

**SCHOOL OF COMPUTING AND INFORMATICS**

**DEPARTMENT OF COMPUTER SCIENCE**

**CCS 323 : GROUP PROJECT**

**TITLE : MEDICARE**

|  |  |
| --- | --- |
| **NAME** | **REGNO** |
| WANDABI GIDEON | CCS/00009/020 |
| LLOYD KATILA | CCS/00008/020 |
| EDWIN ONYANGO | CCS/00032/020 |

A project report submitted in partial fulfillment of the requirement of the Bachelor of Science in Computer science.

# **DECLARATION**

We the undersigned, do hereby declare that this project is our original work and to the best of our knowledge it has not been presented to any other examination body.

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **REGNO** | **SIGNATURE** | **DATE** |
| WANDABI GIDEON SIMIYU | CCS/00009/020 |  |  |
| LLOYD TONY KATILA | CCS/00008/020 |  |  |
| EDWIN ONYANGO | CCS/00032/020 |  |  |

The project report is hereby presented for examination with the approval of the project supervisor:

NAME : MICHAEL ADONGO

SIGNATURE : ....................................

DATE :.....................................

**ABSTRACT**

The Medicare project explores the relevance of management systems to the contemporary society where computers have made tasks easier, more efficient and organized. Medicare system is a planned system of the storage processing and dissemination of data around the hospital in the form of information needed to carry out various functions which come to simplify the tiresome manual way of physically having to visit the hospital, reporting and writing the details on papers. There is need for an automated doctors’ appointment management and emergency handling system that can be used to store all records in a centralized database which should replace the manual system that is currently being used. The manual system has few challenges that include the ever-increasing paper load, difficulty in enforcing access control as well as cases of missing files and information.

The scope of the project cuts across administrators and 2 types of clients; hospitals and other patients.

The research methodology used in this project was System development life-cycle, using waterfall method simply because the project was small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. Use-case diagrams and activity diagrams also helped in coming up with the database class diagram which we used to develop the database.

# **ACKNOWLEDGEMENT**

We take this opportunity to express our gratitude to our project supervisor Mr. Adongo for his even willingness to give us valuable advice and direction under which we executed this project. His constant guidance and willingness to share his knowledge made us understand this project and its manifestations in great depths and helped

**LIST OF ABBREVIATIONS**

1. SDLC
   * System Development Lifecycle
2. UI 
   * User Interface.
   * The system interface that the users interact with.
3. JSON
   * JavaScript Object Notation
4. REST 
   * Representational State Transfer
5. API 
   * Application Programming Interface
6. BG
   * Blood Group

Contents

[**DECLARATION** I](#_Toc124182930)

[**ABSTRACT** II](#_Toc124182931)

[**ACKNOWLEDGEMENT** III](#_Toc124182932)

[**LIST OF ABBREVIATIONS** IV](#_Toc124182933)

[**CHAPTER 1: INTRODUCTION** 1](#_Toc124182934)

[**1.1: Introduction** 1](#_Toc124182935)

[**1.2: Background Study** 1](#_Toc124182936)

[**1.4: Statement of the problem** 1](#_Toc124182937)

[**1.4: Objectives** 2](#_Toc124182938)

[**1.4.1: General Objective** 2](#_Toc124182939)

[**1.4.2: Specific objectives** 2](#_Toc124182940)

[**1.5: Justification** 3](#_Toc124182941)

[**CHAPTER 2: SYSTEM ANALYSIS** 4](#_Toc124182942)

[**2.1: Introduction** 4](#_Toc124182943)

[**2.2: System development methodology** 4](#_Toc124182944)

[**2.3: Feasibility study** 5](#_Toc124182945)

[**2.3.1: Technical Feasibility** 5](#_Toc124182946)

[**2.3.2: Economic Feasibility** 6](#_Toc124182947)

[**2.3.3: Operational Feasibility** 6](#_Toc124182948)

[**2.3.4: Legal and Regulatory Feasibility** 6](#_Toc124182949)

[**2.3.5: Scheduling Feasibility** 6](#_Toc124182950)

[**2.4: Requirements Elicitation** 7](#_Toc124182951)

[**2.5: Data and System Analysis** 8](#_Toc124182952)

[**2.6: System Specifications** 10](#_Toc124182953)

[**2.6.1: Hardware Specifications** 10](#_Toc124182954)

[**2.6.2: Software Specifications** 10](#_Toc124182955)

[**CHAPTER 3: SYSTEM DESIGN** 12](#_Toc124182956)

[**3.1: Introduction** 12](#_Toc124182957)

[**3.2: System Users** 12](#_Toc124182958)

[**3.2.1: Admin User** 12](#_Toc124182959)

[**3.2.2: Staff User** 12](#_Toc124182960)

[**3.2.3: End User** 13](#_Toc124182961)

[**3.3.1: User Interfaces** 13](#_Toc124182962)

[**1.** **Login** 13](#_Toc124182963)

[**3.3.2: Admin interfaces** 18](#_Toc124182964)

[**1 . Login** 18](#_Toc124182965)

[**2.Views** 19](#_Toc124182966)

[**3.4: Database Schema** 23](#_Toc124182967)

[**3.5: Data Flow Diagrams** 25](#_Toc124182968)

[**3.4.1: Context Level Diagram** 25](#_Toc124182969)

[**3.4.2: DFD Level – 1** 26](#_Toc124182970)

[**3.4.3: DFD Level – 2** 27](#_Toc124182971)

[**3.4: Use Case Diagram** 33](#_Toc124182972)

[**3.5: Class Diagram** 34](#_Toc124182973)

[**CHAPTER 4: SUMMARY AND CONCLUSION** 38](#_Toc124182974)

[**CHAPTER 5: REFERENCE** 39](#_Toc124182975)

# **1: INTRODUCTION**

## **1.1: Introduction**

Healthcare is one of the basic human needs that should be readily affordable to every person in an efficient way but most of the time people visiting health center do not receive a standard service. The problem is mainly attributed to the long waiting hours and poor communication between the patient and health providers.

Our project aims to solve these two main problems by providing a system that gives a better way of getting doctor appointments and communicating with health provides to acquire medical advice efficiently.

## **1.2: Background Study**

In the technological advancement bid, computers have greatly contributed to the simplification of the way things are done and in return have contributed to the automation of the various activities that take place in and around our day-to-day activities. According to the way technology has been advancing and evolving, the use of computer and internet technology is slowly replacing some aspects of tedious paper work which can be implemented by using an online automated system. Now that many systems have been automated in Kenya, the health sector should not be left behind. Many systems so far have been developed in relation to medical issues. Software such Microsoft excel are being used to store medical records, but no system has been made to automate the record of medical appointments made by patients and emergency handling. The overall objective of this proposal was to develop an application that is sufficient to automate the medical appointment and emergency handling system.

## **1.4: Statement of the problem**

This country needs a platform that covers automation of health record management system country wide that can be used to handle medical appointments and store health records in a centralized place. The manual system that is currently used is getting challenged by few problems it has. The current system poses problems that include the poor emergency handling methods, ever-increasing paper load, difficulty in enforcing access control as well as cases of missing files and information.

Currently all operations for managing day to day activities in medical department are either not fully or computerized not computerized at all. These results in the following problems:

1. Missing files containing health reports. This result in inconsistency of the patient records.
2. Health fatalities due to poor handling of emergencies and mismanaged hospital wards.
3. Time wastage in hospitals since there is no clear system to handle the medical appointments by patients.
4. Poor in-patient service delivery dues to mismanaged rooms or assigning excess patients to one room
5. Poor out-patient services due to non-existence of a system to monitor patients outside the hospital

**1.4: Objectives**

### **1.4.1: General Objective**

To create a hospital management system that will replace the current obsolete systems to provide better appointment and record management, better handling of emergency and provide an overall efficient and faster medical service.

### **1.4.2: Specific objectives**

1. Reduce waiting times in hospitals by providing a better system to schedule doctors’ appointments.
2. Encourage patients to seek medical attention by providing a way to seek medical advice anonymously.
3. Provide a system that saves patient and doctors time by providing a way for medical inquiries without visiting a health center.
4. Monitor available wards and ICU rooms in a hospital so as to better handle emergencies and new admissions.
5. Provide a door-to-door health service that helps to monitor out-patients.
6. Improve in-patient service delivery by providing a system that helps monitors hospital wards

## **1.5: Justification**

Upon the completion of design of the system, the various users of the system will benefit from the system in the following ways:

1. **Reduced emergency response time:** The system will reduce emergency response time by handling emergencies more efficiently and faster.
2. **Reduced health fatalities:** Reduction in number of health fatalities as the system will automatically assign resources to emergencies.
3. **Appointment booking:** The system will eliminate time wastage in health facilities as appointments will be handled by a better system.
4. **Increases data security and integrity:** The system will reduce the incidences of missing files since the system replaces the current system with a centralized more secure system that enforces access to records for authorized personnel.
5. **Better out-patient service:** The system will improve out-patient service by providing a way for the patients to communicate with the medical personnel.
6. **Better in-patient service:** Improvement in in-patient service delivery by assigning and monitoring hospital rooms.

**CHAPTER 2: SYSTEM ANALYSIS**

## **2.1: Introduction**

Before we begun development of our system, we had to carry out an analysis of the system we hoped to create to satisfy the people that would use the system later. During the process of system analysis, we had to come up with the system development methodology which would guide our development process. We also had to gather data and analyze it so as to come up with the best requirement for our system. The requirements would the guide us on what specifications to put in place so as to come up with ana ideal system.

## **2.2: System development methodology**

The project methodology used in the development of the system is the System Development Life Cycle (SDLC) specifically following the waterfall method which is illustrated in *Figure 2.1*. The waterfall method is a sequential development process that flows like a waterfall through all phases of the Medicare project these include: - analysis, design, development, testing then deployment and maintenance, but for the first face of the project we will deal with the system analysis and design only.

The waterfall method follows a flow of phases in such a way that each phase is completely wrapped up before the next phase begins.

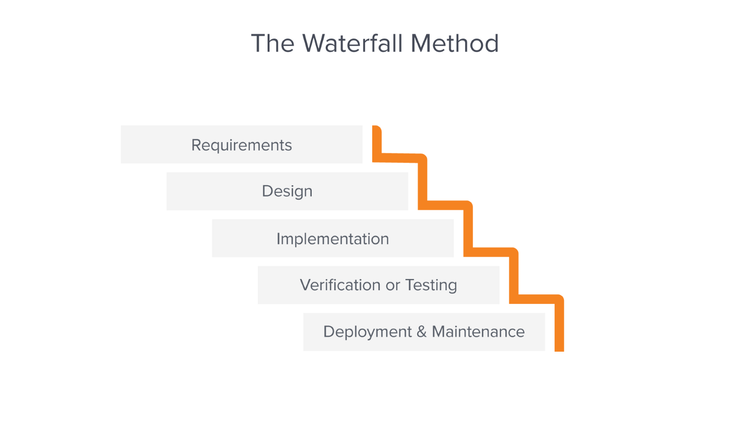


Figure 2.1

## **2.3: Feasibility study**

The Medicare hospital management system is a proposed software application that is designed to improve the efficiency and effectiveness of hospital operations. This feasibility report assesses the technical, economic, operational, legal, and scheduling feasibility of the project, and provides a recommendation about whether the project should be pursued.

### **2.3.1: Technical Feasibility**

The Medicare hospital management system is technically feasible based on the following considerations:

* We have the necessary resources and expertise to develop and implement the system.
* The system will be developed using modern, industry-standard technologies that are widely supported and well-documented.
* We have identified and assessed any potential technical challenges that could impact the project, and have developed plans to address them.

### **2.3.2: Economic Feasibility**

The Medicare hospital management system is economically feasible based on the following considerations:

* The system is expected to generate significant benefits, including improved patient care, increased efficiency, and cost savings.
* We have performed a detailed cost-benefit analysis that indicates a positive return on investment for the project.
* We have secured sufficient funding to cover the development and implementation costs of the system.

### **2.3.3: Operational Feasibility**

The Medicare hospital management system is operationally feasible based on the following considerations:

* The system is designed to be user-friendly and easy to use, with a clear and intuitive interface.
* We have a plan in place to train hospital staff on the use of the system, and will provide ongoing support and maintenance to ensure that it operates smoothly.

### **2.3.4: Legal and Regulatory Feasibility**

The Medicare hospital management system is legally and regulatory feasible based on the following considerations:

* We have reviewed all relevant laws and regulations, and have confirmed that the system is in compliance with all applicable requirements.

### **2.3.5: Scheduling Feasibility**

The Medicare hospital management system is scheduling feasible based on the following considerations:

* We have developed a detailed project schedule that outlines all key tasks and milestones, and that allows for sufficient time to complete the project.
* There are no known scheduling conflicts or constraints that could impact the project.

## **2.4: Requirements Elicitation**

We employed a few data collection methods to help us come up with the requirements for our **Medicare** system.

These methods include:

1. Observations:
   * This was conducted both on and out of campus, we visited the campus health center to check the kind of system they had in place but most of the observation had been done before as we had all visited some health facilities at some time.
   * Each group member brought the problems they observed to our usual meetings where we discussed on system requirements that would improve the current system.
2. Brainstorming:
   * Most of the system requirements came up during our brainstorming sessions as we all have fair experience in system development.
   * After coming up with ideas for our system requirements we conducted interviews to help us come up with a system that would benefit majority if not all users that we were developing the system for.
3. Interviews:
   * We conducted random and casual interviews among our peers to try and find out what kind of a system they would expect from a hospital.
   * The response from the users mainly boiled down to:
4. A fast system
5. Data security (confidentiality and integrity)
6. An interactive user interface (UI)
   * To get help us get more specific requirements and to know which requirements to give a higher priority we later asked the people to choose between:
     1. Speed and Security
     2. User Interaction and Speed

After collection of data by the methods mentioned above, we came up with the following requirements:

1. Speed – The system should be fast
2. Interactive – People needed a fairly interactive system
3. Secure data
4. Modern UI

## **2.5: Data and System Analysis**

The data collected was analyzed and the results were are as follows:

Based on our questions:

1. Speed vs Security
2. User Interaction vs Speed

We got the following responses:

1. For ‘Speed vs Security’, 2 of 5 users preferred speed while 3 chose security over speed
2. For ‘User Interaction vs Speed’ 4 of 5 users chose speed while the remaining preferred user interaction over speed

The results we got are represented in the charts below:

Given the results gathered we had to develop a system that was secure, fast and fairly interactive.

1. **security**

In the security sector, we had to consider it in the two areas our system revolves around:

1. Web Security
   * This was a critical area in our system as most of our system functionalities was web-based. The common threats in the web that we had to deal with are cross-site scripting, cross-site request forgery and indirect object referencing.
2. Database security
   * To ensure data integrity and confidentiality, we had to protect our database against various threats including and not limited to sql injections, excessive database privileges and weak audit trails.
3. **Speed**

In the area concerning speed, it was mostly dependent on the hardware but some aspects were also software-dependent.

We had to choose both a fast and scalable language to program our system and a fast DBMS but also, we had to specify the best hardware and software (OS) that would guarantee that our programming language of choice and the DBMS would run smoothly without straining the system.

1. **User Interaction**

This aspect was dependent on the frontend language we would use. We had to choose a scripting language that would guarantee a great user experience but it should also not interfere with the speed and security aspects of our system.

## **2.6: System Specifications**

The data collected in 2.4 above and analyzed in 2.5 helped us to come up with the system specifications that would carter for the speed, confidentiality, integrity and UI needs of the users expected to use our system.

The following are the system specifications:

## **2.6.1: Functional Specifications**

1. **Electronic medical record (EMR) module:**
   * This module would allow healthcare providers to store and manage patient health information, including medical history, diagnoses, medications, treatment plans, and test results.
2. **Appointment scheduling module:** 
   * This module would allow patients to book appointments with healthcare providers and staff to manage and schedule appointments.
3. **Billing and insurance module:** 
   * This module would handle the processing of insurance claims and patient billing for healthcare services, including integration with Medicare billing and coverage.
4. **Laboratory information management module (LIMS):**
   * This module would track and analyze lab results, including ordering and tracking lab tests, entering and storing test results, and providing reports to healthcare providers.
5. **Pharmacy management module:**
   * This module would track and dispense medications, including prescribing medications, creating medication lists, and generating medication orders and labels.
6. **Inpatient and outpatient management module:** 
   * This module would handle the management of inpatient and outpatient care, including admission, discharge, and transfer processes.
7. **Clinical decision support module:**
   * This module would provide healthcare providers with tools and resources to aid in the diagnosis and treatment of patients, such as clinical practice guidelines and drug interaction databases.
8. **Reporting and data analytics module:**
   * This module would provide reports and analytics on various aspects of the hospital's operations, such as patient outcomes, resource utilization, and financial performance.

