Patient Medical Record Management Using Blockchain

Project Work Report

Submitted in partial fulfillment of the requirements for the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER ENGINEERING

by

Bandaru Bharath Kumar Reg No.15CO113
Kenguva Ananth Kumar Reg No.15CO124
Palempalli Reddi Narasimha Prasad Reg No.15CO133

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,

SURATHKAL, MANGALORE - 575025

May, 2019

DECLARATION

We hereby declare that the Project Work Report entitled **Patient Medical Record Management Using Blockchain** which is being submitted to the **National Institute of Technology Karnataka, Surathkal** in partial fulfilment of the requirements for the award of the Degree of **BACHELOR OF TECHNOLOGY** in **Computer Engineering** is a *bonafide report of the work carried out by us.* The material contained in this report has not been submitted to any University or Institution for the award of any degree.

Bandaru Bharath Kumar (15CO113)
Department of Computer Science and Engineering

Kenguva Ananth Kumar (15CO124)
Department of Computer Science and Engineering

Palempalli Reddi Narasimha Prasad (15CO133)
Department of Computer Science and Engineering

Place: NITK, Surathkal.
Date:

CERTIFICATE

This is to *certify* that the B.Tech Project Work Report entitled **Patient Medical Record Management Using Blockchain** submitted by:

Sl.No. Register Number & Name of Student(s)

(1) 15CO113 Bandaru Bharath Kumar

(2) 15CO124 Kenguva Ananth Kumar

(3) 15CO133 Palempalli Reddi Narasimha Prasad

as the record of the work carried out by them, is accepted as the *B.Tech. Project Work Report submission* in partial fulfilment of the requirements for the award of degree of **Bachelor of Technology** in **Computer Engineering**.

Guide(s)

Mahendra Pratap Singh

Chairman - DUGC

(Signature with Date and Seal)

Acknowledgment

We would like to thank Mahendra Pratap Singh, Assistant Professor, Department of Computer Science and Engineering at National Institue of Technology Karnataka, Surathkal for giving us an opportunity to work under him for the Major Project. We would like to extend our gratitude for his continuous support.

Place: Surathkal

Date:

B. Bharath Kumar - 15CO113

K. Ananth Kumar - 15CO124

P. Reddi Narasimha Prasad - 15CO133

Abstract

Blockchain, a technology used in bitcoin made a revolutionary change in the digital market. The Blockchain acts as an immutable ledger and transactions take place in a decentralized manner and its applications are growing steadily. It is a system of creating a distributed consensus in the online world. One of the major fields that are important in the present world is the medical field. Every person wants their medical records to be stored, secured and controlled by themselves. The blockchain is going to be the best solution to these problems. Blockchain helps to store all the records, makes them secure and access them anywhere. In this report, our focus is to implement a patient medical record management application using blockchain technology.

Keywords: Blockchain, digital market, decentralized, medical records, authentication, confidentiality, data sharing.

Contents

Chapter 1

Introduction

Bitcoin is a cryptocurrency which was built using blockchain and it is proposed as a peer-to-peer digital cash system without the trust of the third parties(Pinyaphat and Chian, 2018). Its major contribution is that it presents a new way of open networks. Decentralization property in blockchain allows information to propagate in a peer-to-peer manner. Blockchain manages transactions, contracts, and agreements which is essential to legal systems in the world. The blockchain is defined as public, permanent, append-only distributed ledger. It has many advantages due to its decentralization property. It provides trust for every transaction. Blockchain technology is used in different fields(Rodelio and Fernande, 2018) like to prevent fake certificates in real estates and contracts(Hiroki et al., 2015) to prevent forgery of documents. Blockchain also helps in mitigating the DDoS attack(Rodrigues et al., 2017). This technology can be used in any field. Our main focus is in the medical field especially for storing and processing the medical records(Kate, 2018) of the patient.

Let us understand the current scenario of the medical records. The health records are fragmented, distributed over different health care institutes. A patient medical history is fragmented into pieces, dispersed and locked within multiple organizations and providers. Sometimes medical records may be lost due to relying on paper-based method. These records can be tampered, misused or even stolen because of intermediaries getting access to it without the patient permission. Lack of necessary information may sometime make it difficult for the health care providers to administrate appropriate treatment. The current status of the medical records is fragmented due to a lack of common protocols, architecture, and standards. If we can find a way to store all

these records then it is possible for the health care providers to administer the good treatment.

Not only storing the records but also accessing them outside the organization is important. Consider the situation where your data is stored in the local physician office computer, in this case, none of your records are available for immediate access to the providers outside of your doctor's office. If you suffer from a serious problem then the hospital or emergency department is going to contact your physician requesting your records, because they want to be careful while giving a drug so that it is not going to interact negatively with the medication you're taking, thus delaying the treatment until the records arrive.

To avoid these situations all the physicians should be able to see the medical records with much ease. With all the above issues considered, blockchain could be a most promising solution(Mertz, 2018). With the blockchain based protocol, every modification regarding a patient would be linked to the chain as a new block. Blockchain can be considered as a virtually incorruptible cryptographic database. The things that can be achieved by using blockchain are *data privacy and security*, *data access and control*, *data storage*. Personal health care records are to be protected with high standard, the blockchain provides this because the data in the blocks cannot be edited i.e. immutable and the data is stored across different systems. Data access and data control are the main features for the patient. Not only the patient should have full access to his data but also he should be able to control who can access the data.

1.1 Blockchain Technology

The blockchain is a cryptographic protocol that connects a network of systems to maintain the shared ledger information. Each block in the chain is a time sequenced chain of events that are created using consensus mechanism. All the blocks in the blockchain are linked using the previous hash value of the block. A block in blockchain contains different fields (Zheng et al., 2017) which are given below

- **Version**: Indicates the set of validation rules that need to be followed.
- **Merkle tree root hash**: Contains the hash value of all the transactions.
- **Time stamp**: current time in seconds.
- **Nonce**: A variable which usually starts with 0 and increases for every block hash value calculation.
- **Parent block hash**: Contains hash value that points to the previous block.

After the successful creation of the blockchain, the first block known as the genesis block is generated automatically and the next new block is added to the genesis block. The maximum number of transactions that a block can store depends on the block size and the size of each transaction.

Some of the characteristics of blockchain are as follows

- **Digitized**: The information stored is digitized thus eliminates the need for manual documentation.
- **Decentralization**: In blockchain, third-party services are no longer needed. Consensus algorithm is used to maintain data consistency in a distributed network.
- **Persistency**: Transactions that are valid will only be accepted by the miners. Once they are included in the blockchain it is impossible to rollback or deletes the transactions.
- **Consensus-based**: A transaction on blockchain can be executed only if all the parties on the network approve it.
- **Chronological and time-stamped**: All the blocks in the blockchain are connected chronologically providing a trail of the underlying transaction.
- **Cryptographically sealed**: Blocks created are cryptographically sealed in the chain. This means that it becomes impossible to delete or edit already created blocks thereby creating true digital assets and ensuring a high level of robustness and trust.

1.1.1 Types of blockchain

There are two types of blockchain called permissionless blockchain and permissioned blockchain(Alhadhrami et al., 2017) that are described below.

1.1.1.1 Permissionless blockchain

These type of blockchains are public blockchains that allow multiple nodes to connect to it and participate in the network to perform the operations without having to rely on trusted third parties. These types of blockchain will not restrict access to any nodes. The blockchain is used to store, validate and maintain records of the transactions that occur in the network. These transactions are grouped and stored in blocks on a public ledger. The transactions that occur in the network are verified and added to the chain by the set of specific nodes called miners. These miner nodes need to solve a difficult puzzle in order to add the block to the chain which is known as the Proof-of-Work mechanism.

1.1.1.2 Permissioned Blockchains

Unlike permissionless blockchain, there are some restrictions imposed on the permission blockchain. It will provide better privacy, confidentiality, and scalability along with the functionalities supplied by the original blockchain model. Permissioned blockchain is in two forms they are consortium blockchain and private blockchain. In consortium blockchain, the consensus process is controlled by a set of preselected trusted nodes. Only these preselected nodes can add the blocks to the blockchain. In consortium blockchain, the permission for reading or writing can be only to the participants or can be made public. The consortium blockchains are considered as partially decentralized because of the restrictions. In private blockchains the write permissions are kept centralized to one organization and the right permissions may be made public or restricted to specific nodes.

1.1.2 Verification and validation of blocks

Before adding the data or transaction to the blockchain, they must undergo a process called 'mining'. All the transactions that are to be validated are grouped together

into blocks for verification. Verification is performed by miners who devote computing power to solving a puzzle. This is known as "proof of work" mechanism. This involves finding a hash value that begins with a certain number of zeros. The hash is calculated by combining the data, timestamp, previous block hash value and nonce and applying encryption algorithm such as SHA-256. This verification is a competitive process as the first miner to solve the puzzle gets the right to broadcast the new block through the network. Once the block is accepted by the rest of the nodes in the network that block is added to the blockchain and all the remaining nodes stop the execution. The node which has the higher resource capacity has a higher probability of solving the puzzle first.

1.1.3 Smart Contracts

A smart contract is a set of programs or functions that are self-executing, self-verifying and tamper resistant that runs on the blockchain(Mohanta et al., 2018). The integration of smart contracts with the blockchain gives lots of flexibility to develop and design the solutions to the real world problems without the involvement of third-party systems. When an event is triggered, automatically these contracts start getting executed. These contracts are immutable and distributed across the blockchain network. The output can be validated by everyone in the network. This makes the system open to the public.

The importance of smart contract integration with blockchain technology has become a focus area for development because it provides a trustful environment for peer to peer transaction and database. Smart contracts are irreversible and trackable. Ethereum blockchain is the largest ones to use smart contracts and a high-level language called solidity which is used for developing smart contracts.

Some of the main characteristics of smart contracts are as follows:

- They are machine readable and high-level language code run on blockchain platforms.
- They are part of an application program.
- They are event-driven.

- They make the system autonomous.
- They are distributed.

Blockchain which incorporates smart contracts offers a number of benefits like accuracy, lower execution risk, speed and real-time updates, lower cost, fewer intermediaries, security and accessibility(Arlindo et al., 2018).

Using smart contracts, the cost that involves in processing can be reduced when used effectively. Niya et al. (2018) used the smart contracts to reduce the cost that was involved in storing the data.

1.2 Uses of blockchain in medical field

The first use is that it can be used to connect different health care centers across the globe to a single application so that all the medical records are linked and can be accessed any where making things easier for the health care providers. It reduces the cost for the maintenance of the local databases. In blockchain, each block added is time stamped, immutable and the data inside is verified before adding, so there will be very less chance to fake the documents which ultimately provides a better security for the records.

So many organizations like freed associates(Stagnaro, 2018), deloitte(Krawiec et al., 2016) have come out and explained how the blockchain technology has the potential to change the health care applications.

Some of the uses that blockchain provides are as follows:

1.2.1 Longitudinal health care records

All the records are linked securely across the various health care provider organizations. Matt et al. (2015) provides a blockchain scenario where patient 'Jane' has an acute episode and is attended by an emergency medical technician (EMT). The details of

Jane's are sent to the primary care physician and hospital to obtain an access to all of her records.

1.2.2 Automated health claims

Blockchain is used to support concepts like trustless and dis-intermediation exchange using smart contracts. Smart contracts enable the nodes to execute a transaction for the event. This logic written in smart contracts ensures correct completion of claims and supports compliance audits. API's combined with the smart contracts and blockchain can be used to process the data anywhere.

1.2.3 Interoperabilty

Blockchain can be used to facilitate the gathering of massive amounts of patient data to aid population health initiatives. Having the same type of interface in all health cares, it is easy to transfer data from one hospital to another unlike in regular EHR.

1.2.4 Online patient access

Using blockchain, the patient can securely access their longitudinal health care records. Since all the records are linked from all the health care providers, smart contracts along with API's can be used for enabling patients to check their medical records anywhere and every time.

1.3 Advantages

Advantages of the blockchain technology are as follows:

1.3.1 Integrity

As the block is added with the consent of the majority of the nodes, the data added is expected to be accurate.

1.3.2 Availability

Being distributed in nature the availability is high. Even if one node fails the other node will be used to execute the queries, because of this distributed nature the application is

able to withstand the denial of service attack. Since the application is autonomous and works 24x7, data retrieval can be done at any place and any time. So interoperability should be satisfied.

1.3.3 Security

Only the index values are kept in the blocks so even if a hacker look at the data they can not get any information, moreover all the data in every stage is encrypted and it can only be decrypted by using the patient private key. So, the application is secured.

