Computer Science Dept.
Faculty Of Computers And Information,
Mansoura University, Egypt.

Blockify

Your Health Comes First

Medical Record Management System Using BlockChain Technology

نظام لإدارة السجلات الطبية بإستخدام تقنية البلوك تشين

Under Supervision of :
Prof. Samir Elmougy
Dr. Sara El-Metwally

Medical Record Management System
Using BlockChain Technology

نظام لإدارة السجلات الطبية بإستخدام تقنية البلوك تشن

CS-04 04/07/2020

أعداد

أحمد مسعد يوسف
أحمد نجيب فيولة
اسلام أنور الحاكمي
أيمان عبد القادر ستو
سامية إيهاب الجندي
فريال إسماعيل إبراهيم
يارا السيد رمضان
احمد مدحت البسيوني
يوسف شريف البساطي
وليد ماهر حسن

أشراف

ا.د / سمير الموجي
د / سارة المتولي

Abstract

Blockify is the the trade name we choose for our project "medical record management system using blockchain technology"

Our project aims to solve the medical information sharing problem, between multiple hospitals (or any medical institutes) and the patients, in a unified global matter.

We use permissioned blockchain technology (Hyperledger fabric) to keep the patients' data private, secure and immune to falsification and corruption.

Through our system patients can access and share their medical records with anyone they permit, and hospitals can access the patients' data on emergencies, all without compromising the security or privacy of the data

Table of Contents

Table of Figures

Table of Abbreviations

CPU	central processing unit
CD-ROMS	Compact Disc Read-Only Memory
APP	Advocate Physician Partners
BCBS	Blue Cross Blue Shield
ZIP	Zone Improvement Plan
SDLC	System development life cycle
DFD	Data flow diagram
UML	Unified Modeling Language
ERD	Entity Relationship Diagram
ER	Entity Relationship
RDBMS	Relational Database Management System
PK	Primary Key
FK	Foreign Key
UI	User interface
UX	user experience
GUI	Graphical user interface
ID	Identity document
HSM	Hardware Security Module
PKCS	Public Key Cryptography Standard
OS	Operating system
API	Application Programming Interface
JSON	JavaScript Object Notation

SDK	Software Development Kit
RPC	Remote Procedure Calls
ECMA	European Computer Manufacturers Association
SQL	Structured Query Language
DB	database
JSON	JavaScript Object Notation
IBM	International Business Machines
DLT	distributed ledger technology
JS	Java script
npm	Node Package Manager
VS	Visual Studio
CLI	Command Line Interpreter
HLF	High-Level Format

Blockif

Your Health Comes First

Chapter 1

Introduction

1.1 Introduction

This chapter covers the scope of the project and gives the brief of the major topics .

According to a recent study by Johns Hopkins, more than 250,000 people in the United States die every year because of medical mistakes,The researchers caution that most of medical errors aren't due to inherently bad doctors. Rather, most errors represent lack of information regarding the patient and the inaccurate Syndrome taken from the patient before doctor's diagnosis [1].

Over 30 years ago Dr.Hampton and his colleagues suggested that the medical history of the patient determined 83% of the diagnoses [2].

Researchers evaluated the relative importance of the medical history and diagnostic studies, they found that between 70% to 90% of medical diagnoses can be determined by the history alone [3].

According to these three studies it is important for both patients and organizations(individual doctors,hospitals, pharmacists,clinics and laboratories) to get a secure and organized record of patient's medical history, which leads doctors to a timely and accurate diagnosis. This process protects patients from the risks of unnecessary testing and is cost-effective.

For illumination our project is: a mobile application that provide for both doctors and patients their own profile that include

an information about them like(name,age, address, more…) and for only patients they can have a record of their medical history published by them or by an authorized doctors on their profiles.

Doctors can't delete or modify or just access any patient's data unless the patient accept it's request for accessing. To Guarantee an accurate medical data about patients and patient can delete any accessed permission at any time.

In emergency time doctors can access patients' medical history as it is hard and sometimes impossible to ask the patient about his medical history or make him accept the access request.

1.2 Problem definition

1.2.1 Medical awareness

The problem of medical awareness is one of the most difficult problems faced by people in egypt as it is difficult to convince them That their medical history will help doctors in diagnosis process and save their lives and having them agree to collect their data.

1.2.2 Decentralized Data

Each institute has its own medical record for each patient, each with a different format, so it's hard for the patients to have a single location where they can get their own old data and the same for doctors if they tried to find

someone's medical history from all of these different institutes and systems.

Sometimes doctors need to rely on the patients to get their medical history immediately, but it's an unreliable method where sometimes it's physically impossible to ask the patient for his own history (due to emergency for example) .

1.2.3 Paper records

Paper records are getting deprecated, hard to maintain, easy to lose and change by anyone non-specialist.

1.3 Project objectives

After talking about the problems facing the patients and institutes,

we aim and propose to design and implement computerize system to solve these problems, namely **Blockify.**

People will use our project as a medical record to write any information about their medical visits, analyzes, x-ray, disease and medication or let their doctors to record these information if they are trusted and authorized by them .

sequentially each one will has its own medical history.

As such we will make it easy for :

1. Patients to access their data and share it with doctors or other people,while maintaining its privacy and security.

2. Doctors and hospitals to access patients data while making sure they are permitted to do so, or have a case of emergency.

3. Doctors, patients or hospitals to registers themselves in the system, while maintaining the basic standards of verification.

Finally we will increase medical care for people by provide them a central place to store their medical information .they doesn't need messy papers or folders and won't get lost or misplaced.

Also make them sure that they have the right information on hand at the right time,wherever they are.

1.4 Project scope

After talking about these problems, we will try to

solve them through a mobile and web application to serve patients and doctors.

First: we will solve the main challenges existing in the current way hospitals manage their data and the lack of sharing with other

institutes as they can easily find all the information from different institutes organized in one place.

Second: patients can have all of their medical information as it is difficult for them to keep up with their medical history,either be it complicated or long.

Third: Your medical history that we help you to collect can be invaluable in an emergency when there's limited time,your medical information could mean the difference between life and death.

In less serious situations,it saves valuable time searching for relevant facts like medications or allergies.

Fourth: specialist's diagnosis will be more accurate as it will depend on the patient's condition in similar situations that he will get it from the patient's medical history.

Contributions are the services provided by the

system to individuals and institutions. Below we will list

the contributions made by the system.

The project As contributes to save people's lives through their knowledge of their health and knowledge of their diseases that can be detected early from their medical history such as hepatitis B, C, cancer, diabetes and herpes virus which are the most diseases that can be predictable from the patient's medical history like it's updatable test blood and his past diseases or from their family medical history

as they can share their information with other patients like their family “we discusses this before” .

Each person will have a medical file containing its own medical history.

Provides medical history will help doctors’ to diagnose the disease as by knowing the patients’ old diseases will increase the percentage of accurate diagnose in a lot of diseases as we mentioned before and also save doctors and patients time.

In emergency it is impossible to ask the patient for his own history and as we mentioned before these times are The Important time they will need to have the medical history of Patient while doctors can’t make a conversation with the Patient , we provide access permission to the patient medical history in these emergency situations.

Our project will provide a single format for all patient’s medical record so it will be more readable and easy to record it from different institutes .

We will also provide a single location (centralized one) for each patient to record all of his medical history in.

We also Provides people with an awareness of the importance of collecting their medical history to preserve their lives.

1.5 Chapter conclusion

In this chapter we talked about the basic idea of the

project, which is giving patients the ability to save their medical history in one global place and its importance to patients and medical institutes as patient can share their information with any institute or another patient.

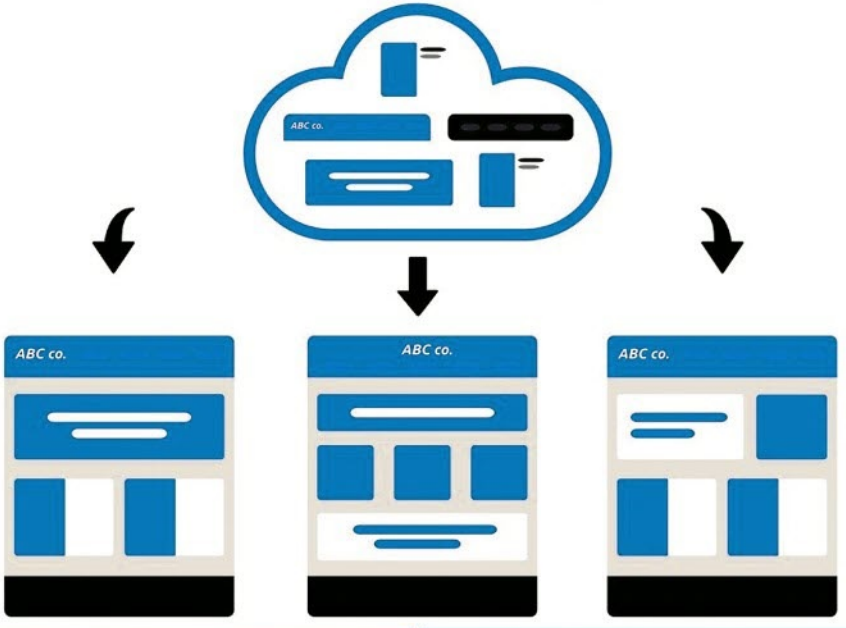
We mentioned some basic points such as the basic problems facing us, such as lack of medical awareness and paper records that can be easy to lose.

Then we talked about the objectives of our projects to solve the problems we faced in the scope of make it easy for patients and doctors to share and access patients' medical data,and save most of peoples' lives in emergency while doctor can access their medical history which will help them find the best diagnosis and more.

Blockif
Your Health Comes First

Chapter 2

Background and Related Work



This chapter will discuss the literature view which showing what other people achieve in the same or familiar fields that research helps us on thinking on applying solutions.

2.1 Background

Blockchain database utilizes blockchain technology to create an immutable ledger consisting of records called blocks that are used to record transactions across many computers so that any involved block cannot be altered retroactively, without the alteration of all subsequent blocks.

Blockchain technology relies on peer-to-peer decentralized transactions. This offers greater security and removes the need for any single controlling entity that retains administration rights over the database.

2.1.1 History Blockchain

It all Start in 1991 when "Haber" and "stornetta" released their research paper titled, "How to Time-Stamp a Digital Document" **[4].**

In this paper, Haber and Stornetta imagined a new way to verify documents.

They proposed time-stamping of documents and linking them to the previous document, they explained how this could form a chain of validating up-to-date documents.

What Haber and Stornetta proposed then would become the precursor to the blockchain now.

Fast forward to 2008 when an unidentified author by the name of "satoshi Nakamoto" published a whitepaper, this paper titled, "Bitcoin: A Peer-to-Peer Electronic Cash System" **[5].**

It was the beginning of Blockchain journey and it's didn't stop in this point but, Blockchain technology is separated from the currency and its potential for other financial, inter-organizational transactions is explored. Blockchain 2.0 is born, referring to applications beyond currency and other applications and business in many fields like "Ethereum" as the most famous example of public network and "Hyperledger fabric" as an example of private blockchain network and this is what we use in the project.

2.1.2 Main Features

Blockchain has a lot of features and the following is the most basic features of why we using blockchain not regular database and why we using private blockchain not public blockchain.

a) Cannot be Corrupted

There are some exciting blockchain features but among them “Immutability” is undoubtedly one of the key features of blockchain technology.
If you ask yourself “why is this technology uncorrupted?” her is the answer, Let’s start with a connecting blockchain with immutability.

Immutability means something that can’t be changed or altered. This is one of the blockchain features that help to ensure that the technology will remain as it is – a permanent, unalterable network. But how does it maintain that way?.

Blockchain works slightly different than the typical banking system, Instead of relying on centralized authorities, it ensures the blockchain features through a collection of nodes, Every node on the system has a copy of the digital ledger. To add a transaction every node needs to check its validity. If the majority thinks it’s valid, then it’s added to the ledger. This promotes transparency and makes it corruption-proof.

So, without the consent from the majority of nodes, no one can add any transaction blocks to the ledger.

Another fact, that backs up the blockchain features is that, once the transaction blocks get added on the ledger, no

one can just go back and change it. Thus, any user on the network won't be able to edit, delete or update it.

b) Decentralized Technology

The network is decentralized meaning it doesn't have any governing authority or a single person looking after the framework. Rather a group of nodes maintains the network making it decentralized.

Let me make it simpler. Blockchain puts us users in a straightforward position. As the system doesn't require any governing authority, we can directly access it from the web and store our assets there, And you store anything starting from cryptocurrencies, important documents, contracts, or other valuable digital assets. And with the help of blockchain, you'll have direct control over them using your private key. So, you see the decentralized structure is giving the common people their power and rights back on their assets.

Let's ask some question and answer them,

Why decentralized is so useful?

How decentralized makes truly changes?

Her we will list the answer of those questions and may be other question came to your mind

I. **Less Failure**: Everything in blockchain is fully organized, and as it doesn't depend on human calculations it's highly fault-tolerant. So, accidental failures of this system are not a usual output.

II. **User Control**: With decentralization, users now have control over their properties. They don't have to rely on any third party to maintain their assets. All of them can do it simultaneously by themselves.

III. **Less Prone to Breakdown**: As decentralized is one of the key features of blockchain technology, it can survive any malicious attack. This is because attacking the system is more expensive for hackers and not an easy solution. So, it's less likely to breakdown.

IV. **No Third-Party**: Decentralized nature of the technology makes it a system that doesn't rely on third-party companies; No third-party, no added risk.

V. **Zero Scams**: As the system runs on algorithms, there is no chance for people to scam you out of anything. No one can utilize blockchain for their personal gains.

VI. **Transparency**: The decentralized nature of technology creates a transparent profile of every participant. Every change on the blockchain is viewable and makes it more concrete.

VII. **Authentic Nature**: This nature of the system makes it a unique kind of system for every kind of person. And hackers will have a hard time cranking it.

c) Enhanced Security

As it gets rid of the need for a central authority, no one can just simply change any characteristics of the network for their benefit. Using encryption ensures another layer of security for the system, but how does it offer so much security compared to already existing techs?.

It's extremely secure because it offers a special disguise – Cryptography, added with decentralization, cryptography lays another layer of protection for users. Cryptography is a rather complex mathematical algorithm that acts as a firewall for attacks. As Every information on the blockchain is hashed cryptographically. In simple terms, the information on the network hides the true nature of the data. For this process, any input data gets through a mathematical algorithm that produces a different kind of

value, but the length is always fixed. And each user will have a private key to access the data but will have a public key to make transactions.

d) Irreversible

Hashing is quite complex, and it's impossible to alter or reverse it. No one can take a public key and come up with the private key. Also, a single change in the input could lead to a completely different ID, so small changes aren't a luxury in the system.

If someone wants to corrupt the network, he/she would have to alter every data stored on every node in the network. There could be millions and millions of people, where everyone has the same copy of ledger. Accessing and hacking millions of computers is next to impossible and costly.

And this features is too hard to bypass, as it will make users don't worry about hackers to taking all there digital assets and worry about there data .

e) Distributed Ledgers

Usually, a public ledger will provide every information about a transaction and the participant. It's all out in the open, nowhere to hide. But in this case, many people can see what really goes on in the ledger.

That's because the ledger on the network is maintained by all other users on the system. This distributed computational power across the computers to ensure a better outcome.

This is the reason it's considered one of the blockchain essential features. The result will always be a higher efficient ledger system that can take on the traditional ones.

So why Distributed Ledgers is on of the blockchain important features ?

I. **No Malicious Changes**: Distributed ledger responds really well to any suspicious activity or tamper. As no one can change the ledger and everything updates real fast, tracking what's happening in the ledger is quite easy with all these nodes.

II. **Ownership of Verification**: Here, nodes act as verifiers of the ledger. If a user wants to add a new block others would have to verify the transaction and then give the green signal. This provides the user with fair participation.

III. **No Extra Favors**: No one on the network can get any special favors from the network. Everyone has to go through the usual channels

and then add their blocks. It's not like you have more power so you'll get more privileges.

IV. Managership: To make the blockchain features work, every active node has to maintain the ledger and participate for validation.

V. **Quick Response**: Removing the intermediates quickens the system response. Any change in the ledger is updated in minutes or even seconds.

f) Consensus

Every blockchain thrives because of the consensus algorithms. The architecture is cleverly designed, and consensus algorithms are at the core of this architecture. Every blockchain has a consensus to help the network make decisions.

In simple terms, the consensus is a decision-making process for the group of nodes active on the network. Here, the nodes can come to an agreement quickly and relatively faster. When millions of nodes are validating a transaction, a consensus is absolutely necessary for a system to run smoothly. You could think of it as a kind of a voting system, where the majority wins, and the minority has to support it.

The consensus is responsible for the network being trustless. Nodes might not trust each other, but they can trust the algorithms that run at the core of it. That's why every

decision on the network is a winning scenario for the blockchain. It's one of the benefits of blockchain features.

There are lots of different consensus algorithms for blockchains over the globe. Each has its own unique way to make decisions and perfecting previously introduces mistakes. The architecture creates a realm of fairness on the web and the algorithm used in the project is "Raft" **[6].**

g) Faster Settlement

Blockchain offers a faster settlement compared to traditional banking systems. This way a user can transfer money relatively faster, which saves a lot of time in the long run. This is one of the best benefits of blockchain features to this day. And with the third party out of the way and it's the concept of how blockchain is faster than other systems using regular database if we compare between blockchain system and banking system or any other system.

h) Permissioned membership

Hyperledger Fabric is a framework for Permissioned networks, where all participants have known identities. When considering a Permissioned network, you should think about whether your blockchain use case needs to comply with data protection regulations. Many use cases in the financial sector and healthcare industry, in particular, are subject to data protection laws that require knowing who the

members of the network are and who is accessing specific data.

i) Performance, scalability, and levels of trust

Hyperledger Fabric is built on a modular architecture that separates transaction processing into three phases: distributed logic processing and agreement ("chain code"), transaction ordering, and transaction validation and commitment. This separation confers several advantages: Fewer levels of trust and verification are required across node types, and network scalability and performance are optimized.

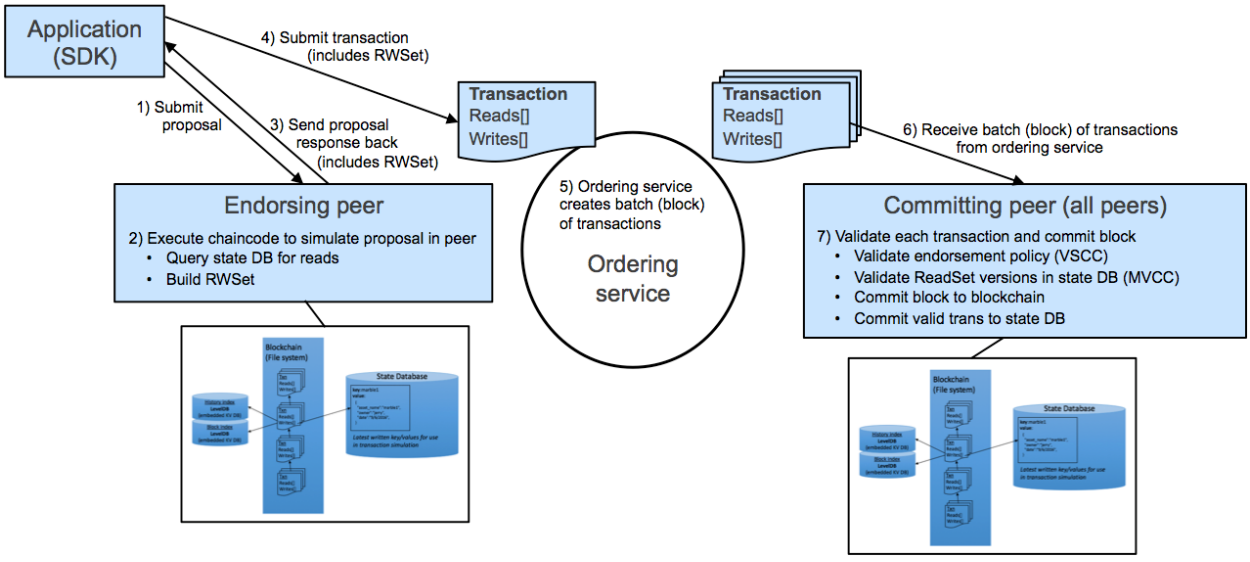


Figure 2.1 Transaction lifecycle in Hyperledger Fabric

Because only the signatures and read/write sets are being sent around the network with the architecture, the scalability and performance are optimized. In addition,

because only the endorsers and the committers truly see the transaction, fewer levels of trust are required in different parts of the Blockchain system, offering more security.

j) Modular architecture supporting plug-in components

The modularity of Hyperledger Fabric architecture enables network designers to plug in their preferred implementations for components, which is an advantage. One of the most requested areas for modularity is "bring your own identity." Some multi-company networks already have identity management and want to reuse instead of rebuild. Other components of the architecture that can be easily plugged in include consensus or encryption, where some countries have their own encryption standards.

k) Protection of digital keys and sensitive data

HSM (Hardware Security Module) support is vital for safeguarding and managing digital keys for strong authentication. Hyperledger Fabric provides modified and unmodified PKCS11 for key generation, which supports cases like identity management that need more protection. For scenarios dealing with identity management, HSM increases the protection of keys and sensitive data.

