

**Towards partial fulfillment for Undergraduate Degree Level Programme  
Bachelor of Technology in Computer Engineering**

*A Project Report on:*

**Management and Security of EHRs using blockchain**

---

Prepared by :

Admission No.

Student Name

U16CO063

Atharva Kalsekar

---

U16CO070

Avinash Jaiswal

---

U16CO080

Rogin Koshy

---

U16CO097

Sameer Mandloi

---

Class : B.TECH. IV (Computer Engineering) 8<sup>th</sup> Semester

Year : 2019-2020

Guided by : Dr. Bhavesh. N. Gohil

---



**DEPARTMENT OF COMPUTER ENGINEERING  
SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY,  
SURAT - 395 007 (GUJARAT, INDIA)**

# ***Student Declaration***

This is to certify that the work described in this project report has been actually carried out and implemented by our project team consisting of

<b>Sr.</b>	<b>Admission No.</b>	<b>Student Name</b>
1.	U16CO063	Atharva Kalsekar
2.	U16CO070	Avinash Jaiswal
3.	U16CO080	Rogin Koshy
4.	U16CO097	Sameer Mandloi

Neither the source code there in, nor the content of the project report have been copied or downloaded from any other source. We understand that our result grades would be revoked if later it is found to be so.

## **Signature of the Students:**

<b>Sr.</b>	<b>Student Name</b>	<b>Signature of the Student</b>
1.	Atharva Kalsekar	
2.	Avinash Jaiswal	
3.	Rogin Koshy	
4.	Sameer Mandloi	

# Certificate

*This is to certify that the project report entitled* Management and Security of EHRs using  
blockchain *is prepared and presented by*

Sr.	Admission No.	Student Name
1.	U16CO063	Atharva Kalsekar
2.	U16CO070	Avinash Jaiswal
3.	U16CO080	Rogin Koshy
4.	U16CO097	Sameer Mandloi

*Final Year of Computer Engineering and their work is satisfactory.*

---

---

SIGNATURE :

GUIDE

JURY

HEAD OF DEPT.

# Abstract

*With the increasing amount of data in the medical field, soon it will be difficult to handle all the past and present medical data like health records of the patients and other medical information given by the doctor. Blockchain is a technology which can be used to solve this problem in a very organized and reliable way. Blockchain is essentially a distributed append-only ledger with a certification authority and has permissioned access. Trust and transparency can be achieved between patients and doctors to easily collaborate with each other using the electronic health records deployed over the chain.*

*Blockchain technology is considered to be the driving force of the next fundamental revolution in information technology. It can be widely used for required implementation and also in a specific application domain. The report presents the insights and practical experience on Blockchain technology and its applications over medical fields using electronic health record(EHR) , as well as theory based exploration of possible cases possible using an EHR.*

**Keywords:** Blockchain - Electronic health records - Hyperledger Blockchain EHR - Blockchain Healthcare.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Application . . . . .	2
1.2	Motivation . . . . .	2
1.3	Objective . . . . .	2
1.4	Contribution . . . . .	3
1.5	Organization of project report . . . . .	3
<b>2</b>	<b>Background Study</b>	<b>4</b>
2.1	EHR . . . . .	4
2.2	Blockchain . . . . .	5
2.2.1	Blockchain categorization . . . . .	5
2.3	Cardiology . . . . .	6
2.4	Medical Council of India . . . . .	7
<b>3</b>	<b>Working and Implementation</b>	<b>8</b>
3.1	Entities . . . . .	8
3.1.1	EHR structure . . . . .	9
3.2	Architecture . . . . .	10
3.3	Working Methodology of the System . . . . .	10
3.3.1	Onboarding . . . . .	10
3.3.2	Uploading the data . . . . .	11
3.3.3	Data retrieval . . . . .	14
3.4	Implementation . . . . .	14
3.4.1	Configuration . . . . .	15
3.4.2	User Interface . . . . .	21
3.4.3	Gantt Chart . . . . .	26
<b>4</b>	<b>Conclusion and Future Work</b>	<b>28</b>
<b>5</b>	<b>Acknowledgement</b>	<b>30</b>
<b>Appendices</b>		
<b>Appendix A Minimum requirements</b>		<b>1</b>
<b>Appendix B Prerequisites</b>		<b>1</b>
B.1	Installing node . . . . .	1
B.2	Installing angular . . . . .	2
B.3	Installing docker . . . . .	2
B.4	Installing VSCode and its extensions . . . . .	3

Appendix C	Installing project modules	4
Appendix D	Running the application	4

# List of Figures

2.1	A typical blockchain . . . . .	5
3.1	Architecture . . . . .	10
3.2	Doctor onboarding process . . . . .	11
3.3	Data update process . . . . .	12
3.4	Patient Interaction with doctor . . . . .	13
3.5	Doctor home page : same patient . . . . .	21
3.6	Patient consent page . . . . .	22
3.7	Diagnosis form . . . . .	22
3.8	Doctor home page : onboarding . . . . .	23
3.9	Patient onboarding form . . . . .	23
3.10	Doctor home page : history . . . . .	24
3.11	Patient consent form . . . . .	24
3.12	Patient history viewed by doctor . . . . .	25
3.13	Patient login . . . . .	25
3.14	Patient history . . . . .	26
3.15	Patient's current doctors . . . . .	26
3.16	Gantt Chart . . . . .	27
B.2.1	Angular version output . . . . .	2
B.4.1	The docker extension . . . . .	3
B.4.2	The IBM block chain platform extension . . . . .	3
D.1	Package open project . . . . .	4
D.2	Exporting wallet . . . . .	5
D.3	Installing the package . . . . .	5
D.4	Instantiate the package . . . . .	5

# List of Tables

3.1	EHR structure . . . . .	9
-----	-------------------------	---



---

# 1 Introduction

---

Developing countries are carrying the burden of the most deadly chronic diseases , including cancer, HIV / AIDs and diabetes. These countries are under pressure to provide superior healthcare services amid a shortage of qualified healthcare professionals. As a key component of medical informatics the Electronic Health Record ( EHR) symbolizes potential solutions for enhanced healthcare. The EHR is a digital version of the paper chart of a patient. The EHR structures hold vital and highly confidential private details for healthcare diagnosis and treatment, often spread and exchanged among peers such as healthcare professionals, insurance agencies, hospitals, academics, patient families, among others.

In the health care context, EHR systems requires to work jointly within and across organizational boundaries in order to advance the effective delivery of health care for individuals and societies and also enable cost savings, and efficiencies . Also, lack of integration makes it difficult to maintain the privacy and security of patient's confidential information. Security of the EHR is a matter of concern especially where patient's data is transmitted over a network. Recently, blockchain technology has been advocated as a promising solution to solving EHR interoperability and security issues in developing countries.

Blockchain is a decentralized, trustless protocol that combines transparency, immutability, and consensus properties to enable secure, pseudo-anonymous transactions stored in a digital ledger. Systems built on the blockchain technology achieve secure distribution of entities amongst untrusted nodes. In health care industry, the blockchain technology has the potential to address the interoperability and security challenges currently present in EHR systems. The technology has the ability to provide technical architecture that enables individuals, health care providers, disparate entities and researchers to securely share electronic patient's data across multiple platforms.

In this project we propose a blockchain system for implementing and maintaining the EHRs.

## 1.1 Application

- This project provides transparency between patients and doctors and makes the patient in-charge of his own data. The age old problem of patients being restricted to a single doctor is solved and gives him the freedom to change doctors without the hassle of maintaining a record of all the previous visits.
- The doctors get a summary of the patient and don't have to manually go through the entire patient history.
- Since the whole record is maintained on the blockchain, it is tamper-proof and trustworthy.

## 1.2 Motivation

The healthcare industry is known to have fallen behind other sectors in the usage and deployment of emerging digital technology, such as the finance sector. Manual processes account for a significant part of the processes. These systems are suffering from a lack of data ownership, poor data quality, poor data security and backup procedures and therefore rarely used for decision making. This also presents difficulties when documenting what is currently occurring in health care to improve epidemic monitoring, preparation, clinician and strategic decision-making. The lack of data integration makes it difficult to offer 360° of patient health history, and to share information between different entities. Interoperability in data health is an issue which remains open until now. The main question is how to give open access to sensitive data (health data), preserve privacy, anonymity and avoid misuse of data.

These existing scenario motivates us to design a system that can make these health data transactions smooth, secure and invulnerable to tampering and interoperable among organizations having least trust among them, however, they share some common assets like patients.

