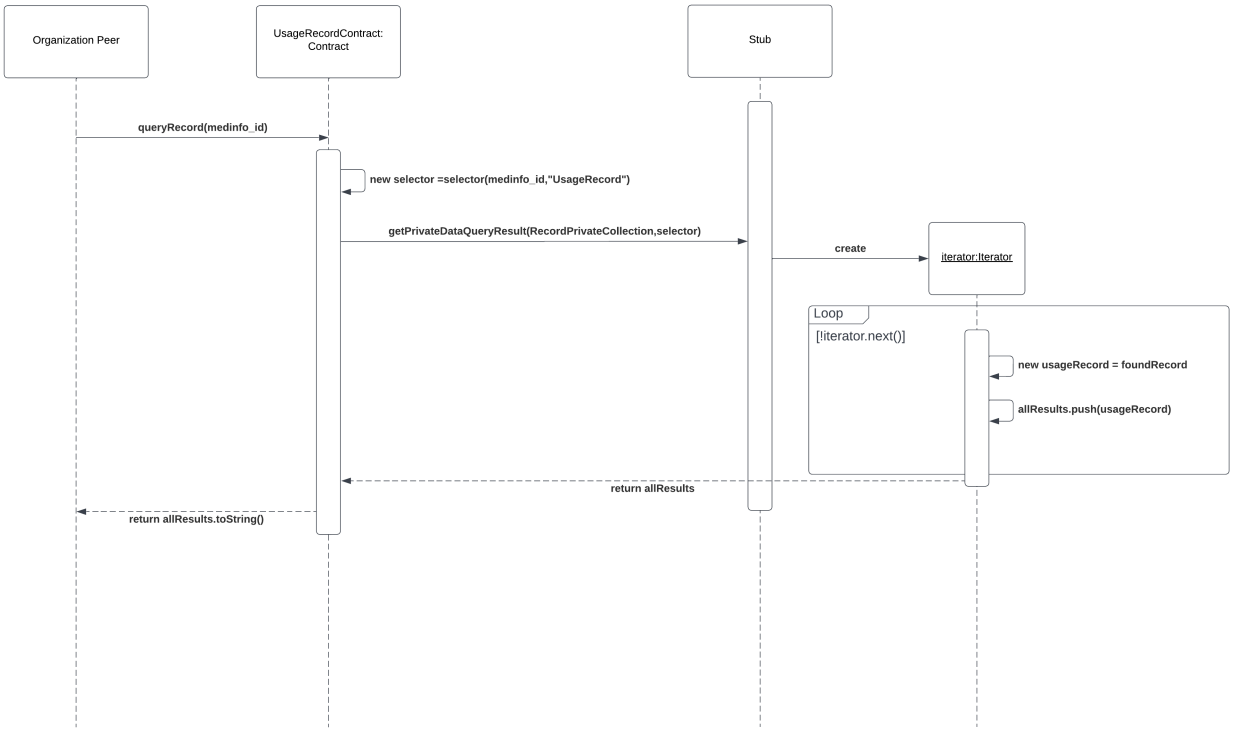


Figure 30: UsageRecordContract Sequence diagram: Query all Usage Record from an existing Medical Information

QueryRecord function is used by patients to query all the usage record associated with their medical information. It receives a uuid and then query from the usage record private data collection all the usage records whose MedicalInfoID matches the provided uuid. Then it return the list of usage records back to user.

2.2 Detail System architecture

2.2.1 System's components introduction

Sona system contains two main components: frontend and backend. Backend contains an operating network, which is empowered by blockchain, and the application backend architecture, which allows application users to communicate with the blockchain network.
The network is utilised by permissioned blockchain based technology called 'HyperLedger Fabric', the blockchain is kept in the ledger and is in charge of keeping track of all transactions of evaluating or invoking the 'world state'[13]. The 'world state' serves as a logical database and stores the blockchain's most recent state. For the 'worldstate', Hyperledger Fabric offers levelDB (by default embedded key-value database that can only be queried by key) or couchDB (a clustered database that allows us to run logical database server on any number of servers or Virtual Machines [2], which also mainly supports JSON data and allows more complicated queries). Due to some complex logical behaviors happening in a transaction, we decide to use couchDB for Sona system.
On Digital Ocean, an American cloud infrastructure provider help us to build robust applications using a comprehensive portfolio of compute, storage, database, and networking products [4], we also host a machine that is primarily used for deploying the Hyperledger Fabric network. Digital Ocean offers platforms for cloud infrastructure-as-a-service to developers, startups, and SMBs. Although ExpressJS server for APIs might be operated on the computer from Digital Ocean Providers, we chose to run API Server on a local machine instead due to the limitations of a small-scale system and to improve response time.
For Application Backend,the API Server, which enables application user to interact with the Hyperledger Fabric network, uses ExpressJS, a framework based on NodeJS that provides a robust set of features for web and mobile application [5]. In order to communicate with the network, we also need to have the platform-specific HyperLedger Fabric CLI tool binaries and configuration files.
The system also uses MongoDB, a document database with the scalability and flexibility that ensure end-to-end security [16], for storing user accounts information with purpose of serving authentication function. The other reason for this additional database is to avoid burdening the ledger from blockchain network when scaling up the system
For the frontend, we build our whole web application's Frontend with ReactJS, a JavaScript library for building userinterfaces that has the combinations between HTML, CSS and JavaScript [17]
There are three different roles of end users in our system: Patient, Doctor, and Researcher. patient and doctor users have the identities to interact with the network in an Organization called 'Clinic', while researcher users participating in the network by the identities provided by different organization named 'Research Facility'. These three types of users can interact with the data of the ledger by distinctively corresponding policy written in the Smart Contracts, including query and update data. The section Access control will discuss in greater depth

about these various application roles.

2.2.2 Backend architecture

2.2.2.1 API Routes

Our system has the main API routes divided into two parts. 'Public Routes' are in charge of handling authentication and registering function, which operate only with MongoDB database side. 'User Routes' are responsible for providing methods for three types of Users: Patients, Doctors, and Researchers to interact with the chaincodes through Gateway, which will be explained later in the section.

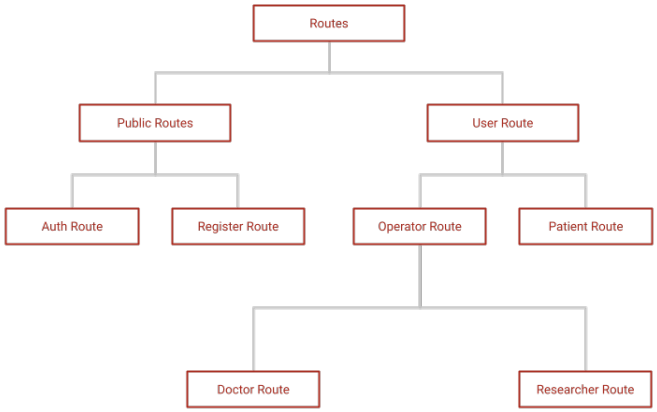


Figure 31: Structure of Sona's API Routes

The responsibilities of 'Auth Route' and 'Register Route' are described by their names, serve for behaviors related to authentication and registration. The 'Patient Route' allows patient users to query their own personal information and medical information including their medical cases, search for doctors , and authorize more or revoke doctors. 'Doctor Route' is in charge of letting doctor users query their own information, read their patients data, and updating their patients' medical information including create medical cases and updating existed medical cases. 'Researcher Route' enables users from the Research Facility to query unidentified health information and to search some unidentified medical data with some specific keywords of sickness.
Some of the methods from User Route would trigger the function that creating a record written in the Smart Contract ,enabling patients to control and monitor how their data are used.

2.2.2.2 Access Control Mechanism

It is a great start to go over our involvement in this application to get a better understanding of how our authentication system operates. Our system has four key functions and is structured in accordance with The Role-based Access Control[18] methodology.
The very first role is known as the "**Network Admin**," which is covered in greater detail in section 2.4.1 Network Admin. For the purpose of simplicity during this stage of the project, there are merely two "Network admin" entities in the network, Clinic Network Admin and Research Network Admin, one for Clinical Organization, one for Research Facility Organization, and it will be configured when the we first sets up the HyperLedger Fabric network for the application. The certificates of two Network Admins will be stored in the network file system folder called 'wallet'.The primary duty of this organization is to give user identification apart from Clinic and Research Facility

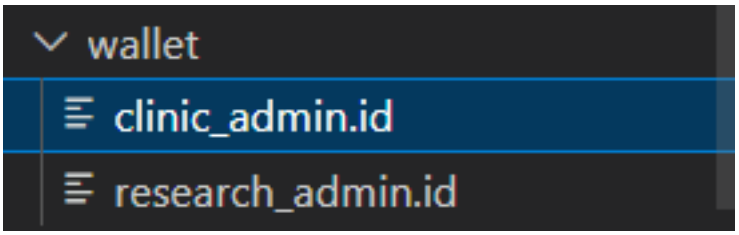


Figure 32: Network Admin's credentials stored in wallet

Three other roles have same name type, called: 'Medical User'.
The "Patient" is the second role; more details on this position are provided in **section 2.3.2 Patient**. Simply said, this is a channel member who is a patient of a clinical organization. In addition to being locally saved on the server, just like "Network Admin," the username of patients will also be stored inside the ledger of the blockchain network. In order to avoid disclosing patient' personal data to third parties, we can utilize this method to retain this role's account information in a private data collection.
The third role is "Doctor," and **section 2.3.3, Doctor Side**, contains further details concerning this job. This character will depict a Clinic doctor who want to contact the channel. A "Doctor" can only be registered by the "Clinic Network Admin" from the Clinic Organization and is only a member of one Clinic.
The fourth role is "Researcher," which is covered in greater detail in **section 2.3.4 Researcher Side**. This character will depict a researcher who wishes to join the channel from a research facility. Only the "Research Facility Network Admin" from the Research Facility Organization is authorized to register as a "Researcher," who is the property of the Research Facility Organization.

2.2.2.3 User Authentication Mechanism

The same authentication process is used by the patient, doctor, and researcher. Because each of these roles must communicate with the Hyperledger Fabric blockchain network, their authentication mechanisms are intricate. On the login page, users may alternatively log in via the webserver operating on port 8080.

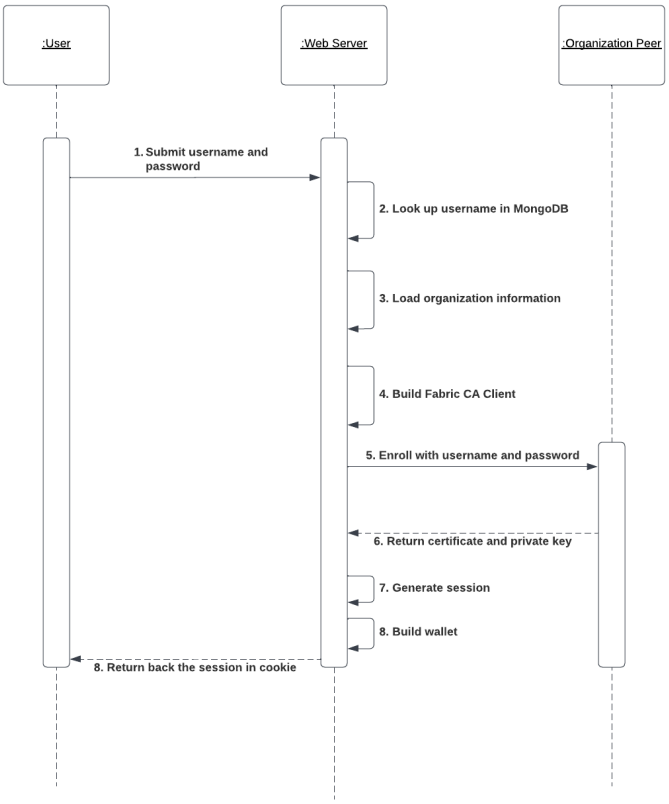


Figure 33: User Authentication's Sequence Diagram

Because the User's credentials are kept inside the Ledger, it is mandatory for the operation to involve the Organization's peer in addition to the Web server for authentication. When the User enters their username and password during web server login, the procedure starts as usual. We need to construct a Fabric CA Client [7] using the Hyperledger fabric NodeSDK library[9] in order to connect with the organization's peer, which is a part of the Hyperledger Fabric network (For a detailed explanation of how to connect to the Hyperledger Fabric Network, see **section 2.3.2 Server and HF connection**). The server will now utilize the provided login to access the information about the account to which it belongs, which is likewise kept in the couchDB ledger. The server will provide an error message if the username cannot be found or the password is incorrect.
The Fabric CA Client can be started once all the data for the organization to which the provided login belongs has loaded properly. The server may now enroll the provided username

and password with the organization's peer with the aid of the Fabric CA Client. The peer will return the certificate and private key of the enrolled User if the enrollment is successful, indicating that the credential is acceptable for enrolment. These return values will be kept in the wallets and are essential for connecting with the channel in a subsequent procedure[10]. If everything up to this point has been legitimate, the SHA-256 hash methods will be used to create this User's session in conjunction with the username and server secret key. After that, all of the certificates and the private key that the User granted in the preceding stages will be stored in a file called "*username*.id" that is created by the wallet of the User's channel. The formed session will then be sent back to the browser and saved in the cookie so that it may be utilized again. If the login credentials are legitimate, the procedure will conclude by either redirecting back to the login page in the other way or to the User dashboard.

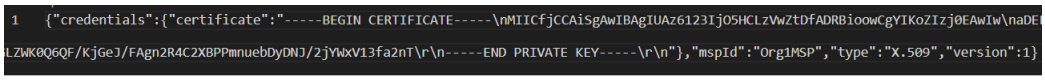


Figure 34: Content of User's Identification provided by Network Admin

2.2.2.4 Authorization mechanism

This section will discuss the operation of the authorization method used by our ExpressJS server and explain how it can guard against both horizontal and vertical privilege escalation.

Our Smart Contracts are written in TypeScript, a language that is mostly the same with JavaScript but implements more Object Oriented Programming concepts. With this language's advantages, we have been able to create several 'identities' such as 'Record', 'Medical Information', 'Case', etc. including three main types of users (patient, doctor and researcher). However, when it comes to authentication and authorization in Hyperledger Fabric, **a lot of information from a user account are required for establishing connection to the network.** Therefore, it is essential to create a class outside of the Ledger that stores all of the connecting information for a user when they sign in to the web server. We decided to create this class at the API Server side.

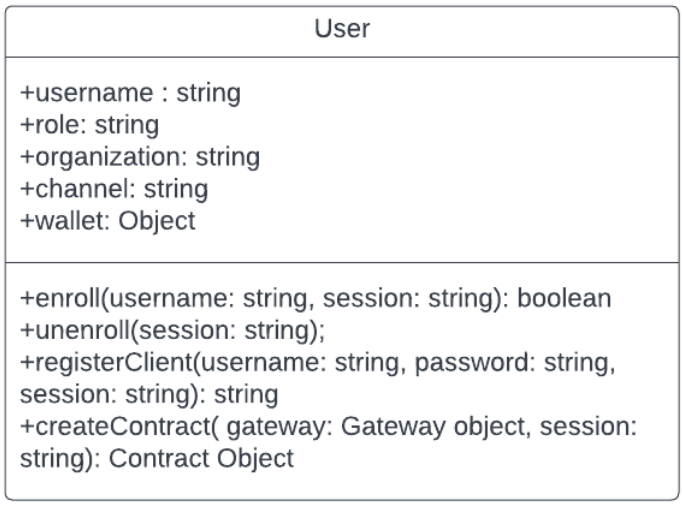


Figure 35: User Class Diagram

This class will include all the data required for the server to establish a connection with the particular user's channel. Patient, Doctor, and Researcher user classes will be used. The server will check the user's role when they log in, and they may then utilize the enrollUser and unEnrollUser methods (which is used when logging out). By invoking the method enrollUser in collaboration with the authentication process, all of these properties will be created (for more information about how to connect with the Hyperledger Fabric Network, please read section 2.2.1.3 Server and HF connection).
The Medical User (patient, doctor, or researcher) will have a session after successful authentication, which will be saved inside the browser's cookie. Together with the session, a user

class for that particular medical user will be created, and it will be saved on the server in an object named loginUser. In this approach, the user session that was previously produced will be mapped to the User class.
In order to minimise information exposure during communication and prevent users from changing their roles or organization configuration, this method seeks to limit user input on each request. The flow of carrying out one of the Medical User's actions is shown in the image below.