2.1.3 Health information

After the scientists developed the first computers in the 1940s, the community and the world realized that these

new devices will provide a lot of services for all of humanity.

In the field of information storage, processing and recovery, and after this date a decade ago doctors and specialists began to try to take advantage of these technologies In real terms through the development of the idea of information management and the role of computer in medicine and health care, which is one of the most important scientific fields and the most widespread and influential.

It's also one of the most important was developed early the idea of electronic medical records, which is a focal point pour It divides many channels of information related to the provision of health care to the patient- the focus of all medical and health care activities. The reasons for the development of the idea of these electronic records as an alternative to traditional paper records.

The most important lies in the ability of the computer and it's software to store, process and restore the various types of information, which represent the most important contents of medical records.

The stages and experiences of the development of these medical records and their integration with the various sources of information have been followed through the information network systems that have led to the idea of

decentralization and communication of information between more than one hospital and a medical institution.

Even more through the Internet, Of patients and healthy to rely on them as a source of medical information. Hence, we can understand the role of computer and the idea of using it in medicine and its relation to health sciences and the relationship of the latter with computer science and medical engineering. Also.

We must recognize the nature of the medical information and distinguish it with some characteristics- from the rest of the human information - such as privacy and standardization and its need for complex and accurate treatment. which lead to the importance of using artificial intelligence applications in the treatment of medical information and its integration with the practice of medicine. [7].

2.1.4 Basics of medical information

The basic concepts associated with the computer systems, equipment, software and generations developed since the emergence of its first models in the fifties, which were so large and expensive that it was owned by only the largest institutions and scientific institutes, must be comprehensively understood to become accessible to individuals at low cost and enormous potential Dreamed of

by major research centers and not even the US space agency itself.

The science continued to provide us with every day and even every new hour in the field of computer development and systems until generations of personal devices, supercomputers and information network servers appeared.

The majority of the capabilities of each type and generation of computer depends on the CPU's ability to represent the computer represented by the human brain, in addition to the huge development of the capacity of computer memory and its capacity and the development of storage devices from electronic chips and tapes and magnetic disks and the emergence of laser discs CD-ROMS of different generations, and these devices have been expanding in the capacity and accuracy of information storage and decreasing at the same time at their cost Similarly, there has been significant progress in the means of inputting information from keyboards and pencils to the development of the ability of the computer to know the human voice and the vocabulary of the various languages.

The research continues until the human scientists promised to talk to the computer as we speak to each other, Such as screens and printers, and finally the information networks which have become the largest and most important

applications, no doubt the Internet, which is also accessible to everyone. [8].

On the other side of the development of all these systems and equipment, the software has also received a significant amount of sophistication to meet the needs of system applications and integration in perfect harmony and harmonization.

Operating systems, programming languages, database systems, network integration systems and information security have evolved into computer systems and application sciences. The products and services that ultimately end up with the fruit of their effort and the fruit of their development are in the hands of an intelligent user who knows what he wants to reach and how he employs all of this to reach what he want [9].

2.2 Review of Relevant Work

There are many apps made and work with centralized database (regular database) like:

2.2.1 Advocate Physician Partners

Advocate Physician Partners (APP) brings together more than 5,000 physicians who are committed to improving health care quality, safety and outcomes for patients across Chicago land and central Illinois.

Formed as a care management collaboration with Advocate Health Care, APP is a leader in population health management and has garnered wide-spread national recognition for its innovative clinical integration program. The comprehensive approach coordinates patient care across the continuum—ensuring care is delivered at the right place and at the right time. This results in more efficiency, improved health outcomes and significant cost savings for patients.

APP also supports its physician members in managing the health of their practices with contracting, medical managements, Electronic Medical Records and value added services including group health and dental insurance and preferred pricing on vaccines, medical surgical supplies and office supplies **[10].**

2.2.2 Community Health Care

This app has been working to rapidly transform how you continue to access the care you need. We are helping our health care community by making virtual care options even more accessible than ever before and are sharing some important updates on what we're doing to protect and serve you in a way that prioritizes your health and safety **[11].**

2.2.3 TRICARE

TRICARE is the health care program for uniformed service members, retirees, and their families around the world.

TRICARE provides comprehensive coverage to all beneficiaries, including:

- Health plans
- Special programs
- Prescriptions
- Dental plans

Most TRICARE health plans meet the requirements for minimum essential coverage under the Affordable Care Act.

TRICARE is managed by the Defense Health Agency under the leadership of the Assistant Secretary of Defense (Health Affairs) [12].

2.2.4 The Blue Cross Blue Shield System

Since 1929, Blue Cross Blue Shield (BCBS) companies have provided healthcare coverage to members, allowing them to live free of worry, free of fear. In every ZIP code, Blue Cross Blue Shield offers a personalized approach to healthcare based on the needs of the communities where their members live and work.

They work closely with hospitals and doctors in the communities they serve to provide quality, affordable healthcare .

They also understand and answer to the needs of local communities, while providing nationwide healthcare coverage that opens doors for more than 107 million members in all 50 states, Washington, D.C., and Puerto Rico. Nationwide, more than 96 percent of hospitals and 95 percent of doctors and specialists contract with Blue Cross Blue Shield companies [13].

2.2.5 Premera blue cross

As the largest health plan in the Pacific Northwest, Premera serves more than 2 million people—from individuals and families to employees of Fortune 100 companies. they provide comprehensive, tailored services to customers in Washington and Alaska that include innovative programs focused on wellness and prevention, disease management, and patient safety. they deliver these programs through health, life, vision, dental, stop-loss, disability, workforce wellness, and other related products and services. their networks include thousands of doctors, healthcare providers, and hospitals [14].

2.3 Relationship between the Relevant Work and Our Own Work

We trying to take these great examples and mixed them with our project, but instead of using the regular different types of databases, we will use the blockchain technology, aiming to the point of building a secure medical record system that no one can steal or change the data of the patients which are recorded.

2.4 Summary

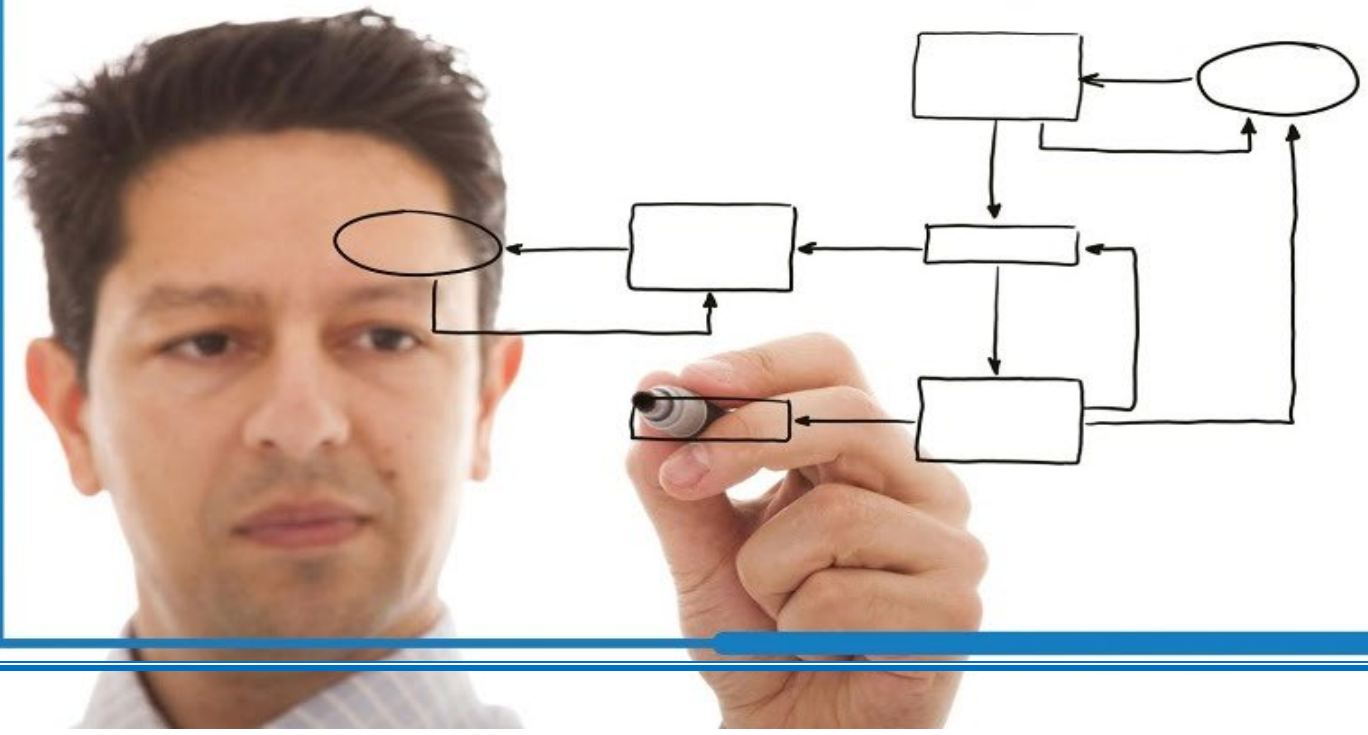
In this chapter, we talked about the previous solutions of the problem and the stage of developing the medical records thanks to the tremendous progress in technology and the ease of access to medical data storage, development.

Blockif

Your Health Comes First

Chapter 3

System Analysis and Design



In this chapter you will get advanced knowledge about the Analysis of the project with lots of details.

3.1 System development life cycle(SDLC)

3.1.1 About system development life cycle

System is a regularly interacting or interdependent group of items forming a unified whole [15].

SDLC is a systematic approach which explicitly breaks down the work into six phases that are required to implement either new or modified a hardware system only, a software system only or a combination of both to meet or exceed customer's expectations.

Therefore we use System Development Life Cycle as a group of phases to create a software component (item) that can integrate with other software components to create the whole system.

System development life cycle phases that will be show in the figure 2-1.

3.1.2 System Development life cycle phases

1. System planning

The Planning phase is the most crucial step in creating a successful system, during this phase you decide exactly

what you want to do and the problems you're trying to solve.

I. Defining the problems, the objectives and the resources such as personnel and costs.
II. Studying the ability of proposing alternative solutions after meeting with clients, suppliers, consultants and employees.
III. Studying how to make your product better than your competitors'.

After analyzing this data you will have three choices: develop a new system, improve the current system or leave the system as it is.

2. System analysis

The end-user's requirements should be determined and documented, what their expectations are for the system, and how it will perform.

A feasibility study will be made for the project as well, involving determining whether it's organizationally, economically, socially, technologically feasible. it's very important to maintain strong communication level with the clients to make sure you have a clear vision of the finished product and its function.

3. System Design

The design phase comes after a good understanding of customer's requirements, this phase defines the elements of a system, the components, the security level, modules, architecture and the different interfaces and type of data that goes through the system.

A general system design can be done with a pen and a piece of paper to determine how the system will look like and how it will function, and then a detailed and expanded system design is produced, and it will meet all functional and technical requirements, logically and physically.

4. Implementation and deployment

this phase comes after a complete understanding of system requirements and specifications, it's the actual construction process after having a complete and illustrated design for the requested system

In the Software Development Life Cycle, the actual code is written here, and if the system contains hardware, then the implementation phase will contain configuration and fine-tuning for the hardware to meet certain requirements and functions.

In this phase, the system is ready to be deployed and installed in customer's premises, ready to become running, live and productive, training may be required for end users to make sure they know how to use the system and to get familiar with it, the implementation phase may take a long time and that depends on the complexity of the system and the solution it presents.

5. System testing and integration

Bringing different components and subsystems together to create the whole integrated system, and then Introducing the system to different inputs to obtain and analyze its outputs and behavior and the way it functions. Testing is becoming more and more important to ensure customer's satisfaction, and it requires no knowledge in coding, hardware configuration or design.

Testing can be performed by real users, or by a team of specialized personnel, it can also be systematic and automated to ensure that the actual outcomes are compared and equal to the predicted and desired outcomes.

6. System maintenance

In this phase, periodic maintenance for the system will be carried out to make sure that the system won't become obsolete, this will include replacing the old hardware and continuously evaluating system's performance, it also includes providing latest updates for certain components to make sure it meets the right standards and the latest technologies to face current security threats These are the main six phases of the System Development Life Cycle, and it's an iterative process for each project.

It's important to mention that excellent communication level should be maintained with the customer, and Prototypes are very important and helpful when it comes to meeting the requirements. By building the system in short iterations; we can guarantee meeting the customer's requirements before we build the whole system.

Many models of system development life cycle came up from the idea of saving effort, money and time, in addition to minimizing the risk of not meeting the customer's requirement at the end of project, some of these models are SDLC Iterative Model, and SDLC Agile Model.

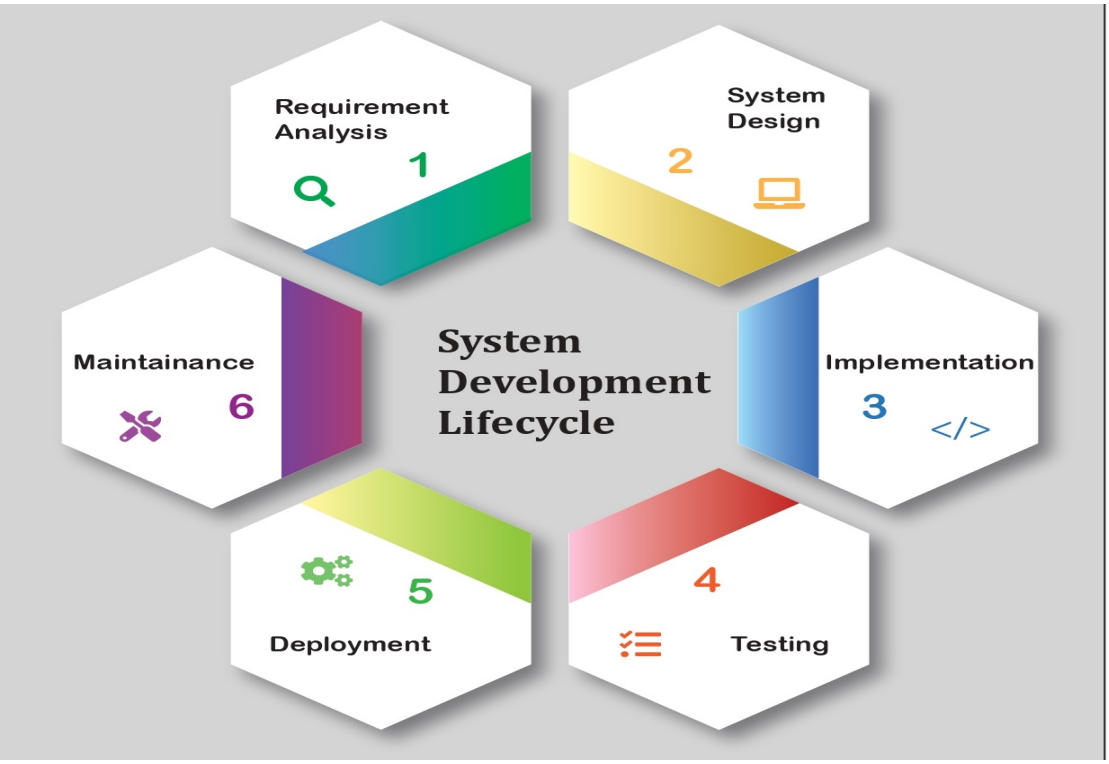


Figure 3.1 System development lifecycle

3.2 Data flow diagram

DFD graphically representing the functions, or processes, which capture, manipulate, store, and distribute data between a system and its environment and between components of a system. The visual representation makes it a good communication tool between User and System designer.

Structure of DFD allows starting from a broad overview and expand it to a hierarchy of detailed diagrams [16].

Data flow diagrams can be divided into logical and physical. The logical data flow diagram Logical data flow diagrams focus on *what* happens in a particular information flow: what information is being transmitted, what entities are receiving that info, what general

processes occur. Physical data flow diagrams focus on *how* things happen in an information flow.

3.2.1 Dataflow Components

1. **External entity:** an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.

2. **Process:** any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules. A short label is used to describe the process, such as "Submit payment".

3. **Data store:** files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label, such as “Orders”.

4. **Data flow:** the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like “Billing detai

3.2.2 Context Diagram

A context diagram gives an overview and it is the highest level in a data flow diagram “DFD Level 0”, containing only one process representing the entire system. It should be split into major processes which give greater detail and each major process may further split to give more detail.

- All external entities are shown on the context diagram as well as major data flow to and from them.
- The diagram does not contain any data storage.
- The single process in the context-level diagram, representing the entire system, can be exploded to include the major processes of the system in the next level diagram, which is termed as diagram 0.

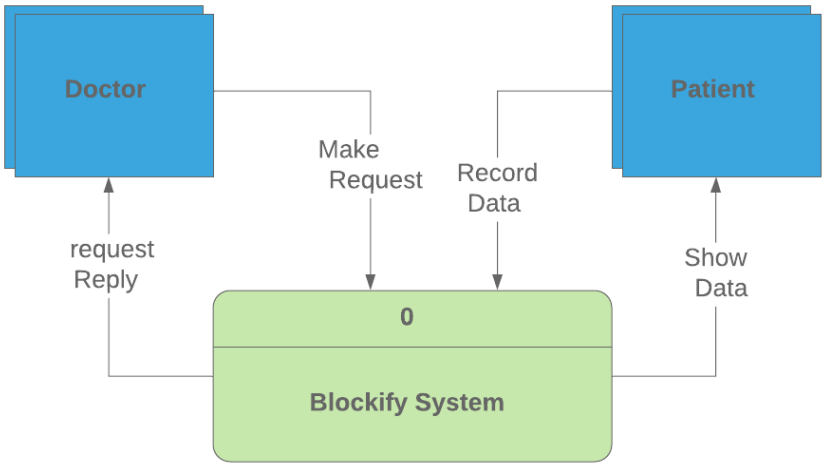


Figure 3.2 Context Diagram

3.2.3 Level 1 DFD

DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses.