## 1.3 Objective

This project primarily aims to develop a blockchain system that implements and maintains electronic health records among various participating medical organizations and patients. This system would provide smooth inter-operability and transfer of patient health records among the peer medical institutions whenever a patient might want to change his/her ongoing consultancy from one medical practitioner to another. The secondary objective of the project would be to train a deep learning model to generate a summary of health records from the retrieved data of patient so that the medical practitioner would get a quick overview about the patient's history. This system would be

made interactable with participants of the network through a web application.

## **1.4 Contribution**

The project aims to develop a blockchain system for the ecosystem of doctors and patients to maintain the EHRs and manage their update and retrieval. The entire system is ensured by reliability of the blockchain consensus protocol. The EHRs will be readily available for easy access for patients and doctors so that they can refer these on demand. Our project lays a foundation for further research for leveraging this to a greater level. The implementation has been currently deployed on hyperledger composer playground with working transactions.

## **1.5 Organization of project report**

Chapter 1 presents a brief introduction of the project, its application in the real world, the motivation behind choosing this topic, its objectives and our contribution towards this topic.

Chapter 2 presents the background study which gives the overview of the technology we are using namely blockchain and EHRs and gives an overview about some of its concepts relevant to our project.

Chapter 3 presents the implementation details and the tech stacks which we are utilizing to implement the project.

Chapter 4 presents our conclusions drawn to possible future works that can be built upon the proposed system.

---

## 2 Background Study

---

Here we present the overview of technologies involved and some concepts that are essential for getting a good idea about their utilization in this project. This section has been divided in the following subsections : 2.1 Presents introduction to the EHRs and their essential requirements. Subsection 2.2 explains the blockchains in a birds eye view and also important components of it which are necessary to understand their role in this project.

### 2.1 EHR

An Electronic Health Record (EHR) is an electronic version of a patients medical history, that is maintained by the provider over time, and may include all of the key administrative clinical data relevant to that persons care under a particular provider, including demographics, progress notes, problems, medications, vital signs, past medical history, immunizations, laboratory data and radiology reports. The EHR automates knowledge exposure and has the ability to streamline workflow for the clinician. The EHR also has the ability to directly or indirectly support other care-related activities through various interfaces including support for evidence-based decision making, quality management , and reporting of outcomes. EHRs are the further part in continued healthcare progress, which can strengthen the relationship between patients and clinicians [1]. The data, as well as its timeliness and availability, will allow providers to make better choices and give better care. For example, the EHR can upgrade patient care by:

- Reducing the occurrence of medical mistakes and keeping patient reports more precise and transparent.
- Making the health information accessible, reducing test duplication, reducing treatment delays and informing patients to make better decisions.
- Reducing medical error by making medical records more accurate and concise.

## 2.2 Blockchain

Blockchains systems are tamper-proof, seldom cryptographic ledgers deployed in a decentralized way ( i.e., without a central repository) and typically without any central authority ( i.e. a bank, corporation, or government). On an intuitive level, they enable a group of users to record transactions in a shared ledger within that group, such that under normal circumstances of operation of the blockchain network no transaction can be modified or changed once published to the system. A typical block chain can be thought of as shown in **Figure 2.1**.

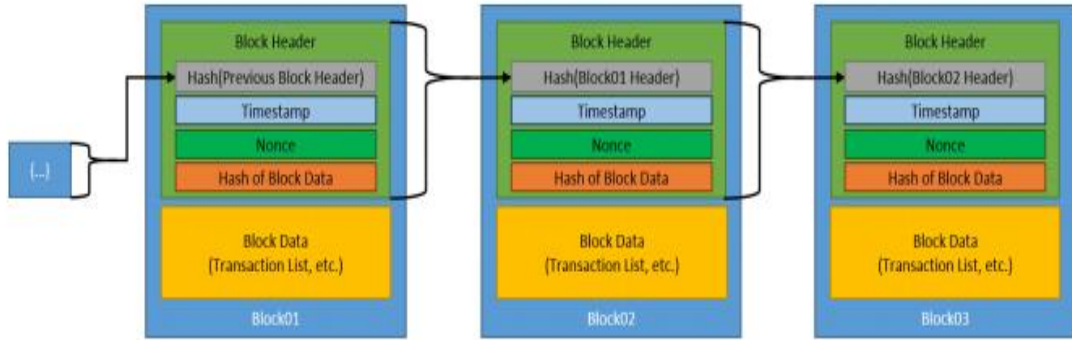


Figure 2.1: A typical blockchain [2]

The technology of blockchain lays the foundation of modern world crypto-currencies, such terminology employed because of the heavy usage of cryptographic functions. For cryptocurrency-based blockchain networks that use mining, users can solve puzzles using cryptographic hash functions in the enticement of getting fixed amount of the cryptocurrency reward. However, blockchain technology is more broadly applicable beyond mere cryptocurrencies. In this work, we focus on the Health Care use case, since this field has received much attention by various researchers around the globe today.

### 2.2.1 Blockchain categorization

Generally based on permission model the blockchain networking systems are categorized, which determines who can look after, cater and modify them (e.g., publish blocks). If anyone can publish a new block, it is permission-less. While in permissioned blockchain only specific authorized users can publish blocks. In simple terms, a permissioned blockchain network is like a corporate intranet that is in some manner controlled, while a permissionless blockchain network is open to all like the public internet, where anyone can participate.

### **A. Permission-less**

1. Being open to all, permission-less networks of blockchain allows any user to publish blocks on to it, without any authorization requirement.
2. These are generally open source software which can be simply acquired by downloading.
3. As anyone is allowed to publish blocks on such a network, it also enables them to read the ledger as well as issue any transactions on it.
4. As these permission-less network of blockchains are open to all, malicious activities may pose a threat to such systems.

### **B. Permissioned**

1. Unlike permission-less blockchains, permissioned blockchain networks, as the name suggests, are those where users who publish blocks to the network must be authorized.
2. It is possible to control read access and who can issue transactions over the network because some authorized group of users are maintaining the blockchain.
3. Open source as well as closed source software can be used to maintain these permissioned blockchains.
4. These networks employ consensus models for maintenance. The group of users maintaining the blockchain have a level of trust with each other which is a result of the confirmation of user's identity required to participate as a member of the network. In case of any misbehaviour or malicious activity, that users authority can be revoked.

Due to the above mentioned features we have decided to use permissioned network of peers in this project.

## **2.3 Cardiology**

Cardiology is the study and treatment of disorders of the heart and the blood vessels. A person with heart disease or cardiovascular disease may be referred to a cardiologist. In this project, we have laid our focus to cardiologist in particular and mentioned the various factors required for a doctor to provide a basic diagnosis. Since we only have one field which we are focusing on and all the doctors are of the same field, all the doctors can view the records added by all other doctors. However, the future scope of this project can be to have doctors from different fields and the doctor of one field can only access the records of the doctors of the same field while the other fields cannot be accessed.

We have used various fields in particular to cardiology for this project. This can be varied according to the field the doctor tends to. The doctor takes inputs like RBC, WBC, heartbeat count etc for the diagnosis. This is explained further in the coming sections.

## **2.4 Medical Council of India**

The Medical Council of India is a statutory body for establishing uniform and high standards of medical education in India till formation of National Medical Commission From 14 October 2019. Here we can use the medical council of India as the certification authority which can be used to verify the doctors. Every doctor has a Doctor Registration number by which he/she can be identified. The authenticity of the doctor can be identified using this number after which the doctor can be onboarded. This is kept as a future work. However, the doctor ID can be corresponded to the Doctor Registration number which validates the doctor.

---

## 3 Working and Implementation

---

In this project we aim to build a system that brings health organizations that have weak trust among them to participate in a peer to peer blockchain network to maintain the data of their patients and share their reports [1]. The blockchain network will consist of doctors or health organizations as peers. This system will be available to them as a web app. The web app will have different variants for doctors and patients. The doctors can update as well as read the data while the patients can only read it. The interactions with the system involved are : onboarding process of a medical practitioner, uploading the new transaction of a patient by the doctor and retrieval of medical reports of patients record. These are described below in the respective order.

### 3.1 Entities

The architecture consists of certain terms that needs to be introduced before we move on to the exact implementation details. These are:-

*Certification Authority* - The certification authority is responsible for handling all the logic of access control, issuing the users identities and permission in the blockchain network of Hyperledger.

*Orderer* -The orderer is used to keep the whole network in a synchronised state. Whenever a new transaction is to be made, the orderer is the one who informs all peers of the transaction. A network can have multiple orderers, it's also advised to keep fewer faults.

*Peers* -Only peers in the business network are allowed to commit transactions. Also each peer has its own copy of the entire world state. It is linked to instances of CouchDB which serve as the database.

