Decentralized Medical Records System Using Blockchain

Graduation Project, Part-II (SWE 497)
Software Engineering Department
CCIS, KSU

Project Advisor:
Amanullah Quadri

Submitted by

Abdulaziz Alobaili	434108440
Wadah Esam	435108270
Abdulrahman Alkailani	435108422
Muhammad Alsadat	435108418

Date Submitted
Wednesday 10 April, 2019

ABSTRACT

The aim of this project is to develop a Decentralized Medical Record System where every patient has a medical record shared among all related stakeholders of the healthcare network. Medical records are shared using the Blockchain technology. This system would improve the overall services for patients, and guarantee that medical records are perpetually free of any alterations, which provides privacy services to ensure role-based access. This system will be implemented as a web application, and another web application is going to be developed for the patients, which will provide the ability to timely and efficiently access their medical records.

Table of Contents

List of Figures

List of Tables

1. Introduction

Healthcare medical records has a long history of being inefficient, expensive and vulnerable. Currently, each hospital has its own copy of a patient's medical record. Every time a patient visits a hospital, the hospital updates the patient's medical record. As years pass, the patient ends up having dozens of separate medical records stored in every hospital he has visited. This system of storing records has been around since the 13'th century, and people have been trying to digitize a system through traditional approaches. It is expensive in terms of the amount of time and effort required to manage medical records and perform internal and external audits across different medical facilities. It is also vulnerable to fraud, malicious modification, or a cyber-attack. Mistakes are difficult to render should they get into the records. This is the current situation before Blockchain.

This project attempts to establish a system that addresses all the above problems by providing an extremely reliable way of storing, retrieving, and sharing medical records among all medical facilities. With Blockchain, all members of the network share a common record on the blockchain. They all have access to this record and it is replicated through peer-to-peer replication technologies across all members of the blockchain network. Privacy services are used to determine who can see what. For example, a doctor and a patient can see the full details of the patient's record, but a pharmacy can only see which drugs were prescribed for a patient.

Blockchain applies 4 important principles: Consensus (determining who within the network gets to validate or approve a transaction), Provenance (a complete record of the patient history through his life), Immutability (impossible to tamper with the blocks once they are written), and Finality (once a piece of information is committed into the blockchain, we have got one system of records across the network which can effectively assure that disputes are very easy to resolve should they occur).

The Decentralized Medical Record System will be implemented using the Blockchain technology. Allowing the medical records to be shared among all registered medical facilities preventing any attempt to mutate the data. A web application is used so that patients can view their records, and medical facilities can only store to and retrieve from existing records.

1.1 Blockchain

1.1.1 What is Blockchain?

Blockchain is a decentralized peer-to-peer database in which all the participants (called nodes) are at the same hierarchical level and because of the existence of a consensus protocol, don't need to trust each other. Moreover, every participant has a copy of the whole ledger of the blockchain.

Decentralized means that the information is not stored in a central server, but in all the individual servers or computers of each participant running a node.

Peer-to-Peer means that the information is shared between participants in pairs. Every participant running a node is connected to other nodes in the network, and all information about own or alien transactions is shared between them in pairs. Then, they replicate and iterate this action with their connections.

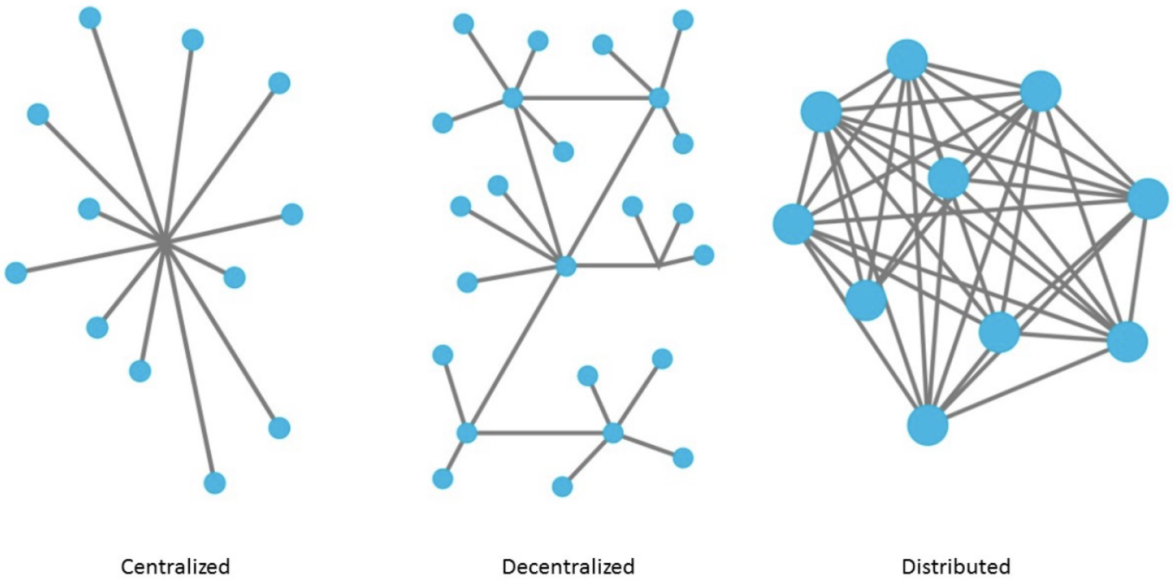


Figure 1 Illustration of a decentralized network compared to other types of networks

1.1.2 Blockchain Principles

- Consensus: Who within the network validates a transaction
    - The most important and revolutionary idea. This principle insures that only correct blocks are added to the blockchain, and in case someone introduces the wrong one, the other honest participants refuse this block.

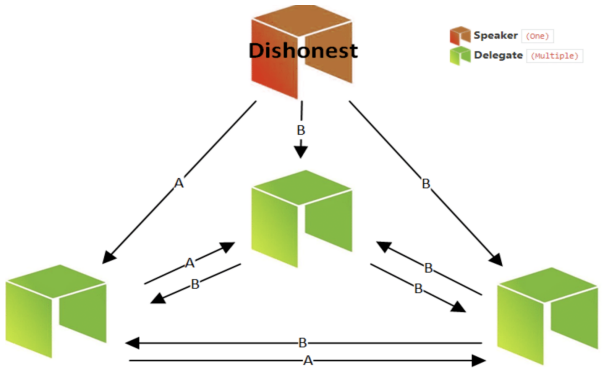


Figure 2 Consensus

- Immutability: Impossible to tamper with the blocks once they are written.
- Provenance: One complete record of the patient throughout his life.
- Finality:
    - Once a piece of information is committed, it's committed forever.
    - Assures that disputes are very easy to solve.

2. Analysis and Design

There are a few electronic medical record systems that in a way serve the patient and medical facilities, but they are either centralized (do not connect facilities in a shared network), or have the possibility of altering the data. In addition, the Ministry of Health authority does not have access, and pharmacy systems are isolated and work independently.

Table 1 lists some of these systems and provide a brief comparison between them and our project:

Table 1 DMRS in comparison to other systems in the domain

OVERVIEW	AdvancedMD	MicroMD	ReliMed	Sevocity	DMRS
PRODUCT NAME	AdvancedEHR	MicroMD EMR	ReliMed EMR	Sevocity EHR	DMRS
DELIVERY PLATFORM					
CLOUD	✓	✓	✓	✓	✓
ON-PREMISE	✗	✓	✓	✗	✓
ADDITIONAL INFORMATION					
MULTI-OFFICE	✓	✓	✓	✓	✓
WEB APP	✓	✓	✗	✓	✓
SOFTWARE FEATURES					
MEDICAL RECORD ACCESS FOR PATIENTS	✓	✓	✗	✗	✓
DATA MANIPULATION	✓	✓	✓	✓	✗
GOVERNMENT ACCESS	✗	✗	✗	✗	✓
SHARED SYSTEM BETWEEN MEDICAL FACILITIES	✗	✗	✗	✗	✓
PHARMACY INTEGRATION	✗	✗	✗	✗	✓
CENTRALIZATION	✓	✓	✓	✓	✗

2.1 Updated Functional Requirements

2.1.1 Hospital Related Requirements

1. Hospital shall be able to check if a patient has a medical record by patient's national ID.
2. Hospital shall be able to create a patient record containing his name, date of birth, emergency contacts, mobile number, national number, and gender.
3. Hospital shall be able to access medical record using patient's national ID.
4. Hospital shall be able to add data to a patient record:
   4.1. Hospital shall be able to add diagnosis details to a patient record.
   4.2. Hospital shall be able to add surgery information to a patient record.
   4.3. Hospital shall be able to add radiology scans to a patient record.
   4.4. Hospital shall be able to add laboratory test results to a patient record.
   4.5. Hospital shall be able to add drug prescription to a patient record.
   4.6. Hospital shall be able to add blood transfer information to a patient record.
5. Hospital shall be able to view a patient's medical record.
6. Hospital shall be able to create a corrective transaction and link it to the original transaction.

2.1.2 Ministry of Health Related Requirements

1. Ministry of Health shall be able to add a new hospital to the network.
2. Ministry of Health shall be able to add a new pharmacy to the network.

2.1.3 Patient Related Requirements

1. Patient shall be able to log in to his medical record using his national identification number.
2. Patient shall be able to view his medical record:
   2.1. Patient shall be able to retrieve all past diagnosis details from a patient record.
   2.2. Patient shall be able to retrieve all past surgery history from a patient record.
   2.3. Patient shall be able to retrieve all past radiology scans from a patient record.
   2.4. Patient shall be able to retrieve all past laboratory results from a patient record.
   2.5. Patient shall be able to retrieve all past drug prescriptions from a patient record.
   2.6. Patient shall be able to retrieve all history of blood transfers from a patient record.