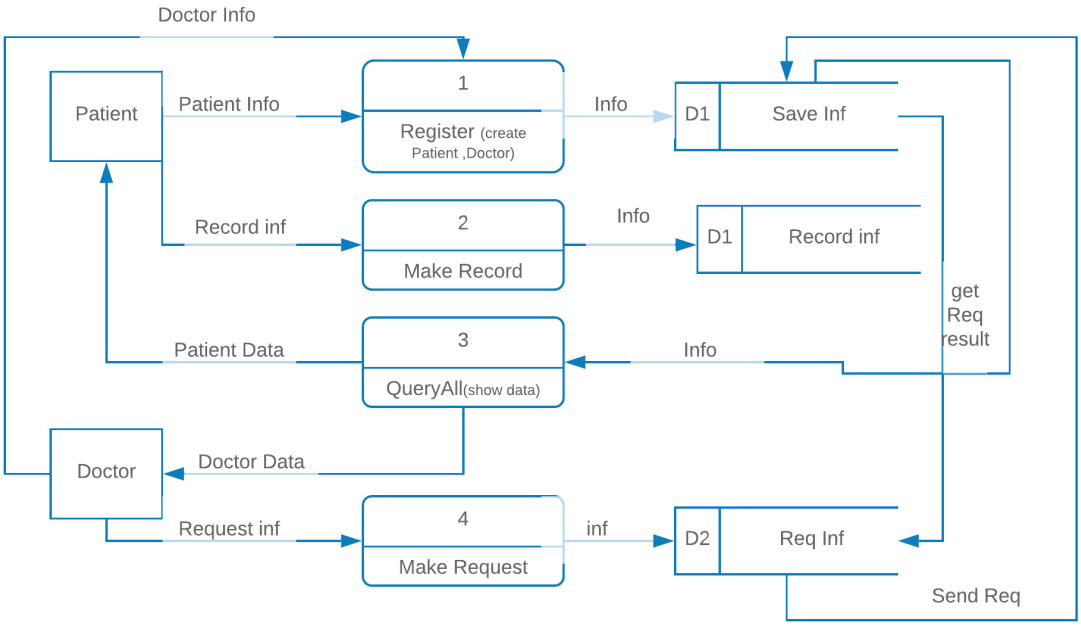


Figure 3.3 DFD level 1

3.2.4 Level 2 DFD

DFD Level 2 then goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system's functioning.

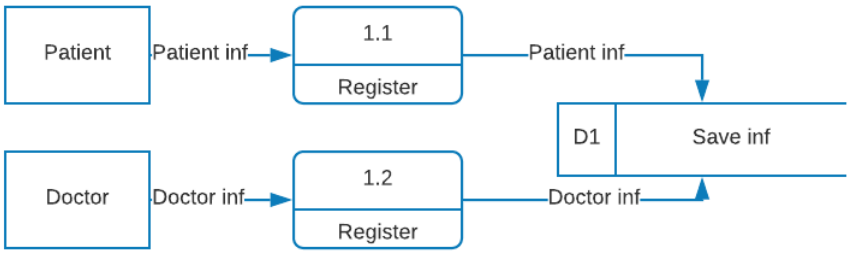


Figure 3.4 DFD level 2

3.3 System analysis diagram in general

3.3.1 Use Case

At its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.

A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well [17].

3.3.2 Use case components

3.3.2.1 System

Is whatever you are developing. It could be a website,a software component, a business process, an app, or any

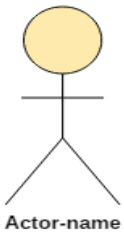
number of other things.you represent a system with a rectangle.

3.3.2.2 Use case

Use cases are elements that really start to describe what the system does. They are depicted with an oval shape and they represent an action that accomplishes some sort of task within the system.

3.3.2.3 Actor

An actor is going to be someone or something that uses our system to achieve a goal, and they are represented by a stick figure.



3.3.2.4 Relationships

Relationships can show how actors and use cases interact with each other. There are different types of relationships (like association,include,extend and generalization) that are represented by varying types of lines and arrows.

1. Association

An actor, by definition, is using a system to achieve a goal. So each actor has to interact with at list one the use cases within system.

When you have an association relationship you draw a solid line between the actor and the use case to show this relationship.

2. Generalization

In UML modeling, a generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent). Generalization relationships are used in class, component, deployment, and use-case diagrams to indicate that the child receives all of the attributes, operations, and relationships that are defined in the parent.

Generalization of an actor means that one actor can inherit the role of the other actor. The descendant inherits all the use cases of the ancestor. The descent has one or more use cases that are specific to that role.

Generalization is shown as a solid directed line with a large hollow triangle arrowhead.

3. Extend

An extend relationship has a base use case and an extend use case.when the base use case is executed,will happen sometimes but not every time.

The extend use case will only happen if certain criteria are met.

Another way to think of it that you have the option to extend the behaviour of the base use case.

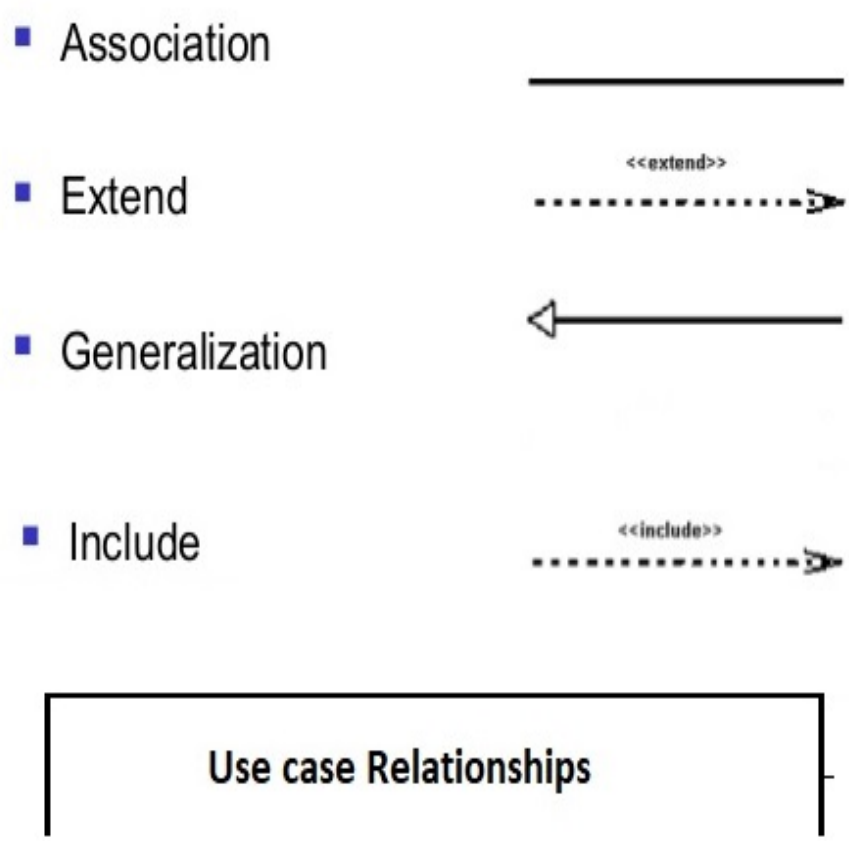
4. Include

An include relationship shows dependency between a base use case and an included use case.

Every time the base use case is executed the included use case is executed as will.

Another case to think of it is a base use case requires an included use case in order to be complete.

When you have include relationship,you draw a dashed line with an arrow that points towards the included use case.



3.3.3 Creating Use Case Diagram

Start by identifying the actors(role of users)of the system. for each category of users, identify all roles played by the user relevant to the system. Identify what are the users required the system to be performed to achieve these goals.

create use cases for every goal, then there should be an association relationship between actors and use case. And Finally you will notice the similarities between use cases, or between actors, start modeling the appropriate relationship between them.

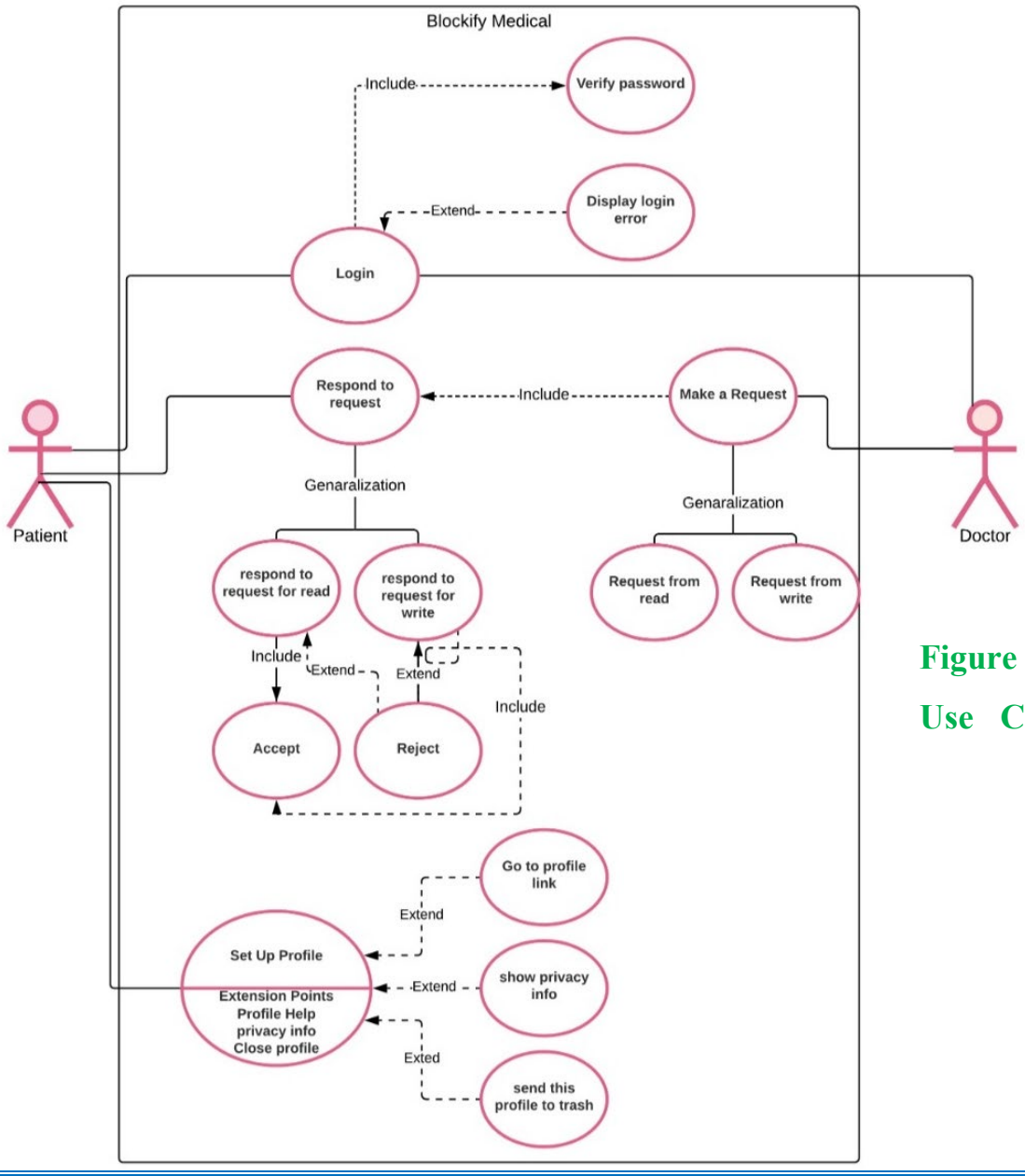


Figure 3.5 Blockify Use Case Diagram

3.3.4 Sequence Diagram

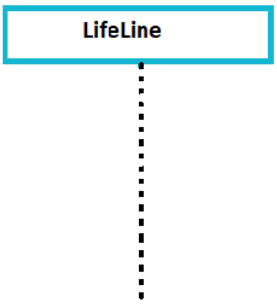
Sequence diagrams are interaction diagrams they are good for showing what's going on, for extracting requirements and for interacting with customers. It is advisable to generate a sequence diagram for every basic flow of every use case.

Sequence diagrams represent specific interactions, commonly known as *scenarios*, among elements. A scenario is a specific interaction among a set of elements, characterized by a specific set of messages arriving among the modeled elements in a specific order.

A sequence diagram shows as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the older in which they occur. This allows the specifications of simple runtime scenarios in a graphical manner [18].

3.3.4.1 Sequence Diagram Notation

A sequence diagram is structured in such a way that it represents a timeline which begins at the top and descends gradually to mark the sequence of interactions. Each object has a column and the messages exchanged between them are represented by arrows.



1- Lifeline notation

A sequence diagram is made up of several of these lifeline notations that should be arranged horizontally across the top of the diagram. No two

lifeline notations should overlap each other. They represent the different objects or parts that interact with each other in the system during the sequence.

A lifeline notation with an actor element symbol is used when the particular sequence diagram is owned by a use case.

Destroying Objects

Objects can be terminated early using an arrow labeled <<destroy>>" that points to an X. This object is removed from memory when that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets .

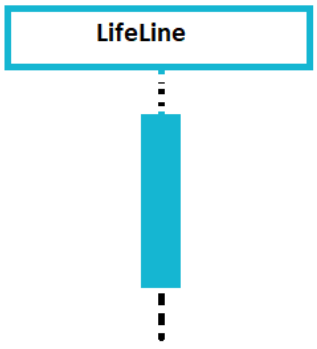
2- Activation Bars

Activation bar is the box placed on the lifeline.It is used to indicate that an object is active (or instantiated) during an interaction between two objects. The length of the rectangle indicates the duration of the objects staying active .

In a sequence diagram, an interaction between two objects occurs when one object sends a message to another. The use of the activation bar on the lifelines of the Message Caller (the object that sends the message) and the Message Receiver (the object that receives the message) indicates that both are active/is instantiated during the exchange of the message.

3- Activation Or Execution occurrence

A thin rectangle on a lifeline represents the period during which an element is performing an operation. The top and the bottom of the rectangle are aligned with the initiation and the completion time resp



4- Message

An arrow from the Message Caller to the Message Receiver specifies a message in a sequence diagram. A message can flow in any direction; from

left to right, right to left or back to the Message Caller itself. While you can describe the message being sent from one object to the other on the arrow, with different arrowheads you can indicate the type of message being sent or received.

The message arrow comes with a description, which is known as a message signature, on it.

The format for this message signature is below. All parts except the message_name are optional**.**

Message Types In Sequence Diagram

i. Synchronous Message

A synchronous message is used when the sender waits for the receiver to process the message and return before carrying on with another message. The arrowhead used to indicate this type of message is a solid one, like the one below.

ii. Asynchronous Message

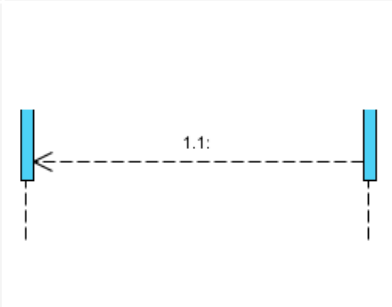
An asynchronous message is used when the message caller does not wait for the receiver to process the message and return before sending other messages to other objects within the system. The arrowhead used to show this type

of message is a line arrow like shown in the example below.

iii. Return Message

A return message is used to indicate that the message receiver is done processing the message and is returning control over to the message caller. Return messages are optional notation pieces, for an activation bar that is triggered by a synchronous message always implies a return message.

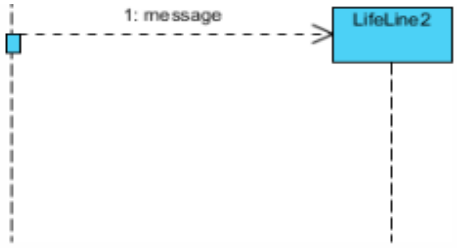
Tip: You can avoid cluttering up your diagrams by minimizing the use of return messages since the return value can be specified in the initial message arrow itself.



iv. Participant creation Message

Objects do not necessarily live for the entire duration of the sequence of events. Objects or participants can be created according to the message that is being sent.

The dropped participant box notation can be used when you need to show that the particular participant did not exist until the create call was sent. If the created participant does something immediately after its creation, you should add an activation box right below the participant box.



v. Self Message

A message an object sends to itself, usually shown as a U sharped arrow pointing back to itself.

3.3.4.2 Creating sequence Diagram

As we have already discussed, the purpose of interaction diagrams is to capture the dynamic aspect of a system. So to capture the dynamic aspect, we need to understand what a dynamic aspect is and how it is visualized.

Dynamic aspect can be defined as the snapshot of the running system at a particular moment.

Following things are to be identified clearly before drawing the interaction diagram:

1-Objects taking part in the interaction.

2-Message flows among the objects.

3-The sequence in which the messages are flowing.

4-Object organization.

3.3.4.3 sequence Diagram For our project

Before drawing the sequence diagram, it's necessary to identify the objects or actors that would be involved in creating a new user account. These would be:

1-Doctor

2-Patient Profile

3-Patient Services

4-Blockchain backend

6-Patient

Once you identify the objects, it is then important to write a detailed description on what the use case does. From this description, you can easily figure out the interactions (that should go in the sequence diagram) that would occur between the objects above, once the use case is executed.Here are the steps that occur in the use case named 'Record request from doctor to patient'.

1-The doctor opens the patient profile.

2-The doctor then selects one of request types.

3-Then the doctor will send request to blockchain backend server .

4-The server will send email or notification to the patient contains these details.

5-The patient will then send the accepted or rejected request to the server.

6-Finally will then send the reply to the doctor either It is rejected or accepted.

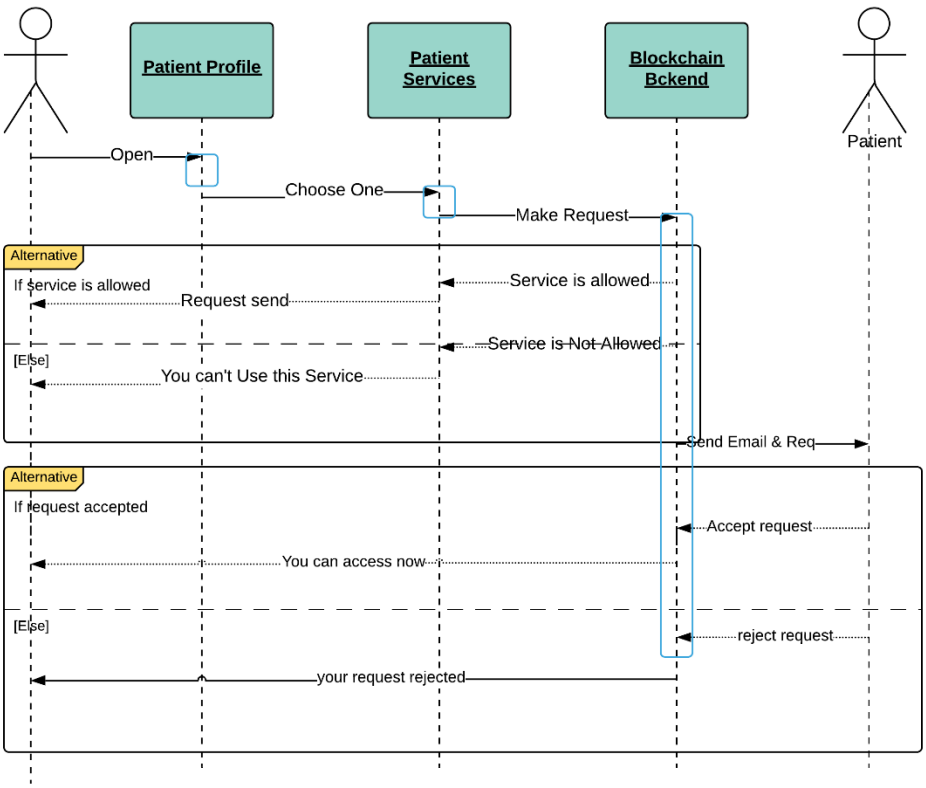


Figure 3.6 Record Request From doctor to patient

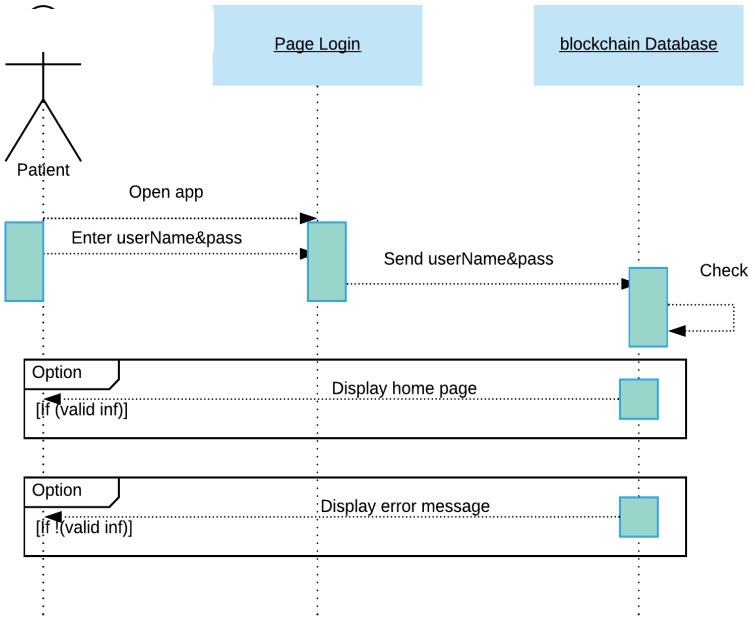


Figure 3.7 Login Sequence Diagram

3.3.5 Activity Diagram

We use **Activity Diagrams** to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram [19].

