HELWAN UNIVERSITY
Faculty of Computers and Artificial Intelligence
Information System Department

MobiCare

A graduation project dissertation by:

Mohamed Moataz Fathi	-	201900737
Omar Anas Abdelhakeem	-	201900502
Fares Hesham Zakaria	-	201900553
Gerges Milad Louis	-	201900241
Heba Adel Ahmed	-	201900929
Reem Hisham Mahmoud	-	201900318

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computers & Artificial Intelligence, at the Information System Department, the Faculty of Computers & Artificial Intelligence, Helwan University

Supervised by: **Dr. Doaa Saad**

June 2023

Acknowledgment

First and foremost, we would express our deepest gratitude and appreciation to our Advisor **Dr. Doaa Saad** for her support, outstanding guidance, and encouragement throughout our graduation project.

As well as Engineer. **Eng. Samar Samir** for his continuous support and help as he was always there for us providing more than the needed time and effort from him since the very beginning of our project.

We would like to thank our families, especially our parents. We hope they are proud of us in our last year of education, we hope that we will start giving back to the community very soon! Thank you for supporting us and providing us with the needed time, effort, encouragement, patience, and assistance over the years.

Finally, our faculty for providing us with the courses that guided us in the right direction in our lives and the help of all the professors that left a great impact on our lives,

Thank you.

Abstract

Main idea:

This system facilitates the connection between doctors and their patients and provides a medical history of the patient's prescription in the form of a pdf, also a professional history of doctors the patients have visited.

With the help of BlockChain, it helped in making the application even better.

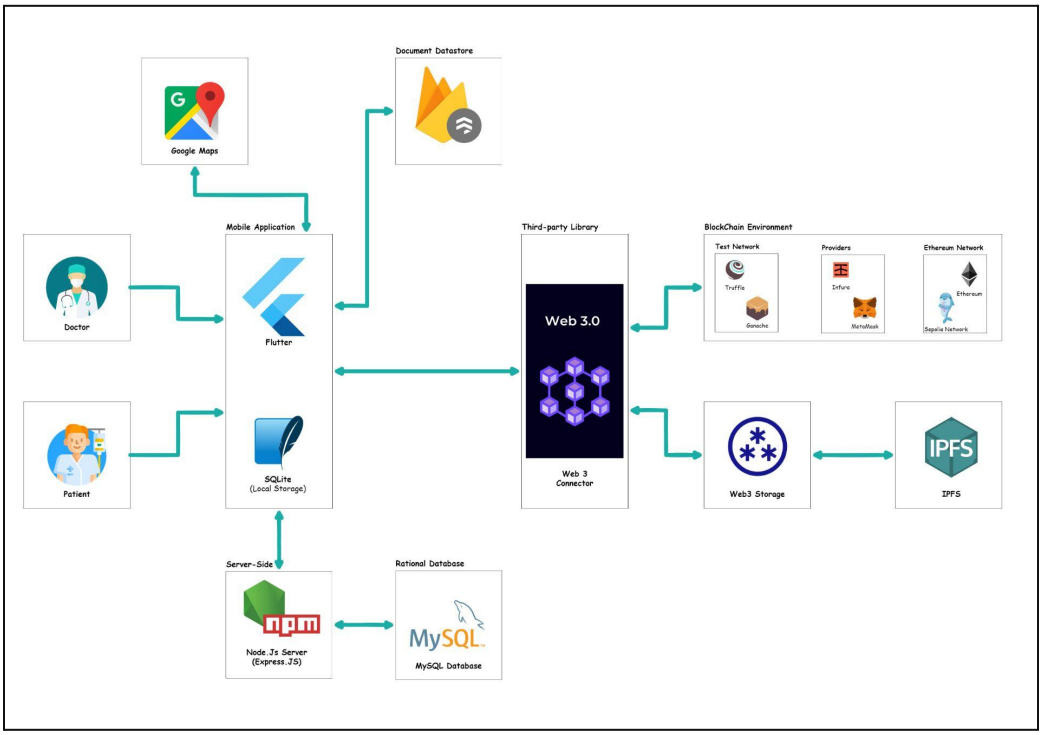
MobiCare Whitepaper 2.1 will outline the vision of the mobile application and the current issues in healthcare, as well as give a brief summary of the blockchain technology used and how the web application is utilizing it to address specific issues to make healthcare better for users.

Structure:

The program is built by connecting various tools and suits and a supportive platform, to form a coherent network that could maintain the implementation and architecture.

Technologies and processes:

- ### Technologies & Tools:
  - illustrating the technologies connection



1) Node.js, npm:

We use the node.js framework to build our application network to control many connections, as it contains specific packages and libraries essential for the web application like npm library. They can be set up and update packages through the cmd command.

Advantages:

- Used for setting up a blockchain environment.
- Helps in building a blockchain testing environment.
- Used in deploying and running Ethereum smart contracts.

2) Truffle:

Essential tool for building dApp /smart contracts. It is considered A testing environment and an Ethereum virtual machine (EVM).

Advantages:

- Support direct compiling smart contacts.
- Link smart contracts together.
- Support deploying smart contracts.
- Manage networks and packages.

3) MetaMask:

We used it as a private Ethereum network. It is a browser plugin and mobile application to connect Ethereum apps with the browser which allows making smart contract transactions and deploying them. It provided an electronic wallet with network IP and account to manage Ethereum smart contracts.

Advantages:

- It is open source with existing features.
- It provides built-in coin purchasing.
- It provides a local key storage.

4) Ganache:

We used it as a personal Ethereum blockchain-enabled us to develop dApps that are used across all the entire application. It provided us with a number of accounts allowing us to test and deploy the application and

dividing them among application different end users. It has a gas price and gas limit to control the mining process.

Advantages:

- Allows employment and running smart contracts transactions.
- Enable quick and easy testing for Ethereum smart contracts.

5) Ethereum:

In MobiCare application we used Ethereum to build its contract with the transactions to create a public decentralized blockchain DApp and create chains of structs including blocks with addresses. It performs a network with the other tools after being connected together, then being compiled and deployed.

Advantages:

- Natively supports smart contracts.
- It is. scalable, programmable, secure, distributed, and decentralized.
- Executes smart contracts and stores data for third-party applications.

6) Web3 storage:

Is considered a bridge that facilitates our connection with Ethereum.

Advantages:

- Web3 and truffle suit make a framework that serves dApps.
- Provides ownership and control to end-users about their data through encryption.
- Provides a trusted secured platform in which data is fully encrypted.
- Anybody can create an address and interact in the blockchain.

7) IPFs:

The InterPlanetary File System (IPFS) is a set of composable, peer-to-peer protocols for addressing, routing, and transferring content-addressed data in a decentralized file system. Many popular Web3 projects are built on IPFS.

Advantages:

- Enables us to exchange blocks between users.

- Creates a network of nodes hashed with a public key.
- Can store small scripts or big databases.

8) Infura:

Infura provides the tools and infrastructure that allow developers to easily take their blockchain application from testing to scaled deployment - with simple, reliable access to Ethereum and IPFS.

Advantages:

- It makes connections with peer-to-peer networks which can require long initialization times.
- It solves the problem of the cost of storing the entire Ethereum blockchain.

9) Sepolia network:

We used it to test the network and to deploy the contract.

Advantages:

- Simulate harsh network conditions.
- Enables fast transaction confirmation time.

10) MySQL:

We used it in storing and scaling this complicated data workload as it stores and provides access to data points.

Advantages:

- Support writing database functions using various languages.
- Support for a huge number of data types.
- It helps in aligning customized databases more closely with the way the development data is represented in the applications.

11) Google Maps:

We used google maps to show the location of the patient for the home visit request.

Advantages:

- Provides the layouts of the road

12) Flutter:

We used flutter to build our interactive mobile application, using MVVM architecture pattern.

Advantages:

- Flutter is a cross-platform development platform so it can work seamlessly across desktop, mobile, and other platforms.
- Same UI and business logic in all platforms.
- Fast development due to the “hot reload” feature.

13) Firebase:

We used firebase in making chatting between doctors and their patients come true, and handling home visit requests.

Advantages:

- Reliable and extensive database.
- Fast and save hosting.
- Firebase cloud messaging for cross platforms.

14) SQLite:

We used sqlite so doctors can add their tasks in the local storage with details and for patients to add and save their medical prescription.

Advantages:

- Storage and portability.
- Fast reading and writing operations.

Chapter 1: Introduction

In this chapter, we are going to discuss and go deeper into the overview of the project and know more about its scope and limitations and explain some terminologies we will find throughout the document.

1.1. Overview

MobiCare is a mobile application, It facilitates the methods of consultation and follow-up between doctors and their patients.

It includes all required data about the patient including prescriptions, medical reports, radiology images, diagnoses, treatment plans, allergies, laboratory results, etc..., the patient is allowed to check them whenever he wants and also he can give access to his doctor to get and upload his medical records.

The system also provides the patient with an alarm for medication appointments, where he is reminded of the date of each medication that he has entered.

The application also allows doctors to upload videos that help spread awareness for users throughout the application and also allows the patient to like and write comments on these videos.

1.2. Objectives.

- ➢ Facilitate communication between doctors and their patients.
- ➢ Storing all medical information belonging to the patient in a secure way.
- ➢ Make it easy for patients to reach out his all medical records.
- ➢ Raise awareness by letting doctors upload videos and let patients interact with it.
- ➢ Reminding patients of their medication appointments.

1.3. Purpose

MobiCare aims to change the traditional health system with an e-health system and improve the quality of health service by delivering easy access to health records used in decision-making related to the patient's care whenever and wherever he wants.

MobiCare aims to make communication between doctors and patients easier and faster with low effort and cost.

MobiCare aims to store patient medical records in a secure way.

MobiCare aims to raise awareness among people through videos uploaded by doctors, talking about the prevention of specific diseases, or talking about their specialization.

1.4. Scope

➢ Planning

Determine what we will do in this project to achieve its goal:

- **Brainstorming** meeting to collect and discuss the ideas we have.
- **Collecting data** about the project and the lack that made us need the App.
- Make a **Competitive Analysis**.
- Make the **Project Plan**.
- Deliver the project in an incremental way (**Sprints**) by **SCRUM Team**.

➢ Designing

Determine the diagrams to be carried out within the project:

- **System Architecture** Diagram.
- **Activity** Diagram.
- **Use-case** Diagram.
- **ERD** Diagram.

➢ Coding

The main functions to be coded in this site are:

- **Registration** for patients.
- **Login** as a patient/doctor.
- **View the profile** of the patient/doctor.
- **Update personal info** of patient/doctor.
- **Logout** as a patient/doctor.
- **Searching** for doctors.
- **Chat** between doctor and his patients.
- **Categorizing chats of** patients according to the doctor's decision.
- **Upload Videos** about medical topics by the doctors and let patients interact with them.
- **Check the doctor's information** by admin.
- **Reminding** patients of their medicines.

➢ Testing

Functional Testing:

- Unit Testing.
- Regression Testing.
- Integration Testing.

Non-functional Testing:

- Performance Testing.
- Stress Testing.
- Security testing.

➢ Documentation

At this stage we were trying to document everything our project passed by starting from validating the idea until it became a real touchable project, we organized our documentation into 8 main chapters as follows:

- **Introduction:**

  In this chapter, we gave a quick summary of the project highlights & limitations.

- **Planning & Analysis:**

  Here we talked about planning the project, this includes the feasibility study & the competitor analysis, what is missing in the available solution, and what requirements we will fulfill in our solution.

- **Software Design:**

  In this chapter, we included all the diagrams that we worked on as a visualization of the functions and scenarios we have, this includes architecture diagram, sequence diagram, activity diagram, and class diagram.

- **Implementation:**

  In this chapter, we will show you the architecture of the application and snapshots of the main functions. We will also mention the technologies we used.

- **Testing:**

  In this chapter, we tested the application in different ways.

- **Result & Discussion:**

  In this chapter, we discussed the expected results for the project and the actual results, what were the difficulties we faced, how we managed or tried to solve them, and what our final thoughts on the project were.

- **Future Work:**

  Here we will discuss what are our future plans for MobiCare and what are the ideas we are planning to add to the application.

- **Conclusion:**

  This chapter summarizes our whole document.

1.5. General Constraints

- Tasks division, which can be not fair enough.
- Indiscipline “Human factor” like being late in delivering tasks or attending meetings.
- Hesitation, especially when it is related to taking a serious step or learning a new technique.
- Time management.
- Learning new technologies may take time.
- Underestimating the objectives that may not lead to realistic or achievable functions.

Chapter 2: Planning and Analysis

In this chapter, we are going to discuss and go deeper into how we planned the project and show the steps and the instructions that we followed in planning the application.

2.1. Project planning

- ### Feasibility Study

  Once the initial due diligence has been completed, the real work begins. Components that are typically found in a feasibility study include the following:

  - #### Executive Summary

    MobiCare is a mobile application, It facilitates the methods of consultation and follow-up between doctors and their patients, also patients can keep their prescriptions to keep up with their meds.

  - #### Description of MobiCare

    MobiCare follows the freeware software standards. Since the system does not consist of a lot of multimedia data transfer, the bandwidth required for the operation of this application is low. No cost will be charged to the potential customers. Bug fixes and maintenance tasks will have an associated cost. MobiCare is a complete web-based application.

  - #### Technology Considerations

    - Backend Development: **Express JS, Web3 JS, Truffle, IPFS**
    - Frontend Development: **React JS**
    - Mobile Development: **Flutter, SQLite, Firebase**
    - Test Tools: **Postman, MetaMask, Ganache**

  - #### Existing Marketplace

    Our marketplace is the medical field. We have competitors, who produce medical services. Competitors like Veezeta and DoctorFly.

    You can find our competitive analysis here.

  - #### Marketing Strategy

    We will focus on our strengths and opportunities to let people know us and use our service. We will begin in a small region like Egyptian clinics and take a survey and feedback to improve our service and our platform to attract more people and help us to grow step by step.

    We can grow through social media platforms such as Facebook, Twitter, and YouTube.

    We will make deals with medications and makeup farms.

Staff

- Scrum Master.
- Product Owner.
- Developers.

Schedule and Timeline

Project prototype began in Aug 2022 and shall end in May 2022 after delivering APIs and running the Flutter application, then the first phase shall deliver in Egyptian clinics in 2023. And then the marketing will be studied again and decide the date of suitable next step.

Estimated Cost

A cost estimate is an approximation of the cost of a program, project, or operation. The cost estimate is the product of the cost estimating process. Several studies estimate the cost of purchasing and installing an electronic health record range from $20,000 to $50,000 per provider.

In our system, we have five components to estimate how much the system will cost:

Physical:

Potential physical costs may include desktop computers and tablets/laptops/mobile devices.

Software:

Potential software costs may include servers and clouds to store our data and deploy our application.

Implementation Assistance:

Potential implementation assistance costs include developers, IT contractors, attorneys, chart conversion, hardware/network installation, and workflow redesign support.

Ongoing Network Fees and Maintenance:

Potential ongoing costs include hardware and software license maintenance agreements and IT support fees.

2.2. Analysis and Limitations of the existing system

The majority of doctors currently work in hospitals and clinics (offline). There is no system for the doctor to work online. However, MobiCare allowed the doctor to better organize his work by allowing him to examine the patient remotely, stay in touch with him, and answer any questions. He can even write down the patient's prescription and save it for later.

2.3. Need for new system

Information saves lives. This is especially true in the medical field. Doctors use MobiCare to make data-driven decisions regarding various facets of patient care. For example, quick access to patient medical histories can bring previous treatments to light. Working offline is normal, but what if we can do the same work, but online?

The patient and the doctor will both benefit from time savings from this application.

Blockchain can modernize electronic health record exchanges by delivering a secure approach for medical data exchange of medical data in the field of healthcare productiveness, by safeguarding it through a dispersed peer-to-peer linkage. **Blockchain** will handle the security, reliability, immutability, and interoperability features.

2.4. Analysis of the new system

- ### User Requirements:
  - The user uses the app with him at any place and any time so our app should be available at any place and at any time.
  - The user can easily see his medical history.
  - The system secures all medical history and profiles for users to avoid data breaches.

- ### System Requirements:
  - Browser: The application works on any browser ex. ”Chrome, Microsoft Edge, Brave, ...etc”.
  - Mobile Application: The application works on Android and IOS.
  - Internet connection: This app uses a cloud server to store data, so you need to be connected to the internet to fetch and send data.

- ## Domain Requirements:
  - Multiple users must be able to use the application simultaneously without corrupting the cloud (whatever form it may be).
  - A server must be set up to host the cloud, and the server must be accessible by all the systems running the inventory tracking software.
  - The web application must verify all values before making the change in the cloud.
  - The application must update capabilities for future models.

- ## Functional Requirements:
  - ### Doctor Activities:
    - System shall allow **“Login”** with his (email and password).
    - System shall allow **“Profile editing”** by editing his info (first name, last name, email, password, address, specialty, clinic name, clinic location) and adding more info such as: (profile photo).
    - System shall be responsible for **“Generating a list of patients”** for the doctor, the list shall contain 15 patients, and if the doctor wants more he should pay for it.
    - System shall allow **“Increase”** the number of patients in his list and **“Pay”** for it, through an online payment service.
    - System shall allow **“Add new patient to the patients' list”** by patient username which is generated by the system.
    - System shall allow **“Searching”** for his patients by (name or phone number).
    - System shall allow **“Chatting”** with his patients only, he can send (text, attachment, and voice notes) to his patients.
    - System shall allow for receiving **“Home visit requests”** for patients, and he can approve or reject them
    - The system shall allow patients' messages to be **"Categorized"** by prioritizing patients' cases, and these chats will appear in the notifications bar and an alert widget when the doctor opens the app.
    - System shall allow the doctor to **“Write notes”** about his patients.
    - System shall allow for **“Writing medical records or uploading it”** to the cloud for his patients and it will be stored as a pdf file.
    - System shall allow for **“Storing medical records”** for his patients so that the patient can send them to another doctor.

- System shall allow the doctor to **"Prepare medical reports"** for his patients.
- System shall allow for **"Uploading videos"**, the video should be stored and displayed with its date, number of likes, and comments. The video will contain a title and description. Note that these videos will display to all users registered in the system.

Patient Activities:

- System shall allow **"Registration"** with all his basic data(first name, last name, email, password, address, gender, and date of birth) and an optional question asking him if he's suffering from a specific illness.
- System shall be responsible for **"Generating a unique code (username)"** for each patient and sending it by Gmail, To make it easy for doctors to add patients to their list.
- System shall allow **"Login"** with his (email and password).
- System shall allow **"Profile editing"** by editing his info (first name, last name, email, password, date of birth, address, profile photo) and adding more info (weight, height, and symptoms).
- System shall allow the patient to **"Check his illness history"** such as his previous consultations, and his prescriptions with the date and time they did them.
- System shall allow **"Searching"** for doctors by (name or specialist).
- System shall allow **"View doctor profile"** to patients and request to join his list.
- System shall allow **"Chatting"** with his doctors, he can send (text, attachment, and voice notes) to his doctor.
- System shall allow requesting a doctor for a **"Home visit"**.
- System shall allow **"Evaluation"** to doctors who perform a consultation.
- System shall allow the patient to **"Add medication schedule"** to stick with his medication, and the system is responsible for **"Reminding"** him of his medication.
- System shall allow patients to **"Like and comment"** on the doctors' videos.

Non-Functional requirements:

Performance:

- The system should be reliable.

- The system should have an active internet connection.

- **Security:**
  - Patient records should be stored using BlockChain.
  - System should filter the input from users in three levels (client side, server-side, and database).

- **Reliability:**
  - The system should be reliable and provide catching of exceptions so that unintended results do not occur such as system crashes, or data validation failures.

- **Usability:**
  - Once the user logs in, he is remembered until he logs out.
  - The system should have a user-friendly interface.

- **Availability:**
  - The system should be compatible with mobile devices and websites.

2.5. Advantages of the new system

- **Easy Access to Patient Data:**

  Doctors by using the application can see the patient's history of prescriptions. It is only a matter of a few clicks and all the requisite information about a patient, from various departments, can be available on the screen.

- **Follow-Up Online:**

  Doctors can examine patients making records with each other, take notes about patients and save it, and can upload medical videos. The doctor shall allow patients' messages to be "Categorized" by prioritizing patients' cases

- **Improved Patient Care:**

  Better and quicker healthcare decisions result from enhanced patient data access and job efficiency. Patients can ask doctors using chat, voice notes. They can also interact with doctor's posts by "Like and comment" to the doctors' videos.

- **BlockChain Technology:**

  In healthcare, it can include record interoperability, improved patient information access, and system monitoring spanning a device's entire life cycle in the blockchain substructure. Blockchain can be used along with other emerging technologies like the Internet of Things (IoT) and Cloud computing for better EHR solutions.

2.6. Risk management

- **Schedule Risk**

Risk	Probability	Impact	Priority
Wrong time estimation	low	high	high
Resources are not tracked correctly	low	medium	medium
Unexpected project growth	low	high	low

  - **Risk Avoidance and Mitigation**
    - Follow up the Gantt chart regularly and reduce the critical paths in the project network.
    - Looking after the major tasks -especially- that the other tasks depend on.
    - Updating the plan to adapt to any newly added requirements without affecting the already existing plan.
    - Reducing wasted time.

- **Technical Risk**

Risk	Probability	Impact	Priority
Changing requirements	high	medium	medium
Database isn’t efficient	low	high	high
Server Failure	low	high	high
Time/Space Complexity	medium	low	low

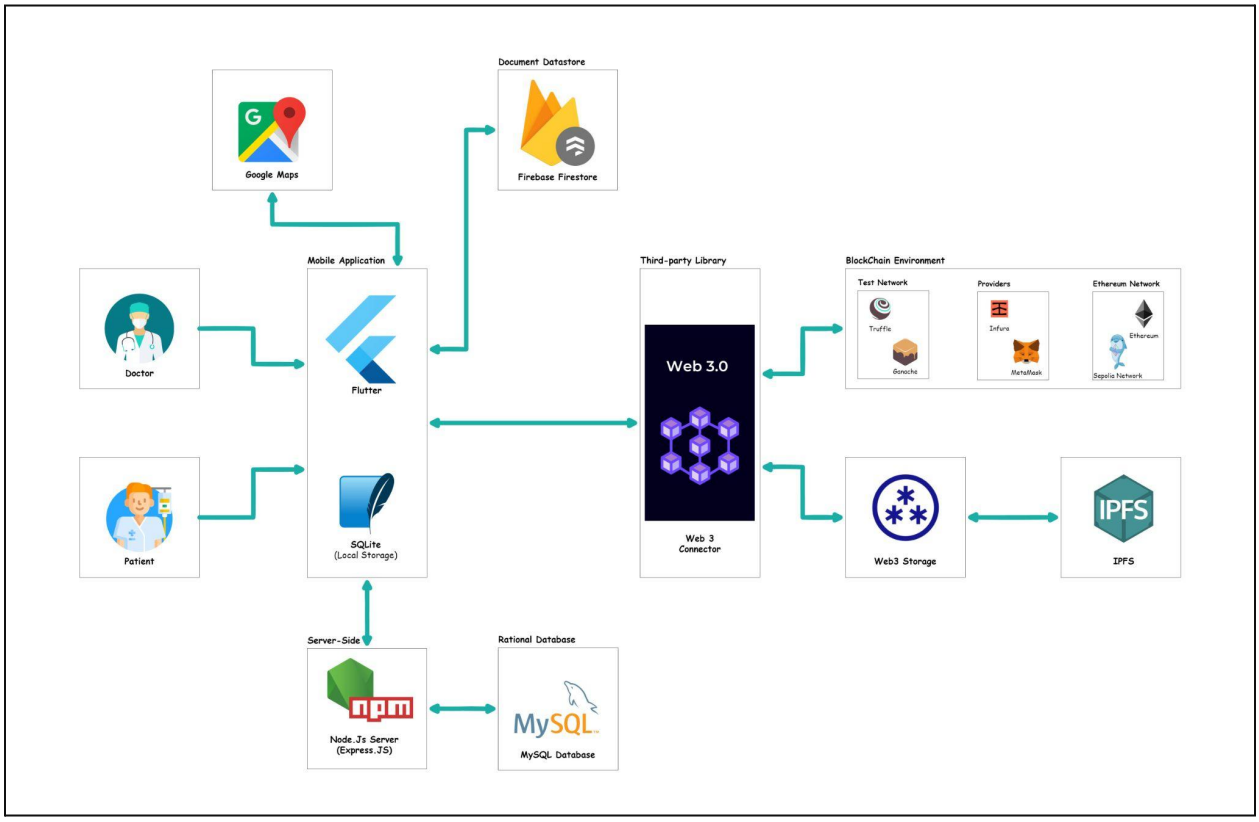
- **Risk Avoidance and Mitigation**
  - Not rushing the analysis phase.
  - Specifying the desired outcome.
  - Detailed test cases and regulations.
  - Analyzing the security threats.
  - Quick adaptation to any new requirements.

Chapter 3: Software Design

In this chapter, we are going to discuss and go deeper into how we design our system and present its diagrams and database.

MobiCare System Architecture

- ### Generic System Architecture

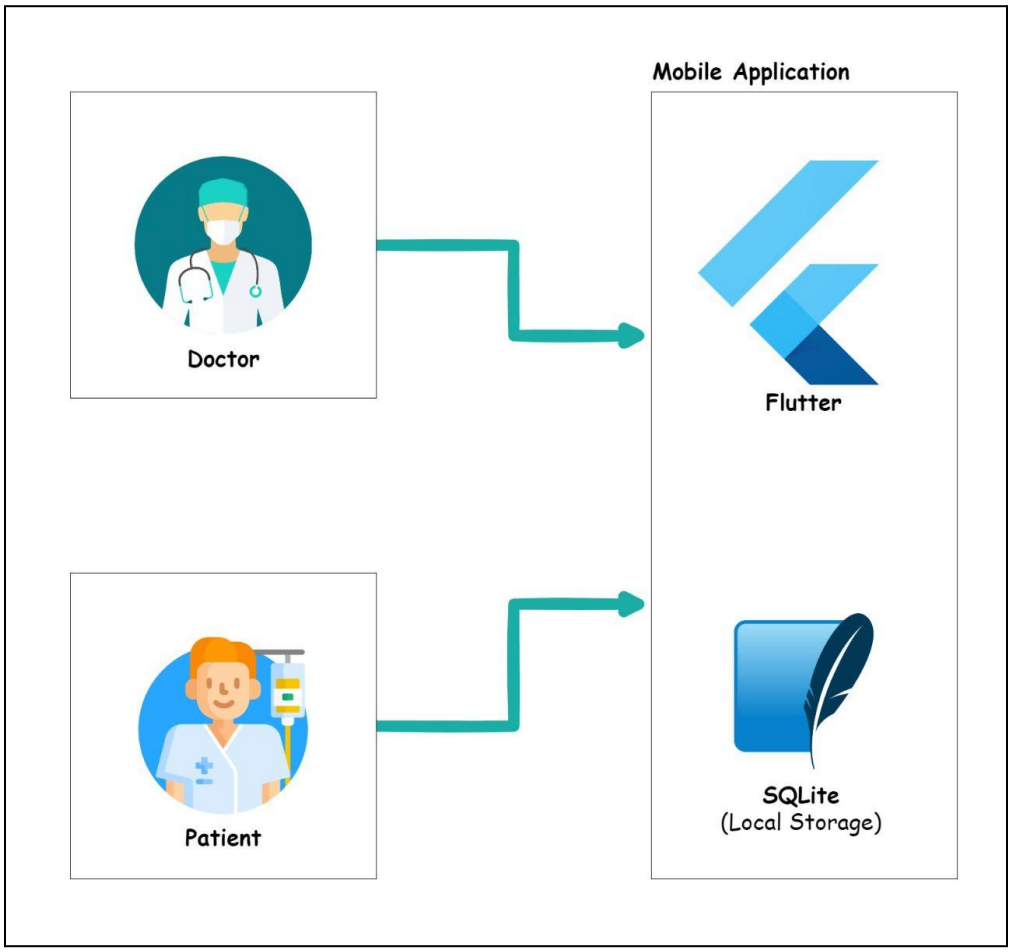


As we see, MobiCare has many components and uses many technologies to store its data.

So, we can explain MobiCare architecture in four main components/technologies:

1. Mobile Application.
2. Relational Database.
3. Document Data Store.
4. BlockChain.

- ## Mobile Architecture



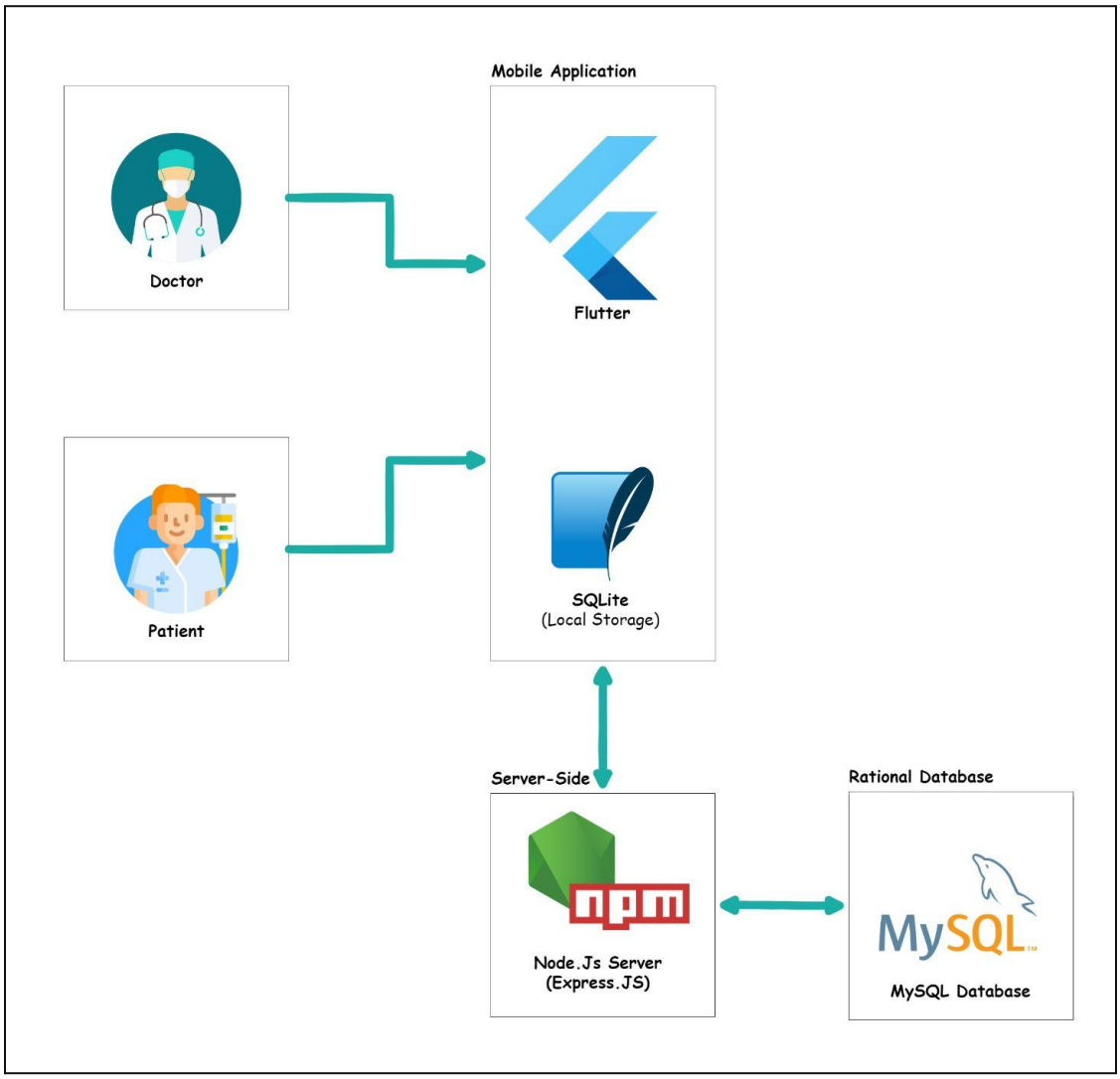
MVVM (Model-View-ViewModel)

MVVM is useful to move business logic from view to ViewModel and Model. ViewModel is the mediator between View and Model which carry all user events and return back the result.