2.1.4 Pharmacy Related Requirements

1. Pharmacy shall be able to view all old drug prescriptions of any patient.
2. Pharmacy shall be able to track which medicines were dispensed.
3. Pharmacy shall be able to mark a specific drug from the drug prescription as dispensed.

2.2 Updated Non-Functional Requirements

2.2.1 Usability

- The app will have a help and training menu – when accessed will play a 3 minutes video that will train the user about how to use the app.

2.2.2 Availability

- There will be a node acting as a backup to the blockchain incase all other nodes shut down, thereby, insuring availability.

2.2.3 Performance

- Response time: Average response time shall be less than two seconds.
- Capacity: System shall be able to process at least 2000 transactions per second (based on the capacity of the Ethereum blockchain).

2.2.4 Supportability

- System shall support English languages with the ability of adding Arabic support in the future.

2.3 Updated Use Case Requirements

The following shows the use case from the hospital perspective:

Figure 3 Hospital Use Case Diagram

The following shows the use case from the ministry of health perspective:

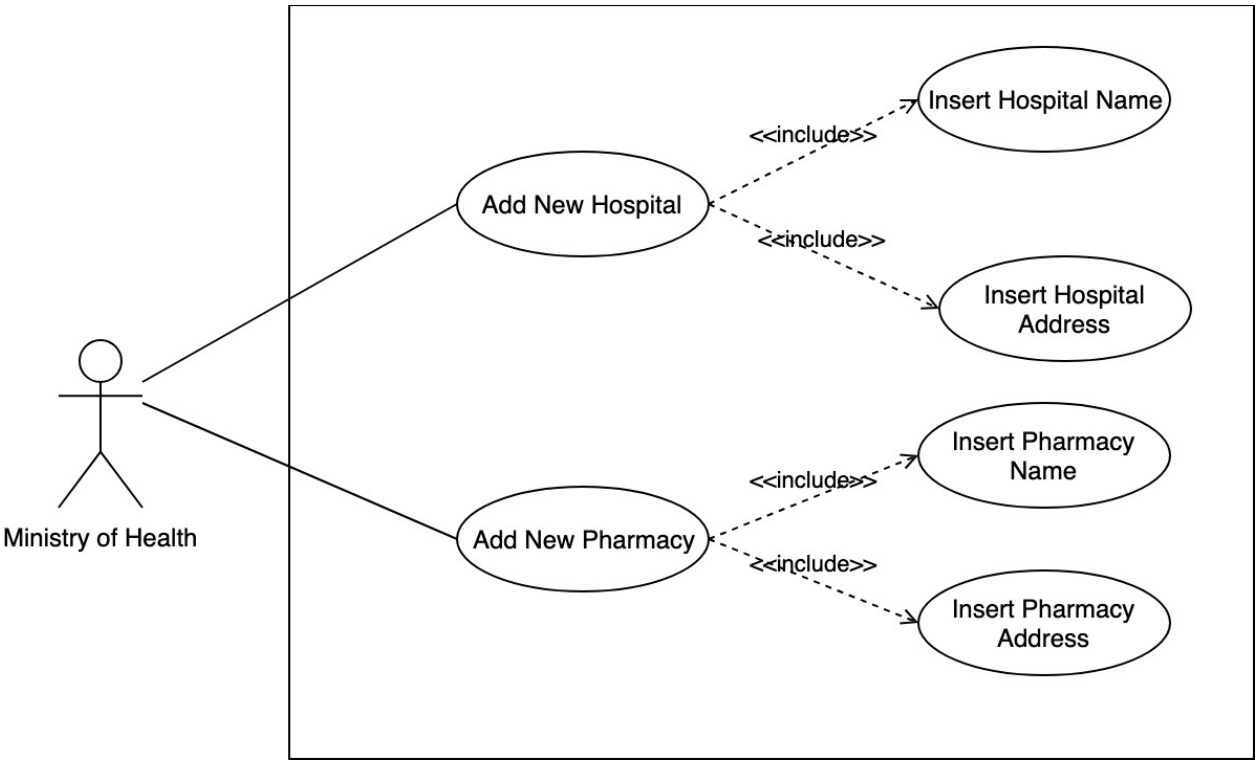


Figure 4 Ministry of Health Use Case Diagram

The following shows the use case from the patient perspective:

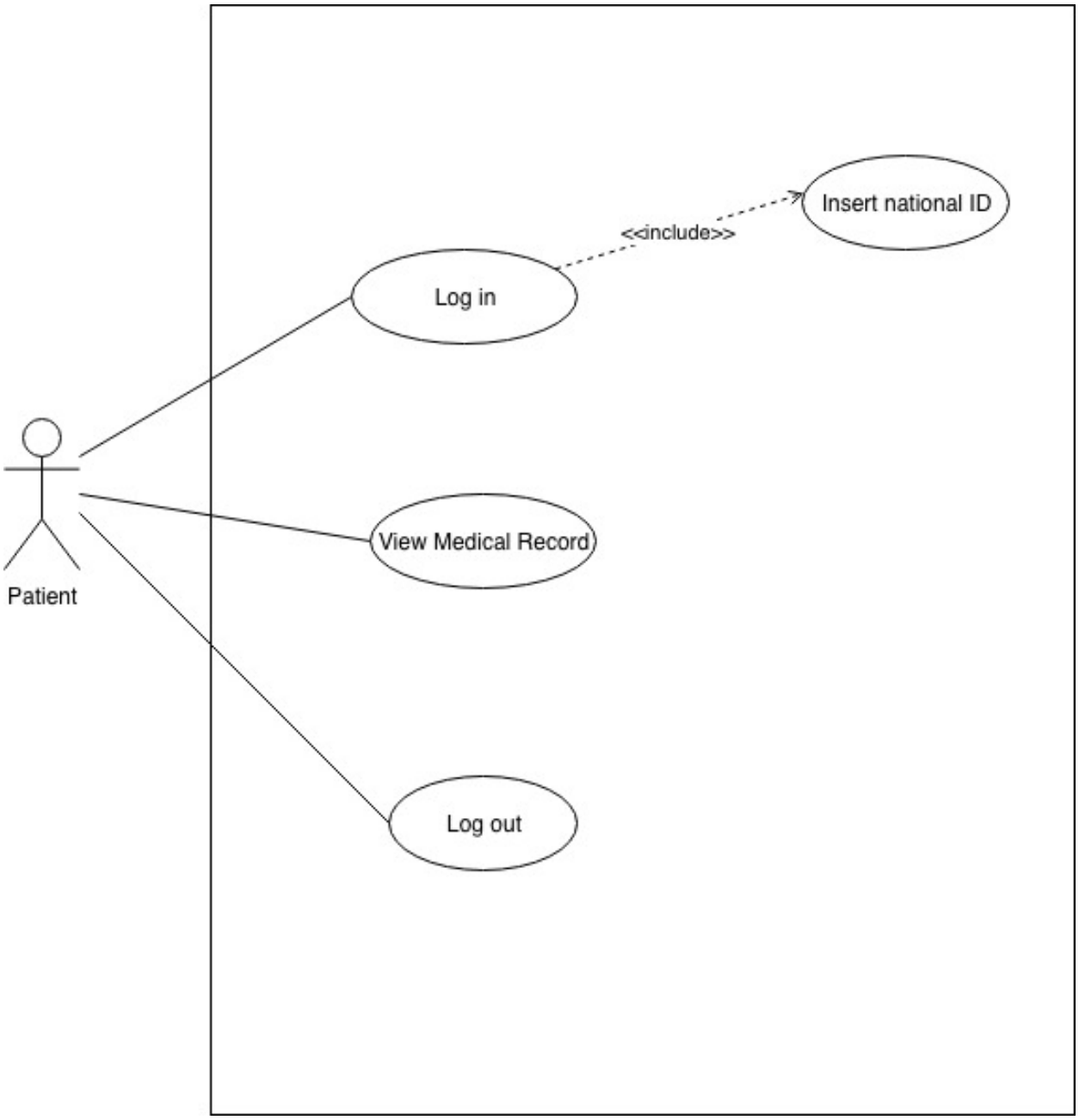


Figure 5 Patient Use Case Diagram

The following shows the use case from the pharmacy perspective:

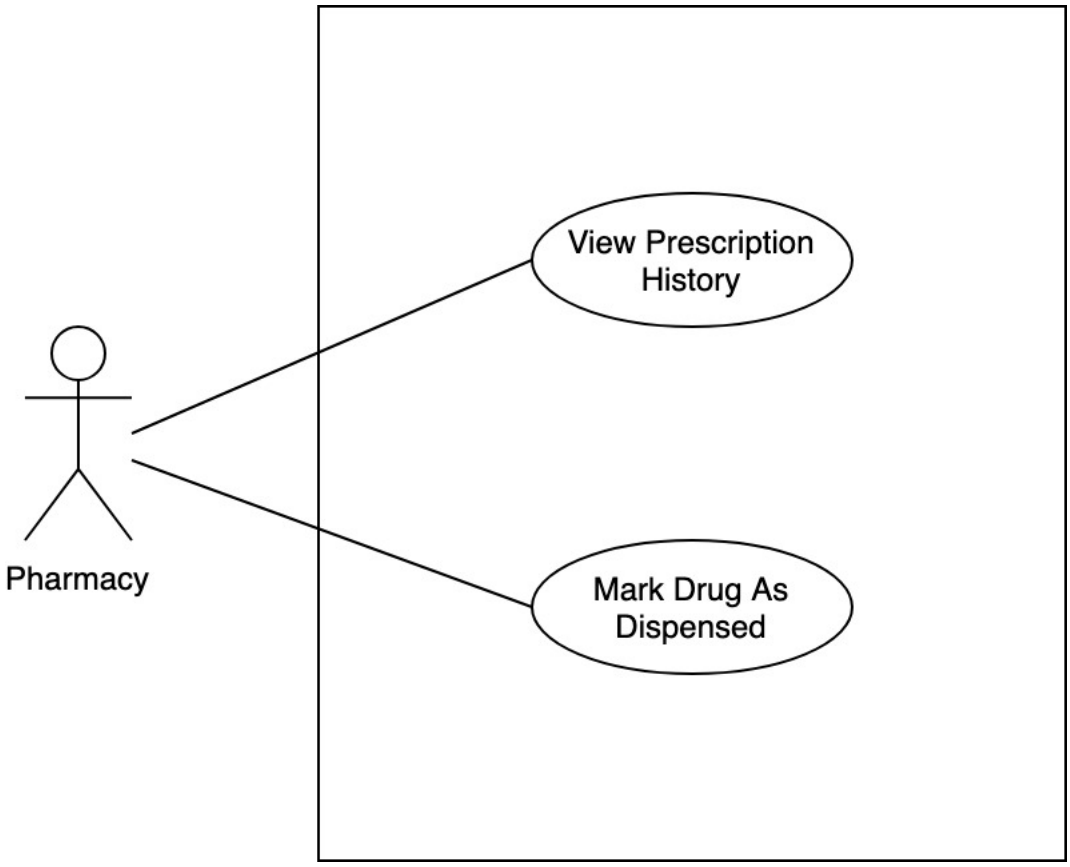
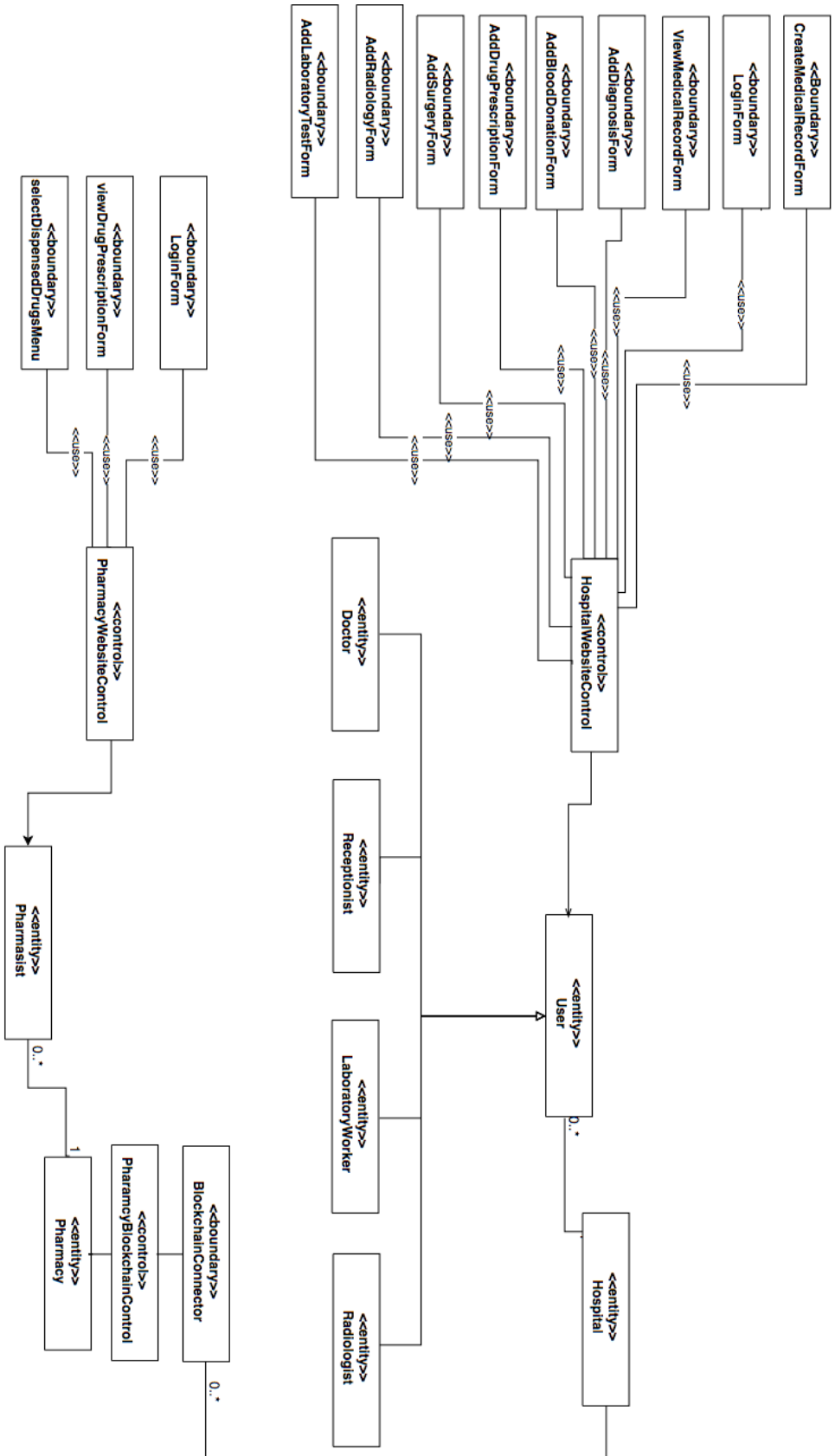
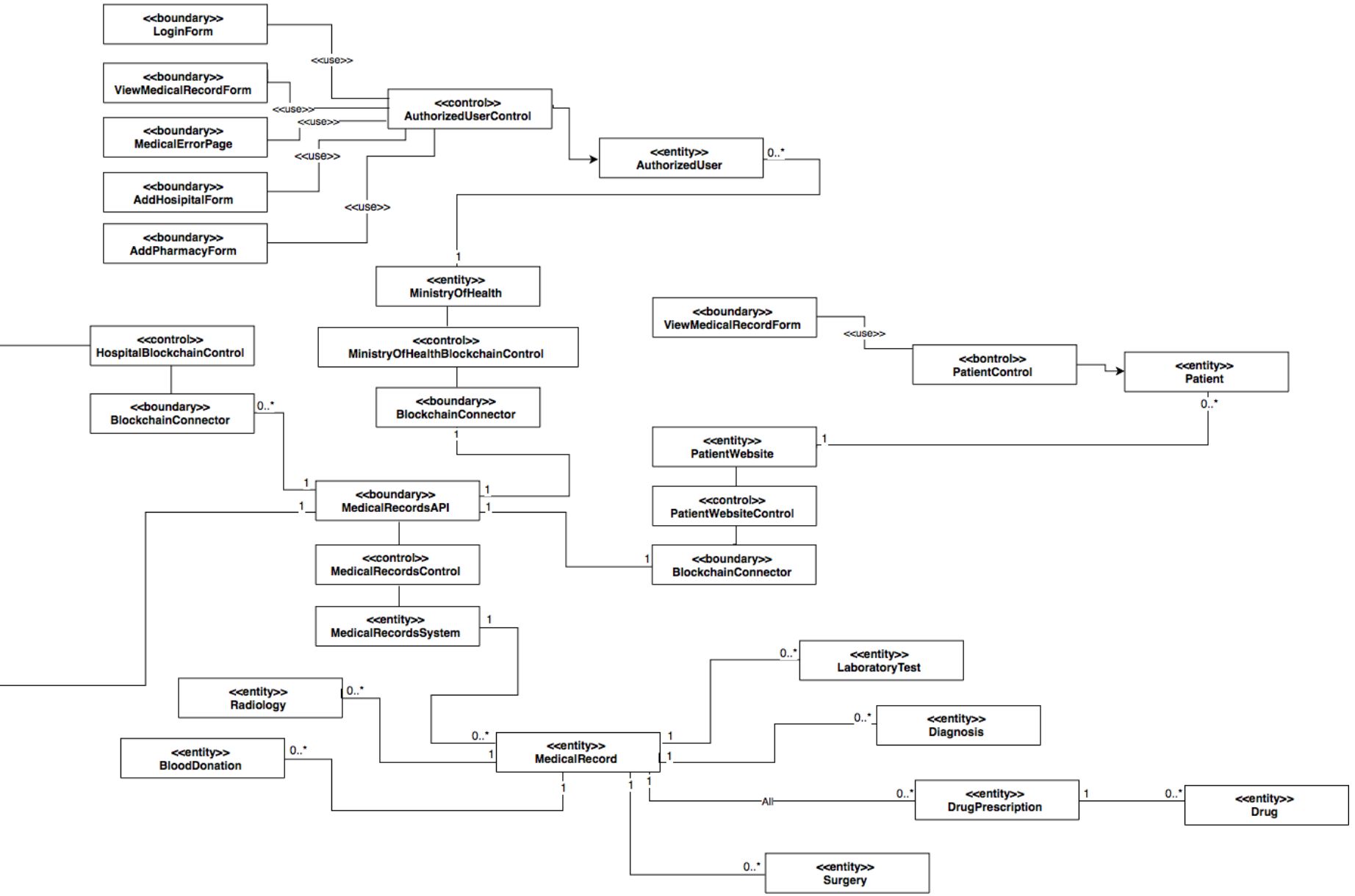


Figure 6 Pharmacy Use Case Diagram

2.4 Design Classes

The following is an abstract analysis class for the overall system. Half of the analysis class is covered in this page and the other half in the next page. For better view of the figure, visit this link:
https://drive.google.com/file/d/11asBf5ViKPbIVNde74T4_y7_y-Z7yBqW/view?usp=sharing




<<boundary>>
LoginForm
<<boundary>>
ViewMedicalRecordForm
<<boundary>>
MedicalErrorPage
<<boundary>>
AddHosipitalForm
<<boundary>>
AddPharmacyForm
<<use>>
<<use>>
<<use>>
<<use>>
<<use>>
<<control>>
AuthorizedUserControl
<<entity>>
AuthorizedUser
0..*
1
<<entity>>
MinistryOfHealth
<<control>>
HospitalBlockchainControl
<<control>>
MinistryOfHealthBlockchainControl
<<boundary>>
BlockchainConnector
<<boundary>>
BlockchainConnector
0..*
1
1
1
1
<<boundary>>
MedicalRecordsAPI
<<control>>
MedicalRecordsControl
<<entity>>
MedicalRecordsSystem
1
<<boundary>>
ViewMedicalRecordForm
<<use>>
<<bontrol>>
PatientControl
<<entity>>
Patient
0..*
<<entity>>
PatientWebsite
1
<<control>>
PatientWebsiteControl
<<boundary>>
BlockchainConnector
1
<<entity>>
LaboratoryTest
0..*
<<entity>>
Radiology
0..*
<<entity>>
Diagnosis
0..*
<<entity>>
BloodDonation
0..*
0..*
<<entity>>
MedicalRecord
1
1
1
1
1
All
0..*
<<entity>>
DrugPrescription
1
0..*
<<entity>>
Drug
0..*
<<entity>>
Surgery

2.5 Sequence Diagram

2.5.1 View Drug Prescriptions

The following shows the sequence diagram between the pharmacy class and other classes in order to view drug prescriptions.

