Hospital Database Management System

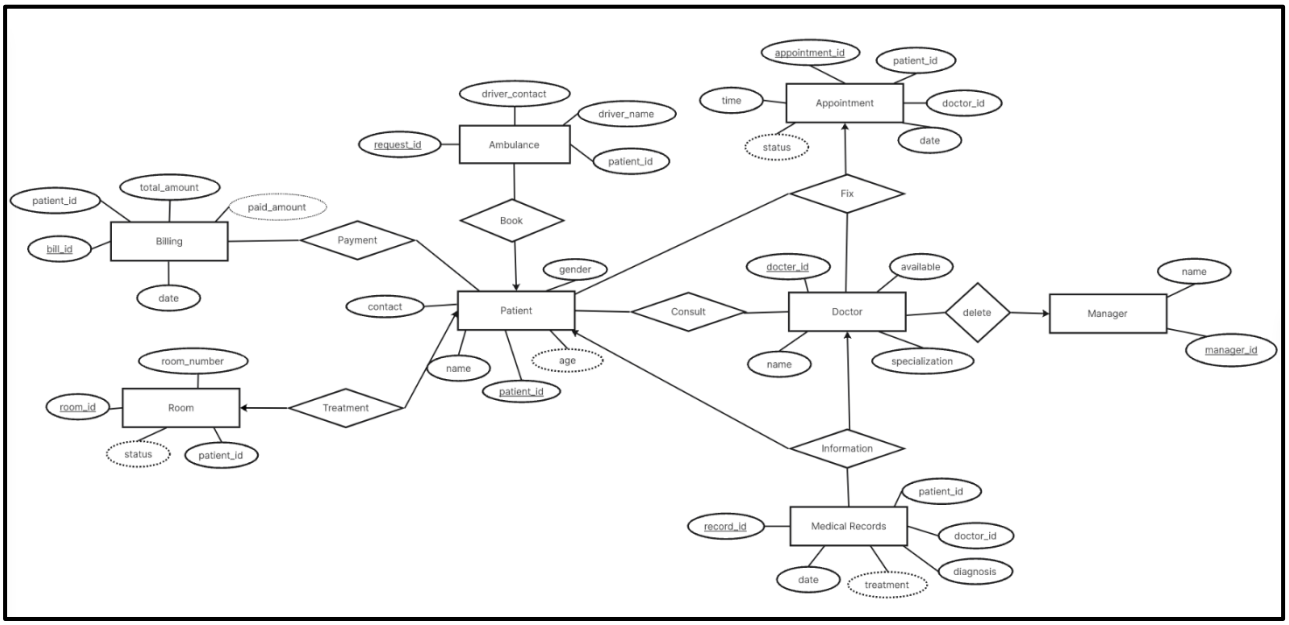
This Project focuses on designing a **Hospital Database Management System (HMS)** using Python and the Streamlit framework. The HMS aims to streamline hospital operations such as patient registration, doctor management, appointment scheduling, billing, medical record maintenance, room assignments, and ambulance requests. This system offers a user-friendly interface for both hospital applicants (patients) and managers, ensuring a smooth flow of healthcare services.

Entity-Relationship (ER) for this Database

The ER diagram represents the relationships between various entities involved in the Hospital Management System, including:

1. **Patient**: The person receiving medical care.
2. **Doctor**: The medical professional providing healthcare.
3. **Appointment**: The scheduling of a patient's meeting with a doctor.
4. **Billing**: The process of generating bills for the provided healthcare services.
5. **Medical Records**: The documentation of diagnoses and treatments.
6. **Room**: The physical space assigned to the patient.
7. **Ambulance**: Emergency transportation service for patients.
8. **Manager**: Person in charge of overseeing hospital operations, including removing doctors.

ER – Diagram:



Python code:

The **Hospital Management System** Python code is a streamlined application built using **Streamlit** for the frontend and Python for backend operations. The frontend offers an intuitive interface where users can register patients and doctors, manage appointments, assign rooms, and generate bills. The backend handles all database operations, including creating, reading, updating, and deleting records via dedicated Python scripts. The system interacts with a SQLite database, ensuring real-time updates and smooth data management. This separation of frontend and backend ensures scalability and ease of maintenance.