The key benefits of using MVVM are

1. Business logic is separate from UI.
2. The view is independent from the ViewModel class and only reads the state from ViewModel
3. Code will be easy to maintain and update in terms of logic & UI
4. Easy to Write the test cases for the project

- ## Server-Side Architecture

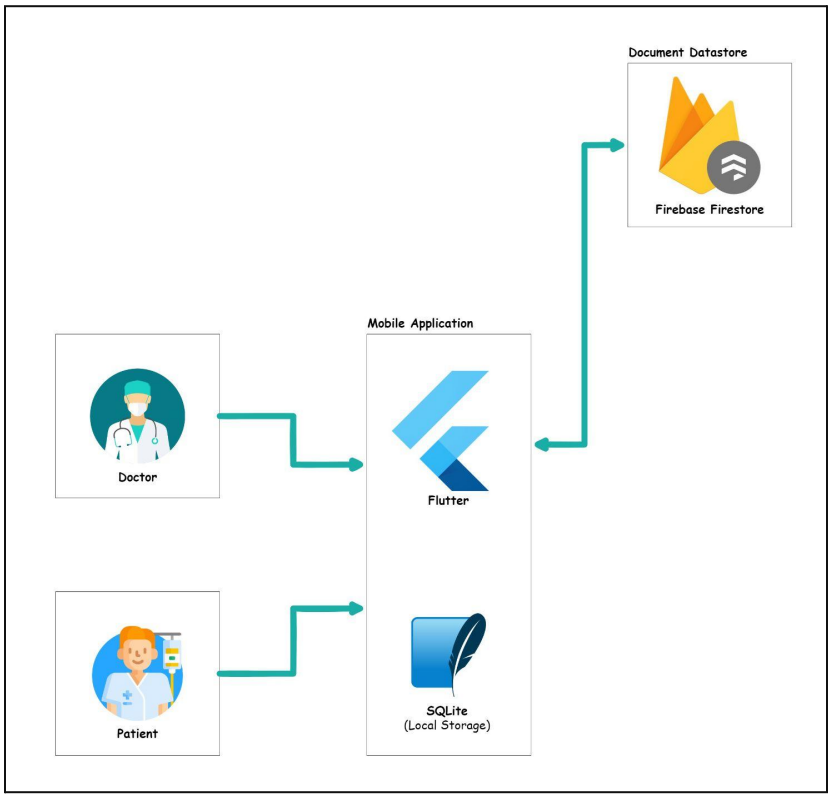


We worked with MVC architecture (Model, View, and Controller) by replacing the view part with REST APIs.

Also, the server-side provides us with an authentication system that verifies the permission of each role (patient, doctor, and admin).

- Model is the data part.
- View is REST APIs part.
- Controller is the request-response handler.

- **Document Data Store**



We used the **Firestore** feature in the **Firebase** - which is (BaaS) - as a **document data store and authentication system**.

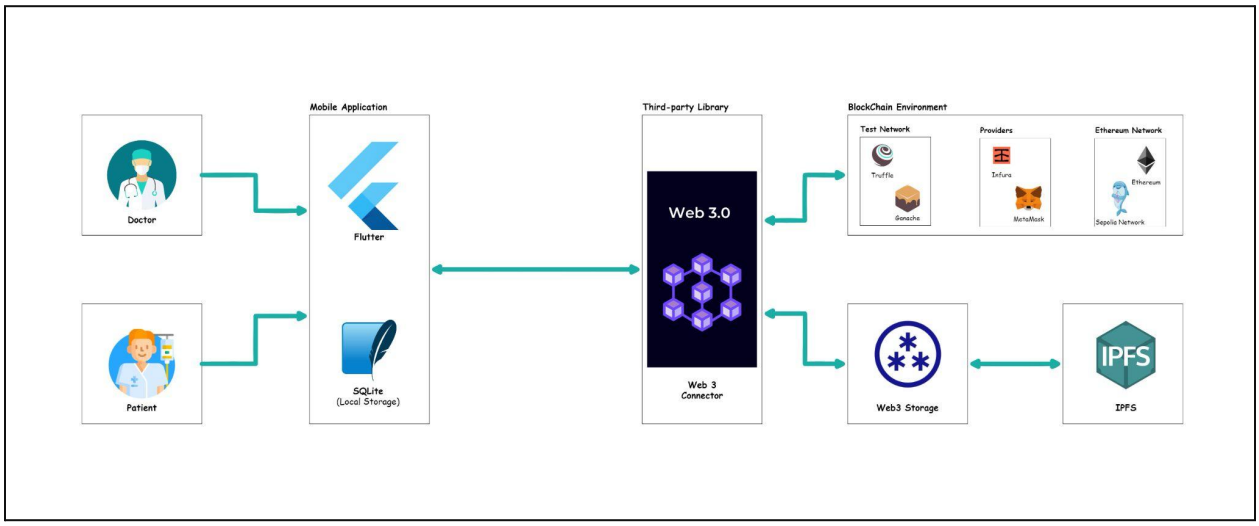
Firebase Firestore provides many features:

- application layers
- unidirectional data flow
- mutable and immutable state
- stream-based architecture

It facilitates dealing with data in several ways:

- Adding new features becomes easier because you can build upon the foundation that you already have.
- The codebase becomes easier to understand, and you're likely to spot some recurring patterns and conventions as you read the code.
- Components have clear responsibilities and don't do too many things. This happens by design if your architecture is highly composable.
- Entire classes of problems go away (more on this later).
- You can have different kinds of components, by defining separate application layers for different areas of concern (UI, logic, services)

- ## BlockChain Network



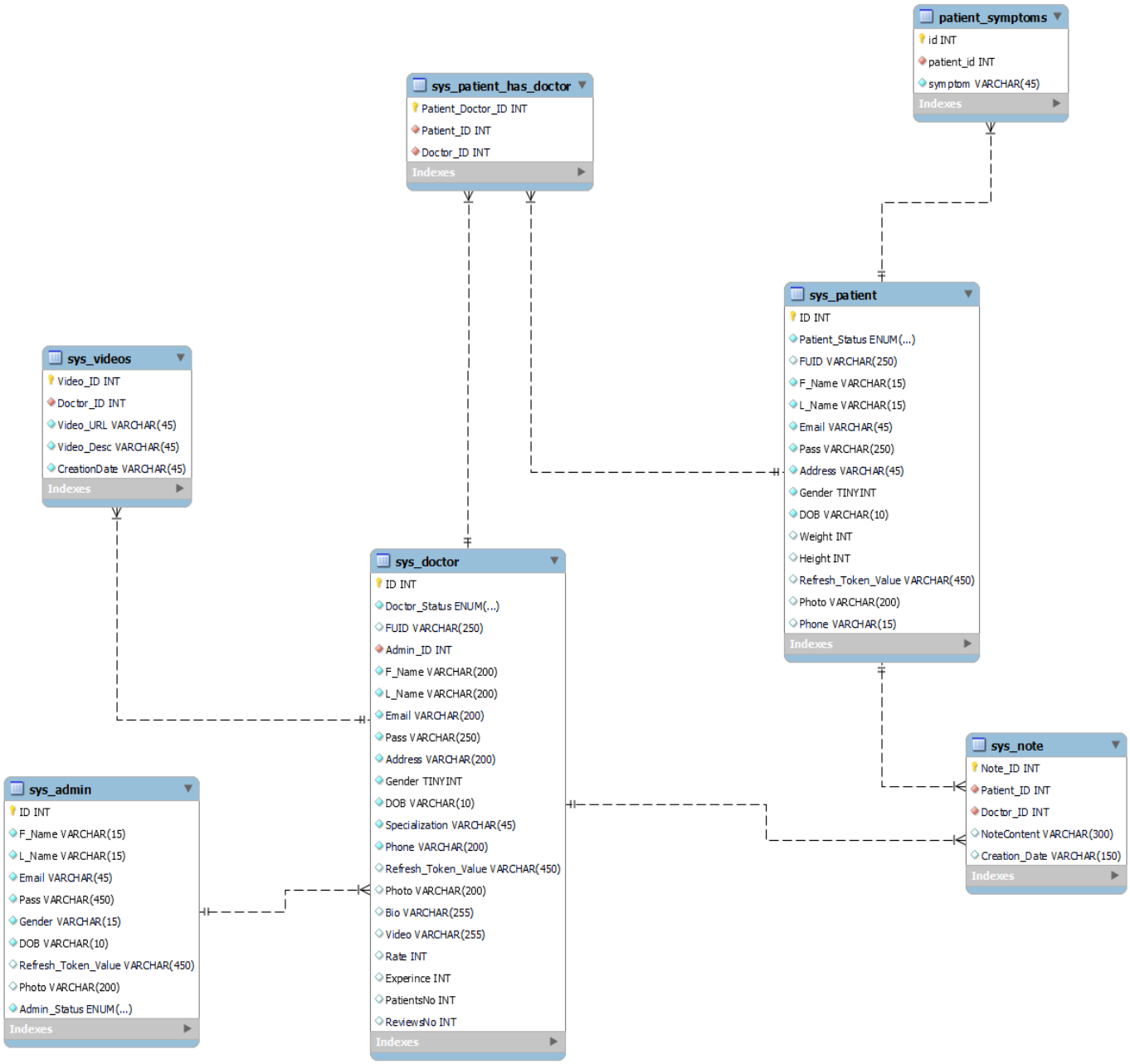
To connect to the blockchain network we used:

- **In the local environment:**
  - **Truffle:** Building the smart contracts. compile and deploy our smart contracts. Dealing with our smart contracts.
  - **Ganache:** Local blockchain network. Deploying our smart contract. Providing test accounts.

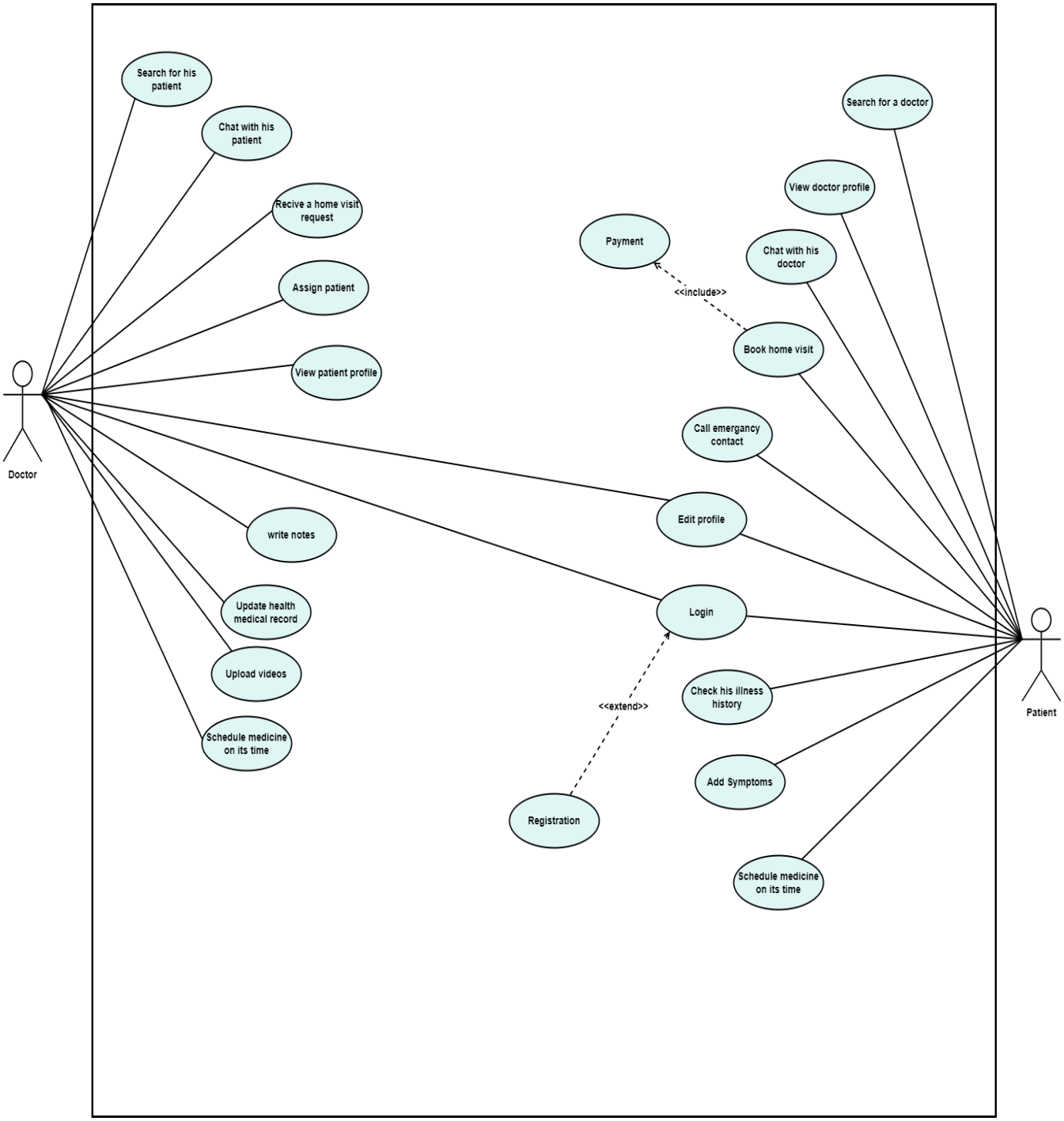
- **Providers to deploy on blockchain network:**
  - **Infura**: Providing test Ethereum to deal with the blockchain network. Providing rpcURL and wsURL to our blockchain project.
  - **MetamMask:** Private Wallet to deal with blockchain activities. Connected to Flutter to deal with the blockchain network.

- **Blockchain Network**
  - **Sepolia Test Network:** Deployed our smart contract on. Our blockchain network deals with our smart contract.
  - **Ethereum:** The main network and the main cryptocurrency which our smart contract deals with. Any transaction created in the Sepolia test network must pay gas fees with Ethereum cryptocurrency.
  - **IPFS (**Interplanetary File System**):** which we use to store our sensitive data - medical records - and respond with **cid (**content identifier**)**, and then we store this cid in the blockchain network with the patient's wallet address to access it when he wants.
  - **Web3.Storage:** like Infura, it helps us to deal with IPFS to store and access our files in the IPFS Distributed network. It's a bridge between us and IPFS.

ERD Diagram


patient_symptoms
id INT
patient_id INT
symptom VARCHAR(45)
Indexes
sys_patient_has_doctor
Patient_Doctor_ID INT
Patient_ID INT
Doctor_ID INT
Indexes
sys_patient
ID INT
Patient_Status ENUM(...)
FUID VARCHAR(250)
F_Name VARCHAR(15)
L_Name VARCHAR(15)
Email VARCHAR(45)
Pass VARCHAR(250)
Address VARCHAR(45)
Gender TINYINT
DOB VARCHAR(10)
Weight INT
Height INT
Refresh_Token_Value VARCHAR(450)
Photo VARCHAR(200)
Phone VARCHAR(15)
Indexes
sys_videos
Video_ID INT
Doctor_ID INT
Video_URL VARCHAR(45)
Video_Desc VARCHAR(45)
CreationDate VARCHAR(45)
Indexes
sys_doctor
ID INT
Doctor_Status ENUM(...)
FUID VARCHAR(250)
Admin_ID INT
F_Name VARCHAR(200)
L_Name VARCHAR(200)
Email VARCHAR(200)
Pass VARCHAR(250)
Address VARCHAR(200)
Gender TINYINT
DOB VARCHAR(10)
Specialization VARCHAR(45)
Phone VARCHAR(200)
Refresh_Token_Value VARCHAR(450)
Photo VARCHAR(200)
Bio VARCHAR(255)
Video VARCHAR(255)
Rate INT
Experince INT
PatientsNo INT
ReviewsNo INT
Indexes
sys_note
Note_ID INT
Patient_ID INT
Doctor_ID INT
NoteContent VARCHAR(300)
Creation_Date VARCHAR(150)
Indexes
sys_admin
ID INT
F_Name VARCHAR(15)
L_Name VARCHAR(15)
Email VARCHAR(45)
Pass VARCHAR(450)
Gender VARCHAR(15)
DOB VARCHAR(10)
Refresh_Token_Value VARCHAR(450)
Photo VARCHAR(200)
Admin_Status ENUM(...)
Indexes

Use Case Diagram



1. **Name** : Registration.
   Initiator : Patient.
   PreConditions : None , (only open application).
   PostConditions : Our users' information will be saved in the system.
   Main success scenario :
   1) Open our application.
   2) Enter your (First name , Last name , Email , Password , Confirm password , Address , Gender , Date of birth).
   3) Press the register button.

   Goal : Your data now is saved in our database and your account has been created.

2. **Name** : Login.
   Initiator : Doctor , Patient.
   PreConditions : Register in our app.
   PostConditions : Go to home screen.
   Main success scenario :
   1) Open our application.
   2) Go to the login screen.
   3) Enter your (Email , and Password).
   4) Press the login button.

   Goal : Login to our application and Enjoy the features of the application.

3. **Name** : Edit Profile.
   Initiator : Patient , Doctor.
   PreConditions : Log in our application.
   PostConditions : your information will be edited and saved in the system.
   Main success scenario :
   1) Log in our Application.
   2) Go to Profile screen.
   3) Press edit profile button.
   4) Just do not late it empty , Edit or set as it all the following your (First name , Last name , Email , Password , Date of birth , Address , Gender , Profile photo) and add more information like your weight and your height in case you are a patient , In case you are a doctor Edit or set as it all the following your (First name , Last name , Email , Password ,Address , Gender , Profile photo , Clinic name , Clinic location , Speciality).

5) Confirm your password.
6) Press edit button.

Goal : Your data will be updated and saved in our system.

4. **Name** : Search for his patients.
   Initiator : Doctor.
   PreConditions : Login.
   PostConditions : List of patients will appear.
   Main success scenario :
   1) Log in our application.
   2) Press on the search icon.
   3) Enter the name or phone number of the patient.

   Goal : List of patients will appear with the same name or in case you enter the phone number your patient will appear.

5. **Name** : Search for doctor.
   Initiator : Patient.
   PreConditions : Login.
   PostConditions : List of doctors will appear.
   Main success scenario :
   1) Log in to our application.
   2) Press the search icon.
   3) Enter doctor name or doctor specialist.

   Goal : List of doctors will appear with the same name or the same specialist.

6. **Name** : Chat with his patient.
   Initiator : Doctor.
   PreConditions : Login , Patient have visited doctor before , and patient should be in the doctor's patients list.
   PostConditions : This Chat will be saved until you delete it.
   Main success scenario :
   1) Log in our application.
   2) Go to the chat screen.
   3) Select your patient.
   4) Type what you need , Send attachment and Send voice notes .

Goal : Have a chat with your patient to monitor the patient's illness.

7. **Name** : Chat with doctor.
   Initiator : Patient.
   PreConditions : Login , You have visited this doctor before , and the patient should be in the doctor's patients list.
   PostConditions : This Chat will be saved until you delete it.
   Main success scenario :
   1) Log in our application.
   2) Go to the chat screen.
   3) Select a doctor you need to chat with.
   4) Type what you need , Send attachment and Send voice notes .

   Goal : Have a chat with the specific doctor.

8. **Name** : View doctor profile.
   Initiator : Patient.
   PreConditions : Login.
   PostConditions : Profile of doctor will appear.
   Main success scenario :
   1) Log in to our application.
   2) Press the search icon.
   3) Enter doctor name or doctor specialist.
   4) List of doctors will appear with the same name or the same specialist.
   5) Select the doctor that you want.

   Goal : Profile of doctor will appear.

9. **Name** : View patient profile.
   Initiator : Doctor.
   PreConditions : Login.
   PostConditions : Profile of patient will appear.
   Main success scenario :
   1) Log in to our application.
   2) Press the search icon.
   3) Enter the patient name.

4) List of patients will appear with the same name.
5) Select the patient that you want.

Goal : Profile of patient will appear.

10. **Name** : Home visit.
Initiator : Patient.
PreConditions : Login.
PostConditions : Doctor will receive a home visit request from a patient.
Main success scenario :
1) Log in our application.
2) Select Doctor.
3) Send a request for a home visit.
4) Wait for the doctor to accept or reject your request.
5) If the doctor accepts the request, select a way for payment (Online or Offline).
6) If online enter your (Card number , Expiry date, CVV , Name).
7) Press the add card button.

Goal : Have a Doctor for a Home visit.

11. **Name** : Receive home visit requests.
Initiator : Doctor.
PreConditions : Login , Some Patient sends a home visit request.
PostConditions : Accept or reject home visit request.
Main success scenario :
1) Log in our application.
2) Some Patient sends a home visit request for you.
3) press accept home visit request.

Goal : Visit a patient in his home.

12. **Name :** Assign patient to doctor
Initiator : Doctor.
PreConditions : Login.
PostConditions : Patient added to doctors' patient list.
Main success scenario :
1) Log in our application.

2) Go to drawer.
3) Select add patient to patient list.
4) Enter patient Id.
5) Press done.

Goal : Patient added to doctors' patient list.

13. **Name** : Payment.

Initiator : Patient.

PreConditions : Login , Patient request doctor for a home visit or ask for a medical consultation.

PostConditions : Payment operation will be done.

Main success scenario :

1) Log in our application.
2) Select online payment after asking for a medical consultation or request doctor for a home visit.
3) Go to the payment screen.
4) Enter your(Card number , Expiry date , CVV , Name).
5) Press the add card button.

Goal : Patient Pays online by his card

14. **Name** : Write notes.

Initiator : Doctor.

PreConditions : Login , Some patient has visited you or you visited him.

PostConditions : Your notes will be saved in our system.

Main success scenario :

1) Log in our application.
2) Select your current patient.
3) press on Write a note.
4) write it.
5) press on the save button.

Goal : All the patient's notes will be Stored in the system and the patient can't see it.

15. **Name** : Add symptoms.

Initiator :Patient.

PreConditions : Login.

PostConditions : Your symptom will be saved in our system.

Main success scenario :

1) Log in our application.
2) Open Drawer.
3) Open profile.
4) Press on the add symptom button.
5) Type name of your symptom.
6) Press ok.

Goal : All the patient's symptoms will be Stored in the system.

16. **Name** : Upload health medical record.

Initiator : Doctor.

PreConditions : Login , Some patient has visited you or you visited him.

PostConditions : Medical records will be saved in a decentralized network.

Main success scenario :

1) Log in our application.
2) Go to the patients list.
3) Select a specific patient.
4) Press on the add prescription button.
5) Connect metamask wallet.
6) Select a file from your device.
7) Press on the upload button.

Goal : All the patient's prescriptions will be Stored in IPFS as a bytes and the patient could download it at any time.

17. **Name** : Check his illness history.

Initiator : Patient.

PreConditions : Login , Patient save his prescriptions and the time of getting them or have a previous medical consultation.

PostConditions : Patient checks his illness history.

Main success scenario :

1) Log in our application.
2) Go to illness history from the drawer.
3) Press on the connect wallet button.
4) See all the previous medical records by its time.

Goal : Patient sees all his illness history that is saved in our system.

18. **Name** : Upload video.
    Initiator : Doctor.
    PreConditions : Login.
    PostConditions : Your video will be stored in our system and all users can see it.
    Main success scenario :
    1) Log in our application.
    2) Go to the home screen.
    3) Go to upload a new video.
    4) Enter the data of your video like (Title of your video, description of your video).
    5) Upload the video that you want.
    6) Press the publish button.

    Goal : All the video will be shown in the videos screen for all users with its number of likes and comments.

19. **Name** : Add medication schedule
    Initiator : Patient.
    PreConditions : Login.
    PostConditions : Our application will remind you of the time of your medicine.
    Main success scenario :
    1) Log in our application.
    2) Go to the medicine alarm screen from the bottom navigation bar.
    3) Enter your medicine's name.
    4) Enter the time of your alarm.
    5) Enter how many times of week or month or day you must take it.
    6) Press on the done button.

    Goal : System will remind patients of the time of medicine.

20. **Name** : Add tasks schedule for doctor
    Initiator : Doctor.
    PreConditions : Login.
    PostConditions : Our application will remind you of the time of your task.
    Main success scenario :
    1) Log in our application.
    2) Go to the task alarm screen from the bottom navigation bar.
    3) Enter your task's name.

4) Enter the time of your alarm.
5) Enter the description of this task.
6) Press on the done button.

Goal : System will remind doctors of the time of task.

21. **Name** : Calling emergency contact.

Initiator : Patient.

PreConditions : Login.

PostConditions : Call the number you have been selected.

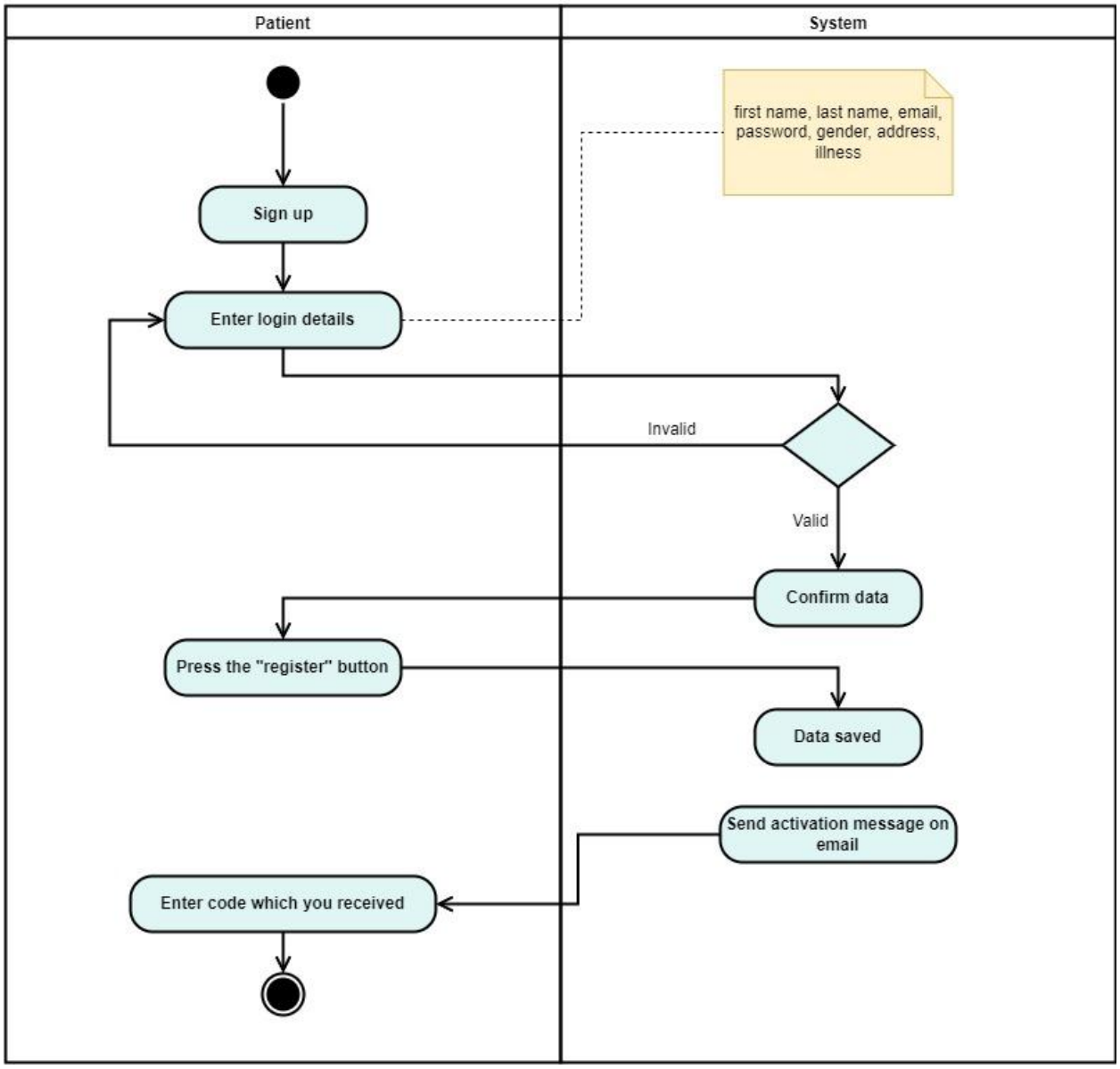
Main success scenario :

1) Log in our application.
2) Go to emergency contact.
3) Select number.
4) Press on the phone icon.
5) call it.