*Block contents*-

- *@PatientID*- The id of the patient being queried.
- *@ehrID*- The unique ID generated for the patients diagnosis EHR.
- *@doctorID*- The id of the doctor currently accessing the EHR
- *@lastModifiedTime* - The time of current transaction
- *@doctorName* - The name of the doctor currently accessing the EHR



### 3.1.1 EHR structure

EHR	Sub-Fields: Datatype
Symptoms	Fainting: Boolean {Yes/No} Heartbeat: int Chest Tightness: Boolean {Yes/No} Chest Pain: Boolean {Yes/No} Swelling: Checkbox -legs -feet -ankles -abdomen Weight: float
Any other problem	Text Area
Patient Feedback	Text Area
Blood Test	WBC: int RBC: int Total cholesterol: float LDL: float HDL: float Triglycerides: int
util	Next appointment date: date Fees: int Payment method: string
Prescribed Medicines	List[{obj}] -> obj {medicine: string, dosage: string}

Table 3.1: EHR structure

Where,

- LDL- Low Density Lipoprotein.
- HDL- High Density Lipoprotein.
- WBC- White Blood Count.
- RBC- Red Blood Count.

## 3.2 Architecture

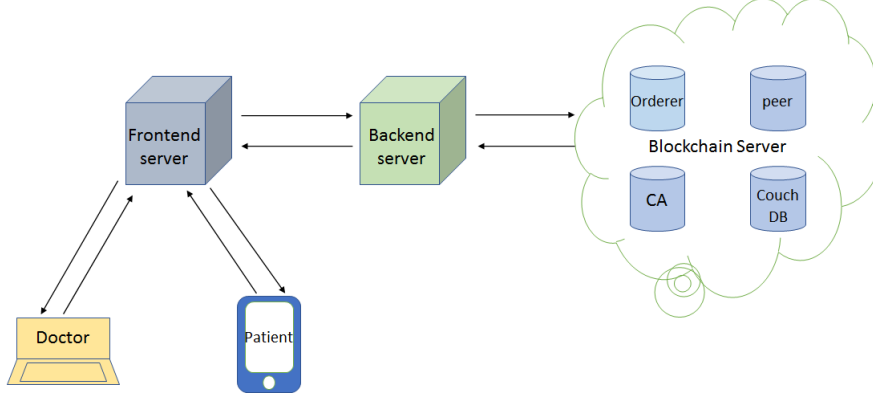


Figure 3.1: Architecture

The **Figure 3.1** describes the following architectural setup of our system. Following are the interactions among them:

1. The patient or the doctor devices interact with the front end servers for authentication purposes and logging in to the system.
2. The front end server interacts with the backend for retrieving patient's details or updating them and conveying transaction alerts to the user.
3. The backend server is responsible for managing the EHRs, validating the update and fetch request and generating verification codes.
4. The block chain server hosts the blockchain and the peer network.

## 3.3 Working Methodology of the System

The system consists of doctors and patients as its peers. The doctor is more privileged than a patient. Both the peers interact with the ledger through an interactive web application. The web application has different interfaces for doctor and patients. The following subsections explain in a step by step manner all the possible interactions with the ledger.

### 3.3.1 Onboarding

The onboarding process will be quite different in case of both the participants. The doctor onboarding can be initiated and completed by the doctor him/her-self but for the patient onboarding, the patient has to visit a doctor at least once in the beginning.

The Doctor onboarding process is as follows:

- (1) Doctor opens the application and fills his details like his name, contact and medical credentials i.e. the registration number provided by Medical Council of India (MCI),

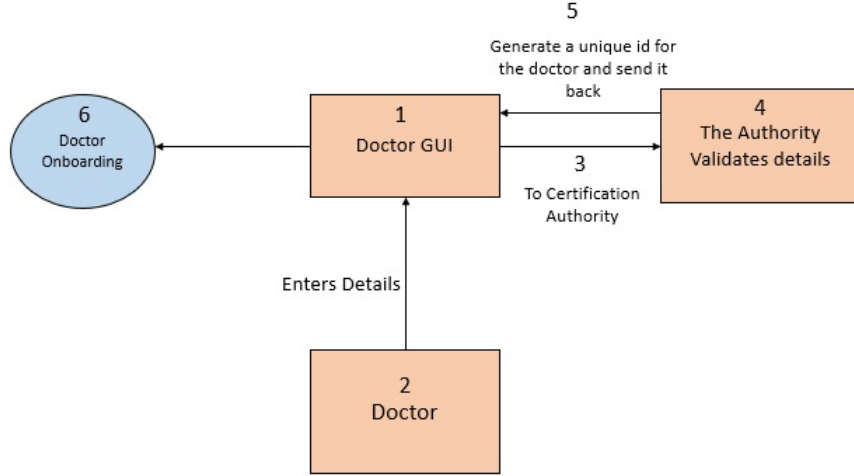


Figure 3.2: Doctor onboarding process

which serves the purpose of authentication and also the doctor as to provide a unique username. Then an onboarding request is sent to the certification authority.

- (2) On receiving an onboarding request the central authority verifies the doctor's credentials and sends appropriate message back to the doctor depending on the status of the onboarding request process.
- (3) The doctor notes this down and this completes the onboarding process as shown in **Figure 3.2**

The Patient onboarding process is as follows:

- (1) The patient visits a doctor who is already registered in the system.
- (2) The doctor then feeds in the patient's email ID and username which henceforth will serve as the patients ID, the uniqueness can be ensured by the doctor using the prompts in the UI.
- (3) This successfully onboards the patient.

### 3.3.2 Uploading the data

Updation or creation of a new record can only be done by a doctor which can be visualized from **Figure 3.3**.

- (1) Doctor logs into his home page.
- (2) Doctor has to chose between new patient and old patient to proceed to its diagnosis.

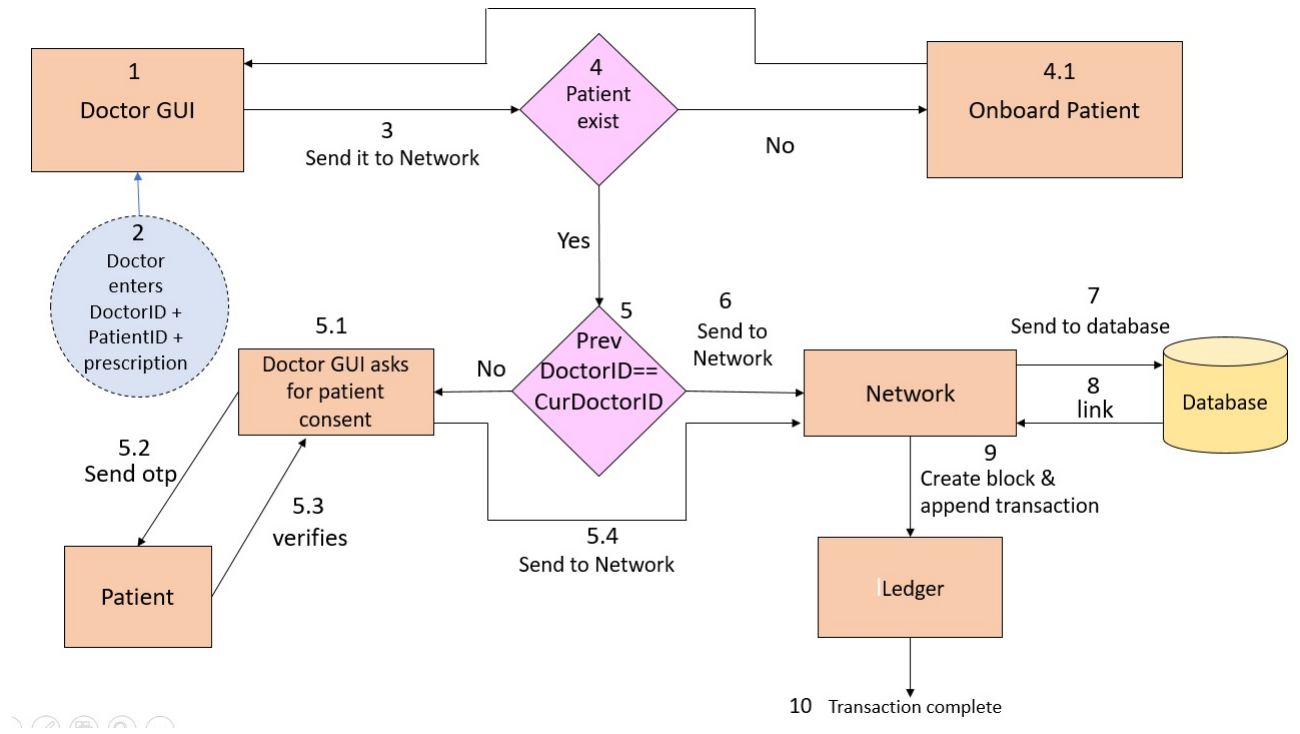


Figure 3.3: Data update process

- (3) Then depending on the selected patient type, an OTP or a verification code is sent to the onboarded patients registered email address.