1.4 Problem Statement

Being healthy is important to all the people in the world. There are a countless number of treatments in the current living scenarios. Keeping track of those treatment records is important. Even though there is a tremendous change in the current technology, still there is a need for improvement in the medical field. Medical records are important to the patient for further treatments. Storing and accessing these medical records is significant. In the present scenario, the interoperability of medical records is difficult, time taking process and patient's access to the medical records is not permitted in some health care centers. A system where the patient can securely store and access all his medical records needs to be developed. Blockchain can provide all the functionalities. We aim to develop an application using blockchain where the patient has access to all of his/her records.

1.5 Motivation

In present days it has become difficult to store and track all the medical records in one place when the records are from different health care providers. In some health care centers, the patients are not permitted to take health reports with them. Sharing of the reports from one hospital to another also becomes difficult. In order to solve the problems that the patients are facing, we aim to develop a blockchain based patient medical record management application.

1.6 Need for the new technology

The present medical record system does not offer the interoperability i.e. the ability of computer systems or software to exchange and make use of information. Let us understand why we need interoperability, in present days medical records are important for any patient. For the patient to continue his/her treatment, previous records became necessary for better treatment because of the countless number of treatments available. Suppose if a patient goes to multiple health cares for the same problem without the previous health records then there can be a chance that he might be given the different type of treatments for the same problem, this again might lead to some new health problem. Suppose if the same patient takes the previous health records then the health care provider may give treatment depending upon the previous course he followed. The health care provider can be able to understand the patient problem clearly and there will be a better chance to provide the correct treatment. If some kind of scans or tests is involved then the patient can not always go through them whenever he visits a new health care.

In some health care centers, the patient records are confined to their health care centers, not allowing the patient to have access to their records. In case of emergency, transferring of the patient records will take much time and it is a laborious process. Since the storage of the records is confined to a single place, there might be a chance of misuse of the records. In these cases, the patient will not have access to their records. To overcome all these problems, there should be a need of development of a new technology that makes the patient the sole owner of their records, able to store all the records, should not be vulnerable to the external attacks, patient should be able to decide read and write access for their records so as to provide strong security against different kind of attacks like tampering of the data, adding fake records, etc. Blockchain provides all the features that are required to overcome these problems.

In the present times, about 25% of the money being spent is wasted on maintaining the records. All the health care centers who are using traditional centralized databases need to spend more money on maintaining the databases, need to hire special engineers

for securing the data from external attacks. If we use blockchain then the administrative cost can be minimized. Since blockchain is distributed across the network we do not have to worry about the external attacks. It also acts as a global database where the patient records can be viewed anywhere. Blockchain can securely maintain a continuously growing list of data records. Blockchain has data integrity with strong cryptography inbuilt. Getting the required medical records will be easy as they can be accessed anywhere from the application. It allows the patient to decide the permissions for their records.

One important property of the blockchain is that it can prove that a unique event has occurred at a certain time. This makes the blockchain unique. Smart contracts are used to manage the transactions effectively and these transactions can be verified by anyone in the network.

1.7 Objectives

The objective is to implement the patient medical record management using blockchain technology with the help of different frameworks, compilers and test tools. We use remix(an IDE for smart contracts), truffle(framework for running the smart contracts), solidity compiler, ganache. The final goal is to come up with a real-life workable solution which can be used in health care centers.

1.8 Organization of the report

The rest of the report is organized as follows. Chapter 2 discusses about the literature survey. Chapter 3 discusses the proposed blockchain based patient record management application. Chapter 4 deals with implementation and results. Chapter 5 contains conclusion and future work.

Chapter 2

Literature Survey

In this chapter, we present a brief overview of previous work done.

New technology was developed to store the patient records digitally known as Electronic Health Records(EHR) or Medical Health records. Tom Joseph et al. (2014) explained the administrative needs that an EHR should have, like patient documentation, quality assurance, tracking patient utilization, health record portability. He also described the uses of EHR application for physician order entry, laboratory systems, clinical documentation, pharmacy system, administrative applications. Finally, he discussed some strategies for software implementation.

As health data is increasing day by day, managing health care records become more costly in terms of security. Interoperability is also a major issue in EHR. So there is a need for the development of new technology to reduce the cost and has the same standard of interface and application throughout the medical cares. The blockchain is the only solution for those problems. There are many types of research done on the blockchain technology, the first application of blockchain was the bitcoin proposed by Nakamoto (2009). He came up with the idea of transferring the money without the help of third parties. His main idea was that all the transactions that were made should be immutable and can be trackable.

Robert (2018) considered the nature of the EHR system and the problems that were being faced. Issues addressed on EHR were ownership of the records, lack of interoperability (sharing of the records/data), no common interface everywhere and security of

data. Due to EHR design shortcomings, it would take physicians much time to access the patient data. In consideration to all those problems, he proposed to use blockchain technology because the protocols of blockchain are universal, the applications in healthcare would be compatible, never distorting the data itself so that all users could safely access the information.

The advantages of this model are patient becomes the owner of their records i.e. they can allow who can and can not see their records. It can be a difficult task for a hacker to extract or tamper with the data because the data in the blockchain are protected by the private and public key of the patient.

Linn (2016) considered the interoperability problem of the present advanced EHR system. Because of the recent advancements in cryptocurrency and blockchain, he came up with a blockchain based solution to solve the interoperability problem. He used the public blockchain as an access-control manager to health records that were stored off the blockchain. His proposal was based on the three key elements, they are scalability, access security, and data privacy. The proposed model was designed based on the bitcoin blockchain. Storing the medical records in all the nodes is expensive, bandwidth intensive and causes the wastage of resources. In order to use the blockchain to its full potential, he stored indexes in the blockchain. The index acts as a catalog in a library. Each transaction in the block contains an encrypted link to the health record, a user's unique identifier, and a time stamp when it is created. To facilitate the user queries and data access efficiency, the transactions contain the data which is used frequently. The medical data is stored in the data repository called data lake which is scalable, stores a wide variety of data like images, document, etc., and the information stored is encrypted and digitally signed for ensuring the privacy and authenticity. Every time a new health care record is created, it is saved in the data lake and a pointer to the record along with the patient unique ID is registered in the blockchain.

The proposed solution facilitates faster and easier interoperability between the systems and can efficiently handle large volumes of data. It also eliminates the need for the development of complex point-to-point data integrations between the differ-

ent systems. As the application is processed in real-time, it improves clinical care in emergency medical situations. The user is given the full access to their data and he can decide the read and write permissions for the other users. The decentralization of blockchain combined with the digital signature ensures the security of the data as the blocks contain the hashed pointers and encrypted information.

Azaria et al. (2016) proposed a blockchain based solution and its implementation to the above problems that were discussed. They developed an application known as MedRec. It uses ethereum based blockchain and smart contracts for the application. This application prioritizes the patient benefits providing trust and continued participation. It also prioritizes usability which aggregates references to all of a user's patient-provider relationships using contracts, thus providing a single point of reference. A synchronization algorithm handles the data exchange between the patient database and the provider database by confirming the permissions stored in the blockchain.

The constructed contracts contain data about the ownership, permissions and data integrity. These policies are designed to implement a set of rules to access a particular medical record. The system structures the records on the blockchain by implementing three types of contracts for navigating a large number of records. They are Registrar contract(RC), Patient-provider relationship contract(PPR), Summary contract(SC). The RC maps participant identification to the ethereum address identity. This identification can also be restricted only to certified institutions. The PPR is issued between two nodes where one node stores and manages medical records of others. SC helps to locate the medical record history and it holds references to PPR representing all participants previous and current logs with other nodes in the system. The SC also implements functionality to enable user notifications. Two different models for mining are proposed. In the first method, the first node that solves the computational puzzle is given some ether as the reward. In the second method, the medical researches and health care authorities are encouraged to mine in the network as for the reward the network beneficiaries release access to aggregate, anonymized medical data as mining rewards.

The advantages of the proposed model are it manages authentication, confidentiality, accountability and data sharing-crucial considerations when handling sensitive information. It integrates existing local databases facilitating interoperability and making the system convenient and adaptable.

From the above, one of the important aspects of the application is linking all the records from the database(records are stored in the database) to the blockchain. The reason for storing the data in the local databases is that a large amount of data can not be stored in the block. It adds much overhead. In order to reduce the overhead and run the application smoothly, storing the index is much easier and takes less space. On each node, a hash table is maintained and it consists of the patient UID and the hash index. This is done to improve the search and retrieval queries when they are accessed. For example, Whenever a patient wants to check his records, a request is sent to the nodes. Each node has a hash index that is stored along with the patient UID and the corresponding files are retrieved from the local database and shown as a whole to the patient.

All the permissions, access to the data, retrieval of the data, adding the new data to the blockchain are managed and executed automatically through smart contracts. The smart contracts start executing whenever a certain event occurs. Using smart contracts, the system can be made autonomous and can be used 24 by 7. Smart contracts play a major role in the application that is to be designed because these smart contracts manage all the functions that are performed in the application.

In order to add the block, the miners need to validate the transactions and mine the block. This mining is done using ‘proof-of-work‘ consensus algorithm. In this application, particular group persons are assigned to mine the block and verify the transactions. These may include medical researchers or a specific unit in a particular health care center. Once a block is created, that block cannot be changed or modified and hence the transactions are immutable.

In the current EHR, the data is being kept in the local databases of health care providers. This becomes a problem when the patient needs to access their data and

sharing the data between two health care providers. Because of these disadvantages in using traditional relational databases for storing patient medical records, there is a need to shift to the new technology. We used blockchain technology for the implementation of the patient medical record management application. Blockchain technology offers many advantages like security, interoperability, etc. In the next chapter, we elucidate how blockchain technology is used for storing medical records and its implementation using various frameworks and tools.

2.1 Summary

In this chapter, we described the problems of the current EHR and the need for new technology. We also described the different solutions that were proposed by the researchers in the patient record management application and explained why we use blockchain technology to implement our patient medical record management application.

Chapter 3

Proposed Blockchain Based Patient Record Management Application

This chapter is divided into four sections. Section 1 discusses the components of the application. Section 2 describes the smart contracts of the application. Section 3 explains the work flow of the application. Section 4 discusses the advantages of using blockchain over database.

3.1 Components of the Application

Our application contains four entities which participate in the process of storing medical records in the hospital. They are:

3.1.1 Receptionist

3.1.2 Doctors

3.1.3 Patient

3.1.4 Lab technician

There are different functions associated with each entity. These functions are implemented using smart contracts which are written in solidity language. Each smart contract is compiled in remix and deployed in private blockchain - provided by ganache - which provides 10 different accounts along with their private keys. We use these deployed contracts for performing and testing the various functions associated with each entity.

3.1.1 Receptionist:

The functions performed by the receptionist:

1. Patient Registration
2. Doctor Registration
3. Lab Technician Registration
4. Appointment booking between the doctor and a patient.

The Dashboard of receptionist contains views for performing each of the above functions. For registration of each of the patient, doctor and lab technician, receptionist enters the data of the particular user and submits it. Once he/she submits the data, an instance of the particular user's contract is created and stored in the blockchain. Here,the term user refers to either patient, doctor or lab technician. Similarly, for the appointments of patient, an instance of appointment's contract is created for each of the appointment created by the receptionist and is stored in the blockchain.

3.1.2 Patient

The patient interface contains all his/her medical records. The data of the medical records are obtained from MedicalRecord contract by calling the RetrievePatientAppData function. Information regarding each appointment is retrieved from the MedicalRecord's contract. All records along with dates are displayed in the page. So, after clicking the 'view' button of each appointment, patient can check his/her medical record regarding the diagnosis and the treatment given by the doctor.

3.1.3 Doctor

The doctor dashboard contains the list of all appointments which are not yet processed. After selecting appointment tab, the doctor can see the details of the patient and the reports of his/her previous appointments which are retrieved from MedicalRecord's contract. Now the doctor has two options, the first option is, he can give treatment directly by assessing the patient without needing for any laboratory tests. In this case, doctor enters the problem of the patient, treatment and any other necessary details. This information is stored as an instance of MedicalRecord's contract and is stored in blockchain after the doctor finalizes and submits the final report. The second option is that if doctor feels that any lab tests are required to diagnose the patient's

health condition, he/she can do so by assigning lab tests. These lab tests are stored as an instance of LabAppointment's contract and is deployed in blockchain. Now, after corresponding lab technician updates the information of lab tests, the doctor can access those records from lab test tab, provides treatment and submits the final report which is finally stored in the blockchain as stated above.

3.1.4 Lab technician

Lab technician view consists of all the lab tests assigned to them by all doctors. He/she can perform the lab test by clicking on each lab tests which are obtained from LabAppointment's contract using 'RetrivePatientLabAppointment' function. He/she can enter all the required fields depending on the test assigned by the doctor and submits lab report.