### **2.6.2: Hardware Specifications**

* + - Core i5 processor or newer
    - At least 4GB RAM
    - At least 20GB of hard drive space
    - At Least 1TB hard drive space in server

### **2.6.3: Software Specifications**

1. **Python 3.7 or higher to provide Django: Any stable version above 4.0**

* Django is a python web framework that is both ***very fast*** and at the same time ***very secure*** as it provides a way to deal with ***cross-site request forgery*** and ***cross-site scripting*** through a module called ***corsheaders***. This module works by only allowing requests from domain names or Ip addresses specified in the site settings and blocks all other domains.
* Django also uses a ***model, view, template*** approach to build the application. The interesting aspect is the ***view*** which means that the request from web will not directly access our database tables but the ***view (a virtual table)*** and therefore protecting our database from ***sql injections***.
* Django also has a module called ***restframework*** which provides a faster and secure way of developing a ***RESTful API.*** It has a way of managing sessions using JSON and therefore no need to reference any object ids in requests which would help us a lot in dealing with the threat of indirect object referencing.
* The ***restframework*** modulealso provides serializers that clean all the data sent from the webpage and helps to prevent ***sql injections.***

1. **MariaDB server 10.5 or higher** 
   * + We chose this DBMS as it is ***fast*** and ***scalable*** which would help improve on the system overall speed.
     + It provides a very ***strong built-in audit trail mechanism*** and therefore incase of any system tampering we would know who was accountable.
     + It also has ***a good access control*** mechanism which would help us in ***controlling access*** to different types data depending on the permissions a user has.
     + The software is also ***open-source*** and therefore much cheaper than other alternatives.
2. **Nodejs: Any stable release above 16.0, to provide the React JS framework.**
   * + React JS uses a ***component-based*** approach instead of the ***traditional DOM-based*** approach. This means that our webpages are ***built up of smaller components*** that can be ***reused later*** during development.
     + React JS also provides ***client-side rendering*** which means that all web content is rendered on the browser by JavaScript. This means that instead of reloading the whole page when a change occurs React JS will only update the parts that have changed.
     + The React Js aspects mentioned above will help make our user interface very interactive without compromising the speed aspect.

In summary the combination of technologies we chose for our system above will ensure our users get the best of user interaction and high speed together with guaranteed data confidentiality and integrity.

**CHAPTER 3: SYSTEM DESIGN**

## **3.1: Introduction**

After the analysis we made in the previous chapter, it was time to come up with the design of our system. We have decided to built the backend with Django, React JS and Tailwind for the frontend and for our database we decide to go with MariaDB.

## **3.2: System Users**

For our system we will have three types of users:

1. Admin User
2. Staff User
3. End User

### **3.2.1: Admin User**

This is the user who is in charge of managing the system configuration and database configurations.

The admin user will be responsible for handling any system errors that may occur in the system while in use. Maintenance of the system is also the work of the system administrator.

This user will have access to all areas of the system including the databases schema. For security purposes the administrator will not be granted access to the information in the database which contains sensitive information like details patients, doctors and the anonymous users.

### **3.2.2: Staff User**

This are the users in charge of providing services to the end user e.g., assigning appointments and treatment. These users include the hospital staff and doctors.

Attributes of these users are:

1. User id
2. Name
3. Address
4. Email
5. Phone Number
6. National Id
7. Password

This Users will have access to some of the records such as patient records, room records and emergency information but will not be allowed to view or alter the database schema. They can insert and alter medical and room records.

They are to be registered by the System Admin.

### **3.2.3: End User**

This are the end users who receive the medical services such as the patients.

These users will only have to their own information like their medical records, room information and their profile but they will only be allowed to alter their profile.

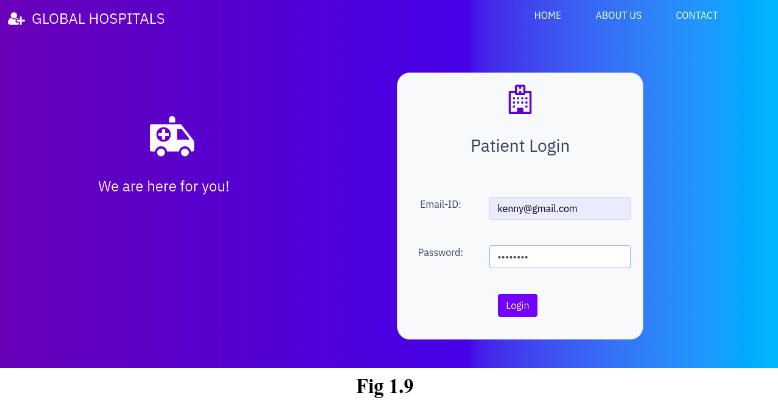
Attributes of these users are:

1. User id
2. Name
3. Address
4. Email
5. Phone Number
6. National Id
7. Password

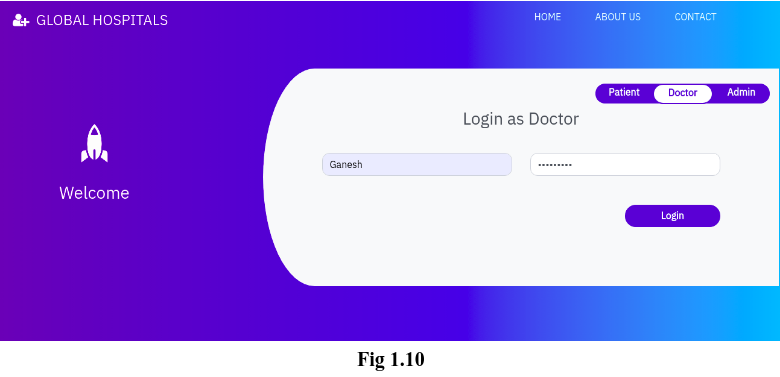
## **3.3.1: ­­­­­User Interfaces**

### **Login**

#### A. Patient Login

****

#### B**. Doctor Login**



**Login Pseudocode**

input: username, password

output: access granted or access denied

Step 1: **Prompt the user for their username and password.**

write "Enter your username:"

read username

write "Enter your password:"

read password

// Step 2: **Retrieve the correct password for the given username from the database**

password\_from\_database =retrieve\_password(username)

// Step 3: **Compare the password entered by the user to the password in the database**

if password\_from\_database == password

// Step 3a: **If the passwords match, grant access**

**write** "Access granted"

else

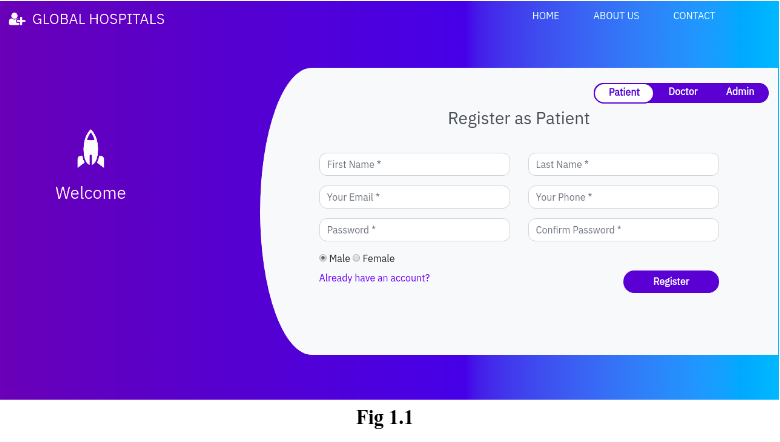
// Step 3b: **If the password doesn’t match, deny access**

write "Access denied"

end if

end procedure

#### Sign up

****

**Sign up Pseudocode**

input: patient's personal information (name, address, phone number, etc.), username, password

output: patient's account created or error message

// Step 1: **Prompt the user for their personal information**

write "Enter your name:"

read name

write "Enter your address:"

read address

write "Enter your phone number:"

read phone\_number

// ... other personal information fields

// Step 2: **Prompt the user for a desired username and password**

write "Enter a desired username:"

read username

write "Enter a desired password:"

read password

// Step 3: **Check if the desired username is already in use**

if username\_exists(username)

// Step 3a: **If the username is already in use, return an error message**

write "Sorry, that username is already in use. Please choose a different one."

else

// Step 3b: **If the username is not already in use, create a new account for the patient**

create\_patient\_account(name, address, phone\_number, username, password)

write "Your account has been created successfully."

end if

end procedure

#### About us page

****

**About us Pseudocode**

input: hospital's information (name, location, history, mission statement, etc.)

output: HTML code for the "About Us" page

// Step 1: **Retrieve the hospital's information from the database**

hospital\_name = retrieve\_hospital\_name()

hospital\_location = retrieve\_hospital\_location()

hospital\_history = retrieve\_hospital\_history()

hospital\_statement = retrieve\_hospital\_statement()

// ... other hospital information fields

// Step 2: **Generate the HTML code for the "About Us" page using the hospital's information**

html\_code += "<h1>" + hospital\_name + "</h1>"

html\_code += "<p>Location: " + hospital\_location + "</p>"

html\_code += "<h2>”History “</h2>"

html\_code += "<p>" + hospital\_history + "</p>"

html\_code += "<h2>”Mission Statement “</h2>"

html\_code += "<p>" + hospital\_mission\_statement + "</p>"

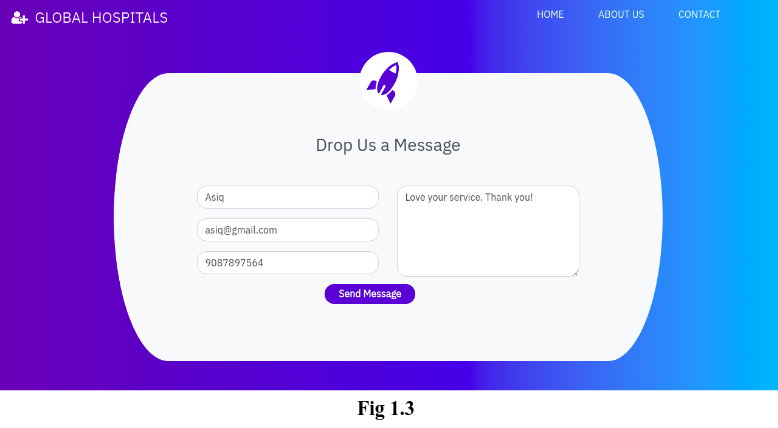
// ... other hospital information field

// Step 3: **Return the HTML code for the "About Us" page**

**return html\_code**

end procedure

1. **Contact us page**

****

**Pseudocode**

input: hospital's contact information (phone number, email, address, etc.)

output: HTML code for the "Contact Us" page

// Step 1: **Retrieve the hospital's contact information from the database**

hospital\_phone\_number = retrieve\_hospital\_phone\_number()

hospital\_email = retrieve\_hospital\_email()

hospital\_address = retrieve\_hospital\_address()

// ... other hospital contact information fields

// Step 2: **Generate the HTML code for the "Contact Us" page using the hospital's contact information**

html\_code = "<h1>”Contact Us</h1>"

html\_code += "<p>”Phone: " + hospital\_phone\_number + "</p>"

html\_code += "<p>Email: " + hospital\_email + "</p>"

html\_code += "<p>Address: " + hospital\_address + "</p>"

// ... other hospital contact information fields

// Step 3: **Return the HTML code for the "Contact Us" page**

**return html\_code**

end procedure

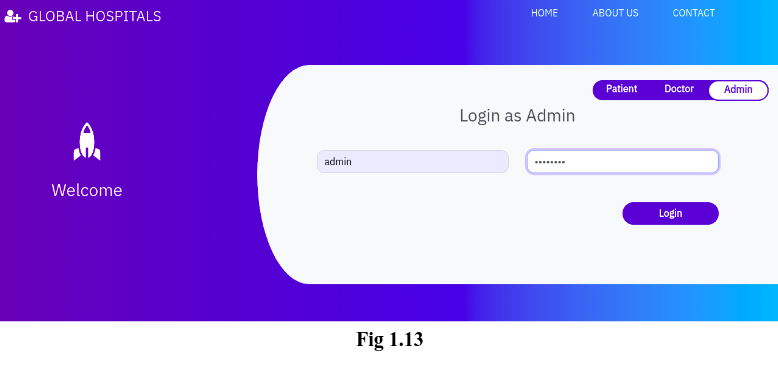
This pseudocode assumes that there are functions called “retrieve\_hospital\_phone\_number()” , “retrieve\_hospital\_email” and “retrieve\_hospital\_address” that takes no input and return the corresponding contact information for the hospital form the database

## **3.3.2: Admin interfaces**

### **1 . Login**

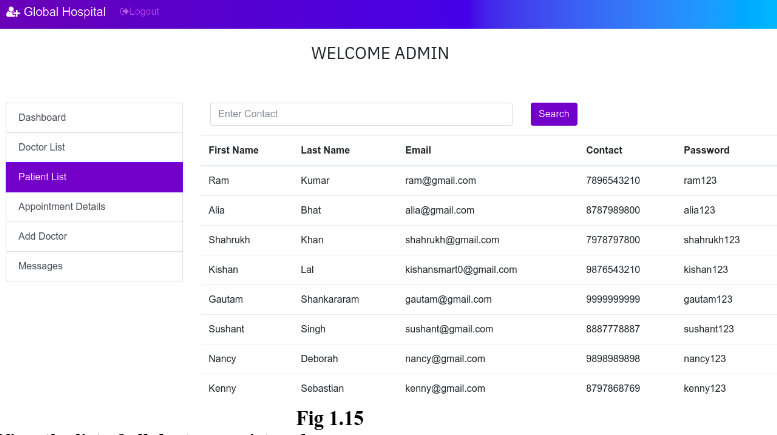
The pseudocodes for login will be reused to login the admin since the procedure is similar.

Some addons might be made on the interface during implementation as depicted in the diagram below.

