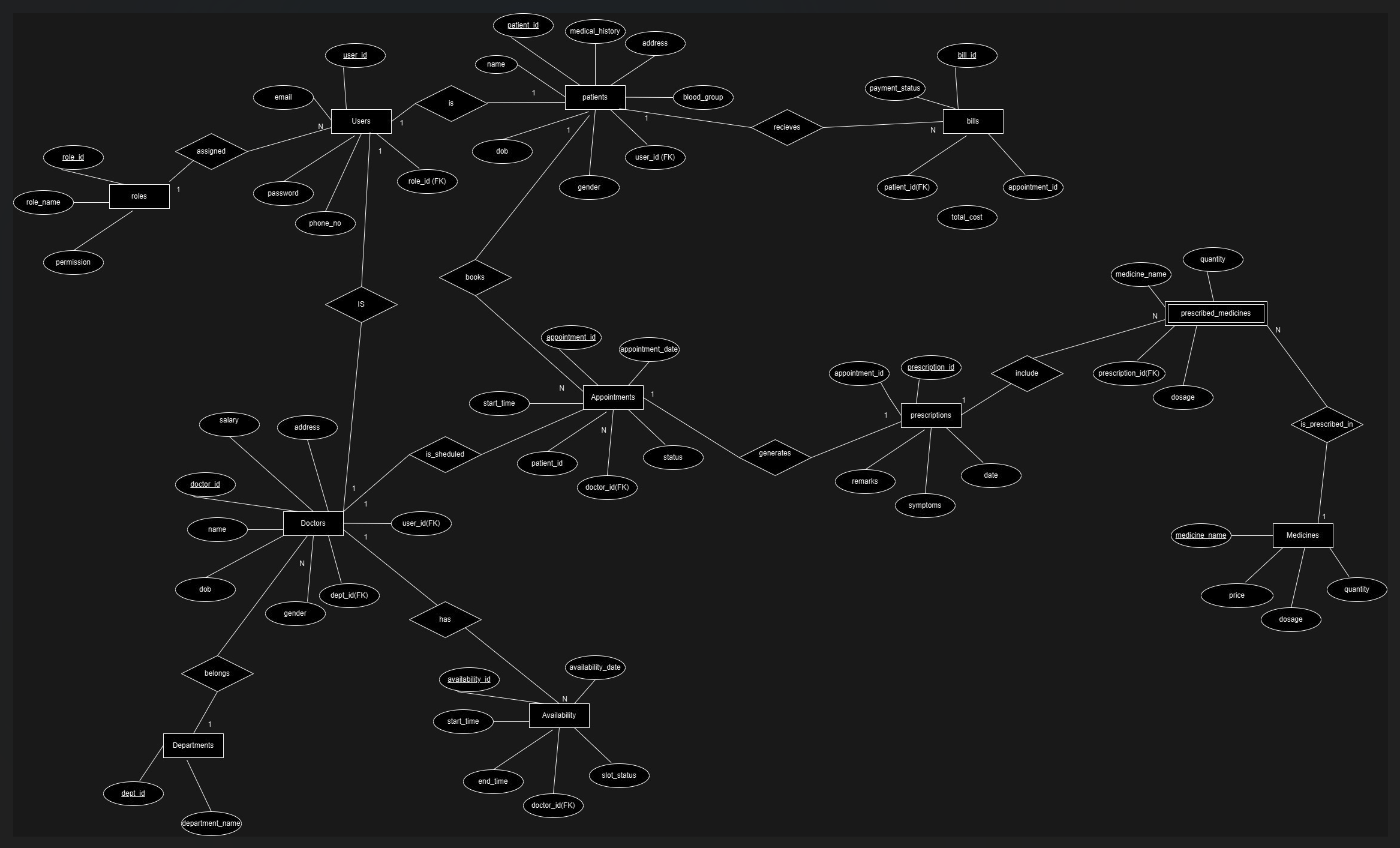
DBMS : ASSIGNMENT 3

Deepan S

CB.EN.U4AIE22117

ER – DIAGRAM:



Code :