3.2 Smart contracts of the Application

In this section, we discusses the various smart contracts that are implemented in the application.

- Patient Contract:
  This contract helps in storing general details of patient like name, email, age etc and functions for storing and retrieving the patient details.

- Doctor Contract:
  This contract helps in storing the details of doctor like name, email, specialization etc and contains functions for storing and retrieving the information of doctor.

- LabTechnician Contract:
  This contract helps in storing the details of the lab technicians and include functions for updating and retrieving their information.

- Appointment Contract:
  This contract includes the list of attributes required for booking an appointment with the doctor like doctor id, patient id, date of appointment, problem descrip-

tion etc and contains the functions for updating the appointment information and retrieving the appointment details.

- LabAppointment Contract:
  It is similar to the appointment's contract but contains attributes for the lab tests assigned by the doctor.

- PatientAppDetails Contract:
  This contract contains the attributes to store the summary of patient medical record which includes the problem of the patient and the treatment given by the doctor.

3.3 Workflow of the application

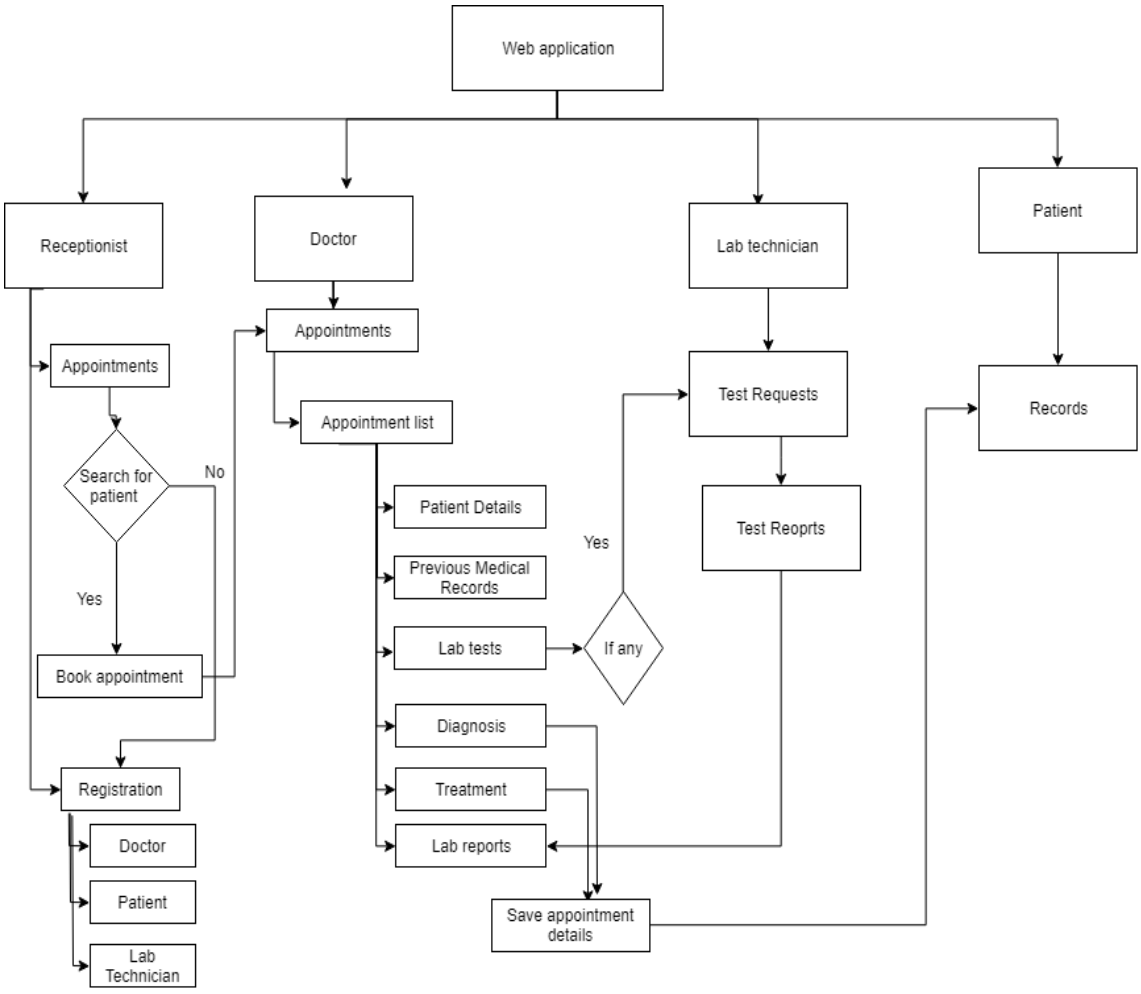

Fig. 3.1: **Workflow of the application**

Figure 3.1 describes the workflow of the application. When the application is started, it opens the home page that contains the information about the application and contains a login button. The login button is used to login to the application. For each login id there are set of roles that are assigned. Based on the roles assigned to the login id, appropriate page is opened.

If the receptionist role is assigned then the receptionist page is opened. The receptionist page contains links for registering patient, doctors and lab technicians, booking appointments. It also displays the names of the registered patients for quick booking. In the registration tab, the receptionist needs to enter the appropriate details for successful registration. The receptionist can book appointment using the patient id and doctor id. Once the booking is conformed, those appointments are updated in doctors tab.

If the doctor role is assigned then the doctor page is opened. The doctor page contains the appointment tab. In the appointment tab, contains the list of pending appointments for that day. When the doctor selects the 'view' button form the list, it is redirected to the page which contains the details of the appointment. This page contains the information of the patient and the previous medical records of the patient. The doctor can assign the lab test from this page, if assigned then details of the tests are updated in the labtechnician page. The final report contains the prescription details and the details of patient problems.

If the lab technician role is assigned then the lab technician page is opened. In this page contains the pending requests assigned to the patient by doctor. The lab technician updates the reports after the tests have completed. Once the reports are submitted, then those changes are reflected in the appropriate appointments.

If the patient role is assigned then the patient page is opened. In this page contains the medical records of the patient. He/She can only look at the medical records and the lab reports and doesn't have the permission to edit them.

3.4 Advantages in using blockchain over database

- Blockchain itself is secured because tampering with the blockchain will change the state of chain and it does not allow it. Whereas in database, we need to have a extra protection layer for safely securing the data.
- The transaction data will be hashed while storing in the blockchain, which will be quite difficult for understanding.
- The data in the blockchain can be accessed using smart contracts. Each smart contract in the system is executed for a particular event. Smart contracts can be changed once they are deployed. They act as the rules for the working of the application.

3.5 Summary

In this chapter, we discussed the different components and their functions present in our application, the smart contracts of the application and the workflow of the application. We also discussed the advantages of using blockchain over the traditional database.

Chapter 4

Implementation and Results

This chapter is divided into five sections. Section 1 presents the different softwares and tools used in implementation of the application. Section 2 describes the implementation of the application modules. Section 3 describes the working of application. Section 4 compares the proposed application with the database based application. Section 5 explains the challenges faced during the implementation of the application.

4.1 Softwares and Tools Used

- **Remix**: It is a web browser based IDE that allows you to write Solidity smart contracts, then deploy and run the smart contract. It is used for compiling, deploying and testing the smart contracts.
- **Ganache**: It is used for setting up personal ethereum blockchain, which can be used to run private blockchain, for testings, execute commands and inspect state while controlling how the chain operates.
- **Metamask**: It is a bridge that allows you to visit the distributed web in the browser. It allows you to run Ethereum dApps right in the browser without running a full Ethereum node.
- **Truffle**: It is a world class development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM).
- **Flask**: It is a micro web framework written in python for developing applications.

4.2 Implementation of the application modules

For the working of the application, there are two different components implemented i.e. the smart contracts and the JavaScript code for each entity that uses the smart contracts.

4.2.1 Smart contracts

All the smart contracts that are implemented in solidity 0.5.x version.

```
pragma solidity ^0.5.0;
pragma experimental ABIEncoderV2;
//import "github.com/Arachnid/solidity-stringutils/strings.sol";

contract Patient{
    struct patient{
        string name;
        string email;
        string gender;
        string birthday;//or struct birthda
        uint contact;
        bool maritalStatus;
        string Address;
        uint id;
    }
    uint Id=0;
    mapping(uint => patient) public Mappings;
    function RegisterPatient(string memory name, string memory email, string memory gender,
                             string memory birthday, uint contact, bool maritalStatus,
                             string memory Address) public returns(uint){
                                 Id = Id+1;
                                 Mappings[Id].name = name;
                                 Mappings[Id].email = email;
                                 Mappings[Id].gender = gender;
                                 Mappings[Id].birthday = birthday;
                                 Mappings[Id].contact = contact;
                                 Mappings[Id].maritalStatus = maritalStatus;
                                 Mappings[Id].Address = Address;
                                 Mappings[Id].id = Id;
                                 return Id;
                             }
    function RetrivePatientDetails(uint key) public view returns(string memory, string memory,
                                    string memory, string memory, uint, bool,
                                    string memory, uint){
                                        patient memory p = Mappings[key];
                                        return(p.name, p.email, p.gender,
                                         p.birthday,p.contact, p.maritalStatus,
                                         p.Address, p.id);
                                    }
    function GetPatients() public view returns(uint){
        return Id;
    }
}
```

Fig. 4.1: **Patient Contract**

Figure 4.1 shows the smart contract for storing the information of the patient. The implementation of the project is done in localhost. The patient details are stored in a structure. Each patient is given an ID. Whenever receptionist registers the patient, the data of the patient is mapped using the ID. The functions included in this contract are the 'RegisterPatient' function - which adds the patient details to the contract, 'RetrivePatientDetails' function - helps in retrieving the details of the patient using the patient ID, 'GetPatients' function - returns the number of patient that are registered.

```
pragma solidity ^0.5.0;
contract Doctor{
    struct doctor{
        string name;
        string email;
        string gender;
        uint contact;
        string department;
        string Address;
        uint Id;
    }
    uint Id=0;
    mapping(uint => doctor) public Mappings;
    function RegisterDoctor(string memory name, string memory email, string memory gender,
                            uint contact,string memory department, string memory Address) public returns(uint){
                                Id = Id+1;
                                Mappings[Id].name = name;
                                Mappings[Id].email = email;
                                Mappings[Id].gender = gender;
                                Mappings[Id].department = department;
                                Mappings[Id].contact = contact;
                                Mappings[Id].Address = Address;
                                Mappings[Id].Id = Id;
                                return uint(Id);
                            }
    function RetriveDocotorDetails(uint key) public view returns(string memory, string memory,
                                    string memory, uint, string memory,
                                    string memory, uint){
        doctor memory d=Mappings[key];
                                        return(d.name, d.email, d.gender,
                                         d.contact, d.department,
                                         d.Address, d.Id
                                            );
                                    }
    function GetDoctors() public view returns(uint){
         return Id;
    }

}
```

Fig. 4.2: **Doctor Contract**

Figure 4.2 shows the smart contract for storing the information of the Doctor. Similar to the patient contract, this contract stores the doctor details in a structure. Each doctor is given an ID. Whenever a new doctor is registered, the data of the doctor is mapped using the ID. The functions included in this contract are the 'RegisterDoctor' function - which adds the doctor details to the contract, 'RetriveDoctorDetails' function - helps to retrieve the details of the patient using the patient ID, 'GetDoctors' function - returns the number of doctors that are registered.

```
pragma solidity ^0.5.0;
contract LabTech{
    struct labtech{
        string name;
        string email;
        string gender;
        uint contact;
        string department;
        string Address;
        uint Id;
    }
    uint Id=0;
    mapping(uint => labtech) public Mappings;
    function RegisterLabtech(string memory name, string memory email, string memory gender,
                             uint contact,string memory department, string memory Address) public returns(uint){
                                 Id = Id+1;
                                 Mappings[Id].name = name;
                                 Mappings[Id].email = email;
                                 Mappings[Id].gender = gender;
                                 Mappings[Id].department = department;
                                 Mappings[Id].contact = contact;
                                 Mappings[Id].Address = Address;
                                 Mappings[Id].Id = Id;
                                 return uint(Id);
                             }
    function RetriveLabtechDetails(uint key) public view returns(string memory, string memory,
                                    string memory, uint, string memory,
                                    string memory, uint){
        labtech memory d=Mappings[key];
                                        return(d.name, d.email, d.gender,
                                         d.contact, d.department,
                                         d.Address, d.Id
                                            );
                                    }
    function GetLabtech() public view returns(uint){
         return Id;
    }
}
```