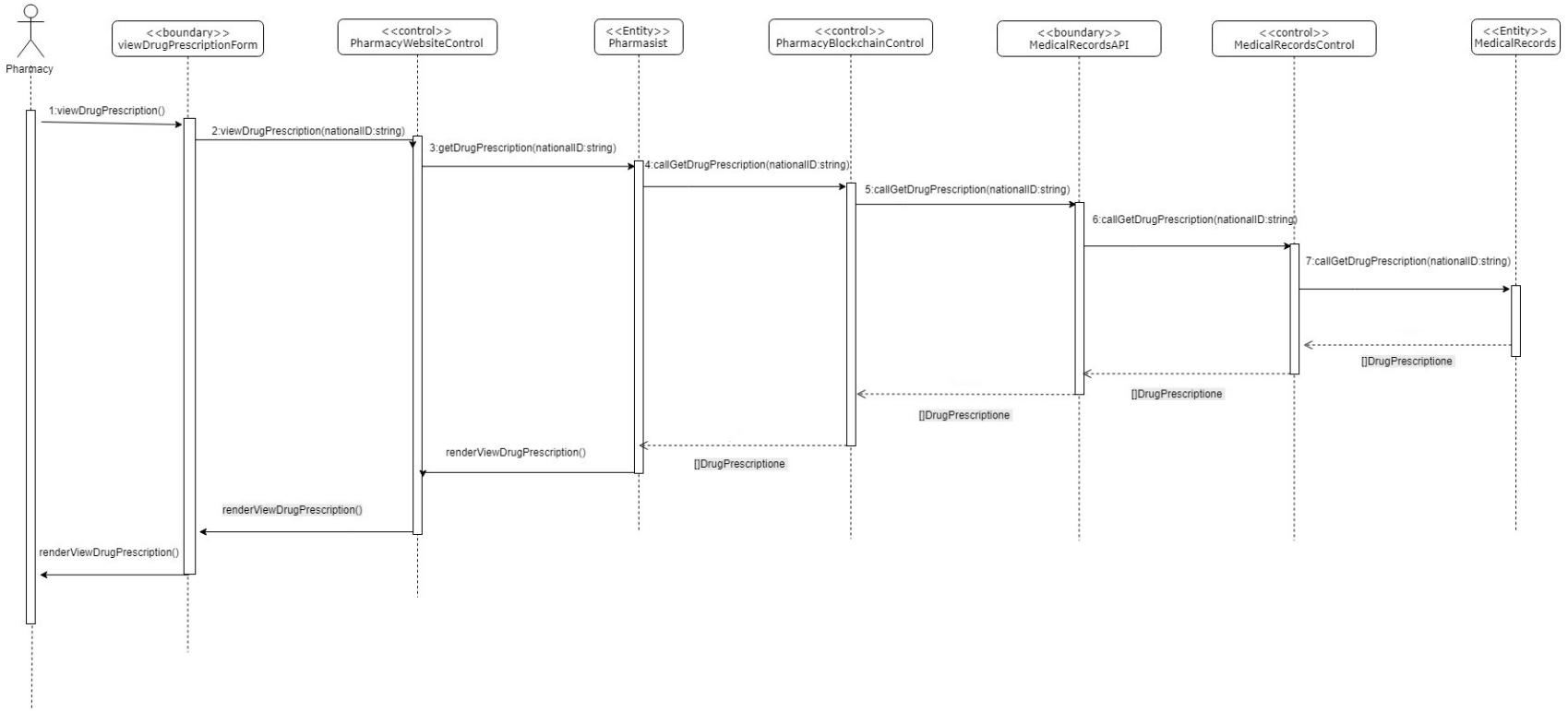


Figure 7 Pharmacy interaction diagram

The following shows the sequence diagram between the hospital class and other classes in order to mark a transaction as a medical error.

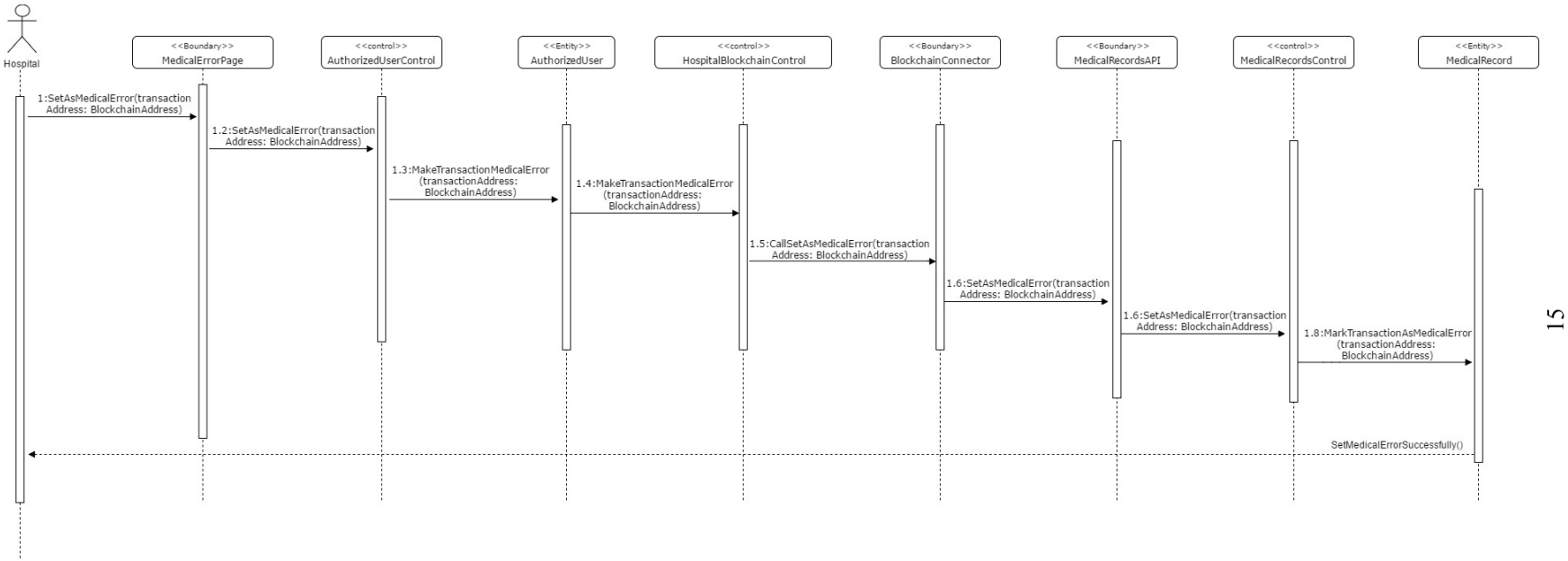


Figure 8 markMedicalError interaction diagram

2.5.2 Mark Transaction as Medical Error

The following shows the sequence diagram between the hospital class and other classes in order to create a medical record for the patient.

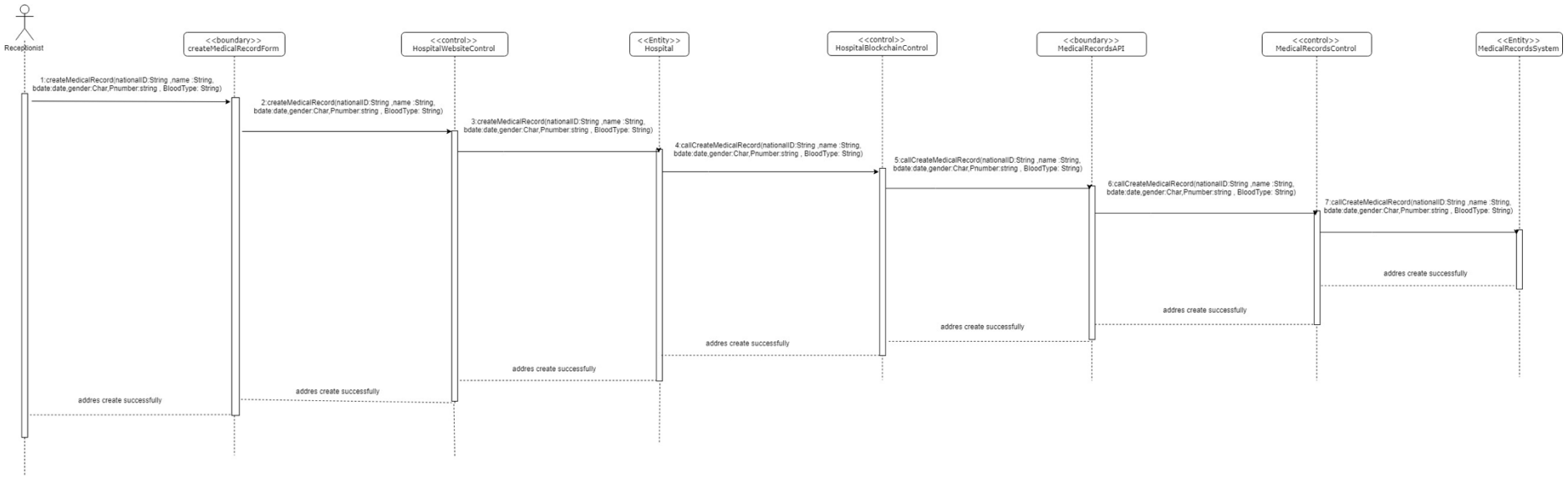


Figure 9 creatPatient interaction diagram

2.5.3 Create Medical Record

The following shows the sequence diagram between the ministry of health class and other classes in order to add a new hospital to the system.