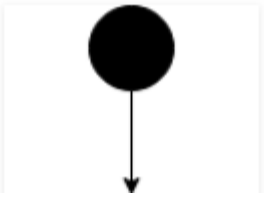
An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.

An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram.

3.3.5.1 Activity Diagram notations

1. Initial State

A small filled circle followed by an arrow represents the initial action state or the start point for any activity diagram. For activity diagram using swim lanes, make sure the start point is placed in the top left corner of the first column.



2. Action Or Activity State

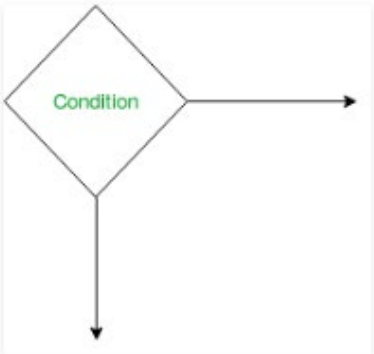
An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.

3. Action Flow Or Control Flow

Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.

An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.

4. Decision Node And Branching



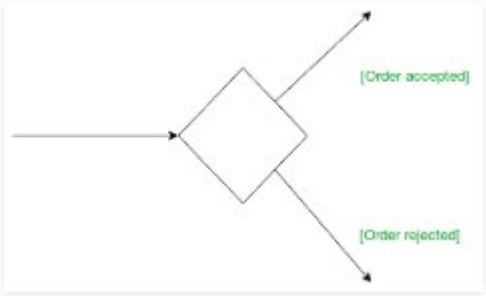
Represent a test condition to ensure that the control flow or object flow only goes down one path.

The outgoing arrows from the decision node can be labeled with conditions or guard expressions.It always includes two or more output arrows.

5. Guards

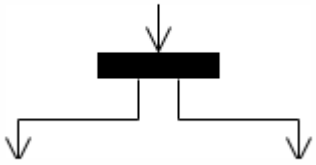
A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets.

The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.



6. Fork

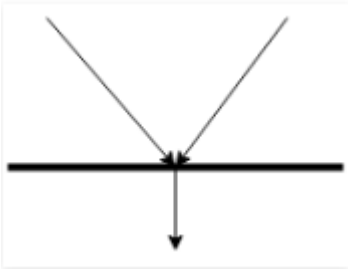
Split behavior into a set of parallel or concurrent flows of activities (or actions).



When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement, We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.

7. Join

Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.



8. Swimlanes

We use swimlanes for grouping related activities in one column. They group related activities into one column or one row.

Swimlanes can be vertical and horizontal. Swimlanes are used to add modularity to the activity diagram.

They usually give more clarity to the activity diagram. It's similar to creating a function in a program.

9. Time Event

We can have a scenario where an event takes some time to complete. We use an hourglass to represent a time event.

10. Final State Or End State

The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram.

3.3.5.2 Create An Activity Diagram

To draw an activity diagram, one must understand and explore the entire system. All the elements and entities that are going to be used inside the diagram must be known by the user. The central concept which is nothing but an activity must be clear to the user. After analyzing all activities, these activities should be explored to find various constraints that are applied to activities. If there is such a constraint, then it should be noted before developing an activity diagram.

Following rules must be followed while developing an activity diagram :

1. All activities in the system should be named.
2. Activity names should be meaningful.
3. Constraints must be identified.
4. Activity associations must be known.

3.3.5.3 Activity Diagram For Our Project

The next Figure 2.27 Active Digram is our active digram that contain all activities in the system and all we mention.

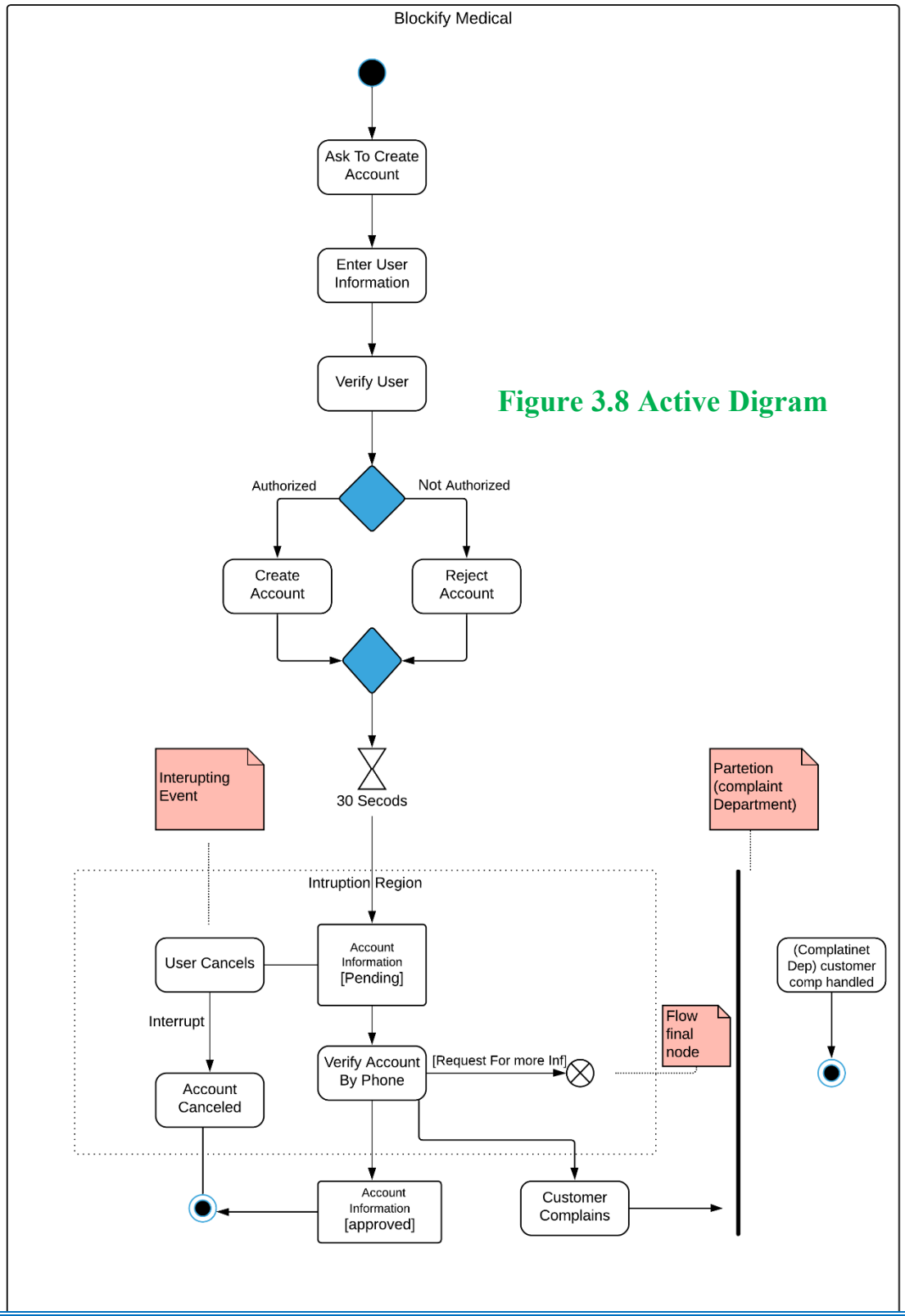


Figure 3.8 Active Digram

3.3.6 State Machine Diagram

State machine diagram typically are used to describe state-dependent behavior for an object. State machine diagrams are usually applied to objects but can be applied to any element that has behavior to other entities such as: actors, use cases, methods, subsystems systems and etc. and they are typically used in conjunction with interaction diagrams (usually sequence diagrams).

3.3.6.1 State Machine Notations

1. **Initial State**

We use a black filled circle represent the initial state of a System or a class.

2. **Transition**

We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labeled with the event which causes the change in state.

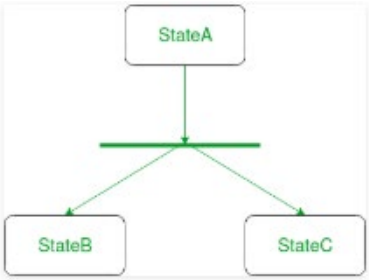
3. **State**

We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.

4. **Fork**

We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states.

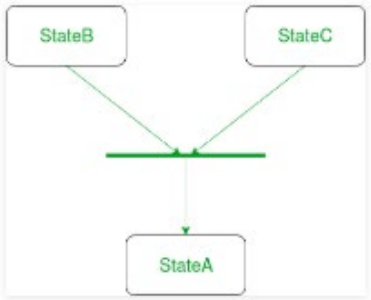
We use the fork notation to represent a state splitting into two or more concurrent states.



5. **Join**

We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state.

We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.



6. Self Transition

We use a solid arrow pointing back to the state itself to represent a self transition. There might be scenarios when the state of the object does not change upon the occurrence of an event, We use self transitions to represent such cases.

7. Composite State

We use a rounded rectangle to represent a composite state also.We represent a state with internal activities using a composite state.

8. Final State

We use a filled circle within a circle notation to represent the final state in a state machine diagram.

3.3.6.2 Creating State machine Diagram

As we discussed before we state machine diagram to state the events responsible for change in state (we do not show what processes cause those events).

We use it to model the dynamic behavior of the system .

And To understand the reaction of objects/classes to internal or external stimuli.

*To Draw State machine diagram Identify the initial state and the final terminating states.

Identify the possible states in which the object can exist (boundary values corresponding to different attributes guide us in identifying different states) then, Label the events which trigger these transitions.

3.3.6.3 State machine diagram for our project

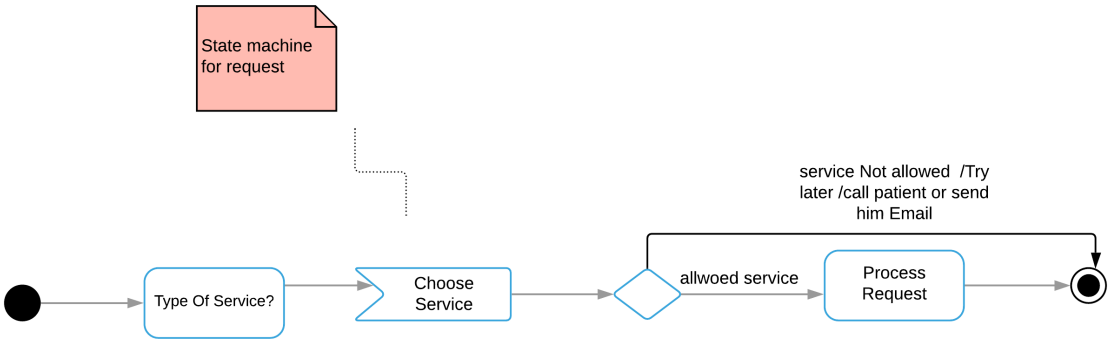


Figure 3.9 State Machine Diagram

3.3.7 Flowchart Diagram

A **flowchart** is a type of diagram that represents a workflow or process. A flowchart can also be defined as a diagrammatic representation of an algorithm, a step-by-step approach to solving a task.

The flowchart shows the steps as boxes of various kinds, and their order by connecting the boxes with arrows. This diagrammatic representation illustrates a solution model to a given problem. Flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields [20].

3.3.7.1 Overview

Flowcharts are used in designing and documenting simple processes or programs. Like other types of diagrams, they help visualize what is going on and thereby help understand a process, and perhaps also find less-obvious features within the process, like flaws and bottlenecks.

There are different types of flowcharts: each type has its own set of boxes and notations. The two most common types of boxes in a flowchart are:

*a processing step, usually called *activity*, and denoted as a rectangular box.

*a decision, usually denoted as a diamond.

3.3.7.2 Common Flowchart Diagram Symbols

1. Flowline

Shows the process's order of operation. A line coming from one symbol and pointing at another. Arrowheads are added if the flow is not the standard top-to-bottom, left-to right.

2. Terminal

Indicates the beginning and ending of a program or sub-process. Represented as a stadium, oval or rounded (fillet) rectangle. They usually contain the word "Start" or "End", or another phrase signaling the start or end of a process, such as ""submit inquiry" or "receive product".

3. Process

Represents a set of operations that changes value, form, or location of data. Represented as a rectangle.

4. Decision

Shows a conditional operation that determines which one of the two paths the program will take. The operation is commonly a yes/no question or true/false test. Represented as a diamond.

5. Input-Output

Indicates the process of inputting and outputting data, as in entering data or displaying results. Represented as a parallelogram.

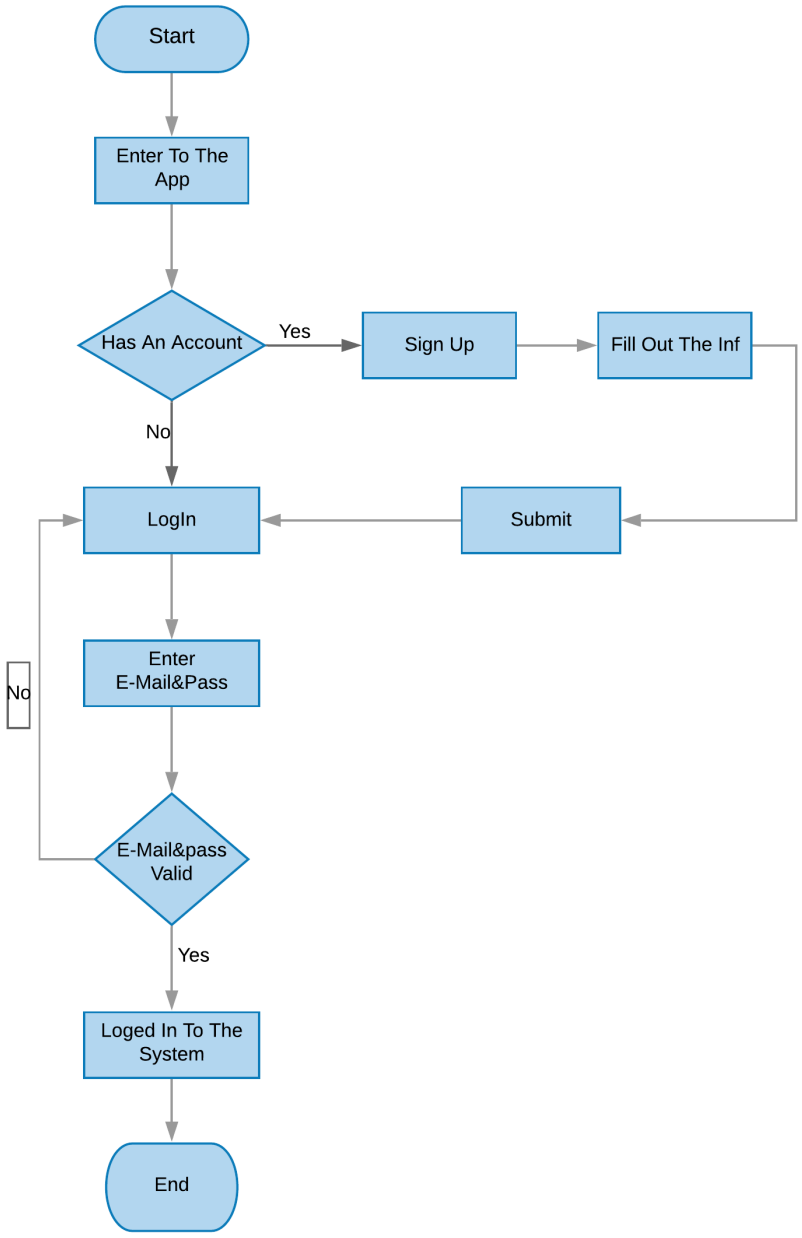


Figure 2.10 FlowChart Diagram

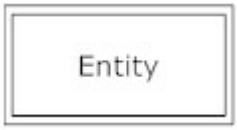
3.4 Entity Relationship diagram (ERD)

3.4.1 ERD Notation Guide

An ER Diagram contains entities, attributes, and relationships. In this section, we will go through the ERD symbols in detail.

1- Entity

An ERD entity is a definable thing or concept within a system, such as a person/role (e.g. Student), object (e.g. Invoice), concept (e.g. Profile) or event (e.g. Transaction) (note: In ERD, the term "entity" is often used instead of "table", but they are the same). When determining entities, think of them as nouns. In ER models, an entity is shown as a rounded rectangle, with its name on top and its attributes listed in the body of the entity shape. The ERD example below shows an example of an ER entity [21].



i. Entity Attributes

It is also known as a column, an attribute is a property or characteristic of the entity that holds it, An attribute has a name that describes the property and a type that describes the kind of attribute it is, such as varchar for a string, and int for integer. When an ERD is drawn for physical

database development, it is important to ensure the use of types that are supported by the target RDBMS.

ii. Primary Key

Also known as PK, a primary key is a special kind of entity attribute that uniquely defines a record in a database table. In other words, there must not be two (or more) records that share the same value for the primary key attribute.

iii. Foreign Key

Also known as FK, a foreign key is a reference to a primary key in a table. It is used to identify the relationships between entities. Note that foreign keys need not be unique. Multiple records can share the same values. The ER Diagram example below shows an entity with some columns, among which a foreign key is used in referencing another entity.

2- Actions

which are represented by diamond shapes, show how two entities share information in the database.

3- Connecting Lines

solid lines that connect attributes to show the relationships of entities in the diagram.

4- Cardinality

Cardinality defines the possible number of occurrences in one entity which is associated with the number of occurrences in another.

In an ER diagram, cardinality is represented as a crow's foot at the connector's ends. The three common cardinal relationships are one-to-one, one-to-many, and many-to-many and there is another cardinal relationships,these relationships called notation styles.

3.4.2 Cardinal Notation Styles

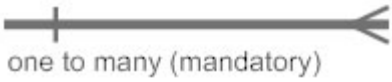
1- One To One

As shown in the figure below.

one to one

2- One To Many

As shown in the figure below.



3- Many To Many

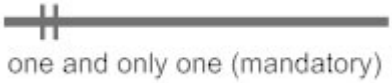
As shown in the figure below.

4- One Or More

As shown in the figure below.

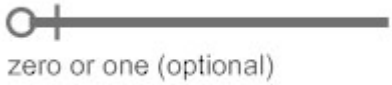
5- One And Only One

As shown in the figure below.



6- Zero Or One

As shown in the figure below.



7- Zero Or Many

As shown in the figure below.