*NOTE :* If the patient is not onboarded then appropriate message is displayed to the doctor.

- (4) The patient then tells this OTP to the doctor. This authentication process shows that the patient and the concerned doctor are in contact which ensures that the doctor is authorized now to update the data of the patient.
- (5) The doctor then is presented with the diagnosis form and after filling it up appropriately, the record can be submitted
- (6) If the record is submitted successfully then message is prompted to the doctor accordingly.

Here various cases are involved like:

Case 1: Patient visits doctor.

Case 1.1: Patient visiting doctor for the very first time.

If the patient is visiting the doctor for the very first time, the patient has to be onboarded first. For the onboarding, the details of the patient like a unique username, First Name, Last Name and the Email Id of the patient needs to be

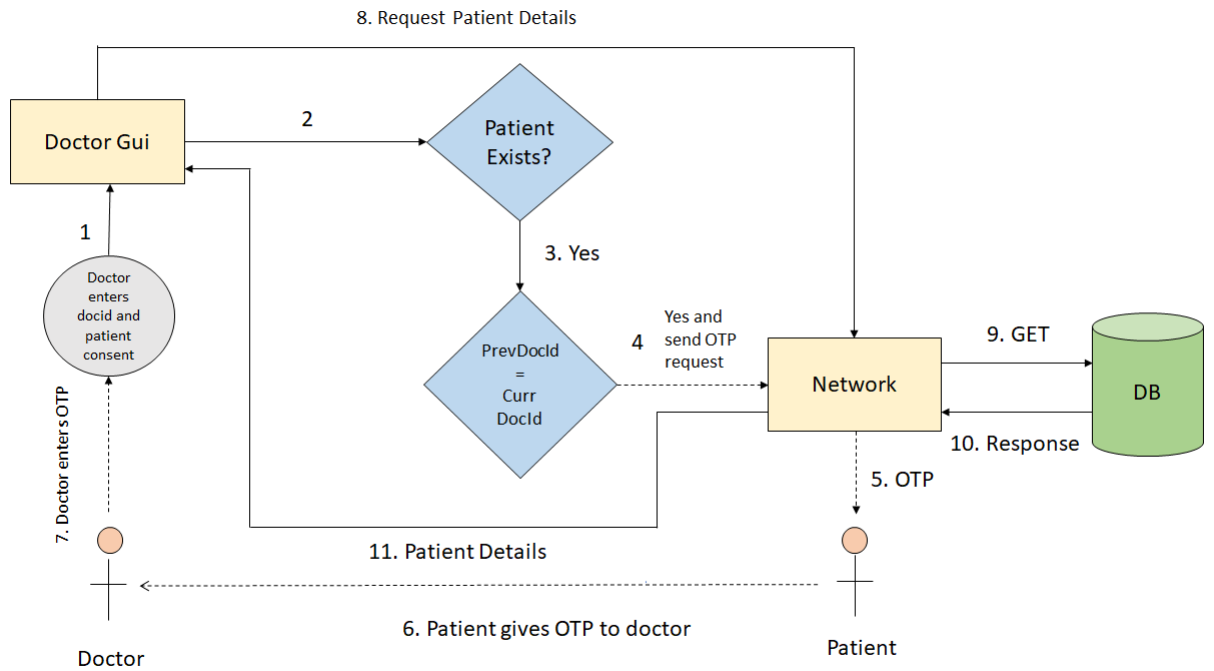


Figure 3.4: Patient Interaction with doctor

added. For further access to the patient's profile, the patient needs to enter the otp sent to his/her email id after which a password can be set for future login.

Case 1.2: Patient has already visited the doctor.

In this case there are two possibilities which are, the patient was assigned to the current doctor itself or the patient was previously under any other doctor. In both the cases the patient has to provide the consent (in the form of OTP) for the doctor to add any diagnosis or even view the history of the patient.

Case 2: A onboarded patient to a new doctor with whom he has no history but was getting treatment from other doctor previously.

In this case,

- (1) The doctor ask for patients consent and then the doctor is asked by the web application to enter verification code which would be sent to the concerned patient.
- (2) The patient tells the verification code to the doctor.
- (3) Then steps (4) to (7) are followed.
- (4) The doctor uploads his DoctorID, the PatientID and medical prescriptions to the web application, which then verifies some rules and send it to the peer network.

- (5) The system then stores the medical prescription data to a secure database which generates a link that maps to the location of this data and sends it back to the system.
- (6) The system generates a block and BlockID for the same.
- (7) This block as a transaction is then appended to the existing ledger. This completes the data updation process.

### 3.3.3 Data retrieval

The patient can read the data anytime just by clicking a button in patients GUI, but if a doctor wants to read a patient's data then again the verification code process is followed as described above.

## 3.4 Implementation

We are using hyperledger fabric for creating our permissioned blockchain. The present state of the EHRs do not ensure that the data will not be corrupted over time due to carelessness on the user's part and also do not track changes made to the EHR over time. To tackle these problems we propose introducing a permissioned blockchain system where all the transactions in the ledger will be append only and the entire history of any patient is saved in the world state once written.

The essential aim of our blockchain model is to track down the entire medical history of the patient being queried. We plan to achieve this by two approaches -:

1. Traversing the entire blockchain and extract those blocks that correspond to the queried patient ID.
2. Using a field that contains the reference to the previous blockchain. Using this we can traverse the entire chain in backward direction and gain access to all the relevant blocks and their data. Using this approach should be time efficient.

The present chain code is written in the composer playground. The main types of files in the folder are -:

- *Model files*- These files have their extension as .cto. These are used by hyperledger to generate the description of all the entities in the chain.
- *Chaincode* - These are script files written in javascript. These manage the logic of the transaction and contain the smart contracts of the chain.
- *Access Control file* - These files determine the accesses given to all the peers of the network. They have their extensions as .acl.

### 3.4.1 Configuration

The User Interface is the part of the application that any user will directly interact with. It handles some of the essential application features like hashing of the password and authorization while sharing the authentication mechanism partly with the backend. The backend is the realm that the user will never actually see. It contains a set of hosted APIs that processes data on request and return responses appropriately. The backend of this application consists of a node server that hosts logic related to data preprocessing and is the site of origination of requests to the blockchain server.

The blockchain server is an imaginary entity that in reality contains four different docker containers hosted at four different ports. These containers are responsible for maintaining the state of our Hyperledger fabric. Things like issuing a certificate to an entity, maintaining the order of execution of the transaction and operating through the ledger are all handled in these docker containers.

Let us now look at both of these servers in more detail and learn about how to set up the development environment and necessary system configuration.

- **The Primary NodeJs server**

The node server is the site that hosts API for our application. Along with the express framework to handle the routing and handling the HTTP server operations, we have used some other npm packages that handle the heavy lifting of activities like session management or mailing services. The list of packages that we use in the application are:

- (1) Fabric-contract-API - This is a package provided by the Hyperledger node SDK that is used for communication with the blockchain server.
- (2) Fabric-shim - The fabric shim is an abstraction that adds new javascript related functionality to the methods provided by the fabric-contract-API. It is a useful library provided along with the SDK that will help to interface the code in Nodejs easily.
- (3) Express - This is a framework for Nodejs built to handle the route and server management. It abstracts many of the Nodejs server related methods and can help you spin up a server using just 10 lines of code. Routing is also made easy using express as readily available endpoints and their handlers can be encapsulated in a single block of code, which otherwise could create havoc in heavy projects.
- (4) Body-Parser - This package is used to handle the post requests coming to the server. It helps in creating a JSON object of the body that is received in the API request. This is useful to work with data when the fields are deeply located using multi-level keying.

- (5) Cors - It is a package that attaches necessary cors headers to all the outgoing responses so that they are not blocked by the browser under the cors policy. It is a very useful utility when it comes to making good APIs.
- (6) Morgan - It is used for logging all the HTTP requests to the terminal. The logs include the type of request, their origin, the route requested and other request related fields. It helps in debugging your backend when the number of API requests exceeds more than a normal threshold.
- (7) Fs- fs is a Nodejs module that is used for all the file system related operations. It can help you read or write a file synchronously as well as asynchronously depending on the use case.
- (8) Uuid - This package is being used to generate random ids for the EHRs in our system. We are using a random number generation techniques that are provided by version 4 of this package.
- (9) Nodemailer - As the name suggests, nodemailer is being used to send emails from our node server for the patient OTPs.
- (10) Mongoose - It is an ORM over the popular NoSQL database, MongoDB. It returns promisified responses that can be used effectively in building scalable nodeJS applications.
- (11) JsonWebToken - This package is being used for generation of the random JWT token that will be used for session management with the frontend of angular.