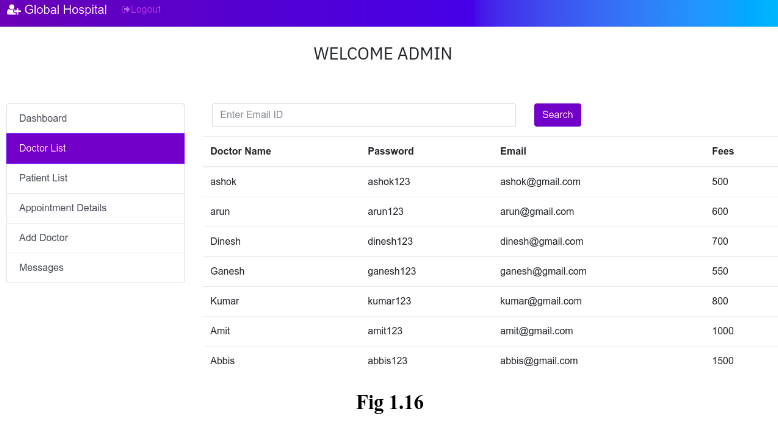


### **2.Views**

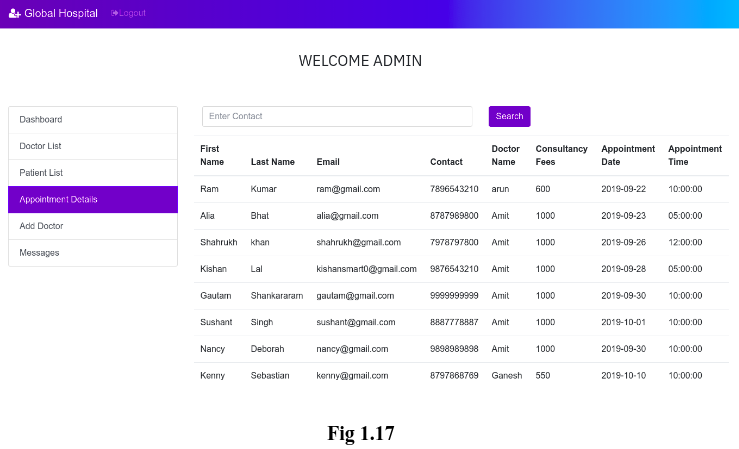
**a . All Patients**

****

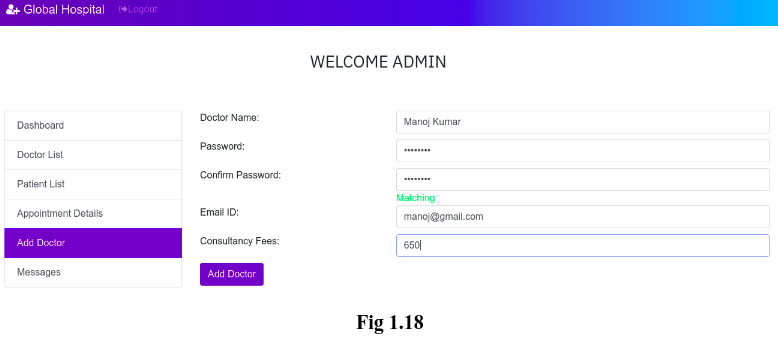
**b. All Doctors**

****

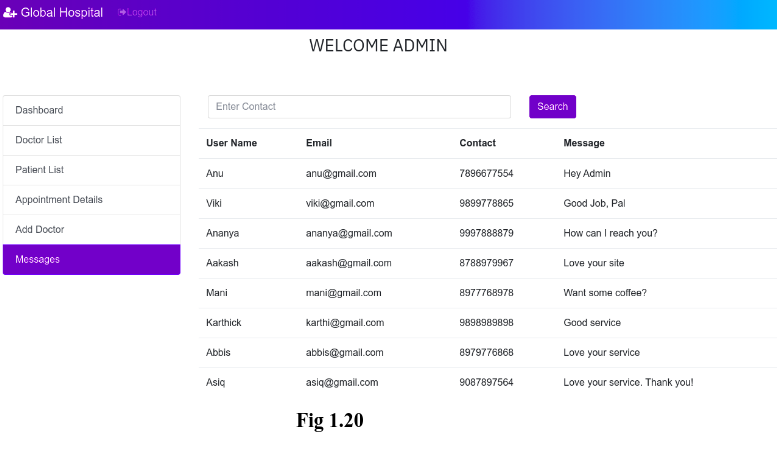
**c. Appointmets list**

****

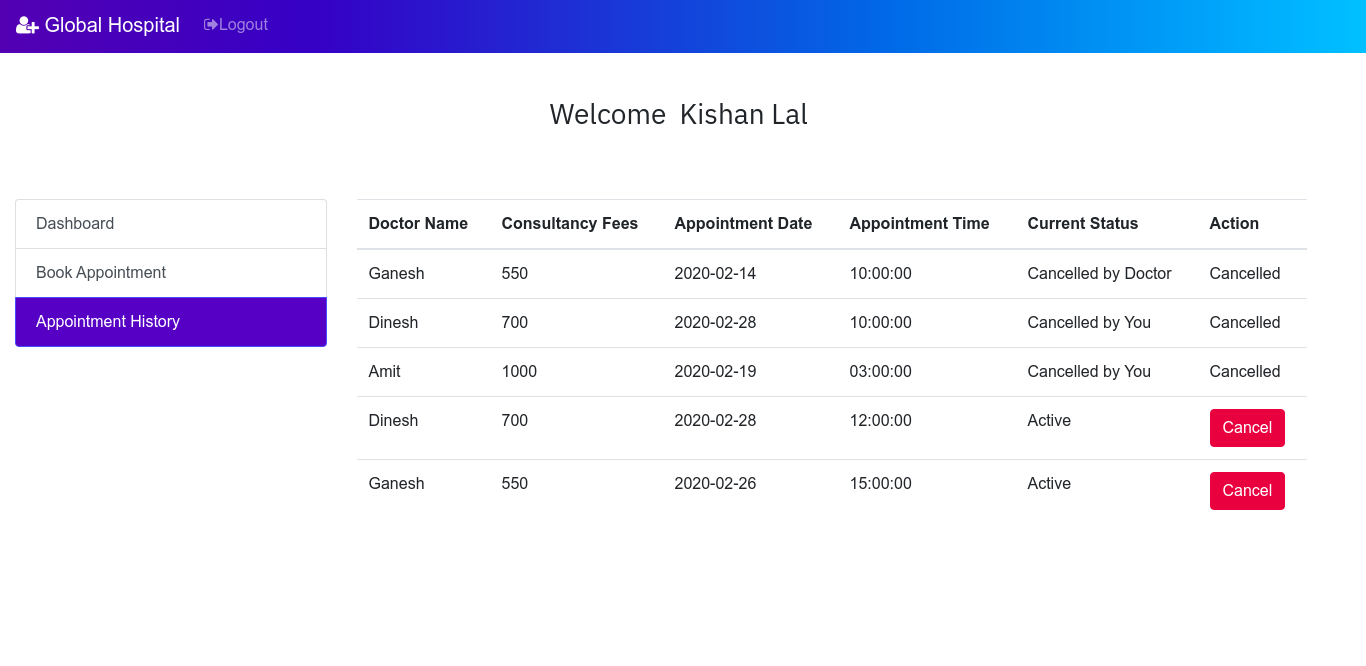
**d . Add Doctor**

****

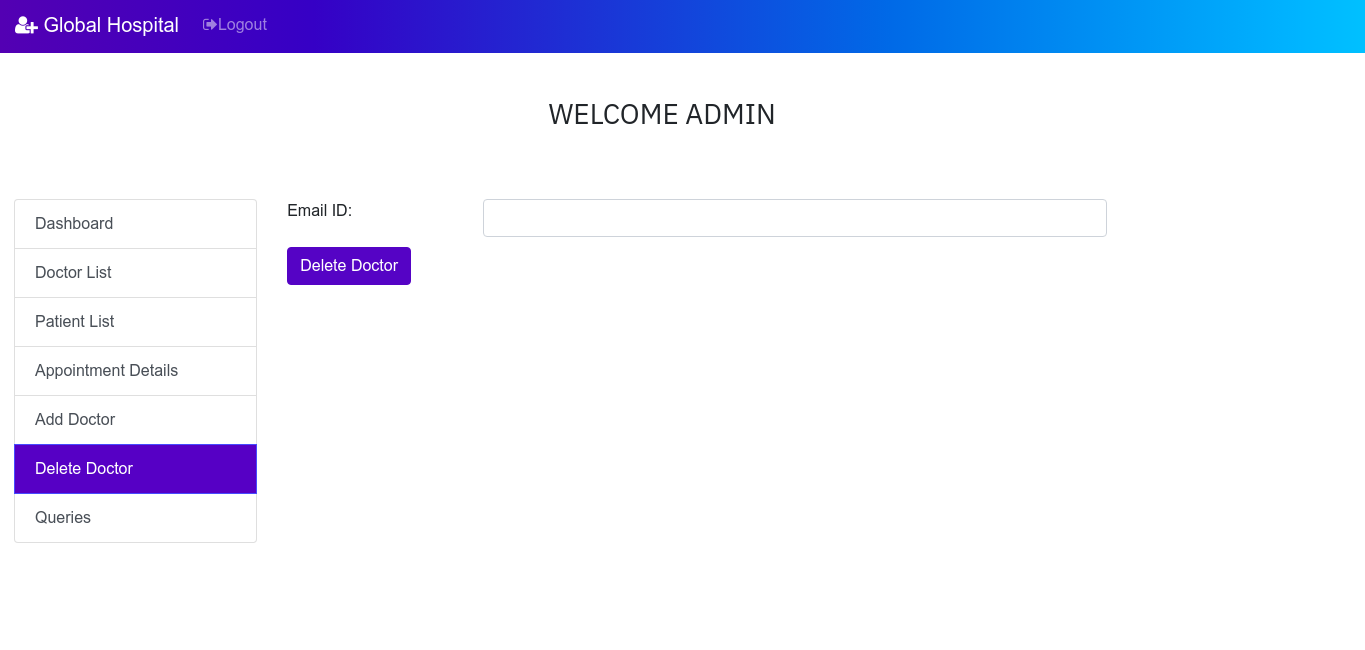
**e . View users feedback**

****

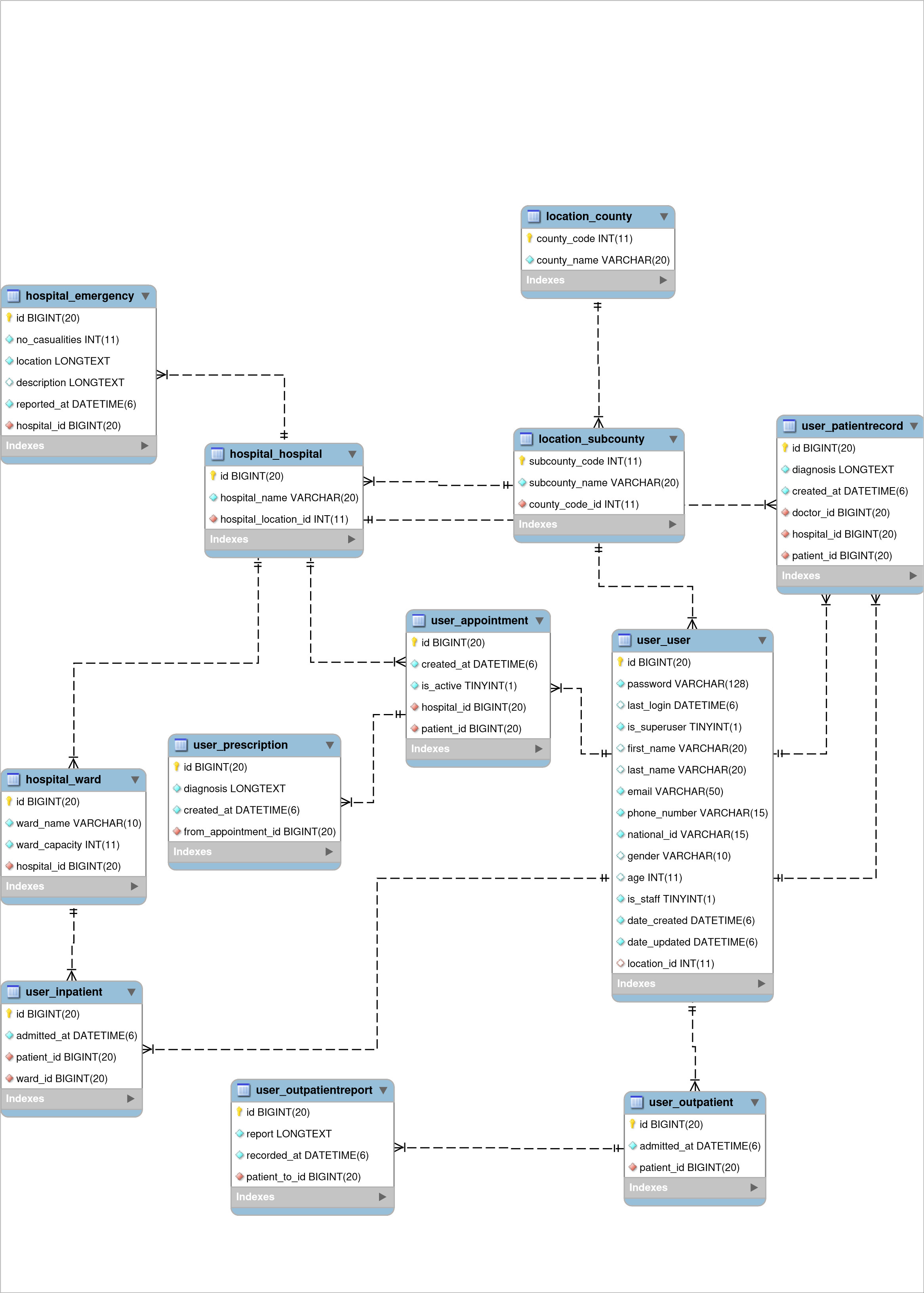
**f. Cancel Appointment**

****

**g. Remove Doctors**



## **3.4: Database Schema**



## **3.5: Data Flow Diagrams**

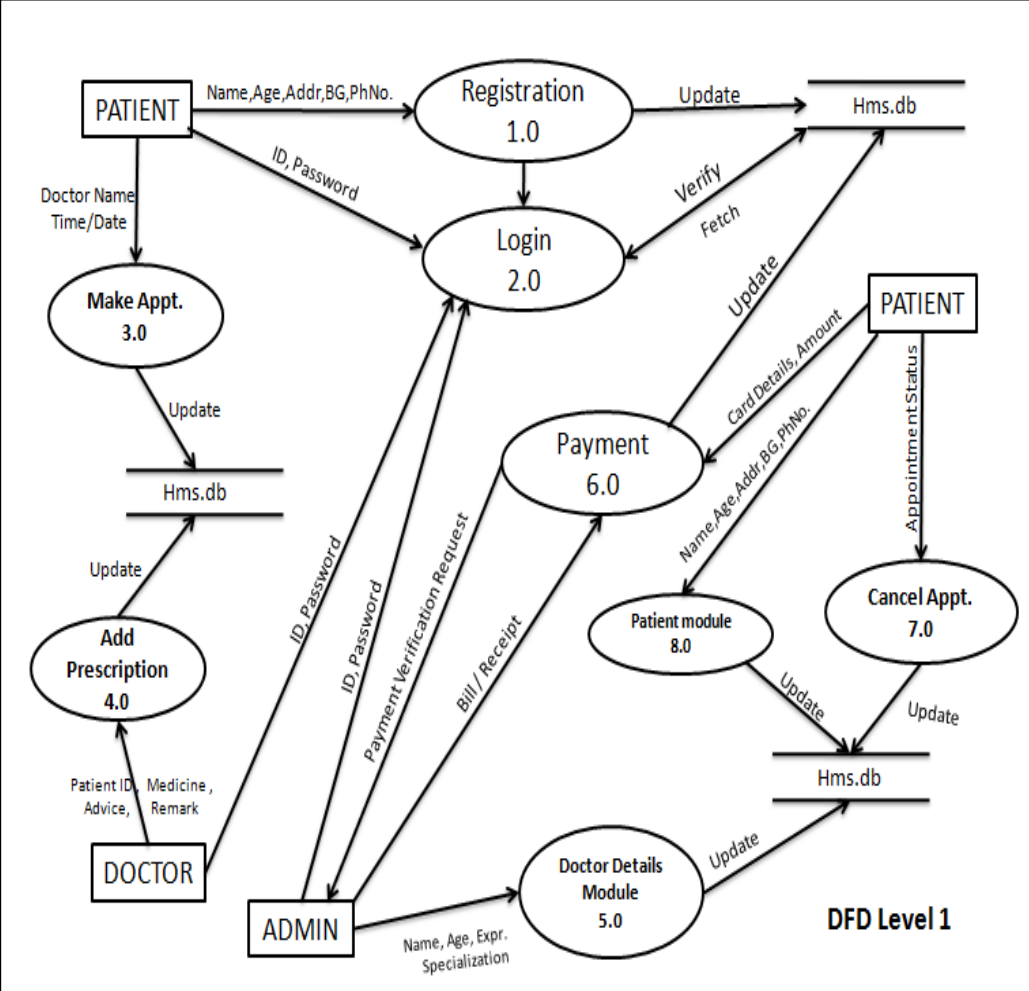
* Coming up with a system design sometimes may be a little tricky, the use and deployment of system design techniques **for better visualization of the system**
* Up to two-level DFDs were made for the whole system description project