3.4.3 Entity Relationship Diagram For Our Project

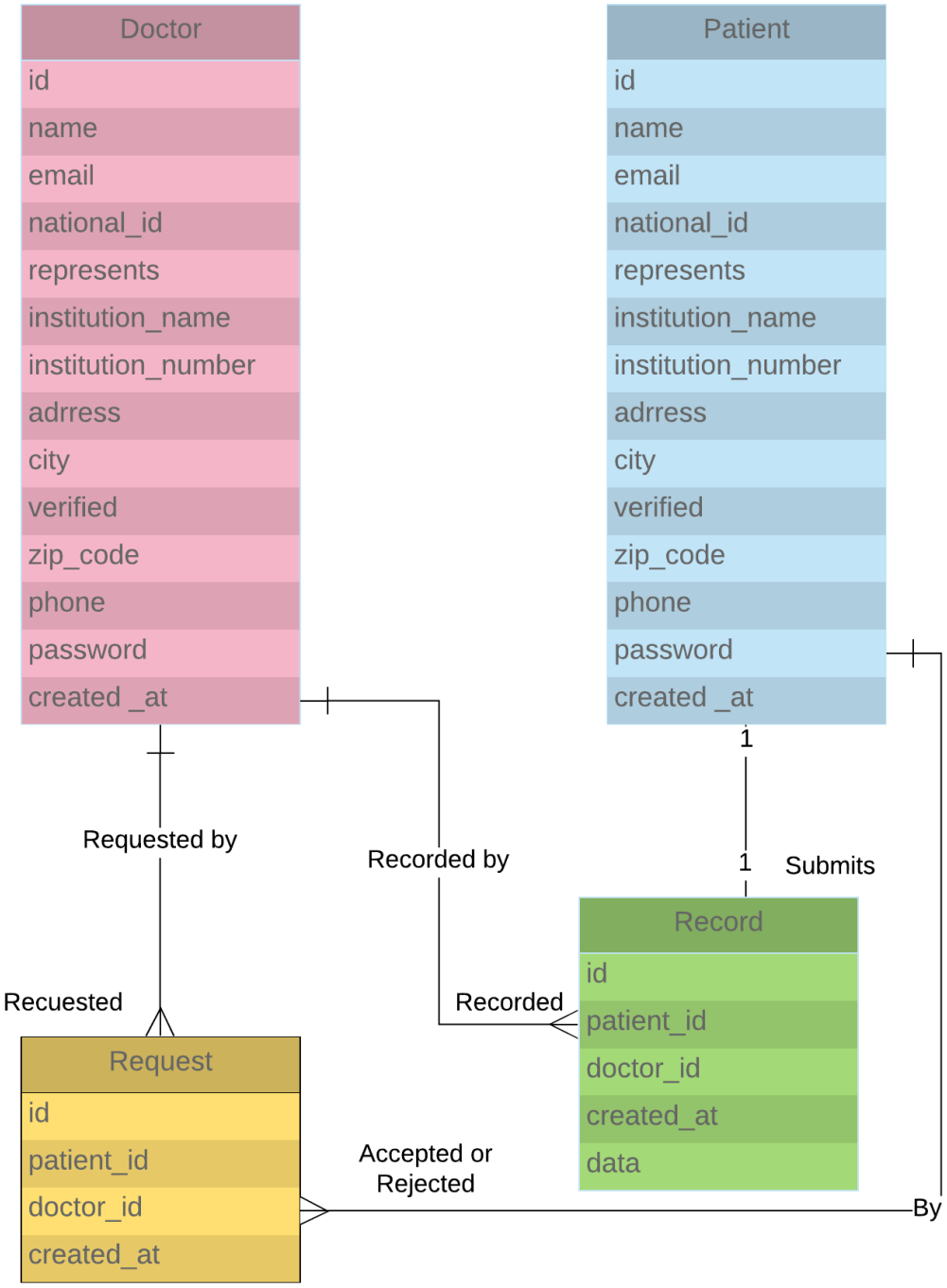


Figure 3.11 Entity Relationship Diagram

3.5 Class Diagram

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram.

The purpose of class diagram is to model the static view of an application. Class diagrams are the only diagrams which can be directly mapped with object-oriented languages and thus widely used at the time of construction. **And the purpose of class diagram can be summarized as :**

Analysis and design of the static view of an application, Describe responsibilities of a system, Base for component and deployment diagrams, And Forward and reverse engineering.

3.5.1 Class Notation

A class notation consists of three parts:

1. Class Name

- The name of the class appears in the first partition.

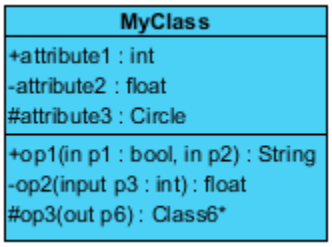
2. Class Attributes

- Attributes are shown in the second partition.
- The attribute type is shown after the colon.
- Attributes map onto member variables (data members) in code.

3. Class Operations **(Methods)**

- Operations are shown in the third partition. They are services the class provides.
- The return type of a method is shown after the colon at the end of the method signature.
- The return type of method parameters is shown after the colon following the parameter name.
- Operations map onto class methods in code

The graphical representation of the class can be draw like the figure below :



3.5.2 Class Relationships

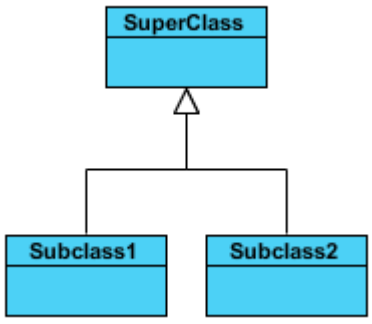
A class may be involved in one or more relationships with other classes. A relationship can be one of the following types:

1- Inheritance

Similarities often exist between different classes. Very often two or more classes will share the same attributes and/or the same methods. Because you don't want to have to write the same code repeatedly, you want a mechanism that takes advantage of these similarities. Inheritance is that mechanism. Inheritance models “is a” and “is like” relationships, enabling you to reuse existing data and code easily. When *A* inherits from *B,* we say *A* is the subclass of *B* and *B* is the super class of *A*. Furthermore, we say we have “pure inheritance” .

when *A* inherits all the attributes and methods of *B*. The UML modeling notation for inheritance is a line with a closed arrowhead pointing from the subclass to the super class.

Inheritance relationship shown in the figure below



2- Association

Objects are often associated with, or related to, other objects. For example, as you see in Figure 2 several associations

exist: Students are ON WAITING LIST for seminars, professors INSTRUCT seminars, seminars are an OFFERING OF courses, a professor LIVES AT an address, and so on. Associations are modeled as lines connecting the two classes whose instances (objects) are involved in the relationship.

When you model associations in UML class diagrams, you show them as a thin line connecting two classes, as you see in Figure blaw. Associations can become quite complex; consequently, you can depict some things about them on your diagrams. The label, which is optional, although highly recommended, is typically one or two words describing the association.

3- Aggregation

A special type of association. It represents a "part of" relationship.

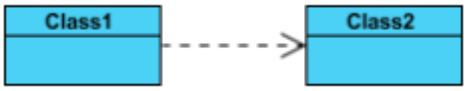
- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with an unfilled diamond at the association end connected to the class of composite.

4- Composition

Sometimes an object is made up of other objects. For example, an airplane is made up of a fuselage, wings, engines, landing gear, flaps, and so on.

5- Dependency

- Exists between two classes if the changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2
- A dashed line with an open arrow.



3.5.3 Class Diagram For Our Project

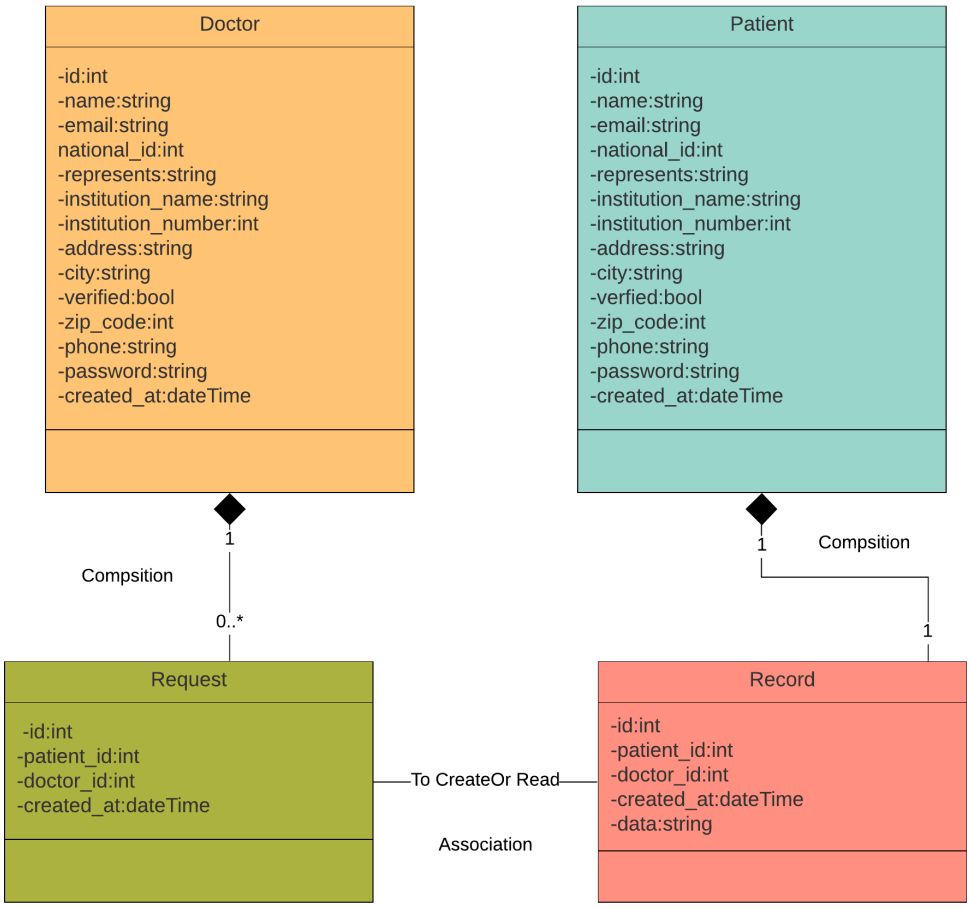
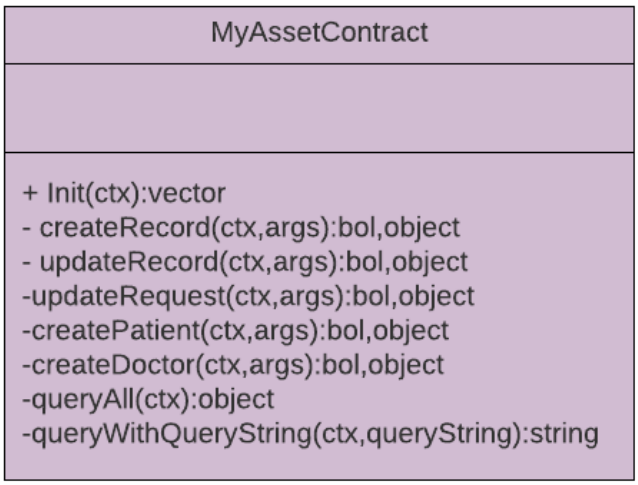


Figure 3.12 Class Diagram

After reading the class diagram for our project you must notice that there is no operations(methods) at any class ,and that is because that all methods in our project are defined in smart contract (**smart contract** defines the executable logic that generates new facts that are added to the ledger using the Hyperledger Fabric blockchain network).

We will discuss what is smart contracts, ledger and hyperledger and more in the next chapters specifically implementation and deployment chapters. but for now you can work with smart contracts

as a class that holds all of our operations that generates new facts and fulfills needs of our users(patient and doctor).



3.6 Chapter Conclusion

System analysis specifies "what the system should do. " And it's importance came from this question. As It helps you in identifying problems and organizing the facts and details of a system.

The definition of a systems analysis is a method of figuring out the basic elements of a project and deciding how to combine them in the best way to solve a problem.

An example of systems analysis is:

"deciding the best way to dry clothes without a clothes dryer such as by hanging them on a line in the sun".

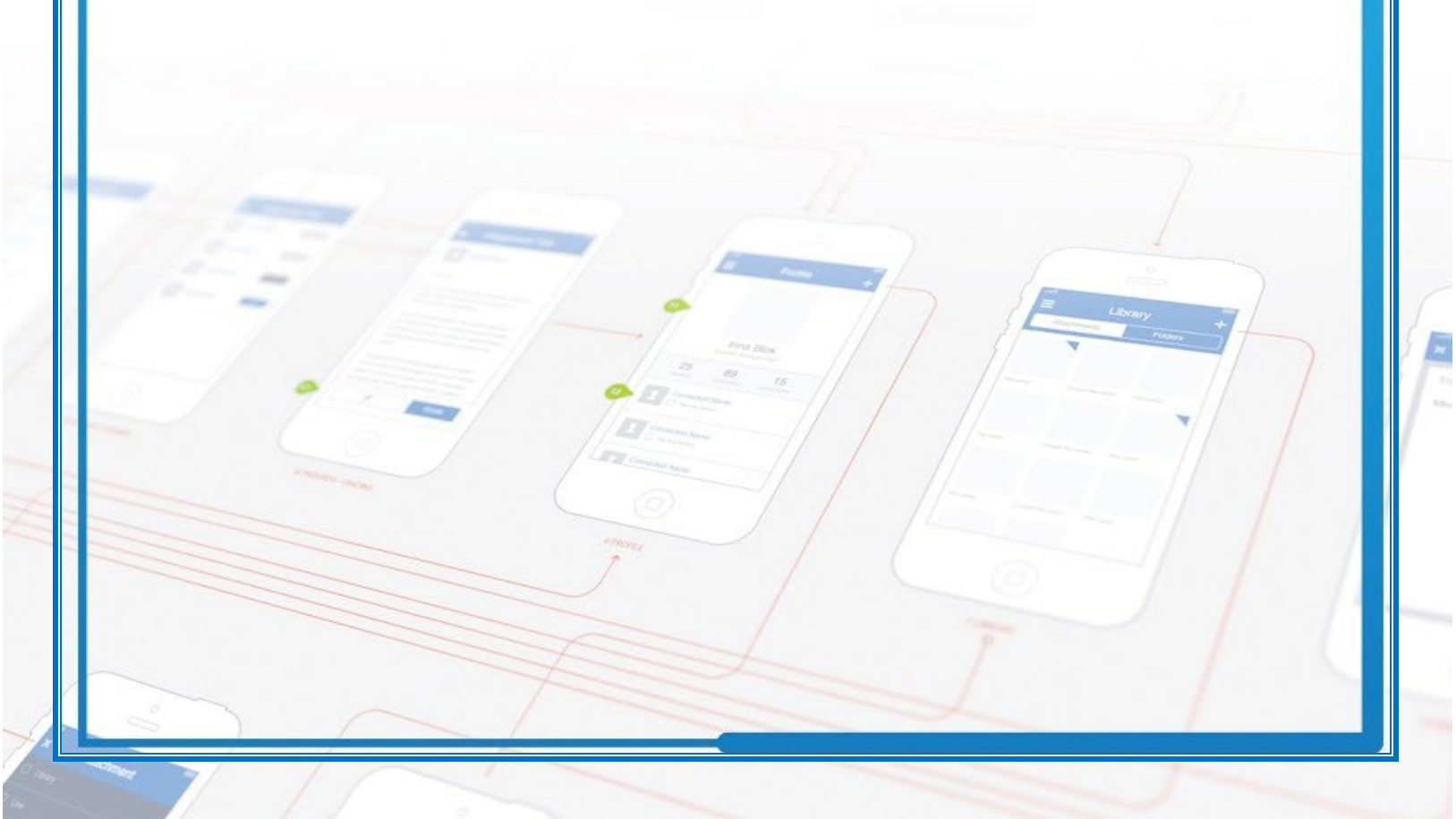
According to these importance system analysis must be use in every organization want a better solution for achieving their goals. as it is an efficient way to solve a problems and study any system.

Blockif

Your Health Comes First

Chapter 4

User Interface



In this chapter we will talk about our algorithm design, system's user interface with some screens from the application and how we think while creating it.

4.1 Database design

In this application there is no regular data base, it's based on blockchain technology where the system is secure and there is no way to access to the data or steal it the design of it comes this way.

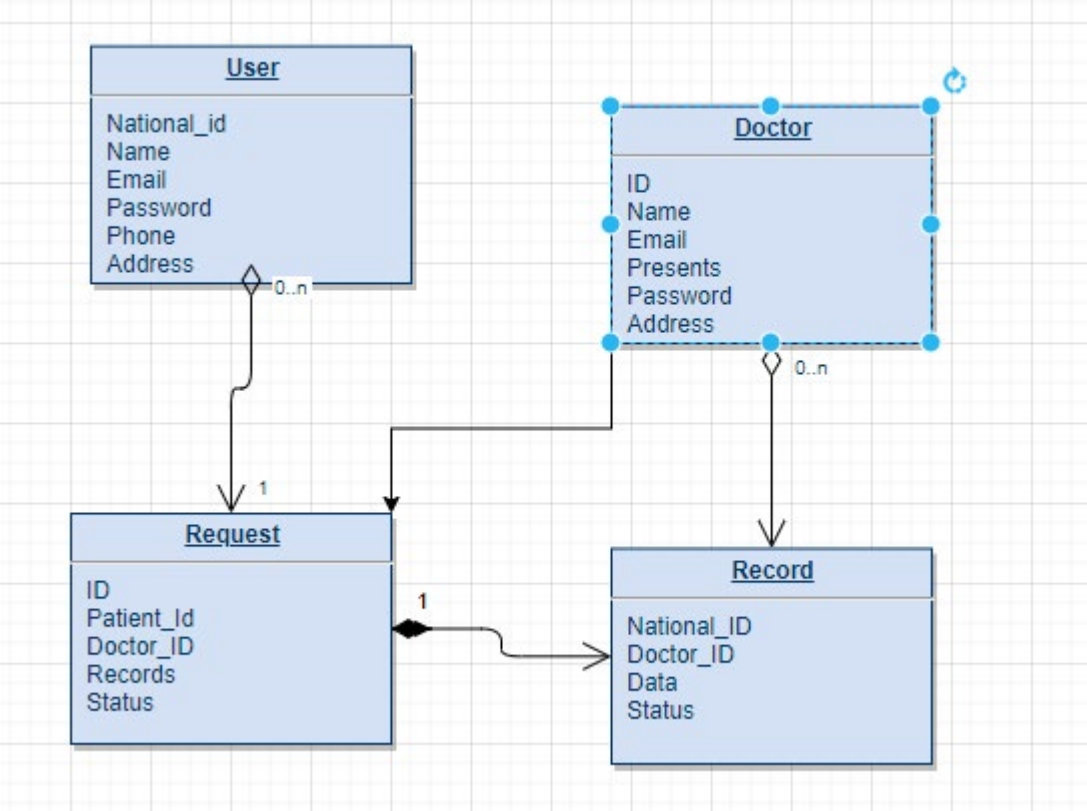


Figure 4.1 Database design

1- A user can have many records.

2- A doctor can create many records for user.

3- A user can have many requests.

4- Both doctor and user can send, accept, reject, and receive requests.

4.2 User Interface

What is UI ? User interface (UI) design is the process of making interfaces in software or computerized devices with a focus on looks or style. Designers aim to create designs users will find easy to use and pleasurable.

In information technology, the user interface (UI) is everything designed into an information device with which a person may interact. This can include display screens, keyboards, a mouse and the appearance of a desktop. It is also the way through which a user interacts with an application or a website. The growing dependence of many companies on web applications and mobile applications has led many companies to place increased priority on UI in an effort to improve the user's overall experience.

4.2.1 Evolution of UI

In early computers, there was very little user interface except for a few buttons at an operator's console. Many of these early computers used punched cards, prepared using keypunch machines, as the primary method of input for computer programs and data. While punched cards have been essentially obsolete in computing since 2012, some voting machines still use a punched card system.

The user interface evolved with the introduction of the command line interface, which first appeared as a nearly blank display screen with a line for user input. Users relied on a

keyboard and a set of commands to navigate exchanges of information with the computer. This command line interface led to one in which menus (lists of choices written in text) predominated.

4.2.2 UI vs. UX Design

Often confused with UX design, UI design is more concerned with the surface and overall feel of a design, whereas the latter covers the entire spectrum of the user experience. One analogy is to picture UX design as a vehicle with UI design as the driving console. In GUIs, you should create pleasing aesthetics and animations that convey your organization's values and maximize usability.

4.2.3 Blockify interface

4.2.3.1 Blockify initial page

Starts with a logo stands for Blockify.

Figure 4.2 initial page

4.2.3.2 Registration page

In this page we can log-in the app if we are already a member using our phone number and password and click on "sign in" button, in case you forgot your password you can click on the "forget password" button and re-enter it using your email or phone number, if it's the first time and we don't have an account we can click on "sign-up" button.

Figure 4.3 Registration

4.2.3.3 Sign-Up page

In this page we can creat a new account by entering our data then click "sign-up".