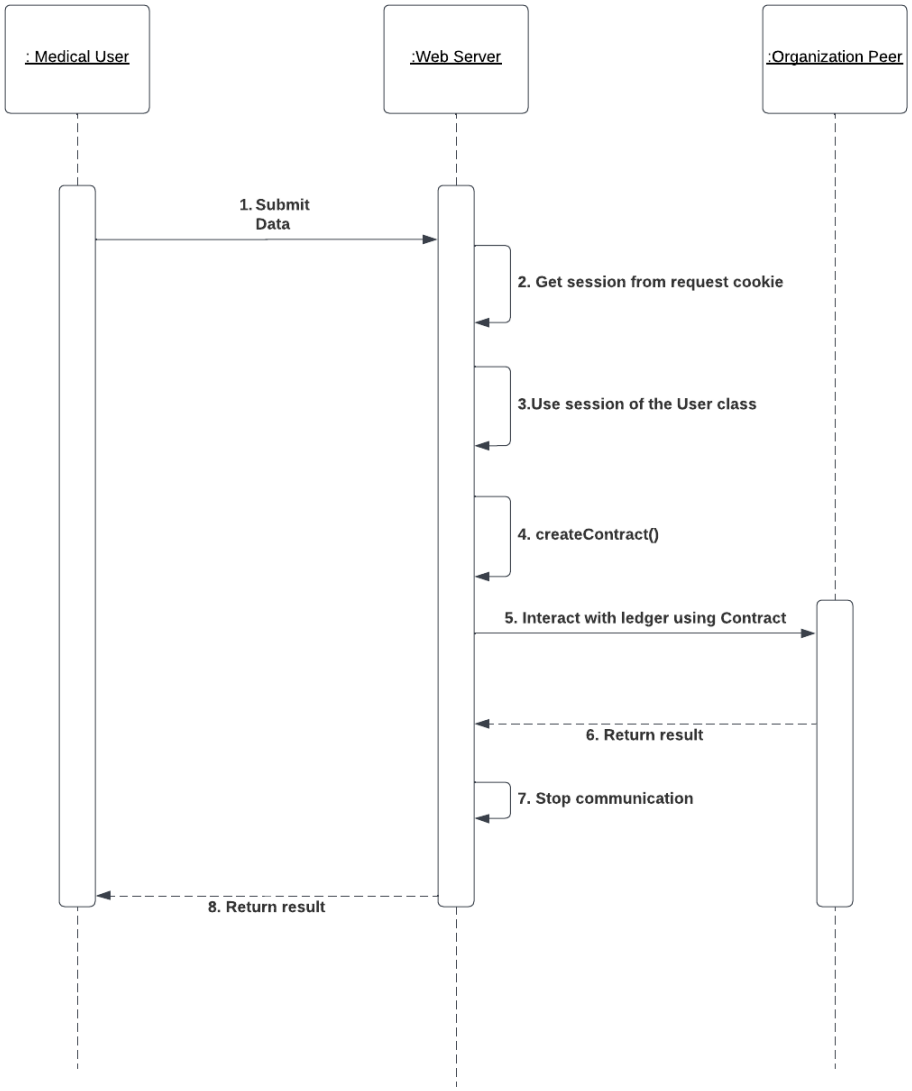


Figure 36: User Authorization Sequence Diagram

The Medical User will cooperate with user credentials inside the request to submit the needed data. The web server will then load the User class that is associated with that session from the loginUser Object after extracting the user's session from the request. It will not be able to obtain the User Object from the loginUser Object if the session is invalid or the user is

not signed in. The server simply builds the contract for connecting to the Channel using the method getContract of that user after receiving the User class of that user (Please read section 2.2.1.3 Server and HF connection for additional details on connecting to the Hyperledger Fabric Network.)
But since everyone has sessions, the issue is how to authorize between the patient, doctor, and researcher. The attribute "role" now enters into play at this point. The 'role' property will be examined after utilizing the session to obtain the User object since each kind of Medical User has a distinct string value in 'role,' allowing us to use this attribute for verifying. The NodeJS Express Middlewares [6] technology is used in routing to achieve this inside the server.

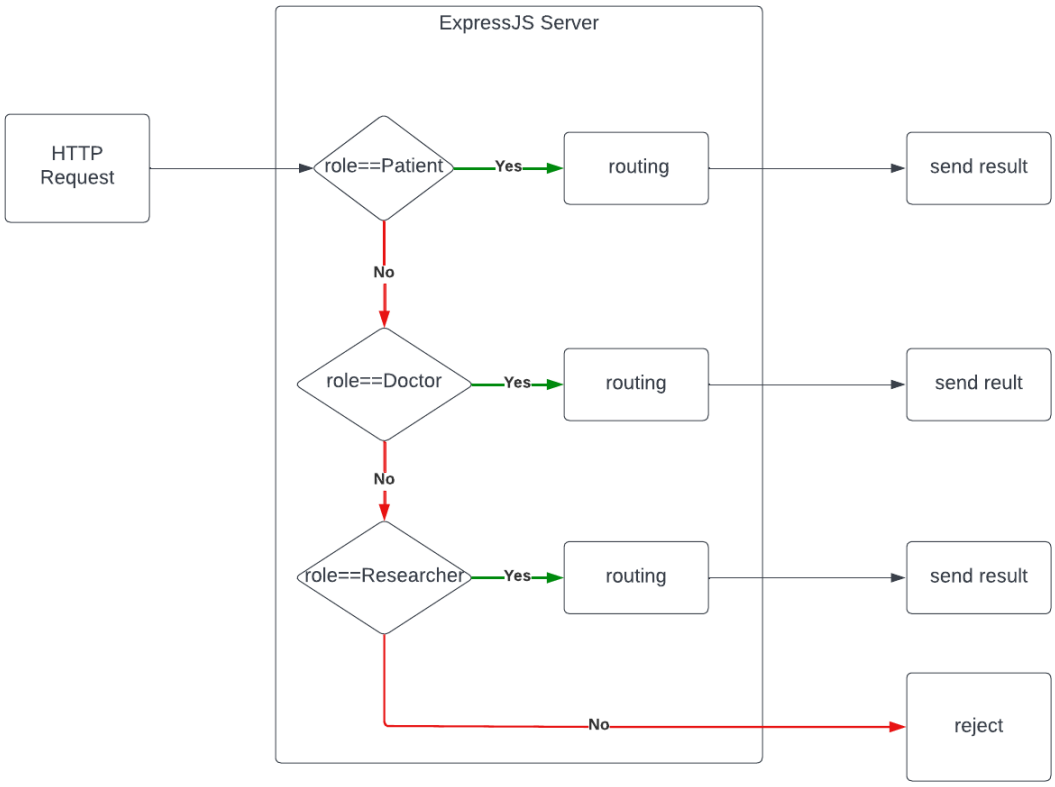


Figure 37: Medical User Role-based Authorization Flow Diagram

2.2.2.5 Private Data Protection mechanism

2.2.2.5.1 What is private data

The option to create a separate channel with just the organizations that require access to the data exists when a group of organizations on a channel need to keep data secret from other organizations on that channel.[11] However, setting up distinct channels in each of these scenarios prevents use cases when we want all channel participants to see a transaction while keeping some of the data private and adds more administrative work (maintaining chaincode

versions, rules, MSPs, etc.).
Fabric enables the creation of private data collections, enabling a specified subset of organizations on a channel to endorse, commit, or query private data without the need to establish a separate channel.[11]
A chaincode specification can explicitly define private data collections. Every chaincode also includes an implied private data namespace for private information particular to a certain organization. If we want to store private information about a single organization, such as information about a resource it owns or an organization's approval for a particular step in a multi-party business process implemented in chaincode, we can use these implicit organization-specific private data collections.

2.2.2.5.2 What is private data collection

As it was stated by Hyperledger Fabric, a collection is the combination of two elements:

1. **Actual private data** is transmitted peer-to-peer via the gossip protocol to only the organization(s) with access rights. This information is kept in a private state database that is accessible by chaincode on the peers of approved entities. The ordering service is not engaged in this and is not privy to the personal information. Note that in order to bootstrap cross-organizational communication, anchor peers must be set up on the channel and each peer must be configured with CORE PEER GOSSIP EXTERNALENDPOINT since gossip distributes the private data peer-to-peer across approved organizations.[11]
2. **A hash of that data**,that is approved, sorted, and written to each peer's ledger on the channel. The hash acts as proof of the transaction, validates the state, and may be used for auditing.[11]

The following diagram illustrates the ledger contents of a peer authorized to have private data and one which is not[11]

2.2.2.5.3 Why does SONA need private data collection

Since the main purpose of implementing blockchain in managing the healthcare system is protecting patient-identified medical data and personal data, we need to provide an extra layer of encryption for their information. When scaling up the system, there would be many more organizations including more clinics or more research facilities, private data collection would serve more effieciently in terms of managing confidential data between organizations.

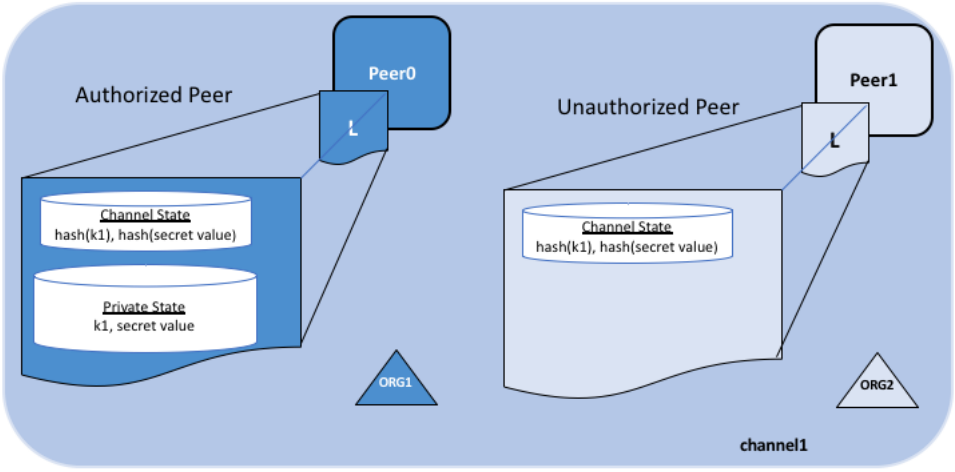


Figure 38: Private Data Collection Example [11]

2.2.2.5.4 How does SONA implement private data collection

For the sake of simplicity, we just have a private data collection for organization 1 - clinic 1. By another words, the Organization peers in Clinic 1 will have a private part in the ledger for Patients Information and Usage Records. The peers in Research Facility Organization, which are not from Clinic 1, will obtain that private part as hash values. This helps prevent the act of privilegde escalation in Researcher side , even if some one in the outside organization break the role-based access control mechanism, they would still not receive the private information of Clinic 1.
There are some other policies that helps to operate the private data collection, which is described at Security Implementation section.
Some cases of transaction that is prohibited by Private Data Collection policy:

- Researcher trying to create another new Patient, which is prohibited since the role Researcher does not have ability to execute that function.

```
patient_contract.submitTransaction('CreatePatient', 'tu cam', new_patient, medid, '2463','cat street',   '44/3',
'female', 'Doctor1');

******** FAILED to run the application: Error: No valid responses from any peers. Errors:
    peer=peer0.org1.example.com:7051, status=500, message=PUT_STATE failed: transaction ID: 1bcef669f9047024bb27e87f0fc7
39083074597de87984e788d61f4606620716: tx creator does not have write access permission on privatedata in chaincodeName:f
abcar collectionName: PateintIdentifiableData
    peer=peer0.org2.example.com:9051, status=500, message=PUT_STATE failed: transaction ID: 1bcef669f9047024bb27e87f0fc7
39083074597de87984e788d61f4606620716: tx creator does not have write access permission on privatedata in chaincodeName:f
abcar collectionName: PateintIdentifiableData
```

Figure 39: Prohibition of creating patient

- Researcher trying to query Patient personal data, which is prohibited since the only the Doctor authorized by that Patient could have that right to invoke this type of transaction.

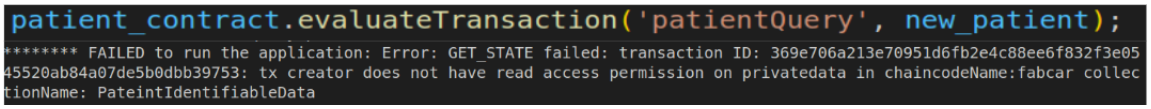


Figure 40: Prohibition of querying patient personal data

- Researcher trying to view the usage records of a patient, which is prohibited the system does not allow anyone except that patient to see his/her usage records.

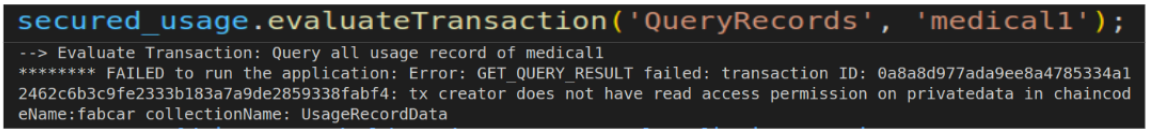


Figure 41: Prohibition of reading patient usage records

2.2.2.6 Privilege escalation protection

We must define privilege escalation in order to set the stage for this discussion. Privilege escalation, or session hijacking, refers to the process by which a single user can take advantage of a bug in the system's architecture to carry out only actions that are allowed for users with greater privileges. This defect can be derived in one of two ways:

- **Privilege elevation**, often referred to as **vertical privilege escalation**, occurs when a user or application with lower privileges has access to features or material that are intended for users or apps with higher privileges [3].
- **Horizontal privilege escalation** occurs when a regular user accesses features or material that is only accessible to other regular users [3].

Neither vertical nor horizontal privilege escalation may affect the web server. As was previously noted, the session cookie plays a crucial part in the authorization mechanism and is necessary for any user-side request to access to the Hyperledger network. The only issue left is that the user can change their session to use a different identity in order to access other features or contents. However, the server's secret key and the username haveh are used to construct the session, making it impossible to build a session using the secret key. In a different scenario, the hacker gets the username and attempts to brute force the password for the correct session; however, if the keyspace is large enough, it will take a long time.
Vertical session hijacking and safeguarding is the next area of concern. Because all three need sessions for authorisation, the web server places a high priority on keeping the privileges of Patients, Doctors, and Researchers separate. According to the earlier description, we could extract the Medical User's User Object using the session information we acquired from the request cookie. For the Medical Users, the characteristic "role" is set in a different way. As a result, the server can quickly determine a User's role by checking at the field "role" in the

User instance.

2.2.3 Server and Hyperledger Fabric Connection

Our primary technology, Hyperledger Fabric, is the foundation upon which this project is built. As a result, the connection between the ExpressJS-based server and the Hyperledger Fabric network is crucial to the success of this project. The primary technology that connects the server to the Hyperledger Fabric network will be described in this section.

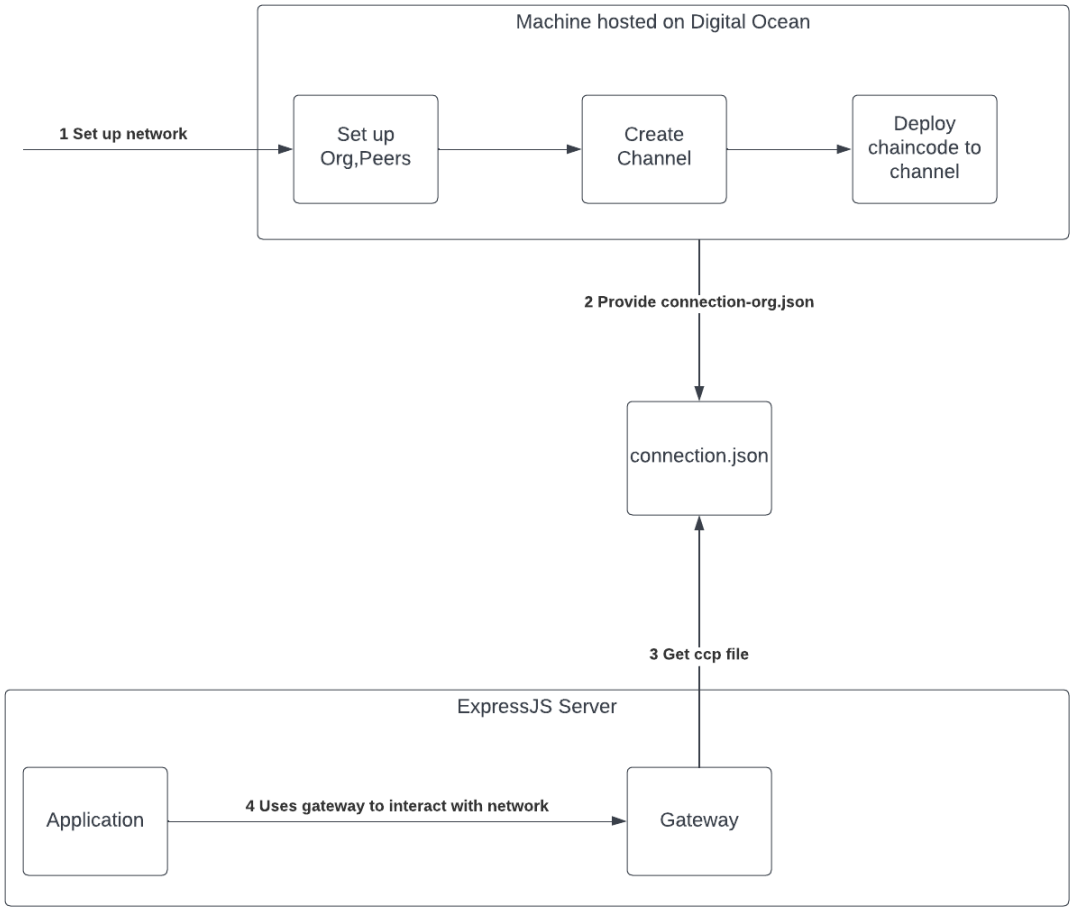


Figure 42: Network Connection Flow with hosted machine on Digital Ocean

The figure above shows the steps of how the Application using ExpressJS Server interact with the Hyperledger Fabric network hosted on Digital Ocean. The machine also use Docker Containers to deploy the network. Every Hyperledger Fabric network component, including peers, orderers, etc., is running on Docker containers as a result of project limitations, indicating the generation of.yaml files is required. Some network processes also use.yaml files as their

primary configuration file in addition to Docker. The sample configuration.yaml file will be imported into the server, as shown in the above graph, and then a new configuration.yaml file will be produced with the appropriate settings for that process. The computer on Digital Ocean will have a functioning network and communication channel when the configuration file has been properly produced. The connection-org.JSON files created by the network will be obtained by ExpressJS and sent to the cpp path. The Gateway will use this ccp path to create connection to the channel existing on the network then enable application to communicate with the blockchain system through smart contracts, which are also deployed on that channel.

Figure 43: How connection file looks like

With this technique, the server may simply change the network. From a security standpoint, this strategy has a big disadvantage. The program is subject to Remote Code Execution[15] vulnerabilities since the bash script eventually needs user input to run. This issue will be discussed more in the Security model Section.