We have main App.py :

import streamlit as st

from auth import validate\_login, signup\_patient

from utils import load\_image, is\_valid\_email

from dashboard import patient\_dashboard, doctor\_dashboard, manager\_dashboard  # Import patient\_dashboard function

import datetime

today = datetime.date.today()

# Load the logo image

image\_data = load\_image('logo.png')  # Path to logo

# Custom CSS

from styles.custom\_css import apply\_custom\_css

apply\_custom\_css()  # Call the function to apply CSS

# Initialize session state for login

if 'logged\_in' not in st.session\_state:

    st.session\_state.logged\_in = False

    st.session\_state.role\_id = None

    st.session\_state.email = None

# Header Section

st.markdown(f"""

<div class="header">

    <div style="display: flex; align-items: center; color: blue;">

        <img src="data:image/png;base64,{image\_data}" alt="Health Hub Logo" style="width: 120px; margin-right: 20px;">

        <div>

            <h1 style="margin: 0;">Health Hub</h1>

            <p class="custom-tagline">Your Trusted Partner in Comprehensive Healthcare Solutions</p>

        </div>

    </div>

</div>

""", unsafe\_allow\_html=True)

# Sidebar for navigation

st.sidebar.title("Navigation")

page = st.sidebar.radio("Go to", ("Home", "Login"))

# Main Content

if page == "Home":

    st.markdown('<div class="content"><h2 style="color: #007bff;">Welcome to Health Hub!</h2><p>Your health is our priority. Explore our services tailored for patients, doctors, and healthcare managers. Join us in taking charge of your health today!</p></div>', unsafe\_allow\_html=True)

    # Additional Information Section

    st.markdown("""

        <div class="content">

            <h3 style="color: #007bff;" >What You Can Do:</h3>

            <ul>

                <li><strong>Patients:</strong> Schedule, view, and cancel appointments.</li>

                <li><strong>Doctors:</strong> Manage appointments, review patient records.</li>

                <li><strong>Managers:</strong> Oversee staff, manage resources.</li>

            </ul>

        </div>

    """, unsafe\_allow\_html=True)

elif page == "Login":

    st.markdown('<div class="content"><h2>Login</h2></div>', unsafe\_allow\_html=True)

    # If the user is not logged in, show the login form

    if not st.session\_state.logged\_in:

        email = st.text\_input('Email')

        password = st.text\_input('Password', type='password')

        if st.button('Login'):

            user = validate\_login(email, password)

            if user:

                role\_id = user[3]  # Assuming role\_id is the fourth column

                st.session\_state.logged\_in = True

                st.session\_state.role\_id = role\_id

                st.session\_state.email = email

                st.success("Login successful!")

                # No need to immediately display the dashboard here

                # It will be handled in the next session state check below

            else:

                st.error("Invalid credentials.")

    # If the user is logged in and role is 'Patient', show patient dashboard

    if st.session\_state.logged\_in and st.session\_state.role\_id == 1:

        patient\_dashboard(st.session\_state.email)

    elif st.session\_state.logged\_in and st.session\_state.role\_id == 2:

        doctor\_dashboard(st.session\_state.email)

    elif st.session\_state.logged\_in and st.session\_state.role\_id == 3:

        manager\_dashboard()

    # Show the signup form only if the user is not logged in

    if not st.session\_state.logged\_in:

        selected\_role = st.radio("Would you like to sign up as a patient?", ["Yes", "No"])

        if selected\_role == 'Yes':

            st.markdown('<h2 style="color: #007bff;">Patient Signup</h2>', unsafe\_allow\_html=True)

            with st.form(key='signup\_form', clear\_on\_submit=True):

                name = st.text\_input('Full Name')

                email = st.text\_input('Email')

                password = st.text\_input('Password', type='password')

                date\_of\_birth = st.date\_input('Date of Birth', min\_value=datetime.date(1900, 1, 1), max\_value=today)

                gender = st.selectbox('Gender', ['Male', 'Female', 'Other'])

                medical\_history = st.text\_area('Medical History')

                blood\_group = st.selectbox('Blood Group', ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'])

                address = st.text\_input('Address')

                signup\_button = st.form\_submit\_button(label='Sign Up')

                if signup\_button:

                    if not name or not email or not password or not address:

                        st.error("Full Name, Email, Password, and Address are required fields.")

                    elif not date\_of\_birth or not gender:

                        st.error("Date of Birth and Gender are required fields.")

                    elif not is\_valid\_email(email):

                        st.error("Please enter a valid email address.")

                    else:

                        success, message = signup\_patient(name, email, password, 1, medical\_history, date\_of\_birth, gender, blood\_group, address)

                        if success:

                            st.success("Sign-up successful!")