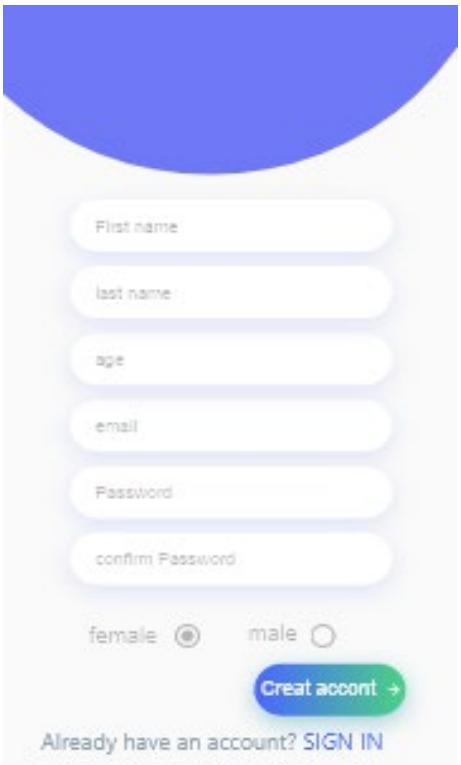


Figure 4.4 Sign-up page

4.2.3.4 Phone_Number page

The page where we add our phone number to verify the account by sending a message contains a code.

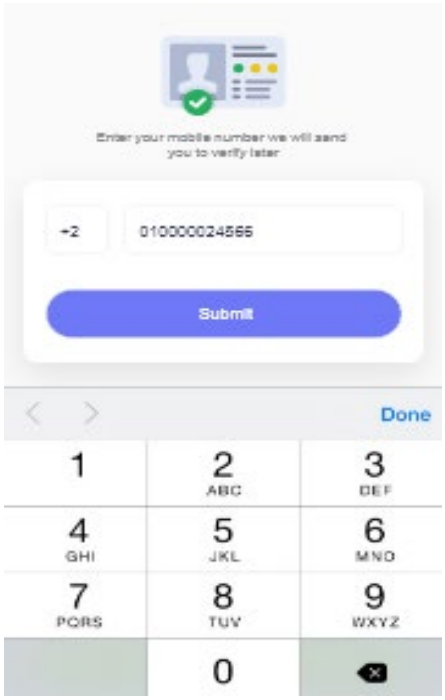
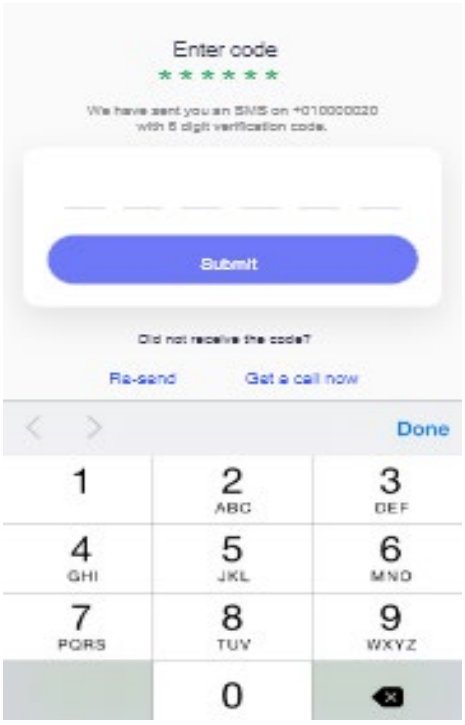


Figure 4.5 add-phone page



4.2.3.5 Verify page

Here we renter the code sent to the number the click "submit", or "re-send" if we don't get it.

Figure 4.6 Verify page

4.2.3.6 Home page

In this page there is a map to the closest hospital and clinic to your location and your latest medical record, in additional to your name, id and a photo.

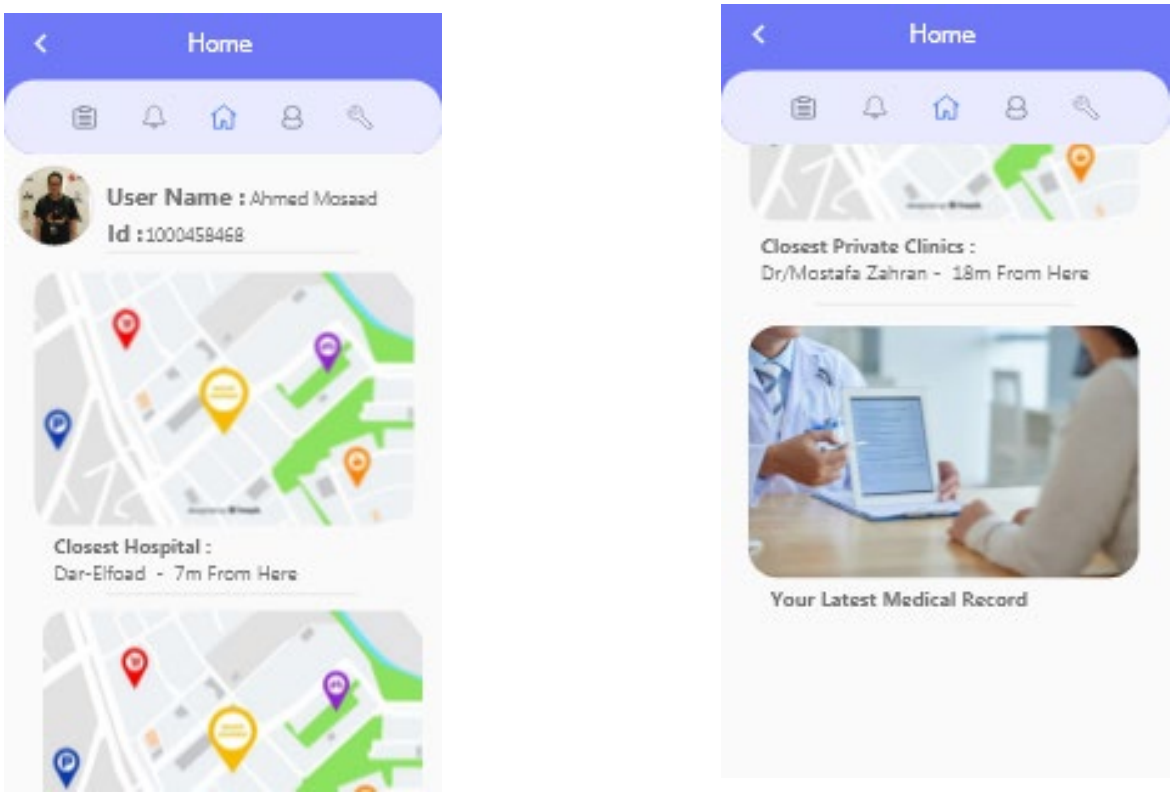


Figure 4.7 Home page

4.2.3.7 Medical_Records page

Here you can see your previous medical records the recent and old ones.

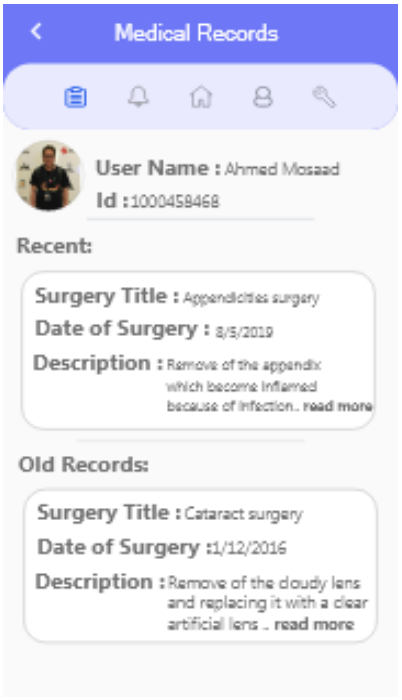


Figure 4.8 Medical Record page

4.2.3.8 Profile page

The page which contains the personal information like name, age, number, address , id ,mail, and if you are a doctor also you can add the doctor feature.

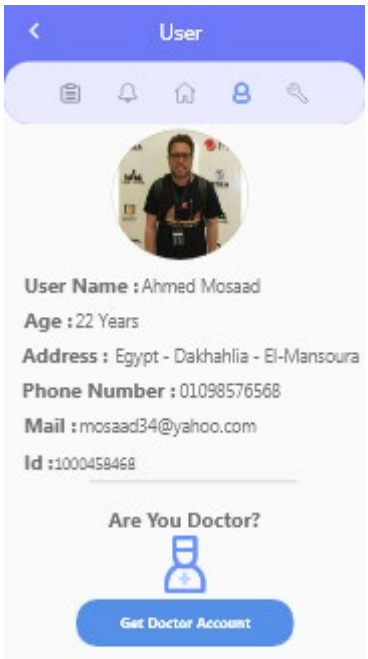


Figure 4.9 Profile page

4.2.3.9 Notification page

In this page you will find the news like if a doctor want to access you can accept or delete the request and also see the doctor profile.

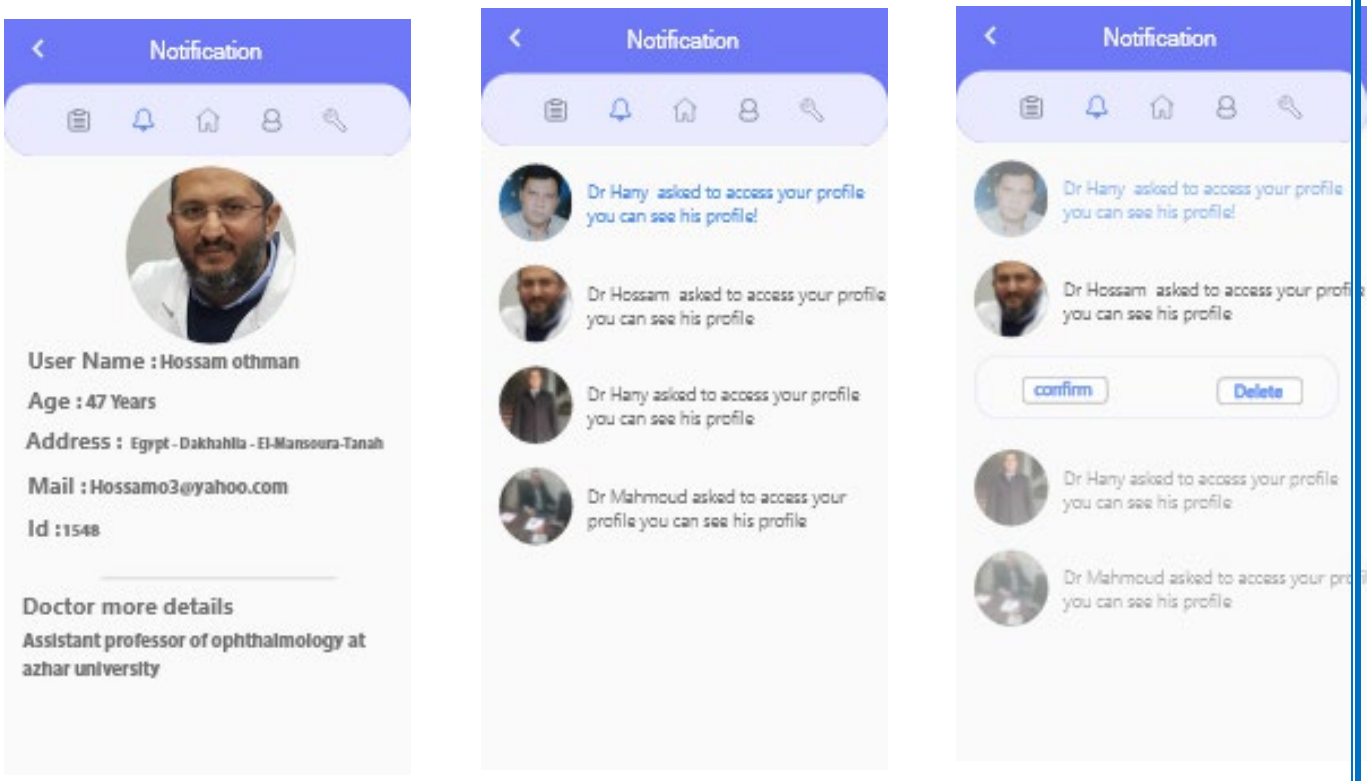
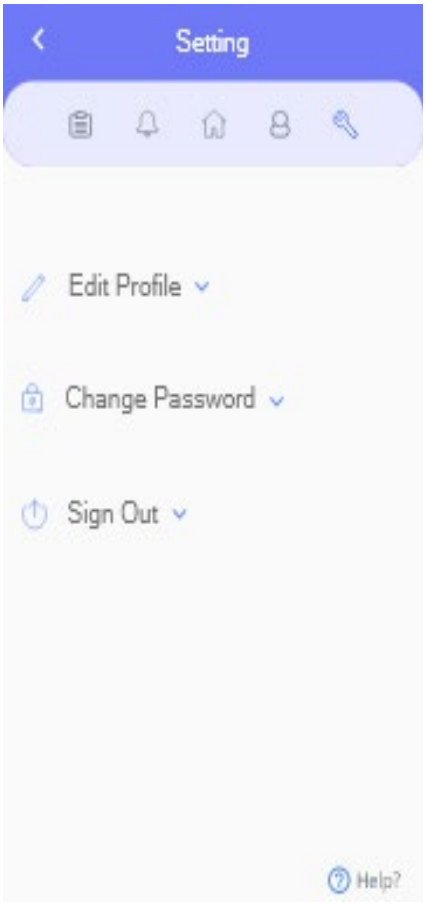


Figure 3.10 Notification page

4.2.3.10 Settings page

Here you can handle your information like editing your profile (name, E-mail, birth date, address, number), change password or log-out.



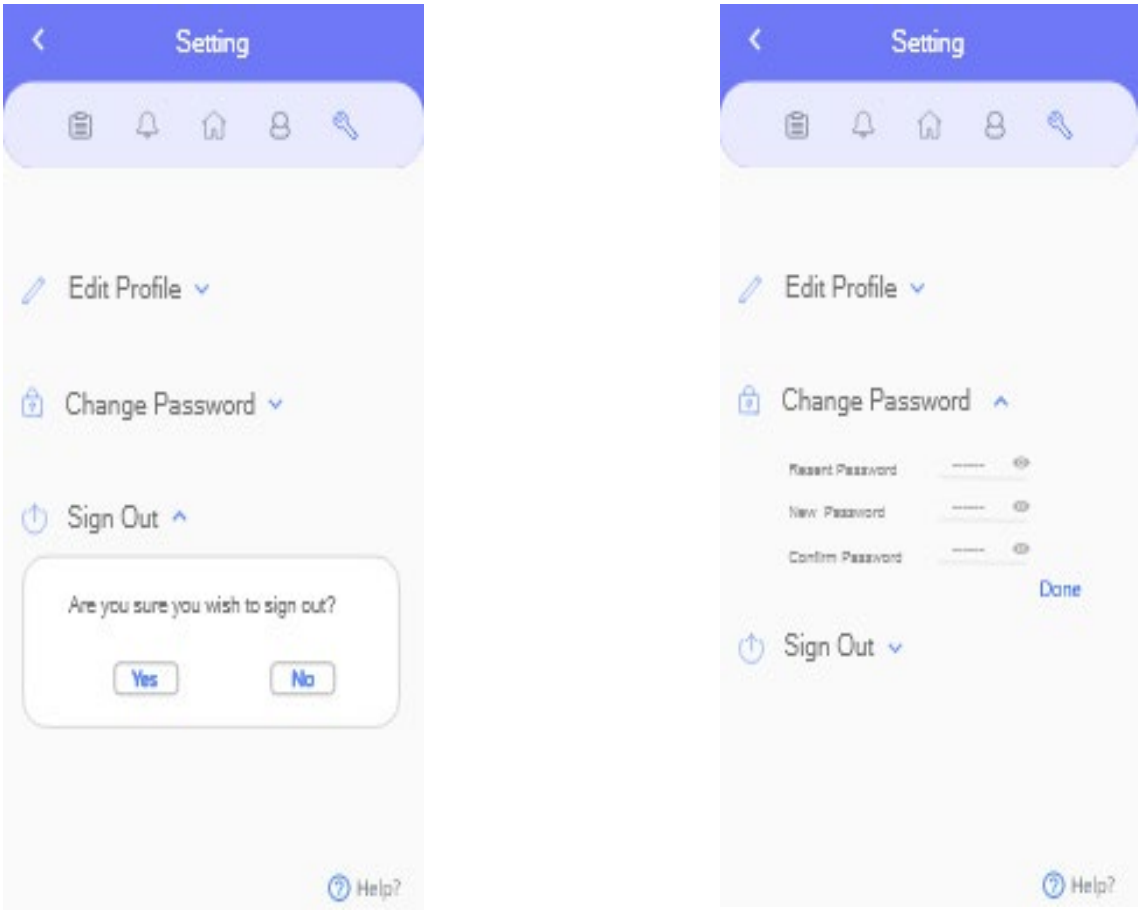


Figure 3.9 Setting page

4.3 Chapter Conclusion

In this chapter we talked about the ease of dealing with the project and achieve the best results through the user interface for all categories. We are looking forward to talking about a mobile application based on blockchain technology and it's use & interface and how to deal with it if you are a patient or a doctor.

There are some pictures illustrating the use of application in our project. At the design and development stage, we are interested in what is shown to the users. It is very important to talk about the user interface and its features, the user's view and aesthetic style and the stage of development from the beginning to the present. The interface of our project is characterized by the fact that it contains records , information and access to many features and medical services for all categories and satisfy all users such as the patient helpes him to interact with the doctor ,clinic, or the hospital.

Blockif

Your Health Comes First

Chapter 5

Implementation

In this chapter we will take about programs and programing languages we have used it in our project.

5.1 Application Anatomy

5.1.1 Application design

Adobe XD

Adobe XD is a vector-based user experience design tool for web apps and mobile apps, developed and published by Adobe Inc. It is available for macOS and Windows, although there are versions for iOS and Android to help preview the result of work directly on mobile devices. XD supports website wireframing and creating click-through prototypes and with XD the application design is created (like screens in Chapter 4: System Design).

5.1.2 Application Back-end

1. Node JS

Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside a

web browser. Node.js lets developers use JavaScript to write command-line tools and for server-side scripting running scripts server-side to produce dynamic web page content before the page is sent to the user's web browser. Consequently, Node.js represents a "JavaScript everywhere" paradigm, unifying web application development around a single programming language, rather than different languages for server- and client-side scripts.

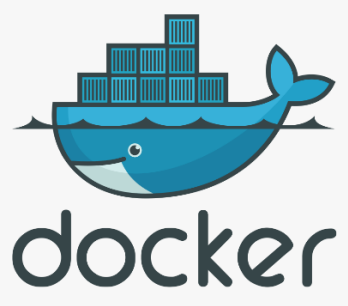
2. Express JS

Express JS, or simply Express, is a web application framework for Node.js, released as free and open-source software under the MIT License. It is designed for building web applications and APIs. It has been called the de facto standard server framework for Node JS.

3. JSON

JSON (JavaScript Object Notation) is a lightweight format for storing and transporting data, used to represent the data and help the application communicate on different devices.

4. Docker



Docker is an open-source tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow us to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications to be shipped with things not already running on the host computer. This gives a significant performance boost and reduces the size of the application, as we use it to create ready to deploy a virtual environment to host the application & the blockchain.

5. Hyperledger Fabric SDK

The Hyperledger Fabric SDK allows applications to interact with a Fabric blockchain network. It provides a simple API to submit transactions to a ledger or query the contents of a ledger with minimal code.

Hyperledger Fabric SDK helps in communicating with the blockchain through JavaScript Language.

6. gRPC

gRPC is a modern open-source high-performance RPC framework that can run in any environment. It can efficiently

connect services in and across data centers with pluggable support for load balancing, tracing, health checking, and authentication. It is also applicable in the last mile of distributed computing to connect devices, mobile applications, and browsers to backend services.

7. ES6

ES6 (ECMAScript6) is a scripting-language specification standardized by ECMA International, we using ECMA 6 Script for writing and styling JavaScript.

8. MongoDB

MongoDB is a cross platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schema, It used in the project for Storing cached data in JSON formats.