Directory structure

```
HospitalManagementSystem/
│
├── database/
│   ├── database.py          # Database initialization and connection
│
├── models/
│   ├── patient.py           # Functions related to patient operations
│   ├── doctor.py            # Functions related to doctor operations
│   ├── appointment.py       # Functions for booking and managing appointments
│   ├── billing.py           # Functions for generating and retrieving bills
│   ├── room.py              # Functions related to room assignments
│   ├── ambulance.py         # Functions related to ambulance requests
│   ├── medical_records.py   # Functions for handling medical records
│   ├── manager.py           # Functions for managerial operations (e.g., remove
doctor)
│
├── main.py                  # Main Streamlit application code
└── README.md                # Description and instructions for the project
```

Let's go through one by one –

database/database.py

```
import sqlite3

def connect_db():
    return sqlite3.connect('database/hospital.db')

def initialize_db():
    conn = connect_db()
    cursor = conn.cursor()


    cursor.execute('''
        CREATE TABLE IF NOT EXISTS patients (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            age INTEGER,
            gender TEXT,
            contact TEXT
        )
    ''')

    # Create Doctors table
    cursor.execute('''CREATE TABLE IF NOT EXISTS doctors (
                        doctor_id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
                        name TEXT,
                        specialization TEXT,
                        available BOOLEAN)''')

    # Create Appointments table
    cursor.execute('''CREATE TABLE IF NOT EXISTS appointments (
                        appointment_id INTEGER PRIMARY KEY AUTOINCREMENT,
                        patient_id INTEGER,
                        doctor_id INTEGER,
                        date TEXT,
                        time TEXT,

                        status TEXT DEFAULT 'Scheduled',
                        FOREIGN KEY(patient_id) REFERENCES patients(patient_id),
                        FOREIGN KEY(doctor_id) REFERENCES doctors(doctor_id))''')

    # Create Billing table
    cursor.execute('''CREATE TABLE IF NOT EXISTS billing (
                        billing_id INTEGER PRIMARY KEY AUTOINCREMENT,
                        patient_id INTEGER,
                        total_amount REAL,
                        paid_amount REAL,
                        payment_date TEXT,
                        FOREIGN KEY(patient_id) REFERENCES
patients(patient_id))''')

    # Create Rooms table
    cursor.execute('''CREATE TABLE IF NOT EXISTS rooms (
                        room_id INTEGER PRIMARY KEY AUTOINCREMENT,
                        room_number TEXT,
                        patient_id INTEGER,
                        status TEXT,
                        FOREIGN KEY(patient_id) REFERENCES
patients(patient_id))''')

    # Create Ambulance Services table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS ambulance (
            request_id INTEGER PRIMARY KEY AUTOINCREMENT,
            patient_id INTEGER,
            driver_name TEXT,
            contact TEXT,
            FOREIGN KEY(patient_id) REFERENCES patients(patient_id)
        )
    ''')

    # Create Medical Records Table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS medical_records (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
            patient_id INTEGER,
            diagnosis TEXT NOT NULL,
            treatment TEXT,
            prescription TEXT,
            created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
            FOREIGN KEY(patient_id) REFERENCES patients(id)
        )
    ''')

    conn.commit()
    conn.close()

initialize_db()
```

- This script initializes the SQLite database, creates tables for patients, doctors, appointments, rooms, and billing. It handles the database connection and ensures all tables are set up.

models/patient.py

```
from database.database import connect_db

def register_patient(name, age, gender, contact):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('''INSERT INTO patients (name, age, gender, contact)
                      VALUES (?, ?, ?, ?)''', (name, age, gender, contact))
    conn.commit()
    conn.close()

def get_patient_info(patient_id=None):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM patients")
    patients = cursor.fetchall()
    return patients
```

This file manages doctor-related operations such as adding a new doctor and retrieving doctor information by ID.