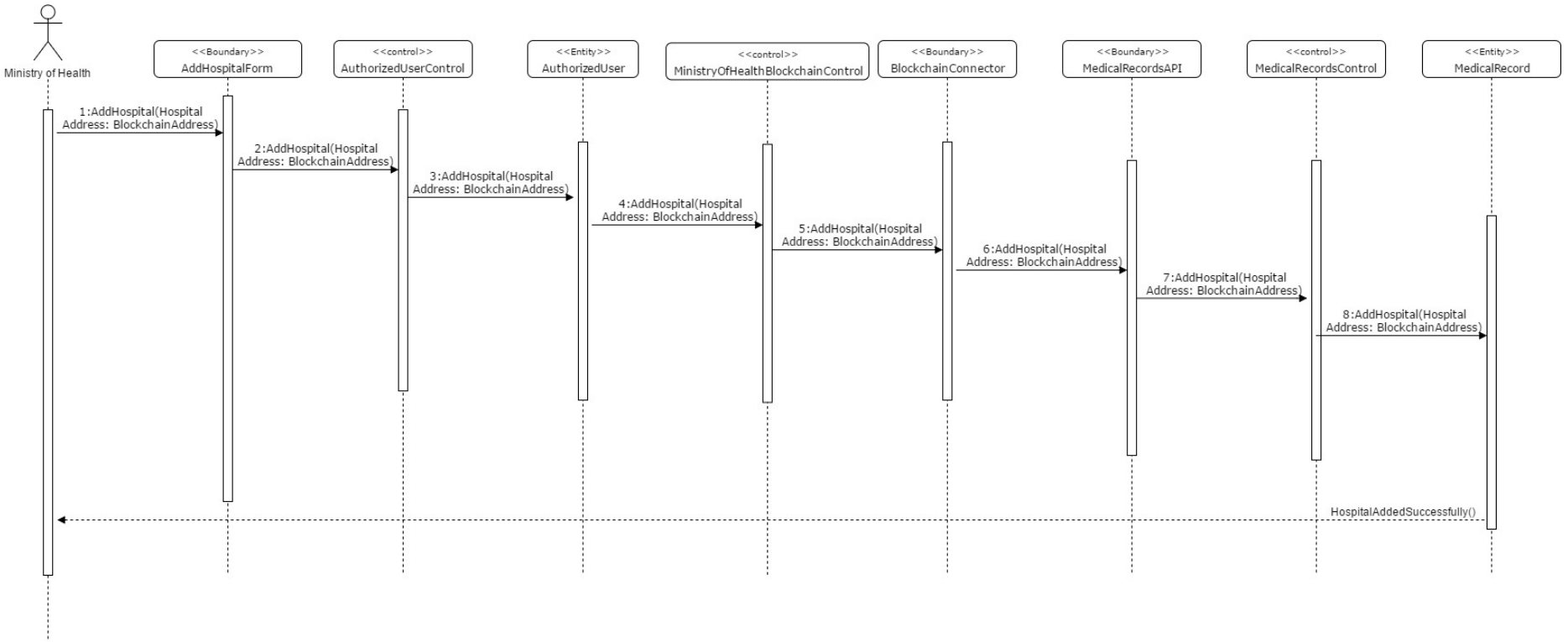


Figure 10 addHospital interaction diagram

2.5.4 Add New Hospital

2.6 Software Architecture

Since it is a decentralized, peer-to-peer system. It consists of a number of nodes, each of which has a local copy of the blockchain. The nodes belong to different hospitals. The nodes communicate to each other in order to gain agreement on the contents of the blockchain, and do not require a central authority to validate the transactions. One of the nodes will represent the web app to which the users will connect to view their medical records. See the figure.

As in figure, a blockchain consists of a chain of blocks, each block contains the following:

- Proof of work nonce: The random value in a block that was used get the proof of work satisfied.
- Data block: which contains the list of transactions (data added to the patient medical record).
- Hash: a hexadecimal number unique to the block.

Previous: the hexadecimal number referencing the previous block.

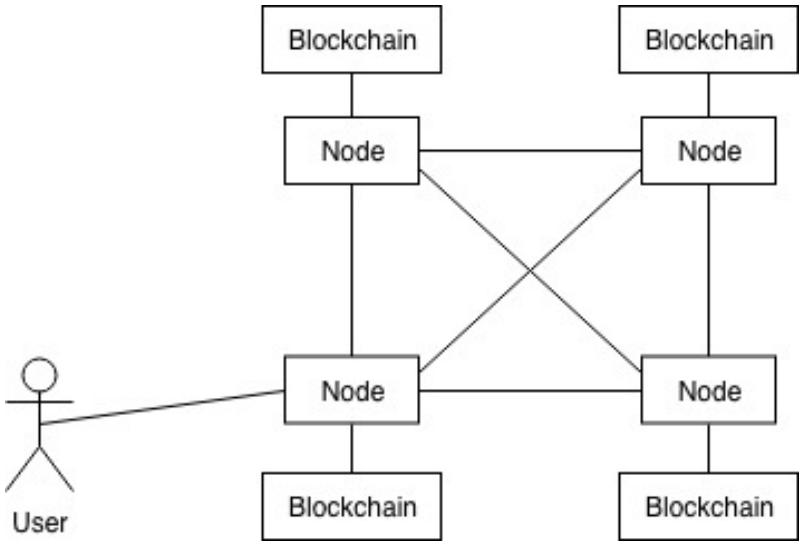


Figure 11 Ethereum network architecture

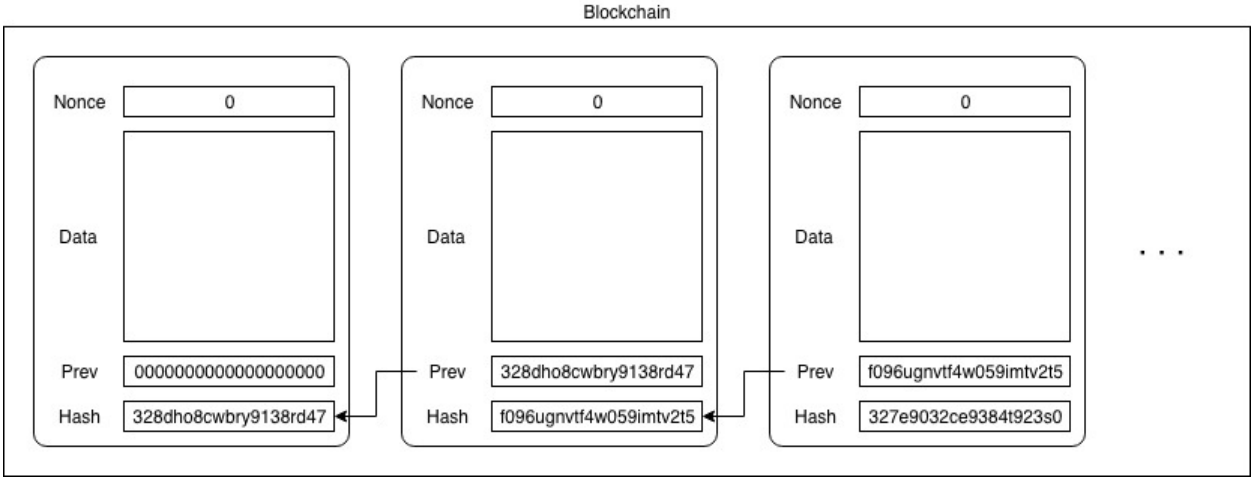


Figure 12 Blockchain architecture

3. Prototype Description

The four systems in this project (Ministry of Health, Hospital, Pharmacy, and Patient) are demonstrated using four computers, each connected to one distributed blockchain network. The blockchain network will act as the data source containing all patients' medical records.

3.1 Implementation Platform

The following is the list of technologies hardware/software in addition to implementation platform used:

- HTML, CSS, and JavaScript
- React library
- Angular framework
- Ethereum Platform
- Solidity (OOP language for writing smart contracts)
- Cloud 9 platform
- Remix (open source tool for writing Solidity smart contracts from a web browser)
- Ganache (personal blockchain for Ethereum development)
- MetaMask (Ethereum blockchain adapter, allows to run decentralized applications from a web browser).