### **3.4.1: Context Level Diagram**

Figure 3.1

### **3.4.2: DFD Level – 1**

Figure 3.2



### **3.4.3: DFD Level – 2**

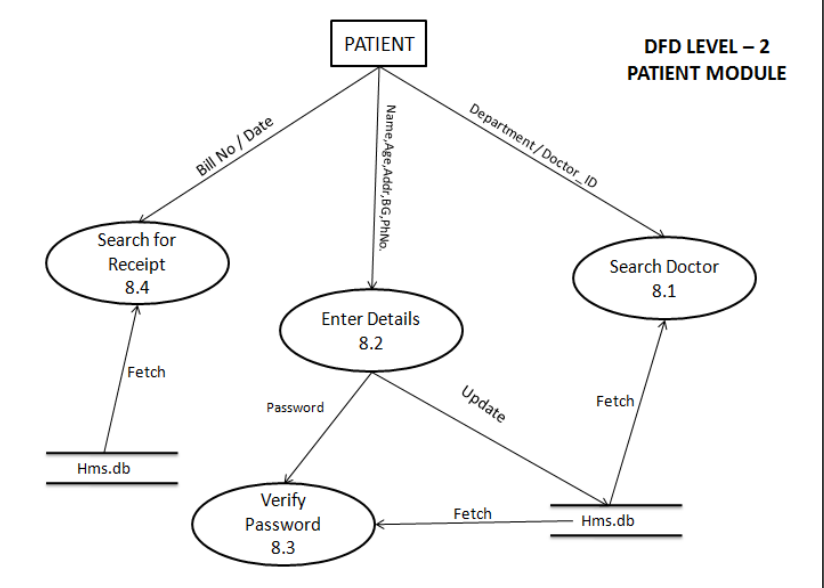
#### 1.Registration

Figure 3.3

#### 2.Login

Figure 3.4

#### 3.Patient module



#### 4.Doctor Module

#### 

#### 5.Add Prescription

#### 

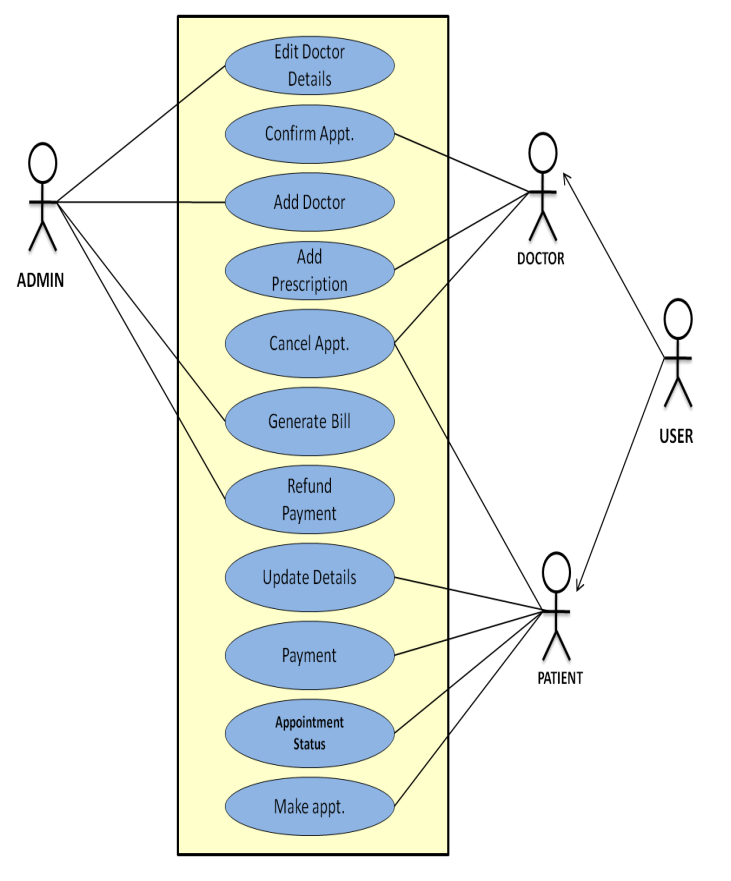
#### 6.Payment

#### 

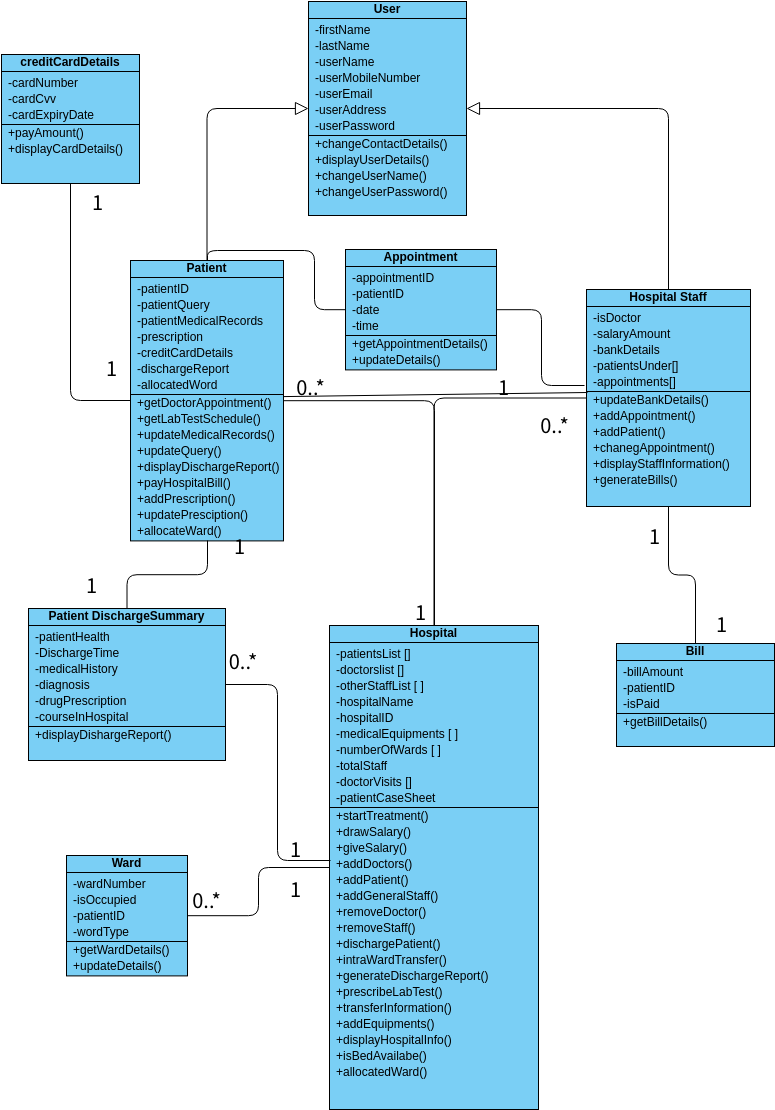
#### 7.Cancel Appointment

#### 

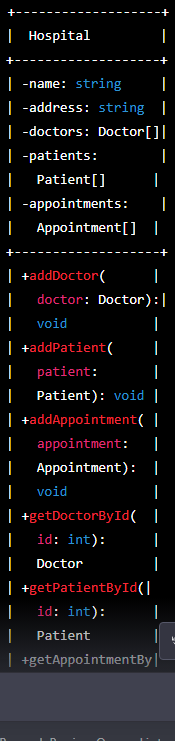
### **3.4: Use Case Diagram**

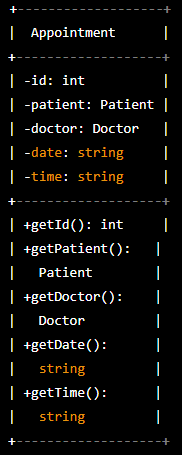


### **3.5: Class Diagram**



#### Optimized class diagrams







1. ***Patient class***
   * It represents a patient at the hospital. It has attributes such as id, name, age, address, and phone, which store the patient's identification number, name, age, address, and phone number, respectively. It also has methods such as getId(), getName(), getAge(), getAddress(), and getPhone(), which return the patient's identification number, name, age, address, and phone number, respectively.
2. **Doctor class**

* It represents a doctor at the hospital. It has attributes such as id, name, and specialty, which store the doctor's identification number, name, and specialty, respectively. It also has methods such as getId(), getName(), and getSpecialty(), which return the doctor's identification number, name, and specialty, respectively.

1. ***The Appointment class***

* represents an appointment at the hospital. It has attributes such as id, patient, doctor, date, and time, which store the appointment's identification number, patient, doctor, date, and time, respectively. It also has methods such as getId(), getPatient(), getDoctor(), getDate(), and getTime(), which return the appointment's identification number, patient, doctor, date, and time, respectively.

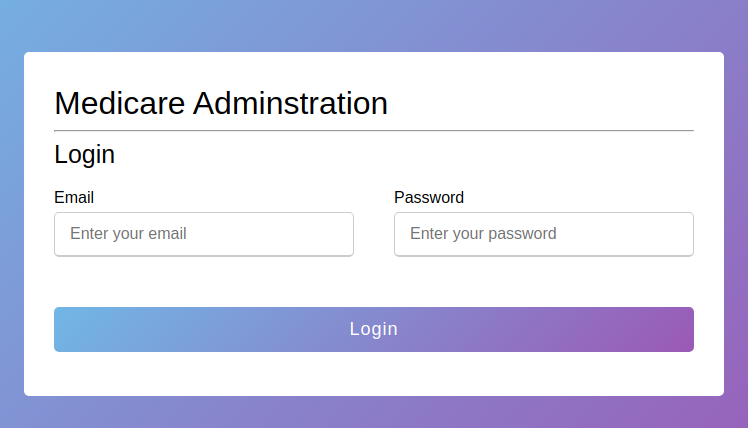
1. **The Hospital class**

* represents a hospital. It has attributes such as name, address, doctors, patients, and appointments, which store the hospital's name, address, list of doctors, list of patients, and list of appointments, respectively. It also has methods such as addDoctor(), addPatient(), addAppointment(), getDoctorById(), getPatientById(), and getAppointmentById(), which allow the hospital to add a doctor, patient, or appointment, and retrieve a doctor, patient, or appointment by their identification number.

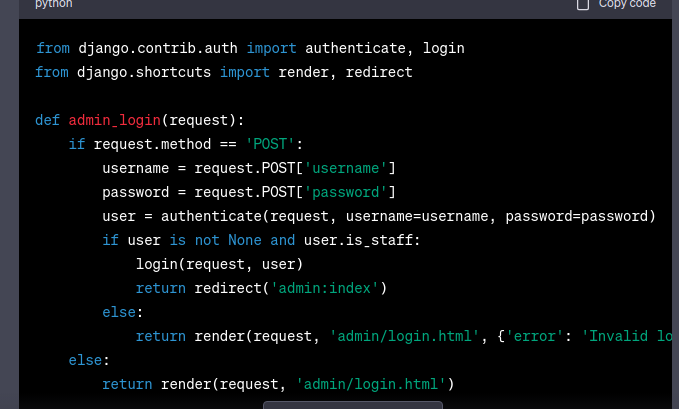
In this class diagram, the Patient, Doctor, and Appointment classes are related to the Hospital class through aggregation. This means that the Hospital class is composed of Patient, Doctor, and Appointment objects, and these objects can exist independently of the Hospital class. The Appointment class is also related to the Patient and Doctor classes through association, as an Appointment object uses the services of a Patient and a Doctor object.

# **CHAPTER 4: TESTING AND IMPLEMENTATION**

ADMIN INTERFACES AND SEGMENTED CODE

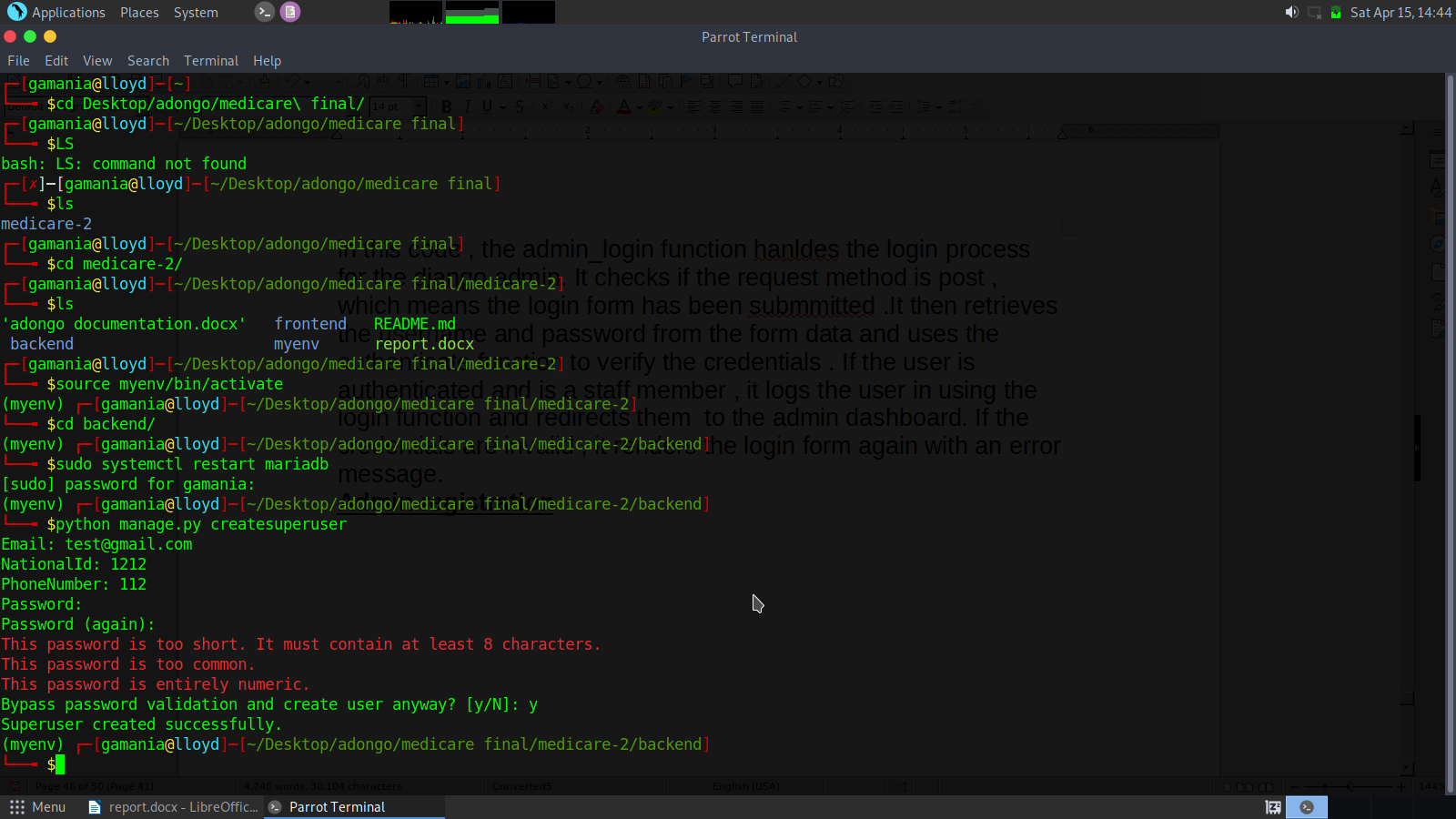


logic code for the interface

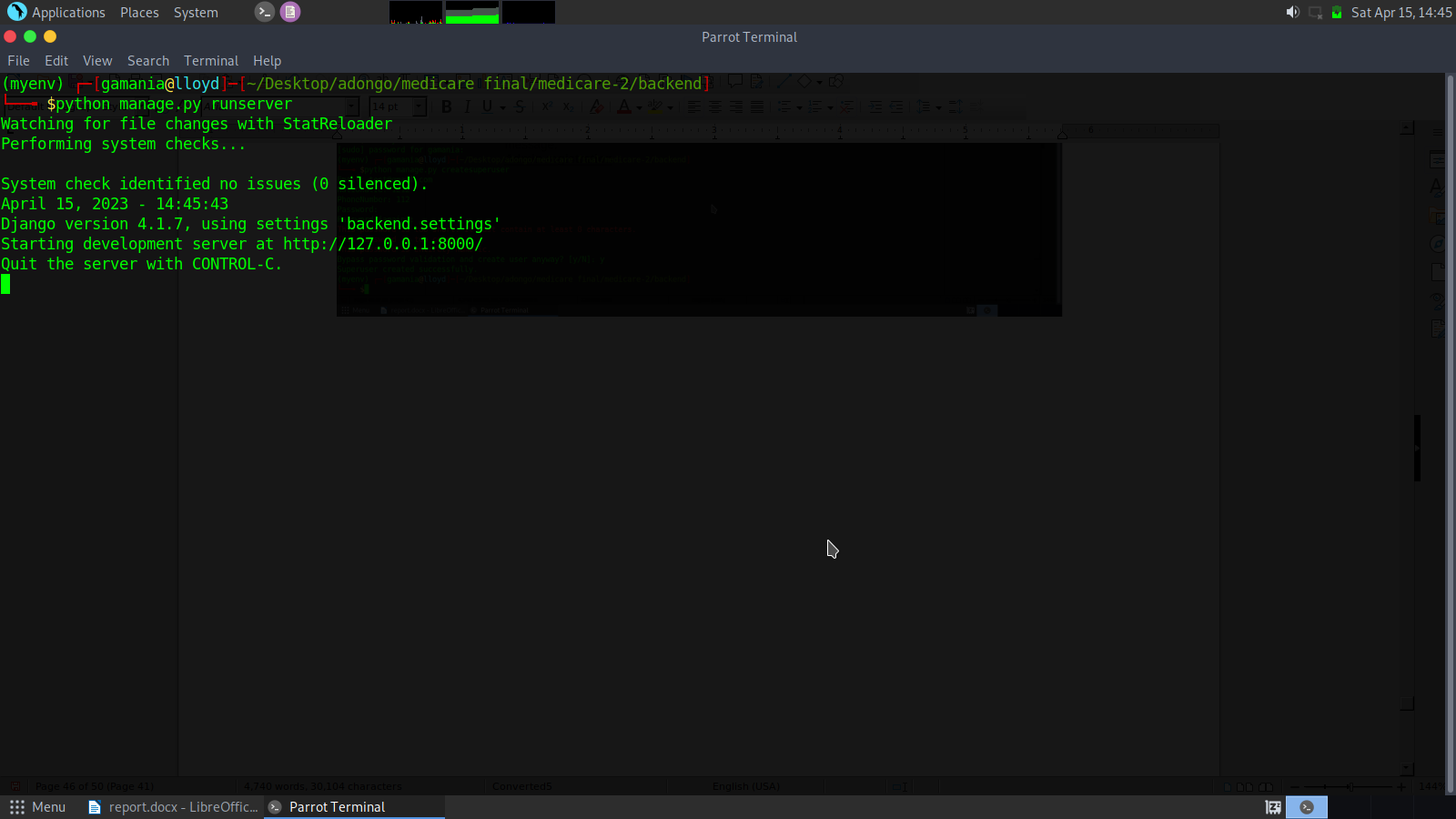


in this code , the admin\_login function hanldes the login process for the django admin. It checks if the request method is post , which means the login form has been submmitted .It then retrieves the username and password from the form data and uses the authenticate function to verify the credentials . If the user is authenticated and is a staff member , it logs the user in using the login function and redirects them to the admin dashboard. If the credentials are invalid , it renders the login form again with an error message.