2.2.3.1 Connection Model

ExpressJS, a NodeJS-based API server, is used by our system. To ensure that our approach for generating the connection is consistent with the library concept, Hyperledger Fabric pro-

vides its NodeJS library, known as Hyperledger Fabric SDK for NodeJS [14], for connecting the NodeJS server and Hyperledger Fabric network. We will describe the operation of the connection model and how to include it into our remote server throughout this part. We will also describe how applying this paradigm enables users to update each peer's ledger.

2.2.3.2 Smart Contract

Understanding the main objective that this model will accomplish is vital. The major objective of setting up the connection is to make it simple for the user, in this example the Patient user, to utilize their login credentials to communicate with the smart contract that is hosted on the Peers.

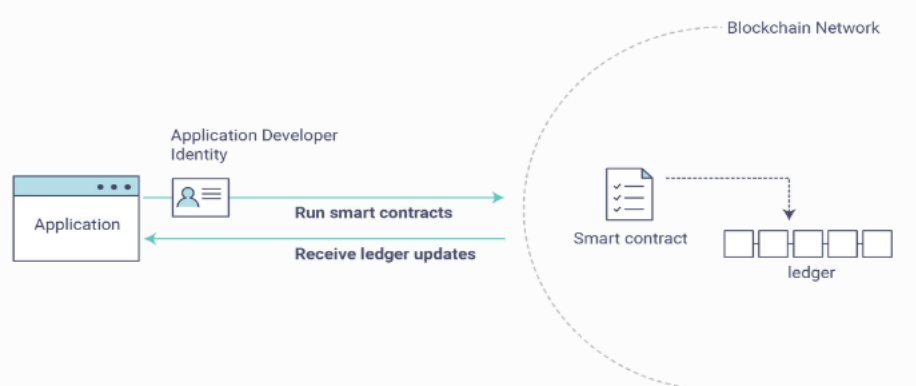


Figure 44: Smart Contract Execution [13]

The ledger, which is also locally saved on each peer inside the channel, may be accessed by the Organization admin or the Organization Server by executing the smart contract. Remember that the ledger is where all the information inside that channel will be kept, allowing the web server, who is responsible for our application's main objective, to communicate with the ledger and carry out the smart contract. The Contract interface, which serves two primary purposes and is provided by the Hyperledger Fabric SDK for NodeJS[12], is used to accomplish this procedure.

- Use submitTransaction to add transactions that store state to the ledger.
- Use evaluateTransaction to analyze transactions that look up state in the ledger.

By using the User Object of Patient and analyzing the method ReadCase on the Smart Contract, the code above shown how to establish a smart contract. The chaincode name and the contract name are the two parameters that the function will receive.

Figure 45: Getting Contract from channel codes

2.2.3.3 Wallet

One of the concepts featured in the Hyperledger Fabric SDK for NodeJS is Wallets [10], which is designed to provide a location for storing a single user's identity within the Hyperledger Fabric network. The user's certificate and private key, both of which were generated by the organization's certificate authority, make up their identity. The library offers three different types of wallets, however our web server only utilizes FileSystemWallet, which stores the identity within a configurable folder. Each channel has its own wallet since our project arranges the wallet directory according to the name of the channel.

2.2.3.4 Gateway

The class Gateway [13] of the Hyperledger Fabric SDK for NodeJS provides the Gateway, which serves as the intermediary between the web server and the Hyperledger Network.
The server will submit a file called Common Connection Profile, sometimes known as "ccp," to the Gateway in order to enable network connectivity. The information saved inside the "ccp" aids the Gateway in knowing where to interact since the Hyperledger Fabric network has more than one peer and the Gateway only has to connect to the matching peer of the user's organization. Each organization will have a separate "ccp" for security reasons in order to protect its connection information from other organizations.When the authentication phase is successful, the "ccp" of the user's organization will be loaded into the User Object of the associated user. The "ccp" will be produced in the JSON format and saved locally on the server of that organization. The hosted machine on Digital Ocean will take care of this 'ccp' by sending the connection-org.json to the file system where API Server is running .The Gateway also requires the user's certificate and private key, which are now kept inside the wallet because this is Hyperledger Fabric[13].

After "ccp" is loaded from the local server and the wallets are successfully generated, the code above offers us a closer look at the getWallet method within the User. The function

Figure 46: Usage of wallet

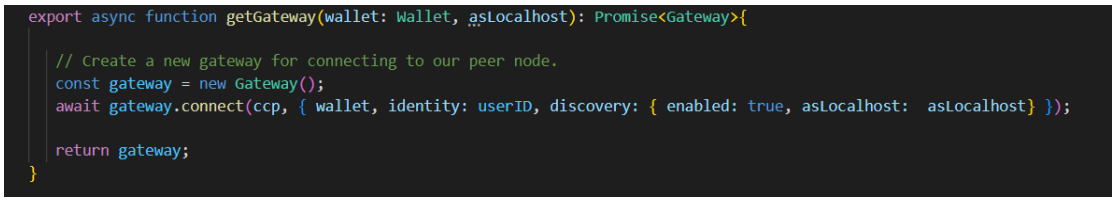


Figure 47: Usage of gateway

must additionally accept the user's session information in order to safeguard the network against users with incorrect session credentials as wallets save identities as files with the name username.id. The contract may be acquired via the appropriate channel when the Gateway has been connected.

2.2.4 Frontend architecture

While the backend is the backbone of the application, frontend plays a key role in interacting between system and client. Our frontend architecture consists of three layers: view layer, state management and API client layer. The view layer handle the React hooks and components. We utilise Redux to manage state in React. Finally, fetching is the technique for interacting between system and client.

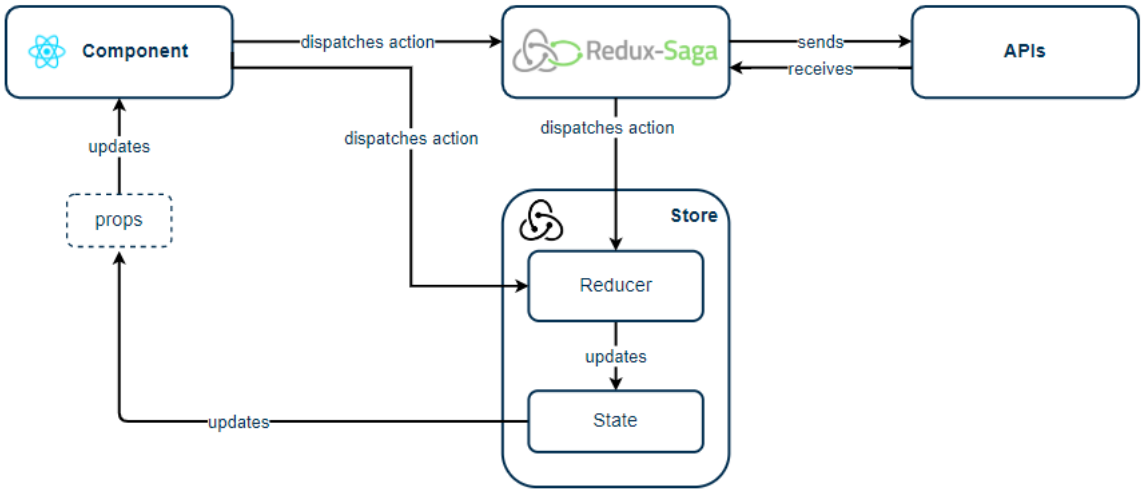


Figure 48: Frontend Architecture [17]

2.2.5 Security model

The Hyperledger Fabric and security implementation are major focuses of blockchain technology. A permissioned blockchain called the Hyperledger Fabric provides several security measures including authentication and consensus methods. The Hyperledger Fabric is the foundation upon which the SONA system is constructed, and as such, it automatically inherits all of its security implementation. Additionally, our project is focused on the user experiences by limiting the customization of the Hyperledger Fabric security model into a more structured security model that is appropriate for our scenario as well as by maintaining a more user-friendly interaction with the security architecture from the end-user perspective.

2.2.5.1 Network identity

The certificate and private information issued by the Certificate Authorities of the user's organization serve as proof of the user's identification. This is important since only a legitimate identity is permitted when joining the organization's peers to the network. The identification is retained inside the Wallets for the Sona project's web server [10]. By channel, the wallets will be sorted. This implies that each channel will have wallets in order to protect all identities from other companies and reduce the likelihood that they will be impacted by Horizontal Privileges Escalation.Additionally, the wallet will keep the user's identities when they successfully verify themselves by tagging them with the user session, and the identities will be deleted after the user logs out. Because the session is unexpected, any user with an invalid session cannot obtain the identity, and they also prohibit the user from interfering with the session.Additionally, the wallet will keep the user's identities when they successfully verify themselves by tagging them with the user session, and the identities will be deleted after the user logs out. Because the session is unexpected, any user with an invalid session cannot obtain the identity, and they also prohibit the user from interfering with the session.

2.2.5.2 Network policy

The policy is a key component in any permissioned blockchain network to establish how the permission functions, including the rights or permission of each business. When discussing organization permissions and channel permissions in Hyperledger Fabric, there are several policies available. However, for the sake of this project, we will just focus on Organization Policy and Channel Policy. To ensure the privacy of personal health data, our system also uses private data gathering, which necessitates modifications to the Hyperledger Fabric default policy.

2.2.5.2.1 Organization policy

The authority of each user's function inside the organization will be determined by the organizational policy. We give the same policy for every organization because our project is still in its early stages, and in this case, Org1MSP is a representative of Clinic 1's network administrator.

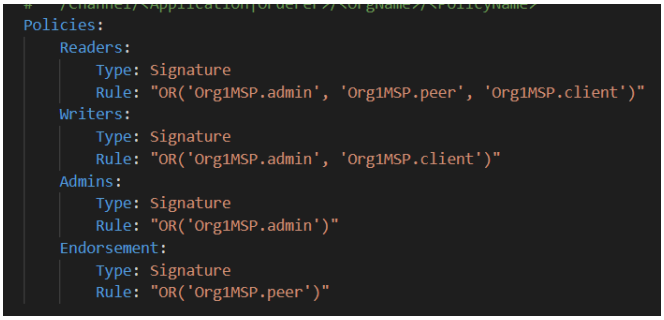


Figure 49: Example of a policy of an Organization

The code shown above exemplifies Clinic 1's policy as well as the policy of every other organization using our project. We have three policies—Readers, Writers, and Endorsement—all of which take the form of signatures. According to the organization's signature policy, a specified position must sign in order for an activity to be completed. For instance, on our network, at least one signature from a Network Admin or a Medical User is required in order to query data (retrieve a block) or update data (add a block to the ledger). When a chaincode is accepted and committed to the channel, the endorsement policy is established, and in our instance, it calls for a single signature from an Organization's Peer.

2.2.5.2.2 Channel Policy

The Channel Policy controls how many peers must approve a request to change a channel configuration within an organization.

```
################################################################################
Channel: &ChannelDefaults
    # Policies defines the set of policies at this level of the config tree
    # For Channel policies, their canonical path is
    #   /Channel/<PolicyName>
    Policies:
        # Who may invoke the 'Deliver' API
        Readers:
            Type: ImplicitMeta
            Rule: "ANY Readers"
        # Who may invoke the 'Broadcast' API
        Writers:
            Type: ImplicitMeta
            Rule: "ANY Writers"
        # By default, who may modify elements at this config level
        Admins:
            Type: ImplicitMeta
            Rule: "MAJORITY Admins"
```

Figure 50: Example of a policy of a channel

The policy that is enforced across all the channels utilizing our project is shown in the code above. As it has been illustrated, all policy types are set to ImplicitMeta, a form of policy that may only be used in connection with channel configuration and that refers to Organization Policy. For instance, the Organization Policy together with the Readers and Writers policy's "ANY Readers" and "ANY Writers" clauses permit every admin or customer to read and write data from the channel.Nevertheless, the endorsement policy rule that states "MAJORITY Endorsement" in conjunction with the requirement of at least one peer signing at the organizational level results in the statement that a channel modification can only be accepted when there is a majority of peer signings across all organizations. One advantage of setting a MetaImplicit policy to MAJORITY is that we do not need to change the channel policy when a network adds or removes an organization.

2.2.5.2.3 Private Data Collection Definition

A limited subset of organizations on a channel can approve, commit, or query private data using Fabric's private data collections feature without having to open a new channel. In this project, we want to protect the Patient Personal Health Data, we must provide a *Private Data Collection Definition* to Fabric. A collection definition contains one or more collections, each of which has a policy definition that lists the organizations that are included in the collection and specifies the properties that are used to limit the release of personal information at the time of endorsement and, optionally, whether the information will be deleted.[11]

The organizations from the Sona Clinic 1 are used in this definition. Only Org1 (Clinic 1)

Figure 51: Private Data Collection Definition in JSON file

is permitted access to the private data according to the definition's policy for PatientIdentifiableData and UsageRecordData. This setup is typical when the ordering service nodes must not have access to the chaincode data. However, the policy in the specification of the PateintIdentifiableData limits access to a certain group of channel organizations (in this case Org1). Furthermore, even though the chaincode level endorsement policy may call for Org1 endorsement, writing to this collection needs the endorsement of an Org1 peer. Additionally, only clients from Org1 may run chaincode that writes to the private data collection since "memberOnlyWrite" is true. We may manage which businesses are permitted to write to particular private data sets in this way.
By the same understanding, only user in Org 1 (or Clinic 1) can read the UsageRecord, and it also requires the endorsement of both Clinic and Research organizations for running the chaincode on UsageRecord data.

2.3 Application flow

2.3.1 Network Admin

The network admin would be an individual or a group that represents for the organization and authorized by the senior people of that hospital. Moreover, actions of management of the network are mainly used by command line interface.

2.3.1.1 Creating channel

Use case description: The network admin creates channel so that every peers and organizations could participate.
Actor: The network admin.
Precondition: The action must be accepted by authorized person or organization.
Basic flow:
This describes steps that the network admin would follow while create a channel:

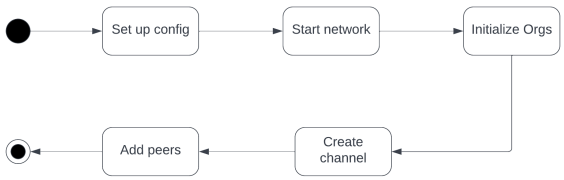


Figure 52: Activity Diagram: Admin Creating Channel

1. The network admin sets up the configuration for the channel.
2. The network admin starts the network.
3. The network admin initializes the organization and ordering service [1]
4. The network admin creates the channel.
5. The network admin joins peers to the channel.

Postcondition: The channel has been established, also, all peers that working with the network admin (the clinic organization including doctors and patients and the research facility including researchers) are joined the channel.

[1]Organizations that agreed to work with the network admin within in the system is implemented. The network admin also declares itself as an ordering service, as mentioned in [8], the network admin is able to define access control for channel.

2.3.1.2 Network Management

This section describes functionalities of network admin, including running the network, visioning activities within the network and ending the network.

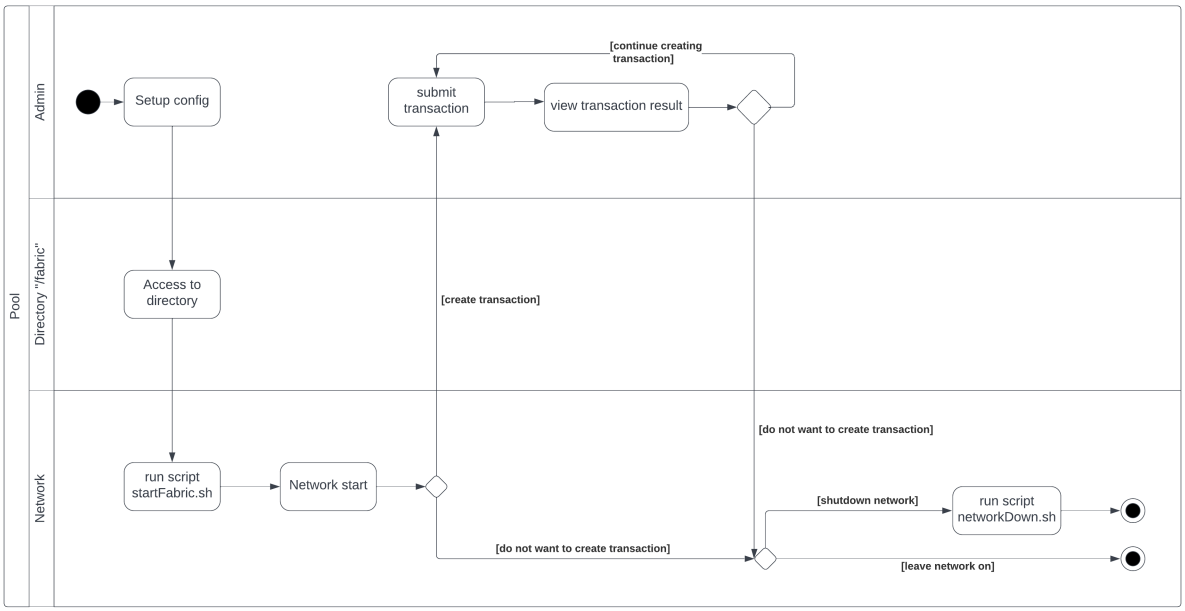


Figure 53: Activity Diagram: Admin Managing Network

2.3.1.2.1 Start the network

Use case description: The network admin operates the network, so that any organizations and peers within the network are able to interact to each other.
Actor: The network admin, HF network.
Precondition: Authorized by the senior people of the hospital.
Basic flow:
The following steps describe how the network admin start the network:

1. Access the right directory, makes sure that the admin is in the directory "/fabric".
2. Using the command line "./startFabric.sh".
3. The scripts in "startFabric.sh" will use the config, which has been set up by admin by previous steps, as initiated requirements then deploying the network

Postcondition: The network starts working.