3.2 Mapping between Requirements and Implemented Functions

The following table shows a mapping between the projected functionalities of the system and the corresponding implemented modules in the Ministry of Health:

Table 2. Ministry of Health mappings

Functional requirements	Modules/Functions/Class that implemented this feature
FR2.1.2.1 Add new hospital	`addHospital(address hospitalAddressI, string memory hospitalName)`
FR2.1.2.2 Add new Pharmacy	`addPharmacy(address pharmacyAddressI, string memory pharmacyName)`

The following table shows a mapping between the projected functionalities of the system and the corresponding implemented modules in the Hospital:

Table 3 Hospital mappings

Functional requirements	Modules/Functions/Class that implemented this feature
FR2.1.1.1 Check if a medical record has already been created.	`checkMedicalRecord(uint256 nationalIDI)`
FR2.1.1.2 Create a medical record	`createMedicalRecord(uint256 nationalID, string memory name, uint dateI, string memory phoneNumberI, string memory genderI, string memory bloodTypeI, string memory emergencyContactI)`
FR2.1.1.3 View medical record using national ID	`getMedicalRecord(uint256 nationalIDI)`
FR2.1.1.4 Add data to medical record	All setter methods in `MedicalRecord.sol`
FR2.1.1.5 View medical record	`getMedicalRecord(uint256 nationalIDI)`
FR2.1.1.6 Mark corrective transaction	`markTransactionAsMedicalError(uint _type, uint _id)`

The following table shows a mapping between the projected functionalities of the system and the corresponding implemented modules in the Hospital:

Table 4 Patient mappings

Functional requirements	Modules/Functions/Class that implemented this feature
FR2.1.3.1 Login using national ID	`getUser()`
FR2.1.3.2 View Medical Record	`getPatientMedicalRecordData(data)`

The following table shows a mapping between the projected functionalities of the system and the corresponding implemented modules in the Hospital:

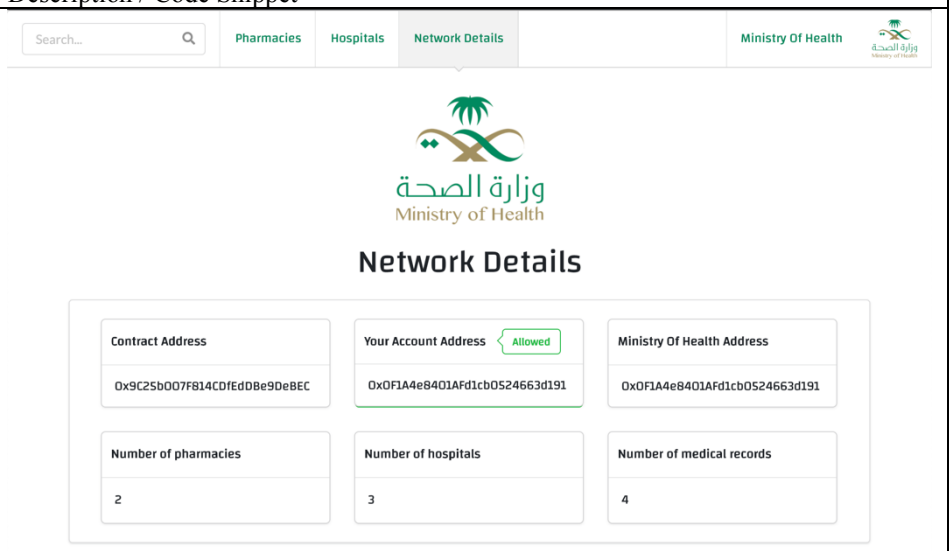
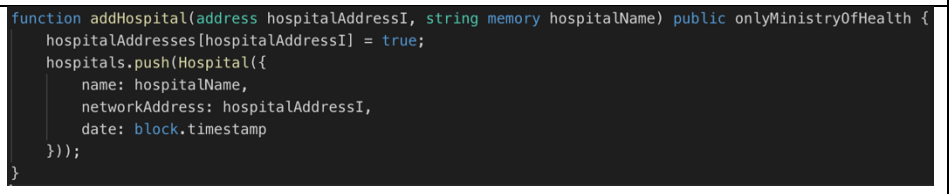
Table 5 Pharmacy mappings

Functional requirements	Modules/Functions/Class that implemented this feature
FR2.1.4.1 View drug prescriptions	`getPatientDrugPrescribtions(data)`
FR2.1.4.2 Check which drugs were dispensed	`getPatientDrugPrescribtions(data)`
FR2.1.4.3 Mark Drug As Dispensed	`markDrugAsDispensed(patientId, drugID, isTaking)`

3.3 Implementation Details

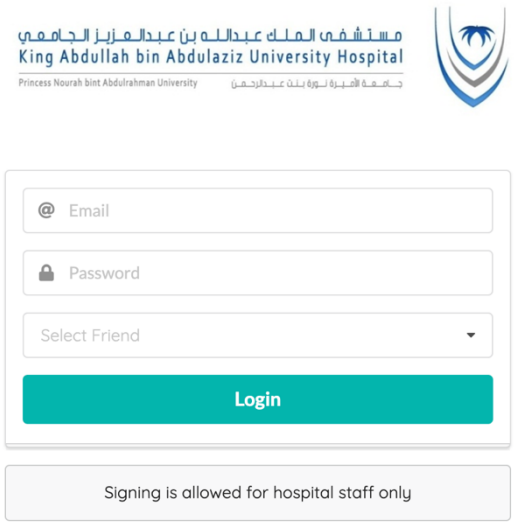
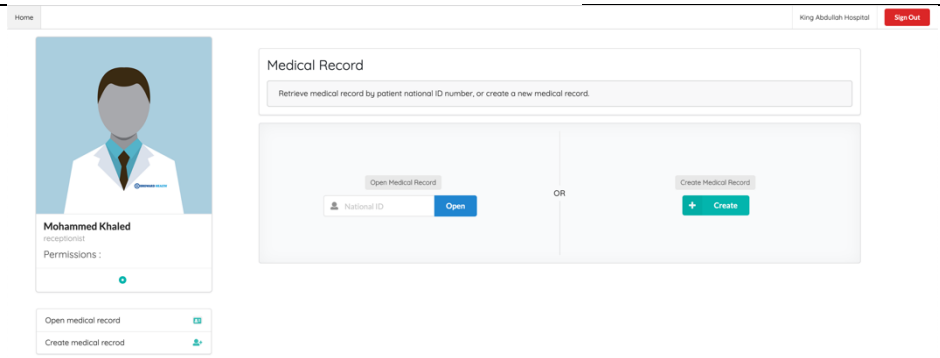
Below are the implementation details for four main features, Add New Hospital, Create Medical Record, Mark Transaction as Medical Error.

3.3.1 Add New Hospital

Step	Description / Code Snippet
Home page This is the starting point of the Ministry of Health system. Here, the details of the connected user are displayed along with statistics (number of pharmacies, number of hospitals, number of medical records). The user can then click the "Hospitals" section from the navigation bar to add a new hospital.	
Submitting the Add New Hospital form The information from the form are saved in the state of the Hospitals component, then a request is sent using MetaMask to create a new transaction containing the details of the new hospital. When successful, a push notification is called notifying the user that the hospital has been added successfully.	
`addHospital` function in the `MedicalRecordsSystem.sol` smart contract This is the smart contract code that will validate that the request is coming from the authorized user and add the hospital to the blockchain as a new transaction.	

3.3.2 Create Medical Record

Step	Description / Code Snippet

Login page This is the starting point of the Hospital system. Here, a sample of one of the connected hospitals is displayed, a staff member can then log in to view the home page. In the case of this prototype, it is assumed that the hospital receptionist is the authorized user to create a new medical record.	
Home page for the receptionists Here, The receptionist can click the "Create" button to open the form for creating the medical record.	

`submitForm` is the function that will verify the form inputs and submit a request to add the new medical record.
The form inputs are saved in the component state.
Using the logged in user account address, the system will call the method `createMedicalRecord` in the smart contract where the arguments are the information from the form.

When the response in successful, the user will be redirected to the newly created medical record.

```
submitForm = async () => {
  const patient = this.state;
  if (patient.gender == '') {
    this.setState({isError: true, errorMessage: 'Please select patient gender.'})
  } else if (patient.bloodType == '') {
    this.setState({isError: true, errorMessage: 'Please select patient blood type.'})
  } else if (patient.name == '') {
    this.setState({isError: true, errorMessage: 'Please enter patient full name.'})
  } else if (patient.birthDate == '') {
    this.setState({isError: true, errorMessage: 'Please enter patient birth date.'})
  } else if (patient.nationalID == '') {
    this.setState({isError: true, errorMessage: 'Please enter patient national id.'})
  } else if (patient.emergencyContact == '') {
    this.setState({isError: true, errorMessage: 'Please select patient emergency contact'})
  }
  else {
    this.setState({isError: false,})
    console.log('every thing looks fine')
    console.log(this.state)

    const accounts = await web3.eth.getAccounts();
    await contract.methods.createMedicalRecord(patient.nationalID, patient.name, patient.birthDate.getTime(),
      patient.phoneNumber, patient.gender, patient.bloodType, patient.emergencyContact)
    .send({ from: accounts[0], gas: '20000000' })
    .then(async () => {
      toast.success("Medical record created successfully", {
        position: "bottom-center",
        autoClose: 3000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        width: 200,
      });
      this.props.history.push(`/${patient.nationalID}/medicalRecord`)
    })
    .catch((e) => {
      toast.error("Error : You have to make this transaction with a verified account", {
        position: "bottom-center",
        autoClose: 3000,
        hideProgressBar: false,
        closeOnClick: true,
        pauseOnHover: true,
        draggable: true,
        width: 200,
      });
    });
    let checkMedicalRecord = await contract.methods.checkMedicalRecord(435108270).call();
    console.log(checkMedicalRecord);
  }
}
```