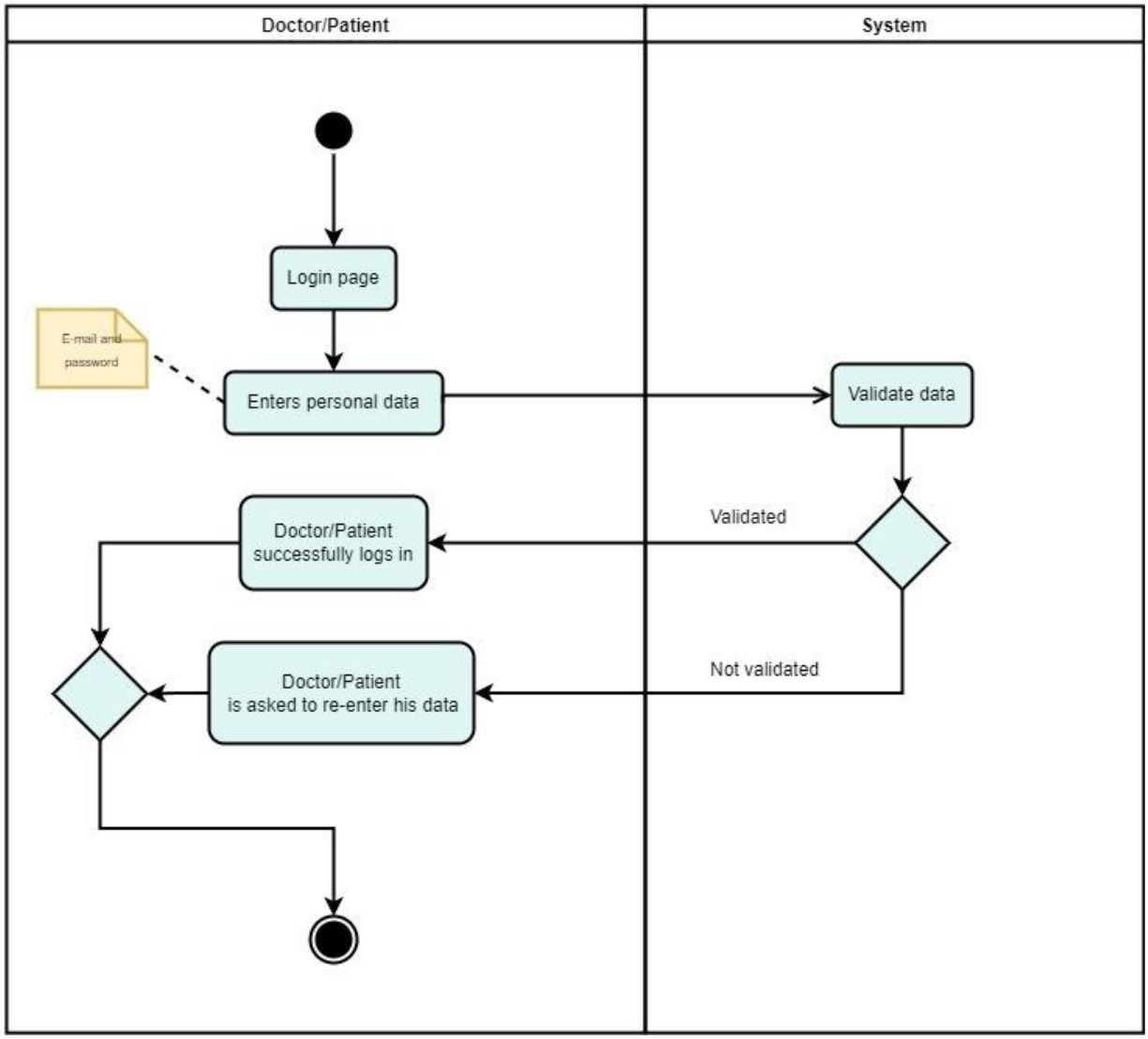
Goal : Patient can see all the emergency contacts in our application and can call any number of them.

Activity Diagram

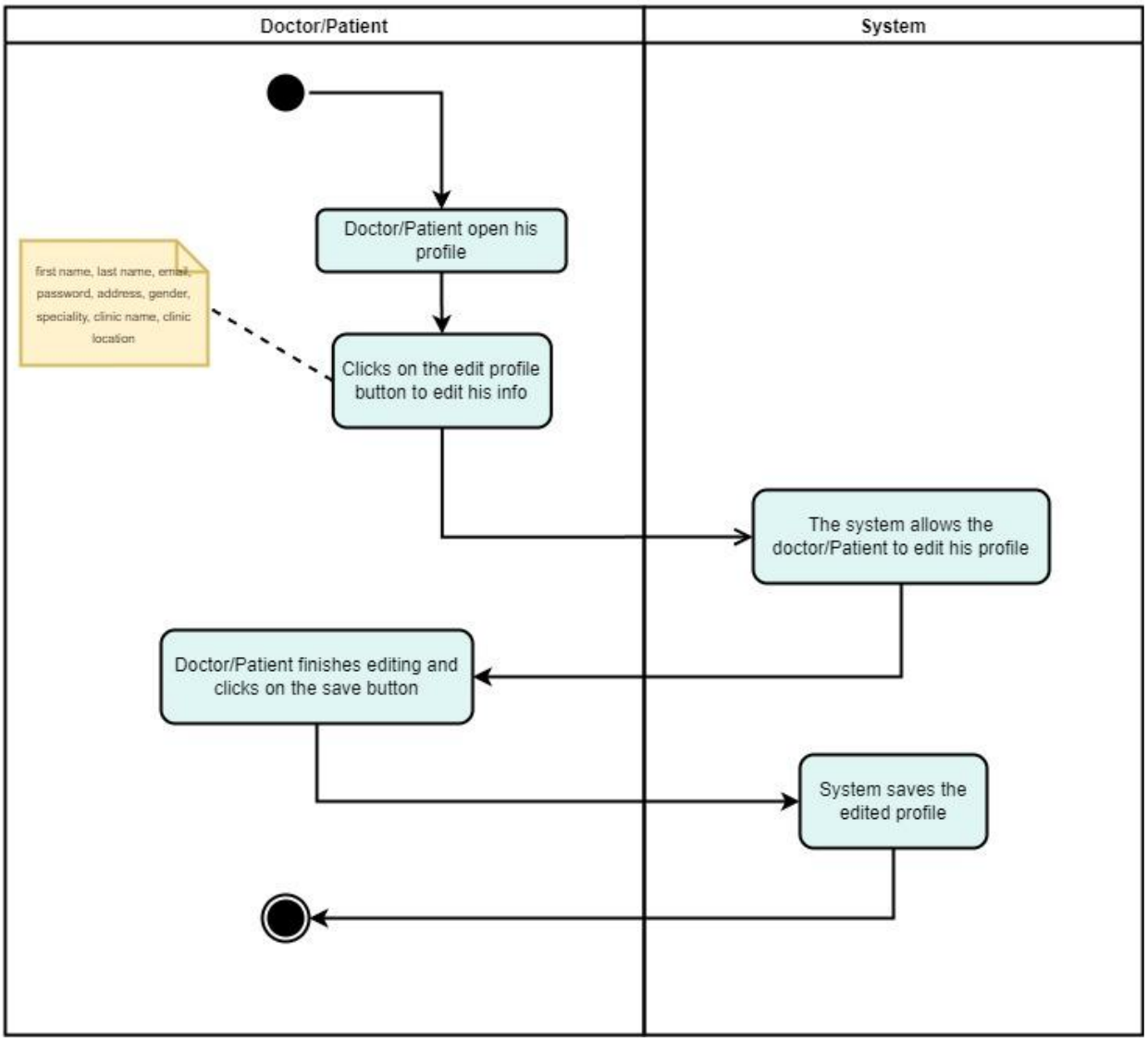
Registration :



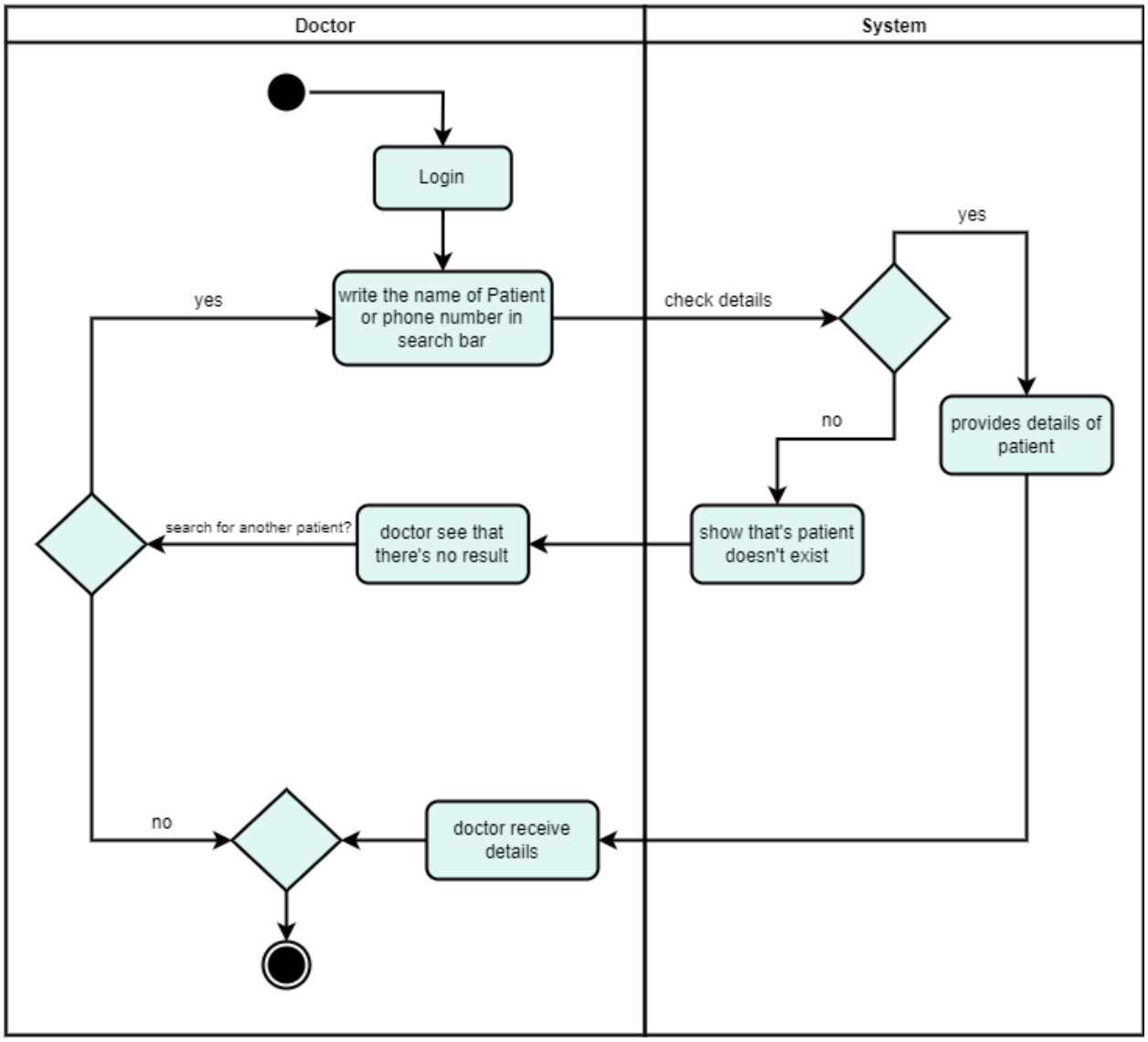
Doctor/Patient login :



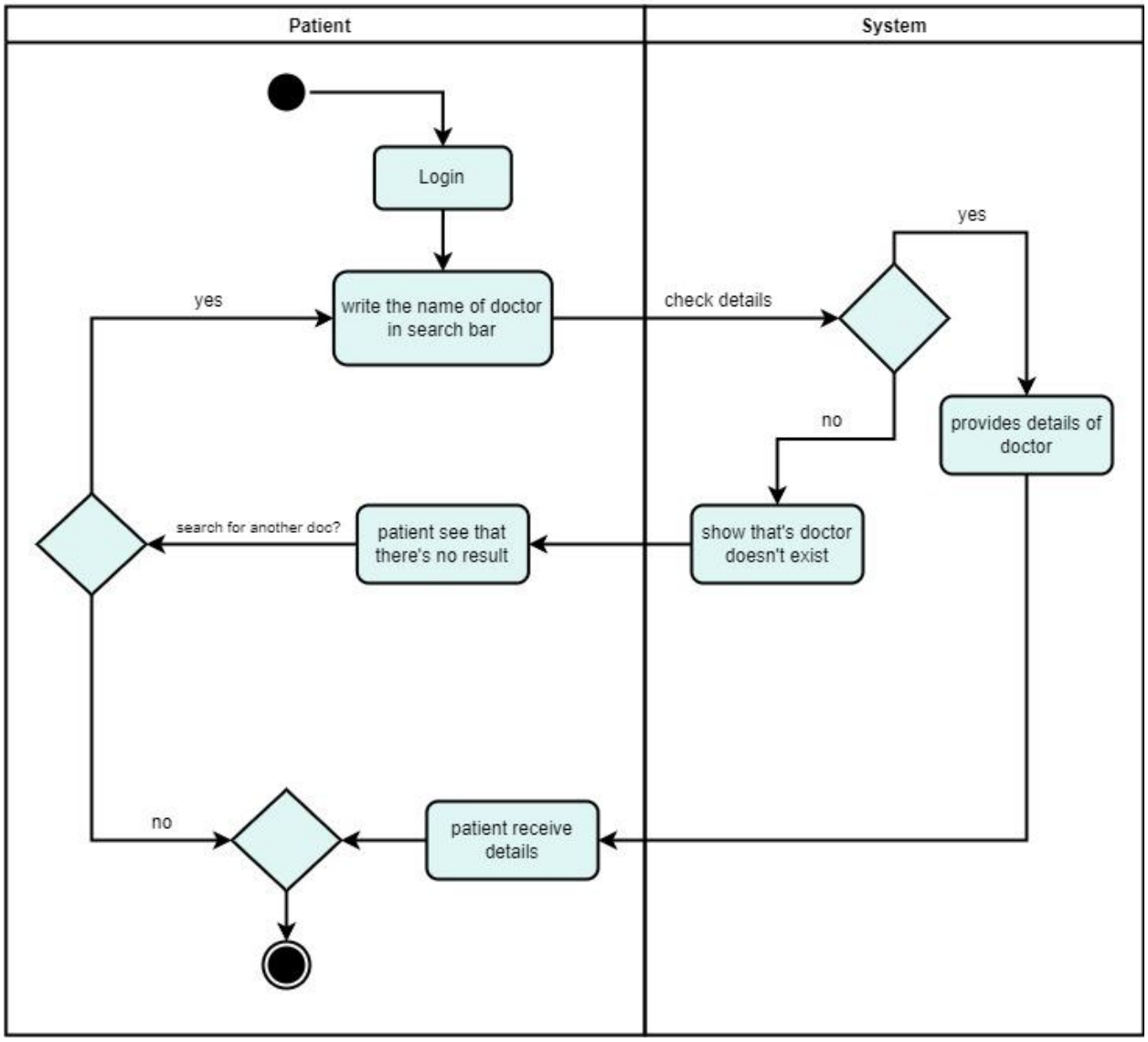
Edit doctor/patient profile :



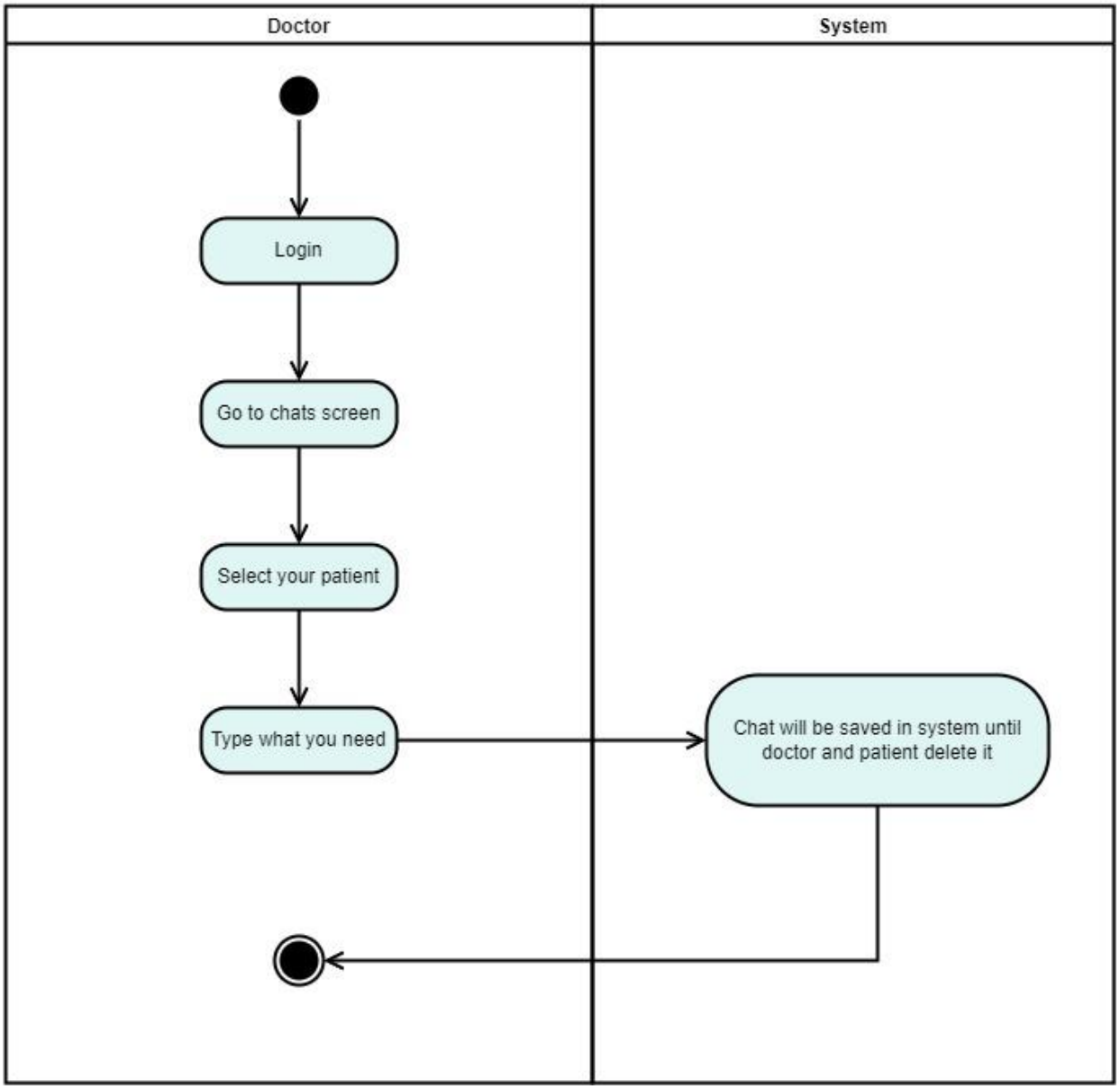
Doctor search about patient :



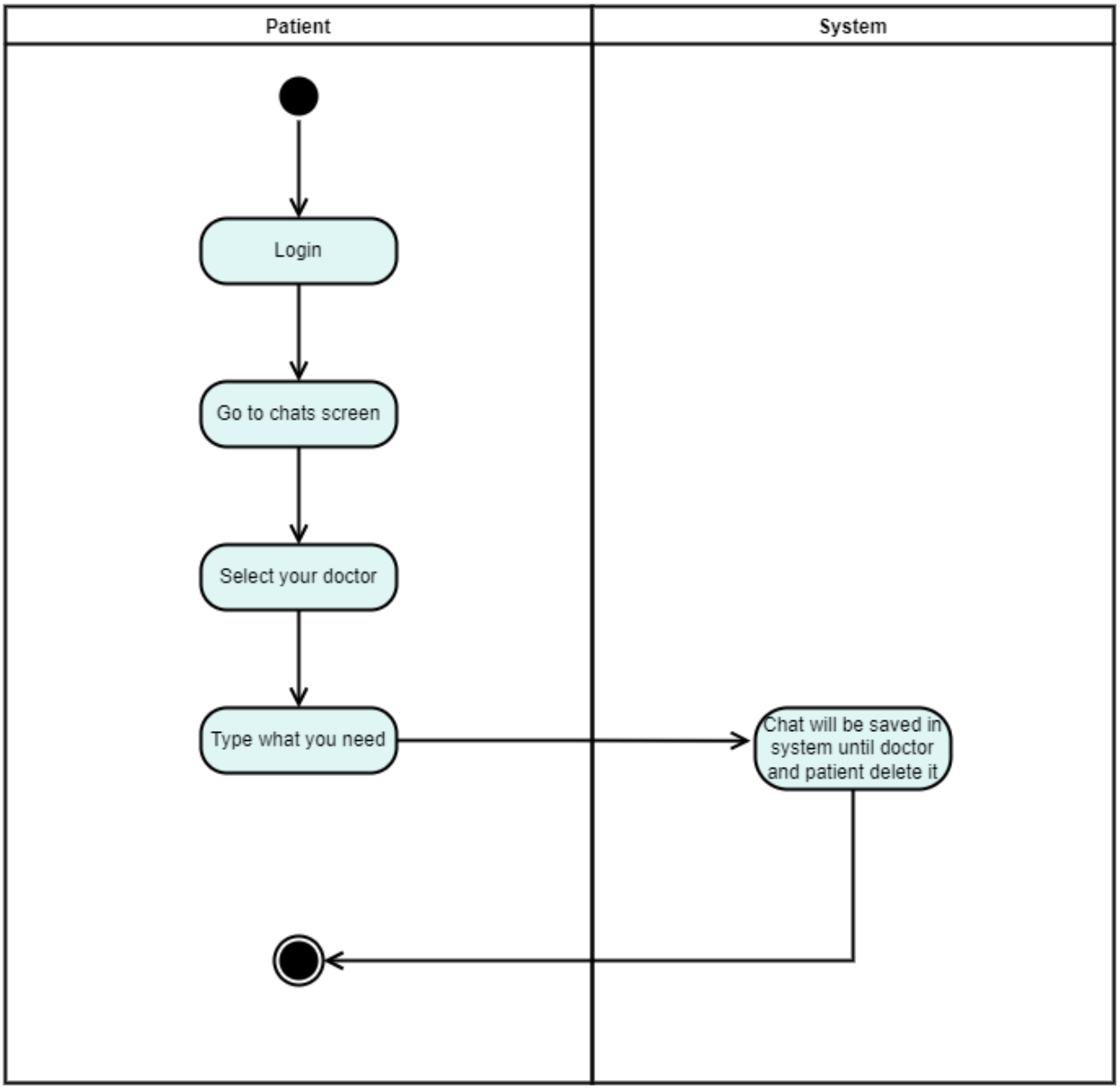
Patient search about doctor :



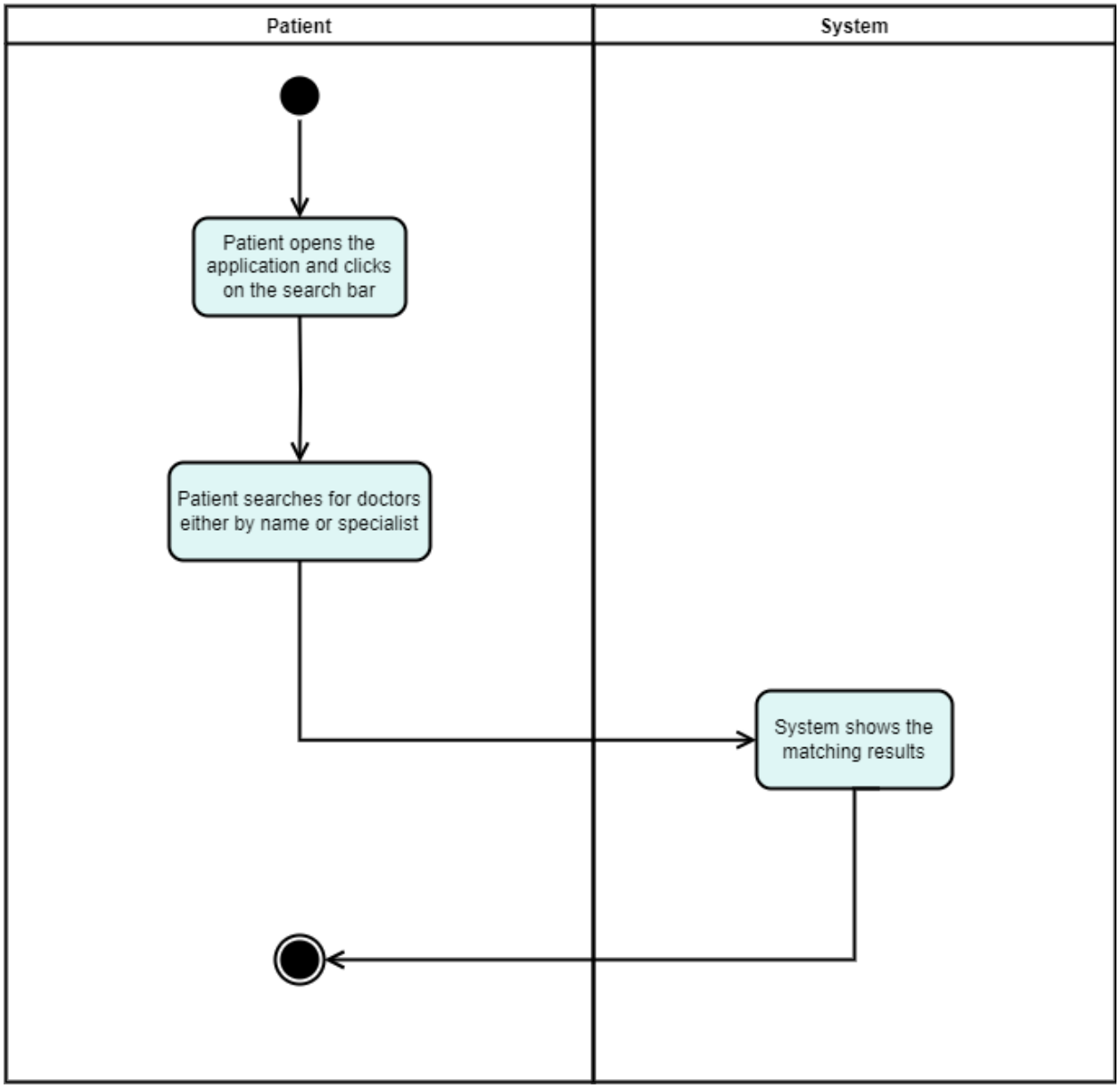
Chat from doctor to patient :



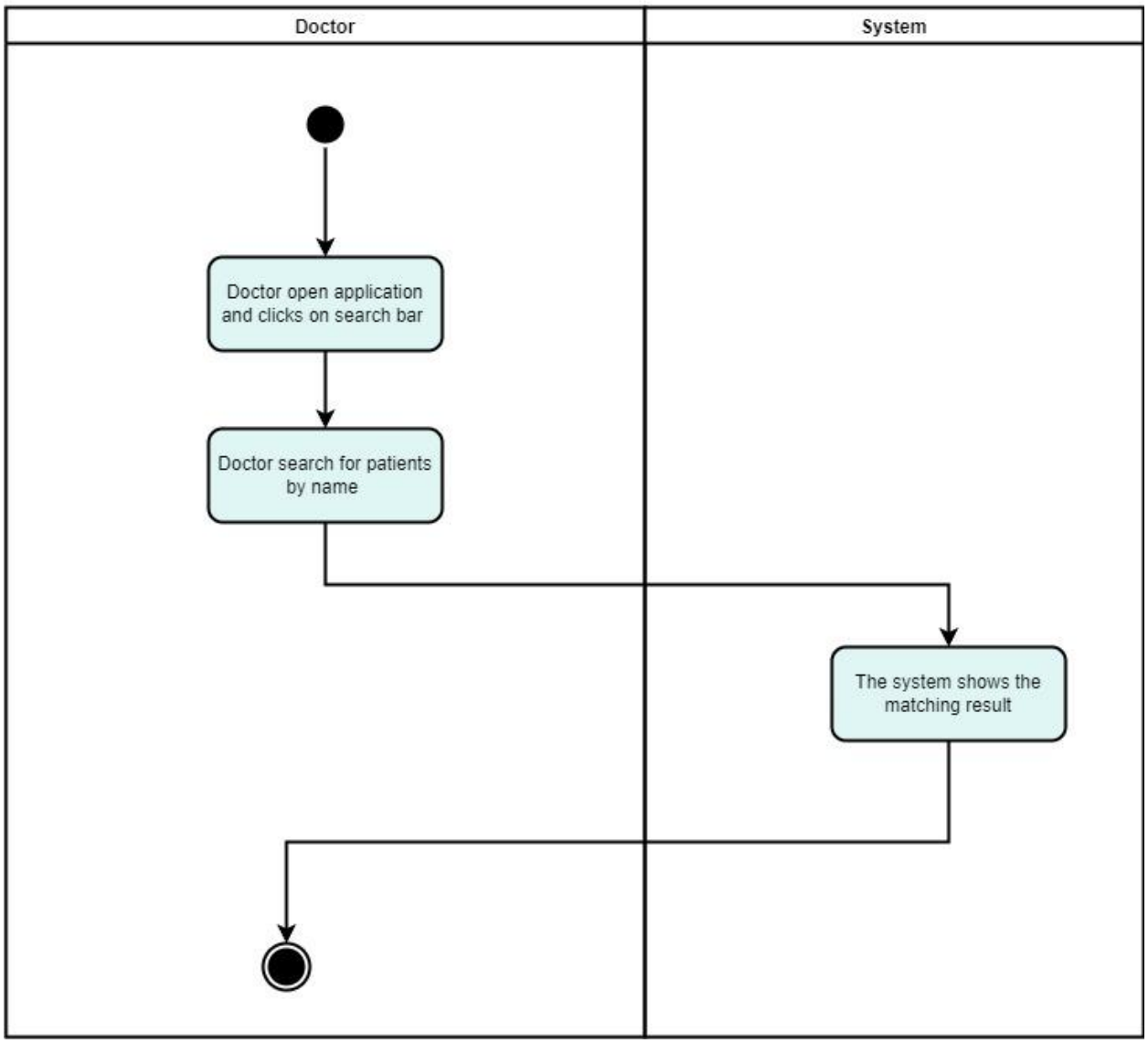
Patient chat with doctor :



