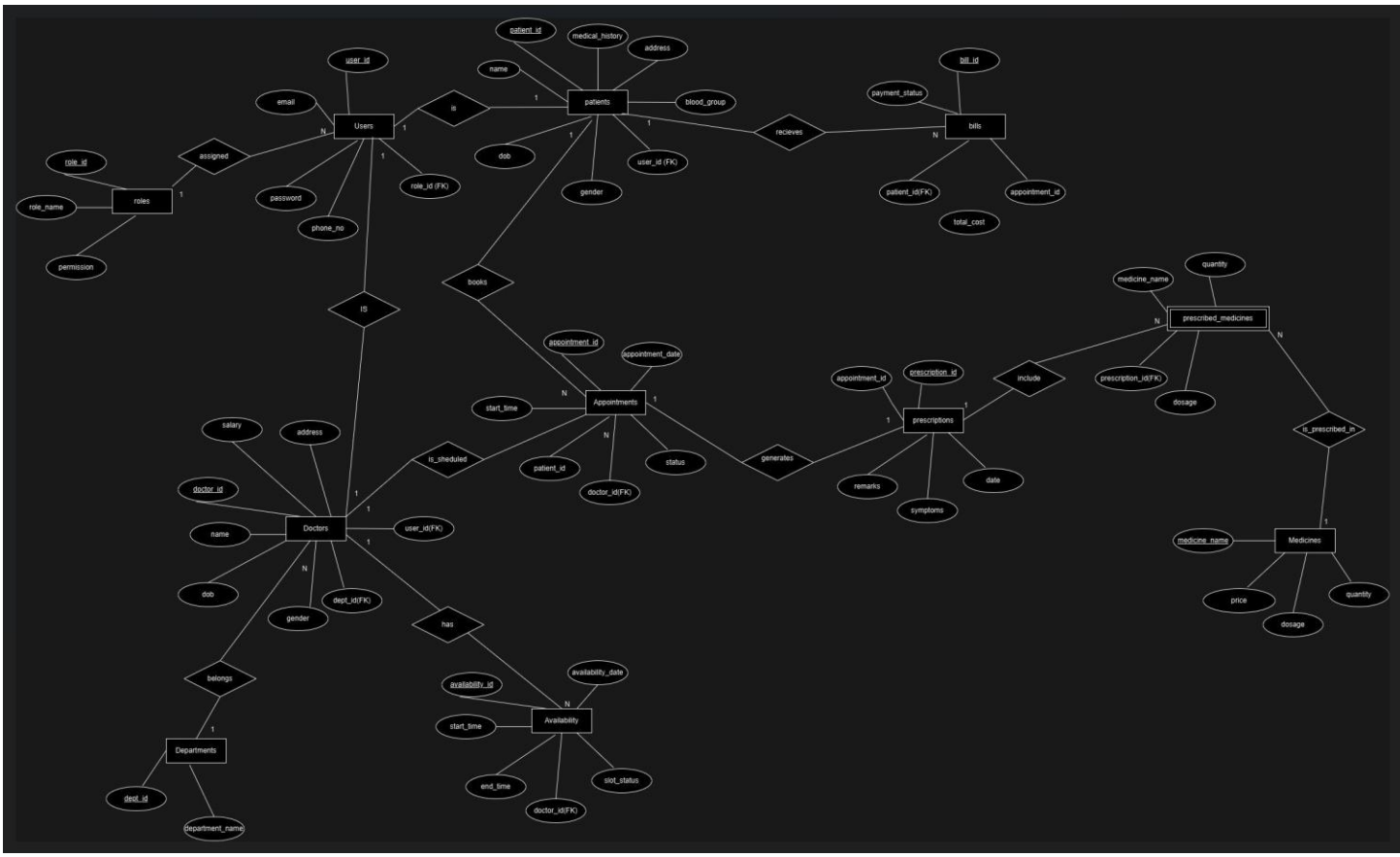
DBMS : ASSIGNMENT 3

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="{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.