Fig. 4.3: **Lab Technician Contract**

Figure 4.3 shows the smart contract for storing the information of the lab Technician. Similar to the patient and doctor contract, the lab technician details are stored in a structure. Whenever a receptionist tries to register a new lab technician, an ID will be generated and assigned to the lab technician. The data of the lab technician is mapped using the ID. The functions included in this contract are the 'RegisterLabTech' function - which adds the lab technician details to the contract, 'RetriveLabtechDetails' function - helps in retrieving the details of the labtechnician using the patient ID, 'GetLabtech' function - returns the number of lab technicians that are registered.

```
pragma solidity ^0.5.0;
contract Appointment{
    struct app{
        uint id; //Patient Id
        string email; //Patient Email
        string gender;
        uint contact;
        string problem;
        string date;
        uint Doct; // Doctor Id
        uint status;
        uint Id; //Id of the transaction
    }
     uint Id=0;
    mapping(uint => app) public Mappings;
    function RegisterPatientAppintment(uint id, string memory email, string memory gender,
                        uint contact, string memory problem, string memory date,
                         uint Doct) public returns(uint){
                             Id = Id+1;
                             Mappings[Id].id = id;
                             Mappings[Id].email = email;
                             Mappings[Id].gender = gender;
                             Mappings[Id].contact = contact;
                             Mappings[Id].date = date;
                             Mappings[Id].problem = problem;
                             Mappings[Id].Doct = Doct;
                             Mappings[Id].Id = Id;
                             Mappings[Id].status = 0;
                             return uint(Id);
                         }
    function RetrivePatientAppointment(uint key) public view returns(uint , string memory , string memory ,
                          uint, string memory, uint, string memory, uint, uint){
                    app memory a=Mappings[key];
                                return(a.id, a.email,a.gender,
                                   a.contact, a.problem, a.Doct, a.date, a.Id, a.status);
                                }
    function UpdatePatientAppointment(uint key) public returns(uint){
        Mappings[key].status=1;
        return key;
    }
    function GetAppointments() public view returns(uint){
         return Id;
    }
```

Fig. 4.4: Appointment Contract

Figure 4.4 shows the smart contract for storing the details of the appointment that are created by the receptionist. It stores the patient Id, doctor Id, date, etc. For each appointment generated by receptionist, an Id is assigned and the details are mapped using the Id. The 'RegisterPatientAppointment' function - helps to store and maps the appointment details using the appointment Id. The 'RetrievePatientAppoinment' function - retrieves the details appointment. The 'UpdatePatientAppointment' function updates the status of the appointment.

```
pragma solidity ^0.5.0;
contract LabAppointment{
    struct app{
        uint Pid; //Patient Id
        string Desc;
        string date;
        uint Doct; // Doctor Id
        uint status;
        uint no_of_tests;
        uint Id; //Id of the transaction
    }
     uint Id=0;
    mapping(uint => app) public Mappings;
    function RegisterPatientLabAppointment(uint id,  string memory Desc, string memory date,
                            uint Doct, uint no_of_tests) public returns(uint){
                                Id = Id+1;
                                Mappings[Id].Pid = id;
                                Mappings[Id].date = date;
                                Mappings[Id].Desc = Desc;
                                Mappings[Id].Doct = Doct;
                                Mappings[Id].Id = Id;
                                Mappings[Id].no_of_tests = no_of_tests;
                                Mappings[Id].status = 0;
                                return uint(Id);
                            }
    function RetrivePatientLabAppointment(uint key) public view returns(uint , string memory , string memory ,
                             uint, uint, uint, uint){
                        app memory a=Mappings[key];
                                    return(a.Pid, a.date, a.Desc, a.Doct, a.Id, a.status, a.no_of_tests);
                            }
    function UpdatePatientLabAppointment(uint key) public returns(uint){
        Mappings[key].status=1;
        return key;
    }
    function GetLabAppointments() public view returns(uint){
         return Id;
    }
}
```

Fig. 4.5: **Lab Appointment Contract**

Figure 4.5 shows the smart contract for storing and processing lab appointments. It includes the functions RegisterPatientLabAppointment, RetrievePatientLabAppointment, UpdatePatientLabAppointment and GetLabAppointments.

```
pragma solidity ^0.5.0;

contract patientdata
{
    struct patientData
    {
        uint id;
        string data;
        uint ide;
    }
       uint Id=0;

    mapping(uint => patientData) public Mappings;

    function EnterPatientAppData(uint id, string memory data) public returns(uint){
                                Id = Id+1;
                                Mappings[Id].id = id;
                                Mappings[Id].data = data;
                                Mappings[Id].ide = Id;
                                return uint(Id);
                            }

    function RetrivePatientAppData(uint key) public view returns(uint , string memory, uint){
                        patientData memory a = Mappings[key];
                                    return(a.id, a.data, a.ide);
                                }

    function GetPatientAppData() public view returns(uint){
         return Id;
    }

}
```

Fig. 4.6: **Patient Appointment details contract**

Figure 4.6 shows the smart contract for storing the information of the patient records. The variable data contains all the details of the records in the form of a string. The 'EnterPatientAppData' function helps to store the appointment details of patient. The 'RetrivePatientAppData' retrieves the data of the appointments. The

'GetPatientAppData' returns the number of appointments.

When contracts are deployed in the blockchain, each contract has an ABI(Application Byte Interface) and is assigned an address . Using the ABI code of the contract and the address of the contract, we can access the contract functions.

4.2.2 Implementation of entities

The application is implemented using flask - a python framework. This section describes all the parts of the code that helps in working of the application. The language used to implement the each entity is JavaScript. We used a module known as web3.js which includes the functions that helps in accessing the smart contracts in the blockchain. Flask is used to connect the different web pages and it also helps in passing the data from one page to another.

We are using the localhost for the implementation of the application and database for storing the login details of the users. The database contains two tables, one that stores the login information of the patient and the role id. The other tables contains the role id and the different roles that are assigned to the user. The user can take four different roles, they are receptionist, doctor, lab technician and patient.

Before working with the implementation, metamask should be installed in the browser and ganache should be up and running. The metamask should be connected to the RPC server that is shown in ganache. By doing so, metamask links the web application to the blockchain network.

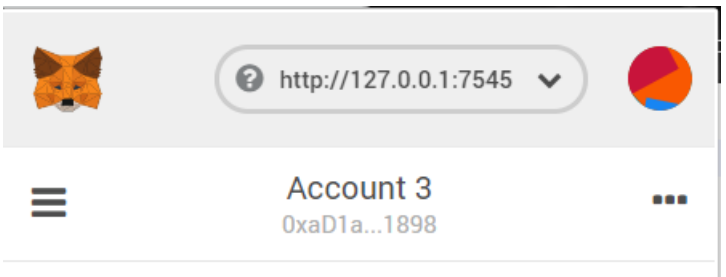

Fig. 4.7: **Metamask testing network connection**

Figure 4.7 shows that the metamask is connected to the 127.0.0.1:7545, a localhost server that runs the blockchain. This can be done by using custom RPC option in the metamask.

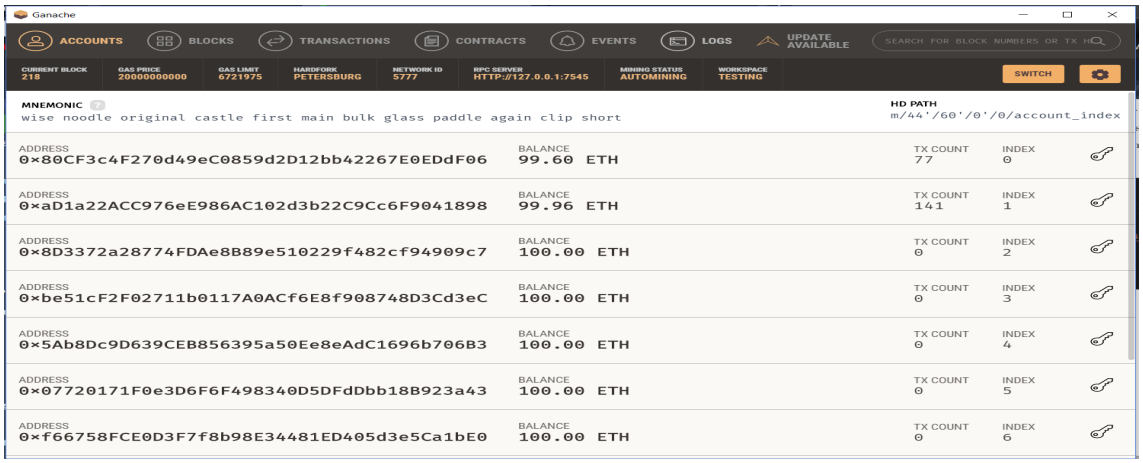

Fig. 4.8: **Ganache Interface**

Figure 4.8 shows the different accounts that the ganache provides which are used for deploying smart contracts in the blockchain and for testing them. Each account is provided with 100 ether from the beginning. So, testing the application should be easier. The ganache does not provide any proof of work mechanism. If the application needs to be tested using proof-of-work mining algorithm then there are different ethereum test networks that can be used to deploy and test the application in real time networks.

Figure 4.8 also shows the RPC server and the link. This link is used for configuring the web3js and connecting the metamask to the blockchain. The ganache has blocks and transactions tab that tracks all the transactions happened in the blockchain.

```
var key;
    window.onload = function () {
        if (typeof web3 !== 'undefined') {
            web3 = new Web3(web3.currentProvider);
            key = web3.eth.accounts[0];
        } else {
            // set the provider you want from Web3.providers
            web3 = new Web3(new Web3.providers.HttpProvider("http://localhost:7545"));
            key = web3.eth.accounts[0];
        }
}
```

Fig. 4.9: **Javascript code for configuring web3**

Figure 4.9 shows the piece of code used to configure the web3js. As mentioned before that the application implementation is done in using localhost, we provide the localhost link to define the web3 instance.

```
var abi1 =[ ···
    var contractaddress1 = '{{ contract_address[1] }}';
    var myAbi1 = web3.eth.contract(abi1);
    var myfunction1 = myAbi1.at(contractaddress1);
```

Fig. 4.10: Javascript code for creating the instance of contract

In order to access the contract, we need the ABI code of the contract and the contract address. Using those two, we connect to the contract using the web3 instance that was defined. Implementation for accessing the contract is shown in the Figure 4.10. myfunction acts as the instance to the contract. Using myfunction, we access the functions in the contract.

The code present in Figure 4.9 and Figure 4.10 are used in every module for configuring and accessing the blockchain and the smart contracts in the network.

Once the contract instance is created, we can use two main functions i.e. sendTransaction, call. The sendTransaction function is used to access the smartcontract and update the values in the contract. Whenever a sendTransaction call is made, metamask pop ups the transaction interface. The transaction interface contains the details of the ether that is required to perform the transaction. When the user clicks accept, then the transaction is made and the values are updated otherwise, the values are not updated and the transaction is rejected.

Now we describes implementation of each entity. We also discusses the working procedure of the code. The code is implemented in javascript. As discussed in the previous section, myfunction is the instance of the contract. In the coming sections.we look at different contract instances. The names used for contract instances are described below:

- myfunction - instance to the patient contract.

- myfunction1 - instance to the doctor contract.
- myfunction2 - instance to the appointment contract.
- myfunction3 - instance to the patient appointment details contract.
- myfunction4 - instance to the lab technician contract.
- myfunction5 - instance to the lab appointments contract.