2.3.1.2.2 View the actions within the network

Use case description: The network admin want to see the activities within the network.
Actor: The network admin, HF network.
Precondition: Authorized by the senior people of the hospital.
Basic flow:
The steps of how the network admin explores activities within the network is introduces as below:

1. Using an appropriate command line. [1]
2. The network displays all results related to the command line.

Postcondition: The admin is able to read an appropriate information related to the command line in use.

2.3.1.2.3 Close the network

Use case description: The network admin want to shut down the network.
Actor: The network admin, HF network.
Precondition: Authorized by the senior people of the hospital.
Basic flow:
The following steps show how the network admin closes the network:

1. Access the right directory, makes sure that the admin is in the directory "/fabric".
2. Using the command line "./networkDown.sh"
3. Network will be shutdown after scripts in the file are executed

Postcondition: The network is down, all the transaction can not perform.

2.3.2 Patient Side

2.3.2.1 Reading medical information

Use case description: Patient wants to read to his or her own medical information.
Actors: Patient, UI HF network.
Preconditions: The patient that has already been verified by the organization as well as logs into the application successfully.
Basic flow:
These below steps show how the patient reads his or her medical cases.

[1] For instance, the network is running on the localhost in port 8080, using the command line "curl "http://localhost:8080/patient/query/[patient name]" to see the information of the patient.

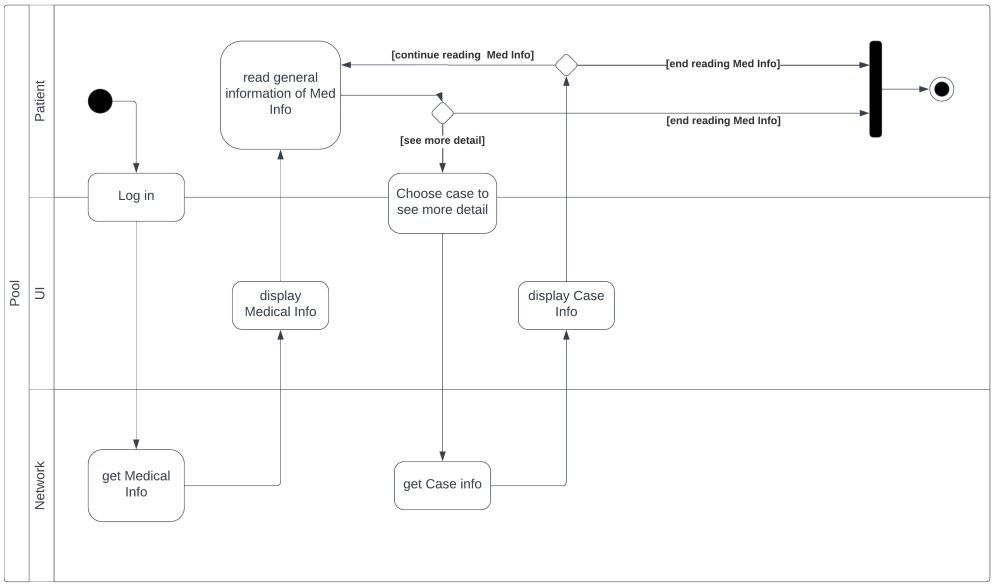


Figure 54: Activity Diagram: Patient Reading Medical Information

1. Patient successfully logs into the application.
2. Network will submit the transaction of getting Patient Data
3. UI displays Medical Information included Medical ID and a List of Medical Cases of that patient.
4. Patient chooses a case from UI.
5. Patient clicks the "See more" button to read the detail of the case.
6. UI displays Case details.

Postcondition: The patient reads details about the case which he or she chooses to see.

2.3.2.2 Reading usage records

Use case description: Patient wants to read to his or her own usage records.
Actors: Patient, UI and HF network.
Preconditions: The patient that has already been verified by the organization as well as logs into the application successfully.
Basic flow:
These below steps show how the patient reads his or her usage records.

1. Network submit the transaction of getting Medical Information with corresponding patient username.
2. UI displays a List of Usage Records included medical information id, case id, operator in charge, time created the usage records.
3. Patient reads the usage record.

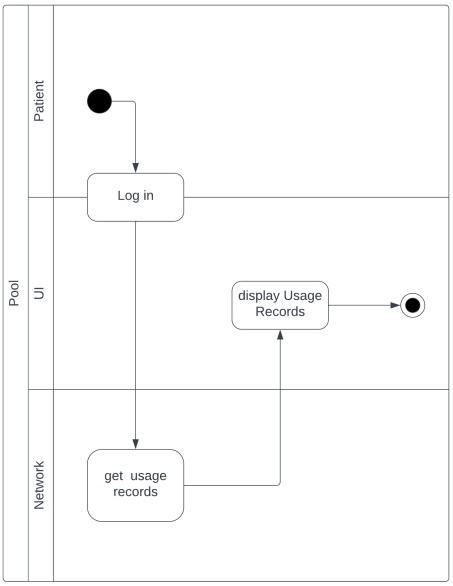


Figure 55: Activity Diagram: Patient Reading Usage Records

Postcondition: The patient reads details about his/her medical information's usage records

2.3.2.3 Authorizing operators

Use case description: Patient wants to authorize operators, such as doctors (or researchers if they request).
Actors: Patient, Operators, UI and HF network.
Precondition: The patient has an information about the operator such as his or her name or been asked for authorization.
Basic flow:
The following steps show how the patient authorizes an operator:

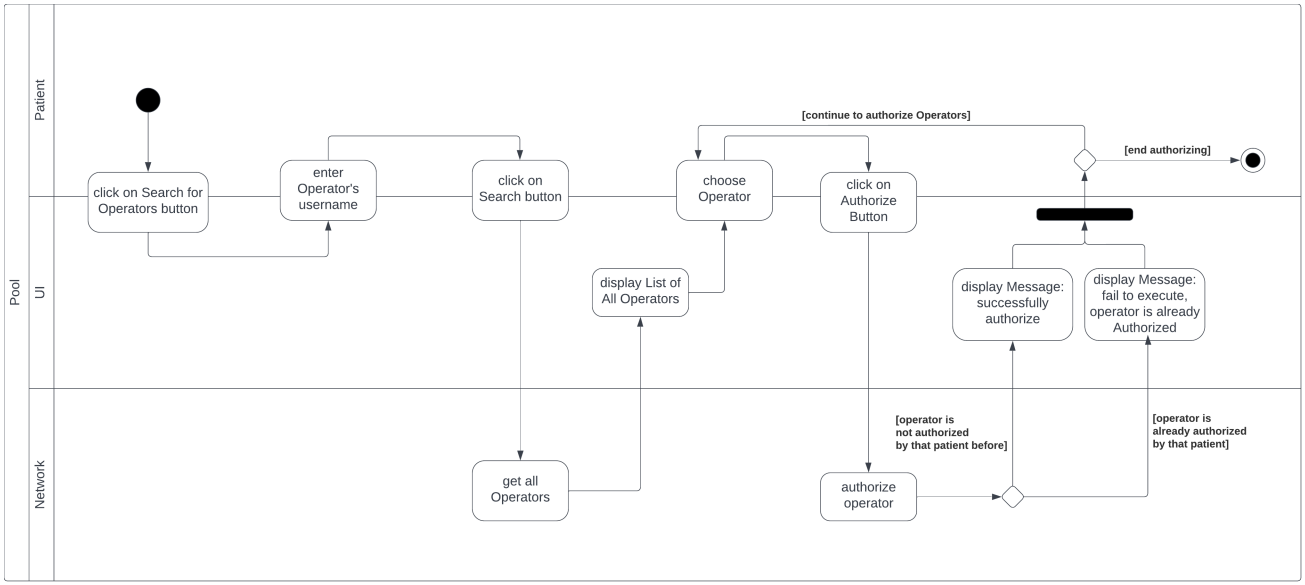


Figure 56: Activity Diagram: Patient Authorize Operators

1. Patient clicks on Search button located in the dashboard.
2. UI redirects to an Authorize Operator Page
3. Patient enters username of the operator that he/she wants to search for.
4. Network submit a get all operators transaction, with filter is the username that patient has entered
5. UI displays a list of existing operators in the system, which are results of searching
6. Patient chooses the operator he/she wants to authorize, then click the Authorize Button on the left side of that operator information
7. Network checks authorized status of that operator, if that operator is not authorized by that patient before, the application will return message of successfully authorize the operator, otherwise, application will display the error message.
8. UI displays the message from network

Postcondition: The operator is able to access to the patient who authorizes the doctor, the action of read or write the medical cases for the patient is allowed.

2.3.2.4 Revoking permission from previous authorized operators

Use case description: The patient wants to revoke the access (from the doctors) to the patient's data.
Actors: Patient, Doctor, UI and HF network.
Precondition: The doctor is previously been in the authorized list of the patient. Also, the patient is logs into the application successfully.
Basic flow:
The process of revoking an operator is described as below:

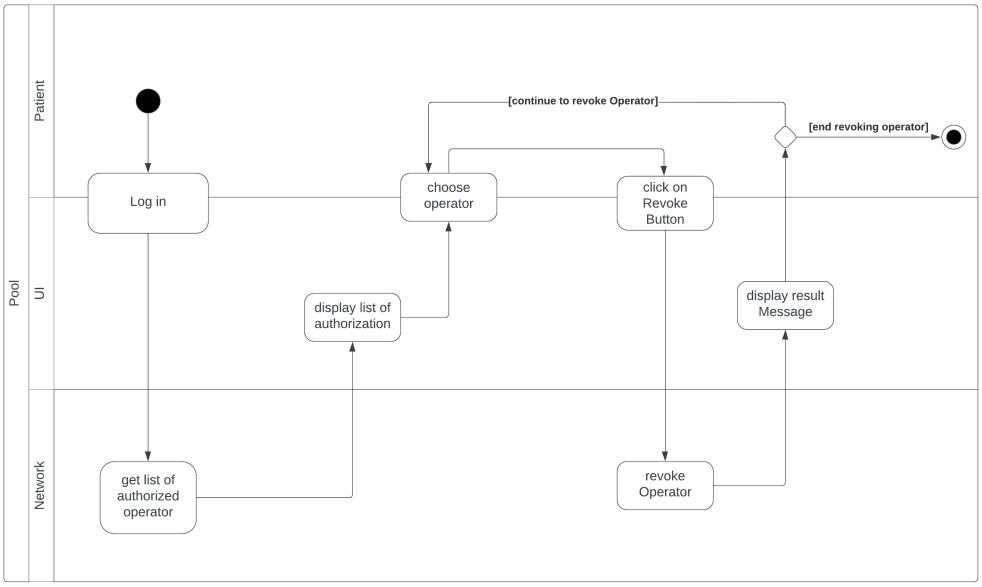


Figure 57: Activity Diagram: Patient Revoke Operators

1. UI displays the list of authorized operators of the patient, which is located in the dashboard page
2. Patient sees his or her list of authorized doctors.
3. Patient clicks on Revoke button.
4. Network invoke transaction of revoking that operator, and return a message.
5. Application displays message of success or failure from network.

Postcondition: The doctor can not access to information related to the patient who revoke the doctor.

2.3.3 Doctor Side

2.3.3.1 Reading the medical information of the patient.

Use case description: The doctor want to read the medical information of the patient that the doctor is currently authorized to work with (those information include the identity of the patient).
Actors: Doctor, UI and HF network.
Precondition: The patient has authorized to the doctor. Also, the doctor must be recognized within the organization.
Basic flow:
The process of reading the medical information of the patient is described as below:

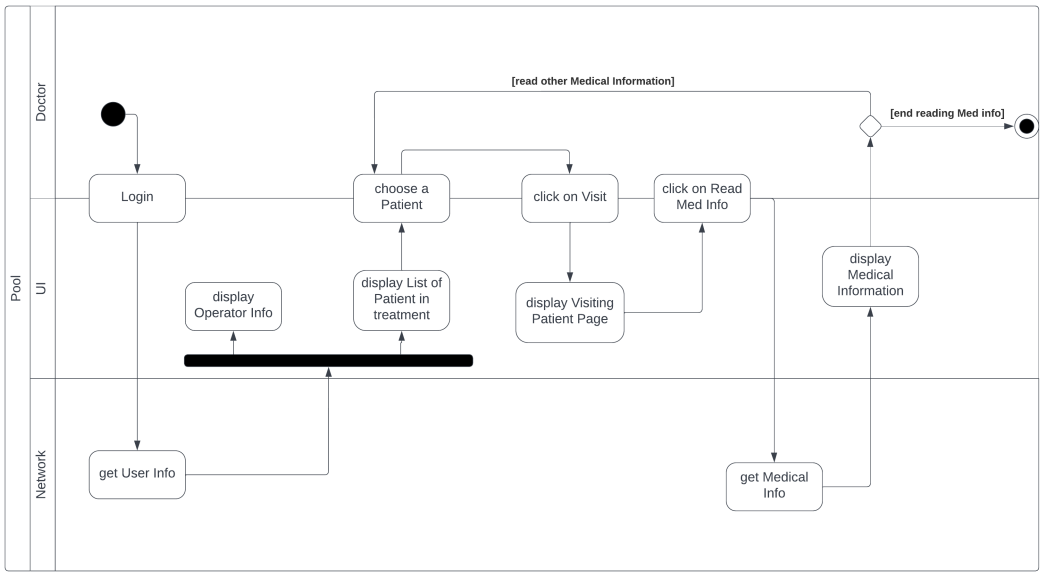


Figure 58: Activity Diagram: Doctor Read Medical Information

1. After successfully authenticated, network will return Doctor user information.
2. UI displays Doctor Page included a list of all patients that the doctor is currently working with.
3. Doctor choose the patient to visit.
4. Application redirect to the Visiting Patient Page
5. Doctor clicks on button 'See medical information'
6. Network return that medical information
7. Application redirects to the Medical Information Page with information received from network.

Postcondition: The doctor can read all cases details that the patient has.

2.3.3.2 Creating medical case for the patient

Use case description: The doctor wants to create the case for the patient.
Actors: Doctors,Patient, UI and HF network.
Precondition: The patient has authorized to the doctor. Also, the doctor must be recognized within the organization.
Basic flow:
The process of creating the medical case of the patient is described as below:

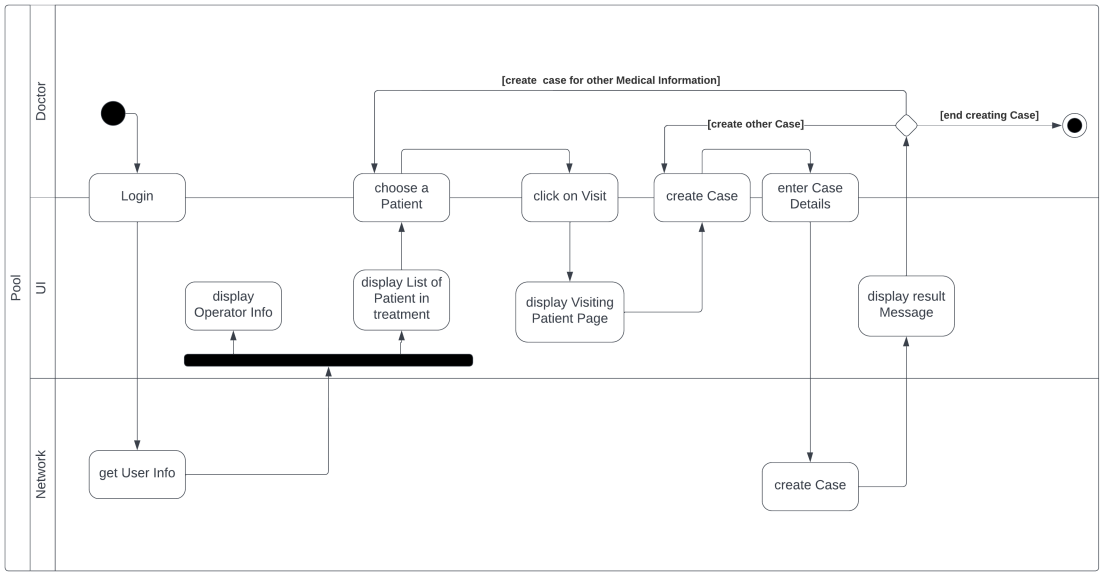


Figure 59: Activity Diagram: Doctor Create Medical Case

1. UI displays list of all patients that the doctor is working with
2. Doctor chooses patient that who wants to create a medical case
3. UI redirects to Visiting Patient Page.
4. Doctor clicks on "Create Case" button.
5. Doctor enters details for the case, then click 'Create'.
6. Network invoke transactions of creating case for that patient medical information, then return the message
7. UI displays message of success or error received from network.

Postcondition: The new case of the patient which reflects the new disease that the patient is currently get is created.

2.3.3.3 Appending medical case of the patient

Use case description: The doctor wants to update the existing case of the patient.
Actors: Doctors, UI and HF network.

Precondition: The patient has authorized to the doctor. Also, the doctor must be recognized within the organization.