                        else:

                            st.error(f"Sign-up failed: {message}")

# Footer

st.markdown('<footer><div class="footer">© 2024 Health Hub. All rights reserved.</div></footer>', unsafe\_allow\_html=True)

Auth.py where we will have helper function to interact with the database:

from database import get\_db\_connection

from datetime import datetime

def validate\_login(email, password):

    """Validate user login credentials."""

    conn = get\_db\_connection()

    cursor = conn.cursor()

    query = "SELECT \* FROM users WHERE email=%s AND password=%s"

    cursor.execute(query, (email, password))

    result = cursor.fetchone()

    cursor.close()

    conn.close()

    return result

def signup\_patient(name, email, password, role\_id, medical\_history, date\_of\_birth, gender, blood\_group, address):

    """Register a new patient."""

    conn = get\_db\_connection()

    cursor = conn.cursor()

    try:

        # Insert into User table

        cursor.execute("INSERT INTO users (email, password, role\_id) VALUES (%s, %s, %s) RETURNING user\_id",

                       (email, password, role\_id))

        user\_id = cursor.fetchone()[0]

        # Insert into Patient table

        cursor.execute("INSERT INTO patients (user\_id, name, medical\_history, date\_of\_birth, gender, blood\_group, address) VALUES (%s, %s, %s, %s, %s, %s, %s)",

                       (user\_id, name, medical\_history, date\_of\_birth, gender, blood\_group, address))

        conn.commit()

        return True, "User registered successfully."

    except Exception as e:

        conn.rollback()

        return False, str(e)

    finally:

        cursor.close()

        conn.close()

def get\_patient\_details(email):

    """Fetch patient details using the user's email."""

    conn = get\_db\_connection()

    cursor = conn.cursor()

    query = """

    SELECT p.name, p.medical\_history, p.date\_of\_birth, p.gender, p.blood\_group, p.address

    FROM patients p

    JOIN users u ON p.user\_id = u.user\_id

    WHERE u.email = %s

    """

    cursor.execute(query, (email,))

    result = cursor.fetchone()

    cursor.close()

    conn.close()

    return result

def get\_doctors():

    try:

        conn = get\_db\_connection()

        cursor = conn.cursor()

        query = """

            SELECT d.doctor\_id, d.name, dept.department\_name

            FROM doctors d

            JOIN departments dept ON d.dept\_id = dept.dept\_id

            """

            # Execute the query and fetch all rows

        cursor.execute(query)

        doctors = cursor.fetchall()

        # Close the cursor and connection

        cursor.close()

        conn.close()

        return doctors

    except Exception as e:

        print(f"Error fetching doctors: {e}")

        return []

def get\_available\_slots(doctor\_id, appointment\_date):

    conn = get\_db\_connection()

    cursor = conn.cursor()

    # Query to fetch available slots

    query = """

    SELECT a.start\_time, a.end\_time

    FROM availability a

    WHERE a.doctor\_id = %s

    AND a.available\_date = %s

    AND a.slot\_status = 'available'

    ORDER BY a.start\_time

    """

    cursor.execute(query, (doctor\_id, appointment\_date))

    available\_slots = cursor.fetchall()

    cursor.close()

    conn.close()

    return available\_slots

# Inside your patient\_dashboard function, update this line:

def book\_slot(patient\_id, doctor\_id, appointment\_date, start\_time):

    # Convert start\_time (datetime.time) to a time object if not already

    if isinstance(start\_time, str):

        start\_time = datetime.strptime(start\_time, "%H:%M").time()  # Adjust format as needed

    # Check for existing appointments that conflict

    if check\_conflict(patient\_id, doctor\_id, appointment\_date, start\_time):

        return False, "Booking failed: Time slot is already booked."