1. **Receptionist Module**:

   The first thing to be done is, it should load available patients and doctors in the receptionist's dashboard page. This is accomplished by using the function GetPatientData and Doctors.

```
function GetPatientData() {
myfunction.GetPatients.call(function (error, result) {
    if (!error) { ···
     else
     alert(error);
   }
 )
}
```

Fig. 4.11: **JS code for getting patient details**

The code in the Figure 4.11 retrieves the patient details from the contract. Here we use call function to read the data. It does not cost any ether.

```
function Doctors(){
  myfunction1.GetDoctors.call(function (error, result) {
    if (!error) { ···
     else
     alert(error);
   }
);
}
```

Fig. 4.12: **JS code for getting doctor details**

The code in the Figure 4.12 retrieves the doctor details. myfunction1 is the instance of doctor contract. So, we use call function to get the doctor details.

```
function RegPatient() {
 var p = document.getElementById('id01');

  p.style.display = "none";

  var p1 = document.getElementById('pname').value;
  var p2 = document.getElementById('pemail').value;
  var p4 = document.getElementById('padd').value;
  var p5 = document.getElementById('pcity').value;
  var p6 = document.getElementById('pstate').value;
  var p7 = document.getElementById('pcontact').value;
  var p8 = document.getElementById('psex').value;
  var p9 = ""+document.getElementById('pbd').value;
  var p10 = document.getElementById('pmar').value;
  var p12 = p4+"-"+p5+"-"+p6;

myfunction.RegisterPatient.sendTransaction(
   p1, p2, p8, p9, p7, p10, p12, function(error, result){
    if(!error) {alert(result);}
   });
 }
```

Fig. 4.13: **JS code for Registration of patient**

In the Figure 4.13 the details of the patient from the patient registration form is captured and is sent as arguments to the patient contract. Here we used the function sendTransaction. This initiates the transaction, then metamask displays the information of the transaction and the cost of the ether required. When the transaction is accepted, the request is processed and added to the blockchain. If the transaction is rejected then the transaction is not processed and the data is not updated. Doctor and Lab technician registration are implemented in the similar way, they use myfunction1 and myfunction4 contract instances for updating the data.

```
function bookAppointment(){
  var p = document.getElementById('id00');
  p.style.display = "none";
  var p1 = document.getElementById('Bpid').value;
  var p9 = document.getElementById('Bpprob').value;
  var p10 = document.getElementById('doctor').value;
  var d=new Date();
  var p11 = ""+d.getDate()+"-"+(d.getMonth()+1)+"-"+d.getFullYear();
  myfunction.RetrivePatientDetails.call(p1, (error, result) => {

  myfunction2.RegisterPatientAppintment.sendTransaction(
    p1,result[1],result[2],result[4],p9,p11,p10, function(error, value){
      if(!error) {}
      else alert(error);
    });
  });
}
```

Fig. 4.14: **JS code for booking appointment**

In the code shown in Figure 4.14, the only inputs from the appointment page are

the patient Id, doctor Id and the patient problem. Using the patient Id, we get the information of the patient. Using this patient's information and other data from the web page, a transaction is created for updating the appointments in the blockchain.

2. **Doctor Module**:

   When the doctor is page is opened, the doctor selects the appointments tab and is redirected to the appointments page. When the appointment page is opened, it should show the appointments of the patients on that day.

```
function GetDoctorAppointments(){
var no_of_patients;
var no_of_doctors;
var patient_id = [];
var patient_name = [];
myfunction2.GetAppointments.call((error, result) => {
    if (!error) {
      no_of_patients = result;

      for (var i = 0; i < no_of_patients; i++) {
        myfunction2.RetrivePatientAppointment.call(i + 1, function (error, results) {
          if (!error) {
            if (results[8] == 0 && results[5] == Number.parseInt({{docId}},10)){
              patient_id = [results[0], results[5]];
              myfunction.RetrivePatientDetails.call(patient_id[0], function (error, resultss) {
                if (!error) {
                  myfunction1.RetriveDocotorDetails.call({{docId}}, function(error, res){
                    patient_name = [resultss[0], resultss[7]];
                  var table = document.getElementById('myTable');
                  var row = table.insertRow(1);
                  var cell1 = row.insertCell(0);
                  var cell2 = row.insertCell(1);
                  var cell3 = row.insertCell(2);
                  var cell4 = row.insertCell(3);
                  var cell5 = row.insertCell(4);
                  cell1.innerHTML = patient_name[0];
                  cell2.innerHTML = res[4];
                  cell3.innerHTML = results[6];
                  cell4.innerHTML = results[3];
                  cell5.innerHTML = '<a class="vap" style="vertical-align:middle" href="/Patient_appointment_details/';
                  cell5.innerHTML += results[7] + '"><span>view</span></a>';
                  });
                }
                else {···
              });
            }
          }
          else {···
    });
  }
    }
    else
      alert(error);
  });
}
```

Fig. 4.15: **JS code for retrieving appointments**

Figure 4.15 retrieves the appointments of the doctor on that particular day. We process all the appointments and check for the doctor id of each appointment, if the doctor id matches then the status of the appointment and the date is checked. Once they are matched, the patient details are retrieved and finally using all the details, the appointments are shown.

```
function Insert_data(){
//data = doctor_id + treatment + Diesease + Complaints + pre_eval + diag +date//+ labrec
kc={{ docId }};
var data=""+kc+"---"+Treatment+"---"+Diagnosis[0]+"---"+Diagnosis[1]+"---";
data+=Diagnosis[2]+"---"+Diagnosis[3]+"---"+date;
myfunction3.EnterPatientAppData.sendTransaction(
  Number.parseInt(patient_id,10), data, function(error, result) {
    if(!error){
      alert("Data Updation Success");
    }
    else{
      alert("Data Updation Unsuccessful:\n"+error);
    }
});
myfunction2.UpdatePatientAppointment.sendTransaction(appointment_key, function(error,result){
        if(!error){
          alert(result);
        }
        else{
          alert(error);
        }
      });
}

function add_report(){
  var table = document.getElementById('tbl_id');
  var row_len = table.rows.length;
  //var column = table.rows[0].cells.length;
  var data="";
  for (var i = 1; i < (row_len-1)*(2); i=i+2){
      var j= "lddep"+i;
      var k= "ldata"+(i+1);
      j = document.getElementById(j).value;
      k = document.getElementById(k).value;
      data += j+"/"+k +"---";
    }
    var d=new Date();
    var p11 = ""+d.getDate()+"-"+(d.getMonth()+1)+"-"+d.getFullYear();
    var no = row_len -1;
    myfunction5.RegisterPatientLabAppointment.sendTransaction(
      patient_id, data, p11, {{ docId }}, no,{{ appId }}, (error,result)=>{
        if(!error){
          alert("Lab tests booked");
        }else{alert(error);}
    });
}
```

Fig. 4.16: **JS code for updating patient records and adding lab results**

In the Figure 4.16, the function Insert_data will create the transaction for updating the patient details. The variable 'data', in the above image, combines all the details of the record and are made into a string. Whenever the appointment data gets updated then the status of the appointment in the contract is updated as well. The function add_report is executed when the doctor assigns lab tests to assess the patient's health.

```
myfunction3.GetPatientAppData.call(function(error, result){
    for(var i=0;i<result;i++){
        myfunction3.RetrivePatientAppData.call(i+1, (error,results) => {
            patient_id = Number.parseInt(patient_id,10);
            if(results[0] == patient_id){
                var data = results[1].split("---");
                // doc_id, tratment,  Diesease + Complaints + pre_eval + diag + date
                myfunction1.RetriveDocotorDetails.call(data[0], function(error,resultss){
                    if(Number.parseInt(data[0],10) == Number.parseInt({{ docId }},10)){
                        var table = document.getElementById('myTable');
                        var row = table.insertRow(1);
                        var cell1 = row.insertCell(0);
                        var cell2 = row.insertCell(1);
                        var cell3 = row.insertCell(2);
                        var cell4 = row.insertCell(3);
                        //var cell5 = row.insertCell(4);
                        cell1.innerHTML = resultss[0];
                        cell2.innerHTML = resultss[4];
                        cell3.innerHTML = data[6];
                        //cell4.innerHTML = results[3];
                        cell4.innerHTML = '<a class="vap" style="vertical-align:middle" href="/appointments_view_by_doctor/'
                        cell4.innerHTML += results[2] + '"<span>view</span></a>';
                    }
                });
            }
        });
    }
});
```

Fig. 4.17: **JS code for retrieving previous patient records and adding lab results**

Figure 4.17 retrieves all the previous records of the patient that are made with the doctor. Similarly, the lab reports are also retrieved and displayed in the lab report section.

3. **Patient Module**:

```
myfunction3.GetPatientAppData.call(function(error, result){
    for(var i=0;i<result;i++){
        myfunction3.RetrivePatientAppData.call(i+1, (error,results) => {
            patient_id = Number.parseInt(patient_id,10);
            if(results[0] == patient_id){
                var data = results[1].split("---");
                // doc_id, tratment,  Diesease + Complaints + pre_eval + diag + date
                myfunction1.RetriveDocotorDetails.call(data[0], function(error,resultss){
                    if(Number.parseInt(data[0],10) == Number.parseInt({{ docId }},10)){
                        var table = document.getElementById('myTable');
                        var row = table.insertRow(1);
                        var cell1 = row.insertCell(0);
                        var cell2 = row.insertCell(1);
                        var cell3 = row.insertCell(2);
                        var cell4 = row.insertCell(3);
                        //var cell5 = row.insertCell(4);
                        cell1.innerHTML = resultss[0];
                        cell2.innerHTML = resultss[4];
                        cell3.innerHTML = data[6];
                        //cell4.innerHTML = results[3];
                        cell4.innerHTML = '<a class="vap" style="vertical-align:middle" href="/appointments_view_by_doctor/'
                        cell4.innerHTML += results[2] + '"<span>view</span></a>';
                    }
                });
            }
        });
    }
});
```

Fig. 4.18: **JS code for retrieving patient records**

Figure 4.18 describes the Patient module which contains only his/her records.

The patient does not have permission to update their records. Patient can only view his/her records. The code in Figure 4.18 retrieves all the patient lab records and are sorted according to date. Similarly using the patient id, his/her profile details are retrieved and displayed.

After clicking view button on a particular record, the record id is passed to another web page with the help of flask, using that record Id, the record details are retrieved and displayed as whole.

4. **Lab Technician Module**:

```
myfunction4.RetriveLabtechDetails.call({{labId}}, (error, result) => {
  document.getElementById("Doc_name").innerHTML = "Name: "+result[0];
  document.getElementById("pname").innerHTML = "<b class='text-info'> Name: </b>   "+result[0];
      document.getElementById("pemail").innerHTML = "<b class='text-info'> EMail: </b>   "+result[1];
      document.getElementById("sex").innerHTML = "<b class='text-info'> Sex: </b>   "+result[2];
      document.getElementById("pbd").innerHTML = "<b class='text-info'> Date of Birth: </b>   "+result[4];
      document.getElementById("pcontact").innerHTML = "<b class='text-info'> Contact: </b>   "+result[3];
      var add = result[5].split("-");
      document.getElementById("padd").innerHTML = "<b class='text-info'> Address: </b>"+add[0]+"<br>    "
      document.getElementById("Padd").innerHTML += "     "+add[1]+"<br>         "+add[2];
});
```

Fig. 4.19: **JS code for retrieving pending lab records**

Figure 4.19 retrieves all the lab records that are yet to be performed. This is achieved by checking the status of all the records and filtering the records which are not yet processed.

Each record contains a view button. On clicking the button, it is redirected to the corresponding lab test page. In that page, the lab technician updates the results. Once the lab technician submits the test results, a transaction is made to update the data of the lab test and the status of the lab appointments.

4.3 Working of Application

In this section, we describe the working of the application in detail

4.3.1 Receptionist page

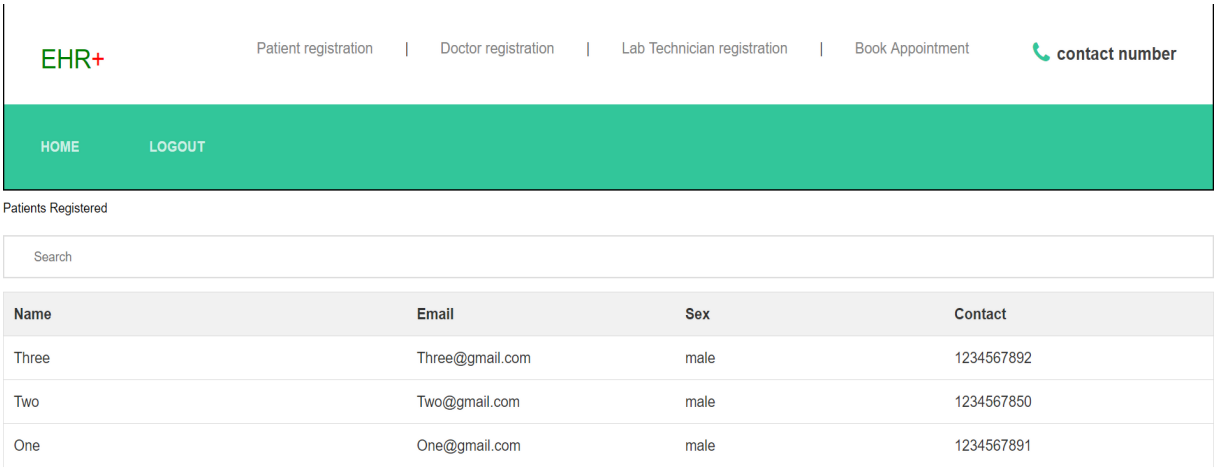

Fig. 4.20: Receptionist interface

Receptionist page contains links to patient registration, Doctor registration, Lab technician registration and book appointment. Each link opens up the respective interface when selected. Figure 4.20 shows the details of the registered patients and contains a search bar for searching the registered patient based on patient name. The header section of the receptionist page also contains the logout link which is used to exit the application.