9. CouchDB

CouchDB is an open-source document-oriented NoSQL database, implemented in Erlang. CouchDB uses multiple formats and protocols to store, transfer, and process its data, it uses JSON to store data, JavaScript as its query language using MapReduce, and we use it In the project for caching data for the back end side use.

5.2 Sample Application Codes

5.2.1 Application endpoints

Import models and dependencies

In the first we import all models we create and all dependencies we need to make project run in good way.

```
'use strict';

const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const morgan = require('morgan');
const util = require('util');
const path = require('path');
const fs = require('fs');
const Patient = require('./models/Patient');
const Doctor = require('./models/Patient');
const Record = require('./models/Record');
const Request = require('./models/Request');
const bcrypt = require('bcrypt')
let network = require('./fabric/network.js');
const jwt = require('jsonwebtoken');
const auth = require('./auth/auth')
var multer = require('multer')
const app = express();
var upload = multer();
app.use(upload.array());
app.use(morgan('combined'));
app.use(express.json());
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(cors());

const configPath = path.join(process.cwd(), './config.json');
const configJSON = fs.readFileSync(configPath, 'utf8');
const config = JSON.parse(configJSON);
```

Figure 5.1 models & dependencies

Doctor register endpoint

Then we made endpoints like register for a normal

Application user (patient) as anyone using application is a patient, all have the same permission but the doctor have more permission like he can see the medical history of the patient, he can make transaction that contain the diagnosed of the patient.

```
app.post('/register-doctor', async (req, res) => {
  if (!req.body.name)
    return res.json({ 'msg': 'name is required' })
  if (!req.body.email)
    return res.json({ 'msg': 'email is required' })
  if (!req.body.national_id)
    return res.json({ 'msg': 'national_id is required' })
  if (!req.body.password)
    return res.json({ 'msg': 'password is required' })
  if (req.body.password != req.body.password_confirmation)
    return res.json({ 'msg': 'passwords do not match' })
  req.body.password = await bcrypt.hash(req.body.password, 10);
  const hash = req.body.password;
  var newUser = new Doctor({
    name: req.body.name,
    email: req.body.email,
    national_id: req.body.national_id,
    address: req.body.address,
    city: req.body.city,
    zip_code: req.body.zip_code,
    represents: req.body.represents,
    institution: {
      'institution_name': req.body.institution_name,
      'institution_number': req.body.institution_number
    },
    password: hash
  });
  req.body.id = newUser.id;
  req.body.created_at = newUser.created_at;
```

```
  req.body.updated_at = newUser.updated_at
```

Figure 5.2 doctor register 1

```
//first create the identity for the voter and add to wallet
  let response = await network.registerParticipant(newUser.id, newUser.name,
newUser.email,
    newUser.national_id, newUser.address, newUser.city,
    newUser.zip_code, newUser.password, newUser.created_at);
  console.log('response from registerParticipant: ');
  console.log(response);
  if (response.error) {
    res.send(response.error);
  } else {
    let networkObj = await network.connectToNetwork(newUser.id);
    console.log('networkobj: ');
    console.log(networkObj);

    if (networkObj.error) {
      res.send(networkObj.error);
    }
    console.log('network obj');
    console.log(util.inspect(networkObj));
    req.body = JSON.stringify(req.body);
    let args = [req.body];
    //connect to network and update the state with voterId
    console.log('beffore ')
    console.log(args)
    let invokeResponse = await network.invoke(networkObj, false, 'createDocto
r', args);
    if (invokeResponse.error) {
      res.send(invokeResponse.error);
    } else {
      console.log('after network.invoke ');
      let parsedResponse = JSON.parse(invokeResponse);
      res.send({ success: true, res: parsedResponse });
    }
  }
```

Figure 5.3 doctor register 2

As shown in figure 5-8 and 5-9 first we ask for the essential that we can make sure through it that user is the doctor as he will

be allowed to see the patient medical record history when they give him access, then after create the user we create the user identity to the voter and add it to the wallet and connect to network and update the network when he make any transaction and update the state with voted id with the last update.

Create Record endpoint

Another important endpoint is creating record that will allow the doctor to create a record to the patient.

```
  app.post('/create-record', auth, async (req, res) => {
    // To Do Validate Inputs
    if (!req.body.national_id)
      return res.json({ 'msg': 'national_id is required' })
    if (!req.body.data)
      return res.json({ 'msg': 'record data are required' })

    console.log(req.body)
    console.log(req.user)
    if (req.user.type != 'doctor')
       return res.json({ success: false, msg: 'User must be a doctor to creat
e a record' })
    // get patient
    let networkObj = await network.connectToNetwork('admin');
    console.log('networkobj: ');
    console.log(util.inspect(networkObj));
    if (networkObj.error) {
      res.send(networkObj);
    }
     var invokeResponse = await network.invoke(networkObj, true, 'queryByData
',
      JSON.stringify({ type: 'patient' }));

    var parsedResponse = await JSON.parse(JSON.parse(invokeResponse));

    if (parsedResponse.error) {
      res.send(parsedResponse.error);
    } else {
      if (parsedResponse.length == 0)
```

```
          return res.json({ success: false, 'msg': 'no patients with this nati
onal_id' })
```

Figure 5.4 create record 1

Create Request endpoint

Doctor also make request to the patient to see his medical record history profile, and the patient also can make request to the doctor and after the doctor accept the request he will be allowed also to se the patient medical record history profile .

```
   app.post('/create-request', auth, async (req, res) => {
     // To Do Validate Inputs
     if (!req.body.national_id)
       return res.json({ 'msg': 'national_id is required' })

     console.log(req.body)
     console.log(req.user)
     if (req.user.type != 'doctor')
      return res.json({ success: false, msg: 'User must be a doctor to create
a record' })
     // get patient
     let networkObj = await network.connectToNetwork('admin');
     console.log('networkobj: ');
     console.log(util.inspect(networkObj));
     if (networkObj.error) {
       res.send(networkObj);
     }
   }
```

Figure 5.5 create request

There are many other endpoints in the app.js like update record and update request and list all assets in the world state and this endpoint is called requests and there is also login endpoint that

used as a way to login the voter to the app and make sure they haven't voted before.

5.2.2 Smart Contract

Import Hyperledger Fabric 1.4 SDK

The fabric-contract-API provide the contract interface with a high-level API for application developers to implement Smart Contracts. Working with this API provides a high-level entry point to writing business logic.

Within Hyperledger Fabric, Smart Contracts can also be referred to as Chaincode. To be more specific, the term chaincode is preferred to be used to refer to the overall container that is hosting the contracts.

```
//import Hyperledger Fabric 1.4 SDK
const { Contract } = require('fabric-contract-api');
const path = require('path');
const fs = require('fs');
```

Figure 5.6 Import Hyperledger Fabric SDK

Then we import constructors and auxiliary function we create

```
//import our file which contains our constructors and auxiliary functio
n
let Request = require('./Request.js');
let Doctor = require('./Doctor.js');
let Patient = require('./Patient.js');
let Record = require('./Record.js');
```

Figure 5.7 constructors

The Next step is define our main contract and write all necessaries function, properties, assets that makes the application work properly

```
class MyAssetContract extends Contract
```

Figure 5.8 define Main Contract

The following figures will be some example of assets written in the our smart contract and the description of what they do will be in the comments in the code.

```
    //create patients
    let patient1 = await new Patient('1', 'eslam', 'koko@koko.com',
    '123456','Dakados -
 Saad Zagloul St','Mit Ghamr',true,'35611','01018211946',
     '$2y$10$V215oJ40PXyM57EFdl/3TOhZ0Wcy2vLhLItROh6ofxMV.Im9joyWa','today');
    let patient2 = await new Patient('2', 'eslam2', 'koko2@koko.com',
    '123456','Dakados -
 Saad Zagloul St','Mit Ghamr',true,'35611','01018211946',
     '$2y$10$V215oJ40PXyM57EFdl/3TOhZ0Wcy2vLhLItROh6ofxMV.Im9joyWa','today');
    //create doctor
     let doctor1 = await new Doctor('1', 'doctor', 'doctor@koko.com',
     '123456', 'doctor', '', '','Dakados -
 Saad Zagloul St','Mit Ghamr',true,'35611','01018211946',
      '$2y$10$V215oJ40PXyM57EFdl/3TOhZ0Wcy2vLhLItROh6ofxMV.Im9joyWa','today')
;
     let doctor2 = await new Doctor('2', 'doctor2', 'doctor2@koko.com',
     '123456', 'institution', 'Mit Ghamr Public Hospital', '2541233','Dakados
 - Saad Zagloul St','Mit Ghamr',true,'35611','01018211946',
      '$2y$10$V215oJ40PXyM57EFdl/3TOhZ0Wcy2vLhLItROh6ofxMV.Im9joyWa','today')
;
```

Figure 5.9 create patients and doctor

```
  async createRecord(ctx, args) {

    args = JSON.parse(args);
    // createRecord
```

```
    let record = await new Record(args.id, args.patient_id, args.doctor_id, a
rgs.created_at, args.data);


    // //update state with ballot object we just created
    await ctx.stub.putState(record.id, Buffer.from(JSON.stringify(record)));

    return {success:true,record:record}
  }
```

Figure 5.10 create record

```
  async createRequest(ctx, args) {

    args = JSON.parse(args);

    // createRecord
    let request = await new Request(args.id ,args.patient_id, args.doctor_id,
 args.created_at);
```

Figure 5.11 create request

```
  async createPatient(ctx, args) {

    args = JSON.parse(args);

    //create a new voter
    let patient = await new Patient(args.id, args.name, args.email, args.nati
onal_id ,args.address, args.city, args.verified, args.zip_code ,args.phone, a
rgs.password, args.created_at);

    //update state with new voter
    await ctx.stub.putState(patient.id, Buffer.from(JSON.stringify(patient)))
;
    return {success:true,patient};
  }
```

Figure 5-12 create patient

```
  async createDoctor(ctx, args) {

    args = JSON.parse(args);

    //create a new voter
    let doctor = await new Doctor(args.id,args.name,
```

```
      args.email,
      args.national_id ,
      args.represents,
      args.institution_name,
      args.institution_number,
      args.address,
      args.city,
      args.verified,
      args.zip_code ,
      args.phone,
      args.password,
      args.created_at);

    //update state with new voter
    await ctx.stub.putState(doctor.id, Buffer.from(JSON.stringify(doctor)));

    return {success:true,doctor};
  }
```

Figure 5-13 create patient 2

5.2.3 Network

A Fabric permissioned blockchain network is a technical infrastructure that provides ledger services to application consumers and administrators. In most cases, multiple organizations come together as a consortium to form the network and their permissions are determined by a set of policies that are agreed to by the consortium when the network is originally configured.

Here we come to the network where we make all we made connected together and connect the application with the blockchain server, first we will import Hyperledger fabric.

```
//Import Hyperledger Fabric 1.4 programming model - fabric-network
'use strict';
```

```
const { FileSystemWallet, Gateway, X509WalletMixin } = require('fabric-
network');
const path = require('path');
const fs = require('fs');
```

Figure 5.14 import fabric network

Then we connect to the config file and to the connection file and to the local fabric and to the contract we have been install in the peer

```
//connect to the config file
const configPath = path.join(process.cwd(), './config.json');
const configJSON = fs.readFileSync(configPath, 'utf8');
const config = JSON.parse(configJSON);
let connection_file = config.connection_file;
let gatewayDiscovery = config.gatewayDiscovery;
let appAdmin = config.appAdmin;
let orgMSPID = config.orgMSPID;
```

Figure 5.15 connect to config file

Here we connect the connection file and set the right encoding which is `utf8`

```
// connect to the connection file
const ccpPath = path.join(process.cwd(), connection_file);
const ccpJSON = fs.readFileSync(ccpPath, 'utf8');
const ccp = JSON.parse(ccpJSON);
```

Figure 5.16 Connect to connection file

Here we connect to contract through the gateway and connect to the deployed contract `patientDoctorContract`

```
    // Connect to our local fabric
    const network = await gateway.getNetwork('mychannel');

    console.log('Connected to mychannel. ');
    // Get the contract we have installed on the peer
    const contract = await network.getContract('patientDoctorContract');
```

Figure 5.17 Connect to local fabric and the peer

Here we connect to the blockchain wallets so every user can access his wallet

```
// Create a new file system based wallet for managing identities.
    const walletPath = path.join(process.cwd(), 'wallet');
    const wallet = new FileSystemWallet(walletPath);
    console.log(`Wallet path: ${walletPath}`);
    console.log(wallet);
```

Figure 5.18 create new wallet for managing identities

Here we register a new wallet for every new user with new identity

```
    // Check to see if we've already enrolled the user.
    const userExists = await wallet.exists(id);
    if (userExists) {
      let response = {};
      console.log(`An identity for the user ${id} already exists in the walle
t`);

      response.error = `Error! An identity for the user ${id} already exists
in the wallet. Please enter
        a different license number.`;

      return response;
    }


    // Check to see if we've already enrolled the admin user.
    const adminExists = await wallet.exists(appAdmin);
    if (!adminExists) {
      console.log(`An identity for the admin user ${appAdmin} does not exist
in the wallet`);

      console.log('Run the enrollAdmin.js application before retrying');
      let response = {};

      response.error = `An identity for the admin user ${appAdmin} does not e
xist in the wallet.
        Run the enrollAdmin.js application before retrying`;
      return response;
```

```
    }
```

Figure 5.19 create new wallet for managing identities – Chaking

Here we use the admin organization to register the new identity in the blockchain as the blockchain is permissioned

```
 // Create a new gateway for connecting to our peer node.
    const gateway = new Gateway();
    await gateway.connect(ccp, { wallet, identity: appAdmin, discovery: gatew
ayDiscovery });

    // Get the CA client object from the gateway for interacting with the CA.
    const ca = gateway.getClient().getCertificateAuthority();
    const adminIdentity = gateway.getCurrentIdentity();
    console.log(`AdminIdentity: + ${adminIdentity}`);
```

Figure 5.20 create new gateway

Here we make sure that the new user has registered with new identity and new wallet in case something didn't go as expected we raise an error exception.

```
// Register the user, enroll the user, and import the new identity into
wallet
    const secret = await ca.register({ affiliation: 'org1', enrollmentID: id,
 role: 'client' }, adminIdentity);

    const enrollment = await ca.enroll({ enrollmentID: id, enrollmentSecret:
secret });
    const userIdentity = await X509WalletMixin.createIdentity(orgMSPID, enrol
lment.certificate, enrollment.key.toBytes());
    await wallet.import(id, userIdentity);
    console.log(`Successfully registered voter ${name}.`);
    let response = {success:true,msg:`Successfully registered voter ${name}.`
};
    return response;
  } catch (error) {
    console.error(`Failed to register user + ${id} + : ${error}`);
    let response = {};
    response.error = error;
    return response;
  }
```

Figure 5.21 Register user into wallet

5.3 Behind the scene

5.3.1 Hyperledger

One of the most famous technologies used for private blockchains is Hyperledger, which was developed by the Linux Foundation, IBM and others, and intends to make the private blockchain implementations easier and more versatile. Hyperledger is open-source and it is not a tool or technology, rather it refers to a project developed by many teams, and has to do with distributed ledger technology (DLT).

Currently, there are four active DLT frameworks in the Hyperledger umbrella, the Sawtooth, the Iroha, the Fabric (which we will mainly see), and the Burrow. Besides that there are four major tool projects:

•Composer: Used for easy creation and deployment of smart contracts in the Hyperledger network, using a high-level language.

•Explorer: Helps with the exploration and visibility over an existing and running Hyperledger network. It is like a block explorer and it is web-based.

•Quilt: Helps with the interoperability of different blockchain networks.

•Cello: Helps with the simplification of the creation and management of blockchains, in a user-friendly environment.

Hyperledger, as a private blockchain development technology, can be used in certain situations, in order to set permissions and have a more secure implementation overall, but it is not intended for every user. The most important characteristics of Hyperledger are the following:

1.No cryptocurrency. In Hyperledger, there is no cryptocurrency needed in order to interact and use the network. In a public blockchain, this is used as an incentive and a way to give something to people that secure the network. Because of the permissioned character and the alternative consensus models, a currency is not mandatory in Hyperledger.

2.Channels. In Hyperledger, channels work as different networks that exist on top of the main network. Channels exist in order to be able to perform confidential transactions between parties that have the authority to be in this channel and do not want other users outside of it to be able to see them. This helps in the privacy and security of the network and can be easily implemented.

3.Permissioned. The Hyperledger networks are permissioned, which means that the owner of the network can control who can get inside and be part of the network, unlike public blockchains, which are permissionless and

everyone can join them. By that, only trusted users can join the network and interact with the blockchain of it, which drastically increases the security of the system.

4.Programmable. Much like Ethereum's smart contracts, Hyperledger has the ability to have a decentralized code called chaincode, which automates the business processes. Users can execute the chaincode on the context of transactions and later the outcome will get recorded in the ledger of the specific channel.

5.3.2 Create a fair e-voting application

Have you ever wondered how exactly the votes in a presidential election counted? What if instead of having volunteers that are spending hours a day counting votes manually, we have an app that was backed by blockchain, recording each vote made by a voter, ensuring double-voting is not possible? That's what this code pattern explains how to do. We aim to build a web-app in which the voter can register with their drivers license, get a unique voter-Id which is used to login to the app, and cast the vote. The vote is tallied on the blockchain, and the web-app shows the current standings of the polls.

Voting using Public Key Infrastructure

At the start of the application, the user registers to vote by providing their driver's license number, registrar district, and first and last name. In this step, we can check to see if the driver's license is valid, and has not been registered previously.

If all goes well, we create a private and public key for the voter with our certificate authority that is running on the cloud and adds those keys to the wallet.

After that, we use our driver's license number to submit our vote, during which the application checks if this driver's license number has voted before and tells the user they have already submitted a vote if so. If all goes well, the political party which the voter has chosen is given a vote, and the world state is updated. The application then updates our current standings of the election to show how many.

votes each political party currently has.

Since each transaction that is submitted to the ordering service must have a signature from a valid public-private key pair, we can trace back each transaction to a registered voter of the application, in the case of an audit.

In conclusion, although this is a simple application, the developer can see how they can implement a Hyperledger Fabric web-app to decrease the chance of election-meddling, and enhance a voting application by using blockchain technology.

5.3.3 Debugging

First thing first is to install requirement tool .