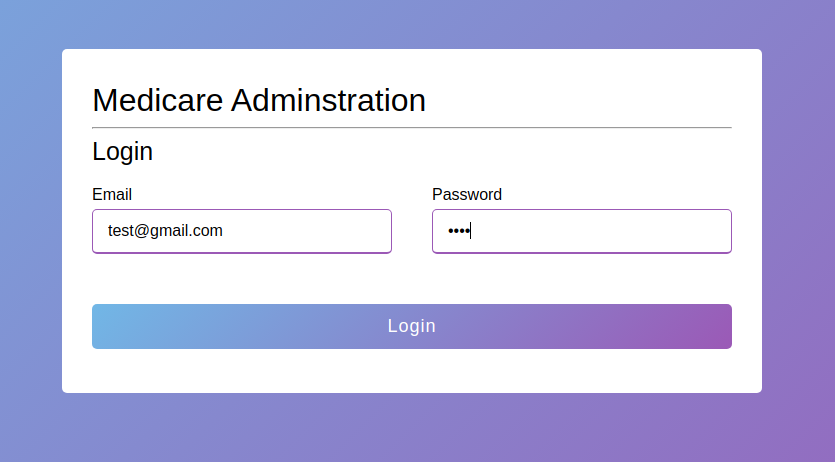
**Admin registration**



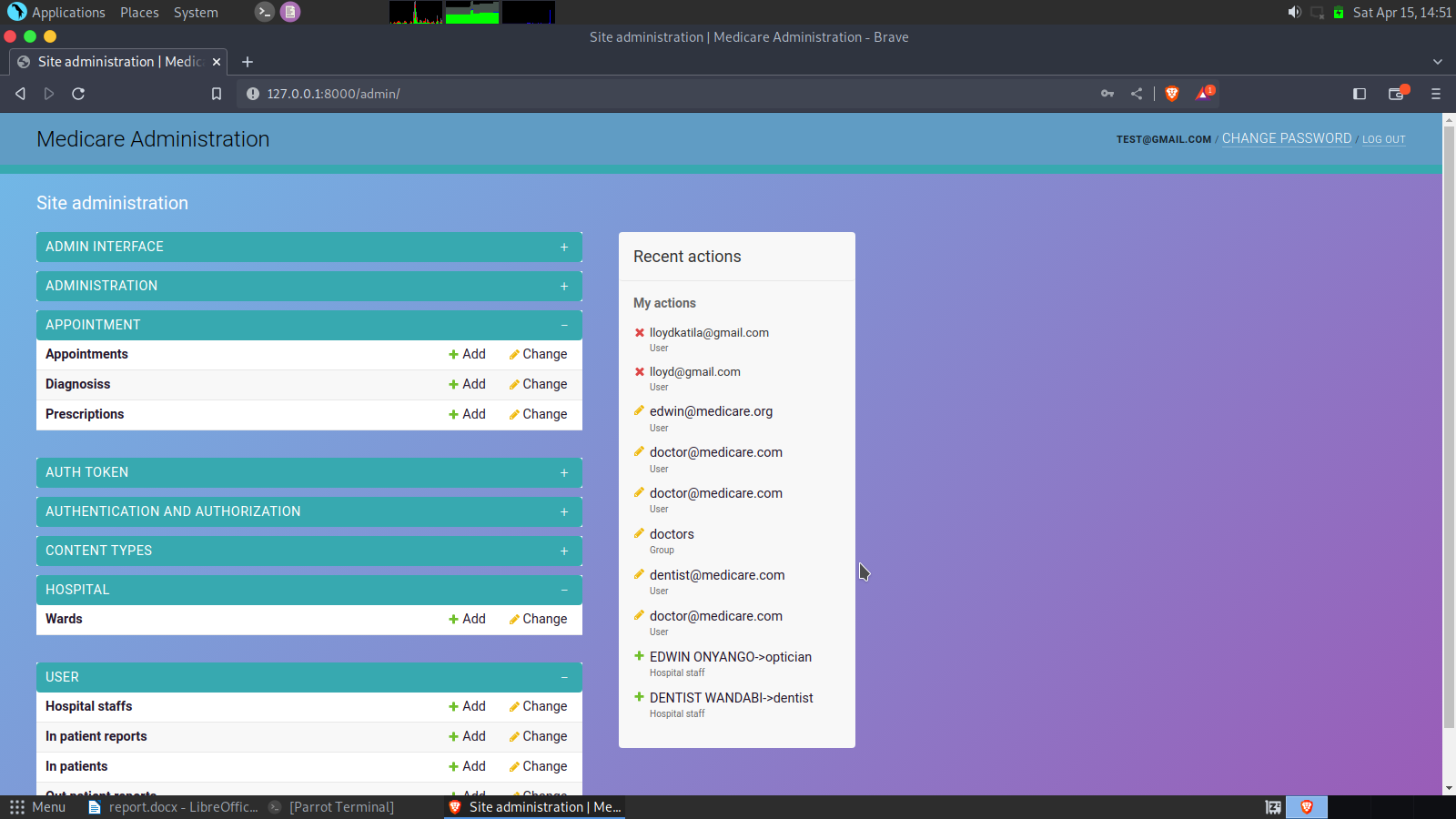
running the server



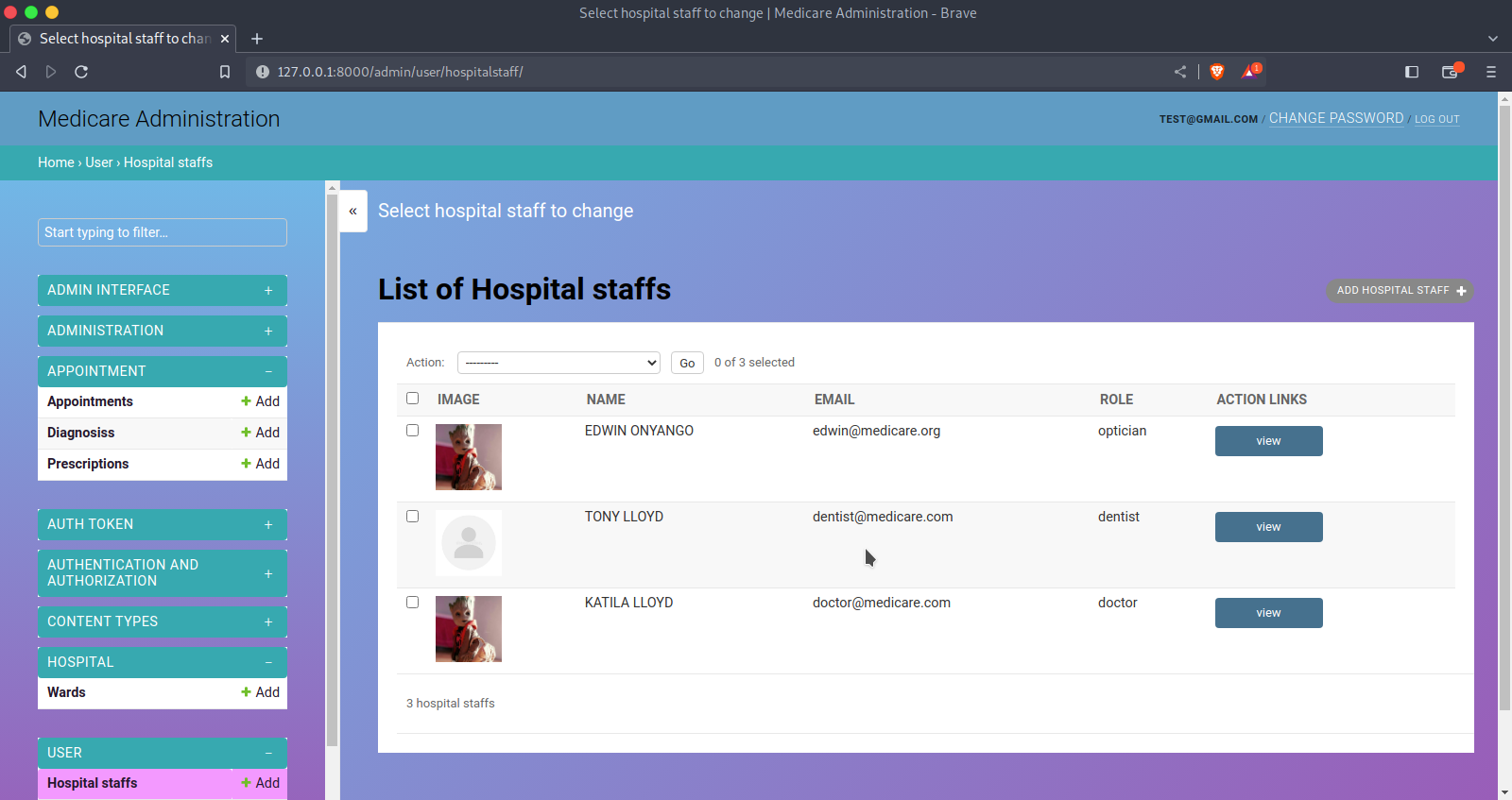
**admin dashboard interface**



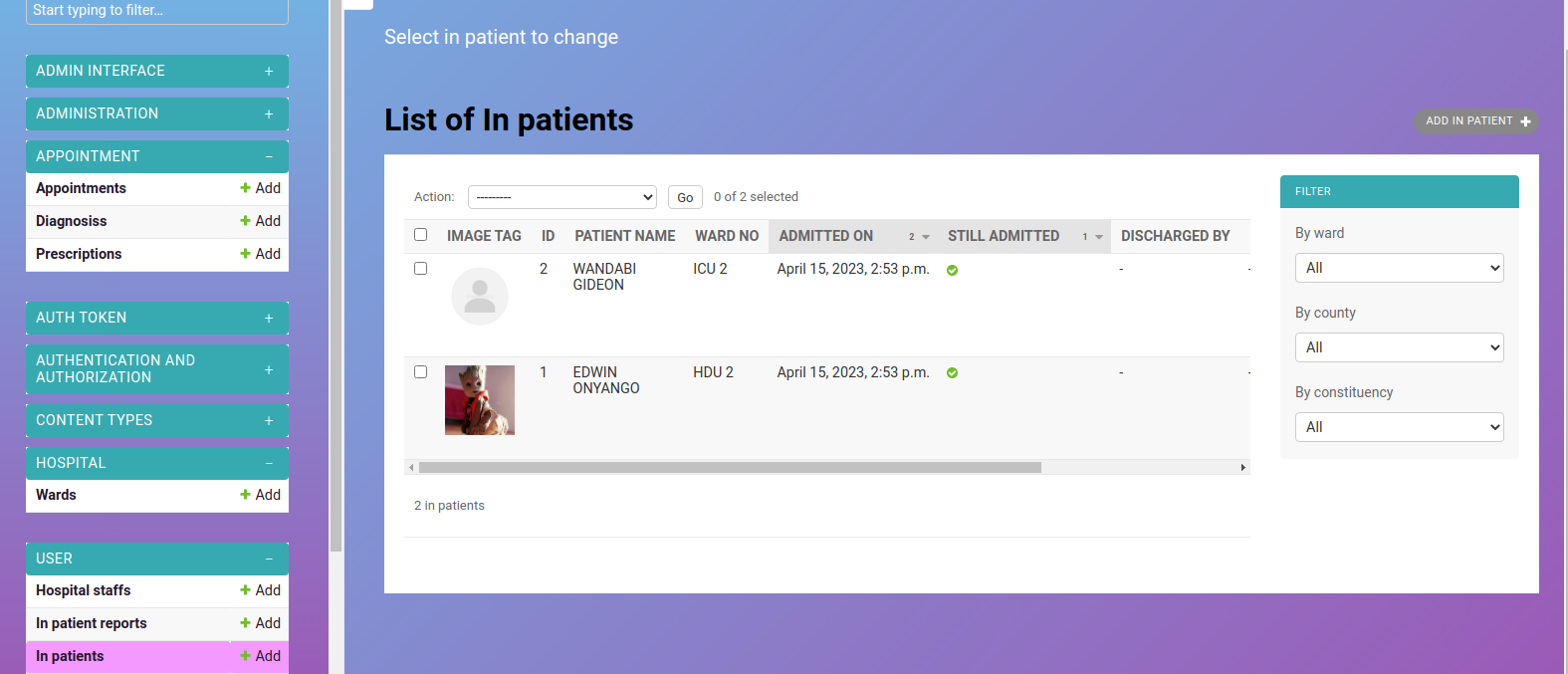
enter the details and then click login



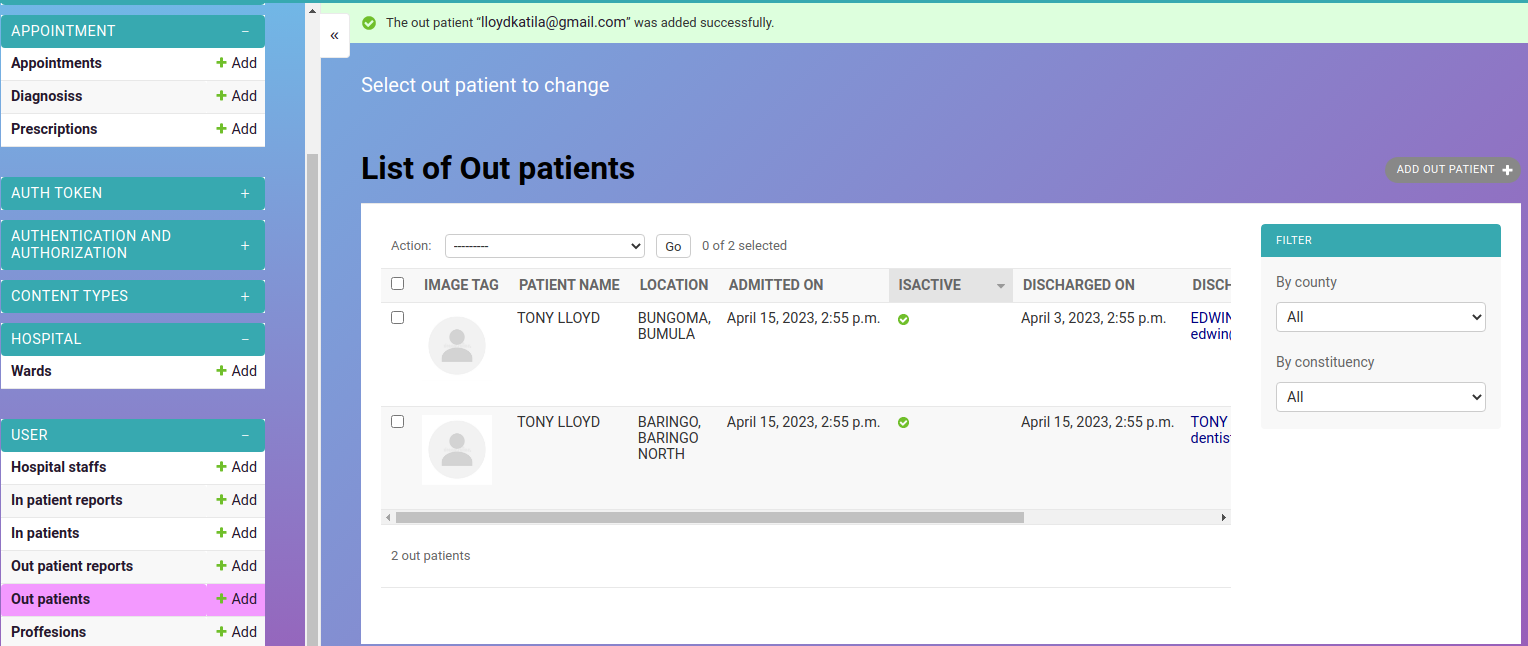
hospital staff view and action button



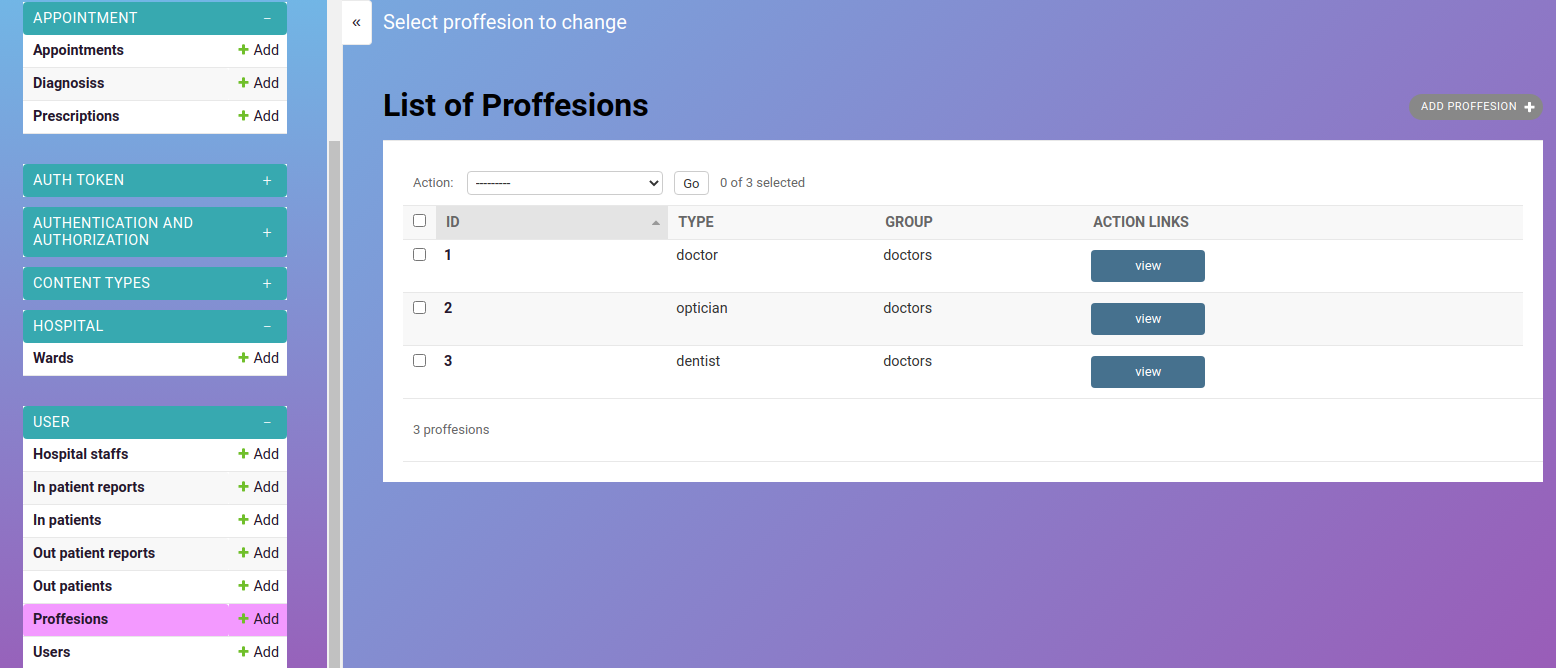
inpatients list view and action button



outpatient list view and action button