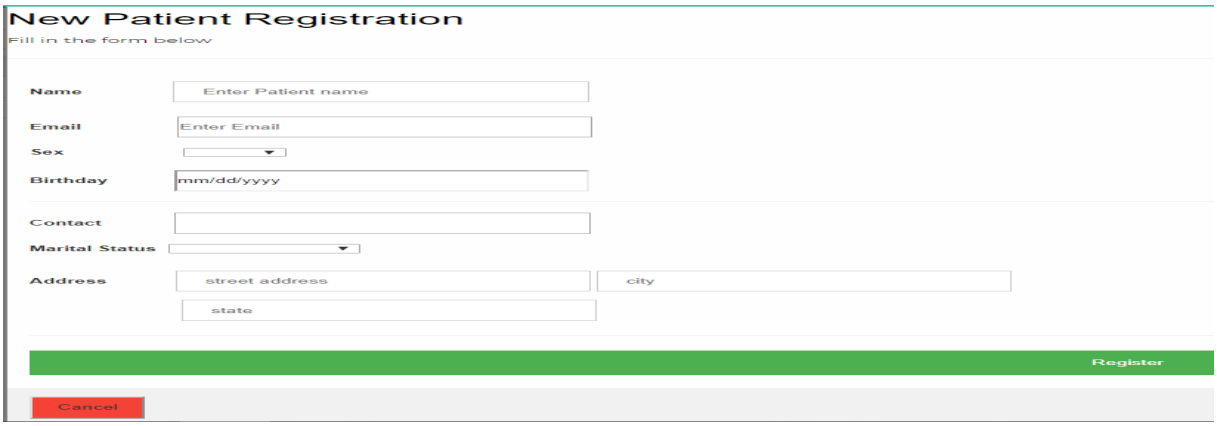

Fig. 4.21: Patient registration interface

Figure 4.21 shows the patient registration interface which contains various input fields so that receptionist enters all the details of the patient like name, email, gender, date-of-birth etc. Once, the receptionist clicks register button, patient information is stored in the blockchain.

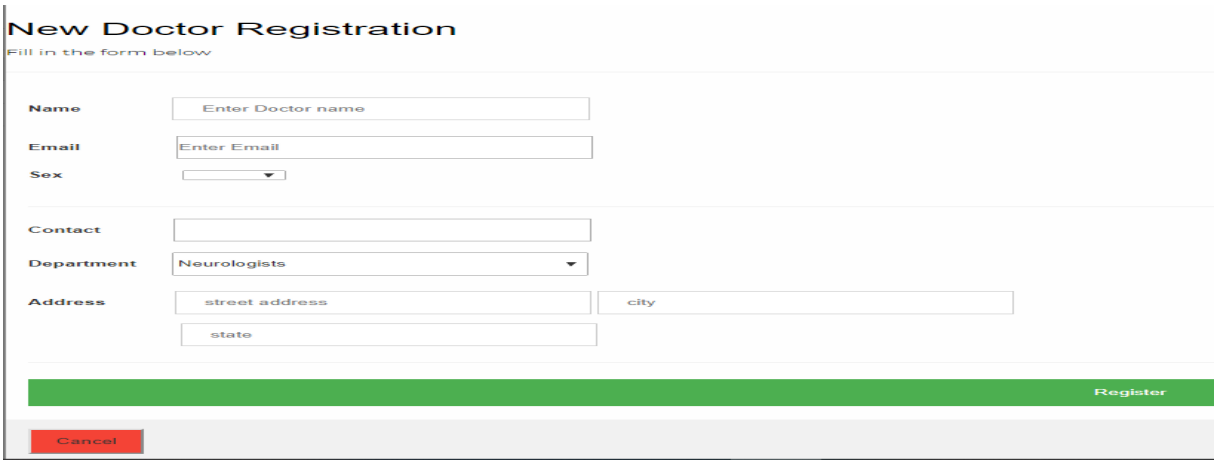

Fig. 4.22: **Doctor registration interface**

Figure 4.22 shows the doctor registration interface which contains input fields for entry of doctor information such as name, age, department, email etc. This data is stored in blockchain once the receptionist clicks register button.

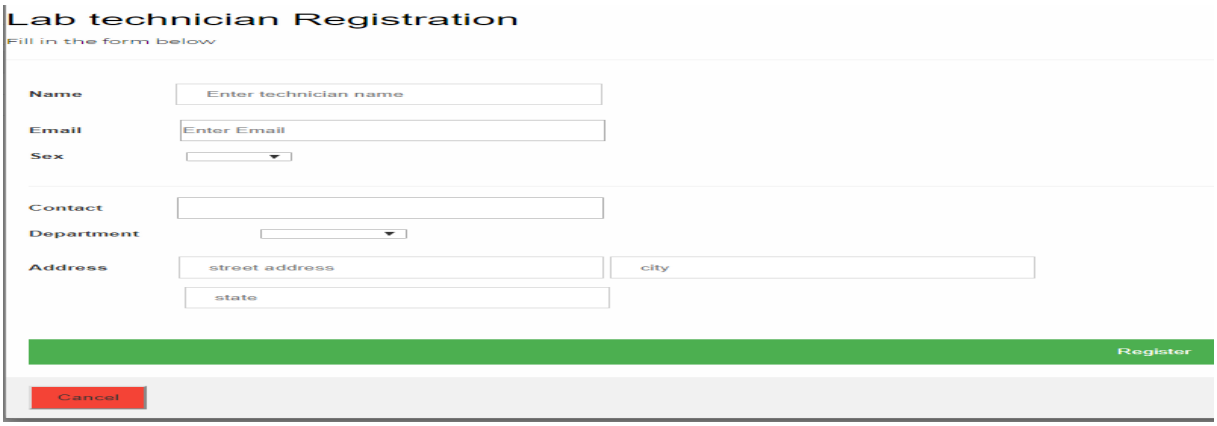

Fig. 4.23: **Lab Technician registration interface**

Similar to the patient and doctor registration interfaces, Figure 4.23 shows interface which includes the input fields for lab technician. Once the receptionist clicks register button, data is stored in blockchain.

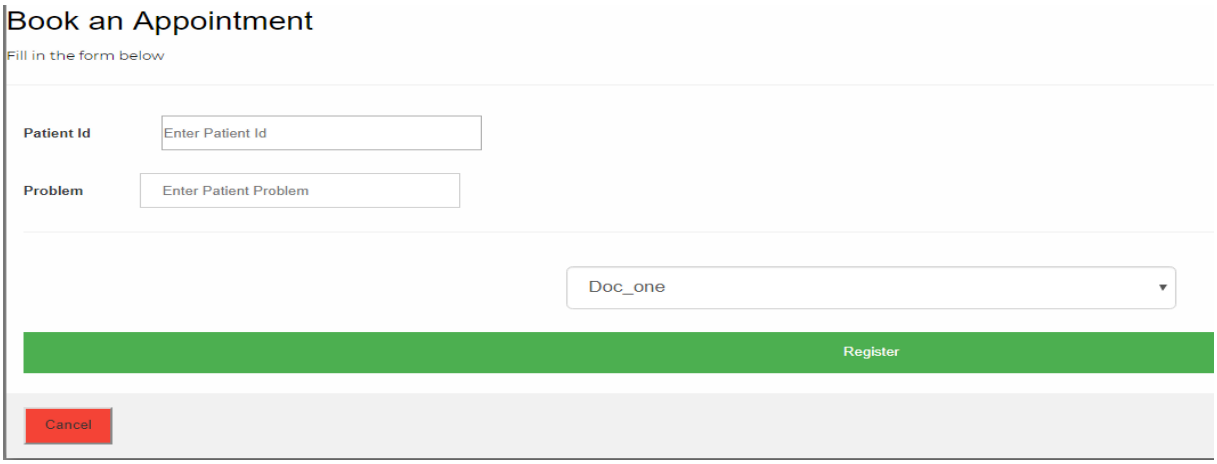

Fig. 4.24: Appointment creation page

Figure 4.24 shows the appointment creation page which contains input fields like the patient id, patient problem and the doctor name of a particular department based on the symptoms of the patient. After entering the required fields, the receptionist can book an appointment by clicking the register button.

4.3.2 Doctor Page

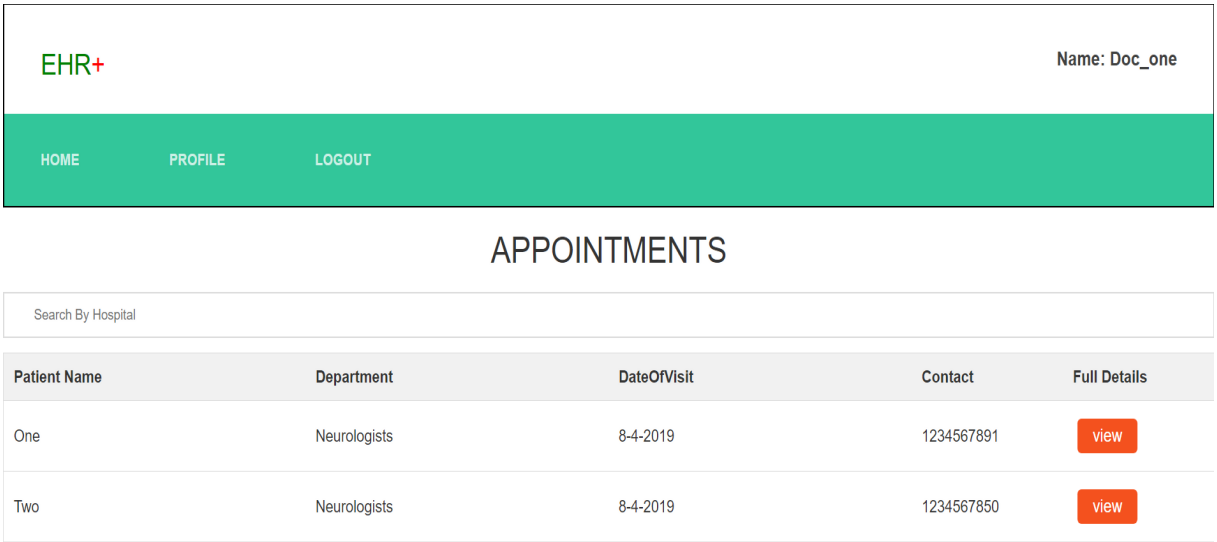

Fig. 4.25: Doctor interface

The header section of Doctor page contains links to profile which shows the profile details of the respective doctor and a logout option to exit from the application. Figure 4.25 also shows the data related to the appointments made by the receptionist to that

doctor. This appointment data contains details like patient name, department of the doctor, date of visit of the patient, contact number of the patient and a 'view' button. The view button of this appointment data helps to know the complete information regarding the patient in a separate page which are discussed below.

Fig. 4.26: **Patient details in appointment page**

After clicking the 'view' button on the appointments data in the doctor page, this page related a particular patient's appointment is opened. Figure 4.26 contains different tabs like patient tab, previous appointments tab, treatment tab, lab reports tab, lab tests tab, diagnosis tab, and a final report tab. Figure 4.26 shows the patient tab which contains the basic details of the patient.

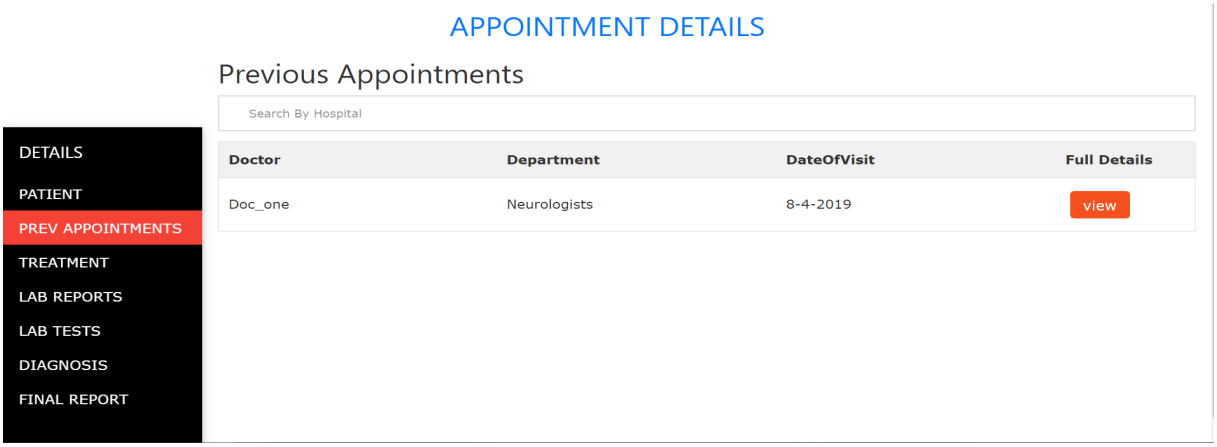

Fig. 4.27: **Patient previous appointments**

The previous appointment tab in Figure 4.27 shows the list of all previous appoint-

ments made by the patient to that doctor. For each previous appointment of the patient contains a 'view' button which shows final report regarding to that appointment.