    # Logic to book the appointment (e.g., inserting into the database)

    try:

        connection = get\_db\_connection()

        cursor = connection.cursor()

        # Insert the appointment into the database

        insert\_query = """INSERT INTO appointments (patient\_id, doctor\_id, appointment\_date, start\_time, status)

                          VALUES (%s, %s, %s, %s, %s)"""

        cursor.execute(insert\_query, (patient\_id, doctor\_id, appointment\_date, start\_time, 'booked'))

        # Update slot status in availability (if required)

        update\_query = """UPDATE availability

                          SET slot\_status = 'booked'

                          WHERE doctor\_id = %s AND available\_date = %s AND start\_time = %s"""

        cursor.execute(update\_query, (doctor\_id, appointment\_date, start\_time))

        # Commit the transaction

        connection.commit()

        return True, "Appointment booked successfully."

    except Exception as e:

        return False, f"Error: {str(e)}"

    finally:

        if connection:

            cursor.close()

            connection.close()

def check\_conflict(patient\_id, doctor\_id, appointment\_date, start\_time):

    """Check for existing appointments that conflict with the requested time."""

    try:

        connection = get\_db\_connection()

        cursor = connection.cursor()

        # Query to check for overlapping appointments for the patient

        check\_query = """

        SELECT COUNT(\*) FROM appointments

        WHERE patient\_id = %s

          AND appointment\_date = %s

          AND start\_time = %s

        """

        cursor.execute(check\_query, (patient\_id, appointment\_date, start\_time))

        count = cursor.fetchone()[0]

        return count > 0  # Return True if there is a conflict

    except Exception as e:

        print(f"Error checking conflicts: {str(e)}")

        return True  # Assume conflict if there's an error

    finally:

        if connection:

            cursor.close()

            connection.close()

def get\_patient\_id(email):

    """Fetch the patient ID based on the user's email."""

    conn = get\_db\_connection()

    cursor = conn.cursor()

    # Query to fetch the patient\_id based on the user's email

    query = """

        SELECT patient\_id FROM patients

        JOIN users ON patients.user\_id = users.user\_id

        WHERE users.email = %s

    """

    cursor.execute(query, (email,))

    result = cursor.fetchone()

    cursor.close()

    conn.close()

    return result[0] if result else None  # Return patient\_id or None if not found

def get\_scheduled\_appointments(patient\_id):

    """Fetch all scheduled appointments for the logged-in patient."""

    conn = get\_db\_connection()

    cursor = conn.cursor()

    try:

        # Fetch all appointments for the logged-in patient

        cursor.execute("""

            SELECT a.appointment\_date, a.start\_time, a.status, d.name, a.appointment\_id

            FROM appointments a

            JOIN doctors d ON a.doctor\_id = d.doctor\_id

            WHERE a.patient\_id = %s

            ORDER BY a.appointment\_date, a.start\_time

        """, (patient\_id,))

        appointments = cursor.fetchall()

        return appointments  # Return the list of appointments

    except Exception as e:

        print(f"Error in get\_scheduled\_appointments: {e}")  # Log the error

        return []  # Return an empty list

    finally:

        cursor.close()

        conn.close()

def delete\_appointment(appointment\_id):

    """Delete an appointment from the database."""

    try:

        conn = get\_db\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT doctor\_id, appointment\_date, start\_time FROM appointments WHERE appointment\_id = %s", (appointment\_id,))

        result = cursor.fetchone()

        if result:

            doctor\_id, appointment\_date, start\_time = result

            cursor.execute("DELETE FROM appointments WHERE appointment\_id = %s", (appointment\_id,))

                # Update the availability table to set the status to 'available'

            cursor.execute("""

                UPDATE availability

                SET slot\_status = 'available'

                WHERE doctor\_id = %s AND available\_date = %s AND start\_time = %s

            """, (doctor\_id, appointment\_date, start\_time))

            conn.commit()  # Commit the transaction

            return True, "Appointment deleted and slot marked as available."