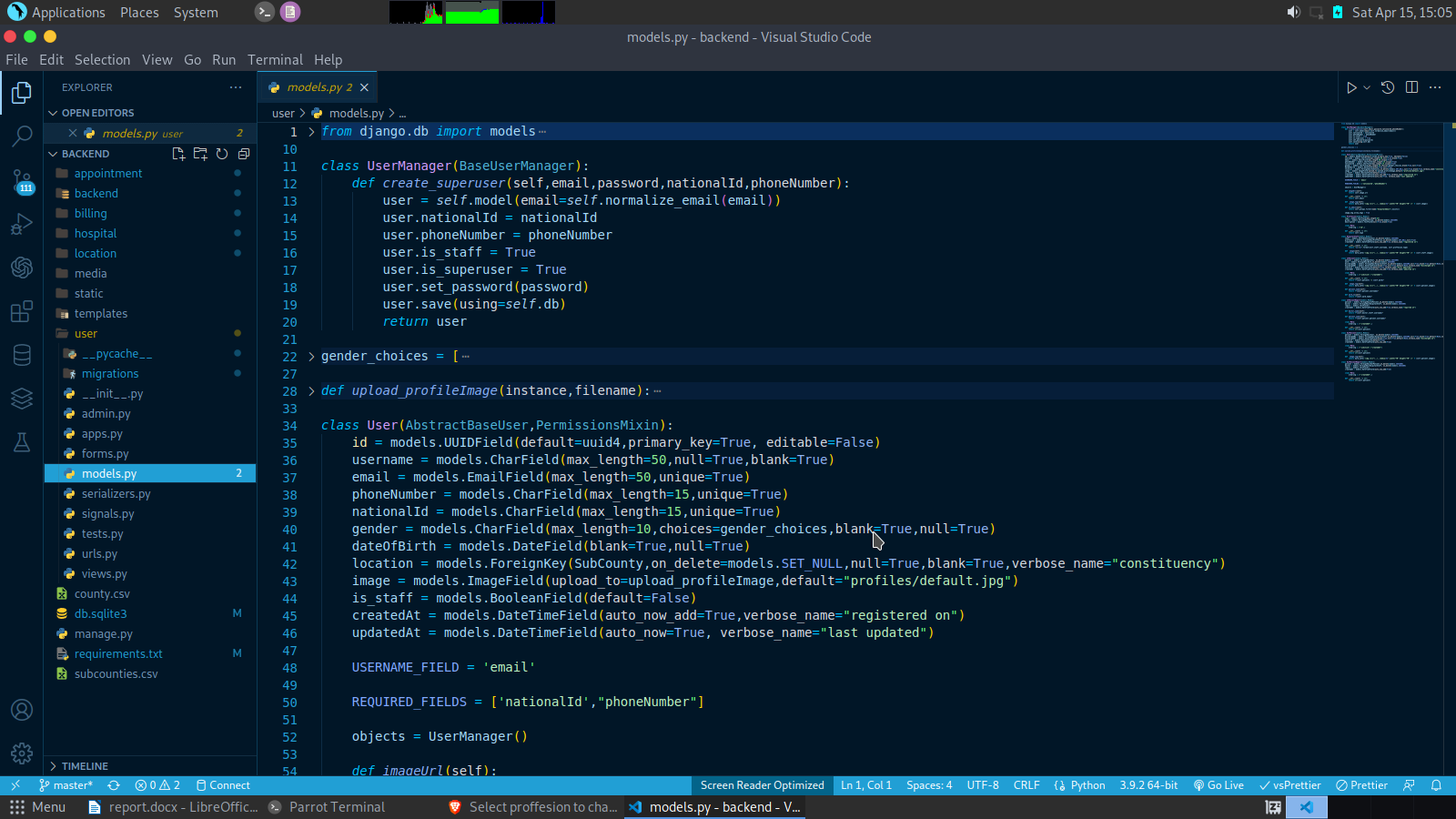


list of doctor's

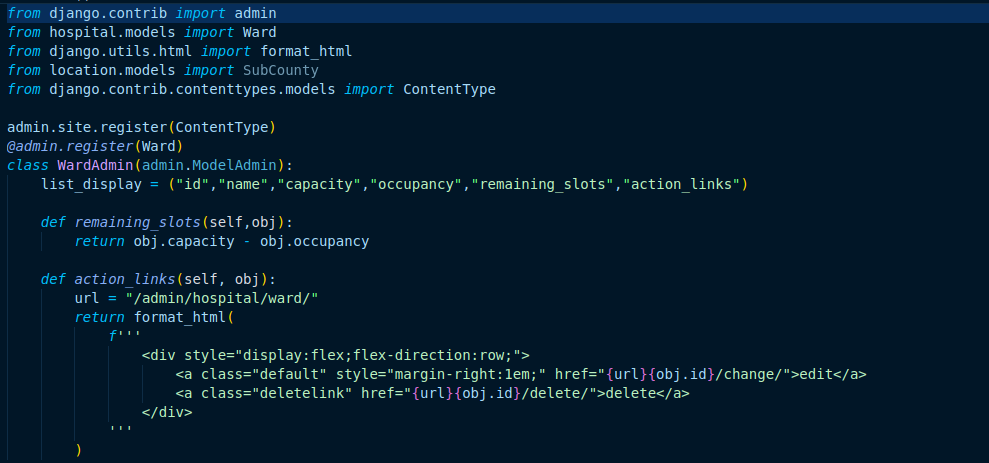
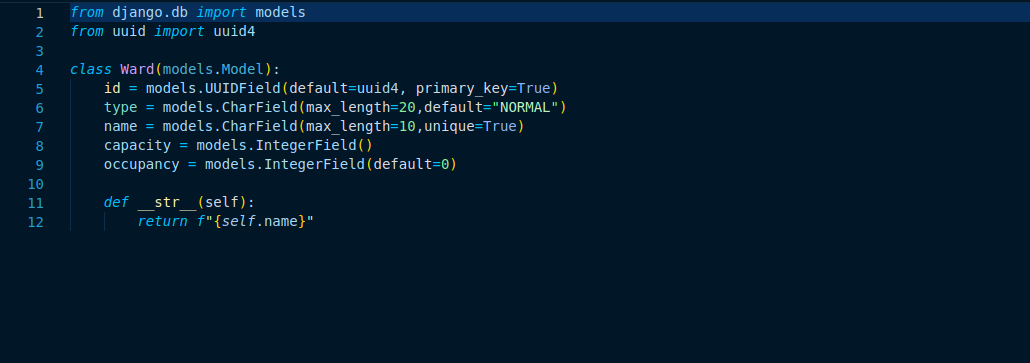


**Segmented code for administrators and doctor's**

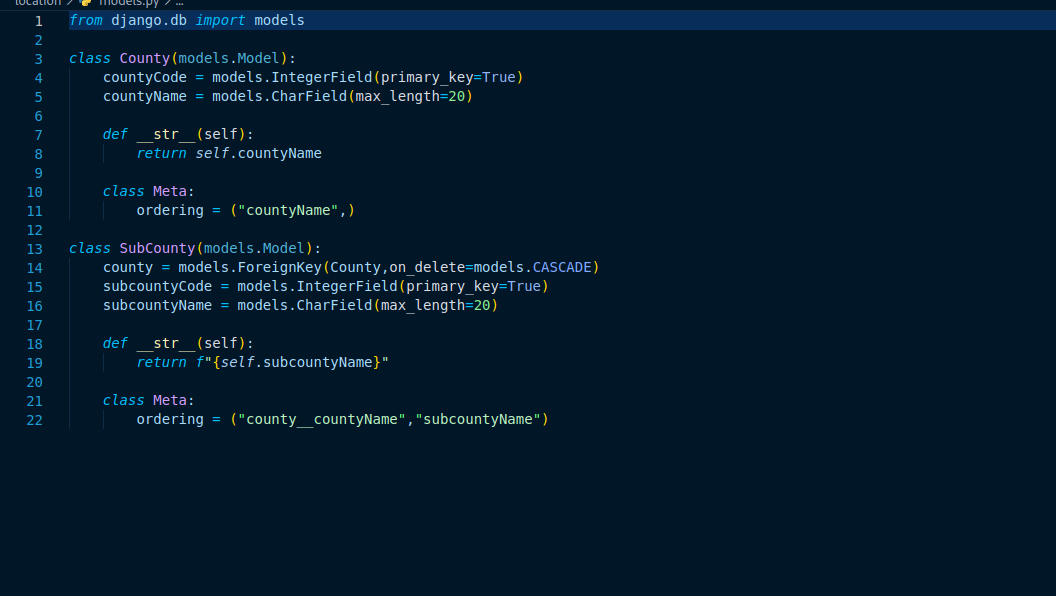
**a)all users funtion**



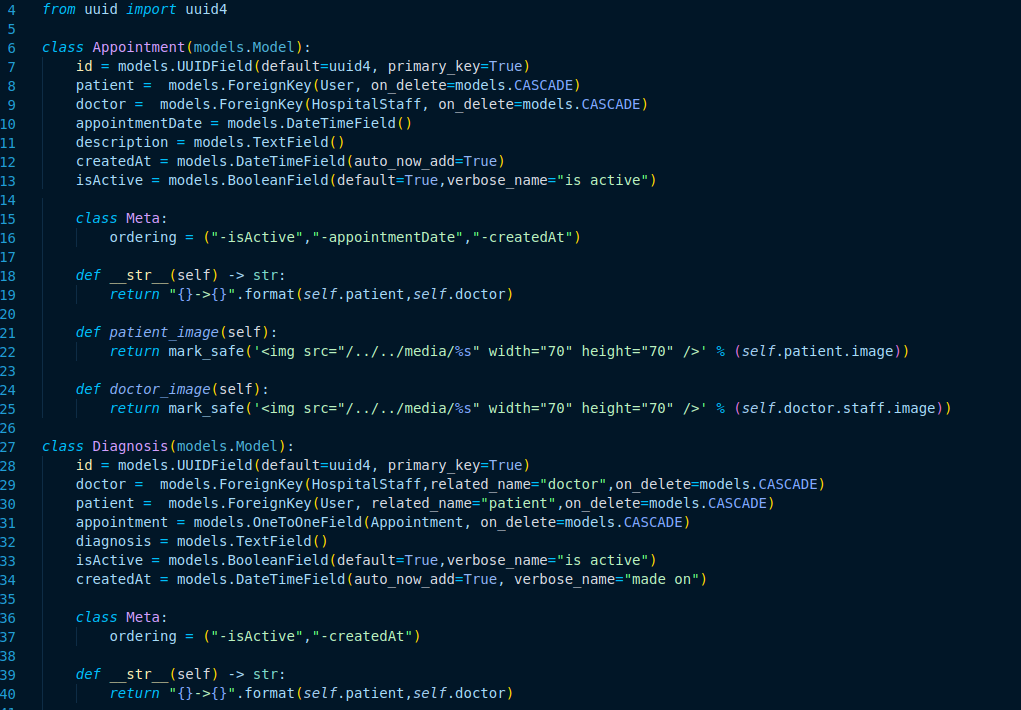
**b) ward models code**

**registration of hospital code**

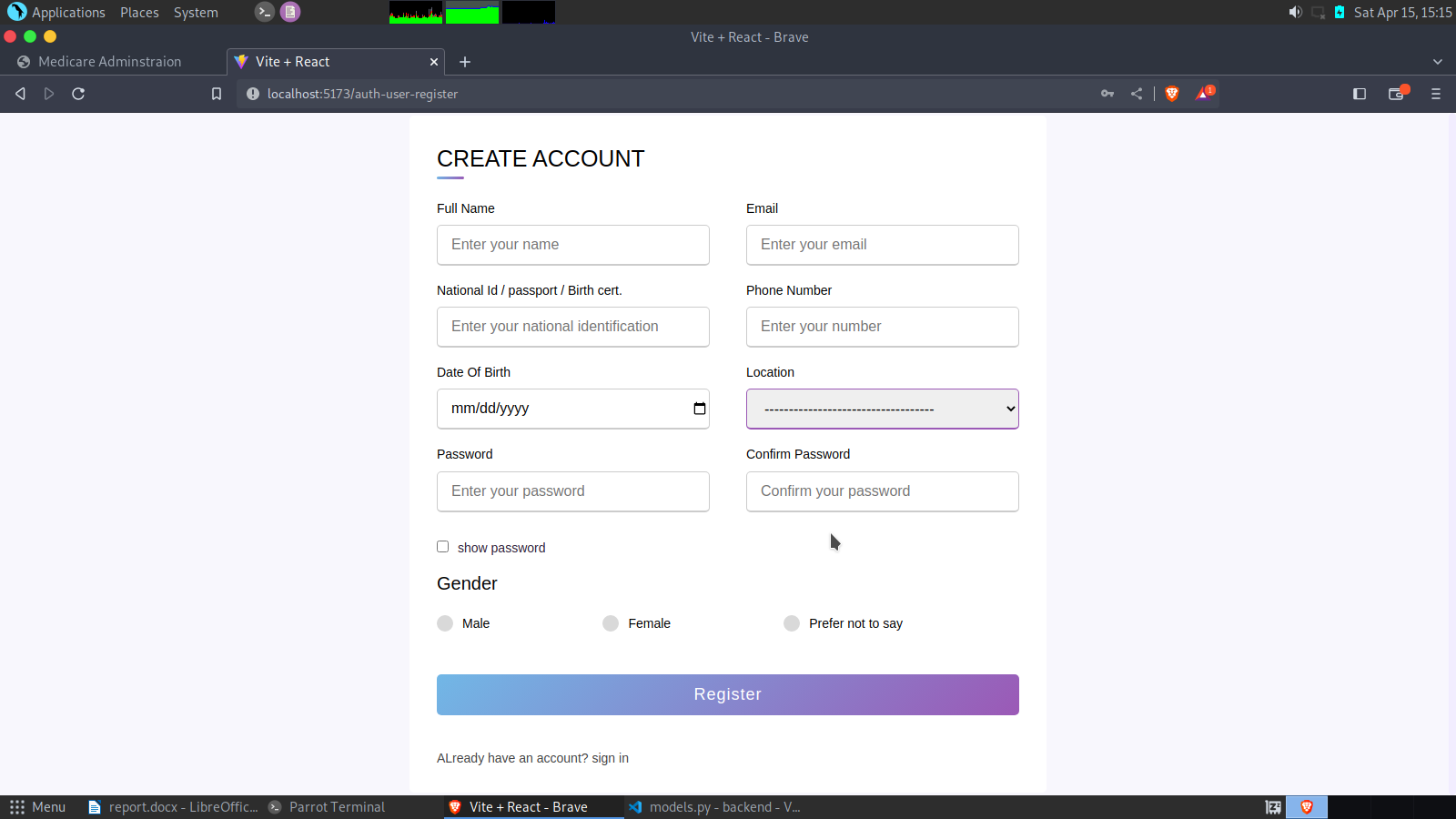
c )location code



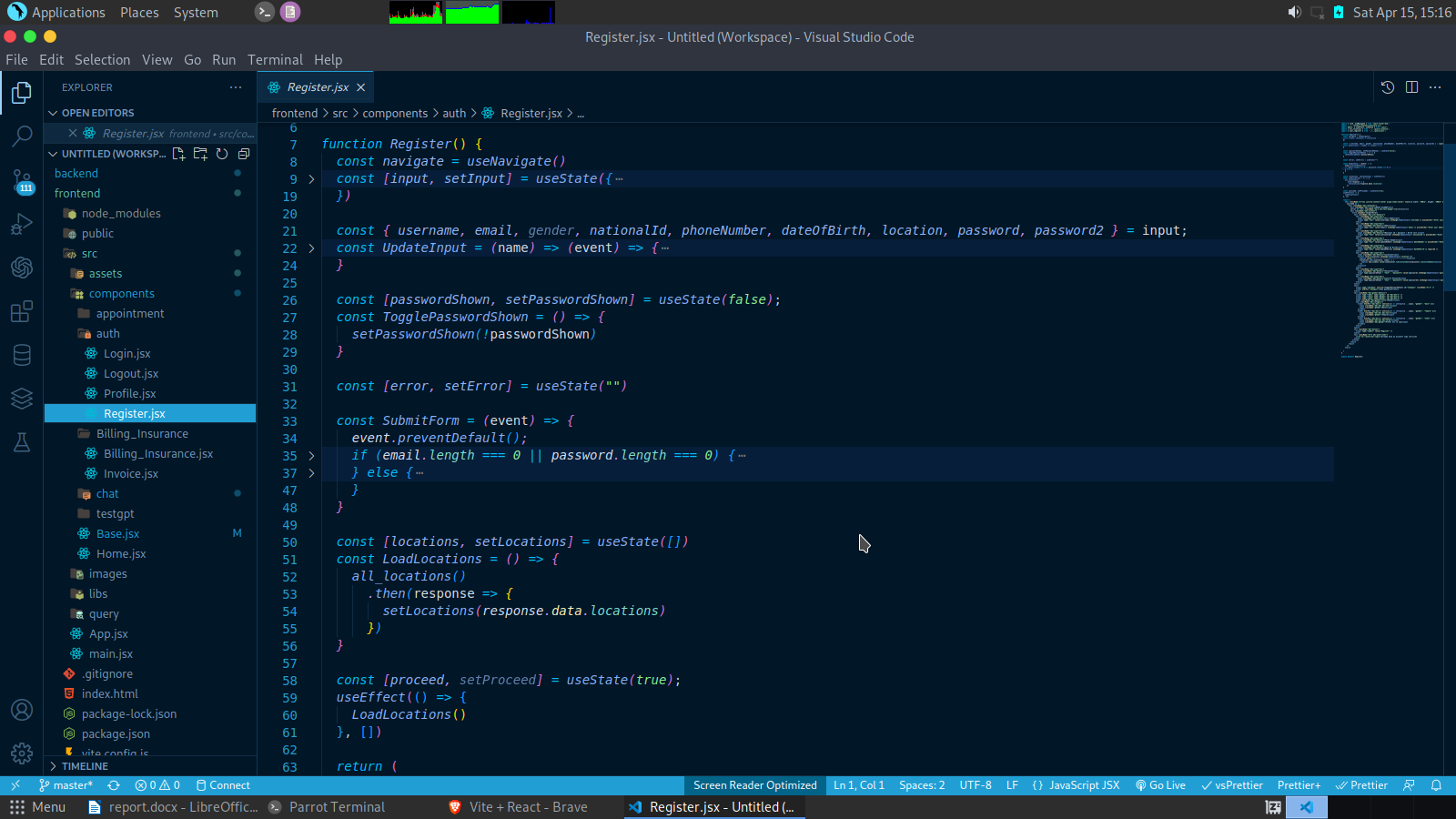
d)appointment models code :