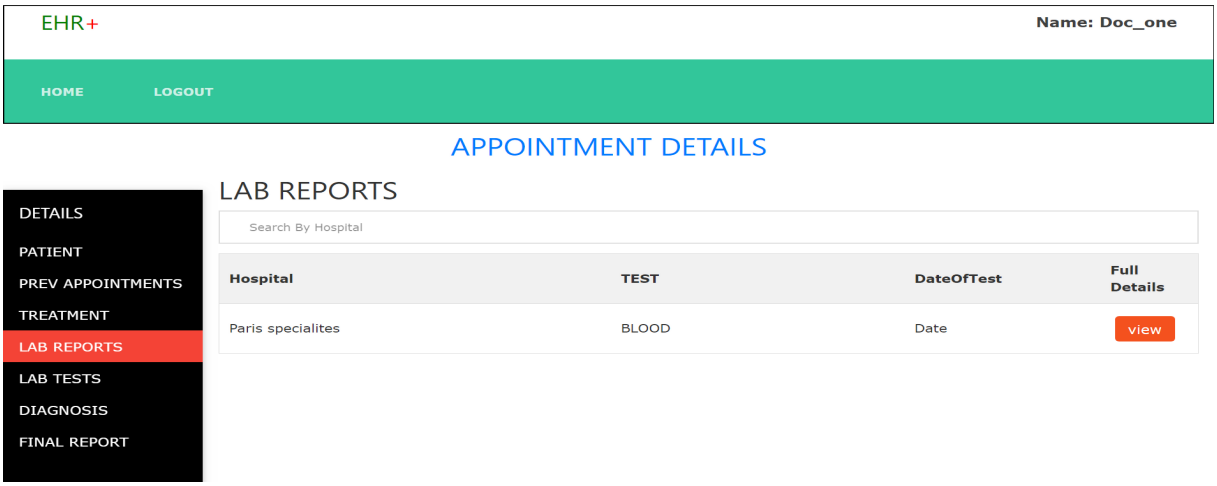

Fig. 4.28: **Patient Lab reports**

In the Lab reports tab shown in Figure 4.28, doctor can see the list of all lab reports which were completed by the lab technician. On clicking each lab report, the doctor can have a detailed look of the report.

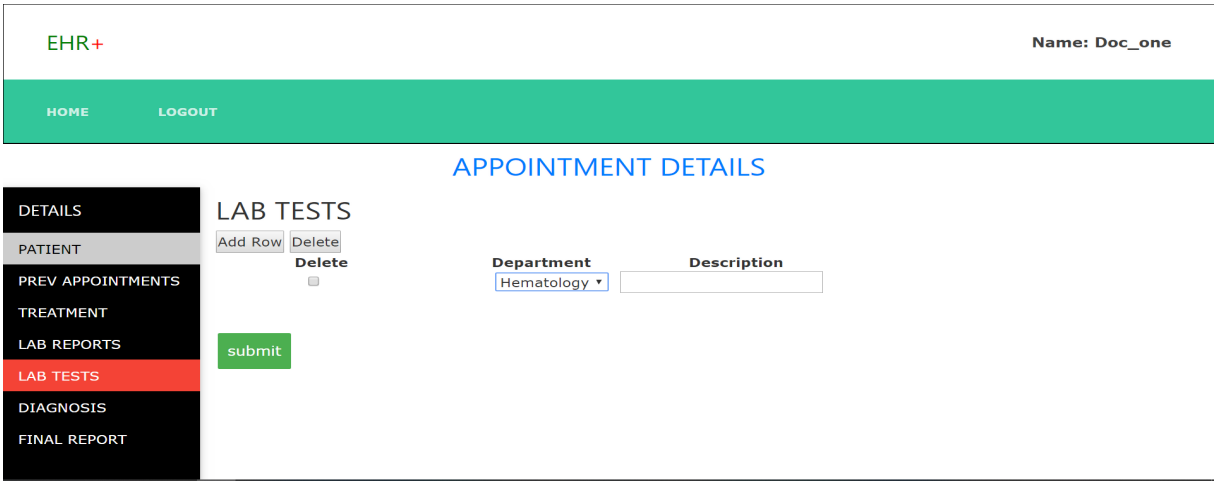

Fig. 4.29: **Lab test assignment**

If doctor feels any tests are required to diagnose the disease, he/she can do so by assigning the lab tests from "LAB TESTS" tab shown in Figure 4.29. Doctor can add or delete multiple lab tests by specifying the type of report and description for each test. Once the doctor clicks 'submit' button, all the requests are stored in blockchain

and sent into respective lab technicians for performing required tests.

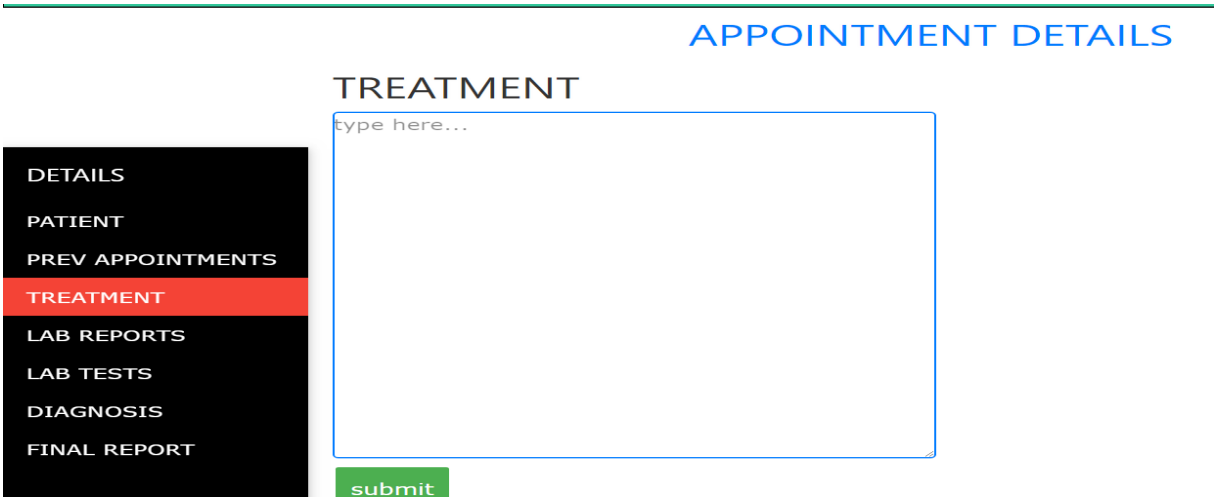

Fig. 4.30: **Treatment Page**

In the treatment tab shown in the Figure 4.30, doctor can prescribes the treatment based on the problems of the patient and lab tests he/she received (if any) and submits it.

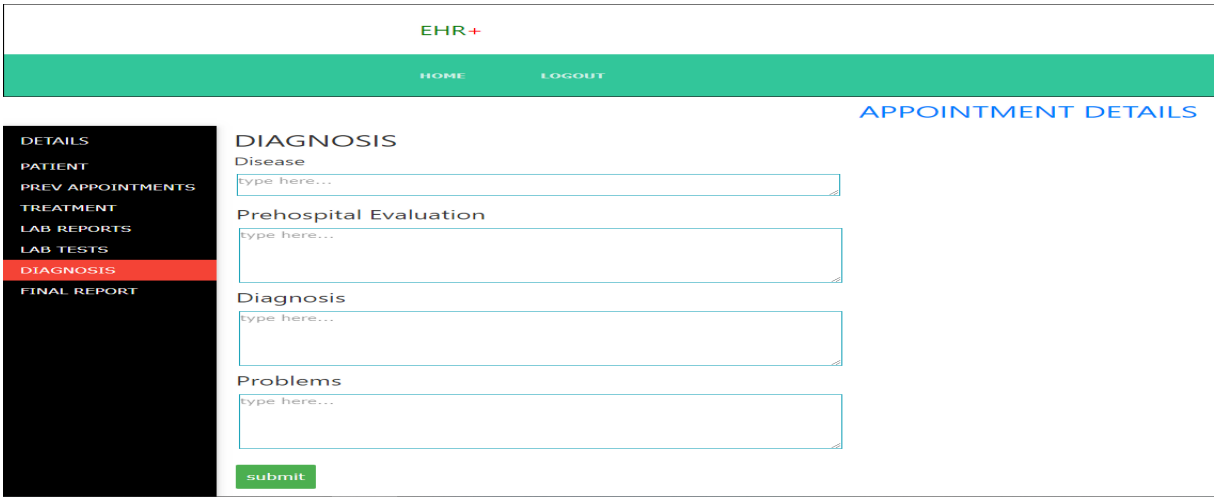

Fig. 4.31: **Diagnosis Page**

The diagnosis tab shown in Figure 4.31 contains input fields like disease of the patient, pre-hospital evaluation, diagnosis given by the doctor and the problems of the patient. After filling all the details, the doctor submits it which will be updated in final report.

Fig. 4.32: **Finial Report Page**

All the data entered in diagnosis tab and treatment tab are updated in the final report shown in Figure 4.32. Once the doctor finalizes the report, he submits it.

4.3.3 Patient Page

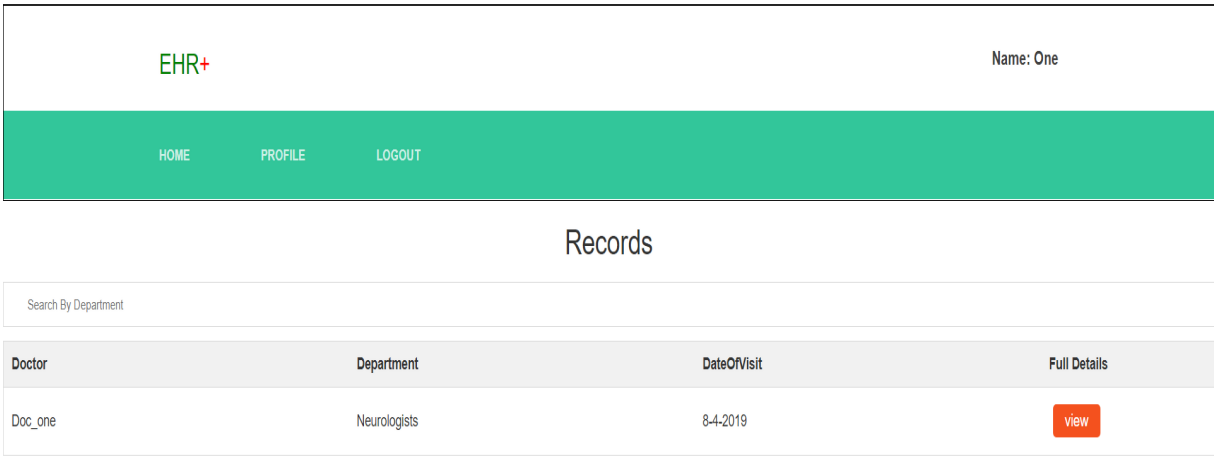

Fig. 4.33: **Patient Records**

Figure 4.33 shows the dashboard page of a patient. It contains the list of all medical records where patient can see the summary of the report by clicking on it. This page also includes links to the profile page and logout page.

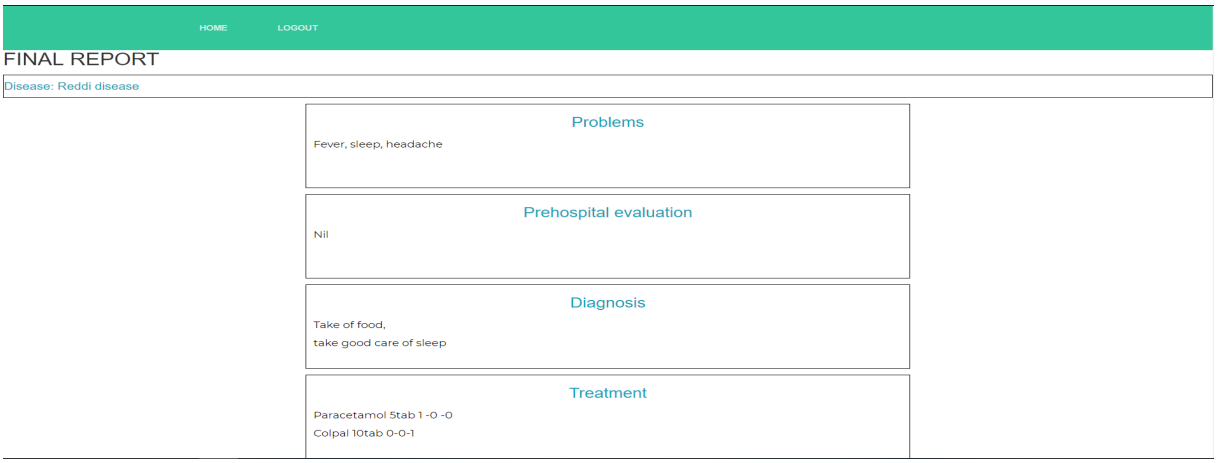

Fig. 4.34: Patient Record details

Figure 4.34 shows the interface for patient record details which is opened when patient clicks 'view' button of medical record. It contains the final report such as problems, diagnosis, treatment given by the doctor etc.