Our primary backend server consists of two very important files i.e the app.js and the network.js. The app.js is the seat of all the router endpoints. We listen for all the requests primarily in the app.js. The network.js is used for communicating with the blockchain server. It contains methods abstracted from the fabric-contract-API which are then used for manipulating or querying the blockchain as per the use case.

- **The app.js:**

The app.js contains nearly 1k lines of code and presents a total of 19 different API endpoints for handling the requests from the frontend server. It is also the seat of database communication and the JSONwebtoken creation. Depending on the use case we have API endpoints hosted that can be used to manipulate data in the blockchain or query them. We have followed a common paradigm of handling the request from that frontend that includes:

- (1) Converting the body of the request into a JSON to easily drill down into the keys and get the values needed readily.
- (2) Connecting the blockchain using the identity of a proper entity which has a presence in the wallet. This way all our blockchain-related requests are authorized and valid under the name of the proper user.



- (3) Checking for the response on this connection request. If the connection is successful we have opened a channel for the user to manipulate or query the blockchain. If not, we return an appropriate error.
- (4) We invoke the proper method to interact with the blockchain depending on whether it is a submitting transaction or an evaluating transaction. A submitting transaction is the one that modified the ledger in the blockchain. An evaluation transaction is the one that is only concerned with querying the chain.
- (5) On the basis of the response received from the blockchain, we segregate the server responses. We follow a common guideline for all the responses from the server i.e we send a JSON object as a response that contains two keys depending on the type of response. The keys are:
  - (a) Action - It is a boolean depending on whether the response is a success or a failure. Failure cover cases of error while transacting with the blockchain or server related issues. We also get a false action when the action being done was not authorized by the server. Success is the only case which returns a true action.
  - (b) Message - It contains the proper message depending on the type of action. In case of failure, a proper error message is returned from the server. In case of success, we return the message with the appropriate value being queried or the result of that manipulation.

The app.js also contains code for connecting to our MongoDB database. It is used for storing the OTPs for the present session of the patient. Lastly, using the express listen method a server is spun up on the designated port which is by default port number 8000.

- **The network.js file:**

After importing the necessary methods like FileSystemWallet, Gateway and X509WalletMixin which would be used for creating a wallet for the entity and then connecting to the blockchain respectively, we move onto accessing our configuration setting mentioned in our config file. The config file contain keys like the app admin, the appAdminSecret the CA Name and the method of the gateway we will be using to connect to the blockchain. The CA name is actually the URL on which our Certificate Authority docker container will be hosted.

When we create a fabric runtime in the Hyperledger blockchain, a connection profile is exported from the runtime which is used in the network.js file for connection purposes. The local\_fabric\_connection.json file contains all the important fields like the URL of the CA, the timeout for the client connection, the name of our organi-

zation and the number of peers along with the URL on which the peer container is hosted. We also get information about the orderer and the timeout associated with it for ordering a transaction.

As we explore the `network.js` file we come across function named `connectToNetwork` that is used to create a wallet for a user in our wallet system. The authorization for the creation of the wallet is provided by the admin of our blockchain. Once the identity for our user has been created in the wallet a connection to the blockchain is made and a reference to that network object is returned. We also find some utility functions that will be used to check the presence of a particular user in the wallet. Once a user is present in the wallet, then only he can connect to the blockchain network.

Later we find functions that are implemented to create a wallet for a user be it doctor or patient. In order to create the identity in the wallet we follow the following guidelines:

- (1) First of all, we check for the values provided for the user for not being null or invalid.
- (2) Later we check if the user is already present in the wallet system. If yes, we return with an error.
- (3) We then check for the presence of admin in our system. In case the admin is not present we create an admin using the `enrollAdmin.js`.
- (4) We then connect through the gateway to our certificate authority. The CA has the privilege of assigning the proper public and private keys to the user that will be used for signing the transactions.
- (5) Finally, we enrol the user with the CA and create a wallet for them into our file system.
- (6) If for any of the steps provided above, we get an error we return it with a proper error message.

- **The blockchain server:**

The core of our application lies with the blockchain server. The reasons for using Hyperledger as the blockchain has already been clarified in a previous chapter. Here we will be looking into the implementation part. First of all, it is important to look at the system requirements that are a must for setting up this chain in the local. Our present system configuration is:

- (1) VSCode - (1.39.0) - It is text editor that flaunts the useful IBM blockchain extension. It is a lightweight, extension driven text editor that is used for maximising the development speed.

- (2) IBM Blockchain Platform VSCode extension - 1.0.31 - This extension is provided by Hyperledger to easily set up and run docker containers on the action of a click.
- (3) Docker - (19.3.8) - This is the container management and creation software that is being used to set up various identities of the participant of our blockchain network.
- (4) Docker-compose - (1.24.1) - It is a framework for docker used defining and running multi-container docker application. Using a simple config file we can configure all the containers that are used by our system.
- (5) NodeJs- (10.x) - This is the framework we will be using for setting up the server of our application.
- (6) Npm - (6.x) - It stands for Node package manager. It is a registry that hosts more than a million packages that are built by the open-source community for node js users.
- (7) Ubuntu - (18.04) - It is the open-source operating system widely used all over the world for efficient development.

- **The EHR-contract.ts file**

A contract is also known as the chain code contains the logic related to the modification or querying the blockchain. It is the file that is being communicated from our primary backend server and the network.js file. As you could have an observer, the language used for writing this contract is not javascript but Typescript. Typescript is actually a compile to JS language that helps with beautiful type checking and is an important up-gradation over the usual javascript.

If you look at the source code you will find an entire folder dedicated to the chain code related files. It is important to create a different context for such files as their function and methods of invoking is completely different from the rest of the code. The folder also contains dist/ directory where all the ts files are kept are compiling to js. From this directory, the blockchain extension creates the .cds package that will be later used for the creation of the blockchain.

The EHR-contract.ts file contains a total of 26 functions that are used to manipulate or query the blockchain ledger. These functions are comprised of the ones used to create the identities of the various user in the ledger, to manipulate the existence of the EhRs and to manage the correspondence of EhRs and the patients, and also the relationship between patients and doctors. The important functions in this file that we should know about are

- (1) getState - on passing the key as the parameter this function queries the chain and returns the value associated with that key.

- (2) putState - On passing the key and the value as the parameters these functions updates the ledger with the new values.
- (3) getHistoryByKey - On passing the key to this function, it queries the chain and returns the entire modification history of the value associated.
- (4) getQueryResult - On passing a selector to this function, it performs a rich text query on the entire ledger and returns an object of the key and values associated with that selector string. It returns all the key and values where the presence of the selector string is found, in the form of an iterator.

We also use some other helper ts files like the ehr.ts, doctor.ts and patient.ts that are used to easily create the entities on passing them the required values and returning a well-formed object, ready to put into the ledger.

It is important to note that everything that is put into the blockchain is in the form of a string. All the arrays and all the object needs to be stringified before storing into the ledger.

### 3.4.2 User Interface

This section describes the user interface for both the types of user viz. doctor and patient.

- Doctor UI : Diagnosis workflow. The red marker surrounding the buttons signifies that button is clicked.
- (1) **Doctor Home** : This page is the default home page for every doctor after he/she is logged in. This page contains the profile picture of the doctor, his/her "DoctorId" and designations. To diagnose a existing patient, the doctor clicks on the username/Patient Ids displayed which correspond to the patients currently designated to the current doctor. To diagnose a new patient, the patient needs to click on the new patient button and enter the Patient Id of the required patient. After entering the otp provided by the patient, the doctor can proceed further. Refer **Figure 3.5**.