models/doctor.py

```
from database.database import connect_db

def register_doctor(name, specialization, available):
    conn = connect_db()
```

```
    cursor = conn.cursor()
    cursor.execute('''INSERT INTO doctors (name, specialization, available)
                      VALUES (?, ?, ?)''', (name, specialization, available))
    conn.commit()
    conn.close()
    doctor_id = cursor.lastrowid
    return doctor_id

def get_doctor_info(doctor_id=None):
    conn = connect_db()
    cursor = conn.cursor()


    if doctor_id:
        cursor.execute('''
            SELECT * FROM doctors WHERE doctor_id = ?
        ''', (doctor_id,))
    else:
        cursor.execute('''
            SELECT * FROM doctors
        ''')

    doctors = cursor.fetchall()
    conn.close()
    return doctors
```

models/appointment.py

```
from database.database import connect_db

def book_appointment(patient_id, doctor_id, date, time):
    conn = connect_db()
    cursor = conn.cursor()

    # Convert the datetime.time object to a string in 'HH:MM' format
    time_str = time.strftime('%H:%M')

    cursor.execute('''
        INSERT INTO appointments (patient_id, doctor_id, date, time)
        VALUES (?, ?, ?, ?)
    ''', (patient_id, doctor_id, date, time_str))

    conn.commit()
    conn.close()

def get_appointments():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM appointments')
```

```
    appointments = cursor.fetchall()
    conn.close()
    return appointments
```

Manages booking of appointments between patients and doctors, including retrieving all appointments for a specific doctor.

models/medical_records.py

```
from database.database import connect_db

# Function to create a medical record for a patient
def create_medical_record(patient_id, diagnosis, treatment, prescription):
    conn = connect_db()
    cursor = conn.cursor()

    # Insert the medical record into the database
    cursor.execute('''
        INSERT INTO medical_records (patient_id, diagnosis, treatment,
prescription)
        VALUES (?, ?, ?, ?)
    ''', (patient_id, diagnosis, treatment, prescription))

    conn.commit()
    conn.close()

# Function to retrieve all medical records for a given patient
def get_medical_records():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM medical_records')
    medical_records = cursor.fetchall()
    conn.close()
    return medical_records  # Returns all medical records as a list of tuples
```

models/billing.py

```
from database.database import connect_db

def generate_bill(patient_id, total_amount):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('''INSERT INTO billing (patient_id, total_amount, paid_amount,
```

```
payment_date)
                      VALUES (?, ?, ?, CURRENT_TIMESTAMP)''', (patient_id,
total_amount, total_amount))
    conn.commit()
    conn.close()

def get_bills():
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('SELECT * FROM billing')
    bills = cursor.fetchall()

    conn.close()
    return bills  # Returns all bills as a list of tuples
```

Contains functions to generate a bill for a patient and retrieve the bill information for a specific patient.

models/room.py

```
from database.database import connect_db

def assign_room(room_number, patient_id, status):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('''INSERT INTO rooms (room_number, patient_id, status)
                      VALUES (?, ?, ?)''', (room_number, patient_id, status))
    conn.commit()
    conn.close()

def get_rooms():
    conn = connect_db()
    cursor = conn.cursor()

    cursor.execute('SELECT * FROM rooms')

    rooms = cursor.fetchall()
    conn.close()
    return rooms  # Returns all rooms as a list of tuples
```

Handles room assignments for patients, allowing the assignment of rooms and retrieving room details for a specific patient.

models/ambulance.py

```
from database.database import connect_db

def request_ambulance(patient_id, driver_name, contact):
    conn = connect_db()
    cursor = conn.cursor()
    cursor.execute('''INSERT INTO ambulance (patient_id, driver_name, contact)
                      VALUES (?, ?, ?)''', (patient_id, driver_name, contact))
    conn.commit()
    conn.close()

def get_ambulance_requests():
    conn = connect_db()

    cursor = conn.cursor()
    cursor.execute('SELECT * FROM ambulance')
    ambulance_requests = cursor.fetchall()
    conn.close()
    return ambulance_requests  # Returns all ambulance requests as a list of
tuples
```

Deals with storing and retrieving medical records of patients, including diagnosis and treatment details.

models/manager.py

```
# manager.py
import sqlite3
from database.database import connect_db

def remove_doctor(doctor_id):
    try:
        conn = connect_db()
        cursor = conn.cursor()

        # Check if doctor exists before trying to delete
        cursor.execute("SELECT * FROM doctors WHERE doctor_id = ?", (doctor_id,))
        doctor = cursor.fetchone()

        if doctor:
            cursor.execute("DELETE FROM doctors WHERE doctor_id = ?",
(doctor_id,))
            conn.commit()
            return 1
        else:
            return 0
    except sqlite3.Error as e:
        return f"An error occurred: {e}"
```

```
    finally:
        conn.close()
```

Provides the functionality for a manager to remove a doctor from the system.