        else:

            return False, "Appointment not found."

    except Exception as e:

        return False, str(e)

    finally:

        if conn:

            cursor.close()

            conn.close()

def get\_completed\_appointments(patient\_id):

    # Connect to the database

    conn = get\_db\_connection()

    cursor = conn.cursor()

    # Query to get completed appointments for the given patient

    query = """

    SELECT a.appointment\_date, a.start\_time, a.status, d.name as doctor\_name, a.appointment\_id

    FROM appointments a

    JOIN doctors d ON a.doctor\_id = d.doctor\_id

    WHERE a.patient\_id = %s AND a.status = 'completed'

    ORDER BY a.appointment\_date DESC;

    """

    cursor.execute(query, (patient\_id,))

    completed\_appointments = cursor.fetchall()

    cursor.close()

    conn.close()

    return completed\_appointments

def get\_doctor\_details(email):

    connection = get\_db\_connection()

    with connection.cursor() as cursor:

        cursor.execute("SELECT doctor\_id, name FROM doctors WHERE user\_id = (SELECT user\_id FROM users WHERE email = %s)", (email,))

        return cursor.fetchone()

def change\_appointment\_status(appointment\_id, status):

    try:

        # Database operations

        conn = get\_db\_connection()

        cursor = conn.cursor()

        print(f"Updating appointment ID: {appointment\_id} to status: {status}")  # Debug

        # Update the appointment status in the database

        cursor.execute("UPDATE appointments SET status = %s WHERE appointment\_id = %s", (status, appointment\_id))

        conn.commit()  # Commit the changes

        print("Appointment status updated.")  # Debug

        return True, "Appointment status updated successfully."

    except Exception as e:

        print(f"Error occurred in change\_appointment\_status: {e}")  # Debug

        return False, str(e)

def get\_upcoming\_appointments(doctor\_id):

    """Fetch upcoming appointments for the doctor."""

    connection = get\_db\_connection()

    with connection.cursor() as cursor:

        cursor.execute("""

            SELECT a.appointment\_date, a.start\_time, p.name, a.status, a.appointment\_id

            FROM appointments a

            JOIN patients p ON a.patient\_id = p.patient\_id

            WHERE a.doctor\_id = %s

              AND a.appointment\_date >= CURRENT\_DATE

              AND a.status != 'completed'  -- Exclude completed appointments

            ORDER BY a.appointment\_date, a.start\_time

        """, (doctor\_id,))

        return cursor.fetchall()

def get\_completed\_appointments(doctor\_id):

    """Fetch completed appointments for the doctor."""

    connection = get\_db\_connection()

    with connection.cursor() as cursor:

        cursor.execute("""

            SELECT a.appointment\_date, a.start\_time, p.name, a.status, a.appointment\_id

            FROM appointments a

            JOIN patients p ON a.patient\_id = p.patient\_id

            WHERE a.doctor\_id = %s AND a.status = 'completed'

            ORDER BY a.appointment\_date DESC

        """, (doctor\_id,))

        return cursor.fetchall()

def add\_doctor(name, dob, gender, address, email, password, dept\_id):

    try:

        # Database logic to add doctor

        conn = get\_db\_connection()

        cursor = conn.cursor()

        # First, create a user entry

        cursor.execute("INSERT INTO users (email, password, role\_id) VALUES (%s, %s, %s) RETURNING user\_id",

                       (email, password, 2))  # role\_id 2 for doctors

        user\_id = cursor.fetchone()[0]  # Fetch the generated user\_id

        # Now, insert the doctor record

        cursor.execute("INSERT INTO doctors (name, dob, gender, address, user\_id, dept\_id) VALUES (%s, %s, %s, %s, %s, %s)",

                       (name, dob, gender, address, user\_id, dept\_id))

        conn.commit()

        return True, "Doctor added successfully."

    except Exception as e:

        conn.rollback()

        return False, str(e)

def remove\_doctor(doctor\_id):

    try:

        # Get the user\_id associated with the doctor

        conn = get\_db\_connection()

        cursor = conn.cursor()