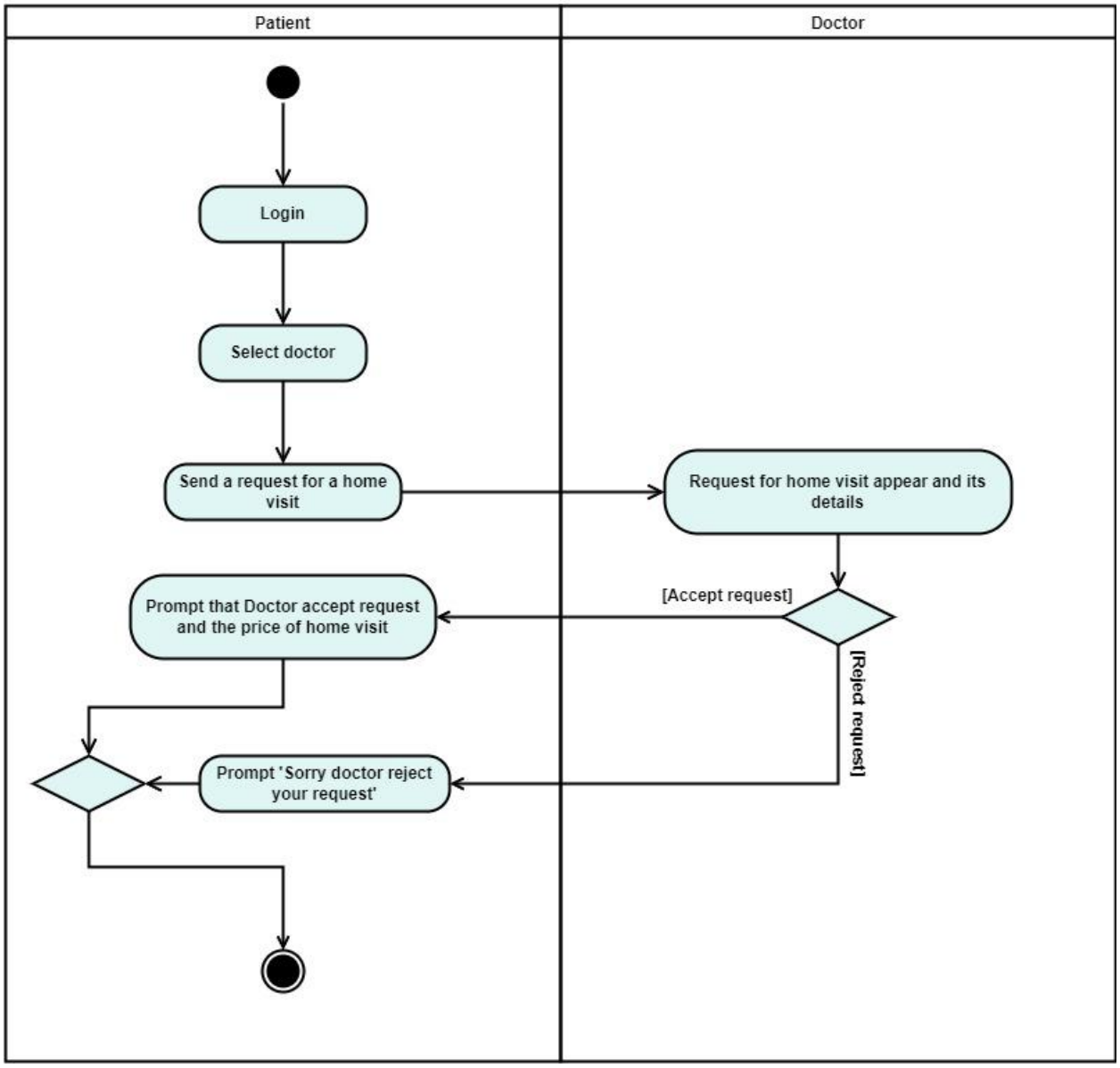
View Doctor Profile



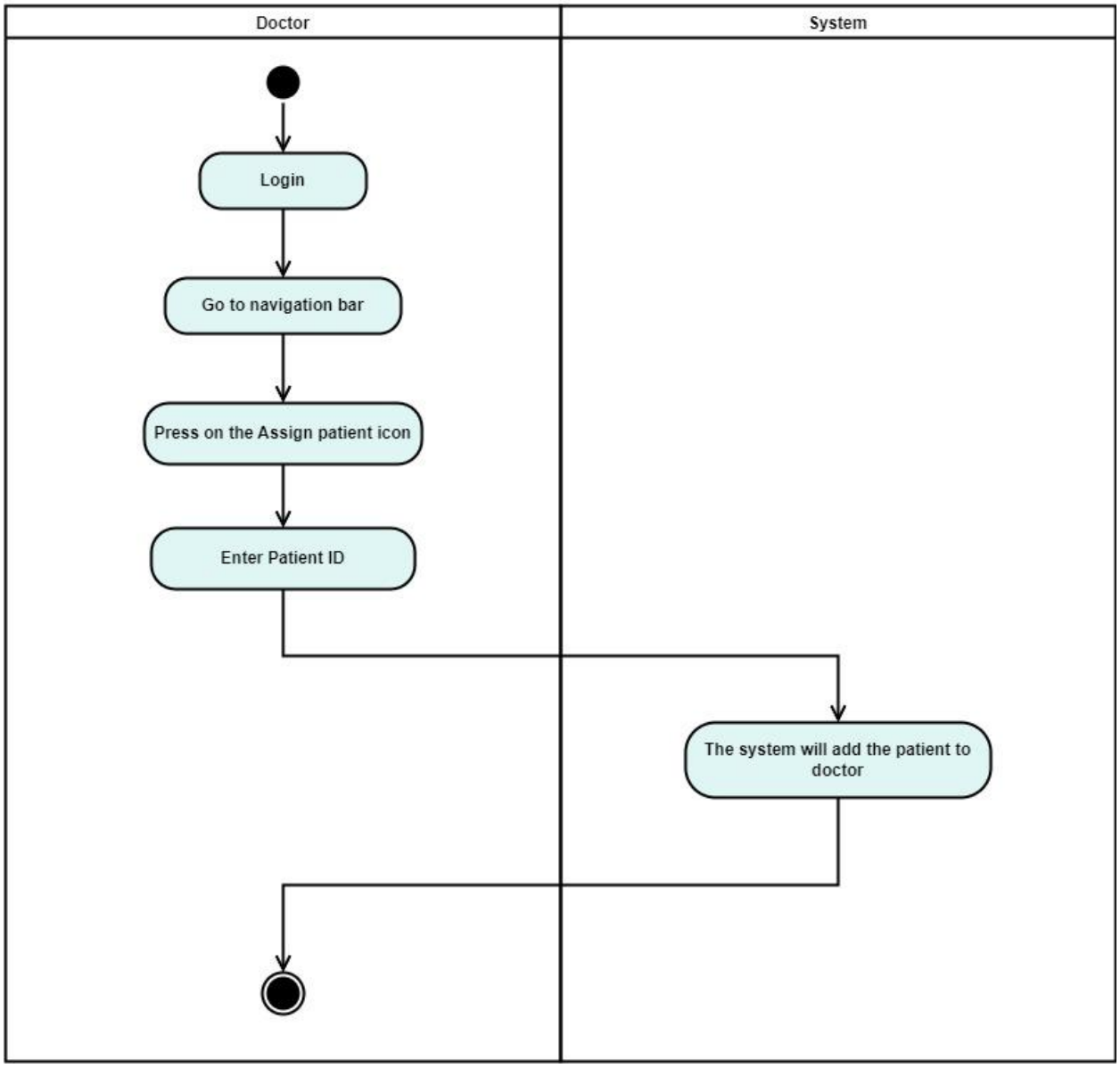
View Patient Profile :



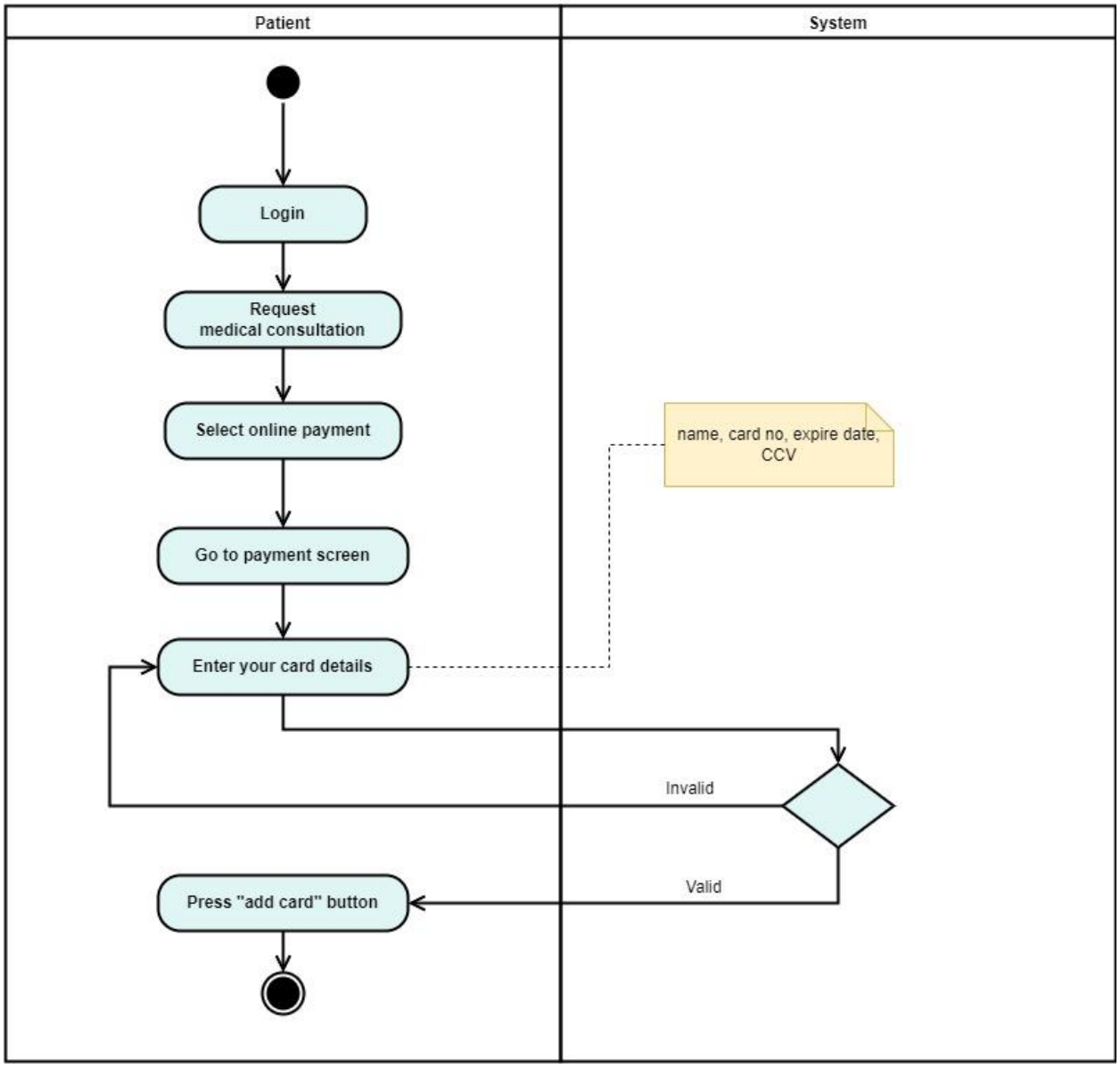
Home Visit request and receive :



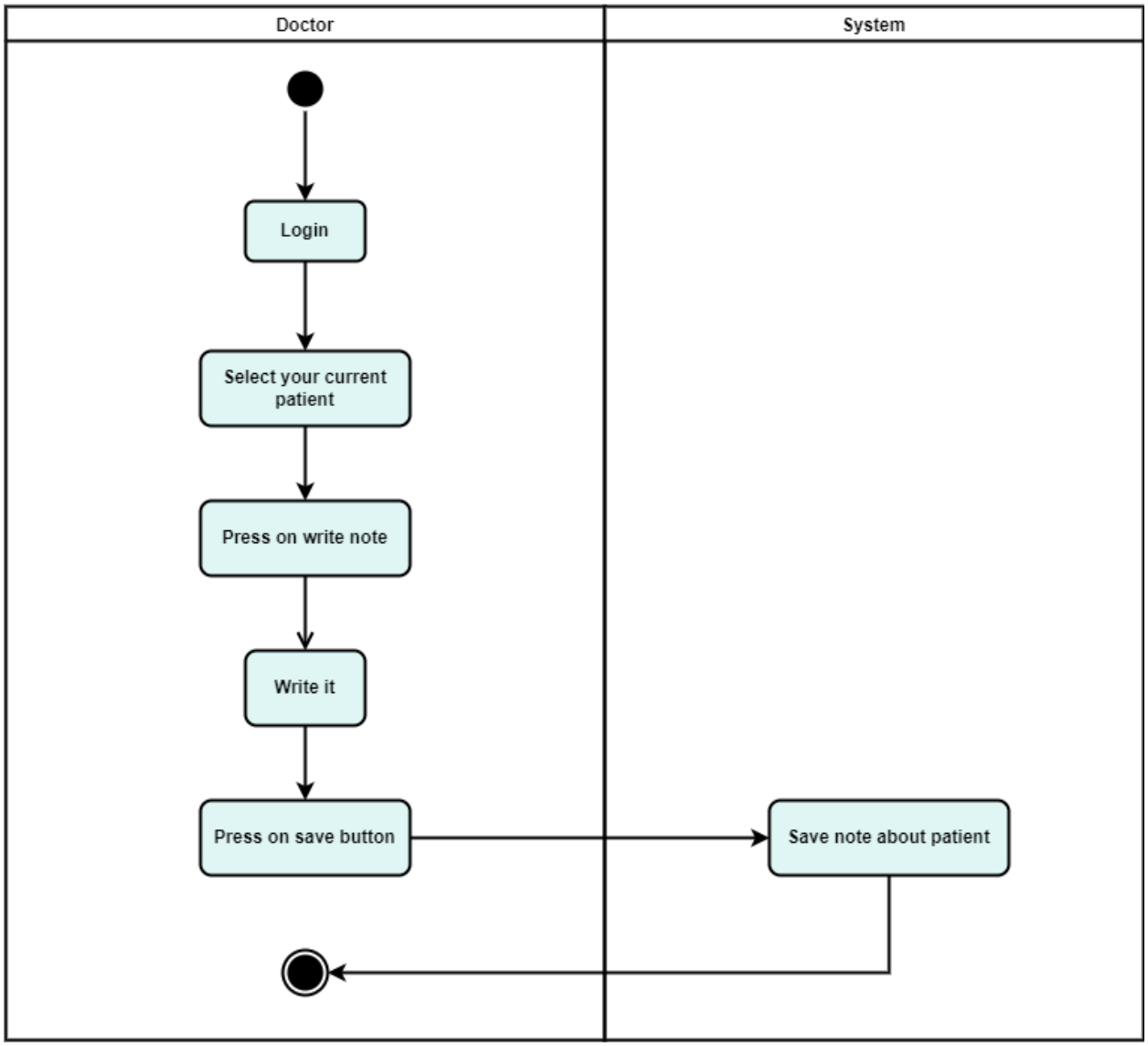
Assign Patient to Doctor :



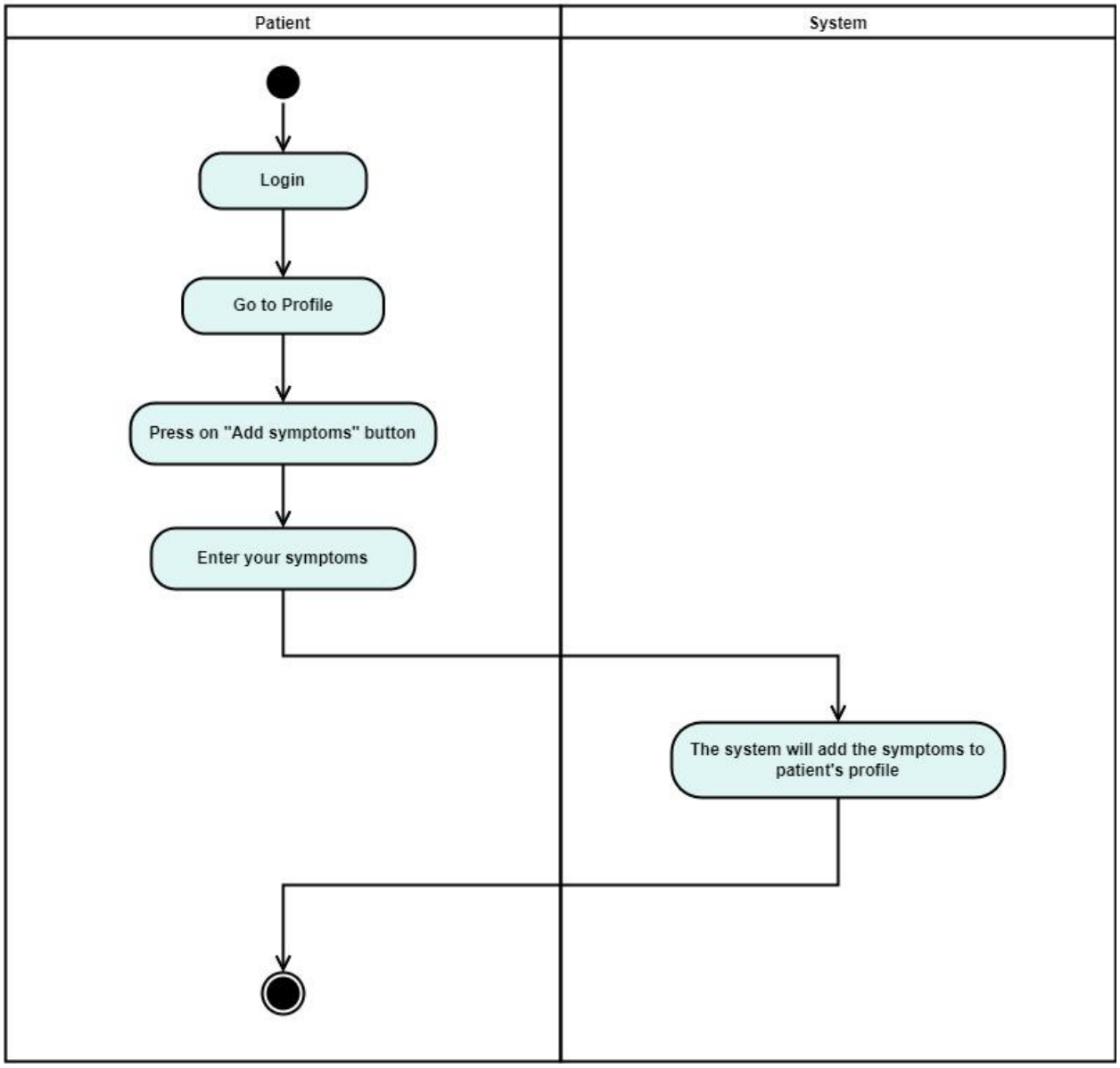
Payment :



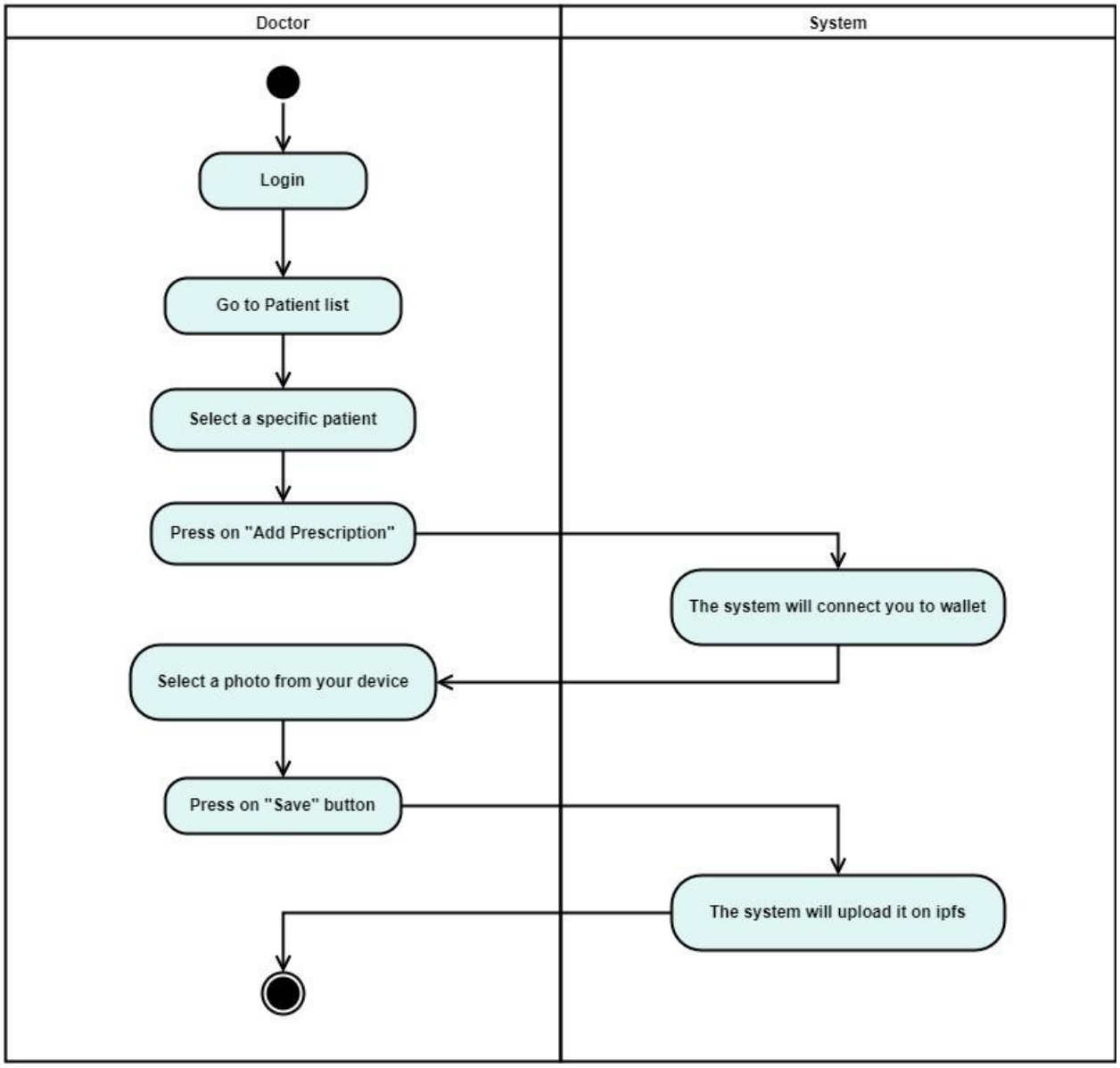
Doctor add note :



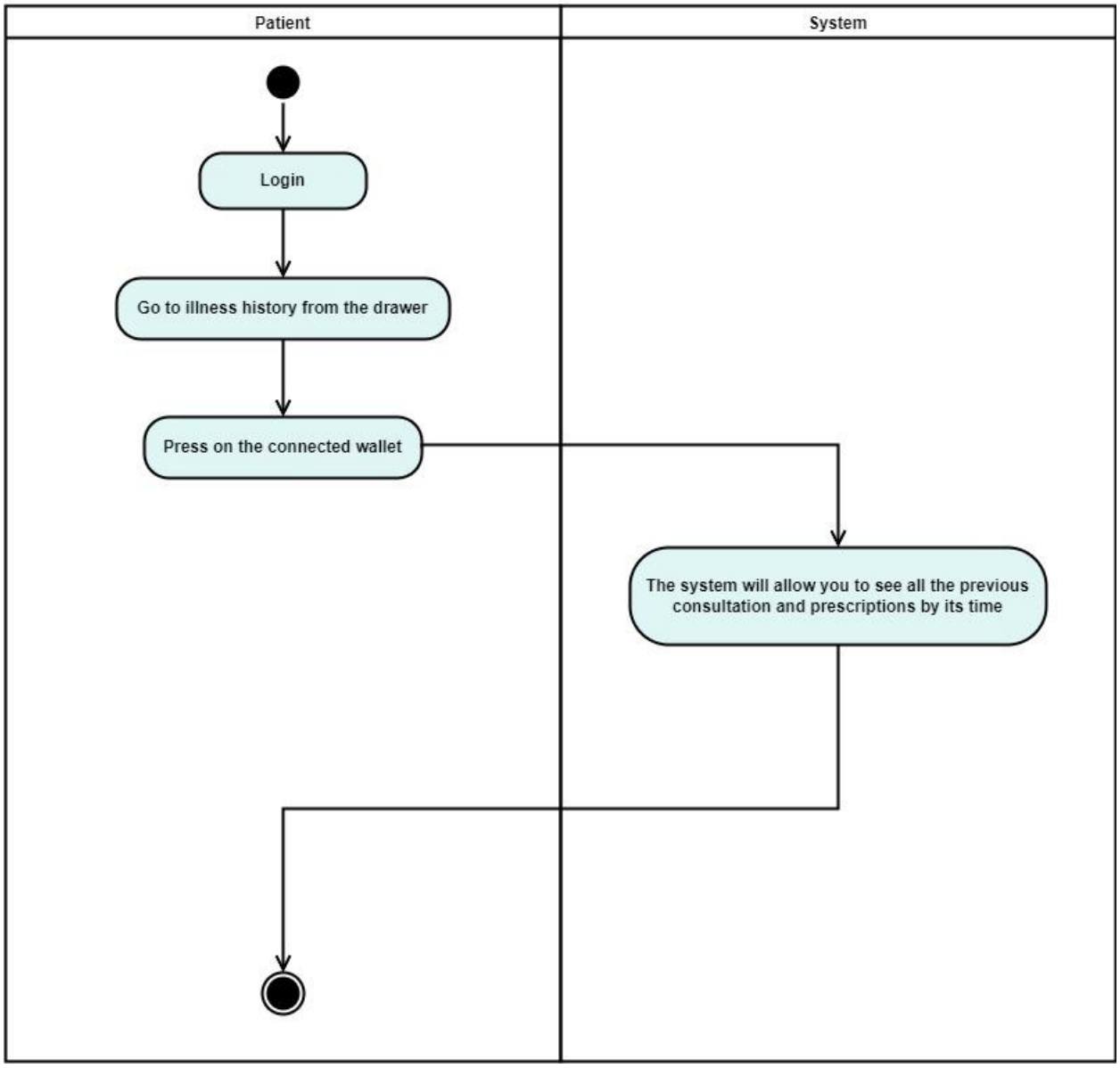
Add symptoms :



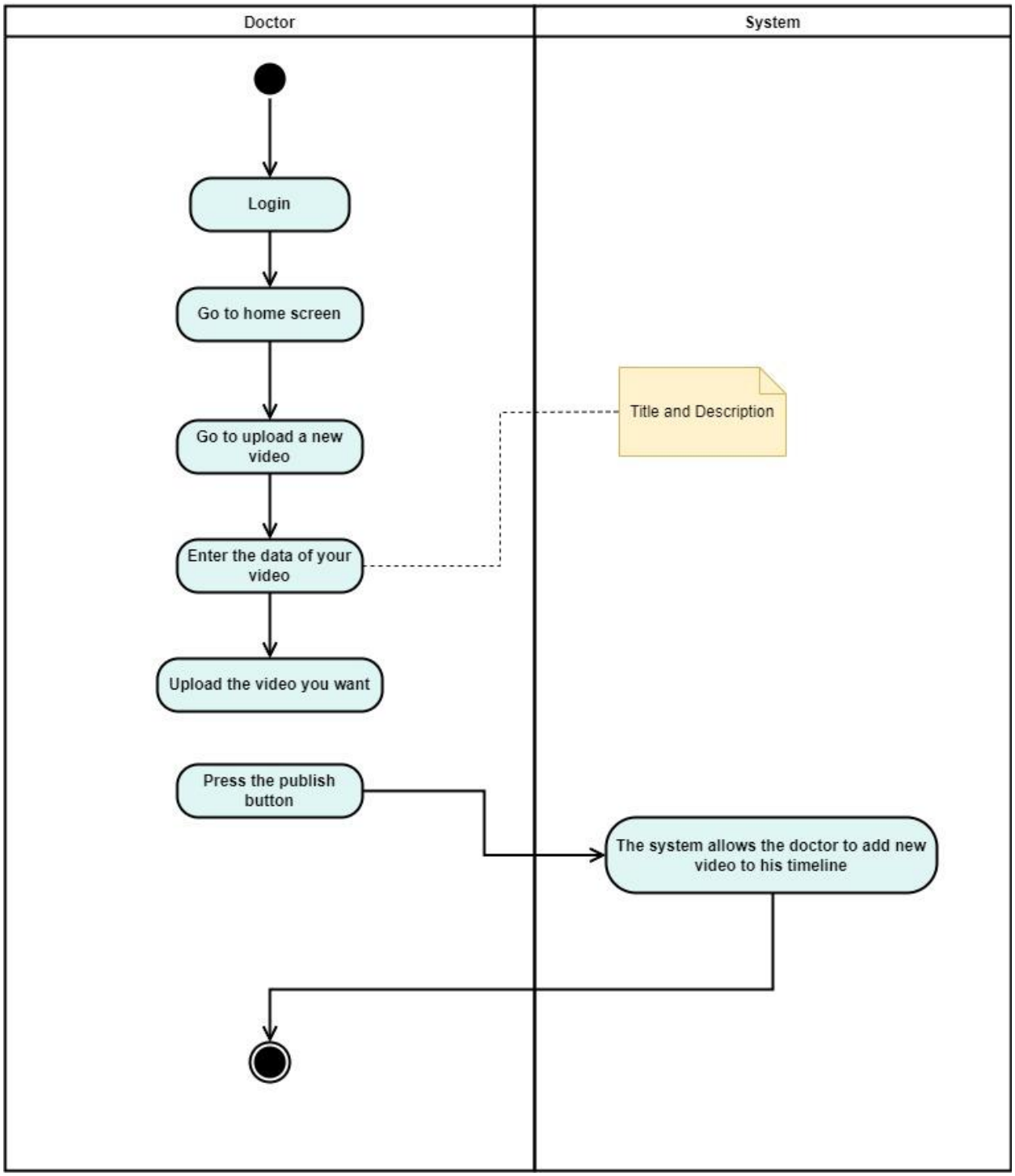
Upload health medical record :



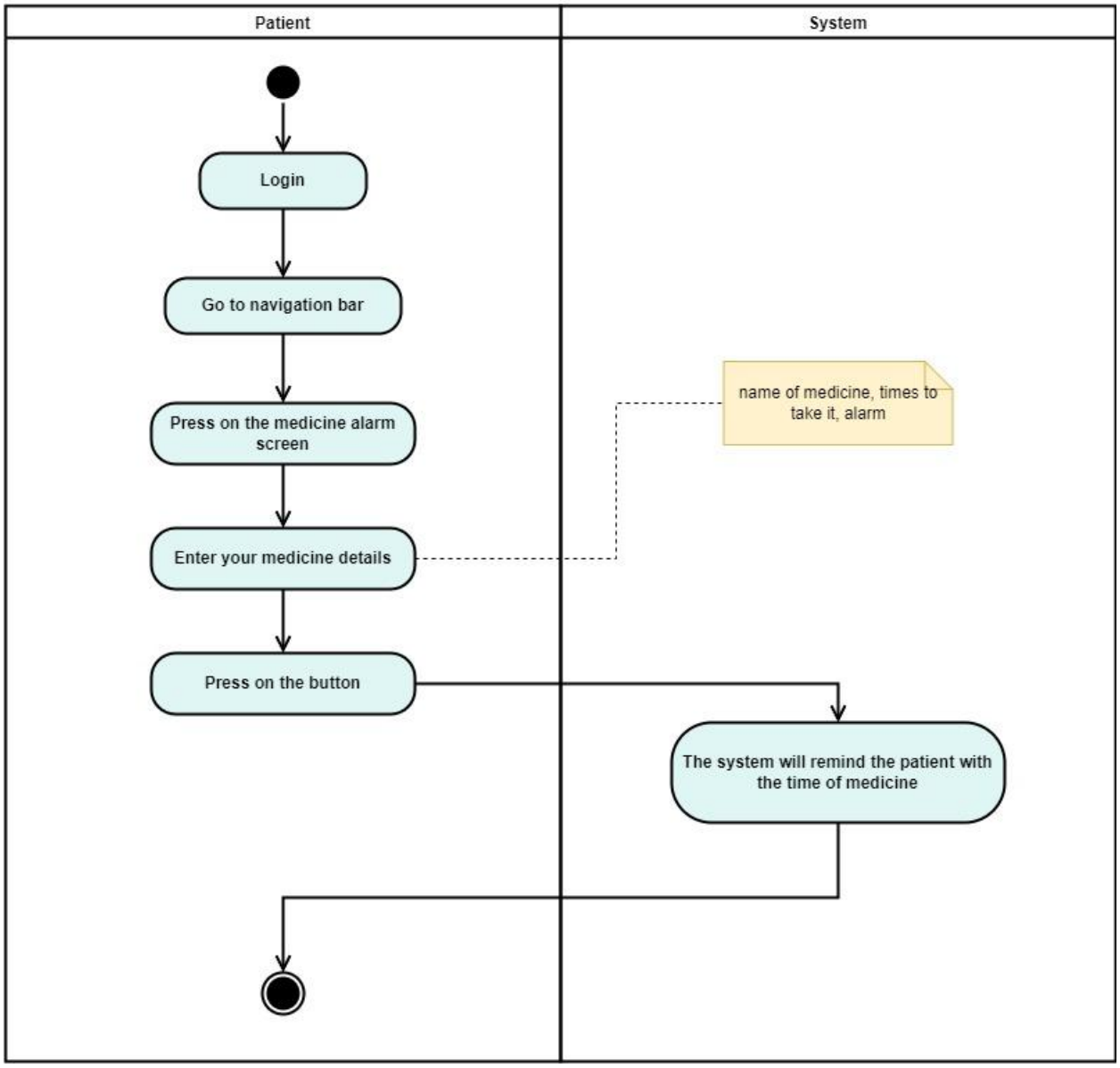
Check illness history :