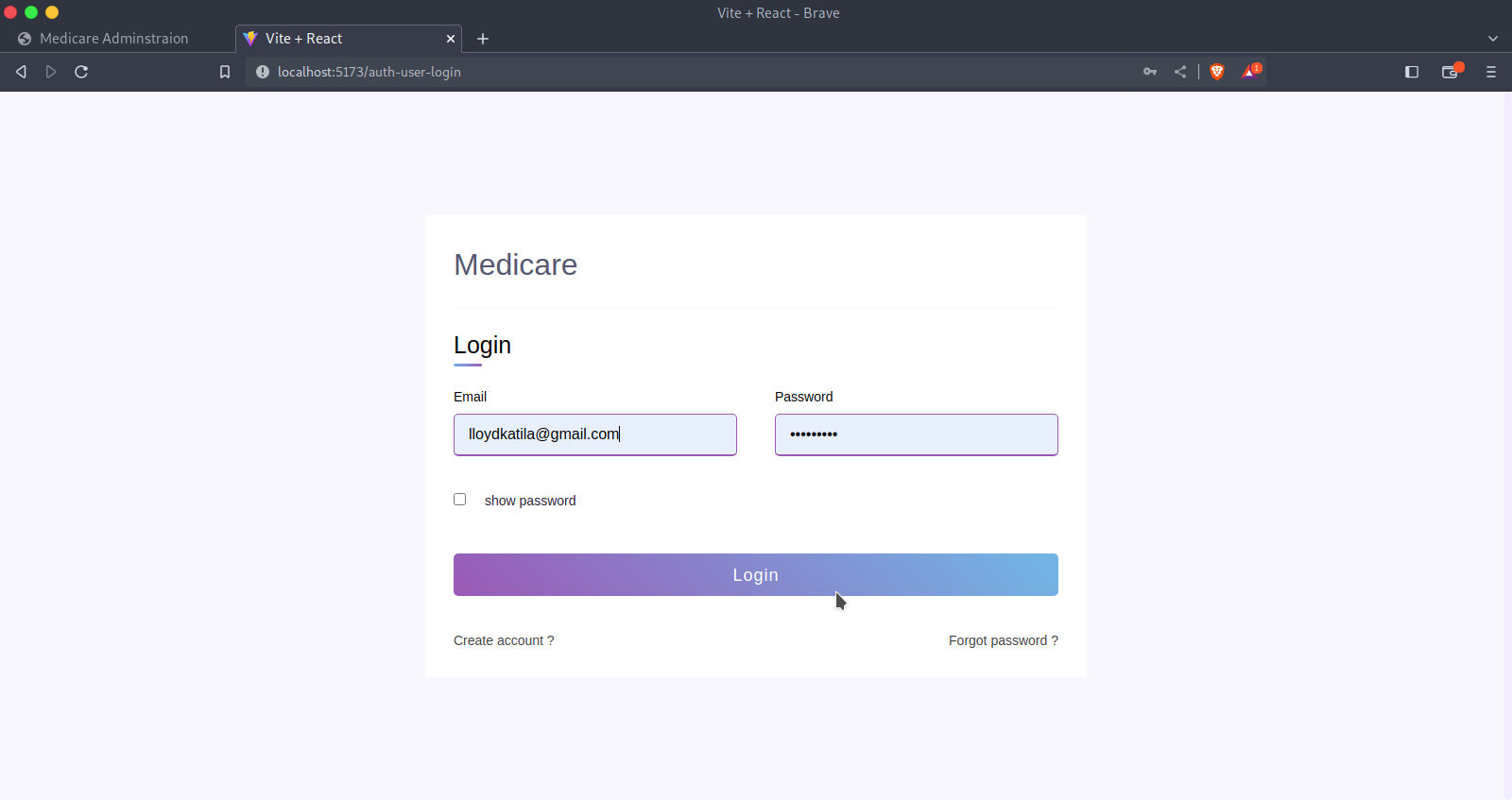
**USER INTERFACES AND SEGMENTED CODE**



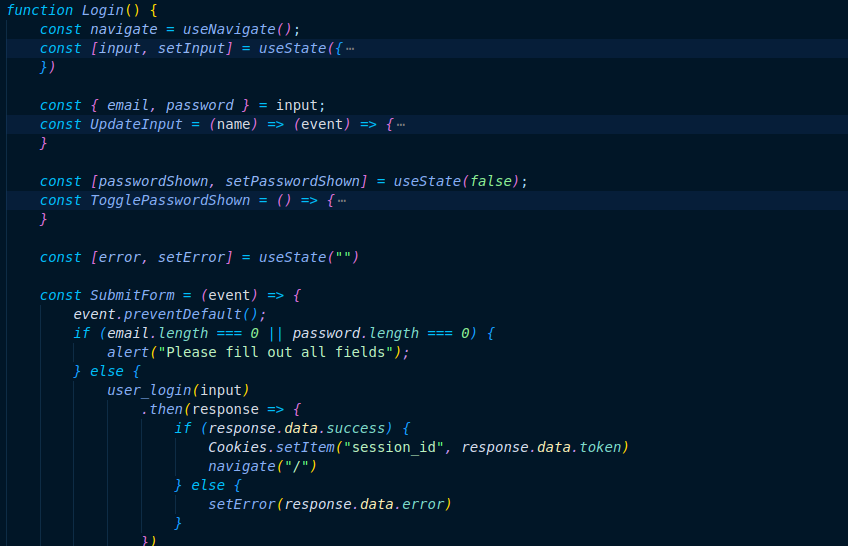
code :



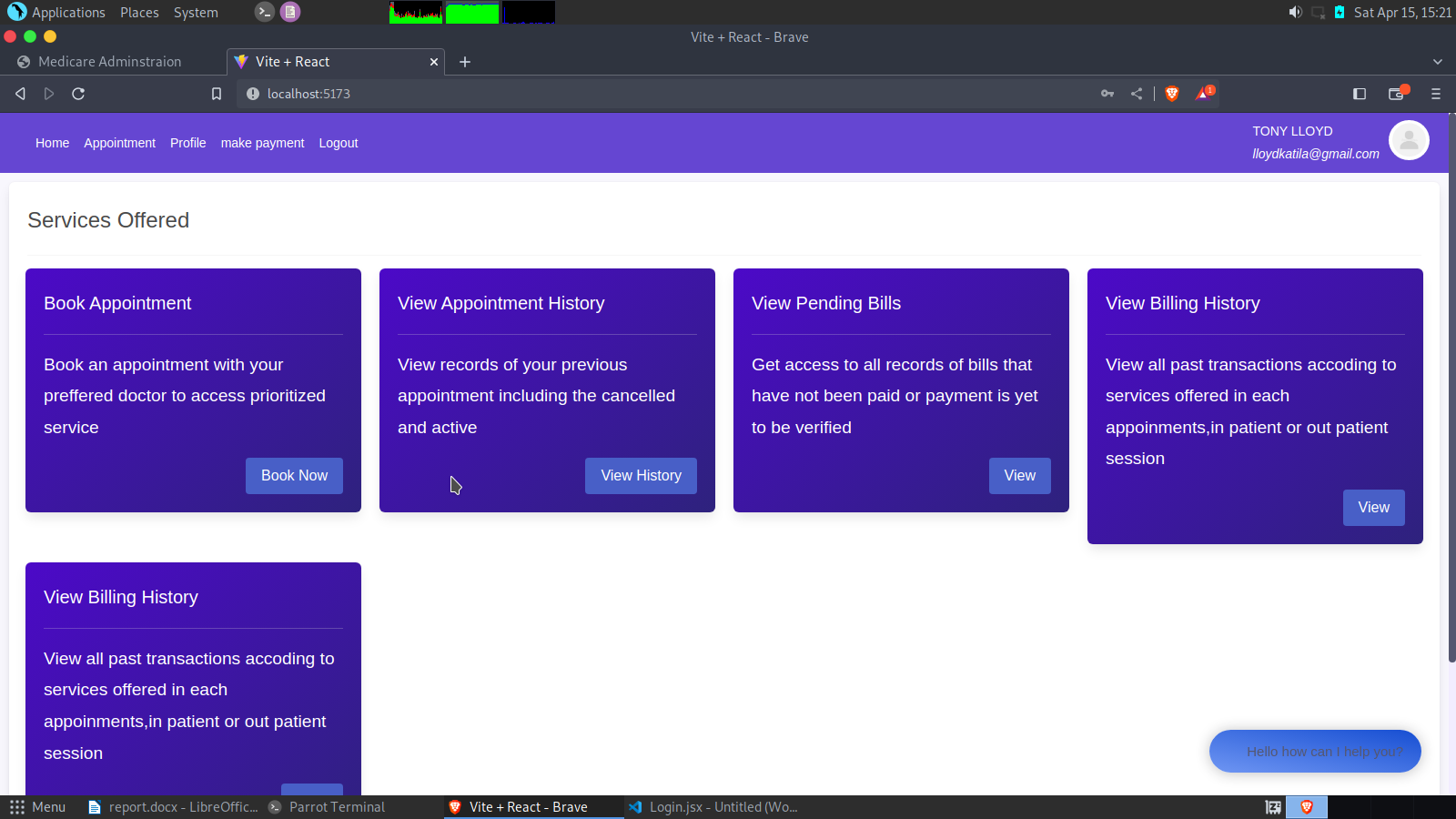
**login interface**



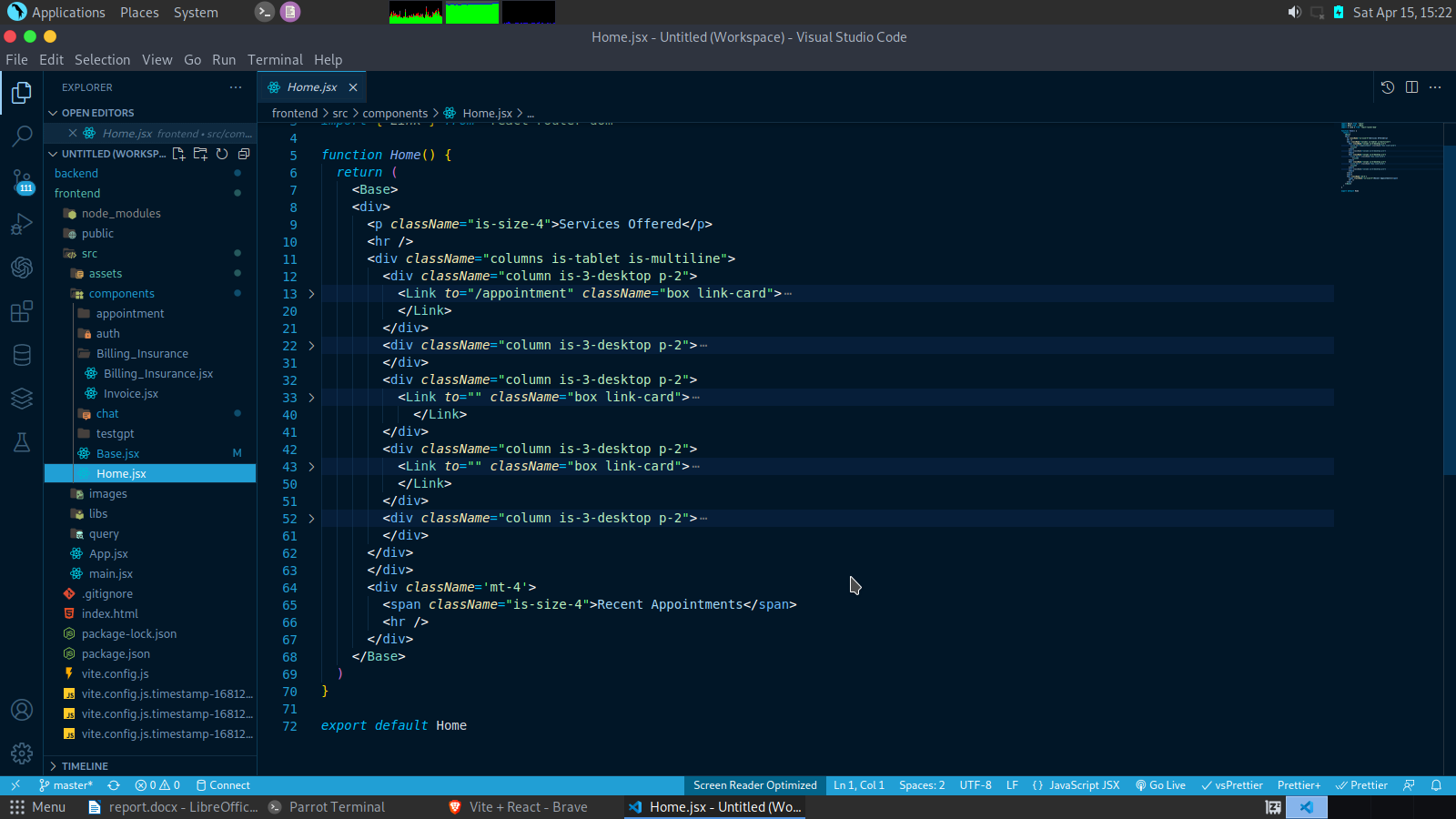
code:



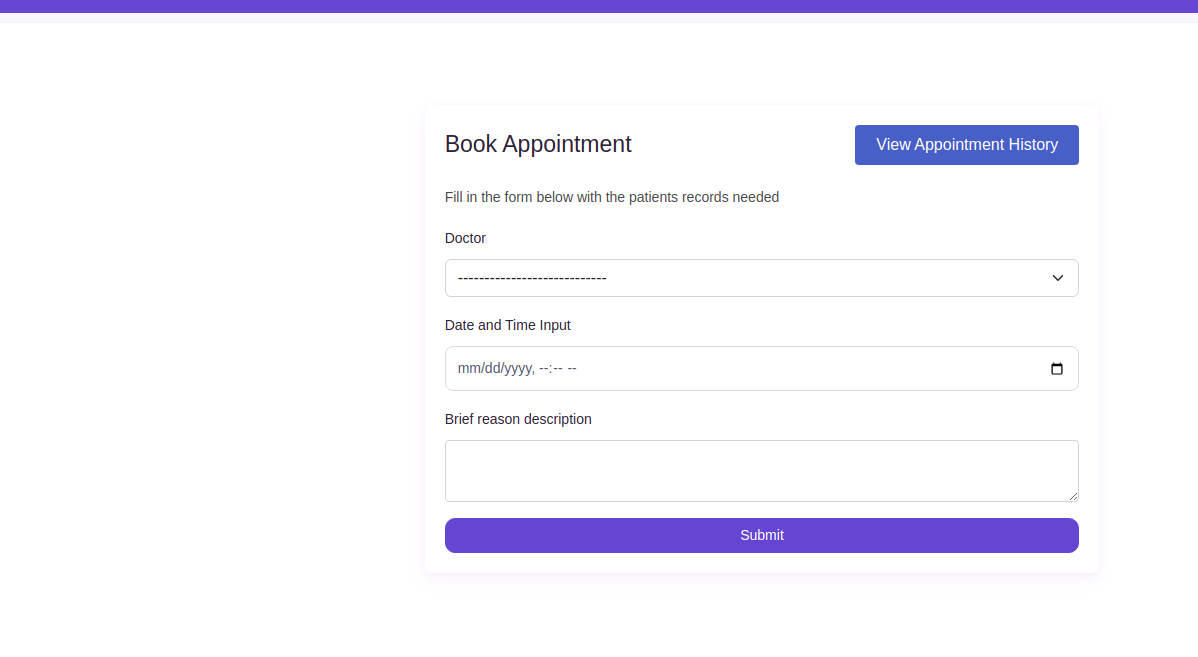
home page :