Basic flow:

The process of appending the medical case of the patient is described as below:

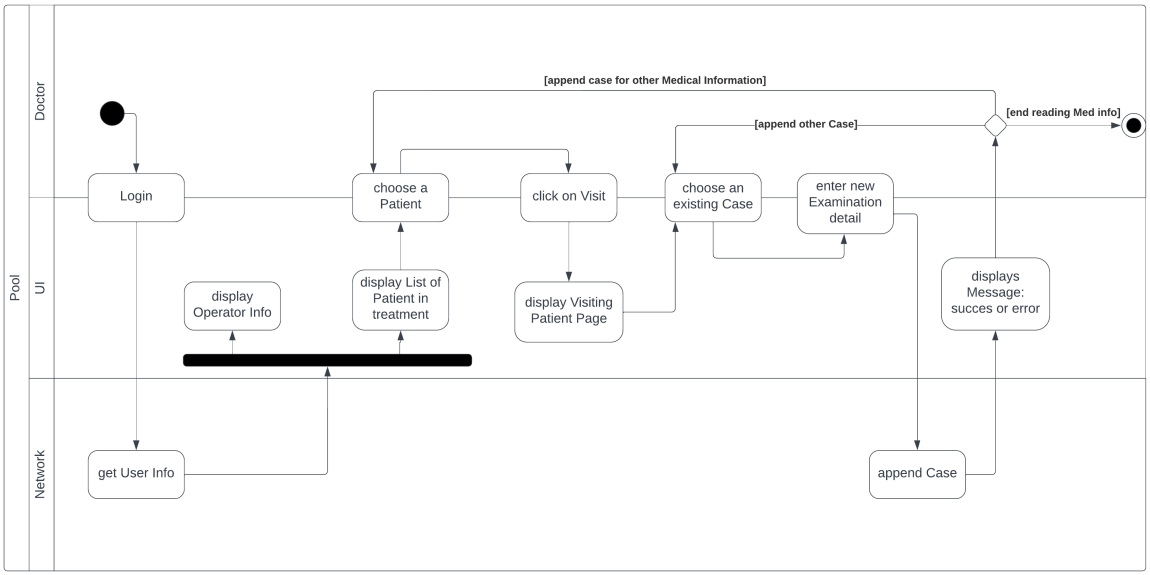


Figure 60: Activity Diagram: Doctor Append a Medical Case

1. UI displays list of patients that the doctor is working with
2. Doctor chooses patient that who wants to append an existing medical case
3. UI redirects to Visiting Patient Page.
4. Doctor chooses an existing Case.
5. Doctor clicks on "Append Case" button.
6. Doctor updates the case by appending the newest information about the patient including a case ID, a test results, diagnoses and treatments. Then doctor clicks 'Append'
7. Network invokes transaction of appendding a case for that patient medical information, then return a message of success or error
8. UI displays message received from Network

Postcondition: The newest information about the current status of the patient is added into the medical case of the patient.

Alternative: The case that the doctor wants to append is not exist. This problem is solved by the above use case.

2.3.4 Researcher Side

2.3.4.1 Accessing to medical cases

Use case description: Researchers want to access to medical case for doing research.
Actors: Researcher, UI and HF network.
Precondition: Researchers must be recognized within their organization (A research facility).
Basic flow:
The following steps describe how the researcher access to medical cases.

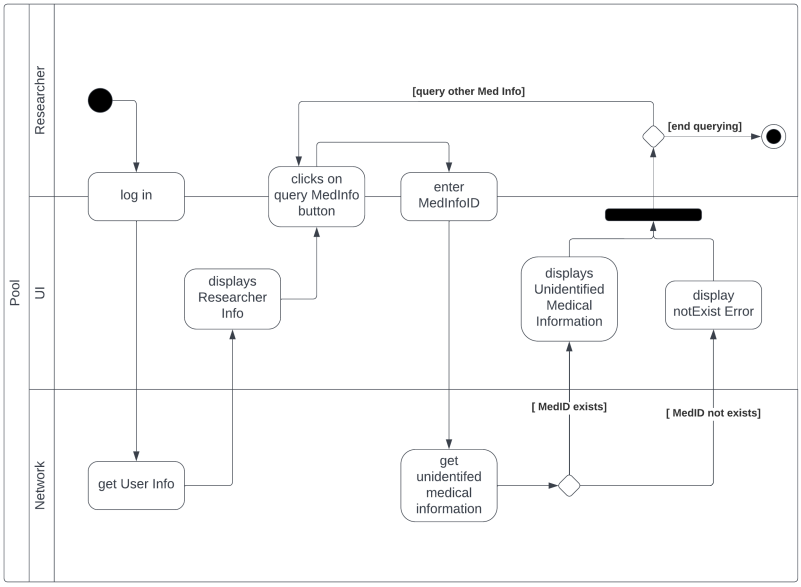


Figure 61: Activity Diagram: Researcher reads a Medical Case

1. Reseacher clicks on 'Query Unidentified Medical Information' button.
2. Researcher enters Medical Information ID.
3. Network invokes transaction of querying an unidentified Medical Information
4. UI displays the Unidentified Medical Information if success, otherwise it pops up an Error Message.

Postcondition: Researchers can read details of all cases. However, the information related to the identity of the patient is not showed.

Alternative: Researchers want to collect all medical cases related to some keywords.

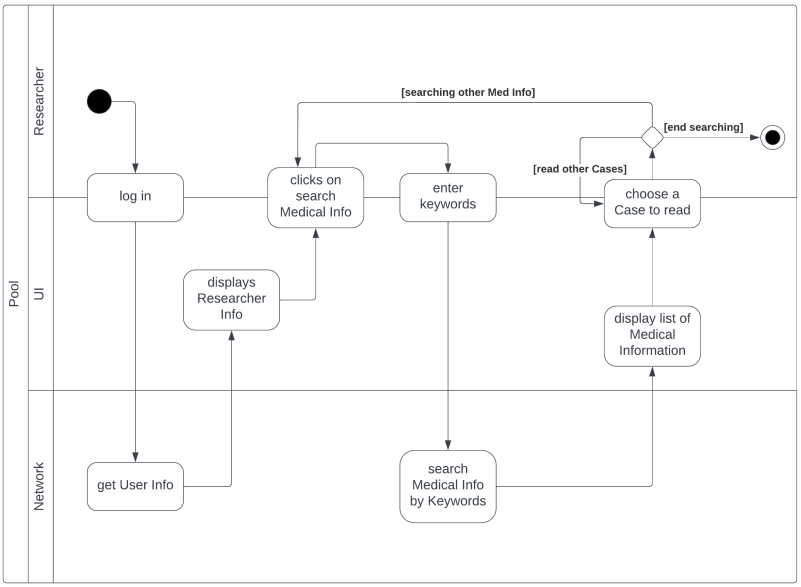


Figure 62: Activity Diagram: Researcher search a medical information by keywords

1. The researcher enter the keyword(s) to the search bar.
2. Network invokes the transaction of searching medical information by keywords
3. The UI displays all cases related to the keyword(s).
4. The researcher can access to those suggested Medical Information.

Postcondition: All the cases related to the keyword(s) of all the Medical Information are displayed to the researcher.

3 VALIDATION

3.1 Result

We have conducted our research in both market and technical side to come up with a solution for a Hyperledger Fabric network and a minimal user interface (UI) design. There are 3 interfaces in our application: patient, doctor and researcher. Patient have right to access personal information, keep track of medical records, and see who try to read or write to their medical profile. While a doctor can create or add some cases to patients who allows him to access. Besides that both doctor and researcher can only search for medical information to serve for their research purposes without personal information.

3.1.1 Scenario designs related to the application's implementation

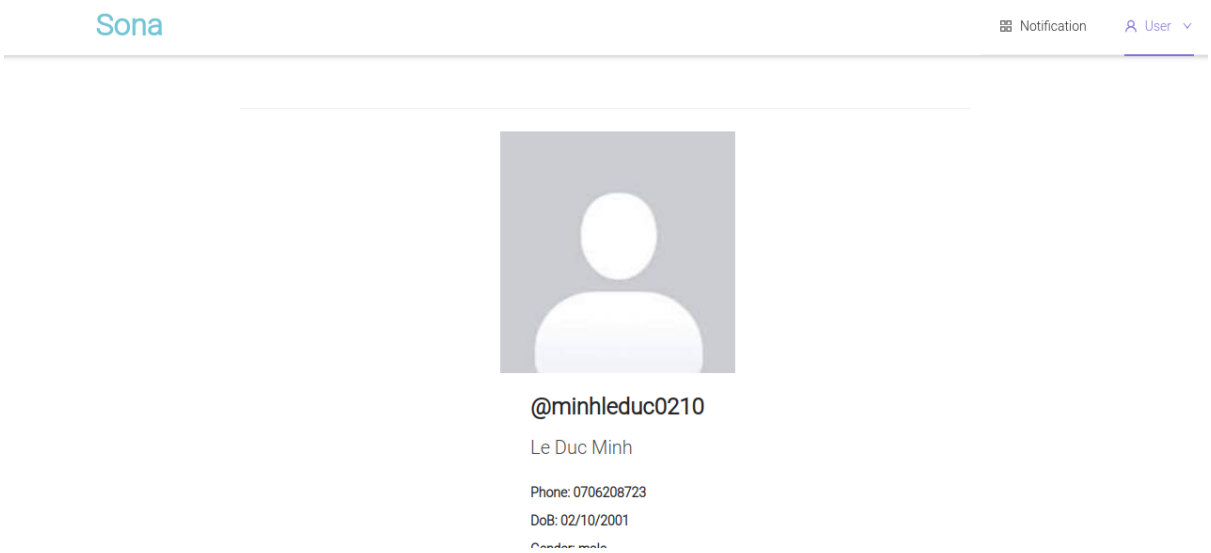
In our application, the function of register is prefer to be functioned by the organization. For instance, doctors' accounts will be created by the clinic as well as the patient' account, the researcher lab would in charge of registering account for researchers.

This section displays results of the application based on several scenarios depend on each entities inside the application. Every action of each entity and how it works are described in section 2.3

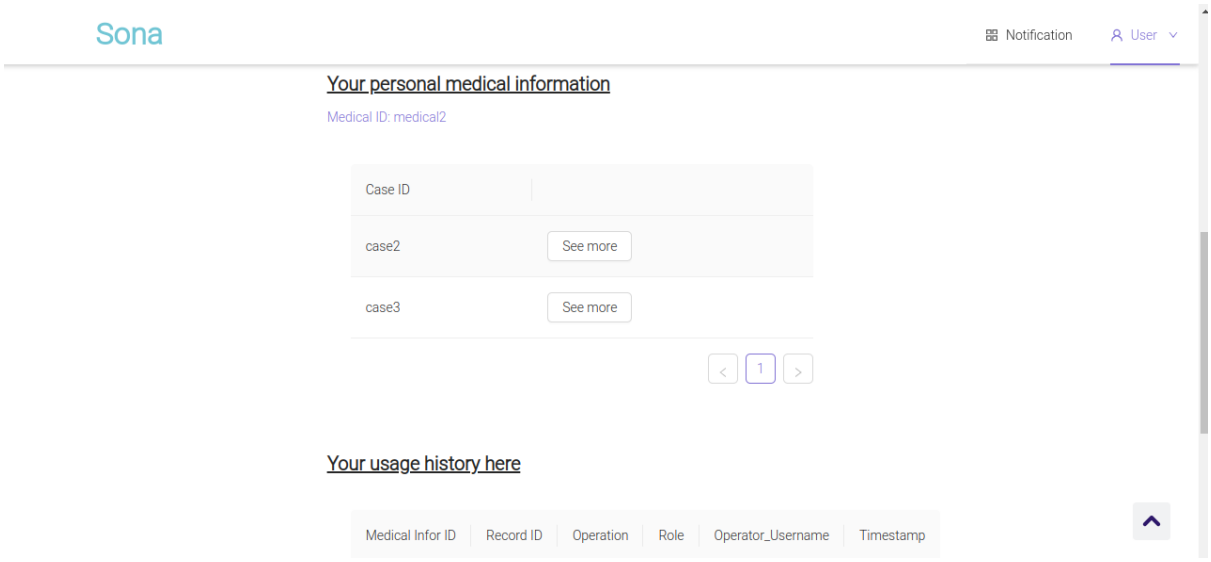
3.1.1.1 Basic scenarios

3.1.1.1.1 Patient side

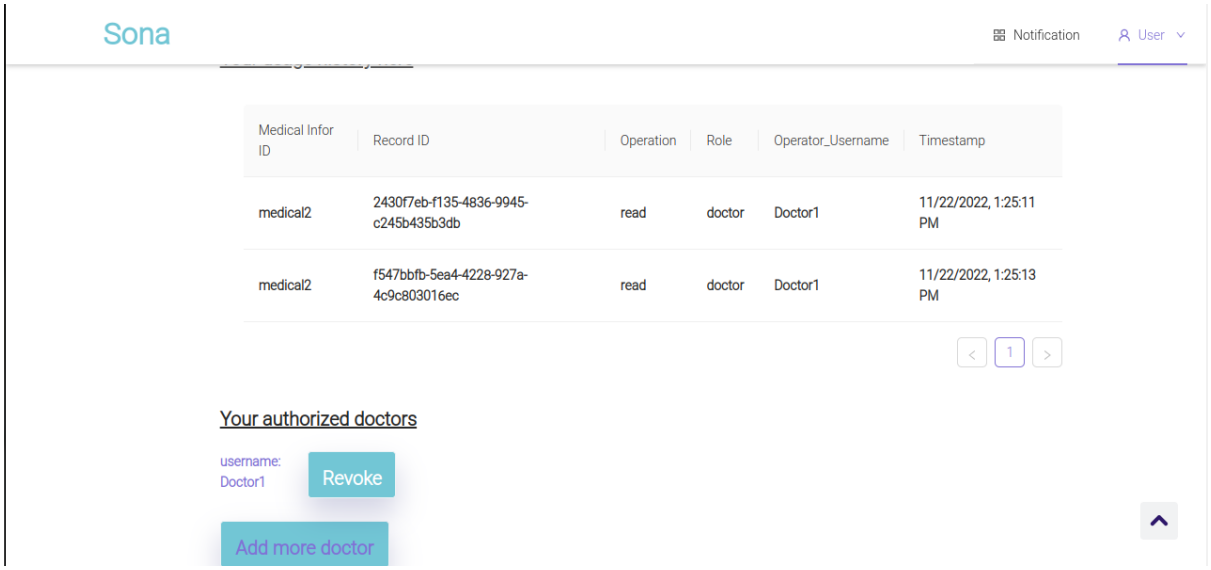
After logs into the application successfully, the patient would able to see his or her medical information, the patient could access to the case to see details, in Figure. 64. Furthermore, patient could also see an usage history that is to show which entity was active on the patient's medical information. In details, as illustrated in Figure. 63(c), while Doctor1 is authorized to treat the patient, the action of the Doctor1 on the medical information of the patient in a specific time on a particular record is recorded to the patient. Also, the patient could see a list of authorized doctors that are currently working with the patient, the function of authorizing and revoking doctors belongs to the patient.



(a) Fundamental information of the patient.



(b) A list of cases that the patient has gone through.



(c) The usage history and the list of authorized doctors.

Figure 63: The information of the patient.

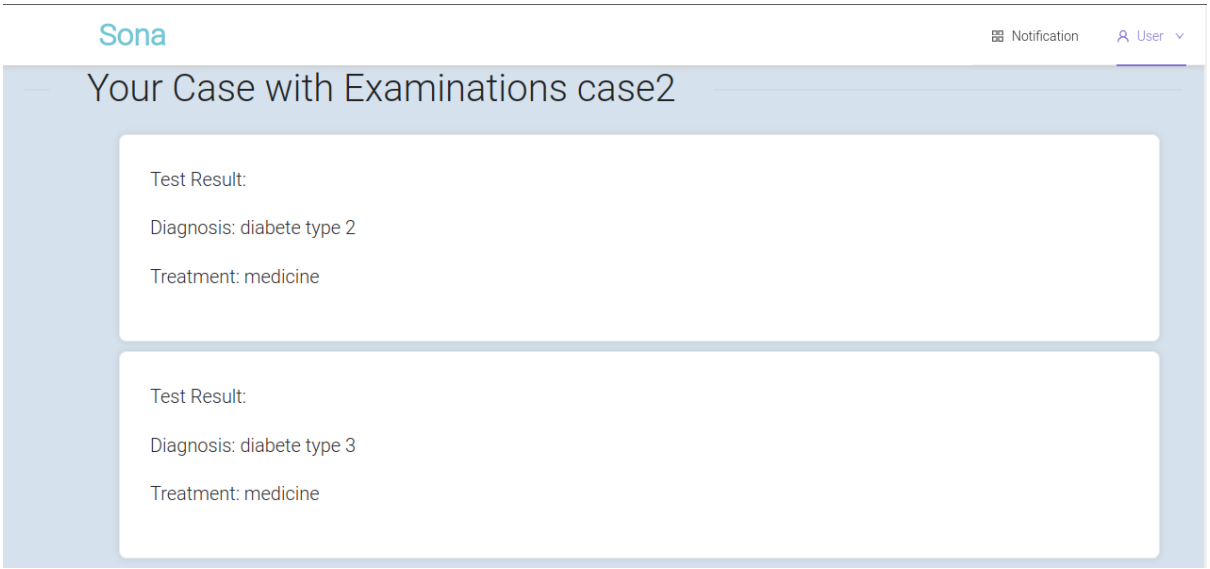
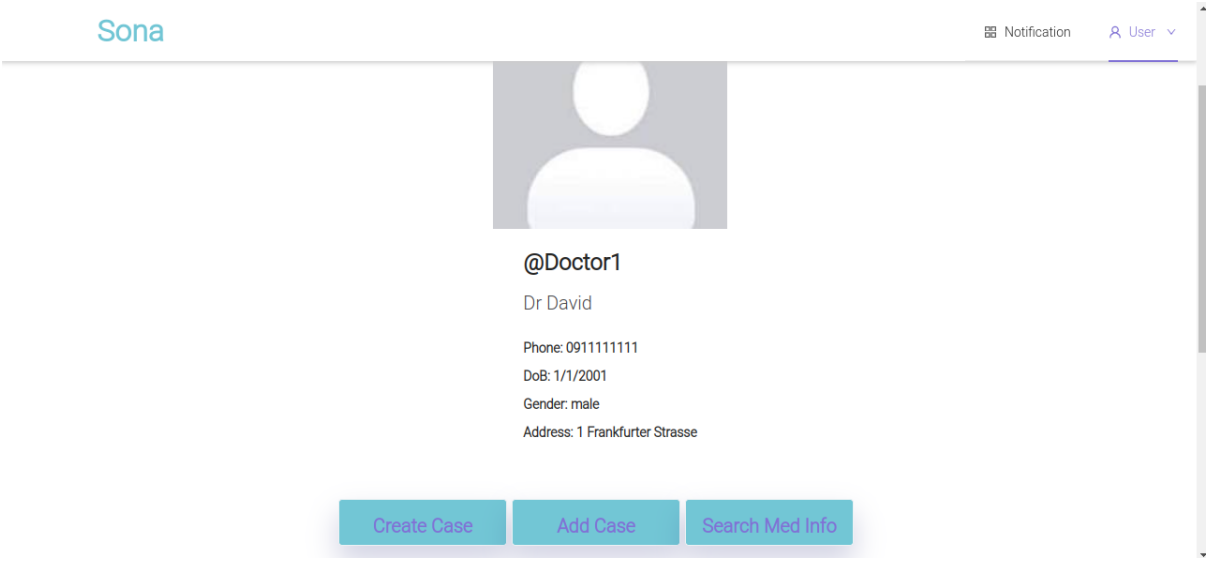