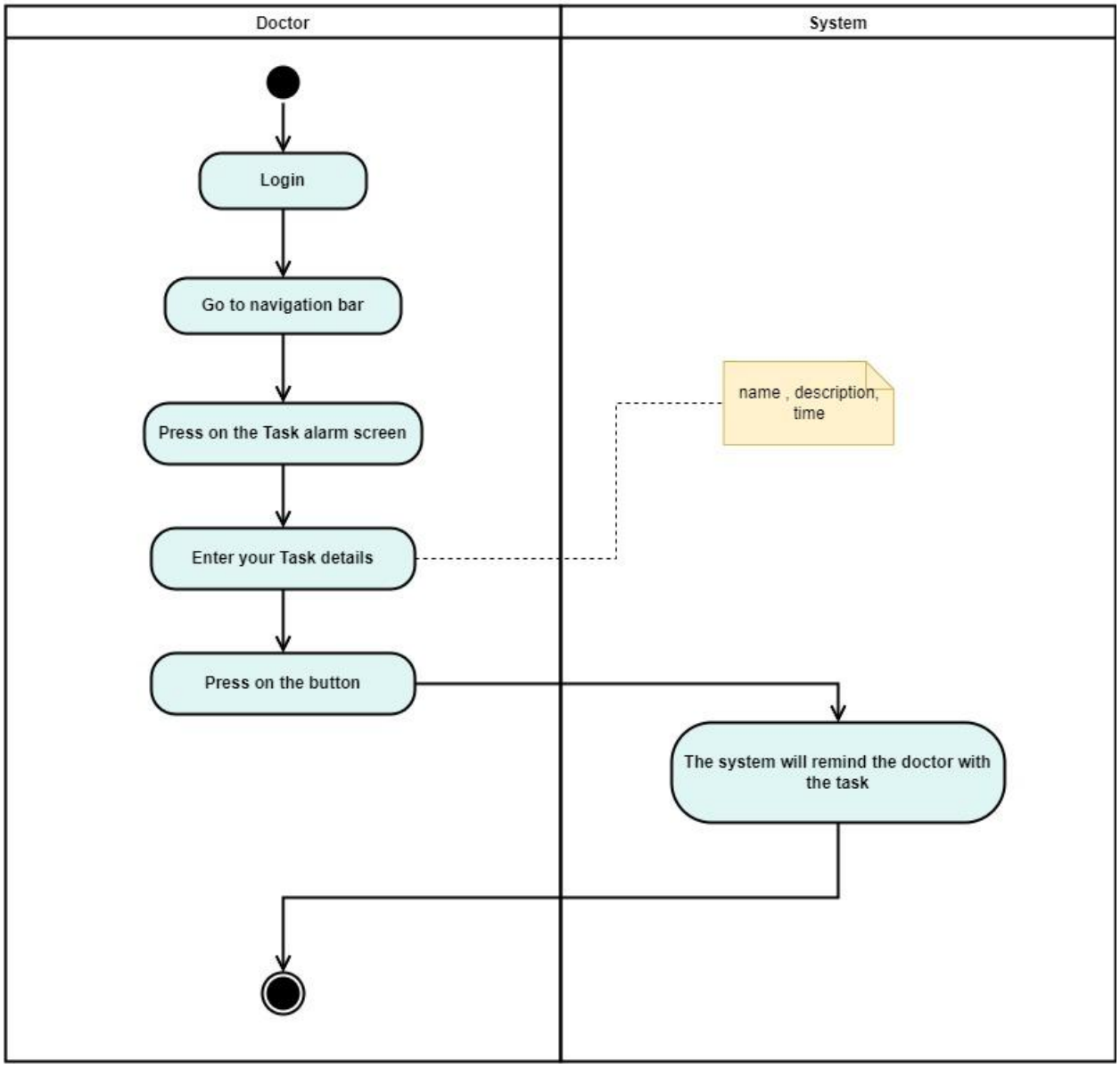
Upload video :



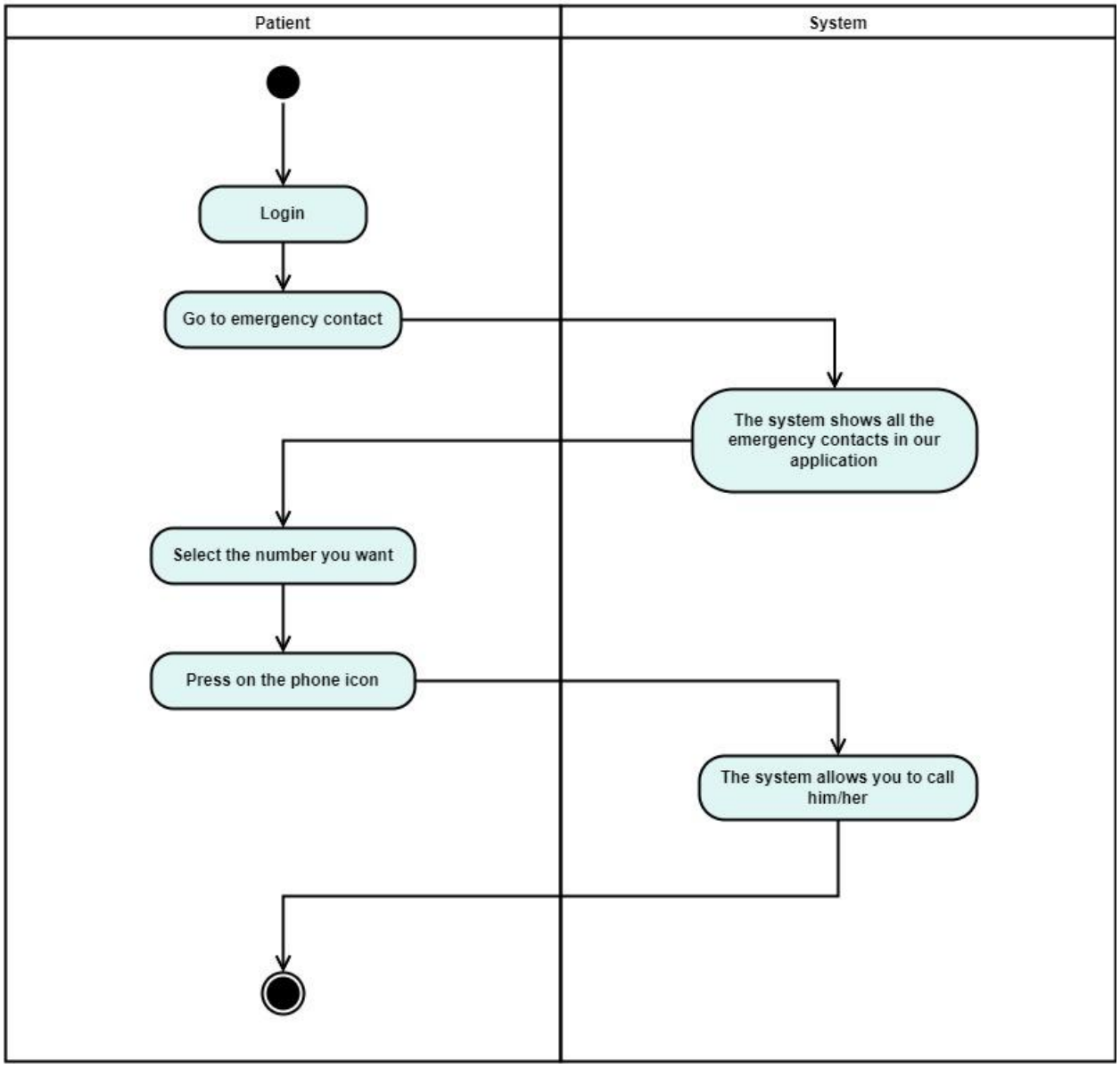
Add medication schedule :



Add Task schedule :



Calling emergency contact :

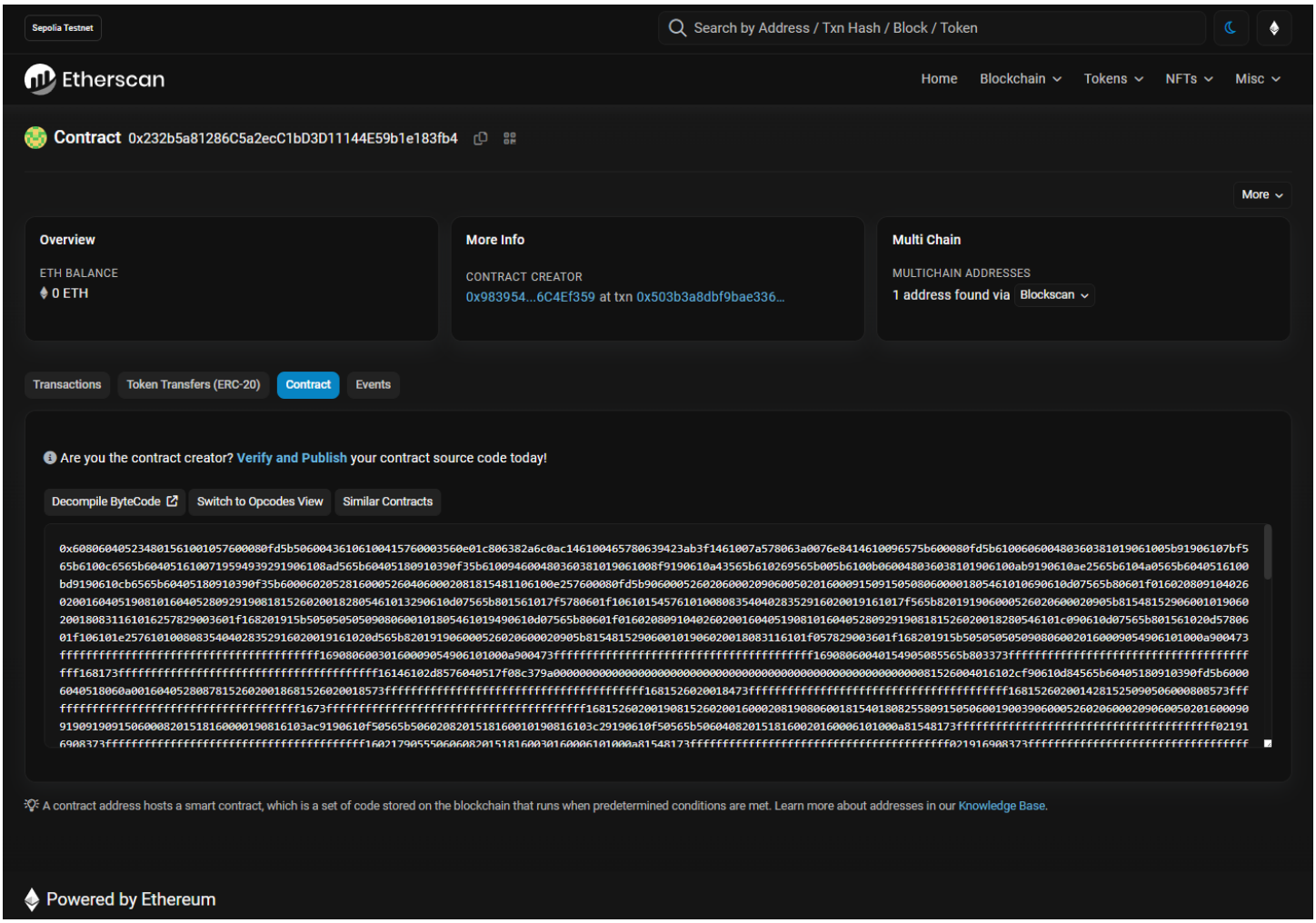


Chapter 4: Implementation

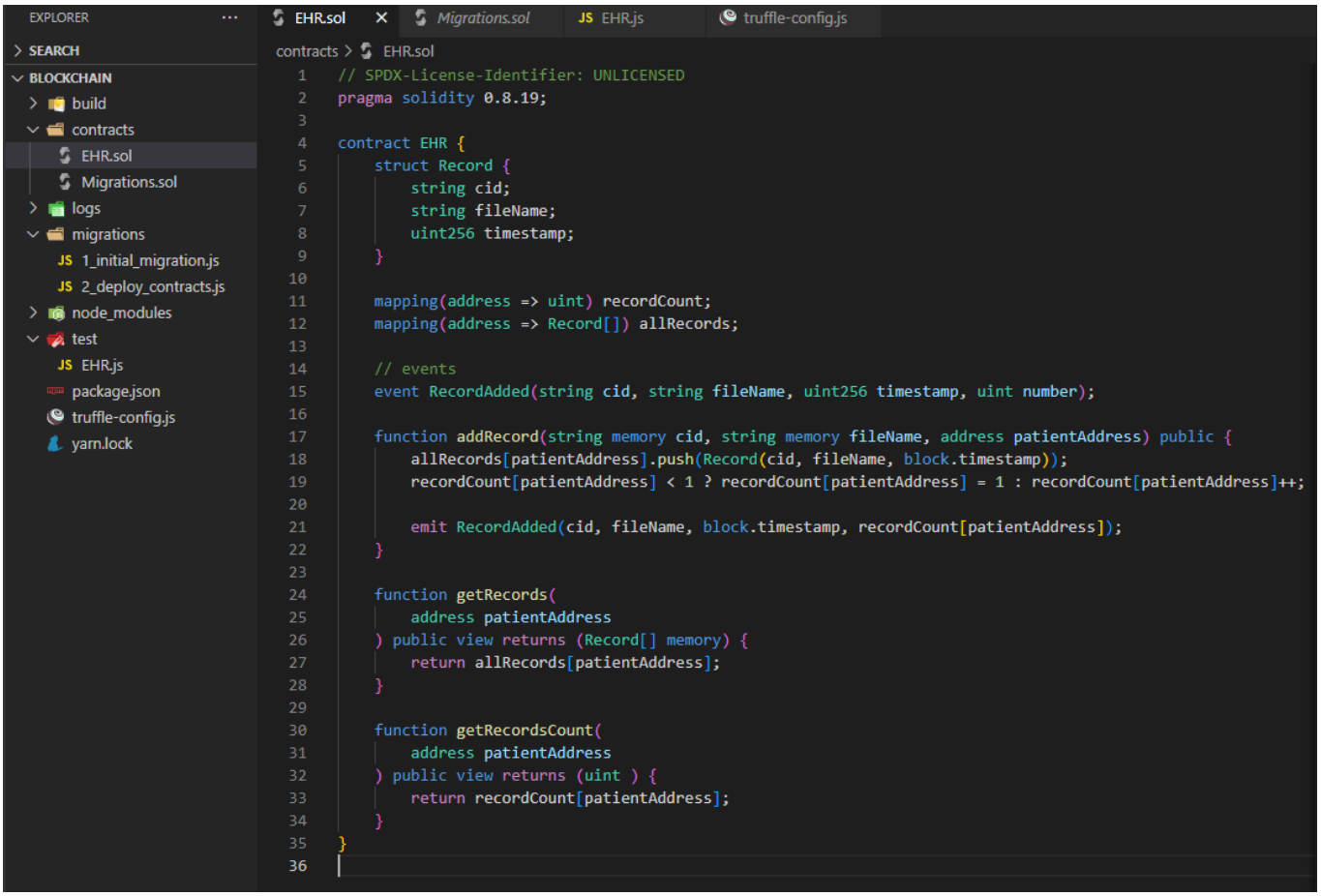
In this chapter, we will discuss and go deeper into MobiCare health system implementation and present its code and the algorithms used to build it.

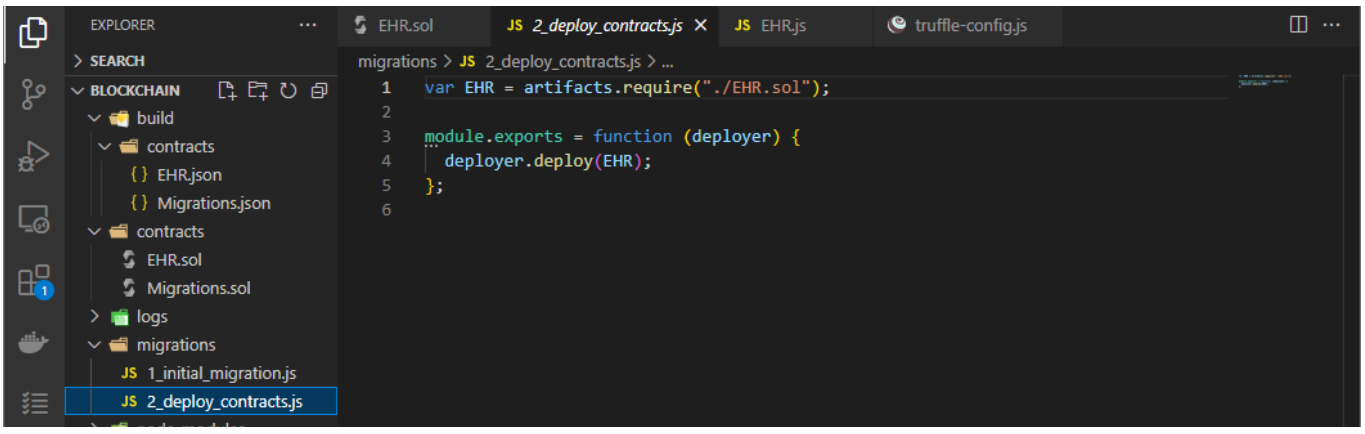
BlockChain:

Deployed our smart contract to Sepolia Test Network

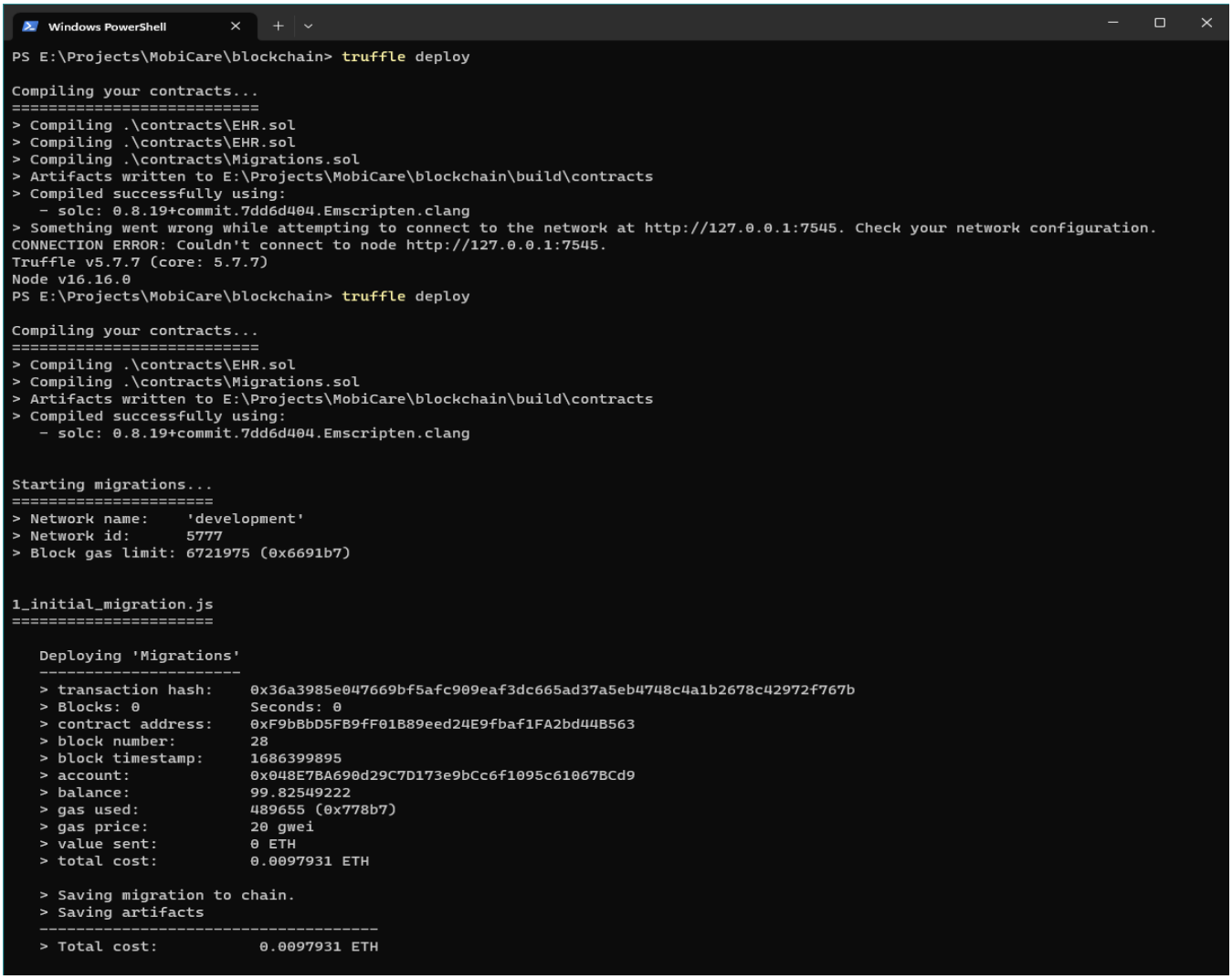


Smart Contract:





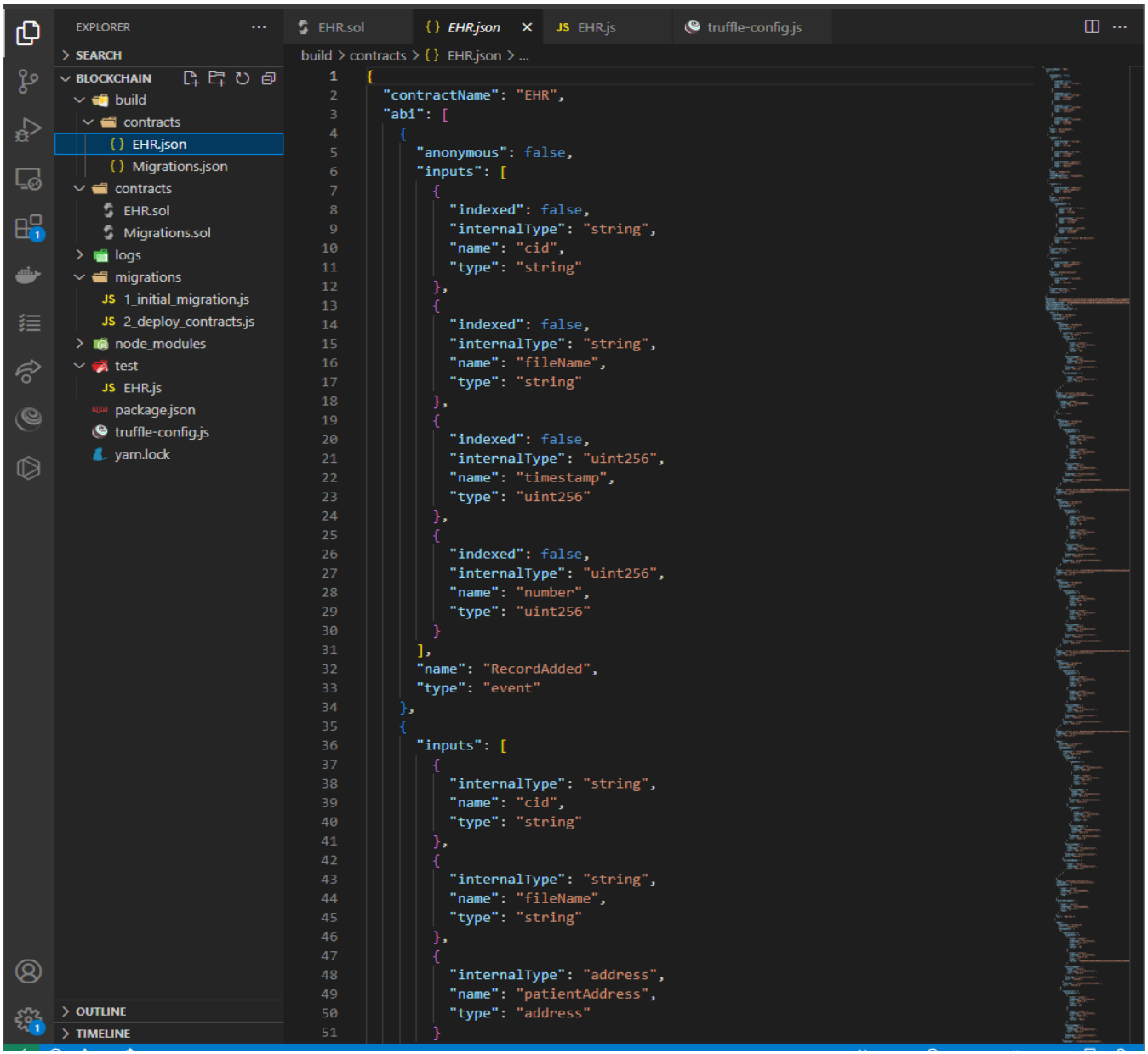
Compiling and Deploying Smart Contracts:



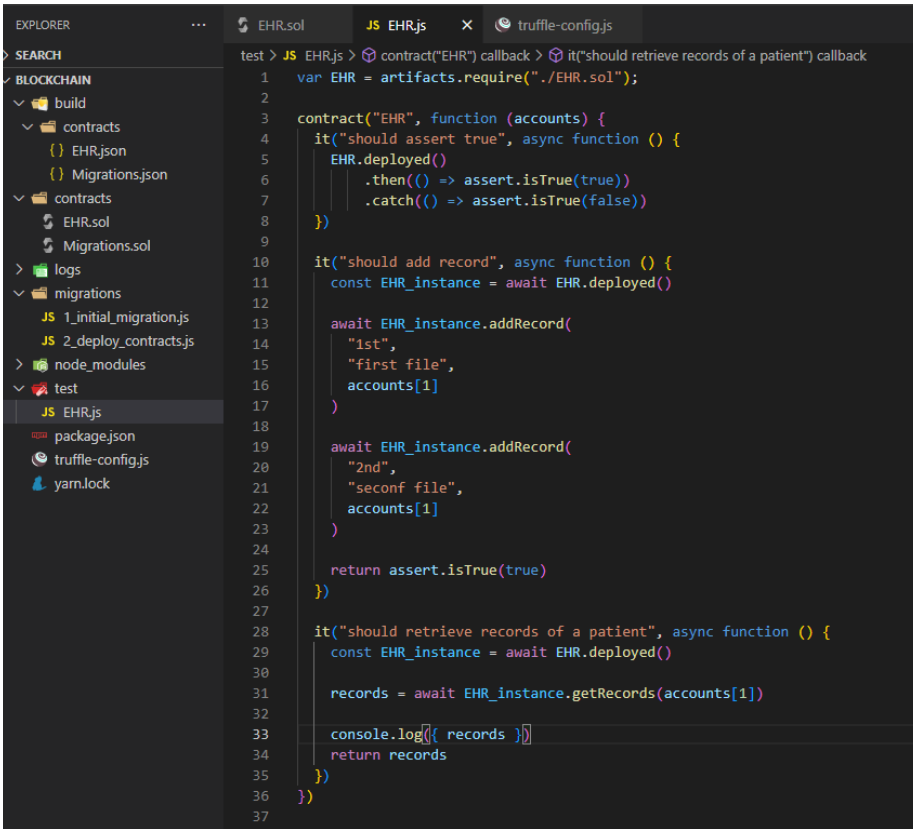
```
Windows PowerShell
PS E:\Projects\MobiCare\blockchain> truffle deploy

Compiling your contracts...
===========================
> Compiling .\contracts\EHR.sol
> Compiling .\contracts\EHR.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to E:\Projects\MobiCare\blockchain\build\contracts
> Compiled successfully using:
   - solc: 0.8.19+commit.7dd6d404.Emscripten.clang
> Something went wrong while attempting to connect to the network at http://127.0.0.1:7545. Check your network configuration.
CONNECTION ERROR: Couldn't connect to node http://127.0.0.1:7545.
Truffle v5.7.7 (core: 5.7.7)
Node v16.16.0
PS E:\Projects\MobiCare\blockchain> truffle deploy

Compiling your contracts...
===========================
> Compiling .\contracts\EHR.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to E:\Projects\MobiCare\blockchain\build\contracts
> Compiled successfully using:
   - solc: 0.8.19+commit.7dd6d404.Emscripten.clang


Starting migrations...
======================
> Network name:    'development'
> Network id:      5777
> Block gas limit: 6721975 (0x6691b7)


1_initial_migration.js
======================

   Deploying 'Migrations'
   ----------------------
   > transaction hash:    0x36a3985e047669bf5afc909eaf3dc665ad37a5eb4748c4a1b2678c42972f767b
   > Blocks: 0            Seconds: 0
   > contract address:    0xF9bBbD5FB9fF01B89eed24E9fbaf1FA2bd44B563
   > block number:        28
   > block timestamp:     1686399895
   > account:             0x048E7BA690d29C7D173e9bCc6f1095c61067BCd9
   > balance:             99.82549222
   > gas used:            489655 (0x778b7)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0097931 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:           0.0097931 ETH
```

```
2_deploy_contracts.js
=====================

   Deploying 'EHR'
   ---------------
   > transaction hash:    0x1e406a270669f99325bc1266bd6ec89ca424e727656ca49d4e58368cb5507afe
   > Blocks: 0            Seconds: 0
   > contract address:    0xc2bE9d7eba57918455147DB253DDD4b72135dDDa
   > block number:        30
   > block timestamp:     1686399896
   > account:             0x048E7BA690d29C7D173e9bCc6f1095c61067BCd9
   > balance:             99.80522748
   > gas used:            971020 (0xed10c)
   > gas price:           20 gwei
   > value sent:          0 ETH
   > total cost:          0.0194204 ETH

   > Saving migration to chain.
   > Saving artifacts
   -------------------------------------
   > Total cost:           0.0194204 ETH

Summary
=======
> Total deployments:   2
> Final cost:          0.0292135 ETH
```

Contract ABI



Unit Testing:



```
EXPLORER
SEARCH
BLOCKCHAIN
  build
    contracts
      EHR.json
      Migrations.json
  contracts
    EHR.sol
    Migrations.sol
  logs
  migrations
    1_initial_migration.js
    2_deploy_contracts.js
  node_modules
  test
    EHR.js
  package.json
  truffle-config.js
  yarn.lock
```