Debug Backend

- Node JS (v8.x or greater) [22]

- npm (v5.x or greater) [23]
- Visual Studio Code [24]
- IBM Blockchain Platform Extension for VSCode [25]
- Download the pre-requists IBM Extension
- Docker [26]

Debug the UI

- check flutter's own get started page [27]
- Open the flutter submodule in VS Code (preferred) or android studio
- press F5

working with the contract files

i. open /backend/contract folder in VS code
ii. index.js is the file that represents the smart contract entry point.
iii. package.json contains the chaincode version and dependencies, remember to change the version after making changes.
iv. to package the current version of the chain code into a .cds file, right-click on SMART CONTRACT in the extension menu and Package Open Project

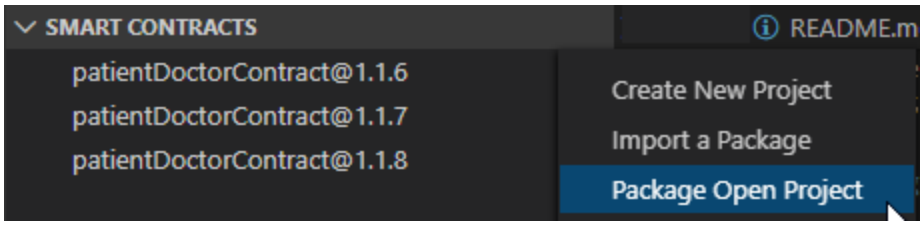
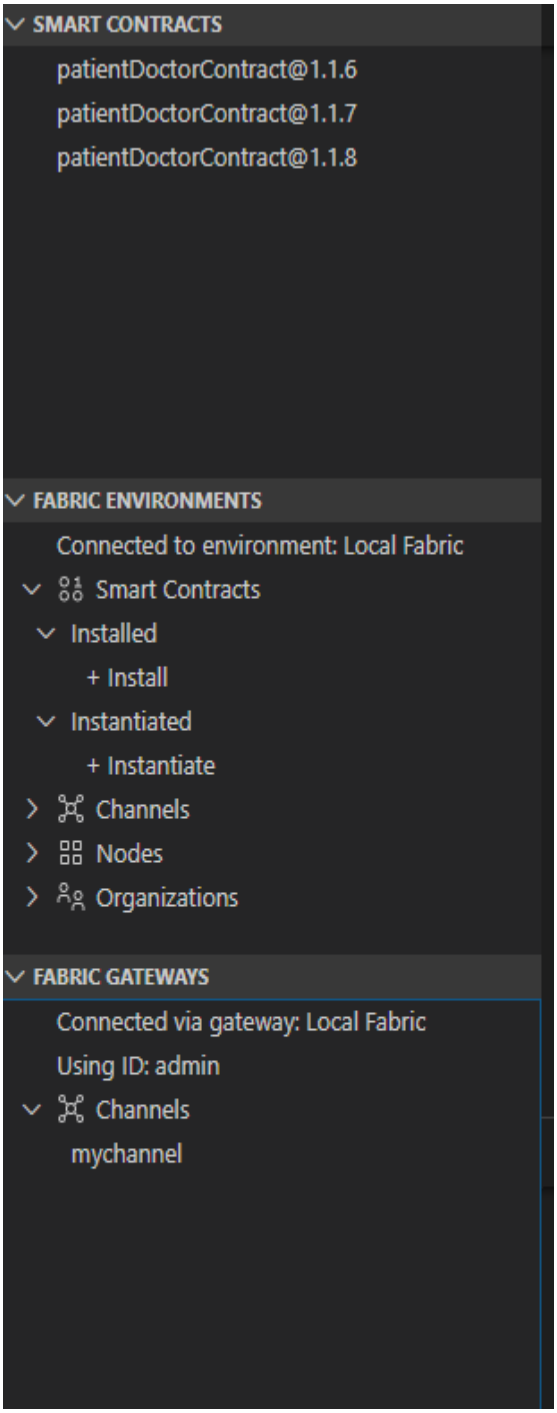


Figure 5.22 vs-code-package-chaincode



Working with the server files

I. in the extension menu under Fabric Environments, click Local Fabric (click to start)

II. it will download a test network of 1 peer that can be used for debugging chaincode

III. Under Fabric Gateways, click on Local Fabric, if it connects, it means that the test network is up and running the extension should look something like figure 5.26.

Figure 5.23 vs-code-after-activation

VI. install and instantiate the chaincode

VII. Under the Local Fabric network, click Install and select the latest packaged chaincode version, the click instantiate

(agree on default parameters, just press Enter)

VIII. under Fabric Wallets in the extension, right-click on the Org1 wallet, and export wallet put it in web-app/server/peer-wallet

IX. go to backend/web-app/server/src/

X. run npm start, you should see a similar output to this

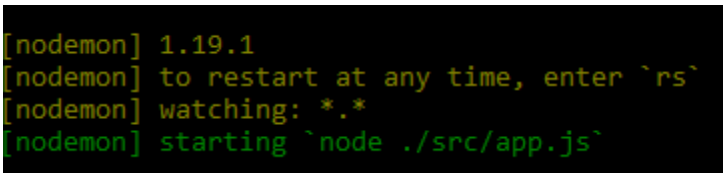


Figure 5.24 cmd-start-server

XI. now everything is ready, you can now send requests to the test blockchain! you can use Postman to send requests, by adding the collection in Blockify.postman_collection.json

5.4 System Testing

5.4.1 Unit Testing

UNIT TESTING is a level of software testing where individual units and components of the software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output. In procedural programming, a unit may be an individual program,

function, procedure, etc. In object-oriented programming, the smallest unit is a method, which may belong to a base/ super class, abstract class or derived/ child class. (Some treat a module of an application as a unit. This is to be discouraged as there will probably be many individual units within that module.) Unit testing frameworks, drivers, stubs, and mock/ fake objects are used to assist in unit testing.

Hyperledger Fabric chaincode unit testing

Testing stage is a critical requirement for software quality assurance, CCKit[28], a programming toolkit for developing and testing Hyperledger Fabric Golang chaincode, enhances the development experience with an extended version of MockStub for chaincode testing.

Chaincode (smart contract) testing

Tests must ensure that chaincode works as expected:

- particular input payload leads to a particular business object state change
- particular (invalid) input payload leads to validation or other errors
- particular object state allows a subset of state transition (state machine)

Any software testing (chaincode or web application for example) may either be a manual or an automated process.

Manual software testing is led by a team or individual who will manually operate a software product and ensure it behaves as expected. In the case of chaincode tests you can manually invoke chaincode via peer CLI tools.

Automated software testing is the practice of instrumenting input and output correctness checks for individual units of code. During automated testing, code are executed in a test environment with simulated input.

Chaincode DEV mode

Deploying a Hyperledger Fabric blockchain network, chaincode installing and initializing, is quite complicated to set up and a long procedure. Time to re-install / upgrade the code of a smart contract can be reduced by using chaincode dev mode. Normally chaincode is started and maintained by peers. In “dev” mode, chaincode is built and started by the user. This mode is useful during chaincode development phase for rapid code/build/run/debug cycle turnaround. However, the process of updating the code will still be slow.

MockStub - mocked chaincode stub

Mocking is a unit testing phenomenon that helps to test objects in isolation by replacing dependent objects with complex behavior with test objects with pre-defined/simulated behavior. These test objects are called Mock objects.

The shim package contains a MockStub implementation that wraps calls to a chaincode, simulating its behavior in the HLF peer environment. MockStubdoes don't need to start multiple Docker containers with the peer, world state database, chaincodes and allows them to get test results almost immediately.

MockStub essentially replaces the SDK and peer environment and allows to test chaincode without actually starting your blockchain network. It implements almost every function the actual stub does but in memory.

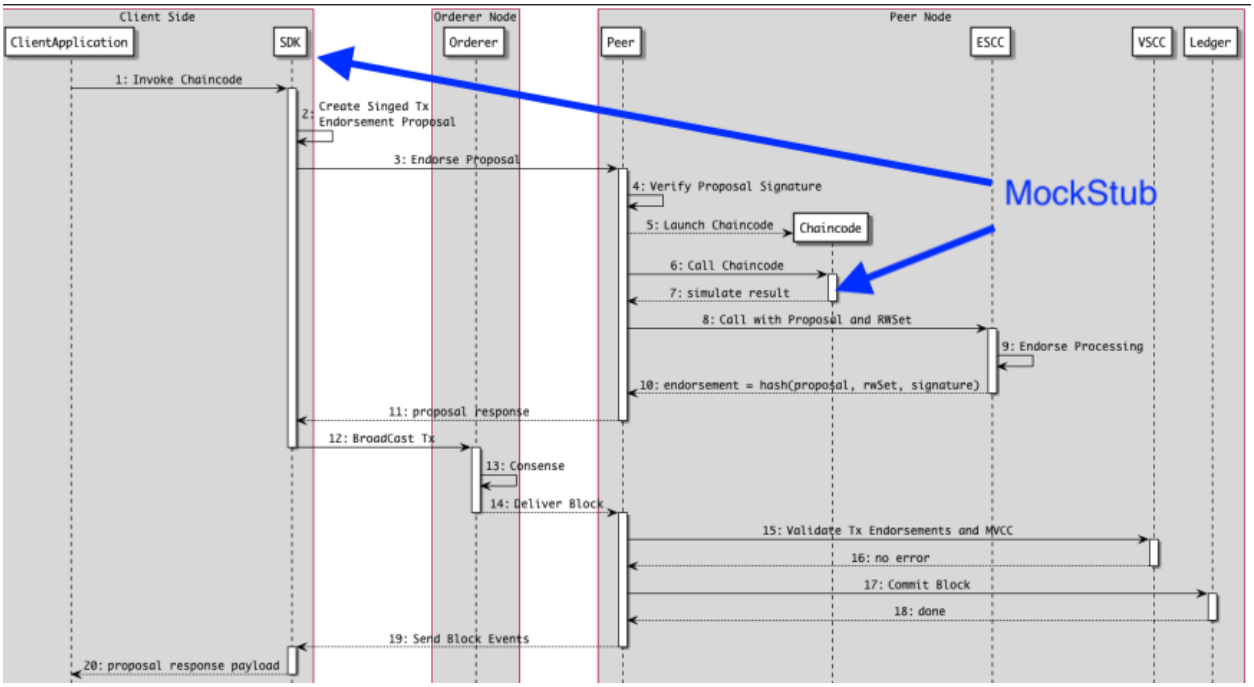


Figure 5.25 MockStub

CCKit MockStub

CCKit testing package contains:

- MockStub with implemented GetTransient and other methods and event subscription feature
- Test identity creation helpers
- Chaincode response expect helpers

Chaincode interface functions described in file chaincode.go, so we can see all possible operations (transactions) with chaincode data:

```
Query("list", queryCPapers)

    // Get method has 2 params - commercial paper primary key components
    Query("get", queryCPaper, defparam.Proto(&schema.CommercialPaperId{}))
.
    Query("getByExternalId", queryCPaperGetByExternalId, param.String("ext
ernalId")).

    // txn methods
    Invoke("issue", invokeCPaperIssue, defparam.Proto(&schema.IssueCommerc
ialPaper{})).
    Invoke("buy", invokeCPaperBuy, defparam.Proto(&schema.BuyCommercialPap
er{})).
    Invoke("redeem", invokeCPaperRedeem, defparam.Proto(&schema.RedeemComm
ercialPaper{})).
    Invoke("delete", invokeCPaperDelete, defparam.Proto(&schema.Commercial
PaperId{}))
```

Figure 5.26 Test query

During tests we can check Response attribute:

- Status (error or success)
- Message string (contains error description)
- Payload contents (marshaled JSON or Protobuf)

For example, we can simply test that Init method (invoked when the chaincode is initialized) returns successful status code.

```
BeforeSuite(func() {
    // Init chaincode with admin identity

    adminIdentity, err := testcc.IdentityFromFile(MspName, `testdata/admin
.pem`, ioutil.ReadFile)
    Expect(err).NotTo(HaveOccurred())

    expectcc.ResponseOk(
        paperChaincode.
            From(adminIdentity).
            Init())
})
`
```

Figure 5.27 Test Function

Test the "Issue" method

We expect that invocation of issue chaincode method will result in:

- response with Ok status
- event IssueCommercialPaper is fired

In the test, we can invoke the issue method via MockStub, check response status, and check chaincode event. Chaincode events can be received from chaincodeEventsChannel. The BeEquivalentTo method of the expected functionality comes in handy to compare the event payload.

```
It("Allow issuer to issue new commercial paper", func(done Done) {
    //input payload for chaincode method
    issueTransactionData := &schema.IssueCommercialPaper{
```

```
        Issuer:       IssuerName,
        PaperNumber:  "0001",
        IssueDate:    ptypes.TimestampNow(),
        MaturityDate: testcc.MustProtoTimestamp(time.Now().AddDate(0, 2, 0
)),
        FaceValue:    100000,
        ExternalId:   "EXT0001",
    }

    // we expect tha `issue` method invocation with particular input paylo
ad returns response with 200 code
    // &schema.IssueCommercialPaper wil automatically converts to bytes vi
a proto.Marshall function
    expectcc.ResponseOk(
        paperChaincode.Invoke(`issue`, issueTransactionData))

    // Validate event has been emitted with the transaction data
    Expect(<-
paperChaincode.ChaincodeEventsChannel).To(BeEquivalentTo(&peer.ChaincodeEv
ent{
        EventName: `IssueCommercialPaper`,
        Payload:   testcc.MustProtoMarshal(issueTransactionData),
    }))

    // Clear events channel after a test case that emits an event
    paperChaincode.ClearEvents()
    close(done)
}, 0.1)
```

Figure 5.28 Test Issue

After we test all function and issues we will test the application with fill up input with real data to see if it work in proper way or not and the test in succeed.

```
'use strict';
const { ChaincodeStub, ClientIdentity } = require('fabric-shim');
const { MyAssetContract } = require('..');
const winston = require('winston');

const chai = require('chai');
const chaiAsPromised = require('chai-as-promised');
const sinon = require('sinon');
const sinonChai = require('sinon-chai');
```

```
chai.should();
chai.use(chaiAsPromised);
chai.use(sinonChai);

class TestContext {
  constructor() {
    this.stub = sinon.createStubInstance(ChaincodeStub);
    this.clientIdentity = sinon.createStubInstance(ClientIdentity);
    this.logging = {
      getLogger: sinon.stub().returns(sinon.createStubInstance(winston.cre
ateLogger().constructor)),
      setLevel: sinon.stub(),
    };
  }
}
describe('MyAssetContract', () => {

  let contract;
  let ctx;
  beforeEach(async () => {
    contract = new MyAssetContract();
    ctx = new TestContext();
    ctx.stub.getState.withArgs('1001').resolves(Buffer.from('{"value":"my
asset 1001 value"}'));
    ctx.stub.getState.withArgs('1002').resolves(Buffer.from('{"value":"my
asset 1002 value"}'));
  });
});
```

Figure 5.29 Test with real data

5.5 Chapter Conclusion

In this chapter, we talked about programs and programming languages and their great contribution to the progress we have achieved in our time and contribution to the development of our idea. The most important of which is Hyperledger Fabric Chaincode which allows components, such as consensus and membership services, to be plug-and-play. Its modular and versatile design satisfies a broad range of industry

use cases. It offers a unique approach to a consensus that enables performance at scale while preserving the privacy and this is what we need beside other technologies we mention to make the best way to save the privacy of the patient and the doctor.

Blockif

Your Health Comes First

Chapter 6

Conclusion and Future work

6.1 Conclusion

There are problems in the Medical industry and health care systems as, many people die due to misdiagnosis from the doctors, which is mainly caused by the lack of an accurate comprehensive medical history of the patient. currently, the patient is the one responsible to tell his own history to the doctors, carrying medical bills, radiology scans, and lab results everywhere he goes. And According to a study conducted by the American Academy of Allergy Asthma & Immunology is 2% to 5% of the population that uses health care may have multiple drug intolerance syndrome.

All this main problem will be solved when creating a system that store medical record history of all patient and of course there are many of them and they have deadly issues as they using the regular database and here blockchain technology come as a hero to solve all this problem here we create **BlockifY**.

Blockify aims to provide patients with a secure and permissioned way to store their medical records and share it with their doctors.

We aim to do that using a Permissioned Blockchain Network as a backend, which provides an immutable history that ensures no one can tamper with the data, Our Mission is creating a secure system that records the history of the patients using blockchain technolog to connect patients and medical institutes from all of the world through our system, And our Vision and the passion that motivates us is To make all hospitals and

patients connected digitally with each other without using paper and make a medical record history for each person.

6.2 Future Work

We will seek to achieve the following goals to improve our project for the better:

- Implement desktop application for hospitals.
- Integration with IOT devices like smartwatches and health cares devices to provide the medical record profile with all necessary additional data.
- Provide an emergency system for patients with chronic diseases to remember taking the medicine.
- Integration with smart assistants to remember their Dates with doctors and remember them taking the medicine and assist doctors also.
- Enrich the website with large data and use it with some algorithms to work to predict and organize doctors time table and make the priority for critical cases first.

Refernces:

1. https://www.nursingcenter.com/cearticle?an=00006205-201404000-00006&Journal_ID=54012&Issue_ID=2409928

2. https://www.hopkinsmedicine.org/news/media/releases/study_suggests_medical_errors_now_third_leading_cause_of_death_in_the_us

3.https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2277113/

4. https://bitcoin.org/bitcoin.pdf

5. https://raft.github.io/

6. https://nodejs.org/en/download/

7. https://www.thestudentroom.co.uk/showthread.php?t=4334666

8. https://quizlet.com/21945369/chapter-3-hardware-cpu-flash-cards/

9. https://study.com/academy/lesson/systems-development-life-cycles-software-development-process.html

10.http://www.advocatedocs.com/app-more/

11. https://www.commhealth.org/

12. https://tricare.mil/About

13. https://www.bcbs.com/about-us/the-blue-cross-blue-shield-system

14. https://www.premera.com/visitor/about-premea

15. https://airbrake.io/blog/sdlc/what-is-system-development-life-cycle

16. https://www.lucidchart.com/blog/data-flow-diagram-tutorial

17. https://en.wikipedia.org/wiki/Use_case_diagram

18. https://creately.com/blog/diagrams/sequence-diagram-tutorial/

19. **Learn UM: in 1 Day Book By Runtgta**

20. **Flowchart (Plain and simple) Book by Joiner Associates**

21. **Database Design Using Entity-Relationship Diagrams (Foundations of Database Design) Book by Sikha Bagui**

22. https://nodejs.org/en/download/

23. https://www.npmjs.com/get-npm

24. https://code.visualstudio.com/download

25. https://github.com/IBM-Blockchain/blockchain-vscode-extension

26. https://www.docker.com/products/docker-desktop

27. https://flutter.dev/docs/get-started/install/windows

28. https://github.com/s7techlab/cckit

Chapter 4 is based on: UI is Communication Book by Everett N. McKay

ملخص رسالة : نظام إدارة السجلات الطبية بإستخدام تقنية البلوك تشين

تتمحور فكرة المشروع حول إنشاء نظام لإدارة السجلات الطبية بإستخدام تقنية البلوك تشين المعروف باسم بلوكيفاي ، حول إنشاء نظام يحتوي على السجلات الطبية للمرضى،و حفظها على شبكة البلوك تشين ،وذلك لعدة أسباب أولها المحافظة على الأرواح ،والتقليل من نسب الوفيات الناتجة عن اتخاذ قرارات طبية خاطئة بسبب عدم معرفتهم بالحالة الصحية، و التاريخ المرضي للمرضى . أيضا معرفة إذا كان المريض يعاني من أي نوع من أنواع الحساسية تجاه مادة فعالة في تصنيع الدواء، أو أي شئ آخر و ذلك يساعد الطبيب المعالج في معرفة الحالة الكاملة للمريض، و اتخاذ القرار المناسب له ، وسبب آخر مهم أيضا وهو أن الأنظمة التي تعمل على حل هذه المشكلة و التي تعمل باستخدام قواعد بيانات مركزية ،أو باستخدام دفاتر الأوراق في المستشفيات، هو أن بسهولة يمكن التلاعب بالبيانات سبب من الأسباب أو تلفها أو سرقتها ما تحتويه علي خصوصيات لأي للمرضى ، سواء تم سرقتها بشكل ورقي أو تم اختراقها عبر المواقع و وهو تطبيق على الهواتف "بلوكيفاي" السيرفرات ، و لهذه اتجهنا لإنشاء المحمولة وتم برمجته و برمجة قواعد البيانات الخاصة به باستخدام تقنية البلوك تشين ،وذلك لكثرة المزايا في هذه التقنية حيث سوف يتمكن المريض من إعطاء الإذن للطبيب المعالج لرؤية سجله المرضي، و لن يستطيع أي شخص آخر النظر في السجل المرضي لأي شخص كان الا مستشفيات الطوارئ فقط هي التي تملك الإذن للسجلات المرضية لأي شخص، و ذلك من أجل إنقاذ حياتهم لأن المرضى يذهبون هناك وهم غير

ادرين على فعل أي شئ ، و بهذا نكون قد ساعدنا علي تقليل أعداد الوفيات الناتجة عن التشخيص الخاطئ من الطبيب المعالج أو للأسباب السابق ذكرها في الرسالة البحثية.

جامعة المنصوره

كلية الحاسبات و المعلومات

قسم علوم الحاسب

نظام لإدارة السجلات الطبية بإستخدام تقنية البلوك تشين

مقدم من

أحمد مسعد يوسف

أحمد نجيب فويله

اسلام انور الحاكمي

أيمان عبد القادر ستو

سامية ايهاب الجندي

فريال اسماعيل ابراهيم

يارا السيد رمضان

احمد مدحت البسيوني

يوسف البوساطي

وليد ماهر حسن

أشراف

ا . د / سمير الموجي

د / سارة المتولي