        # Fetch the user\_id for the given doctor\_id

        cursor.execute("SELECT user\_id FROM doctors WHERE doctor\_id = %s", (doctor\_id,))

        user\_id = cursor.fetchone()

        if user\_id:

            user\_id = user\_id[0]  # Extract the user\_id from the result

            # Remove the doctor

            cursor.execute("DELETE FROM doctors WHERE doctor\_id = %s", (doctor\_id,))

            # Remove the corresponding user

            cursor.execute("DELETE FROM users WHERE user\_id = %s", (user\_id,))

            conn.commit()

            return True, "Doctor and corresponding user removed successfully."

        else:

            return False, "Doctor not found."

    except Exception as e:

        conn.rollback()

        return False, str(e)

def get\_departments():

    conn = get\_db\_connection()

    try:

        cursor = conn.cursor()

        cursor.execute("SELECT dept\_id, department\_name FROM departments")

        return cursor.fetchall()  # This will return a list of tuples (department\_id, department\_name)

    except Exception as e:

        print(f"An error occurred while fetching departments: {e}")

        return []  # Return an empty list in case of error

    finally:

        conn.close()  # Ensure the connection is closed

We shall have our dashboard.py which help the respective to interact with his/her functionalities:

import streamlit as st

from auth import get\_patient\_details, get\_patient\_id, get\_available\_slots, book\_slot, get\_scheduled\_appointments, delete\_appointment

from auth import get\_doctor\_details, get\_upcoming\_appointments, get\_completed\_appointments, change\_appointment\_status, get\_doctors

from auth import add\_doctor, remove\_doctor, get\_departments

from datetime import date

import datetime

today = datetime.date.today()

def patient\_dashboard(email):

    # Title and Introduction

    st.markdown('<h2 style="color: #007bff; text-align: center;">Patient Dashboard</h2>', unsafe\_allow\_html=True)

    # Fetch and display patient details

    patient\_details = get\_patient\_details(email)

    if patient\_details:

        name, medical\_history, date\_of\_birth, gender, blood\_group, address = patient\_details

        medical\_history = medical\_history if medical\_history else "None"

        st.markdown(f""" <div style="background-color: #f0f8ff; padding: 10px; border-radius: 10px;">

        <h3 style="color: #343a40;">Welcome back, {name}!</h3>

        <p style="color: black;">Email: {email}</p>

        <p style="color: black;"><strong>Date of Birth:</strong> {date\_of\_birth}</p>

        <p style="color: black;"><strong>Gender:</strong> {gender}</p>

        <p style="color: black;"><strong>Blood Group:</strong> {blood\_group}</p>

        <p style="color: black;"><strong>Address:</strong> {address}</p>

        <p style="color: black;"><strong>Medical History:</strong> {medical\_history}</p>

        </div> """, unsafe\_allow\_html=True)

    else:

        st.error("No patient details found.")

        return

    # Horizontal divider

    st.markdown("<hr>", unsafe\_allow\_html=True)

    # Appointment Booking Section

    st.markdown('<h3 style="color: #007bff;">Book an Appointment</h3>', unsafe\_allow\_html=True)

    # Doctor list with specializations

    doctors = get\_doctors()

    doctor\_dict = {doctor[0]: f"{doctor[1]} - {doctor[2]}" for doctor in doctors}

    # Select Doctor and Date

    doctor\_id = st.selectbox("Choose your doctor", options=list(doctor\_dict.keys()),

                             format\_func=lambda x: doctor\_dict[x])  # Show names and specializations

    appointment\_date = st.date\_input("Select appointment date", min\_value=date.today())

    # Fetch available slots based on selected doctor and date

    available\_slots = get\_available\_slots(doctor\_id, appointment\_date)

    if available\_slots:

        # Format the available slots for selection

        formatted\_slots = [(slot[0].strftime("%H:%M"), slot[1].strftime("%H:%M")) for slot in available\_slots]

        selected\_slot = st.selectbox("Available Time Slots", options=formatted\_slots,

                                      format\_func=lambda x: f"{x[0]} to {x[1]}")

        if st.button("Book Appointment", key="book\_btn"):

            with st.spinner("Booking your appointment..."):

                patient\_id = get\_patient\_id(email)  # Fetch patient ID based on email

                if not patient\_id:

                    st.error("Error: Patient ID not found.")