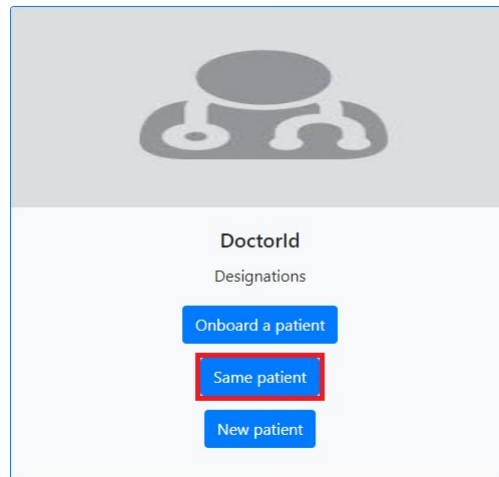
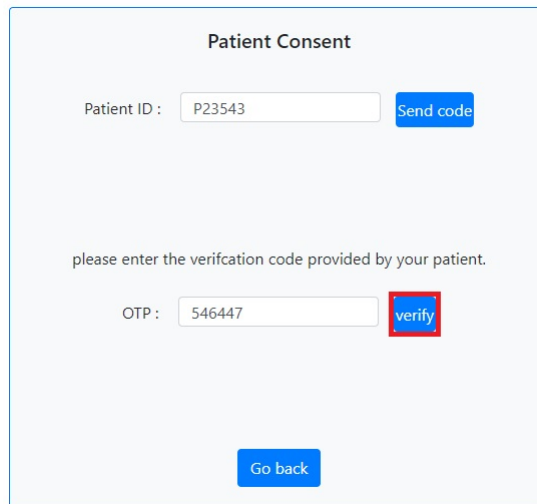


Figure 3.5: Doctor home page : same patient

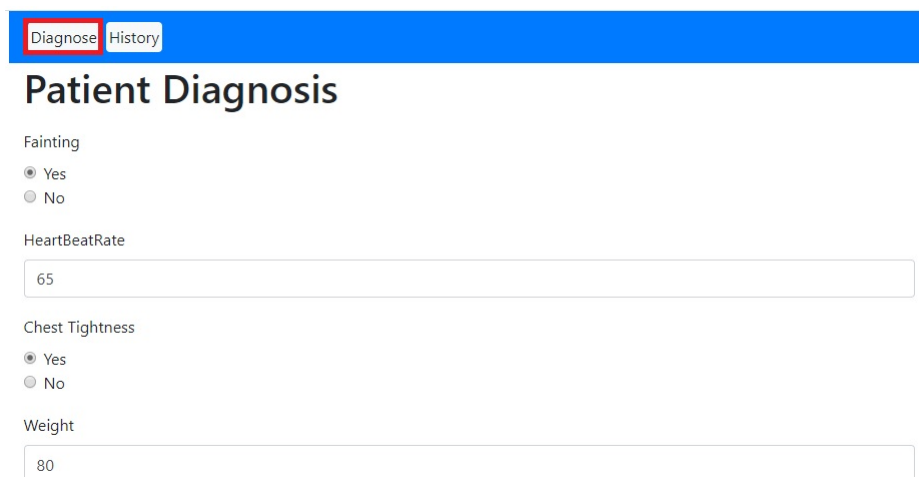
- (2) **Patient Consent** : This page prompts the doctor to enter the "PatientId" of the patient if the patient is not present under the doctor. Once the doctor enters it and clicks "Send code" button, an "OTP" is sent to the patient via the email already provided. The patient then tells this to the doctor and doctor clicks "verify" button to proceed. Refer **Figure 3.6**.



The image shows a 'Patient Consent' form. At the top, it says 'Patient Consent'. Below that, there is a 'Patient ID' field with the value 'P23543' and a 'Send code' button. Further down, it says 'please enter the verification code provided by your patient.' Below this, there is an 'OTP' field with the value '546447' and a 'verify' button. At the bottom, there is a 'Go back' button.

Figure 3.6: Patient consent page

- (3) **Diagnosis** : This is the generic diagnosis form for the cardiologists. Here the doctor enters the diagnosis data and on clicking "submit" button this is stored into the EHR blockchain. Refer **Figure 3.7**.



The image shows a 'Patient Diagnosis' form. At the top, there is a blue header bar with two tabs: 'Diagnose' (highlighted with a red box) and 'History'. Below the header, the title 'Patient Diagnosis' is displayed. The form contains several fields: 'Fainting' with radio buttons for 'Yes' (selected) and 'No'; 'HeartBeatRate' with a text input field containing '65'; 'Chest Tightness' with radio buttons for 'Yes' (selected) and 'No'; and 'Weight' with a text input field containing '80'.

Figure 3.7: Diagnosis form

- Doctor UI : Patient onboarding workflow. The red marker surrounding the buttons signifies that button is clicked.

(1) **Doctor Home** : This page is the default home page for every doctor after he/she is logged in. This page contains the profile picture of the doctor, his/her "DoctorId" and designations. To onboard a new patient, the doctor clicks on the "Onboard a patient" button. Refer **Figure 3.8**.



Figure 3.8: Doctor home page : onboarding

(2) **Patient onboarding** : This page asks the doctor to enter basic details about a patient to onboard him/her. refer **Figure 3.9**.

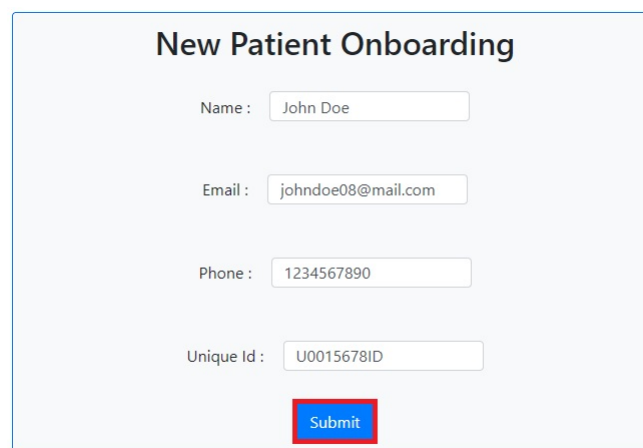
The image shows a screenshot of a web application interface for a "New Patient Onboarding" form. The form is titled "New Patient Onboarding" in bold black text. Below the title, there are four input fields, each with a label and a text box: "Name : John Doe", "Email : johndoe08@mail.com", "Phone : 1234567890", and "Unique Id : U0015678ID". At the bottom of the form, there is a blue "Submit" button highlighted with a red rectangular border.

Figure 3.9: Patient onboarding form

- **Doctor UI : History Retrieval workflow.** In order to retrieve the history details of the patient the patient first needs to get the consent of the patient via the OTP. We will consider the case of retrieving the history details of the an old patient who has been following up the doctor.

- (1) **Doctor Home :** This page is the default home page for every doctor after he/she is logged in. This page contains the profile picture of the doctor, his/her "DoctorId" and designations. The patient IDs of the patients existing under the doctor are also displayed. To get the history details of an already existing patient, the doctor has to click the Patient Id of the required patient. Refer **Figure 3.10**.

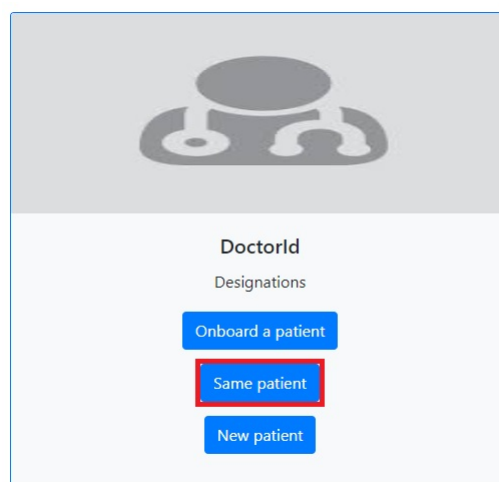


Figure 3.10: Doctor home page : history

- (2) **Patient Consent :** This page asks for the patient consent i.e the OTP. The patient provides them and after verifying it, the doctor is led to the page where he/she can add diagnosis or view the patient history. refer **Figure 3.11**.

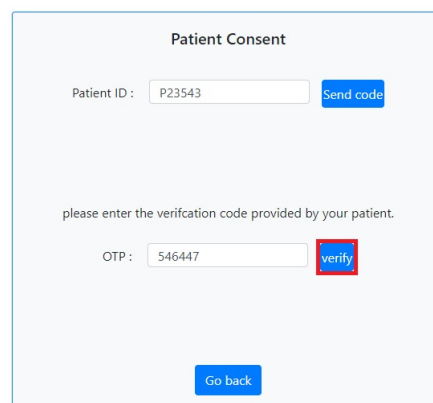
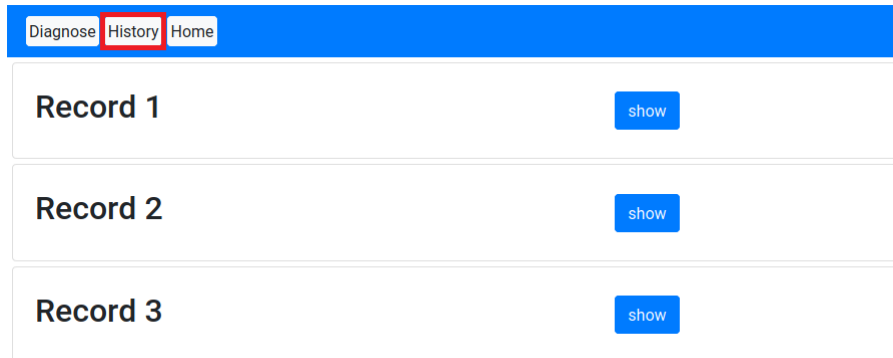


Figure 3.11: Patient consent form



- (3) **History Retrieval :** Here the doctor can click on the history and a structured result will be returned back. This result will be in the form of records entered by different doctors. On clicking the record, we can see the particular details of the record and the doctor can make an analysis of the history of the patient.