`createMedicalRecord` function in the `MedicalRecordsSystem.sol` smart contract

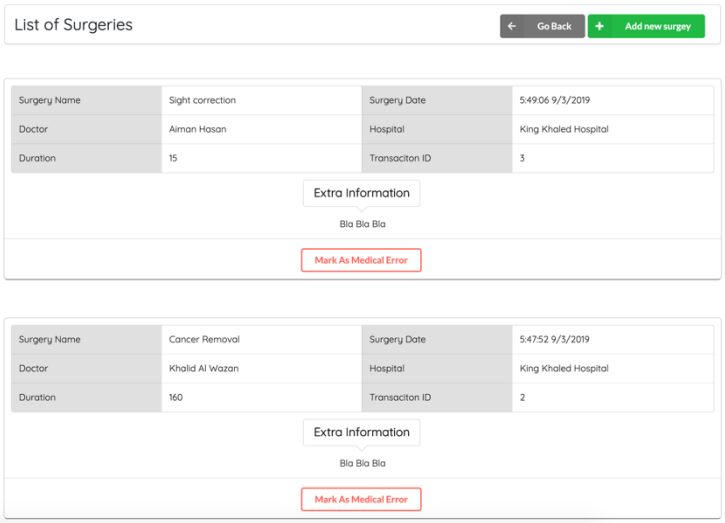
Here, the smart contract verifies that the request is coming from the authorized user and finds the hospital name from which the medical record is created, then saves the new medical record as a transaction in the blockchain.

```
function createMedicalRecord(uint256 nationalID, string memory name, uint dateI, string memory phoneNumberI, string memory genderI,
    string memory bloodTypeI, string memory emergencyContactI) public onlyHospitalsAndPharmacies {
    string memory hospitalName = "noName";
    for (uint i = 0; i < hospitals.length; i++) {
        if (hospitals[i].networkAddress == msg.sender) {
            hospitalName = hospitals[i].name;
        }
    }
    MedicalRecord newMedicalRecord = new MedicalRecord(nationalID, name, dateI, phoneNumberI, genderI, bloodTypeI,
        emergencyContactI, hospitalName);
    medicalRecords[nationalID] = address(newMedicalRecord);
    medicalRecordsCount++;
}
```

3.3.3 *Mark Transaction as Medical Error*

<table>
<tr><th>Step</th><th>Description / Code Snippet</th></tr>
<tr><td>Medical Record Surgeries Page

This is one of the pages that contain medical information that could be marked as a medical error. The user can click the “Mark As Medical Error” button on any surgery believed to be erroneous.

There is time limit to when this button will appear. After that limit. The information will be permanent in the patients record.</td><td> 
</td></tr>
<tr><td>Logic for marking a surgery as a medical error

Here, the function <code>markAsMedicalError</code> takes one argument, id. Then it calls the smart contract function <code>markTransactionAsMedicalError</code> and passes the id to that function. When the request is finished, the system then updates the state of the component and the UI to show that the surgery has been marked as a medical error.</td><td><pre>
markAsMedicalError = async (id) => {
  const accounts = await web3.eth.getAccounts();
  const medicalRecordID = this.props.match.params.id;
  const medicalRecordAddress = await contract.methods.getMedicalRecord(medicalRecordID).call()
  let medicalRecordContract = await new web3.eth.Contract(
     medicalRecordABI,
     medicalRecordAddress
    );

  // mark for reusability
  await medicalRecordContract.methods.markTransactionAsMedicalError(1, id).send({ from: accounts[0], gas: '200000000' })
  .then(async() => {
    let cloned = cloneDeep(this.state.transactions);
    for(let i = 0; i < cloned.length; i++){
      if (cloned[i].id == id) {
        if (cloned[i].isCorrectionFor !== '') {
          // mark for reusability
          // this is a hack for the transactions
          // that are a correction for other trasaction ,
          // so you have to mark the corrected transactions
          // as medical erros as well
          await medicalRecordContract.methods.markTransactionAsMedicalError(1, cloned[i].isCorrectionFor)
            .send({ from: accounts[0], gas: '200000000' })
        }
        cloned[i].isCorrectionFor = 'true';
      }
    }
    this.setState({transactions: cloned});
    // mark for reusability
    toast.success("Surgery has been set as a medical error.", {
      position: "bottom-center",
      autoClose: 3000,
      hideProgressBar: false,
      closeOnClick: true,
      pauseOnHover: true,
      draggable: true,
      width: 200,
    });
  })
  .catch(() => {
    toast.error("Error : Something worng happened", {
      position: "bottom-center",
      autoClose: 3000,
      hideProgressBar: false,
      closeOnClick: true,
      pauseOnHover: true,
      draggable: true,
      width: 200,
    });
  })
}
</pre></td></tr>
</table>

`markTransactionAsMedicalError` smart contract function. This function verifies the transaction type (in this case, the surgery type) and changes the `isCorrectionFor` parameter to `true`. The `isCorrectionFor` parameter can be either `true` or an address.
If it's `true`, then it is considered a medical error without referring to another corrective transaction.
If it's an address, then the transaction is considered a correction for a past transaction and that transaction will have it's `isCorrectionFor` value set to `true`.

```
function markTransactionAsMedicalError(uint _type, uint _id) public {
    if (_type == 1) { // Surgery
        for ( uint i = 0 ; i < surgeries.length ; i++) {
            if (surgeries[i].id == _id) {
                surgeries[i].isCorrectionFor = "true";
            }
        }
    } else if (_type == 2) { // LaboratoryTest
        for ( uint j = 0 ; j < laboratoryTests.length ; j++) {
            if (laboratoryTests[j].id == _id) {
                laboratoryTests[j].isCorrectionFor = "true";
            }
        }
    } else if (_type == 3) { // Diognosis
        for ( uint n = 0 ; n < diagnosises.length ; n++) {
            if (diagnosises[n].id == _id) {
                diagnosises[n].isCorrectionFor = "true";
            }
        }
    } else if (_type == 4) { // BloodDonation
        for ( uint m = 0 ; m < bloodDonations.length ; m++) {
            if (bloodDonations[m].id == _id) {
                bloodDonations[m].isCorrectionFor = "true";
            }
        }
    } else if (_type == 5){ // DrugPrescribtion
        for ( uint y = 0 ; y < drugPrescribtions.length ; y++) {
            if (drugPrescribtions[y].id == _id) {
                drugPrescribtions[y].isCorrectionFor = "true";
            }
        }
    } else if (_type == 6){
        for ( uint o= 0 ; o < radiologies.length ; o++) {
            if (radiologies[y].id == _id) {
                radiologies[y].isCorrectionFor = "true";
            }
        }
    }
}
```

4. Testing

In the testing phase, the Mocha framework was used to test the smart contract functions. The Mocha framework runs on the Node.js JavaScript runtime environment, which allows running JavaScript programs outside of a web browser.

4.1 Expected Test Scenarios

1. Deploying a `MedicalRecordsSystem.sol` smart contract.
2. Add hospital using the Ministry of Health account.
3. Add Pharmacy using the Ministry of Health account.
4. Create a medical record using a Hospital account.
5. Counting the number of hospitals, pharmacies, and medical records correctly.
6. Getting data of a medical record.
7. Check if a medical record is deployed and has an address.
8. Check if the data in the medical record are saved correctly.
9. Add a new surgery to a medical record and save its information correctly.
10. Add a new laboratory test to a medical record and save its information correctly.
11. Add a new diagnosis to a medical record and save its information correctly.
12. Add a new blood donation to a medical record and save its information correctly.
13. Add a new drug prescription to a medical record and save its information correctly.
14. Add a new radiology scan to a medical record and save its information correctly.
15. Mark a drug prescription as dispensed.
16. Mark a transaction as a medical error.

4.2 Unit Test

Bellow are sample unit test from the file `MedicalRecordsSystem.test.js` and `MedicalRecord.test.js`. The complete 16 tests are available in the files.

Table 6 Unit Tests

Function	Inputs	Expected Outputs	Resulting Outputs	Pass/Fail
`addHospital`	`address hospitalAddressI, string memory hospitalName`	`address` becomes `true`	As expected	Pass
`addPharmacy`	`address pharmacyAddressI, string memory pharmacyName`	`address` becomes `true`	As expected	Pass
`createMedicalRecord`	`uint256 nationalID, string memory name, uint dateI, string memory phoneNumberI, string memory genderI, string memory bloodTypeI, string memory emergencyContactI`	calling `checkMedicalRecord` and passing the national id returns `true`.	As expected	Pass

Below is a screenshot of the result of running the Mocha test for all the 16 tests:

Figure 13 Result of running the Mocha test

4.3 Functional Test

In this section, different functions of the system are tested based on scenario testing:

Function to be tested: Add hospital

ID	Scenario	Expected Output	Preconditions	Actual Output	Test Status
1	Open the Ministry of Health using the authorized address	The "allowed" tag should be displayed in the dashboard	--	As Expected	Pass
2	Click the "Hospitals" button	The user is presented with different hospitals already added and a form to add a new hospital	--	As Expected	Pass
3	Fill the form and click the "Add Hospital" button	The user is presented with a notification that the hospital has been successfully added and the new hospital is added to the list	The hospital is not already added	As Expected	Pass

Function to be tested: Create Medical Record

ID	Scenario	Expected Output	Preconditions	Actual Output	Test Status
1	Login to the hospital system with the authorized account (receptionist)	The login is successful and the correct permissions are displayed in the side bar of the home screen	--	As Expected	Pass
2	Click the "Create" button	The user is presented with the form to create a new medical record	--	As Expected	Pass
3	Fill the form and click the "Create" button	The user is presented with a notification that the medical record has been successfully created and the user is redirected to the page of that medical record	There is no medical record already created with the same national ID	As Expected	Pass

Function to be tested: Mark Transaction as Medical Error

ID	Scenario	Expected Output	Preconditions	Actual Output	Test Status
1	Login to the hospital system with the authorized account	The login is successful and the correct permissions are displayed in the side bar of the home screen	--	As Expected	Pass
2	Enter the patient's national ID and click the "Open" button	The user is redirected to the medical record page of that national ID	The medical record is already created	As Expected	Pass
3	Click one of the types	the user is redirected to the page containing all the transactions	The transactions are already created	As Expected	Pass
4	Click the "Mark as Medical Error" button	The transaction is labeled with the "Medical Error" label and the "Mark as Medical Error" button disappears	The transaction is not more than 30 minutes old. Otherwise the "Mark as Medical Error" button will not be visible	As Expected	Pass

4.4 Usability Test

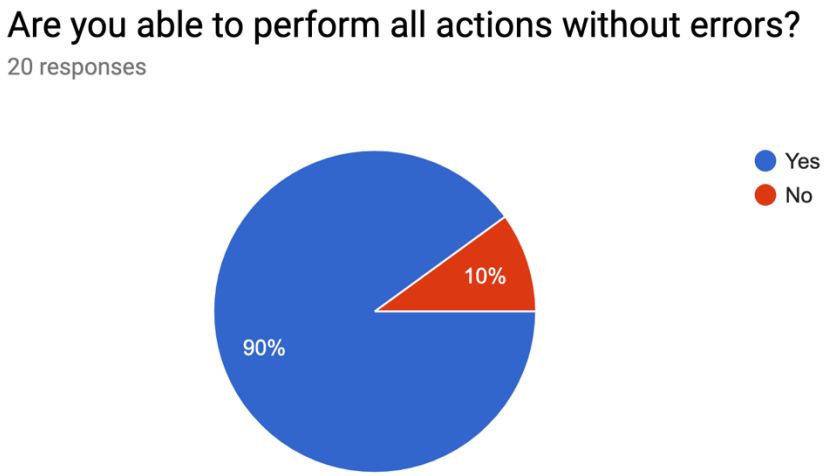
Since the system target's the health sector, measuring the usability requires deploying the system in a real production environment where the Ministry of Health along with multiple hospitals and pharmacies are actively interacting with it. In this graduation project it was not possible to release the system to production and test the usability of the system from the end-user point of view.