                    return

                # Find the corresponding start\_time from the selected slot

                start\_time = next((slot[0] for slot in available\_slots if slot[0].strftime("%H:%M") == selected\_slot[0]), None)

                if not start\_time:

                    st.error("Error: Start time not found for selected slot.")

                    return

                success, message = book\_slot(patient\_id, doctor\_id, appointment\_date, start\_time)  # Pass the patient\_id

                if success:

                    st.success(f"Appointment booked for {appointment\_date} at {selected\_slot[0]}.")

                else:

                    st.error(f"Booking failed: {message}")

    else:

        st.warning("No available slots for this doctor on the selected date.")

    # Horizontal divider

    st.markdown("<hr>", unsafe\_allow\_html=True)

    # Display Scheduled Appointments

    st.subheader("Your Scheduled Appointments")

    patient\_id = get\_patient\_id(email)

    appointments = get\_scheduled\_appointments(patient\_id)

    if appointments:

        for appointment in appointments:

            st.write("Debug: Appointment data:", appointment)  # Display the full appointment tuple for debugging

            # Try to adjust based on actual structure of appointment

            try:

                appointment\_date = appointment[0]

                start\_time = appointment[1]

                status = appointment[2]

                doctor\_name = appointment[3]

                appointment\_id = appointment[4]  # Check if appointment\_id is in position 4

                st.markdown(f"""

                    <div style="background-color: #e0f7fa; padding: 10px; margin-bottom: 10px; border-radius: 5px;">

                        <p style="color: black;">Doctor: {doctor\_name}</p>

                        <p style="color: black;">Date: {appointment\_date}</p>

                        <p style="color: black;">Time: {start\_time}</p>

                        <p style="color: black;">Status: {status}</p>

                    </div>

                """, unsafe\_allow\_html=True)

                # Add a button for deletion of this appointment

                if st.button(f"Cancel Appointment with {doctor\_name} on {appointment\_date} at {start\_time}", key=f"del\_btn\_{appointment\_id}"):

                    with st.spinner("Deleting your appointment..."):

                        success, message = delete\_appointment(appointment\_id)

                        if success:

                            st.success(f"Appointment with {doctor\_name} on {appointment\_date} at {start\_time} successfully deleted.")

                        else:

                            st.error(f"Failed to delete appointment: {message}")

            except IndexError:

                st.error("Error: Appointment data structure is not as expected.")

    else:

        st.write("No scheduled appointments found or an error occurred.")

        # Horizontal divider

        st.markdown("<hr>", unsafe\_allow\_html=True)

        # Display Completed Appointments

    st.subheader("Completed Appointments")

    completed\_appointments = get\_completed\_appointments(patient\_id)

    if completed\_appointments:

        for appointment in completed\_appointments:

            appointment\_date = appointment[0]

            start\_time = appointment[1]

            doctor\_name = appointment[3]

            st.markdown(f"""

            <div style="background-color: #d1f0d1; padding: 10px; margin-bottom: 10px; border-radius: 5px;">

                <p style="color: black;">Doctor: {doctor\_name}</p>

                <p style="color: black;">Date: {appointment\_date}</p>

                <p style="color: black;">Time: {start\_time}</p>

            </div>

            """, unsafe\_allow\_html=True)

    else:

        st.write("No completed appointments found.")

def doctor\_dashboard(email):

    doctor\_details = get\_doctor\_details(email)

    if not doctor\_details:

        st.error("Doctor not found.")

        return

    doctor\_id = doctor\_details[0]

    st.title(f"Welcome back, {doctor\_details[1]}!")