Diagnose History Home	
Record 1	show
Record 2	show
Record 3	show

Figure 3.12: Patient history viewed by doctor

- Patient UI: Patient history read. The different processes and functionalities of the patient are mentioned here.
- (1) **Patient Login:** Just as it is shown in the figure, there are two options for patient login, the option for OTP and the option for entering the password. Since no password is set, the patient can click on the OTP option and after entering the OTP received, the patient is prompted to set a new password as shown in the **Figure 3.13**



Choose the method of Login

if this is your first login

Generate OTP

OTP : \*\*\*\*\* Check

or

password :

Login

[click here to login as doctor](#)

[Go back to home](#)

Figure 3.13: Patient login

- (2) **History Retrieval:** The patient has the option to click on history which will lead to a detailed history record of the patient.

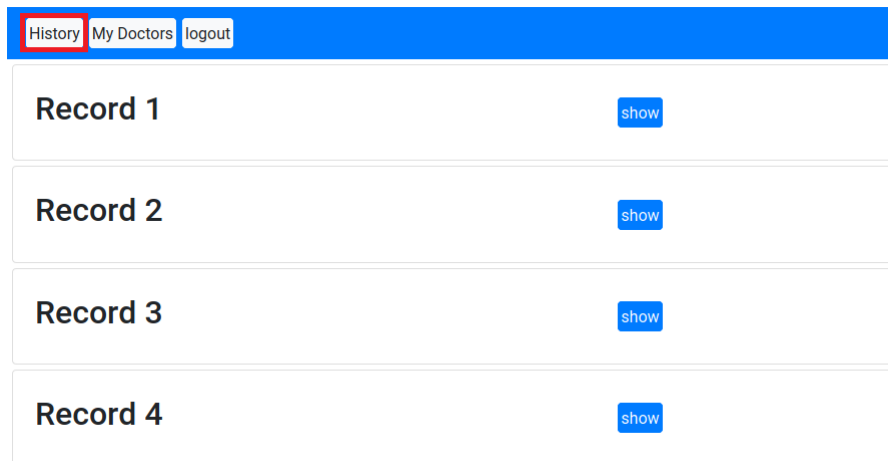


Figure 3.14: Patient history

- (3) **List of Doctors:** Clicking on this, the patient can view the list of doctors the patient visited.

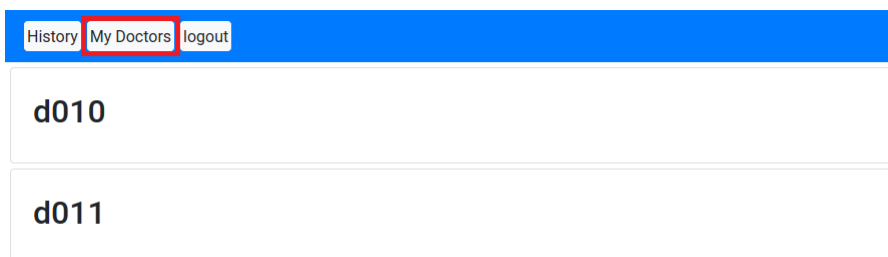


Figure 3.15: Patient's current doctors

### 3.4.3 Gantt Chart

The following chart presents the various phases of the project and the work done in each of them. Refer **Figure 3.16**

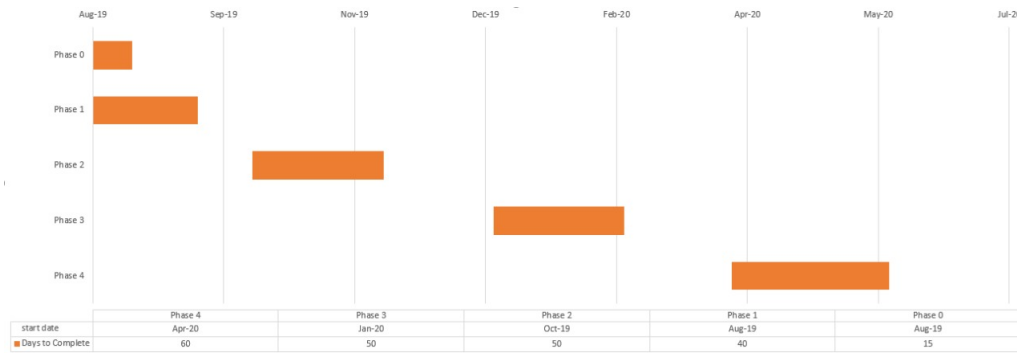


Figure 3.16: Gantt Chart

Where,

- Phase 0- Ideation and Topic Deciding: The whole plan and ideation about the project was done in this phase.
- Phase 1- Basic Demonstration on Composer playground: In this phase, a basic demonstration of the project was made on the IBM Blockchain Composer playground where the whole environment was online.
- Phase 2- Setting up the local environment: In this phase, all the required set up was made in the local environment and a basic working of the project was demonstrated on the terminal itself.
- Phase 3- Addition of functionalities along with front end and backend: Here, the frontend and the backend servers were made along with the blockchain server which had some functionalities and a demonstration of the same was made.
- Phase 4- Developing a full fledged application with all the decided functionalities: Finally a complete application was developed by integrating all the servers which has all the required functionalities.

---

## 4 Conclusion and Future Work

---

The proposed system diminishes the shortcomings in the health sector if the implementation is leveraged to a state-of-art efficiency. Although its heavy computational requirements and its complex structure, this system is practical enough as it provides the required privacy, interoperability and anonymity by utilizing the best technological frameworks available at the present age.

Further efforts can be made to utilize this system for predicting the probability of chronic disease to a patient [3] or early prediction of stroke [4] or alzeihmers [5] by drawing various insights from the data. This can also be used to make valuable contributions to the personalized health care for individual patients [6]. The data from the EHRs can be accessed by various academic or medical institutions for research purposes.

## References

- [1] A. Donawa, I. Orukari, and C. E. Baker, “Scaling blockchains to support electronic health records for hospital systems,” 2020.
- [2] D. Yaga, P. Mell, N. Roby, and K. Scarfone, “Blockchain technology overview,” Oct 2018. [Online]. Available: <http://dx.doi.org/10.6028/NIST.IR.8202>
- [3] J. Liu, Z. Zhang, and N. Razavian, “Deep ehr: Chronic disease prediction using medical notes,” 2018.
- [4] C. S. Nwosu, S. Dev, P. Bhardwaj, B. Veeravalli, and D. John, “Predicting stroke from electronic health records,” 2019.
- [5] H. Li, M. Habes, D. A. Wolk, and Y. Fan, “A deep learning model for early prediction of alzheimer’s disease dementia based on hippocampal mri,” 2019.
- [6] J. Zhang, K. Kowsari, J. H. Harrison, J. M. Lobo, and L. E. Barnes, “Patient2vec: A personalized interpretable deep representation of the longitudinal electronic health record,” *IEEE Access*, vol. 6, p. 65333–65346, 2018. [Online]. Available: <http://dx.doi.org/10.1109/ACCESS.2018.2875677>

---

## 5 Acknowledgement

---

We take this opportunity to express our sincere gratitude and special thanks to Dr. Bhavesh. N. Gohil who took time out to hear, guide and show the right path. We would also like to express our gratitude to the faculty of Computer Engineering Department, Sardar Vallabhbhai National Institute of Technology, Surat for helping us directly or indirectly. We will strive to use the gained knowledge in the best possible way.

## Appendix A Minimum requirements

Having Ubuntu OS is essential for proper installation and functioning of our system. You need to have at least Ubuntu OS 16.04, 4 GB RAM, 10 GB free space and active internet connection in your system for successful installation and working.

## Appendix B Prerequisites

After executing following steps one by one your system will be equipped with all the prerequisites needed for the further process.