Main code: app.py

```
import streamlit as st
import pandas as pd
from models.patient import register_patient, get_patient_info
from models.doctor import register_doctor, get_doctor_info
from models.appointment import book_appointment, get_appointments
from models.billing import generate_bill, get_bills
from models.room import assign_room, get_rooms
from models.ambulance import request_ambulance, get_ambulance_requests
from models.medical_records import create_medical_record, get_medical_records
from models.manager import remove_doctor
from database.database import initialize_db, connect_db

# Initialize the database when the app starts
initialize_db()

# Sidebar for Navigation
st.sidebar.title('''
            Welcome
            - Choose your role and action''')

# Streamlit Application Title
st.title("🏥 Hospital Management System")
st.write("----")
role = st.sidebar.selectbox("Select Role", ["Dashboard", "Applicant", "Manager"])

if role == "Dashboard":
    # Description of the Hospital Management System
    st.subheader("About This System")
    st.write("""
    This Hospital Management System (HMS) is designed to streamline hospital
operations and enhance patient care.
    The system allows for the efficient management of patients, doctors,
appointments, billing, medical records,
    room assignments, and ambulance requests.
    Each entity plays a vital role in ensuring smooth healthcare delivery.
    """)

    # Entity Descriptions
    st.subheader("Entity Descriptions")
    st.write("- **Patient**: Individuals seeking medical care.")
```

```
    st.write("- **Doctor**: Medical professionals providing healthcare services.")
    st.write("- **Appointment**: Scheduled meetings between patients and
doctors.")
    st.write("- **Billing**: Financial transactions for medical services
provided.")
    st.write("- **Medical Records**: Documentation of patient diagnoses and
treatments.")
    st.write("- **Room**: Physical spaces assigned to patients for care.")
    st.write("- **Ambulance**: Emergency transport service for patients.")
    st.write("- **Manager**: Authority to manage and oversee hospital staff and
operations.")


# Applicant Actions
if role == "Applicant":
    action = st.sidebar.selectbox("Choose Section",
        ["Register Patient", "Register Doctor", "Book Appointment", "Generate
Bill",
        "Add Medical Records", "Assign Room", "Request Ambulance", "View Database
Records"])

    # Patient Registration Form
    if action == "Register Patient":
        st.subheader("👤 Register Patient")
        with st.form("patient_form"):
            name = st.text_input("Patient Name")
            age = st.number_input("Patient Age", min_value=0)
            gender = st.selectbox("Gender", ["Male", "Female", "Other"])
            contact = st.text_input("Contact Number")
            submitted = st.form_submit_button("Register Patient")
            if submitted:
                register_patient(name, age, gender, contact)
                st.success(f"Patient {name} registered successfully!")

    # Doctor Registration Form
    if action == "Register Doctor":
        st.subheader("👨‍⚕️ Register Doctor")
        with st.form("doctor_form"):
            doctor_name = st.text_input("Doctor Name")
            specialization = st.text_input("Specialization")
            available = st.checkbox("Available")
            doc_submitted = st.form_submit_button("Register Doctor")
            if doc_submitted:
                doc_id = register_doctor(doctor_name, specialization, available)
                st.success(f"Doctor {doctor_name} registered successfully with
Doctor ID [{doc_id}]!")

    # Book Appointment
    if action == "Book Appointment":
        st.subheader("📅 Book Appointment")
        with st.form("appointment_form"):
```

```
            patient_id = st.number_input("Patient ID", min_value=1)
            doctor_id = st.number_input("Doctor ID", min_value=1)
            date = st.date_input("Appointment Date")
            time = st.time_input("Appointment Time")
            appointment_submitted = st.form_submit_button("Book Appointment")
            if appointment_submitted:
                book_appointment(patient_id, doctor_id, date, time)
                st.success(f"Appointment booked for Patient ID {patient_id} with
Doctor ID {doctor_id}!")

    # Billing Section

    if action == "Generate Bill":
        st.subheader("💰 Generate Bill")
        with st.form("billing_form"):
            billing_patient_id = st.number_input("Patient ID for Billing",
min_value=1)
            total_amount = st.number_input("Total Amount", min_value=0.0)