EHR.sol | EHR.js | truffle-config.js

test > EHR.js > contract("EHR") callback > it("should retrieve records of a patient") callback

```
1  var EHR = artifacts.require("./EHR.sol");
2
3  contract("EHR", function (accounts) {
4    it("should assert true", async function () {
5      EHR.deployed()
6          .then(() => assert.isTrue(true))
7          .catch(() => assert.isTrue(false))
8    })
9
10   it("should add record", async function () {
11     const EHR_instance = await EHR.deployed()
12
13     await EHR_instance.addRecord(
14       "1st",
15       "first file",
16       accounts[1]
17     )
18
19     await EHR_instance.addRecord(
20       "2nd",
21       "seconf file",
22       accounts[1]
23     )
24
25     return assert.isTrue(true)
26   })
27
28   it("should retrieve records of a patient", async function () {
29     const EHR_instance = await EHR.deployed()
30
31     records = await EHR_instance.getRecords(accounts[1])
32
33     console.log({ records })
34     return records
35   })
36 })
37
```

```
PS E:\Projects\MobiCare\blockchain> truffle test
Using network 'development'.


Compiling your contracts...
===========================
> Compiling .\contracts\EHR.sol
> Compiling .\contracts\EHR.sol
> Compiling .\contracts\Migrations.sol
> Artifacts written to C:\Users\dell\AppData\Local\Temp\test--10572-M
yEGPD051vU4
> Compiled successfully using:
   - solc: 0.8.19+commit.7dd6d404.Emscripten.clang


  Contract: EHR
    ✓ should assert true
    ✓ should add record (612ms)
{
  records: [
    [
      '1st',
      'first file',
      '1686399554',
      cid: '1st',
      fileName: 'first file',
      timestamp: '1686399554'
    ],
    [
      '2nd',
      'seconf file',
      '1686399555',
      cid: '2nd',
      fileName: 'seconf file',
      timestamp: '1686399555'
    ]
  ]
}
    ✓ should retrieve records of a patient (132ms)


  3 passing (952ms)
```

Development Environment

```
PS E:\Projects\MobiCare\blockchain> truffle console
truffle(development)> const contract = require('truffle-contract');
undefined
truffle(development)> const EHR_artifact = require('./build/contracts
/EHR.json');
undefined
truffle(development)> EHR_artifact
{
  contractName: 'EHR',
  abi: [
    {
      anonymous: false,
      inputs: [Array],
      name: 'RecordAdded',
      type: 'event'
    },
    {
      inputs: [Array],
      name: 'addRecord',
      outputs: [],
      stateMutability: 'nonpayable',
      type: 'function'
    },
    {
      inputs: [Array],
      name: 'getRecords',
      outputs: [Array],
      stateMutability: 'view',
      type: 'function'
    }
  ],
  metadata: '{"compiler":{"version":"0.8.19+commit.7dd6d404"},"langua
ge":"Solidity","output":{"abi":[{"anonymous":false,"inputs":[{"indexe
d":false,"internalType":"string","name":"cid","type":"string"},{"inde
xed":false,"internalType":"string","name":"fileName","type":"string"}
,{"indexed":false,"internalType":"uint256","name":"timestamp","type":
"uint256"},{"indexed":false,"internalType":"uint256","name":"number",
"type":"uint256"}],"name":"RecordAdded","type":"event"},{"inputs":[{"
internalType":"string","name":"cid","type":"string"},{"internalType":
"string","name":"fileName","type":"string"},{"internalType":"address"
,"name":"patientAddress","type":"address"}],"name":"addRecord","outpu
```

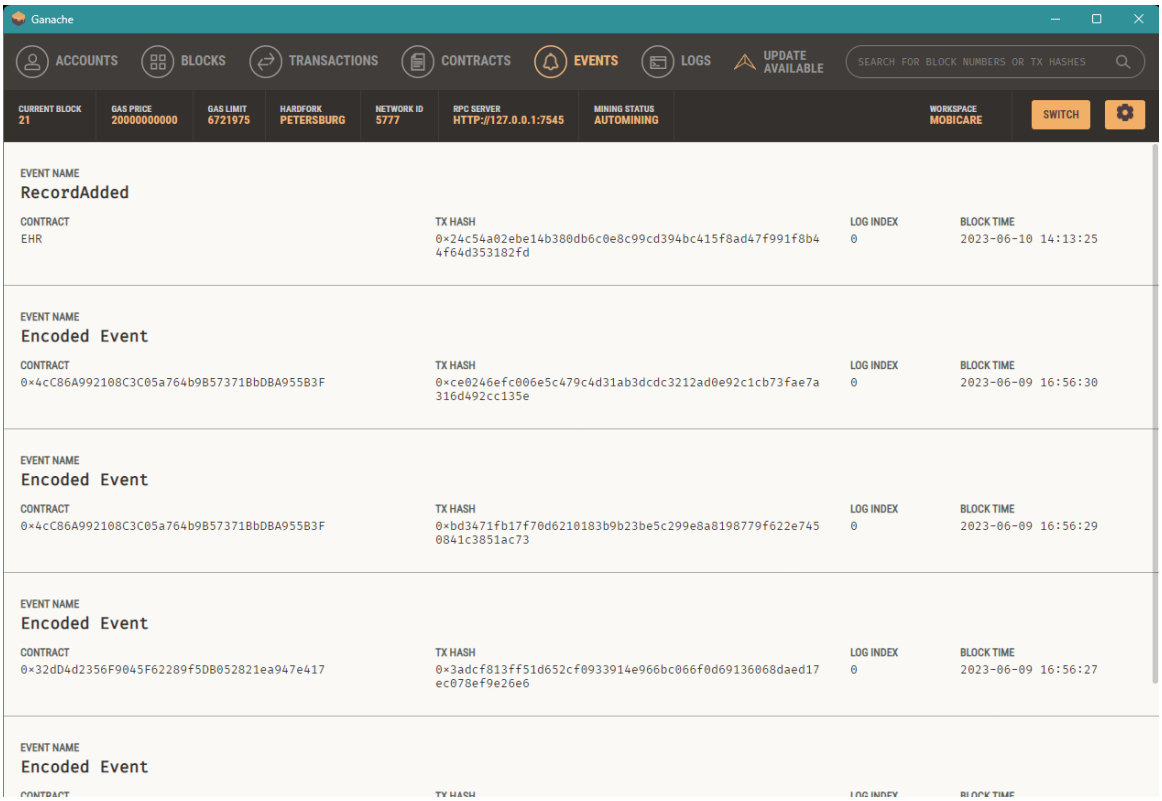
Test Accounts and Deployed Contract Object

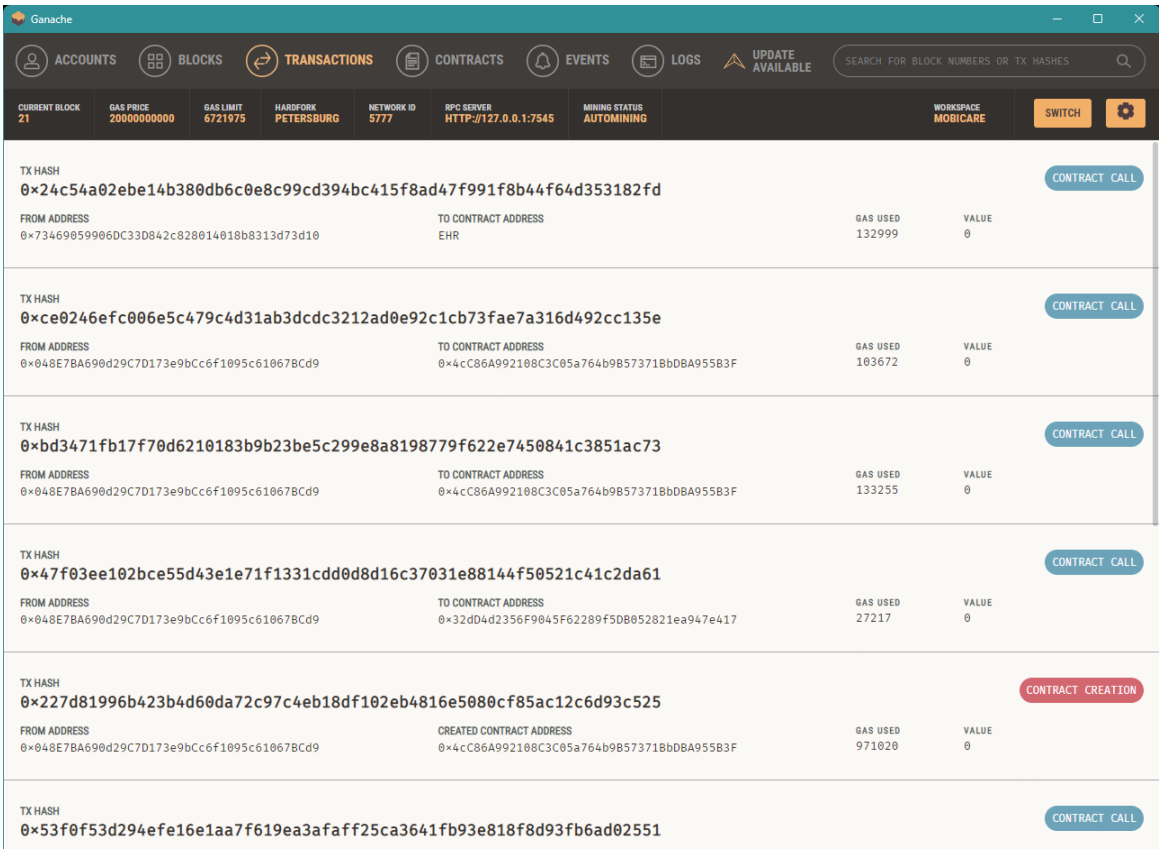
```
truffle(development)> web3.eth.getAccounts()
[
  '0x048E7BA690d29C7D173e9bCc6f1095c61067BCd9',
  '0x73469059906DC33D842c828014018b8313d73d10',
  '0x23a30aeEd9182162ffAB46c517616e040d716C46',
  '0x84a27496FFd8a0A262990Bf5a5f4Ad73037814C3',
  '0x5779cF00de1d91300581BdBc7F50A394057cC547',
  '0xA2A6a824759B92635dd6557344431F72cC258Dd5',
  '0x4591ccD7357A51517cBEE27BEC9691bF625a0747',
  '0x11057bF9230A19d792bd0716e95EE9016dDBDE26',
  '0x42970Bad6F0d3C642A9DEca9A59e6E2cb28057eC',
  '0x4AdE65734bE25b0770aEbc3aFD5b83a2Ce26d90b'
]
truffle(development)> EHR.deployed()
TruffleContract {
  constructor: [Function: TruffleContract] {
    _constructorMethods: {
      setProvider: [Function: setProvider],
      new: [Function: new],
      at: [AsyncFunction: at],
      deployed: [AsyncFunction: deployed],
      defaults: [Function: defaults],
      hasNetwork: [Function: hasNetwork],
      isDeployed: [Function: isDeployed],
      detectNetwork: [AsyncFunction: detectNetwork],
      setNetwork: [Function: setNetwork],
      setNetworkType: [Function: setNetworkType],
      setWallet: [Function: setWallet],
      resetAddress: [Function: resetAddress],
      link: [Function: link],
      clone: [Function: clone],
      addProp: [Function: addProp],
      toJSON: [Function: toJSON],
      decodeLogs: [Function: decodeLogs]
    },
    _properties: {
      contract_name: [Object],
      contractName: [Object],
      gasMultiplier: [Object],
      timeoutBlocks: [Object],
      autoGas: [Object],
      numberFormat: [Object],
      abi: [Object],
      metadata: [Function: metadata],
      network: [Function: network],
      networks: [Function: networks],
```

Add Record (create new transaction)

```
truffle(development)> EHR.deployed().then(instance => instance.addRec
ord("cid", "fileNe", '0x42970Bad6F0d3C642A9DEca9A59e6E2cb28057eC', {f
rom: '0x73469059906DC33D842c828014018b8313d73d10'}))
{
  tx: '0x24c54a02ebe14b380db6c0e8c99cd394bc415f8ad47f991f8b44f64d3531
82fd',
  receipt: {
    transactionHash: '0x24c54a02ebe14b380db6c0e8c99cd394bc415f8ad47f9
91f8b44f64d353182fd',
    transactionIndex: 0,
    blockNumber: 21,
    blockHash: '0x2f30c61ddfc307aa900e75b3bc8a8a8a6b9bf909d8f0521007b
458d2e41b1f6a',
    from: '0x73469059906dc33d842c828014018b8313d73d10',
    to: '0xafbd5bcba19f9489150badd4f59db81d6ec0ab4c',
    cumulativeGasUsed: 132999,
    gasUsed: 132999,
    contractAddress: null,
    logs: [ [Object] ],
    logsBloom: '0x00000000000000000080000004000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000040000000000000000000000000000
00000000000010000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000008
00000000000000000000000000000040000000000000',
    status: true,
    effectiveGasPrice: '0x4a817c800',
    rawLogs: [ [Object] ]
  },
  logs: [
    {
      address: '0xaFbd5BcBA19f9489150baDD4f59Db81d6EC0aB4c',
      blockHash: '0x2f30c61ddfc307aa900e75b3bc8a8a8a6b9bf909d8f052100
7b458d2e41b1f6a',
      blockNumber: 21,
      logIndex: 0,
      removed: false,
      transactionHash: '0x24c54a02ebe14b380db6c0e8c99cd394bc415f8ad47
f991f8b44f64d353182fd',
      transactionIndex: 0,
      id: 'log_63c94376',
      event: 'RecordAdded',
      args: [Result]
    }
  ]
}
```

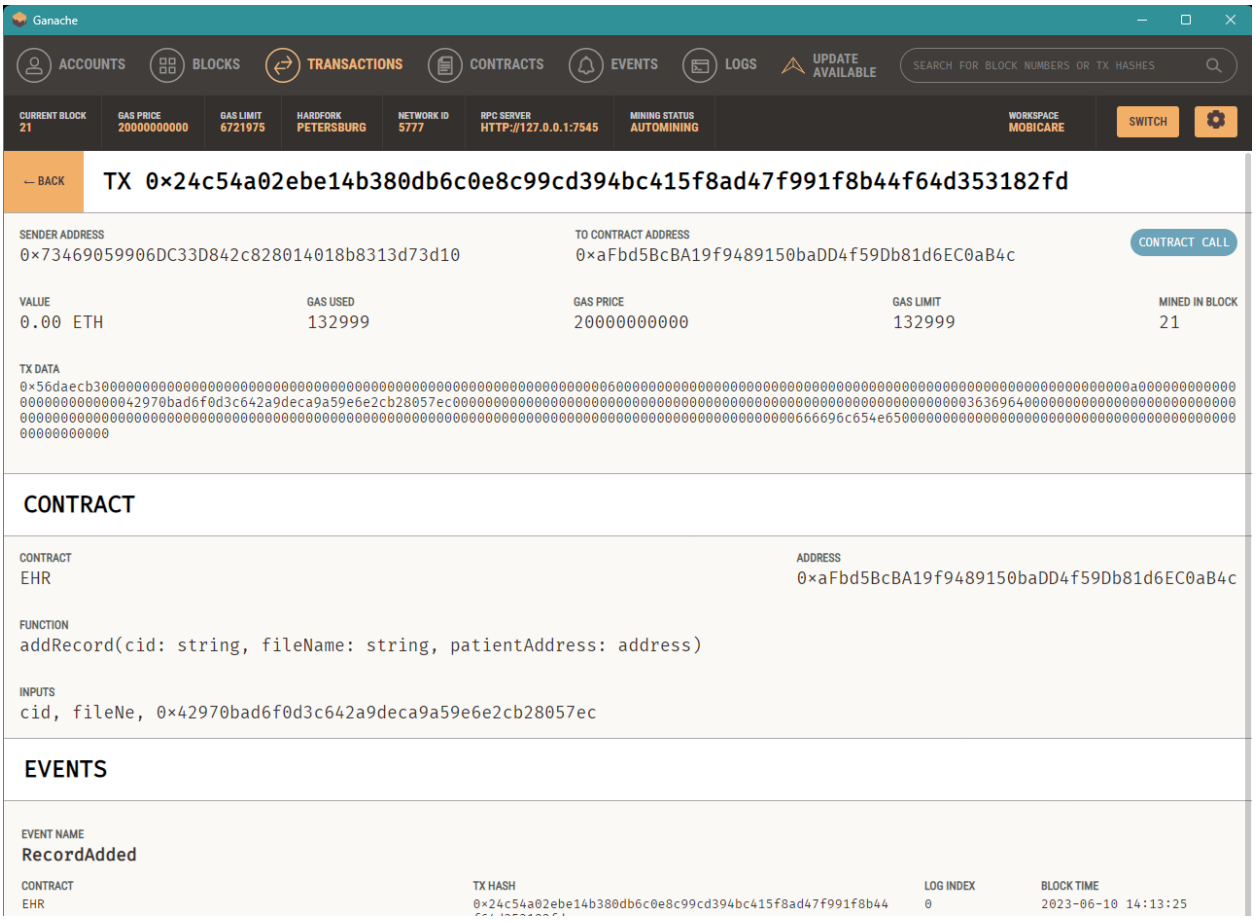
Events and Transactions





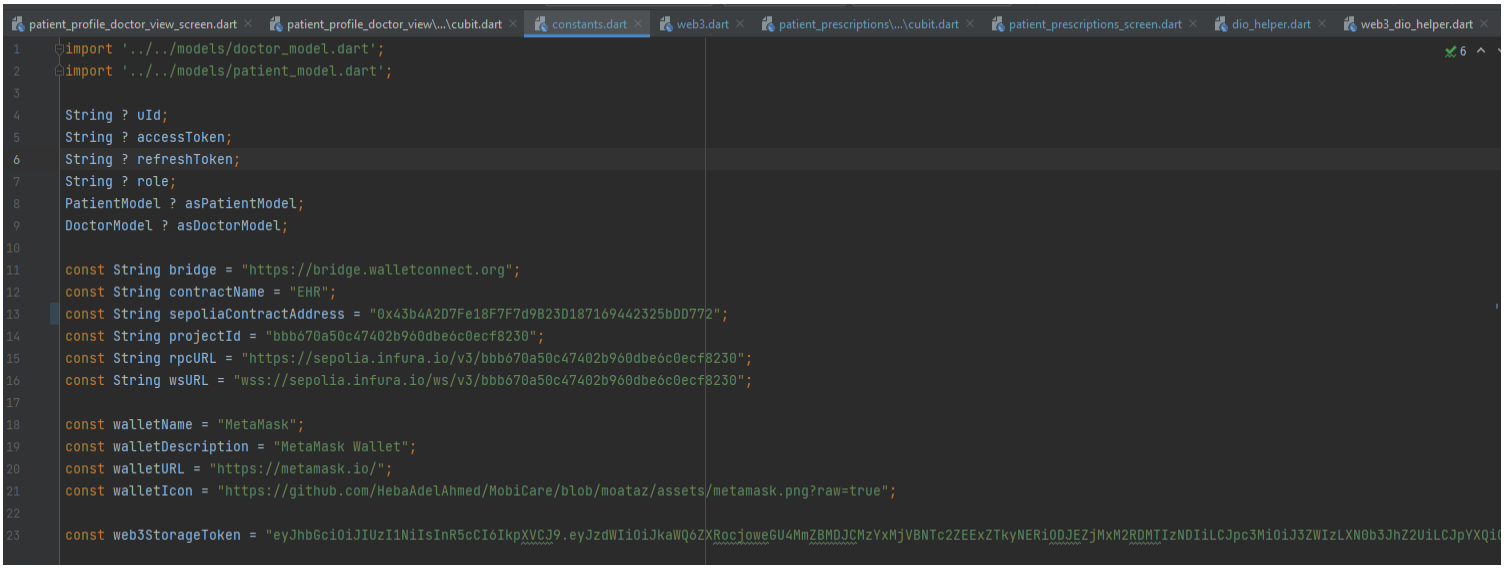
Get Records by patient address

```
truffle(development)> EHR.deployed().then(instance => instance.getRec
ords('0x42970Bad6F0d3C642A9DEca9A59e6E2cb28057eC'))
[
  [
    'cid',
    'fileNe',
    '1686399205',
    cid: 'cid',
    fileName: 'fileNe',
    timestamp: '1686399205'
  ]
]
```



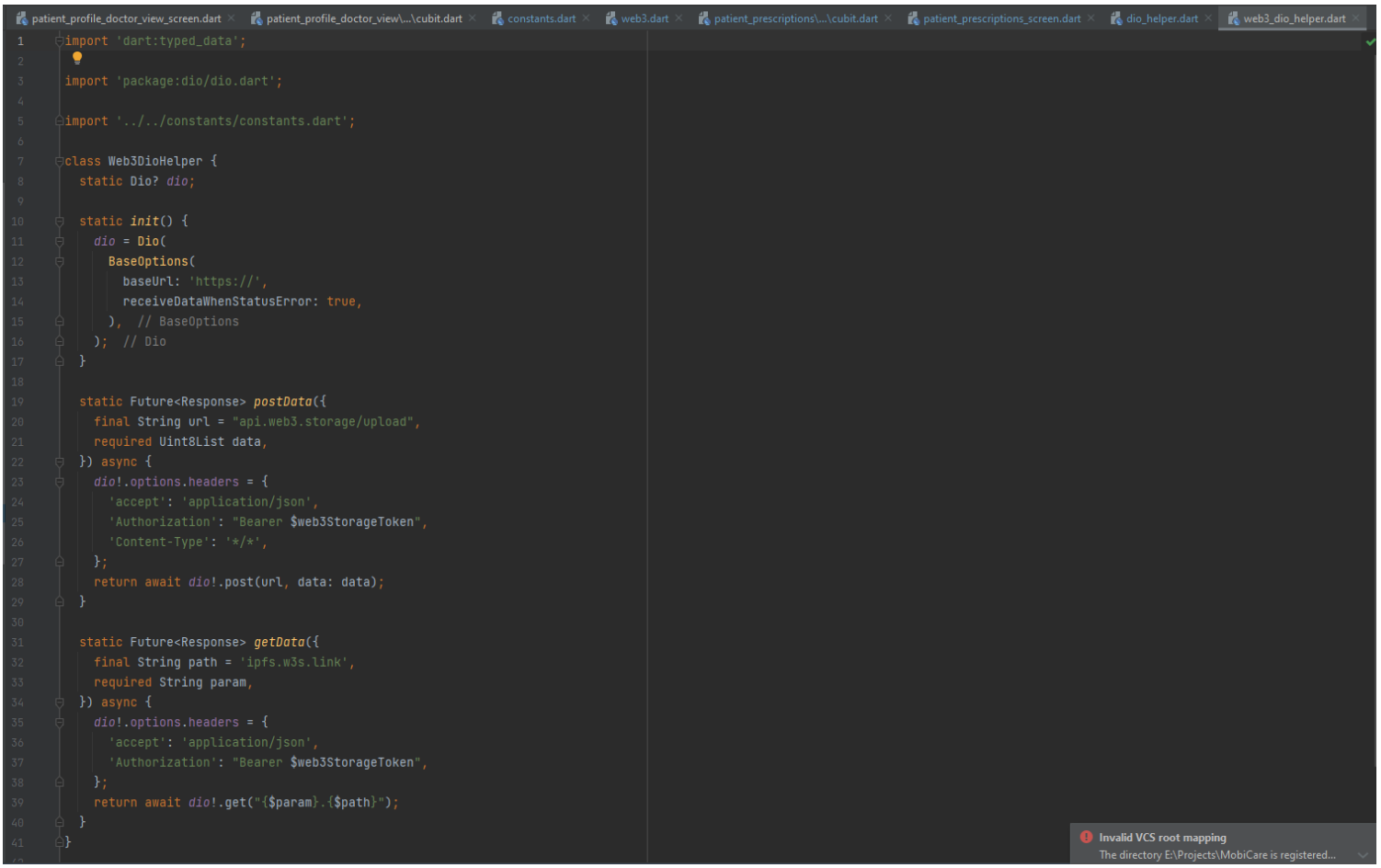
Flutter Blockchain:

Constants we used to integrate blockchain



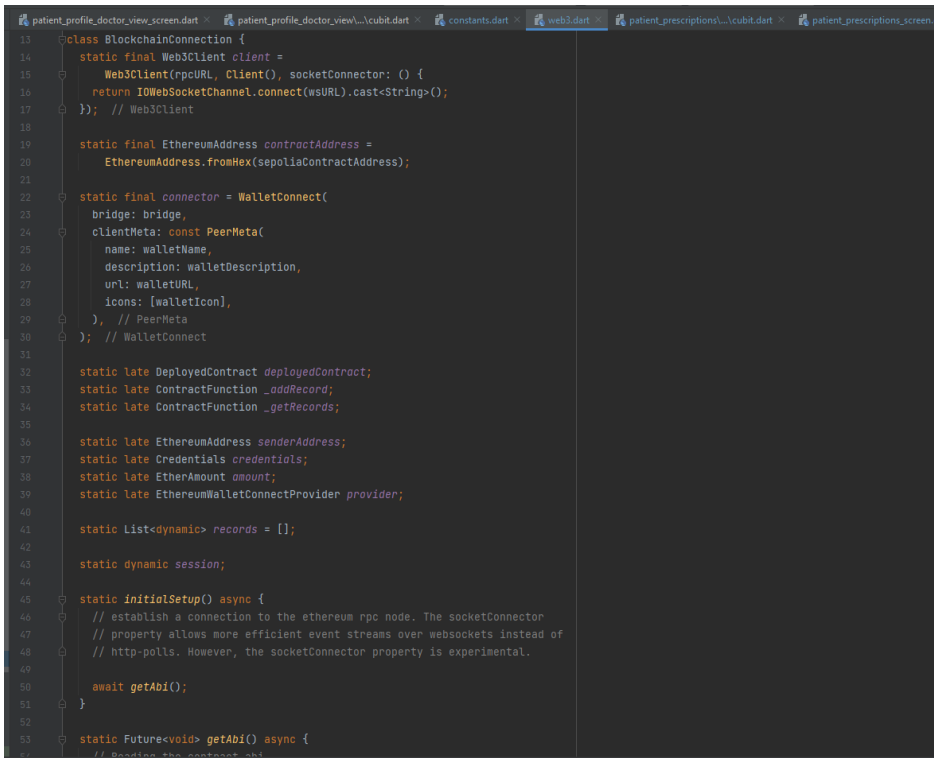
```dart
import '../../models/doctor_model.dart';
import '../../models/patient_model.dart';

String ? uId;
String ? accessToken;
String ? refreshToken;
String ? role;
PatientModel ? asPatientModel;
DoctorModel ? asDoctorModel;

const String bridge = "https://bridge.walletconnect.org";
const String contractName = "EHR";
const String sepoliaContractAddress = "0x43b4A2D7Fe18F7F7d9B23D187169442325bDD772";
const String projectId = "bbb670a50c47402b960dbe6c0ecf8230";
const String rpcURL = "https://sepolia.infura.io/v3/bbb670a50c47402b960dbe6c0ecf8230";
const String wsURL = "wss://sepolia.infura.io/ws/v3/bbb670a50c47402b960dbe6c0ecf8230";

const walletName = "MetaMask";
const walletDescription = "MetaMask Wallet";
const walletURL = "https://metamask.io/";
const walletIcon = "https://github.com/HebaAdelAhmed/MobiCare/blob/moataz/assets/metamask.png?raw=true";

const web3StorageToken = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJkaWQ6ZXRocjoweGU4MmZBMDJCMzYxMjVBNTc2ZEExZTkyNERiODJEZjMxM2RDMTIzNDIiLCJpc3MiOiJ3ZWIzLXN0b3JhZ2UiLCJpYXQi
```

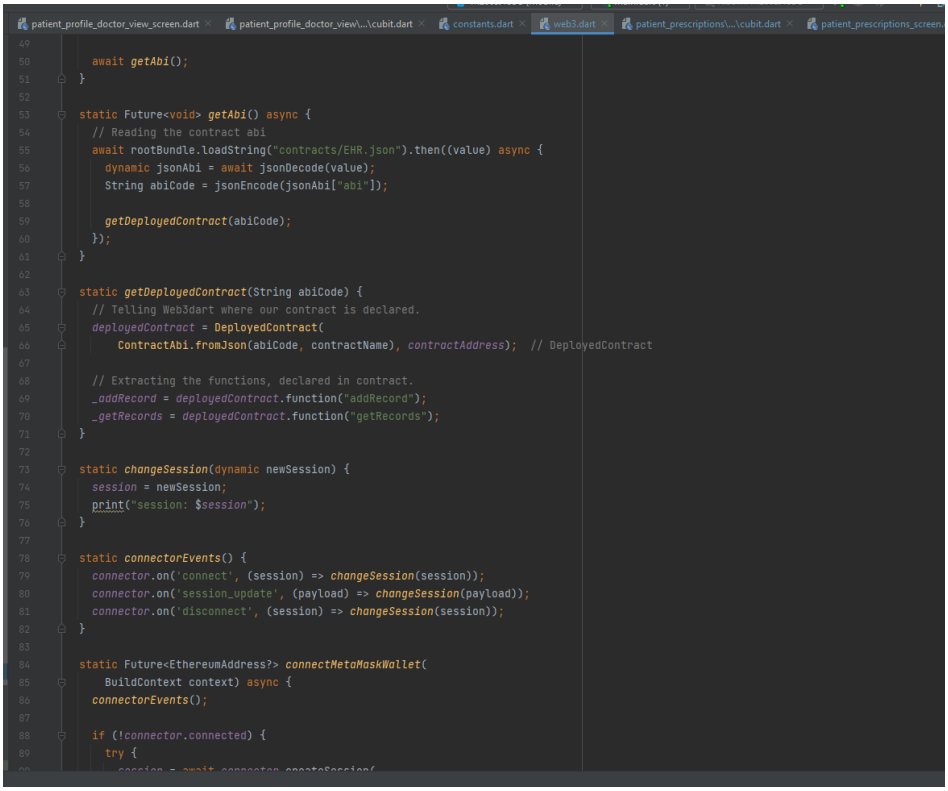
End points of web3 storage

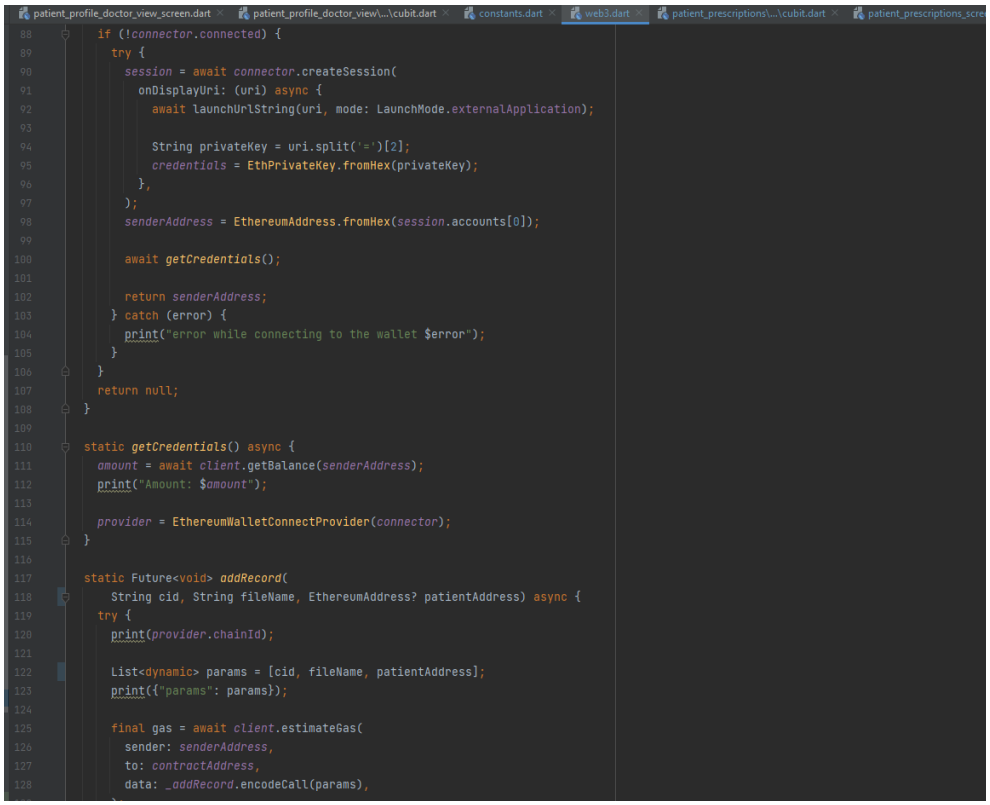


```dart
import 'dart:typed_data';

import 'package:dio/dio.dart';

import '../../constants/constants.dart';

class Web3DioHelper {
  static Dio? dio;

  static init() {
    dio = Dio(
      BaseOptions(
        baseUrl: 'https://',
        receiveDataWhenStatusError: true,
      ),  // BaseOptions
    );  // Dio
  }

  static Future<Response> postData({
    final String url = "api.web3.storage/upload",
    required Uint8List data,
  }) async {
    dio!.options.headers = {
      'accept': 'application/json',
      'Authorization': "Bearer $web3StorageToken",
      'Content-Type': '*/*',
    };
    return await dio!.post(url, data: data);
  }

  static Future<Response> getData({
    final String path = 'ipfs.w3s.link',
    required String param,
  }) async {
    dio!.options.headers = {
      'accept': 'application/json',
      'Authorization': "Bearer $web3StorageToken",
    };
    return await dio!.get("{$param}.{$path}");
  }
}
```