4.3.4 Lab Technician Page

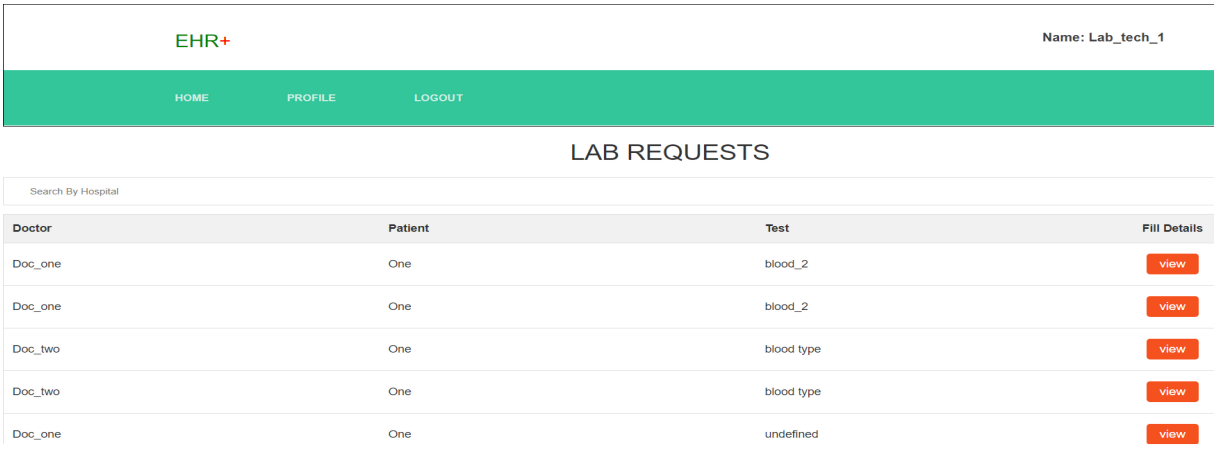

Fig. 4.35: Lab test requests

Figure 4.35 shows the dashboard page of the lab technician. It contains all the lab requests placed by a doctor for the patient. Each lab request includes doctor name, patient name and test to be performed and a view button. On clicking 'view' button, lab technician can enter the test information.

4.4 Comparison of proposed application with Database based application

- All the data regarding the application is stored in the blockchain.
- Once the data is entered in the blockchain, it can not be deleted unlike in the database where there is a possibility of the data to be erased.
- Multiple hospitals do not need to have their own servers because different hospitals can connect to the same blockchain and this blockchain acts as the universal database to all the users.
- All the data is stored distributed in the blockchain unlike in the database where it is kept at a single place.

4.5 Challenges

In this section, we discuss the challenges faced during the implementation of our application. They are listed as follows:

- Selection of appropriate tools and frame works which suits our application.
- Proper integration of the tools for effective working of the application.
- Passing data from one page to another.
- Selecting the right versions of solidity compiler and web3.
- Using JavaScript for accessing the smart contract through web3.

4.6 Summary

In this chapter, we explained the various tools and softwares used for developing the application. We have discussed the implementation of the smart contracts, entities present in the application and their working procedure. We have compared our application with the database based application. We also described the challenges faced during the implementation of the application.

Chapter 5

Conclusion and Future work

In this report, we have demonstrated patient medical record management application using the blockchain technology. We explained the entities i.e receptionist, doctor, patient and lab technician and their respective functions which participate in the process of storing medical records. The advantages of this system are that the patient can look at his/her medical records from anywhere. Once the medical record is generated, it can not be delete. The data is stored securely in the blockchain unlike in the traditional database which requires an extra layer for protecting the data. Our application is a plug-in type, where functions can be added or modified easily.

Our application is confined to a single hospital where the hospital stores their patient's data using blockchain technology, but this can be extended to multiple hospitals which solves the problem of interoperability that helps to transfer the patient data from one health care provider to another. This application can also be tested in the ethereum test networks like Rinkeby test network for real-time analysis and results.

References

Alhadhrami, Zainab, Salma Alghfeli, Mariam Alghfeli, Juhar Ahmed Abedlla, and Khaled Shuaib (2017), "Introducing Blockchains for Healthcare." In *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, 1–4.

Arlindo, F. da Conceic, Flavio S. Correa da Silva, Vladimir Rocha, Angela Locoro, and Joao Marcos M. Barguil (2018), "Eletronic Health Records using Blockchain Technology." *Research gate, INCT of the Future Internet for Smart Cities*.

Azaria, Asaph, Ariel Ekblaw, Thiago Vieira, and Andrew Lippman (2016), "MedRec: Using Blockchain for Medical Data Access and Permission Management." In *2nd International Conference on Open and Big Data, Institute of Electrical and Electronics Engineers*, 25–30.

Hiroki, Watanabe, Shigeru Fujimura, Atsushi Nakadaira, Yasuhiko Miyazaki, Akihito Akutsu, and Jay Kishigami (2015), "Blockchain Contract: A Complete Consensus using Blockchain." In *Institute of Electrical and Electronics Engineers 4th Global Conference on Consumer Electronics*, 577–578.

Kate, Barlow (2018), "The blockchain: The first step towards its healthcare transformation." URL `http://www.global-engage.com/life-science/the-blockchain-the-first-steps-toward-its-healthcare-transformation/`.

Krawiec, R J, Dan Housman, Mark White, Mariya Filipova, Florian Quarre, Dan Barr, Allen Nesbitt, Kate Fedosova, Jason Killmeyer, Adam Israel, and Lindsay Tsai (2016), "Blockchain: Opportunities for Health Care." URL `https://www2.deloitte.com/content/dam/Deloitte/us/Documents/public-sector/us-blockchain-opportunities-for-health-care.pdf`.

Linn, L. A. (2016), "Blockchain for health data and its potential use in health it and health care related research." *ONC/NIST Use of Blockchain for Healthcare and Research Workshop, Gaithersburg, MD, USA:ONC/NIST*.

Macdonald, Liu Thorrold, and Julien (2017), "The Blockchain: A Comparison of Platforms and Their Uses Beyond Bitcoin." *The University of Queensland, Research gate*.

Matt, Weiss, Dan Elitzer, and Joe Gerber (2015), "How bitcoins technology could reshape our medical experiences." URL `http://www.coindesk.com/bitcoin-technology-could-reshape-medical-experiences`.

Mertz, Leslie (2018), "Block Chain Reaction: A Blockchain Revolution Sweeps into Health Care, Offering the Possibility for a Much-Needed Data Solution." *IEEE Plus, Institute of Electrical and Electronics Engineers*, 4–7.

Mohanta, Bhabendu Kumar, Debasish Jena, and Soumyashree S Pand (2018), "An Overview of Smart Contract and Use cases in Blockchain Technology." *Institute of Electrical and Electronics Engineers, 9th ICCCNT*, 12–20.

Nakamoto, Satoshi (2009), "Bitcoin: A peer-to-peer electronic cash system." URL `https://bitcoin.org/bitcoin.pdf/`.

Niya, Sina Rafati, Florian Shupfer, Thomas Bocek, and Burkhard Stiller (2018), "Setting up Flexible and Light Weight Trading Contracts with Enhanced UserPrivacy Using Smart Contracts." *IEEE/IFIP Network Operations and Management Symposium, Institute of Electrical and Electronics Engineers*, 1–2.

Pinyaphat, Tasatanattakool and Techapanupreed Chian (2018), "Blockchain: Challenges and applications." In *International Conference on Information Networking, Institute of Electrical and Electronics Engineers*, 473–475.

Robert, Pearl (2018), "Blockchain, bitcoin and the electronic health record." URL `https://www.forbes.com/sites/robertpearl/2018/04/10/blockchain-bitcoin-ehr/`.

Rodelio, Arenas and Proceso Fernande (2018), "CredenceLedger: A Permissioned Blockchain for Verifiable Academic Credentials." In *International Conference on Engineering Technology and Innovation, Institute of Electrical and Electronics Engineers*.

Rodrigues, Bruno, Thomas Bocek, David Hausheer Andri Lareida, Sina Rafati, and Burkhard Stiller (2017), "A Blockchain-Based Architecture for Collaborative DDoS Mitigation with Smart Contracts." In *IFIP International Conference on Autonomous Infrastructure, Management and Security, Springer*.

Stagnaro, Chet (2018), "White paper: Innovative blockchain uses in health care." URL `https://www.freedassociates.com/wp-content/uploads/2017/08/Blockchain_White_Paper.pdf`.

Tom Joseph, Seymour, Dean Frantsvog, and Tod Graeber (2014), "Electronic Health Records (EHR)." *American Journal of Health Sciences, Research Gate*.

Zheng, Z., Xie. S, Dai. H, Chen. X, and Wang. H (2017), "An overview of blockchain technology: Architecture, consensus, and future trends." In *Proceedings of the 2017 IEEE Big Data Congress, Honolulu, Hawaii, USA*, 557–564.

Brief Bio-Data

Bandaru Bharath Kumar

B.Tech 4th year

Department of Computer Science and Engineering

National Institute of Technology Karnataka, Surathkal

Mobile: 8985669267

Email: bandarubharath7@gmail.com

Permanent Address

Bandaru Bharath Kumar

Rajahamsa Great Empire, Flat no:304

Shiridi Nagar, Ananthapur

Andhra Pradesh, 515001

Education

10^{th} - Narayana E-Techno School, Near APSRTC Bus-stand, Ananthapur, Andhra Pradesh, India, 2013. (9.3)

12^{th} - Narayana Junior College, Attapur, Hyderabad, Telangana, India, 2015. (98%)

(B. Tech) -

- 1 Sem - 8.05 (CGPA)
- 2 Sem - 8.21 (CGPA)
- 3 Sem - 7.87 (CGPA)
- 4 Sem - 7.66 (CGPA)
- 5 Sem - 7.71 (CGPA)
- 6 Sem - 7.84 (CGPA)
- 7 sem - 7.84 (CGPA)

Kenguva Ananth Kumar

B.Tech 4th year, 15CO124

Department of Computer Science and Engineering

National Institute of Technology Karnataka, Surathkal

Mobile: 9182842727

Email: ananthkenguva820@gmail.com

Permanent Address

Kenguva Ananth Kumar

Vivekananda Colony, Flat no:5

Vizianagaram

Andhra Pradesh, 535003

Education

10^{th} - Gurajada Public School, Vizianagaram, Andhra Pradesh, India, 2013. (9.8)

12^{th} - Sri Chaitanya Junior College, Vishakapatnam, Andhra Pradesh, India, 2015. (98.2%)

(B. Tech) -

- 1 Sem - 7.80 (CGPA)
- 2 Sem - 7.74 (CGPA)
- 3 Sem - 7.50 (CGPA)
- 4 Sem - 7.32 (CGPA)
- 5 Sem - 7.53 (CGPA)
- 6 Sem - 7.61 (CGPA)
- 7 sem - 7.70 (CGPA)

Palempalli Reddi Narasimha Prasad

B.Tech 4th year, 15CO133

Department of Computer Science and Engineering

National Institute of Technology Karnataka, Surathkal

Mobile: 9490161179

Email: reddinarasimha1998@gmail.com

Permanent Address

P Narasimhulu setty

H. No. 5/165-A

Penagalur village mandal and post

Rajampet, kadapa

Andhra Pradesh, 516127

Education

10^{th} - Raju High School, Rajampet, Kadapa, Andhra Pradesh, India, 2013. (9.8)

12^{th} - NRI Arts Sciences Jr College,New Balajy Colony,Tirupati,AP, India, 2015. (98.5%)

(B. Tech) -

- 1 Sem - 8.55 (CGPA)
- 2 Sem - 8.56 (CGPA)
- 3 Sem - 8.52 (CGPA)
- 4 Sem - 8.32 (CGPA)
- 5 Sem - 8.36 (CGPA)
- 6 Sem - 8.32 (CGPA)
- 7 sem - 8.31 (CGPA)