However, the prototype web apps (Ministry of Health, Hospital, Pharmacy, and Patient) that were created for this project have been demonstrated to a select number of students and faculty members in the College of Computer Sciences and Information to get an idea of the level of usability of this prototype.

The demo targeted 15 students and 5 faculty members and concluded with the following points:

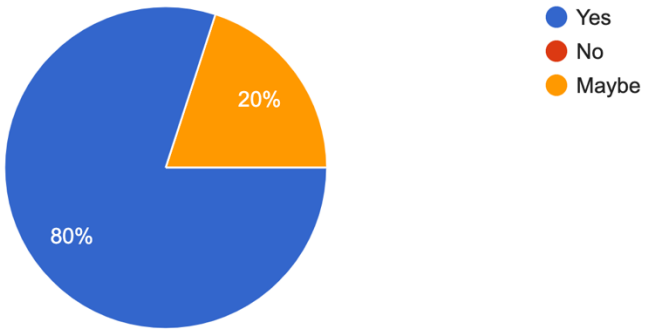
- The system operated as expected and all transactions behaved normally.
- The blockchain network along with its smart contracts can be interfaced with different stakeholder web apps. E.g. Ministry of Health, hospitals, pharmacies, but the level of compatibility with different web application technologies is not clear.

Charts from the survey:



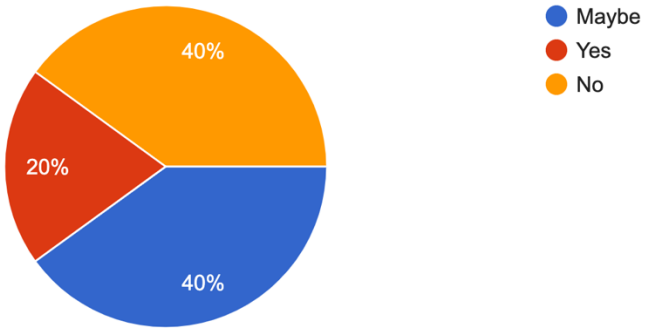
Do you think it's possible to connect this system with more hospitals and pharmacies?

20 responses



Do you know exactly how different hospitals and pharmacies will be able to interface with the system?

20 responses



5. Deployment of the System

The following diagram shows our suggestion of how the final deployment of the system will be after the development is done.

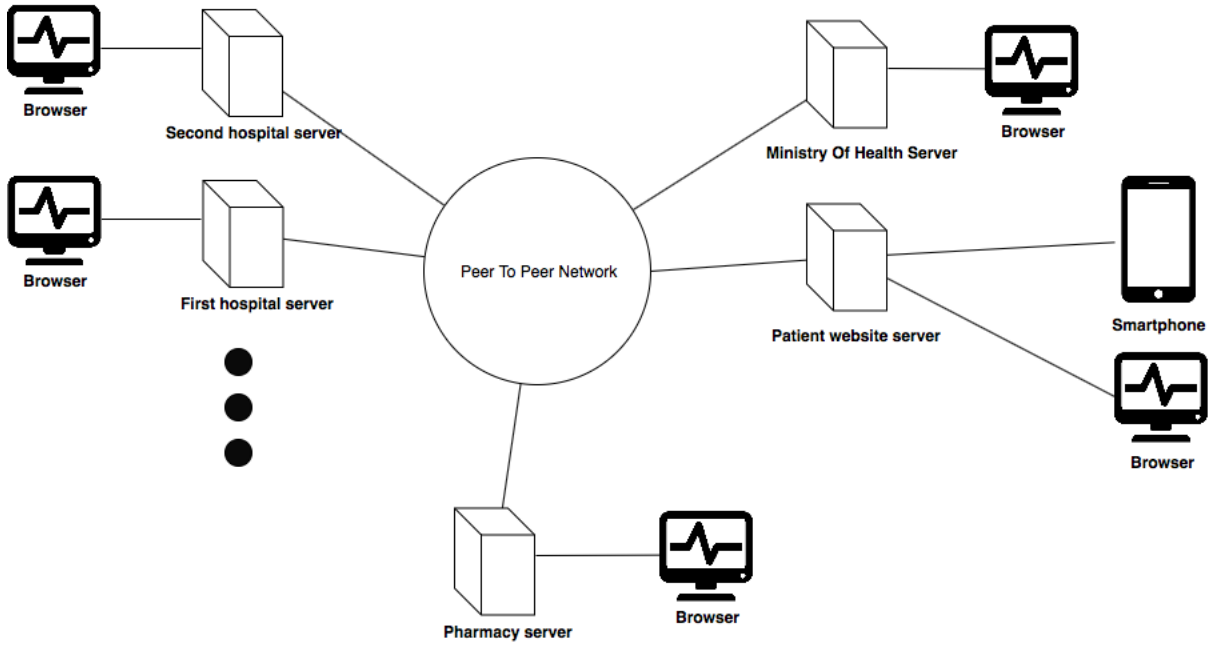


Figure 14 System Deployment

6. Limitation of the System

Because of the concept of currency in Ethereum decentralized apps, each account will have to spend virtual money known as Ether with each transaction (request to the network). We could not remove the concept of currency, but to work around it, the private network was configured so that each account created will have a very large balance and each transaction is cheap (nearly free), allowing each account to preform millions of transactions and the system running for hundreds of years without worrying about the currency issue.

In addition, the current version of Solidity does not allow for a function to return 8 or more variables or have one function to contain 8 or more arguments. This limited the number properties a model can have. When 8 or more variables are used, the compiler throws an error with the message "Stack too deep, try removing local variables."

Finally, it was not possible to store media attachments (radiology scans, images… etc.) directly in the Ethereum blockchain because it would significantly affect the performance of the blockchain and would cost about 640,000 Gas of the virtual currency to request the attachment and display it to the user. As a workaround for this issue, a centralized secure database was used to store the images and the hash of the image was then stored in the blockchain.

7. Conclusion and Future Work

In conclusion to this project, we have learned a lot about the Blockchain technology and the development of a decentralized system using Blockchain. The vision of the project was realized, but there is much to learn and improve. The idea of storing and managing medical records in the Ethereum blockchain insured that a patient's medical record was kept secure from any form of corruption. We saw how the system will help reduce the chances of losing a medical record or having its content altered. This was the goal and the inspiration that drove us to build this project. We hope that this document properly recorded all of the team members' efforts.

Future works are indeed needed to make this project a reality and release it in a production environment. Meetings with stakeholders would help improve the requirements and adjust the system to meet the needs of different hospitals and pharmacies. Further research would help bypass many of the limitations we faced during the development of the system. Migrating the existing physical and digital medical records to the system and deploying the system is also a major step that need to be done.

8. Reference

1. Electronic medical records software
    - https://www.softwareadvice.com/medical/electronic-medical-record-software-comparison/ ; Accessed on 2 October, 2018
2. Blockchain and healthcare
    - https://medium.com/swlh/how-blockchain-technology-can-transform-the-healthcare-sector-604d1bcdc16b ; Accessed on 2 October, 2018
3. Software quality factors
    - http://fac.ksu.edu.sa/sites/default/files/2.software_quality_factors_0.pptx ; Accessed on 15 October, 2018
4. Ethereum blockchain course
    - https://drive.google.com/open?id=164l7E236Bhaq7w3rnjGziK4SBE2p4x8c ; Accessed on 10 October, 2018
5. Ethash algorithm
    - https://github.com/ethereum/wiki/wiki/Ethash ; Accessed on 25 November, 2018
6. SHA256 algorithm
    - https://en.wikipedia.org/wiki/SHA-2 ; Accessed on 25 November, 2018
7. A closer look at Ethereum signatures
    - https://hackernoon.com/a-closer-look-at-ethereum-signatures-5784c14abecc ; Accessed on 20 November, 2018
8. Universally Unique Identifiers
    - https://en.wikipedia.org/wiki/Universally_unique_identifier ; Accessed on 18 November, 2018
9. React
    - https://reactjs.org/ ; Accessed on 10 April, 2019
10. Angular
    - https://angular.io/ ; Accessed on 10 April, 2019
11. Ethereum platform
    - https://www.ethereum.org/ ; Accessed on 10 April, 2019
12. Solidity
    - https://github.com/ethereum/solidity ; Accessed on 10 April, 2019
13. Cloud9
    - https://aws.amazon.com/cloud9/ ; Accessed on 10 April, 2019
14. Remix IDE
    - https://github.com/ethereum/remix-ide ; Accessed on 10 April, 2019
15. Ganache
    - https://trufflframework.com/ganache ; Accessed on 10 April, 2019
16. MetaMask
    - https://metamask.io/ ; Accessed on 10 April, 2019