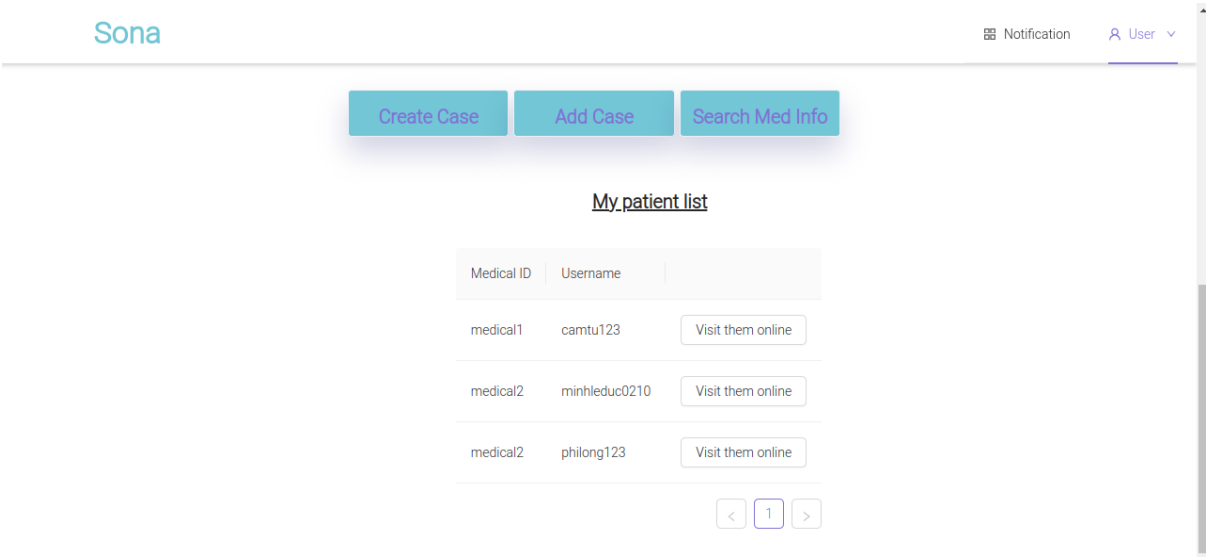
Figure 64: Details of the patient's case

3.1.1.1.2 Doctor side

The doctor, when enter successfully to the application, he or she could interact with the medical information of patients that the doctor is currently working with as in Figure. 65. The doctor is able to create, append and search for a case for a patient who authorize the doctor. The details of case while appending or creating have to be filled in (Figure. 66 and Figure. 67). In Figure. 68, the doctor needs to fill in the medical id to see the cases related to that medical id.

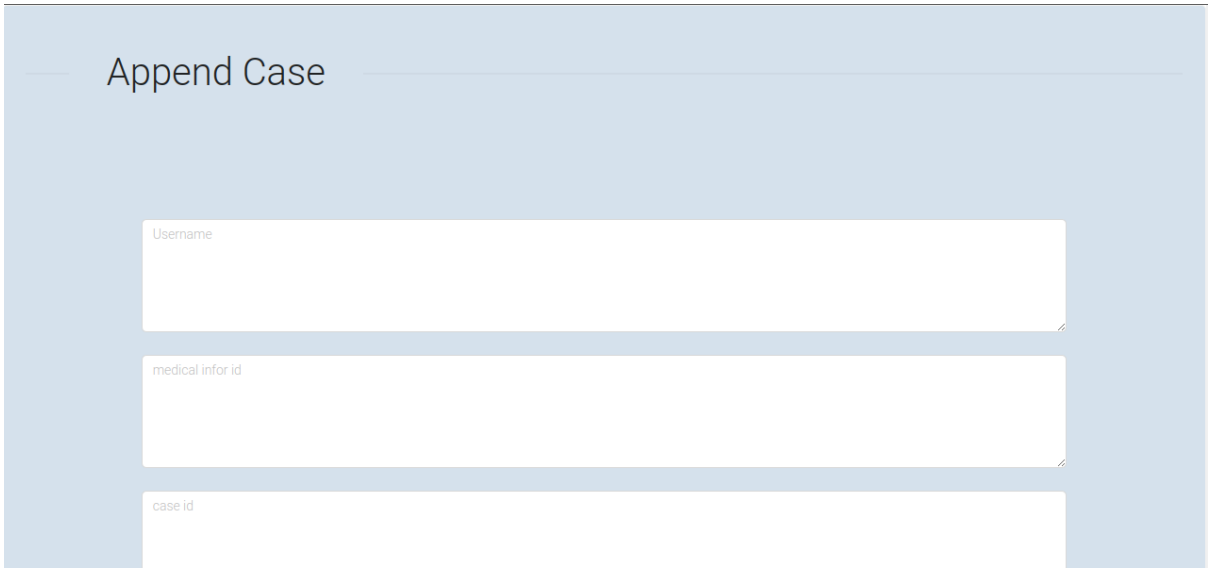


(a) Fundamental information of the doctor.

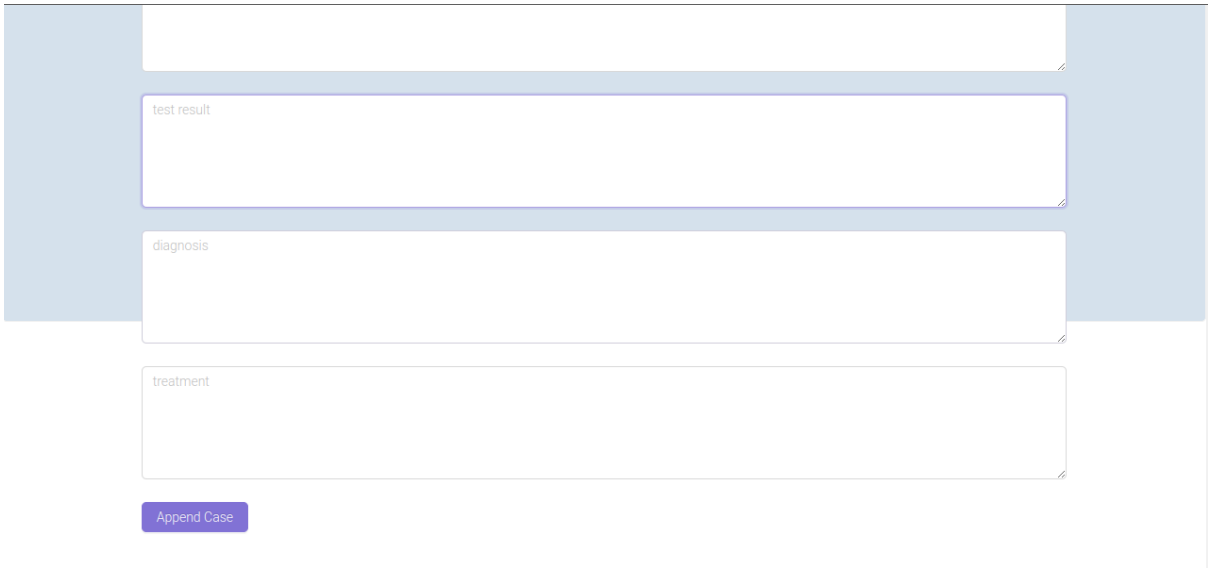


(b) The list of patients that the doctor is currently working with

Figure 65: The information of the doctor.

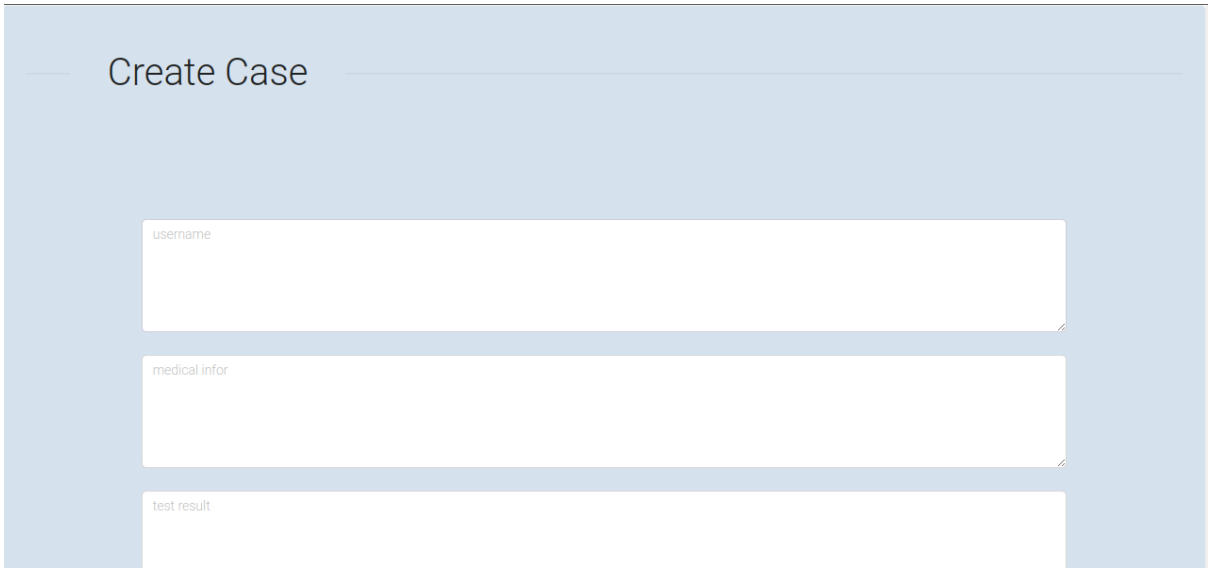


(a) Details of the append case form.

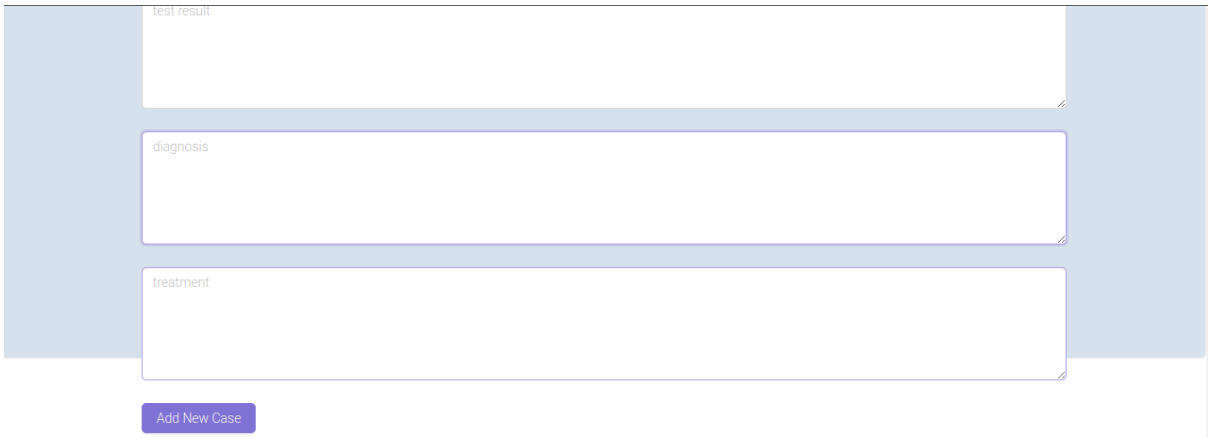


(b) Details of the append case form.

Figure 66: Append case function of the doctor

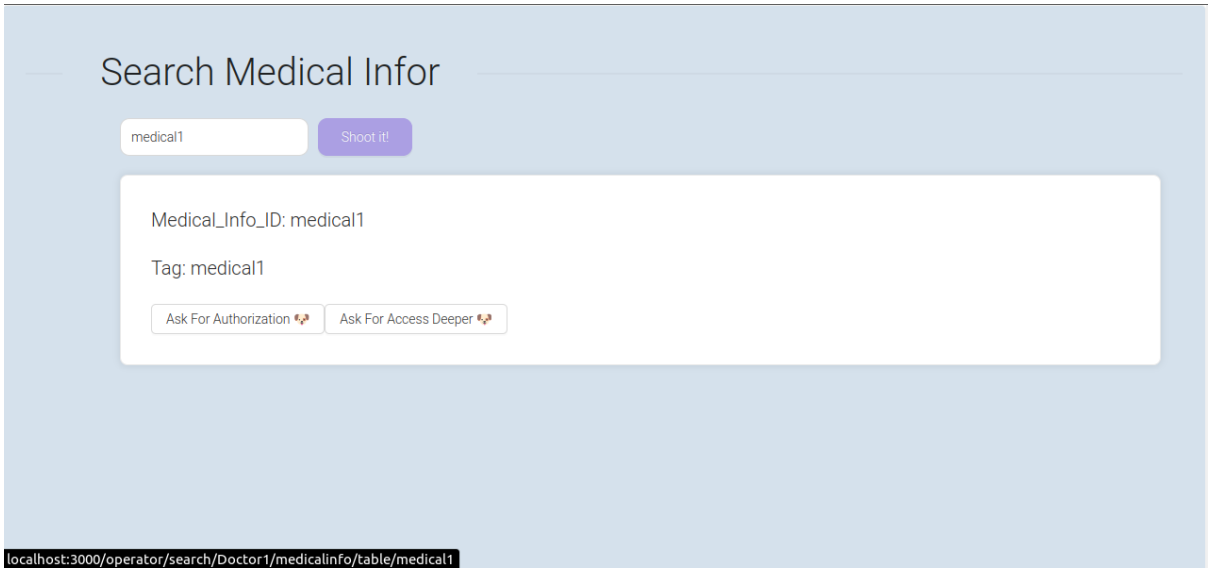


(a) Details of the create case form.

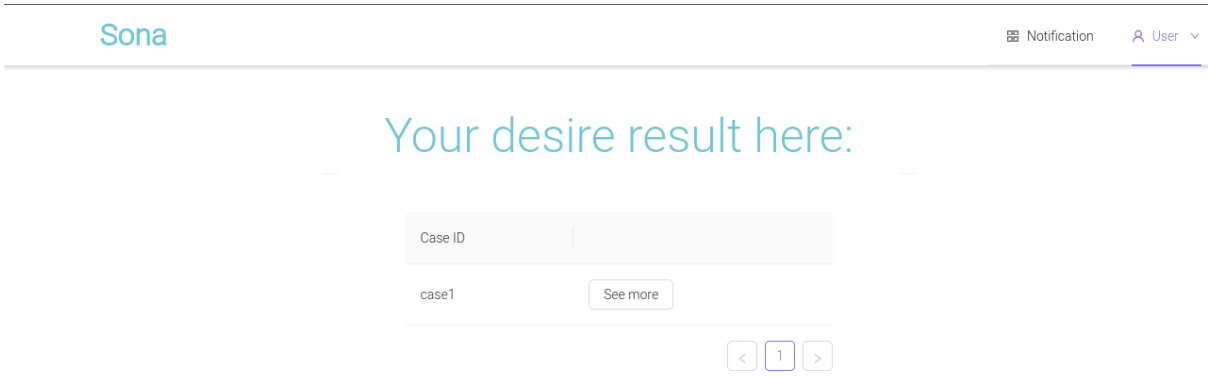


(b) Details of the create case form.

Figure 67: Create case function of the doctor



(a) Searching case by the medical id.