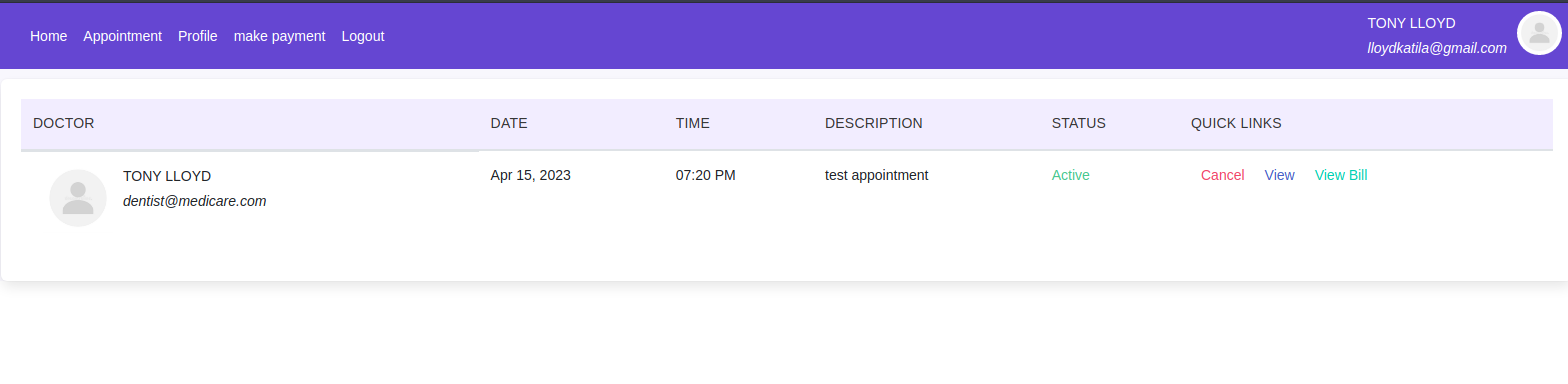


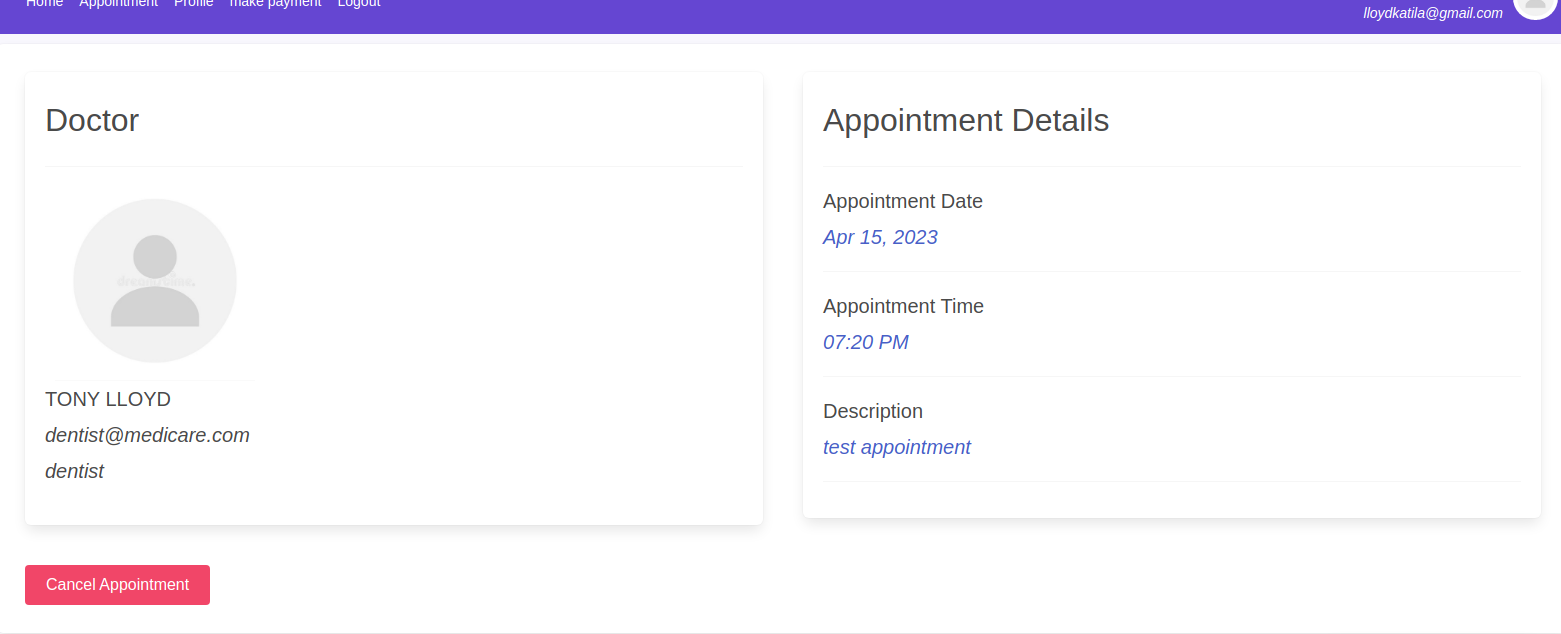
code for home :



book appointment interfaces :

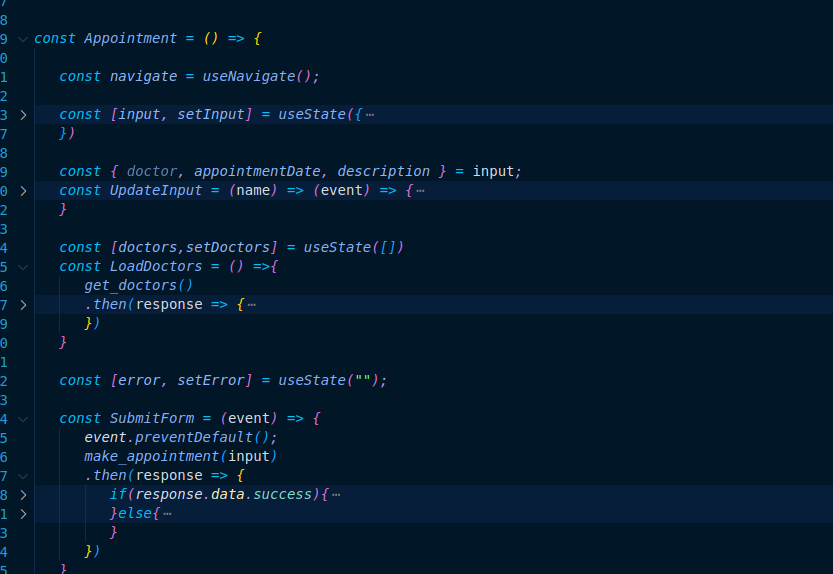




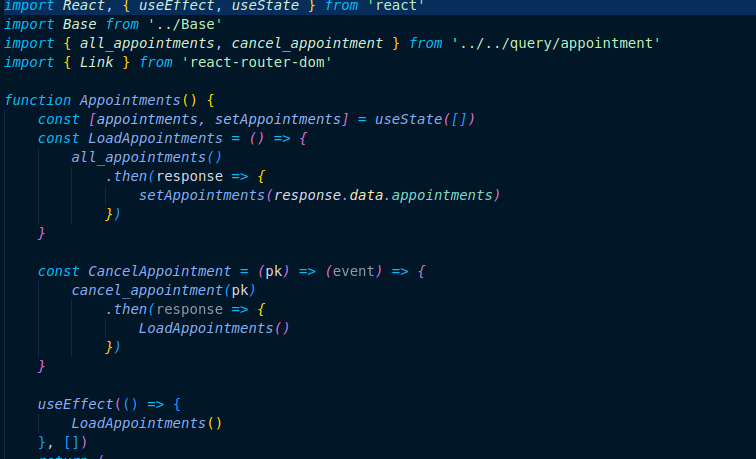


code :

a)book appointment



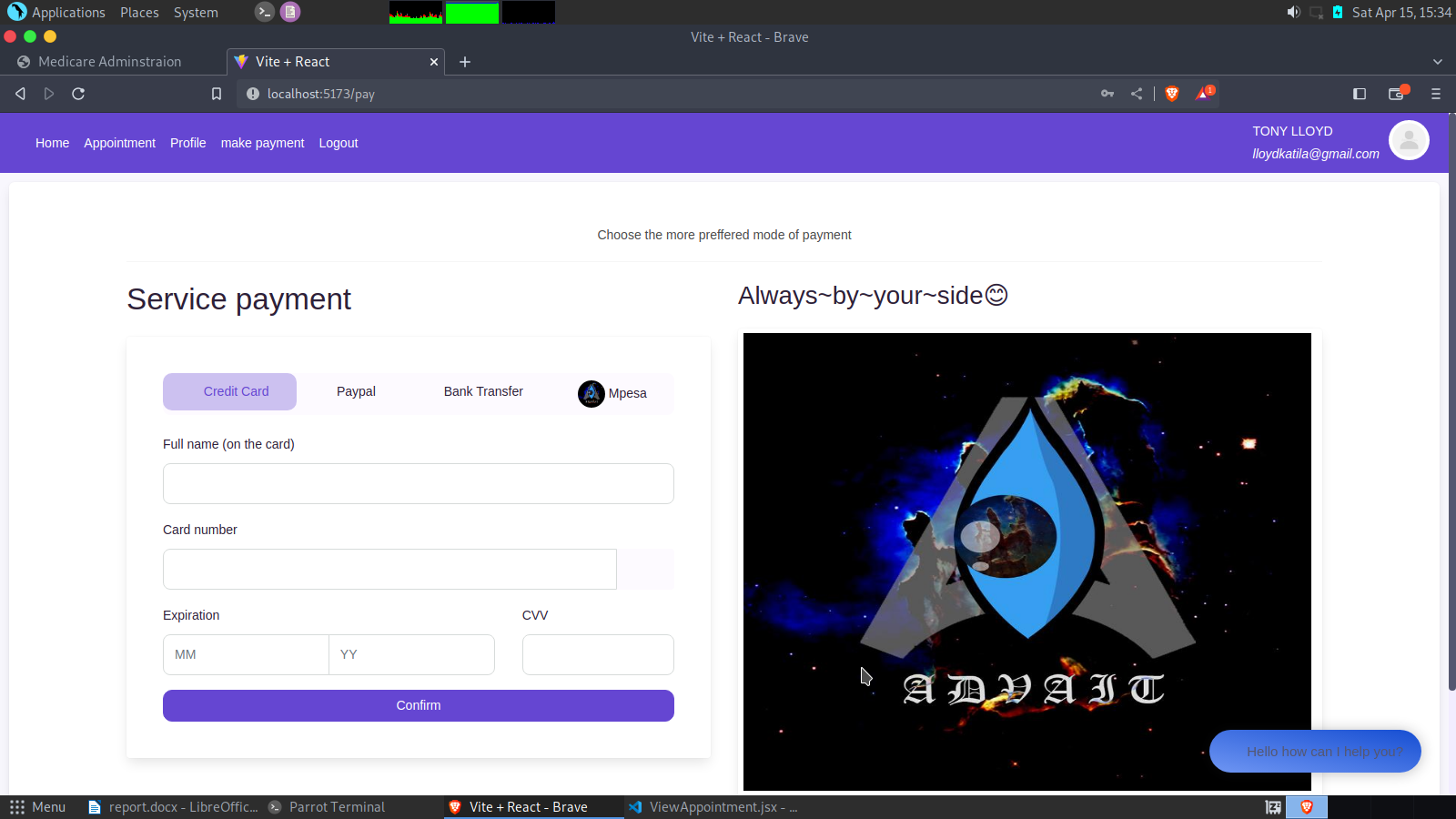
b) view booked appointment



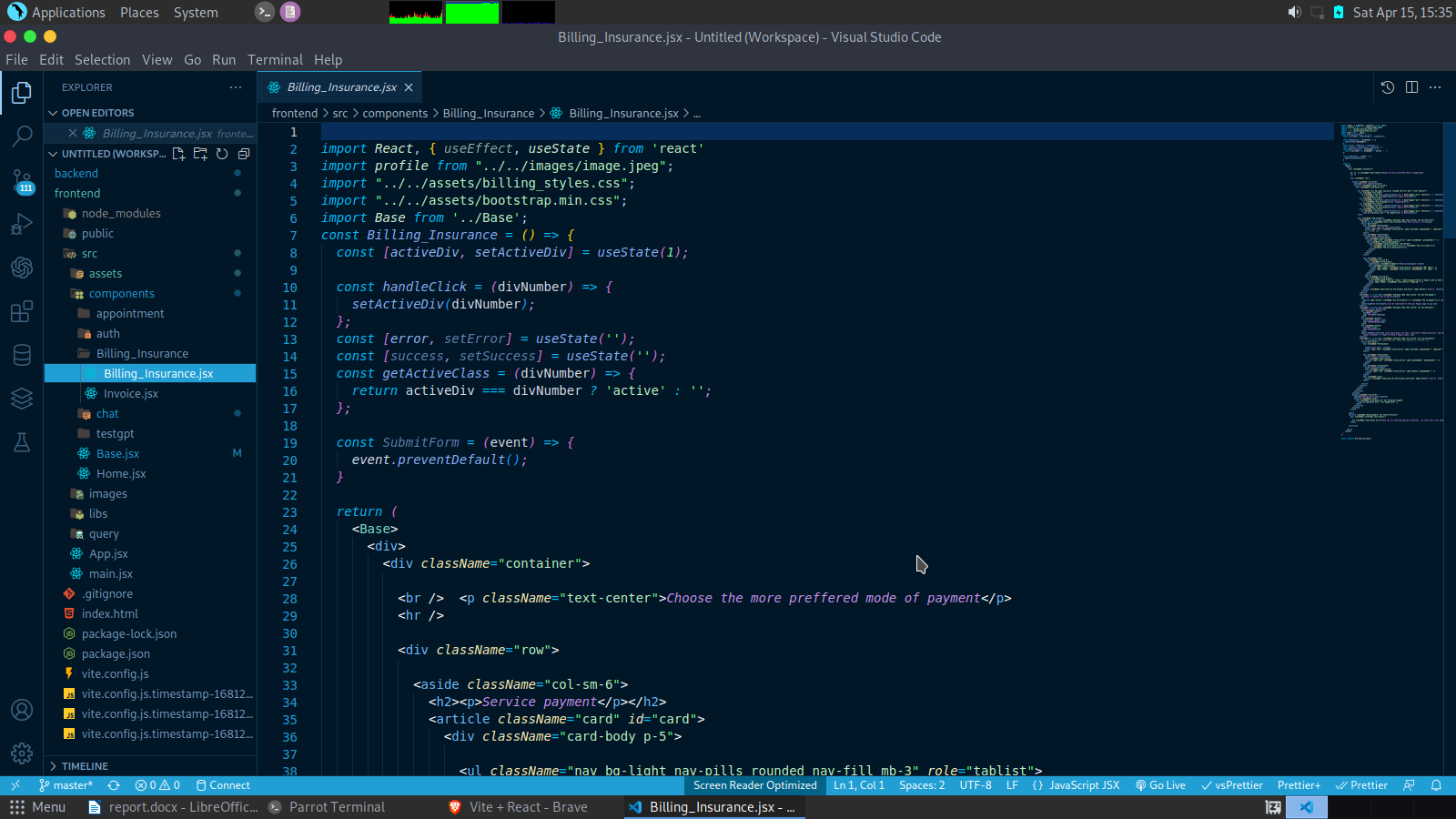
c)view single appointment



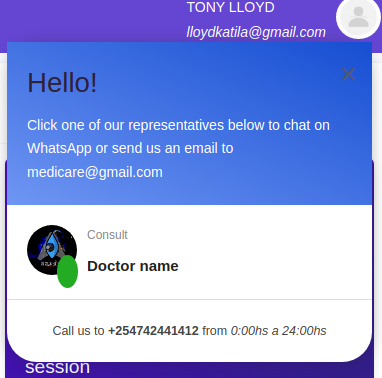
**Billing interfaces**

****

code:



**consult us module:**



code:

# **CHAPTER 5: REFERENCE**

1. The waterfall system development methodology:
   1. [https://business.adobe.com/blog/basics/waterfall#:~:text=The%20Waterfall%20methodology%20%E2%80%94%20also%20known,before%20the%20next%20phase%20begins](https://business.adobe.com/blog/basics/waterfall" \l ":~:text=The Waterfall methodology — also known,before the next phase begins)
   2. <https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model>