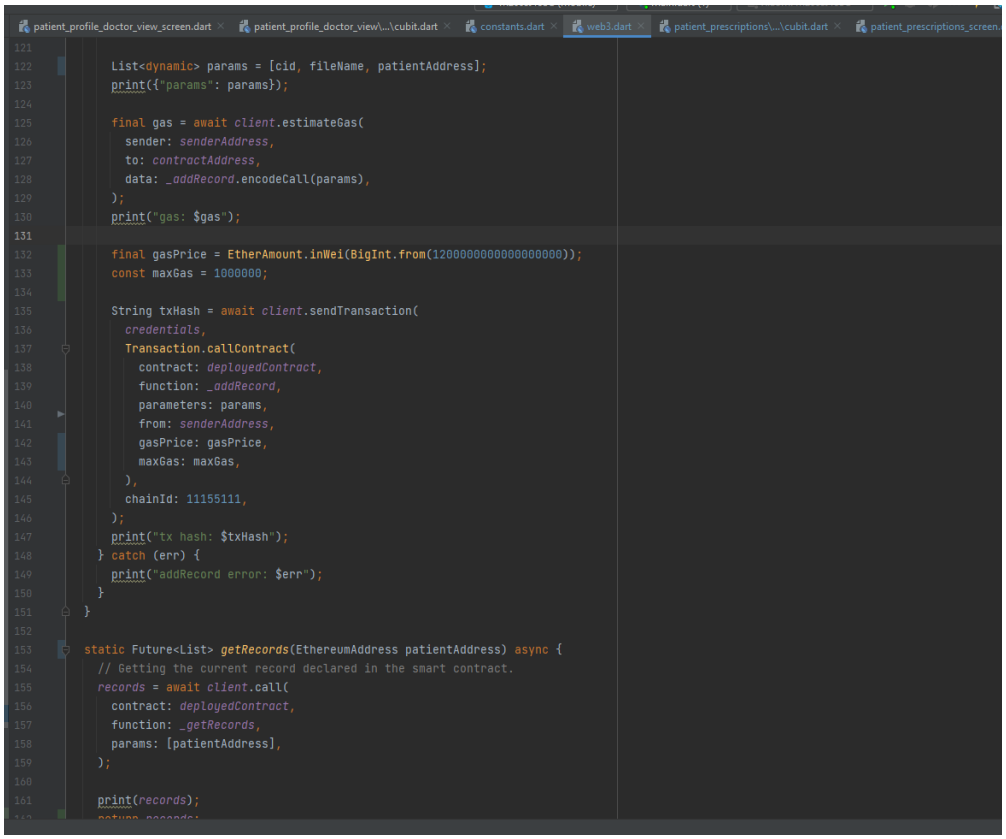
Web3 connection





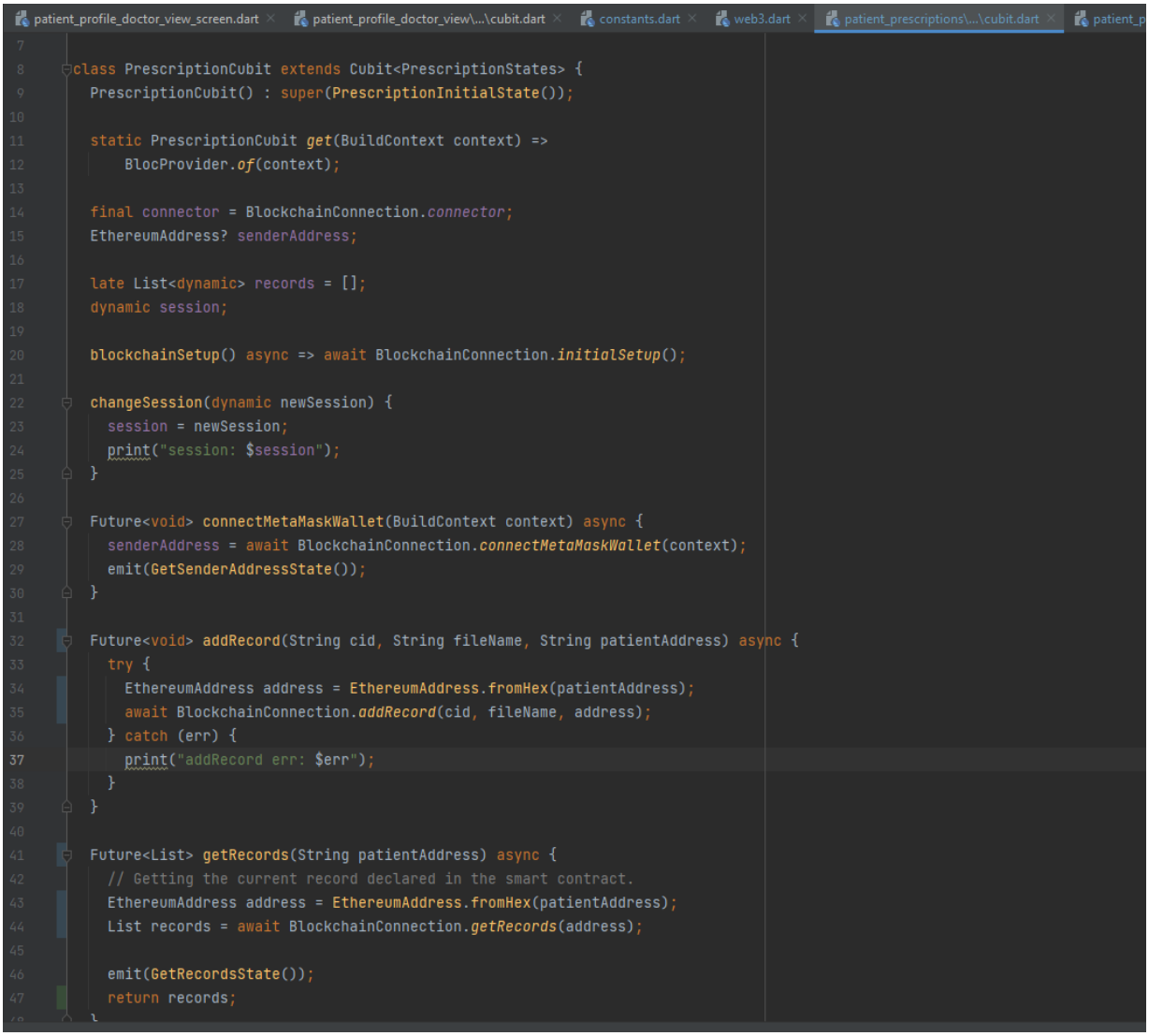


```dart
if (!connector.connected) {
  try {
    session = await connector.createSession(
      onDisplayUri: (uri) async {
        await launchUrlString(uri, mode: LaunchMode.externalApplication);

        String privateKey = uri.split('=')[2];
        credentials = EthPrivateKey.fromHex(privateKey);
      },
    );
    senderAddress = EthereumAddress.fromHex(session.accounts[0]);

    await getCredentials();

    return senderAddress;
  } catch (error) {
    print("error while connecting to the wallet $error");
  }
}
return null;
}

static getCredentials() async {
  amount = await client.getBalance(senderAddress);
  print("Amount: $amount");

  provider = EthereumWalletConnectProvider(connector);
}

static Future<void> addRecord(
    String cid, String fileName, EthereumAddress? patientAddress) async {
  try {
    print(provider.chainId);

    List<dynamic> params = [cid, fileName, patientAddress];
    print({"params": params});

    final gas = await client.estimateGas(
      sender: senderAddress,
      to: contractAddress,
      data: _addRecord.encodeCall(params),
```



```dart
    List<dynamic> params = [cid, fileName, patientAddress];
    print({"params": params});

    final gas = await client.estimateGas(
      sender: senderAddress,
      to: contractAddress,
      data: _addRecord.encodeCall(params),
    );
    print("gas: $gas");

    final gasPrice = EtherAmount.inWei(BigInt.from(120000000000000000));
    const maxGas = 1000000;

    String txHash = await client.sendTransaction(
      credentials,
      Transaction.callContract(
        contract: deployedContract,
        function: _addRecord,
        parameters: params,
        from: senderAddress,
        gasPrice: gasPrice,
        maxGas: maxGas,
      ),
      chainId: 11155111,
    );
    print("tx hash: $txHash");
  } catch (err) {
    print("addRecord error: $err");
  }
}

static Future<List> getRecords(EthereumAddress patientAddress) async {
  // Getting the current record declared in the smart contract.
  records = await client.call(
    contract: deployedContract,
    function: _getRecords,
    params: [patientAddress],
  );

  print(records);
```

Cubit that connect blockchain with UI

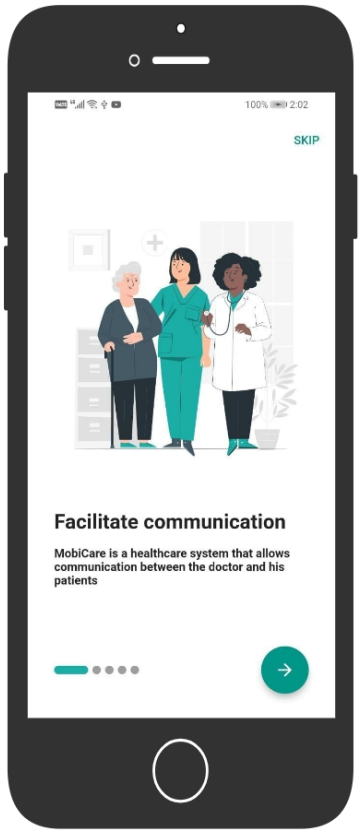



```dart
class PrescriptionCubit extends Cubit<PrescriptionStates> {
  PrescriptionCubit() : super(PrescriptionInitialState());

  static PrescriptionCubit get(BuildContext context) =>
      BlocProvider.of(context);

  final connector = BlockchainConnection.connector;
  EthereumAddress? senderAddress;

  late List<dynamic> records = [];
  dynamic session;

  blockchainSetup() async => await BlockchainConnection.initialSetup();

  changeSession(dynamic newSession) {
    session = newSession;
    print("session: $session");
  }

  Future<void> connectMetaMaskWallet(BuildContext context) async {
    senderAddress = await BlockchainConnection.connectMetaMaskWallet(context);
    emit(GetSenderAddressState());
  }

  Future<void> addRecord(String cid, String fileName, String patientAddress) async {
    try {
      EthereumAddress address = EthereumAddress.fromHex(patientAddress);
      await BlockchainConnection.addRecord(cid, fileName, address);
    } catch (err) {
      print("addRecord err: $err");
    }
  }

  Future<List> getRecords(String patientAddress) async {
    // Getting the current record declared in the smart contract.
    EthereumAddress address = EthereumAddress.fromHex(patientAddress);
    List records = await BlockchainConnection.getRecords(address);

    emit(GetRecordsState());
    return records;
  }
```

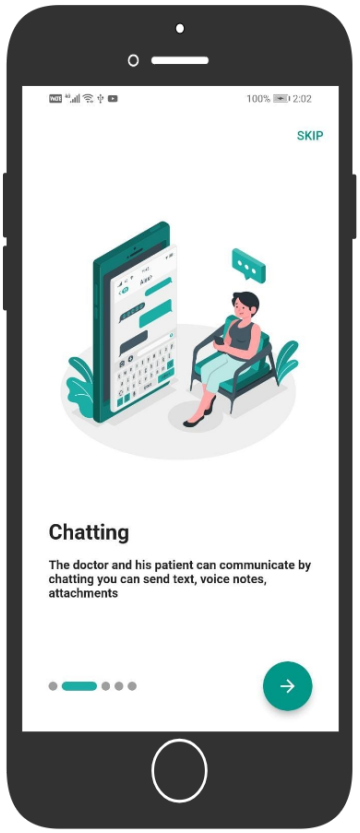
Mobile Screens:

Splash Screen

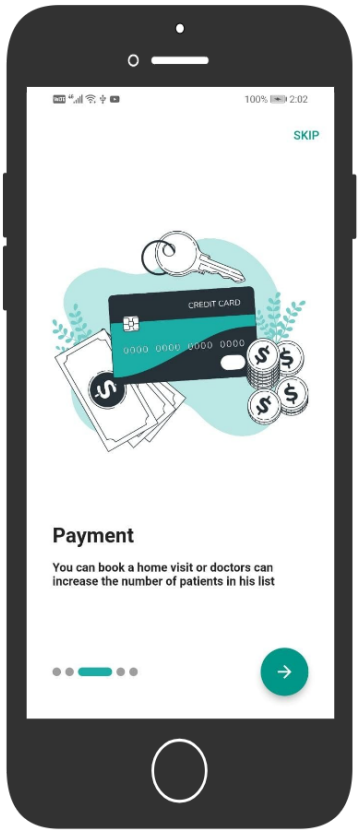
On boarding Screen - 1



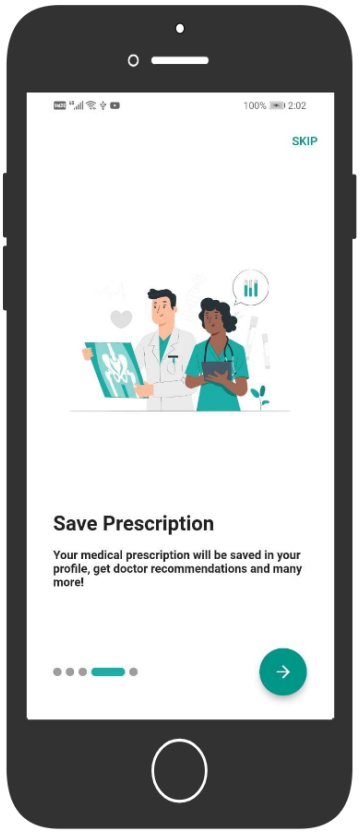
On boarding Screen - 2



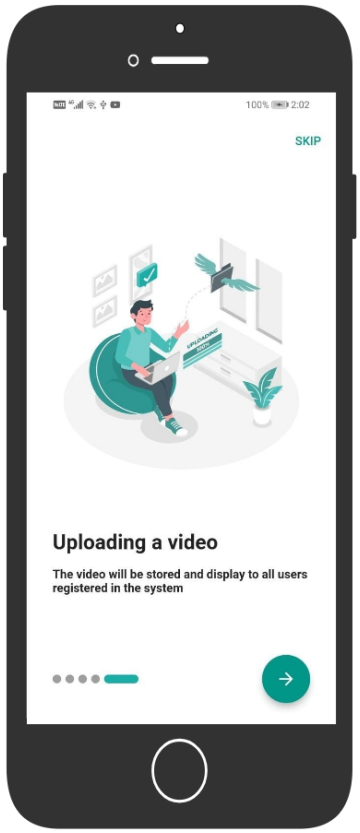
On boarding Screen - 3



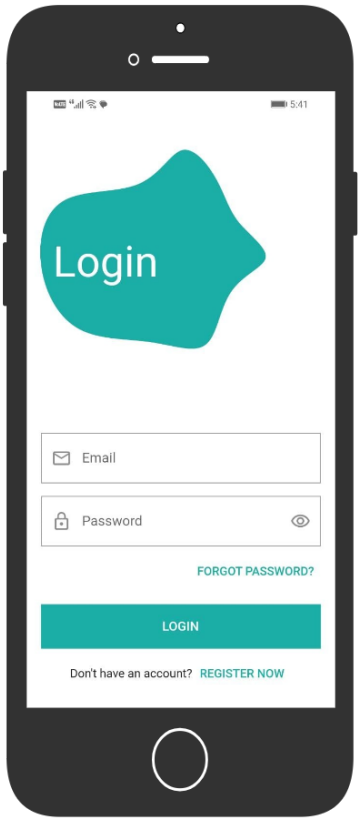
On boarding Screen - 4



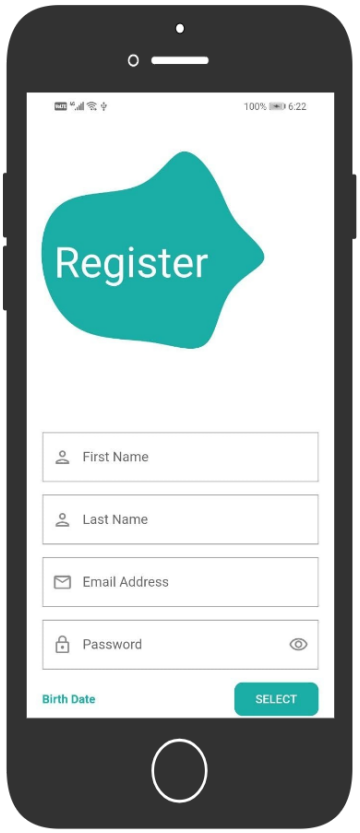
On boarding Screen - 5



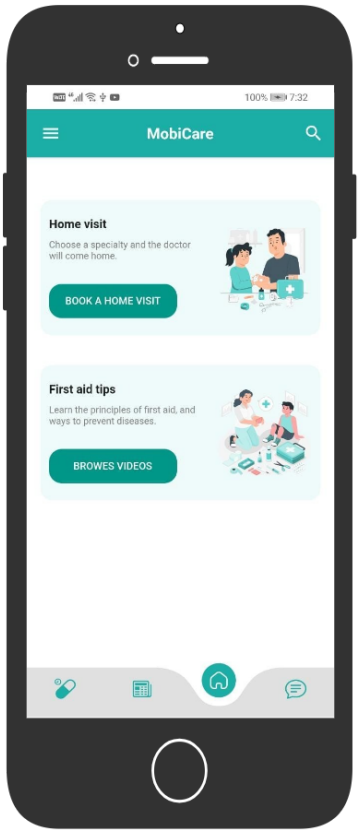
Login Screen



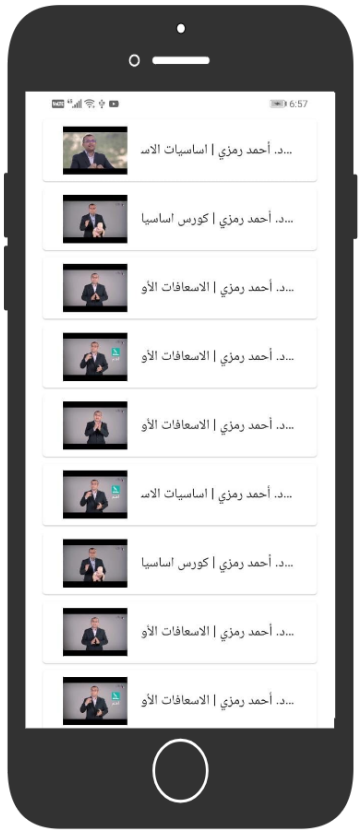
Register Screen



Home Screen - Patient

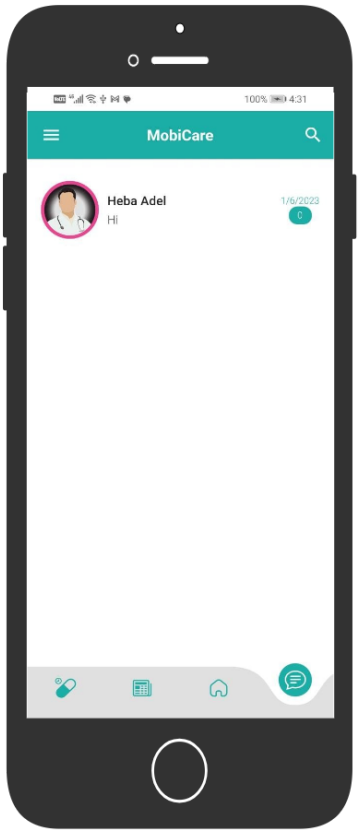


First aid tips - Patient

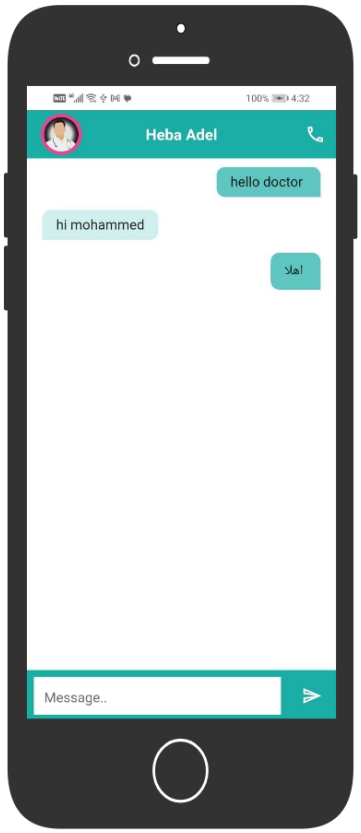


Browse first add tip Screen - Patient

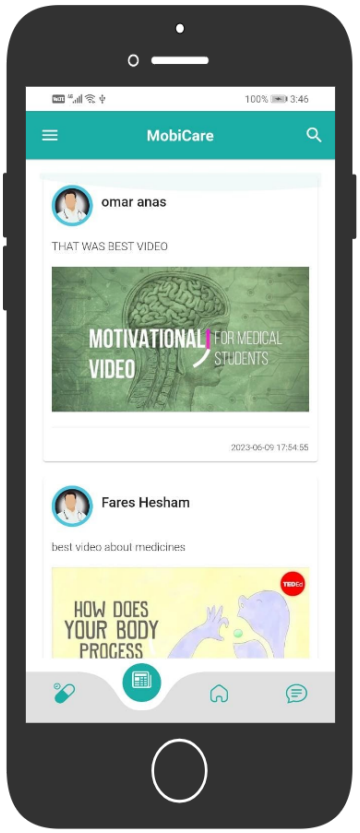
Chat Screen - Patient



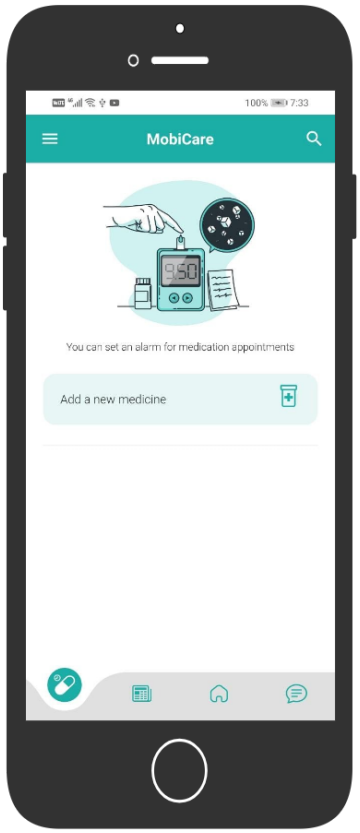
Inner Chat Screen



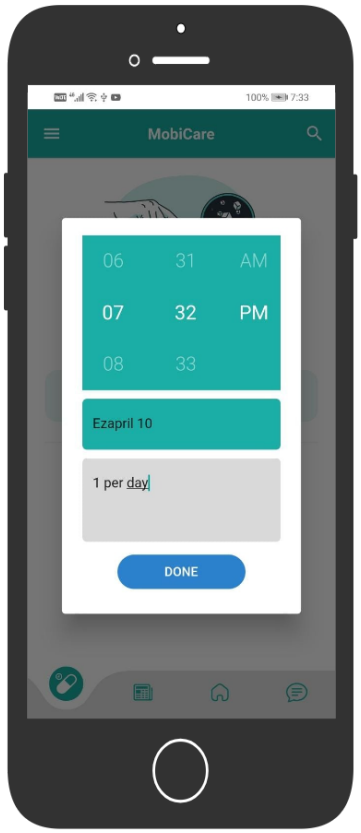
Videos Screen - Patient



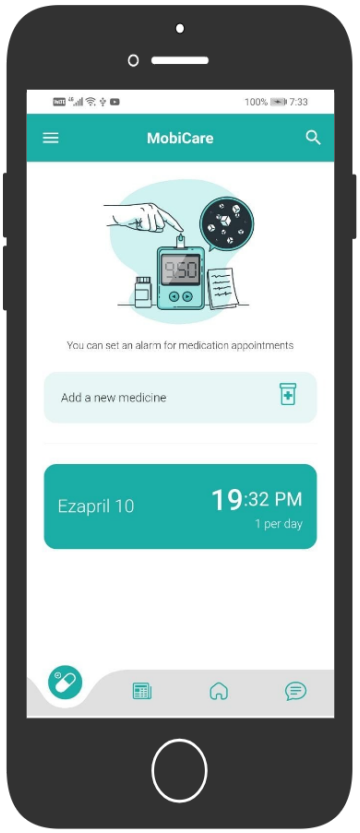
Empty midecation schedule - Patient



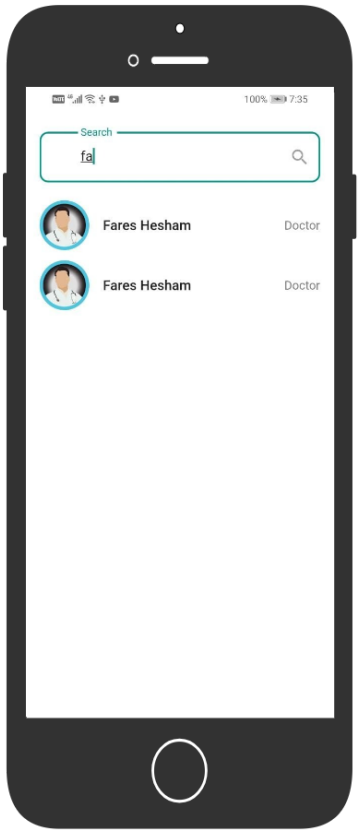
Add medication pop up - Patient



Added medication schedule - Patient

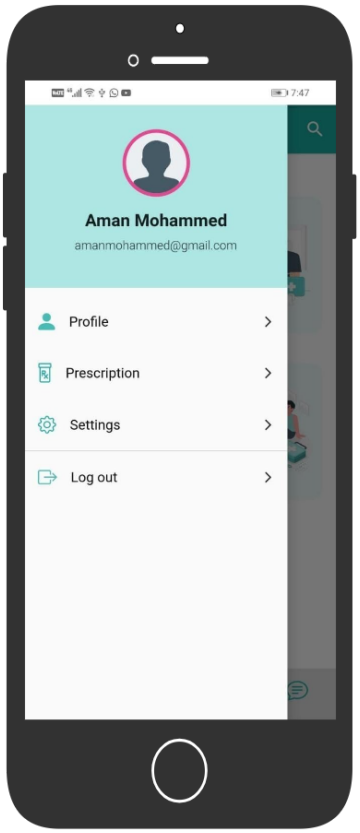


Search about doctor with result - Patient

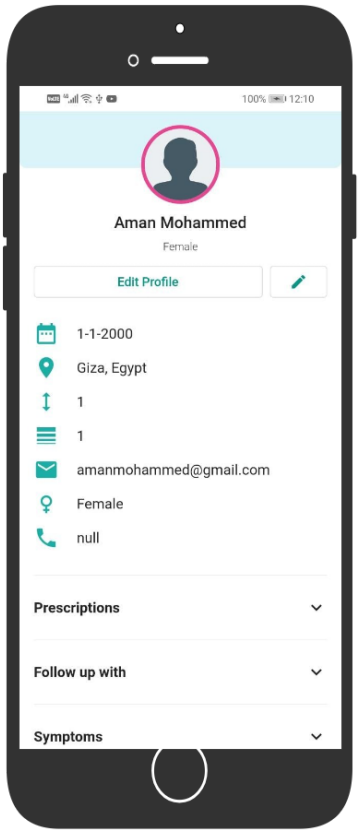


Search about doctor with no result - Patient

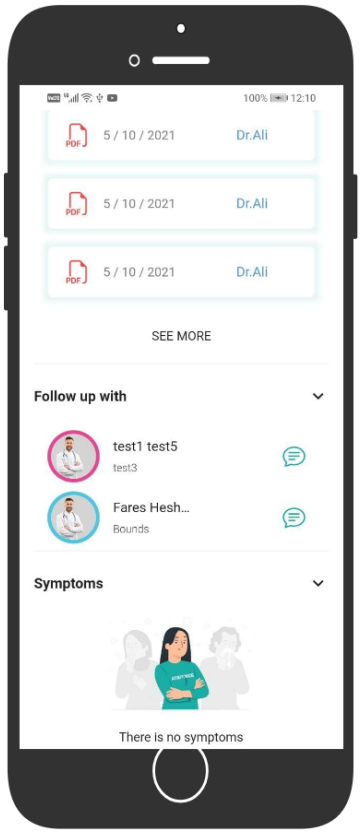
Drawer Screen - Patinet



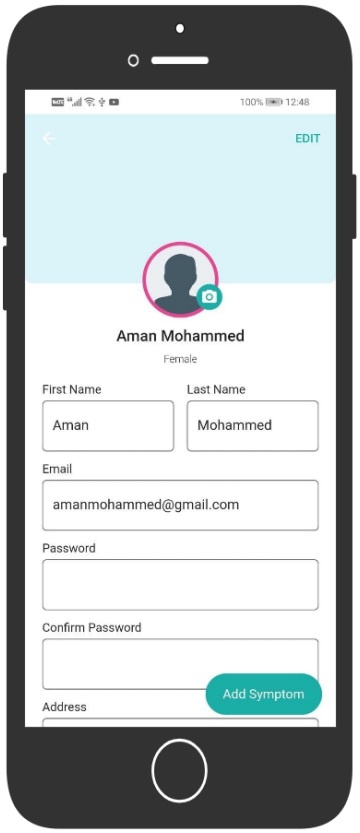
Patient Profile - Patient



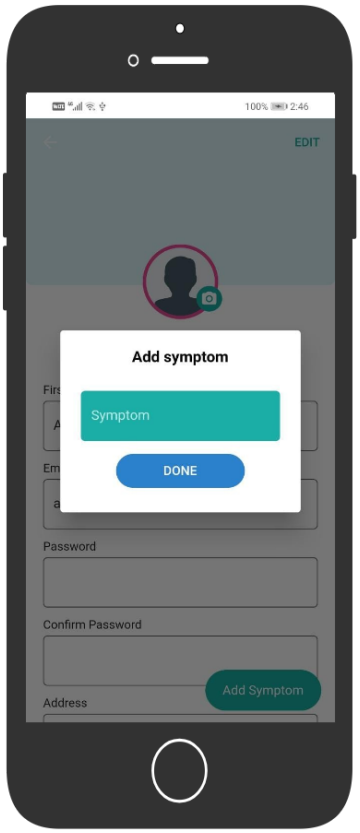
Profile Screen - Patinet



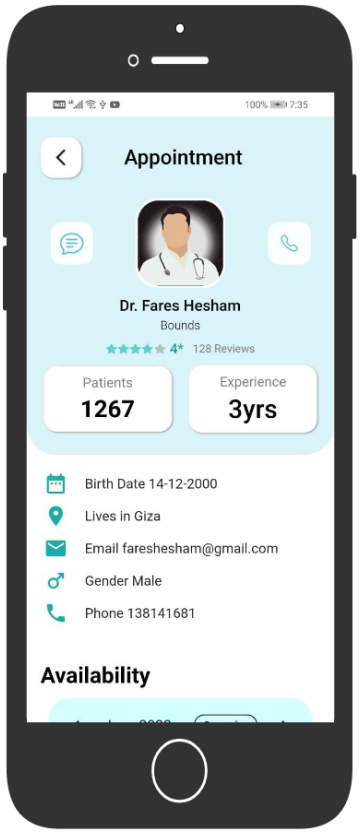
Edit profile - Patient



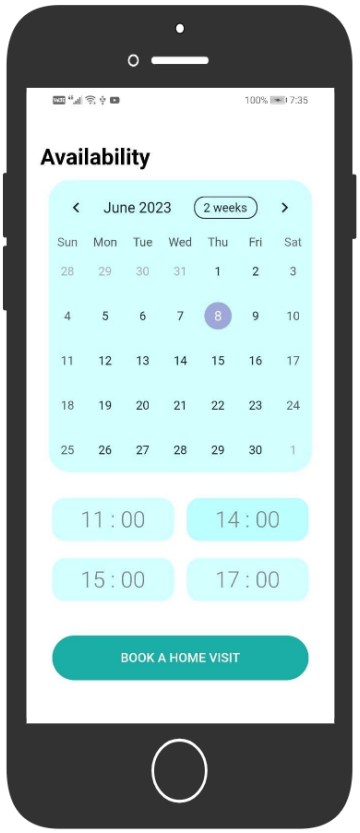
Add Symptom - Patinet



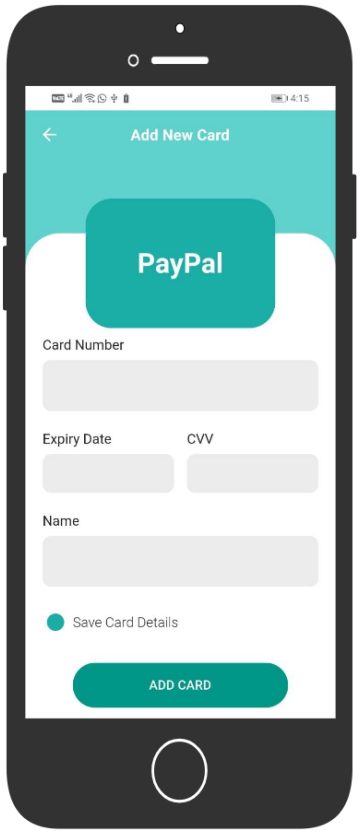
Doctor profile - Patient



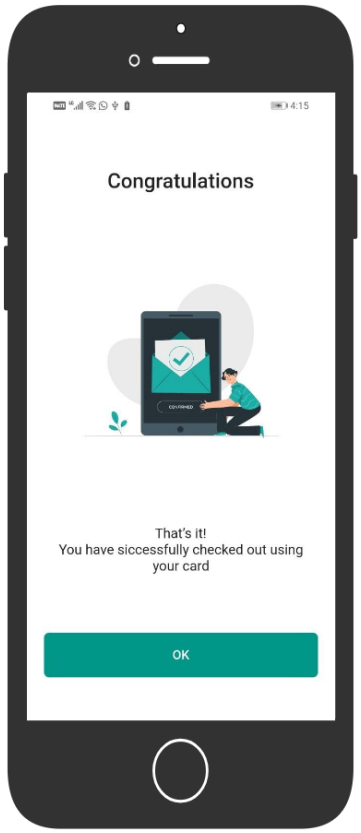
Availability Screen



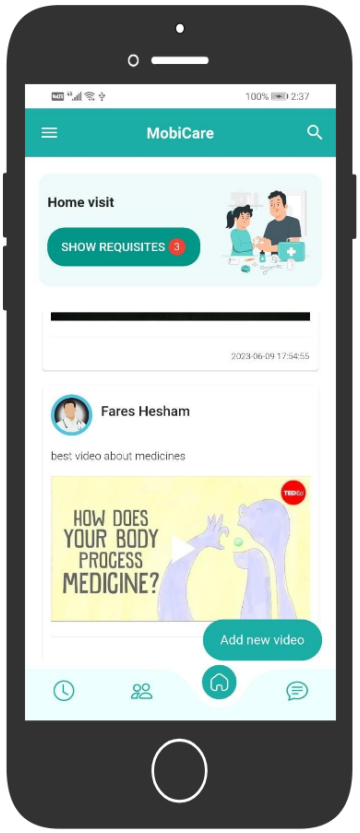
Payment Screen



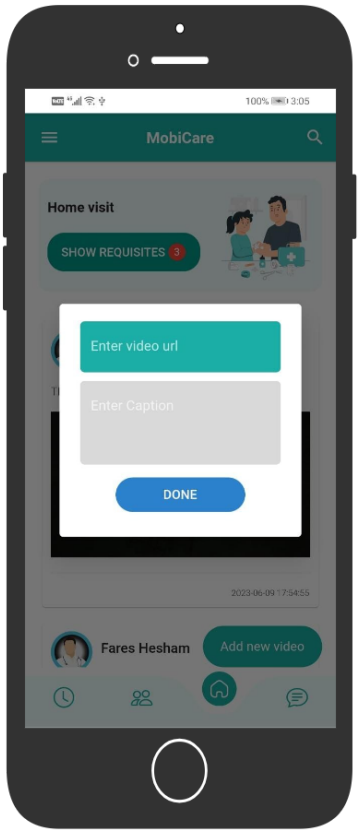
Payment done Screen



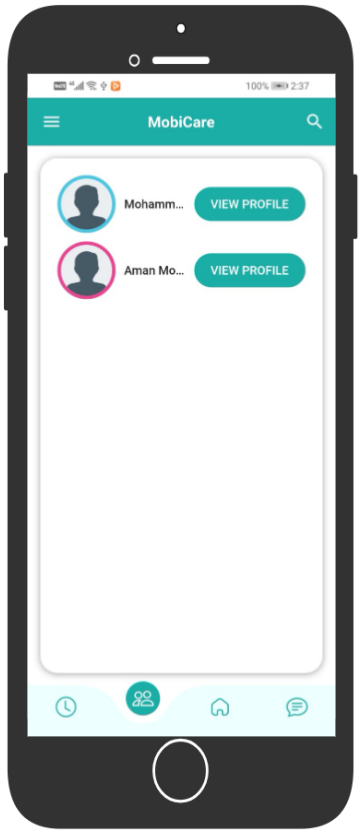
Home Screen - Doctor



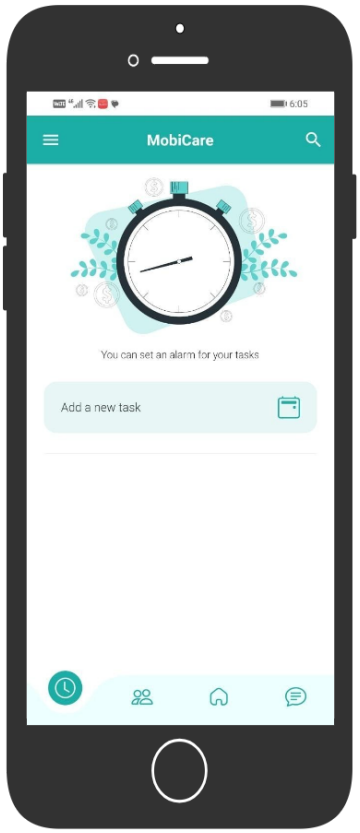
Add video pop up - Doctor



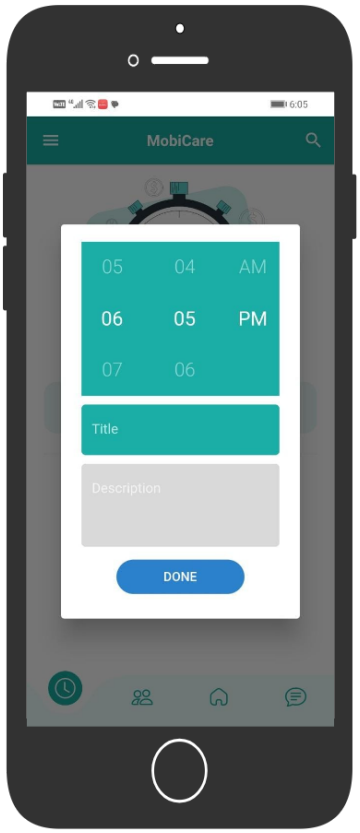
Patinet list - Doctor



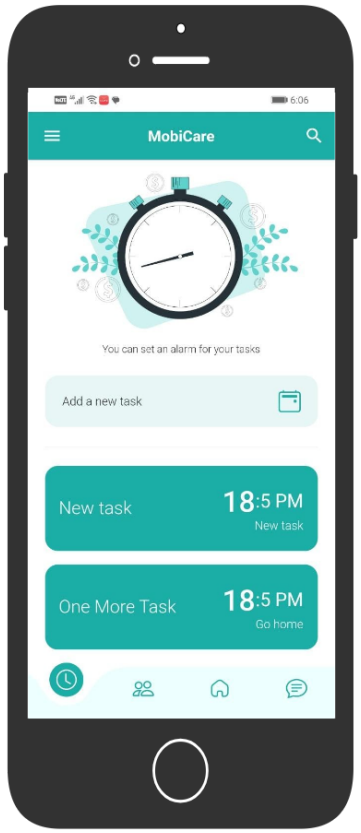
Empty task schedule - Doctor



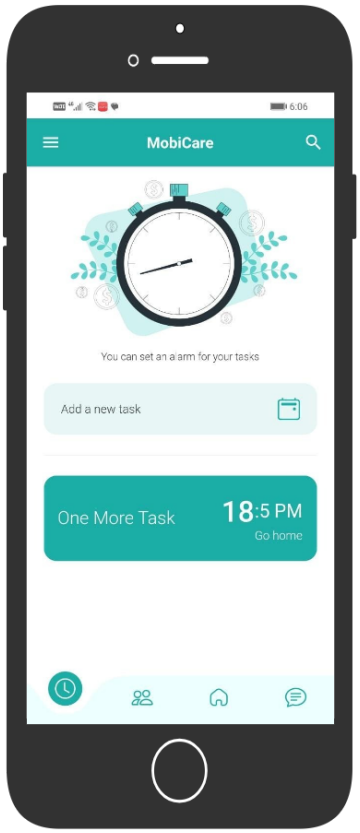
Add task schedule pop up - Doctor



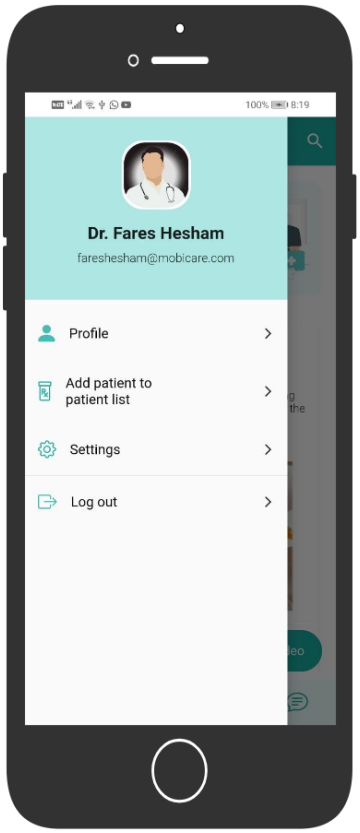
Task added - Doctor



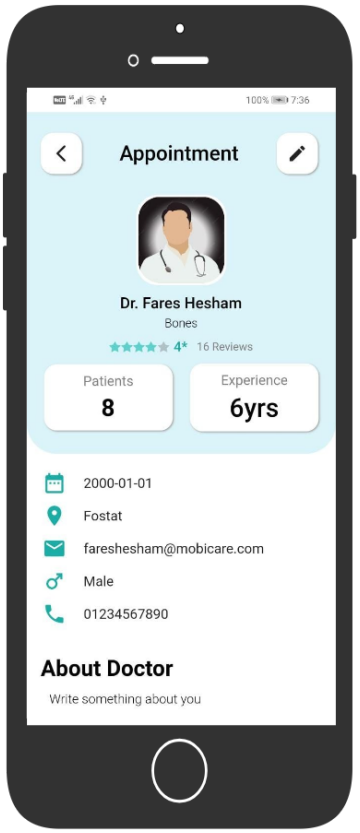
Task deleted when move - Doctor



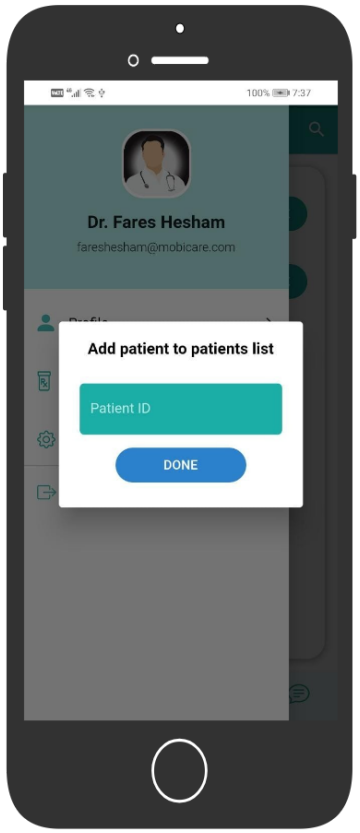
Drawer Screen - Doctor



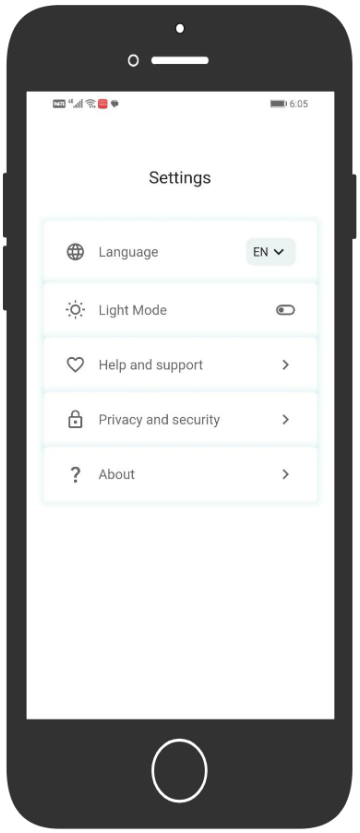
Doctor Profile - Doctor



Add patient to doctor list pop up - Doctor

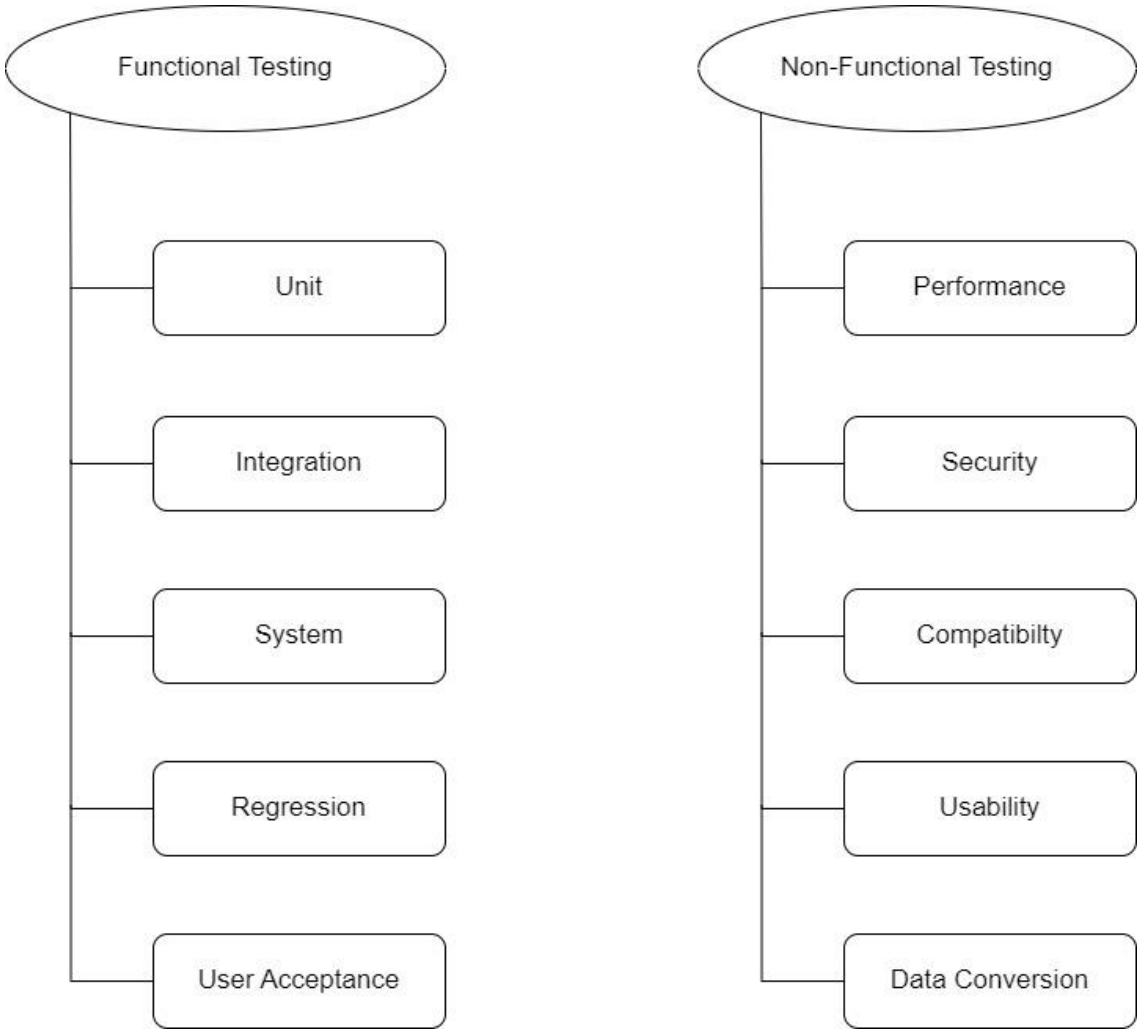


Setting Screen



Chapter 5: Testing

In this chapter, we are going to discuss and go deeper into MobiCare health system testing and present the types of testing to be used and the test cases we examined our application through.



Functional Testing:

Unit Testing

Testing of individual items (e.g., modules, programs, objects, classes, etc.) Usually as part of the coding phase, in isolation from other development items and the system as a whole.

Integration testing

Testing the interfaces between major (e.g., systems-level application modules) and minor (e.g., individual programs or components) items within an application that must interact.

Additional Testing

System Testing

Testing a system behavior as a whole when development is finished, and the system can be tested as a complete entity.

Regression Testing

To check older functionality after integrating new functionality.

Acceptance testing

Testing to ensure that a development is ready to be deployed into the business, operational, or production environment.

Non-Functional Testing:

Performance Testing

Accomplished a designated function regarding processing time and throughput rate.

Load Testing

Measuring the behavior of within creasing load which the component or system can handle.

Stress Testing

Evaluate a system or component at or beyond the limits of its specified requirements.

Security Testing

How well the system protects against unauthorized internal or external access.

Test cases:

Test scenario objective:

Verify that on passing valid values a user should get registered on the application.

Assumptions/dependencies:

Email, password, full name, age, gender and address.

Test ID	Prerequisites	Step no.	Description	Expected result	Actual result
TC_1	User has no account on the app	1	Verify that the unregistered user can navigate to the registration screen	User navigates to the registration screen	As expected
		2	Verify that user can write his first name in the “First name” text field	User can enter his first name In the “First name” text field	As expected
		3	Verify that user can enter his last name in the “Last name” text field	User can enter his last name In the “Last name” text field	As expected
		4	Verify that user can enter his email in the “Email” text field	User can enter his Email in the “Email” text field	As expected
		5	Verify that user can enter his password in the “Password” text field	User can enter his password in the “Password” text field	As expected
		6	Verify that the	The icon is	As expected

Test ID	Prerequisites	Step no.	Description	Expected result	Actual result
			password visibility toggle icon works	interactive and and toggles the visibility of the password	
		7	Verify that user can enter his address in the “Address” text field	User can enter his address the “Address” text field	As expected
		8	Verify that the “Date of birth” picker works	User can pick his date of birth	As expected
		9	Verify that “Gender” radio button works	User can select his gender	As expected
		10	Verify that the “Register” button works	User can press the “Register” button	As expected
		11	Verify that the user is successfully registered on the application	User is now registered on the application	Passed

Test scenario objective:

User login with the correct email address and password.

Assumptions/dependencies:

Correct email

Correct password

Test ID	Prerequisites	Step no.	Description	Expected result	Actual result
TC_2	User has an account on the app	1	Verify that the signed up user can navigate to the login screen	User navigates to the login screen	As expected
		2	Verify that user can enter his email in the “Email” text field	User can enter his email in the “Email” text field	As expected
		3	Verify that user can enter his password in the “Password” text field	User can enter his password in the “Password” text field	As expected
		4	Verify that the password visibility toggle works	The icon is interactive and and toggles the visibility of the password	As expected
		5	Verify that the user can press the “login” button	User presses the “login” button	As expected
		6	Validate the user’s data	User’s data are valid	As expected
		7	Verify that the user has successfully	User is successfully logged in his	Passed

			logged in to his account	account	

Test scenario objective:

Adding new patients by their id to the doctor's patient list.

Assumptions/dependencies:

Know the patient's id.

Test ID	Prerequisites	Step no.	Description	Expected result	Actual result
TC_3	Patient's list has a space and the doctor knows the patients ID	1	Verifying that the doctor can enter his profile through the drawer	Doctor presses profile button from drawer and enters his profile	As expected
		2	Verifying that the doctor can find his patient list in his profile	Doctor scrolls to his patient list in his profile	As expected
		3	Verifying that the doctor can open and view his patient list	Doctor can open and view his patient list	As expected
		4	Verifying that the "Add patient" button is working	Doctor presses on the "Add patient" button and he can add patients	As expected

		5	Verifying that the doctor can add patients by their ID	Doctor enters the patient ID to add them	As expected
		6	Verifying that the patient is added to the doctor's patient list	Patient is successfully added to the doctor's patient list	Passed

Test scenario objective:

Patient wants to view his stored medical prescriptions.

Assumptions/dependencies:

Connected to MetaMask wallet.

Test ID	Prerequisites	Step no.	Description	Expected result	Actual result
TC_4	Patient has stored prescriptions and a wallet	1	Verifying that patient can navigate through drawer to view his prescriptions	Patient selects prescription from drawer	As expected
		2	Verifying that medical prescriptions are stored with date	Every medical prescription is stored on the app with their date	As expected

		3	Verifying that the “connect wallet” button is working	Patient can press the “connect wallet” button	As expected
		4	Verifying that the patient is connected to his wallet	Patient is connected to his wallet successfully	As expected
		5	Verifying that the patient can view his stored prescriptions	Patient can view and open his prescriptions	Passed

Test scenario objective:

Patient making a home visit request

Assumptions/dependencies:

Enter data from his credit card for payment.

Test ID	Prerequisites	Step no.	Description	Expected result	Actual result
TC_5	Patient is logged in and has enough money for booking	1	Verifying that the search bar works	Patient can use the search bar to search for a doctor by his username	As expected

		2	Verifying that the patient can view doctor's profile	Patient presses on the doctor's username and navigates to his profile	As expected
		3	Verifying that the "Book a home visit" button works and sends a request to the doctor	Patient presses on the "Home visit" button and a request is sent to the doctor	Passed

Test scenario objective:

Doctor adding medical prescription to a patient.

Assumptions/dependencies:

Connected to MetaMask wallet

Get Patient address on MetaMask

Test ID	Prerequisites	Step no.	Description	Expected result	Actual result
TC_6	Patient had a previous appointment with the doctor	1	Verifying that the doctor can check his patient list from the bottom navigation bar	Doctor goes to his patient list via navigation bar	As expected
		2	Verify that doctor can select a patient	Doctor can select any patient from	As expected

			from his patient list	his patient list	
		3	Verify that the “Add medical prescription” button works	Doctor presses the “Add medical prescription” button and he can add the prescription	As expected
		4	Verify that the Doctor can import the medical prescription to the Patient	Doctor can import the prescription	As expected
		5	Verify that the prescription is added with date and stored on the application	The medical prescription is added with date	Passed

Chapter 6: Results and Discussion

In this chapter, we are going to talk about the results of our project, whether they have been achieved or not, and also the differences between the desired result and the actual one.

Results

Expected result

- The patient's medical records are saved securely in one place.
- The patient has easy access to his medical records including previous consultations, and his prescriptions with the date and time they did them.
- The patient gives access to his data to the doctors and removes it whenever he wants.
- The patient gives access to the doctor using Card authentication devices.
- The patient can easily chat with the doctor and vice versa.
- The patient can add a medication schedule to stick with his medication.
- Doctors have easy access to the patient's health records used in decision making related to the patient's care.
- Doctors can view their patients and vice versa.
- Doctors can write medical records or upload them to the IPFS and access it using blockchain technology.
- Doctor can receive Home visit requests for patients, and he can approve or reject them.
- Doctors can upload videos on their homepage and all users can interact with it.
- Lap stuff gets the required tests from the patient account and loud it after releasing it.

Actual result

- The patient's medical records are saved securely in one place.
- The patient has easy access to his medical records including previous consultations, and his prescriptions with the date and time they did them.
- The patient gives access to his data to the doctors and removes it whenever he wants.
- The patient can give access to another doctor to view his records.
- The patient can easily chat with the doctor and vice versa.
- The patient can add a medication schedule to stick with his medication.
- Doctors have easy access to the patient's health records used in decision making related to the patient's care.

- Doctors can view their patients and vice versa.
- Doctors can write medical records or upload them to the IPFS and access it using blockchain technology.
- Doctors can upload videos on their homepage and all users can interact with it.
- Lap stuff gets the required tests from the patient account and loud it after releasing it.

Discussion

As a conclusion for the previous points, we have managed to meet most of the expectations we planned for except for some points such as Home visit requests and building an authentication method using Wi-Fi cards devices, instead we made the authentication method using the wallet address and the private key.

Due to shortage of time, we canceled the feature that made the doctors have access to Home visit requests, but we managed this by making a homepage to doctors that allow them to make videos and patients could react with these videos.

We managed making any patient give access to another doctor to view his records.

Chapter 7: Conclusion

In this chapter, we sum up the whole thing and summarize the things mentioned before.

Conclusion

Now that we have come a really long way after our journey has finally ended at FCAI-HU, working at this project and seeing it grow month after month, day after day, surely we had some ups and downs regarding the limited resources and narrow time but we did our best to make it happen and it was a worthy challenge, as you see in this report we made sure to have every detail written down to show how the app was successfully created, we listed every technology, aspect, and structure we used, we hope it will help younger colleagues one day to create something like it as it made us proud and we gained great experience from it that we will be needing in the future, “MobiCare” is not just an idea anymore it is an actual thing and we did it, not only to thank our professors and teacher assistant but also to help in improving healthcare in Egypt.

We are hoping that “MobiCare” will play a huge role in our society, we are looking forward to enhancing it in the future with better ideas and resources if available to facilitate the connection between doctors and patients even more and adding new features like first aid tips, personalized feed and emergency contacts.

Last but not least, we have faith that our project and the hard work that was put into it will set a good example for others, to help and encourage them to try to make a good difference in society as every tiny step counts.

Chapter 8: Future Work

In this chapter, we will propose what are the future plans we have concerning our application as it will not end by the end of the discussion.

MobiCare in Future Work

Here is our plan for the coming versions of MobiCare:

1. The doctor and his patients can join a video call at any time.
2. System will detect prescriptions through the OCR machine learning model to know the name of the medicine written.
3. System will recommend the nearest pharmacy to the patient to buy the medicine.
4. IoT device that will facilitate the transmission of medical records by simply bringing the phone close to the device's sensor.
5. IoT devices will measure the pulse, blood pressure, and temperature to monitor the patient's medical condition.
6. The patient's data will be present in all hospitals so that when any accident occurs, the hospital will recognize the patient's medical condition and the medications he was taking, and assist him in the correct way.

References:

Name	Resources
Diagrams	draw.io
Flutter	https://www.udemy.com/course/complete-flutter-arabic/
BlockChain	https://drive.google.com/drive/folders/18Z_tbwgFwOxSftY9ovqlDlQml86LG4rW
Node.JS	https://nodejs.org/en/docs

Project Links :

Name	Link
UX Design	https://www.figma.com/file/PK3q00grd4mBxQ2jH04yJB/MobiCare?type=design&node-id=0-1&t=A9c2HnEGoDAEEZuf-0
Mobile	https://github.com/HebaAdelAhmed/MobiCare
Server	https://github.com/omar-anas/GP_BE_ROSHETA
BlockChain	https://sepolia.etherscan.io/address/0x43b4a2d7fe18f7f7d9b23d187169442325bdd772