    # Fetch upcoming appointments

    upcoming\_appointments = get\_upcoming\_appointments(doctor\_id)

    completed\_appointments = get\_completed\_appointments(doctor\_id)

    # Display Upcoming Appointments

    st.subheader("Upcoming Appointments")

    if upcoming\_appointments:

        for appointment in upcoming\_appointments:

            appointment\_date, start\_time, patient\_name, status, appointment\_id = appointment

            st.write(f"Date: {appointment\_date}, Time: {start\_time}, Patient: {patient\_name}, Status: {status}")

            if st.button(f"Mark as Completed: {appointment\_id}"):

                st.write("Mark as Completed button clicked.")  # Debug

                # Change the appointment status to completed

                success, message = change\_appointment\_status(appointment\_id, 'completed')

                if success:

                    st.success(f"Appointment marked as completed.")

                else:

                    st.error(f"Failed to mark appointment as completed: {message}")

    else:

        st.write("No upcoming appointments found.")

    # Completed Appointments Section

    st.subheader("Completed Appointments")

    if completed\_appointments:

        for appointment in completed\_appointments:

            appointment\_date, start\_time, patient\_name, status, appointment\_id = appointment

            st.markdown(f"""

                    <div style="background-color: #e0f7fa; padding: 10px; margin-bottom: 10px; border-radius: 5px;">

                        <p style="color: black;">Patient\_name: {patient\_name}</p>

                        <p style="color: black;">Date: {appointment\_date}</p>

                        <p style="color: black;">Time: {start\_time}</p>

                        <p style="color: black;">Status: {status}</p>

                    </div>

                """, unsafe\_allow\_html=True)

    else:

        st.write("No completed appointments found.")

def manager\_dashboard():

    st.title("Manager Dashboard")

    st.write("Welcome back, Admin!")

    # Option to select action

    option = st.selectbox("Choose an action", ["Add Doctor", "Remove Doctor"])

    if option == "Add Doctor":

        with st.form(key='add\_doctor\_form'):

            name = st.text\_input("Name")

            dob = st.date\_input('Date of Birth', min\_value=datetime.date(1900, 1, 1), max\_value=today)

            gender = st.selectbox("Gender", ["Male", "Female", "Other"])

            address = st.text\_area("Address")

            email = st.text\_input("Email")

            password = st.text\_input("Password", type="password")

            departments = get\_departments()

            dept\_dict = {dept[0]: dept[1] for dept in departments}  # dept[0] is department\_id, dept[1] is department\_name

            # Select department by name

            dept\_id = st.selectbox("Department", options=list(dept\_dict.keys()), format\_func=lambda x: dept\_dict[x])

            submit\_button = st.form\_submit\_button("Add Doctor")

            if submit\_button:

                success, message = add\_doctor(name, dob, gender, address, email, password, dept\_id)

                if success:

                    st.success(message)

                else:

                    st.error(f"Failed to add doctor: {message}")

    elif option == "Remove Doctor":

        # Fetch the list of doctors to remove

        doctors = get\_doctors()  # Assuming this returns a list of tuples (doctor\_id, name, specialization)

        doctor\_dict = {doctor[0]: f"{doctor[1]} - {doctor[2]}" for doctor in doctors}

        doctor\_id\_to\_remove = st.selectbox("Select Doctor to Remove", options=list(doctor\_dict.keys()),

                                            format\_func=lambda x: doctor\_dict[x])  # Show names and specializations

        if st.button("Remove Doctor"):

            with st.spinner("Removing doctor..."):

                success, message = remove\_doctor(doctor\_id\_to\_remove)  # Call the remove\_doctor function

                if success:

                    st.success(message)

                else:

                    st.error(f"Failed to remove doctor: {message}")

Then a database.py file to establish a connection and use it in our functions to interact with our database:

import psycopg2

def get\_db\_connection():

    return psycopg2.connect(

        host="localhost",

        database="hospital",

        user="postgres",

        password="g0916032p"

    )

Then we also have a small utils.py file where I have stored some functions for email verification and so on :

import base64

import re

email\_pattern = r'^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

def is\_valid\_email(email):

    return re.match(email\_pattern, email) is not None

def load\_image(image\_file):

    with open(image\_file, "rb") as image:

        return base64.b64encode(image.read()).decode()

I also do have a small css file which the styles of certain markdown depends on but I think it not necessary to include in this. I have only included the main logic and functionalities designed.