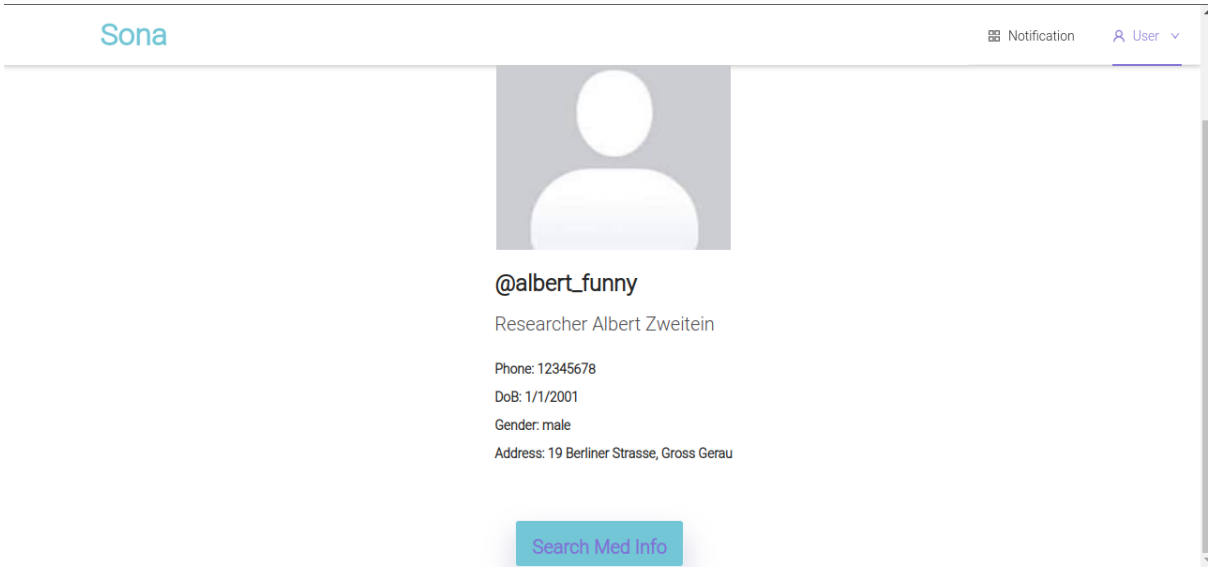


(b) A result return by the application which is a searched case

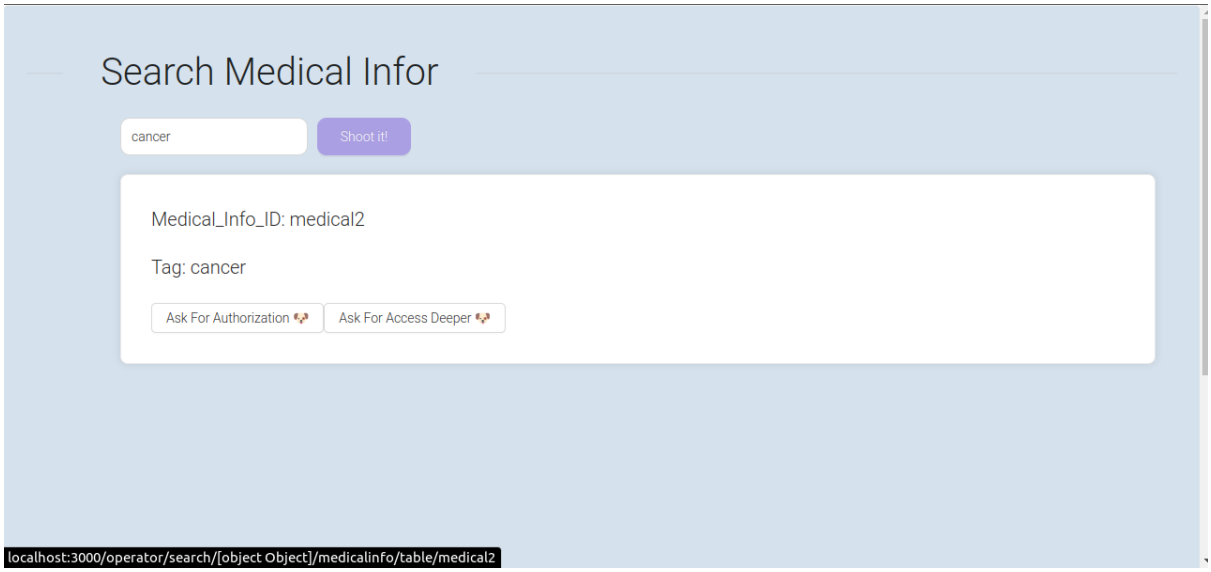
Figure 68: Search case function of the doctor

3.1.1.1.3 Researcher side

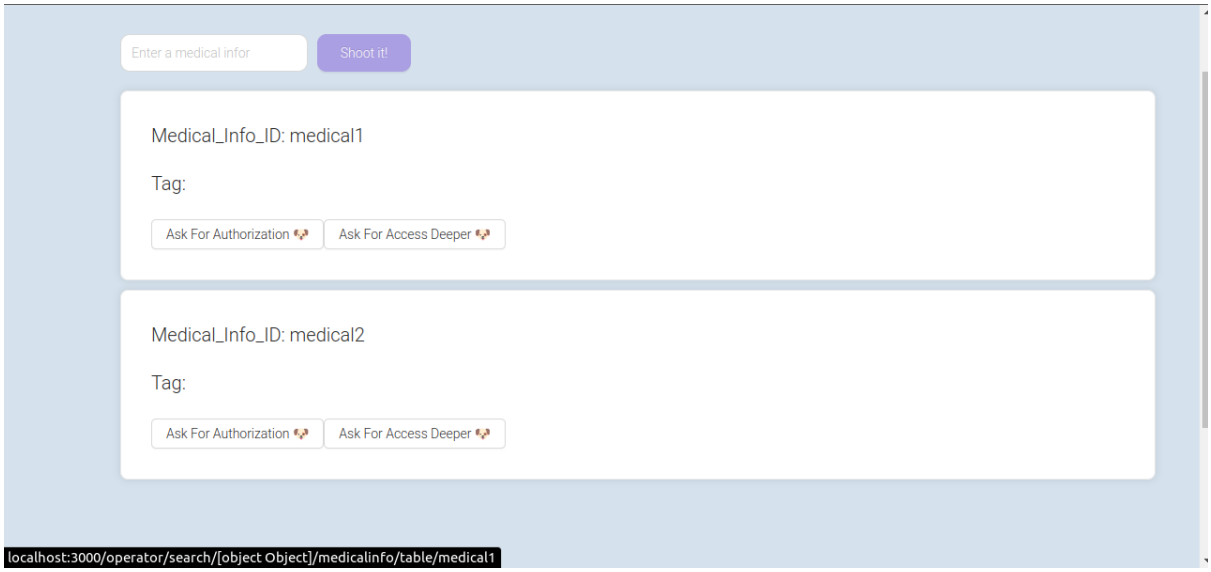
The researcher would be able to see all cases, furthermore, the researcher is able to see cases that related to some specific description as a way to scope their research. Clearly, from Figure. 69 (c), the researcher is not able to see the identity of the patient related to those cases.



(a) The information of the researcher.



(b) Search case related to keyword

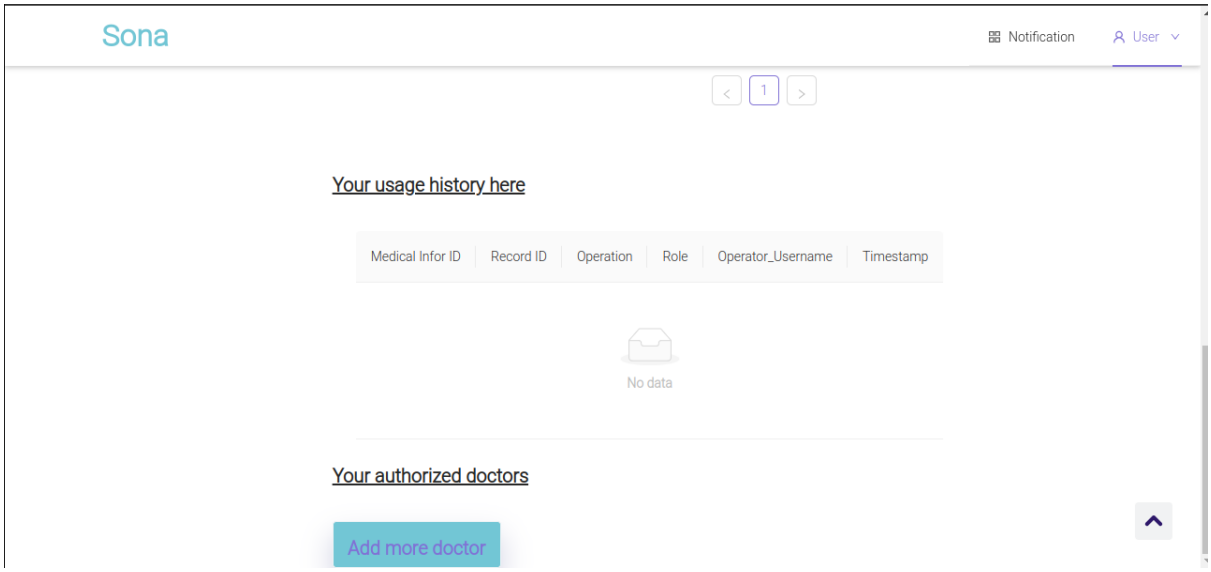


(c) Search all cases

Figure 69: The researcher side

3.1.1.2 Permissioned scenarios

This section describes how authorization would works in the application. Permissoned blockchain has a power of to identify an user to treat the user within a network, for instance, whether of not the user is allowed in that channel, or the channel could behave differently according to the identity of the user. In the scope of this application, when a patient no longer cooperates with the doctor or feels that the doctor's treatment is no longer suitable for them, they can choose not to cooperate with the doctor anymore. To do so, they will deny a doctor's access to their information, in other words, they will deny their authorization to that doctor.



(a) The patient revokes the doctor.

(b) The doctor is not longer able to access to the medical information of the patient

Figure 70: Revoke the patient's permission to the doctor

Also, the patient can choose another doctor and authorize that doctor (Figure. 71) as a way

to allow that doctor to perform treatment on the patient, also, that doctor could access to the medical information of the patient (Figure. 72).

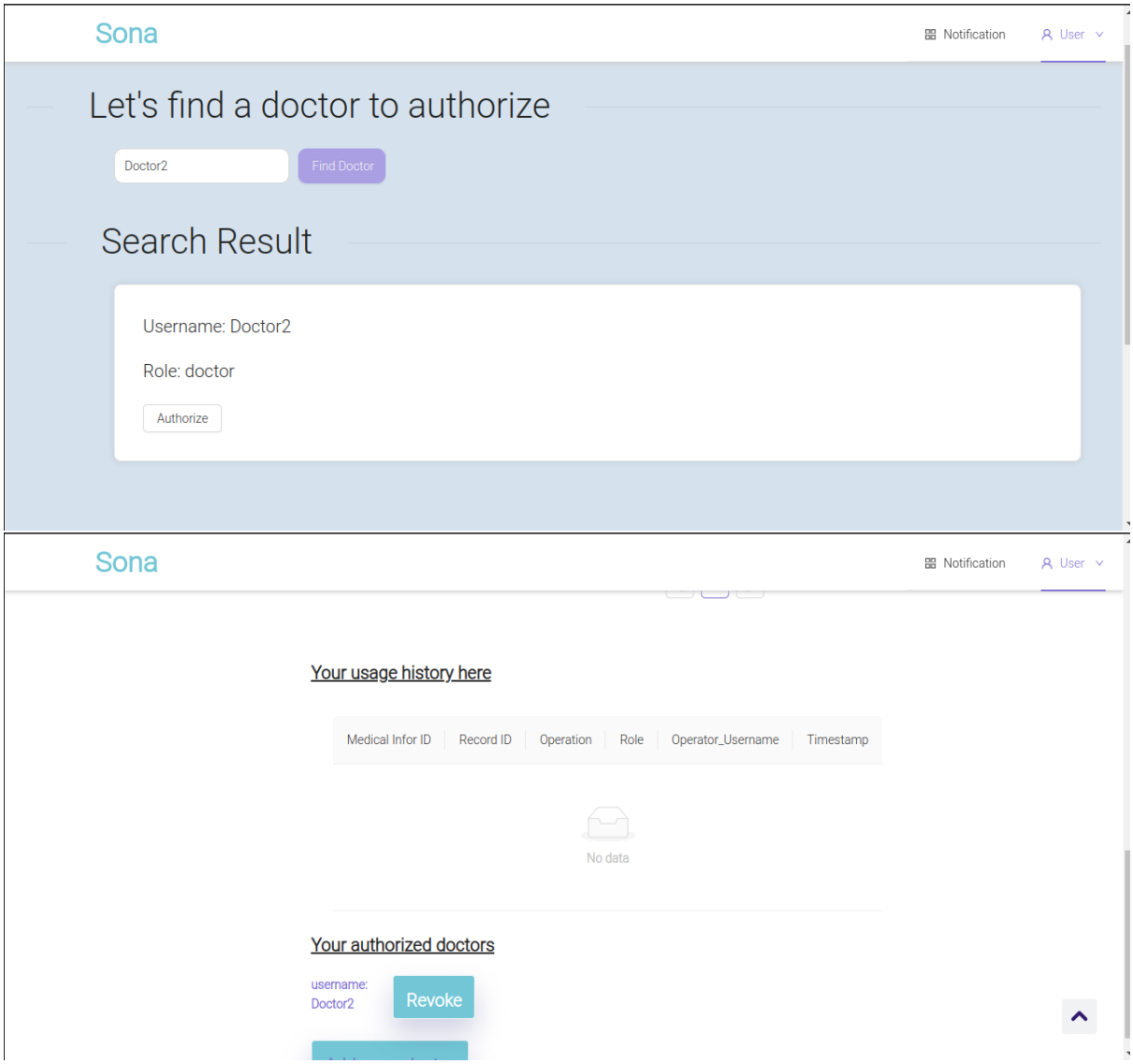


Figure 71: Authorizing another doctor.

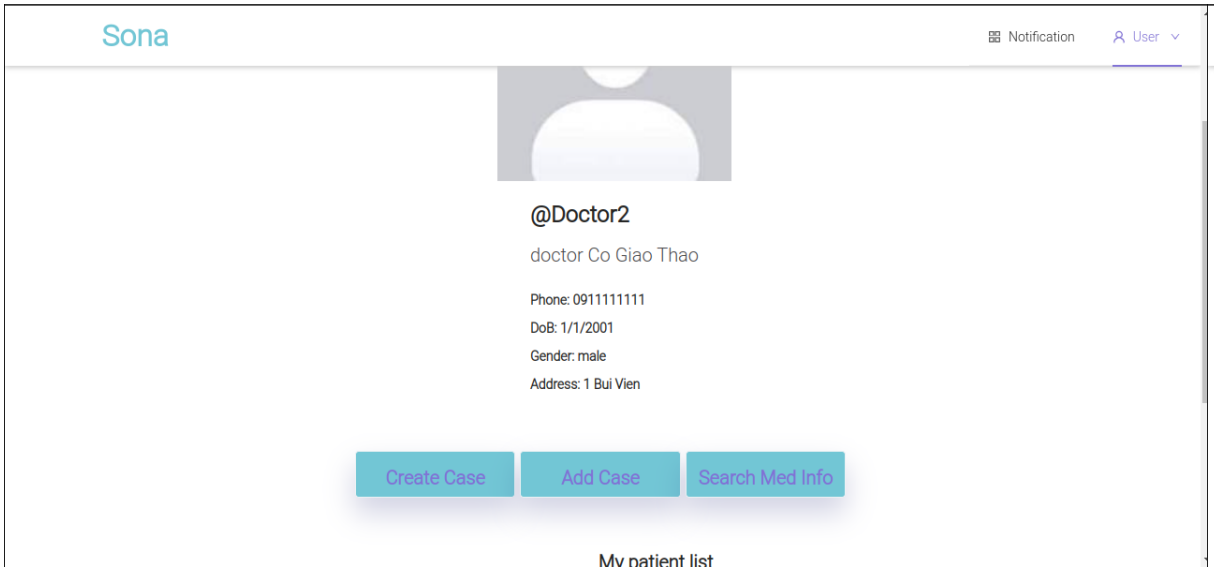


Figure 72: The doctor has just been authorized.

3.1.2 The system architecture and users

Our system is designed so that each role in the network has its own characteristics and functions. Different users joining the system with different roles will have different permissions and restrictions. These depict differences in the limitations and authority of the roles in the health system, as well as illustrate how the permissioned blockchain can be applied in the real healthcare system.

Each role has its own characteristics that make it different. The network admin is a person or a group of person authorized by the authorities in the hospital, however, its properties are not recorded inside the system, partly because the system will take care of recording the inside transactions and this admin' actions that do not directly affect transactions within the network. If there is a problem related to the application, such as the network is down, the admin can be traced. In addition to the network admin, the application also consist of other entities such as patients, operators including doctors and researchers, moreover, medical information, examination, usage record and case are also considered as entities in our application. Figure 6, 7, 8, 9, 10 and 11 in section 2.1.4.2 and 2.1.4.3 illustrates properties of each entities within the network. Clearly, features of patients and operators are identified separately, however, in the case of smart contracts, those entities have a little bit of coupling, for instance, medical information contains a list of cases and case could contain a list of examinations. The usage record hold almost all attributes of entities, for instance, properties related to the medical information (Case_ID, Medicalinformation_ID and Record_ID), operation (append, add, read), operators (Roles, OperatorName) and Time.

Figure 4 and 5 presented in section 2.1.4.1 describe different functions of participants within the blockchain network. As this is a blockchain, all attributes are private unless the correct permissions are given, the right of access to personal medical information should be performed

by the patient. For example, we see the use case of the operator, querying the operator's patient information first needs to verify the patient's authorization. Some problems may arise as to what happens in the event that the patient is not able to perform the authorizing task, there would be alternative policies set up on a case-by-case basis, for example, we would allow the family member or person who is bringing the patient or the hospital will be responsible for the patient to perform the task.

3.2 Validation

Our solution is not a brand new one in the industry. Although we launched and tested our prototype in the laboratory environment, the solution is proved robust and valuable in the industry. There are multiple blockchain healthcare platform like 'Patientory ',The HIPPA-compliant health information exchange (HIE) powered by blockchain secured the crucial data using improved cybersecurity protocols and EMR interoperability. MedicalChain, Farmatrust, Solve.Care are also projects from healthcare that use Hyperledger Fabric, which are potentially the leading company in the market at the moment.