### B.1 Installing node

- (1) Open your terminal by using **ctrl + alt + t** and run command **sudo apt install curl** .
- (2) Now, run **curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.11/install.sh | bash** . This installs Node Version Manager (nvm) in your system.
- (3) Now close the current terminal and open a new one.
- (4) Run **nvm --version** to check if it is installed properly or not. If everything went well then you should see the version number output in the terminal.
- (5) To see the available versions of node run **nvm ls-remote**. This will display you the list of available versions for download.
- (6) From the list resulting from above choose any version of the form 10.x.x (LTS : —) and run **nvm install 10.x.x**.
- (7) If all went well, you will see v10.x.x on running **node -v**
- (8) If you already had a different version of node installed then run **nvm use 10.x.x** and then **node -v** to see if the node is switched to version 10.x.x.
- (9) Now execute **sudo apt install npm** to install Node Package Manager (npm).
- (10) Use **npm -v** to check the version of installed npm. The latest available version of npm at the time of our project was v6.14.5. This completes the node setup.

## B.2 Installing angular

Follow these steps to install angular v9.1.7 on your system.

- (1) Open your terminal by using **ctrl+alt+t** and run command **npm install -g @angular/cli@9.1.7**.
- (2) After that run **ng version**.
- (3) If all went well, you should get something like **Figure B.2.1**. This completes the angular installation.



```
Angular CLI
Angular CLI: 9.1.7
Node: 10.20.1
OS: linux x64

Angular: 9.1.9
... animations, common, compiler, compiler-cli, core, forms
... language-service, platform-browser, platform-browser-dynamic
... router
Ivy Workspace: Yes

Package                                  Version
-----
@angular-devkit/architect                0.901.7
@angular-devkit/build-angular            0.901.7
@angular-devkit/build-optimizer          0.901.7
@angular-devkit/build-webpack            0.901.7
@angular-devkit/core                     9.1.7
@angular-devkit/schematics               9.1.7
@angular/cli                             9.1.7
@ngtools/webpack                         9.1.7
@schematics/angular                     9.1.7
@schematics/update                       0.901.7
rxjs                                     6.5.5
typescript                               3.8.3
webpack                                  4.42.0
```

Figure B.2.1: Angular version output

## B.3 Installing docker

This section will help you to install Docker engine on your computer.

- (1) Open your web browser and go to <https://download.docker.com/linux/ubuntu/dists/>
- (2) Select your Ubuntu OS version from the list provided there and browse to pool/stable/, choose amd64, armhf, arm64, ppc64el, or s390x, and download the .deb file for the Docker Engine version 18.06.3-ce.



- (3) Then open the terminal in the folder where the **.deb** file is downloaded and run **sudo dpkg -i packageName.deb**
- (4) To verify that it is properly installed run **sudo docker run hello-world**
- (5) In the terminal run **sudo chmod 777 /var/run/docker.sock**

## B.4 Installing VSCode and its extensions

This section will guide you to installing Visual Studio Code which is a code editor and its extensions essential for this project.

- (1) Download VSCode for ubuntu by going to this link <https://code.visualstudio.com/download>. Version 1.39 is recommended
- (2) After downloading the **.deb** open the terminal in that folder and run **sudo dpkg -i packageName.deb** to install it.
- (3) Now, open VSCode and go to extensions by pressing **ctrl+shift+x**
- (4) Search for docker as shown in **Figure B.4.1** and install that extension. At the time of our project we used v0.8.2, which is also recommended to use.

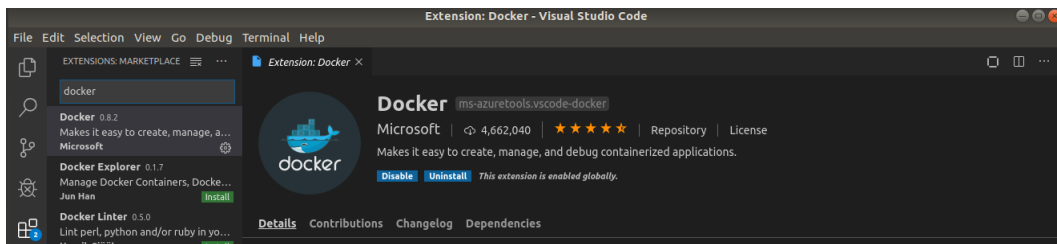


Figure B.4.1: The docker extension

- (5) Search for IBM blockchain platform as shown in **Figure B.4.2** and install that extension. At the time of our project we used v1.0.28, which is also recommended to use.

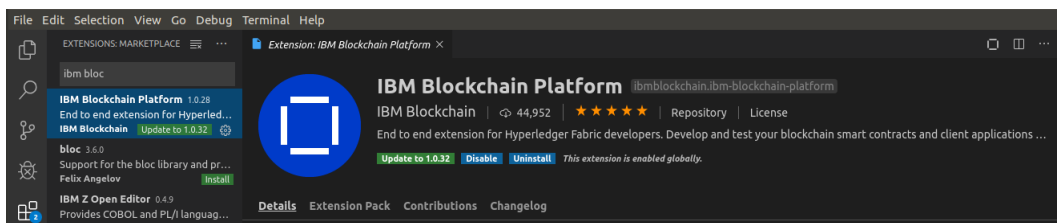


Figure B.4.2: The IBM block chain platform extension

This completes all the extension installation.

## Appendix C Installing project modules

To download all the files and modules of our project follow the steps mentioned in this section.

- (1) Download the zip file from our Github repository at <https://github.com/littlestar642/ehrBlockchain>
- (2) Unzip the downloaded file and open terminal in that unzipped folder.
- (3) Then run the following commands `cd server/` , `npm install`, `cd ..` , `cd client/` , `npm install`, `cd ..` , `cd contract/` , `npm install`. This completes the section.

## Appendix D Running the application

To run the application follow these steps

- (1) Open the **contract** with VSCode.
- (2) Go to block chain platform and select **package open project** as shown in **Figure D.1**.

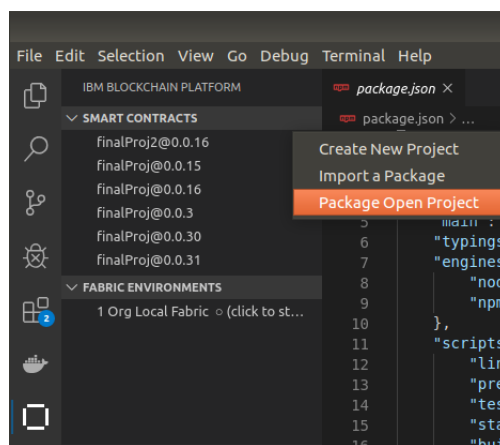


Figure D.1: Package open project

- (3) Now close this VSCode. Go back to the unzipped folder and open this folder in VSCode.
- (4) Now again go to the block chain extension. Export the wallet **Org1** as shown in **Figure D.2** by placing it in the **server** folder and rename it to **wallet**

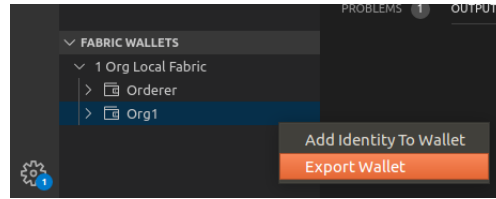


Figure D.2: Exporting wallet

- (5) You will notice the presence of packaged project from previous step, Now connect to the environment by clicking on **1 Org Local Fabric**. On successful connection the VSCode editor will prompt you a message.
- (6) On successful connection you will get various options in the environment pane on the left. Here install the package as shown in **Figure D.3**.

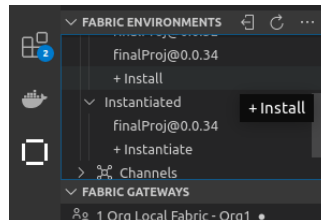


Figure D.3: Installing the package

- (7) Now instantiate the installed package in the same pane by following **Figure D.4**.

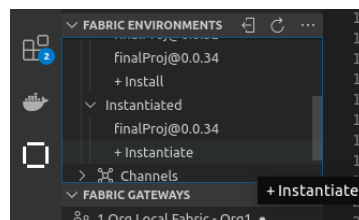


Figure D.4: Instantiate the package

- (8) Now open your Ubuntu terminal in the **server** folder and run command **node src/app.js**. This will run a server listening on port 8000.
- (9) Open new Ubuntu terminal in the **client** folder and run command **ng serve**. This will setup a server listening on port 4200.
- (10) Access the homepage of our application by going to **https://localhost:4200/homepage**. This completes run procedure of our system This system can now be used by the user.