3.2.1 Hyperledger Fabric Solution in Healthcare System

Our prototype utilises 5 points of the Hyperledger Fabric that many non-blockchain solution could not directly provide:

- **Interoperability**: The interoperability that blockchain technology provides is among its main benefits. To guarantee that the data can be accessible across hospitals, the present HIS employs many sorts of protocols and standards. For data sharing and storage, the majority use CDA, FHI, and HL7 2.X. However, this could make integrating new platforms or systems more difficult. Blockchain is the answer. Since blockchain functions as a decentralized database, it can resolve any interoperability problems. Thanks to the APIs, the data may be accessed with a focus on the common data format. The platforms and protocols that are now utilized to access and store data may also be smoothly integrated with blockchain.
- **Integrity**: The integrity of blockchain also provides solutions to several issues facing the present healthcare sector. The use of blockchain in healthcare today has a significant impact on how data is transferred across systems. This causes mistakes, which negate the value of the data that has been saved. Data integrity can be continuously protected at all levels thanks to blockchain technology. It also makes it possible to delete several instances of outdated patient data. Additionally, once the data is posted to the blockchain, malevolent actors cannot edit it, protecting the data's integrity. When consulting a doctor, only the patients have the power to alter the specifics. Numerous

case studies for blockchain in healthcare provide evidence that this technology can add the necessary integrity to prevent data theft or misuse.

- **Security**:The conventional healthcare sector is now plagued by data leaks, which have the potential to cost businesses millions of dollars. They significantly rely on trial and error because there is no suitable answer, and this is failing to safeguard all of their platforms, including data integrity. Data theft and tampering are serious issues that need to be addressed. Blockchain uses private keys for data encryption, and only the recipient has the key to decrypt the content.
- **Maintenance Cost**: The present healthcare systems also have a serious issue with maintenance costs. A specialist team is required to maintain the present system across various activities and to make sure that all the functions are coordinated and operating as intended. Blockchain is a distributed, decentralized network, hence it is immune to this issue.
- **Universal Access**: Everyone who uses blockchain has access to it. Because it is not reliant on a centralized authority, it enables global access. When necessary, authorized entities may quickly access data, and various techniques akin to smart contracts can automate the entire process. In conclusion, blockchain for medical applications is promising and ought to be promoted at work.

3.2.2 Typescript as a programming language for Smart Contracts?

Typescript is one of the most friendly and popular programming languages in the world. Using Typescript for our smart contract is based on the research in other blockchain and web-based platform. For example, NEAR platform, which is a potential platform in blockchain network and also cryptocurrency, fully support Typescript for writing smart contracts Another example, Etherium, which is a very famous blockchain application, also support Javascript (NodeJS) for smart contract development. Hence, instead of spending time on getting familiar with a brand-new language, our team decides to make use of the benefit of Typescript.

Table 1: Advantages and disadvantages of which languages are supported in Hyperledger Fabric.

	Advantages	Disadvantages
Golang	Golang is a programming language that able to develop fast and scalable blockchain systems Golang is supported much in Hyperledger Fabric.	Not having experience on this language.
Java	Strong in Objected-oriented programming	Java has no backup facilities Hyperledger Fabric does not have many updates for Java. Memory consuming.
Typescript	Typescript is deployable across multiple platforms Group members have experiences on Javascript before Typescript supports writing code in OOP form.	Typescript needs to be converted in to Javascript while running a program.

3.2.3 CouchDB for the Ledger

There are several options for choosing a suitable database for storing data. Depending on the orientation of the application, the appropriate database types will be selected. For instance, if the application aims to have fixed or predefined schema and suited for complex queries, structured query language would be a best choice (MySQL, PostgreSQL), besides, if one aims to have a dynamic schema, non structured query language is a best fit (MongoDB, GraphQL or Cassandra). However, Hyperledger Fabric serves two type of databases, LevelDB and CouchDB. After the network activates, it is impossible to update the database, thus weighing the advantages and disadvantages art essential. The table 2 illustrates features that considered as the benefit and drawback of two mentioned databases.

Since Sona is an application, in which, a large amount of transactions will be performed as well as storing a large amount of data, using CouchDB for storing and processing data is a better choice. The JSON format is easy to use, ensuring flexibility in data design, page formatting and also in data retrieval and indexing, all of which make data processing easier. Furthermore, CouchDB also contains all the functionality of the LevelDB database, such as accessing and setting data based on keys, which makes CouchDB not only equal but also outweigh LevelDB.

Table 2: Advantages and disadvantages of LevelDB and CouchDB.

	Advantages	Disadvantages
LevelDB	Simpler to employ. _In comparison to CouchDB, a simple, quick database with minimal overhead. _Because it is quick and simple to use, it works well with straightforward Hyperledger Fabric projects that involve few transactions. _LevelDB allows for batch modifications and both forward and backward iterations over the data.	No indexing or query support. _Simple actions like Put(key, value), GET(key, value), and Delete(key) are the only ones it supports. _As complicated searches and indices are not supported, querying huge data from a LevelDB database is difficult and inefficient. _It is not a SQL database, hence it lacks all the well-known components of a relational data model. Only key-value pairs can be used to store data.
CouchDB	_CouchDB lets you store data in the database as arrays or dictionaries as it is a document-oriented database. _JSON queries and indices are permitted. _Using indices to more effectively and flexibly query massive datasets of chaincode. _An HTTP URI is used to interact with CouchDB data, allows us to access data using HTTP (GET, DELETE, PUT, and POST) actions.	_The Hyperledger Fabric network coexists with CouchDB as a separate database. _More effort is required to run the network in terms of database maintenance, setup, configurations, etc.

4 DISCUSSION

Hyperledger Fabric is one of the state-of-the-art platforms to develop core function of our application, we concentrate on making high and robust mechanisms based on the network

4.1 Decentralized mechanism

Hyperledger Fabric has a decentralized mechanisms with a distributed ledger, which is replicated across many participants in the network. Database must be store in all nodes instead of one single node, which can reduce single point failure and optimize resource distribution. In our application, we have several nodes which all stored the database of patients personal information, medical information and their history of their medical usage. Moreover, with the help of chaincode, the process of self-validating data become controllable, for example, patients can authorize for doctor to see their records and personal information but for unauthorized researchers, the access for records is limited without personal information.

4.2 Consensus mechanism

Hyperledger Fabric supports for consensus by ordering service or ordering nodes. We use "majority endorsement" in the endorsement policy at the channel level. Only when the network reaches more than 50% consensus, can a change be executed. A big advantage of using this mechanism is that the performance increases in comparison with some network like Etherium, which is necessary for 100% of agreement between nodes.

4.2.1 Security implementation

Security is one of our most concern in Sona because of the necessary integrity of medical data, section *2.2.5 Security model describes* how security is implemented in Sona in a more detailed way. Hyperledger Fabric also provides various security methods which helps data stored in Sona is safer from some data thief. In our local storage, we store user login information such as username and password. We use token-based login system to ensure the security of the whole application. In details, once user logins, an encrypted refresh token and access token are created. Access token is expired in about two hours while refresh token is about one or two months. When the access token is expired, it is necessary to use refresh token to make a new access token. We use both tokens in the form of JSON Web Token (JWT) with SHA256

algorithms encrypted and store them in the HttpOnly Cookie. Besides that, we put user's username in the JWT payload. With these security implementation, we intend to help patient can access to the websites smoothly, quickly and securely.

4.3 Limitations

Due to the scope of time, we manage to make the simplicity and lightweight system. However, in the industry scenario, the data can be expanded enormously. For example, the involvement of many clinics and hospitals result in the the increasing amount of patient' data stored in both blockchain and local database. In fact, our prototype is explicitly experimented in for a medium set of users and doctors, which is hard to test the whole performance of the application. As a result, it is necessary to consider the scale-up for industry version in the future.

5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

Overall, Sona is a electrical healthcare storage that aims to empower patients in controlling their data and prevents data leak from some internal people for commercial purposes. As the rising of blockchain technology and distributed system, the internet has sounded an alarm about protecting the integrity of data. Clearly, medical data is one of the most important and sensitive information which worth the highest level of protection. We believe our solution will bring the security to user's data and also enhance the awareness of each citizen about the risks of stealing their data, especially in medical fields.

5.2 Future Work

Although the scope of the module can be shorter, we always head to the long term purpose to scale our application. Particularly, we intend to enhance in three aspects: business, technology and user experience. Firstly, we plan to gain more attention for the community with a marketing strategy. Sona can become a helpful blockchain service for the hospitals all over the world. Secondly, technology involves in scaling product features and enhance performance. Sona can be broaden in drug traceability, clinical trials, and drugs claims and billing. Improving function not only makes a robust and various service but also lead to a more profitable business. However, feature enhancement lead to a problem in performance and user experience. We solve the performance by using a more complex, and scale microservice architecture, which will involve many tiny services working together through remote procedure call and message queues to orchestrate the user requests. Moreover, we can also use some additional techniques like caching and database duplication to improve system performance significantly. Last but not least, user experience will be working closely with UI designs. It is necessary to continuously research on patient's experience when using our application and remarkably upgrade our UI.

6 APPENDIX

6.1 Appendix A: Roadmap and Work Distribution

Gantt Chart				
	Sprint 1	Sprint 2	Sprint 3	
Tasks	Week 1	Week 2	Week 3	G Day
Analyse market and review members' idea				
Draw diagrams and write product backlog				
UI Wireframe design				
Outline System Requirements				
Set up Hyperledger Fabric network				
Define the smart contract data types				
Ledger design				
Smart Contracts design				
Smart Contracts implementation				
Chaincode execution				
Chaincode API for ReactJS				
Backend web architecture				
UI Design and Mockup version				
Login / Register API				
Frontend Architecture				
Tailwind and React components				
React Pages and Fetch data				
Prototype version				
MVP launch				
Sale presentation				
Technical presentation				

Figure 73: Project's Gantt Chart

Task distribution					
Responsibilities	Bui Le Phi Long	Huynh Cam Tu	Le Duc Minh	Nguyen Quoc Trung	Truong Canh Thanh Vinh
Hyperledger Fabric Network	x	x	x	x	x
System Overall Requirements	x	x	x	x	x
System Entities and Users	x	x	x		
Ledger Design	x	x			
Smart Contracts Design	x	x			
Application Chaincode Implementation	x	x	x		
Chaincode API	x	x	x		
Authentication & Authorization (MongoDB) API					x
UX/UI				x	x
Redux state management				x	
Presentation / Pitching / Sprint Preview	x	x	x	x	x
Final Documentation	x	x	x	x	x

Figure 74: Tasks Contribution

Final Document Contribution					
Responsilibities	Bui Le Phi Long	Huynh Cam Tu	Le Duc Minh	Nguyen Quoc Trung	Truong Canh Thanh Vinh
Introduction (Objectives, Motivation, Project scope, Limitation, Definition of terms)				x	
System Design and Implementation (System overview, User Scenarios, System entities and Organizations, System Implementation)		x			
Detail System Architecture (System components introduction, Backend architecture, Server and HF Connection)	x				
Frontend Architecture					x
Security Model	x				
Application Flow	x		x		
Result			x		x
Validation	x		x		x
Discussion					x
Conclusions and Future work					x

Figure 75: Final Documentation Contribution

6.2 Appendix B: Glossary

- API: Application Programming Interface
- JSON: JavaScript Object Notation
- SMBs: Small and middle-size business
- yaml: Ain't Markup Language
- SDK: Software development kit
- Auth: Authentication and Authorization
- cpp: Connection configuration path
- uuid: Universally unique identifier
- msps: Membership Service Providers

6.3 Appendix C: Source code

Please visit our github repository: `https://github.com/philongbuile/SONA`
Tutorial for testing our system:

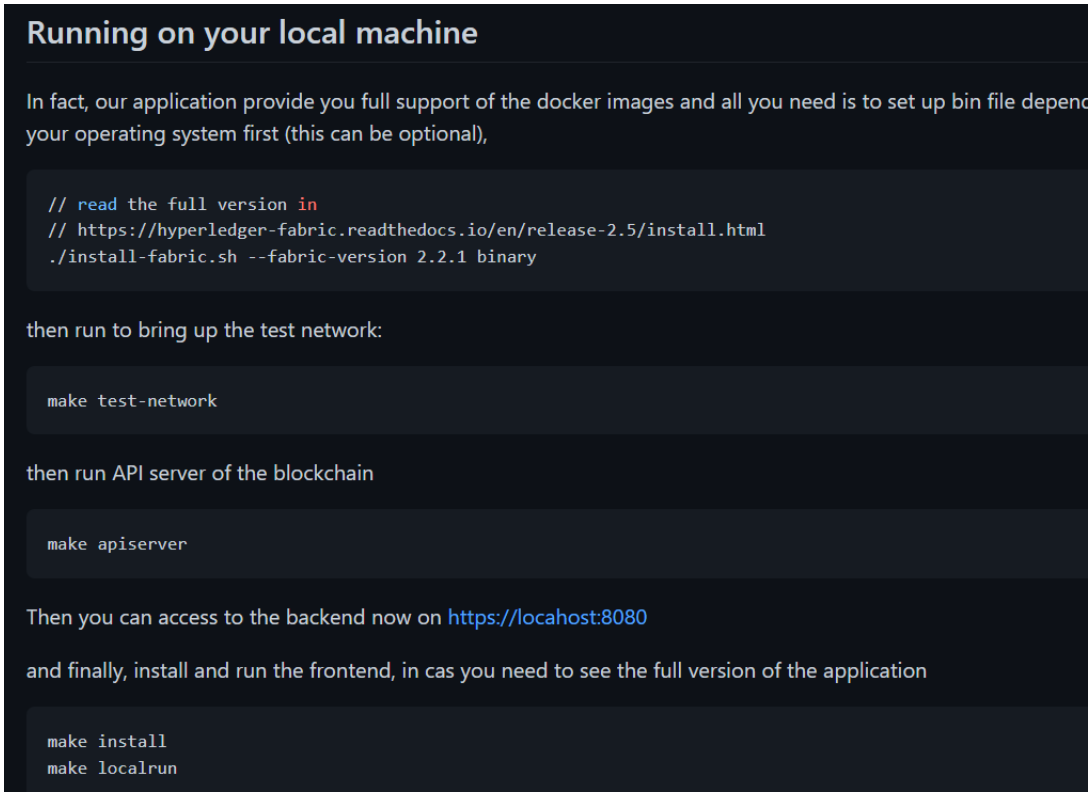


Figure 76: Tutorial for running local network and web app system

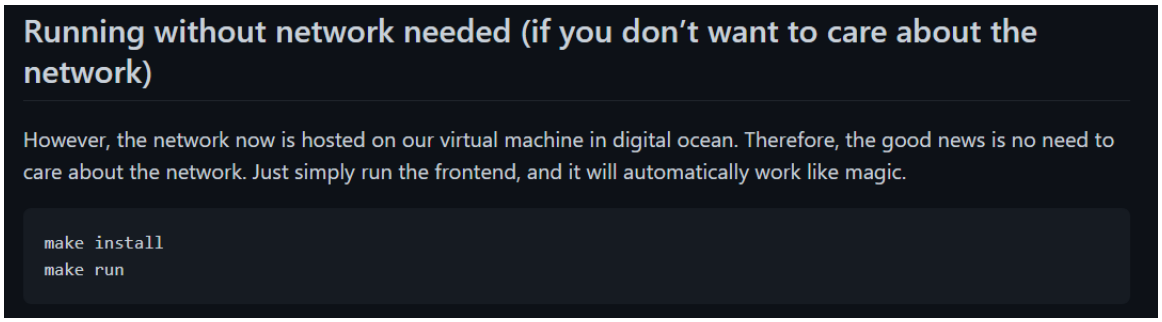


Figure 77: Tutorial for running online host network web application

REFERENCES

[1] A. D. H. Agency. Myhealth record the big picture. 2021. URL https://www.digitalhealth.gov.au/initiatives-and-programs/my-health-record/statistics.

[2] CouchDB. Couchdb first glance. 2022. URL https://couchdb.apache.org.

[3] Cynet. Understanding privilege escalation and 5 common attack techniques. 2022. URL https://www.cynet.com/network-attacks/privilege-escalation/.

[4] DigitalOcean. Digital ocean product for developer brand. 2022. URL https://www.digitalocean.com.

[5] ExpressJS. Fast, unopinionated, minimalist web framework for node.js. 2022. URL https://expressjs.com.

[6] ExpressJSTeam. Using middlewares. 2022. URL https://expressjs.com/en/guide/using-middleware.html.

[7] H. Fabric. Hyperledger fabric ca client. 2022. URL https://hyperledger.github.io/fabric-sdk-node/release-2.2/FabricCAClient.html.

[8] H. Fabric. The ordering service. 2022. URL https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering_service.html.

[9] H. Fabric. Hyperledger fabric sdk for node.js. 2022. URL https://hyperledger.github.io/fabric-sdk-node/release-2.2/index.html.

[10] H. Fabric. Wallet. 2022. URL https://hyperledger-fabric.readthedocs.io/en/release-2.2/developapps/wallet.html.

[11] HyperledgerFabric. Private data collection. 2022. URL https://hyperledger-fabric.readthedocs.io/en/latest/private-data-arch.html.

[12] HyperledgerFabric. Smart contract. 2022. URL https://hyperledger-fabric.readthedocs.io/fa/latest/smartcontract/smartcontract.html.

[13] HyperLedgerFabric. A blockchain platform for the enterprise. 2022. URL https://hyperledger-fabric.readthedocs.io/en/latest/.

[14] HyperledgerFabric. Nodejs sdk for hyperledger fabric. 2022. URL https://hyperledger.github.io/fabric-sdk-node/release-2.2/index.html.

[15] Impreva. Remote code execution. 2022. URL https://www.imperva.com/learn/application-security/remote-code-execution/.

[16] MongoDB. What is mongodb. 2022. URL https://www.mongodb.com/what-is-mongodb.

[17] ReactJS. An overview of react documentation and related resources. 2022. URL https://reactjs.org/docs/getting-started.html.

[18] E. Zhang. What is role-based access control (rbac)? examples, benefits, and more. 2022. URL https://www.mongodb.com/what-is-mongodb.