            paid_amount = st.number_input("Paid Amount", min_value=0.0)
            bill_submitted = st.form_submit_button("Generate Bill")
            if bill_submitted:
                generate_bill(billing_patient_id, total_amount)
                st.success(f"Bill generated for Patient ID {billing_patient_id}!")

    # Medical Records Section
    if action == "Add Medical Records":
        st.subheader("📋 Add Medical Record")
        with st.form("medical_record_form"):
            record_patient_id = st.number_input("Patient ID", min_value=1)
            diagnosis = st.text_input("Diagnosis")
            treatment = st.text_input("Treatment")
            record_date = st.date_input("Record Date")
            medical_record_submitted = st.form_submit_button("Add Medical Record")
            if medical_record_submitted:
                create_medical_record(record_patient_id, diagnosis, treatment,
record_date)
                st.success(f"Medical record for Patient ID {record_patient_id}
added successfully!")

    # Assign Room
    if action == "Assign Room":
        st.subheader("🛏️ Assign Room to Patient")
        with st.form("room_form"):
            room_number = st.text_input("Room Number")
            patient_room_id = st.number_input("Patient ID for Room Assignment",
min_value=1)
            room_status = st.selectbox("Room Status", ["Occupied", "Available"])
            room_submitted = st.form_submit_button("Assign Room")
            if room_submitted:
                assign_room(room_number, patient_room_id, room_status)
```

```
                st.success(f"Room {room_number} assigned to Patient ID
{patient_room_id}.")

    # Request Ambulance
    if action == "Request Ambulance":
        st.subheader("🚑 Request Ambulance")
        with st.form("ambulance_form"):
            ambulance_patient_id = st.number_input("Patient ID for Ambulance",
min_value=1)
            driver_name = st.text_input("Driver Name")
            contact_ambulance = st.text_input("Driver Contact")
            ambulance_submitted = st.form_submit_button("Request Ambulance")

            if ambulance_submitted:
                request_ambulance(ambulance_patient_id, driver_name,
contact_ambulance)
                st.success(f"Ambulance requested for Patient ID
{ambulance_patient_id}.")

    # View Database Records Section
    if action == "View Database Records":
        st.subheader("📊 View Database Records")

        # Display Patients Table
        if st.button("View all Patients"):
            patients = get_patient_info()
            if patients:
                patients_df = pd.DataFrame(patients, columns=["Patient ID",
"Name", "Age", "Gender", "Contact"])
                st.dataframe(patients_df)
            else:
                st.write("No patients found.")

        # Display Doctors Table
        if st.button("View all Doctors"):
            doctors = get_doctor_info()
            if doctors:
                doctors_df = pd.DataFrame(doctors, columns=["Doctor ID", "Name",
"Specialization", "Available"])
                st.dataframe(doctors_df)
            else:
                st.write("No doctors found.")

        # Display Appointments Table
        if st.button("View all Appointments"):
            appointments = get_appointments()
            if appointments:
                appointments_df = pd.DataFrame(appointments, columns=["Appointment
ID", "Patient ID", "Doctor ID", "Date", "Time", "Status"])
                st.dataframe(appointments_df)
```

```
            else:
                st.write("No appointments found.")

        # Display Medical Records Table
        if st.button("View all Medical Records"):
            records = get_medical_records()
            if records:
                medical_records_df = pd.DataFrame(records, columns=["Record ID",
"Patient ID", "Doctor ID", "Diagnosis", "Treatment", "Date"])
                st.dataframe(medical_records_df)
            else:
                st.write("No medical records found.")

        # Display Billing Table

        if st.button("View all Billing"):
            bills = get_bills()
            if bills:
                bills_df = pd.DataFrame(bills, columns=["Bill ID", "Patient ID",
"Total Amount", "Paid Amount", "Date"])
                st.dataframe(bills_df)
            else:
                st.write("No bills found.")

        # Display Rooms Table
        if st.button("View all Rooms"):
            rooms = get_rooms()
            if rooms:
                rooms_df = pd.DataFrame(rooms, columns=["Room ID", "Room Number",
"Patient ID", "Status"])
                st.dataframe(rooms_df)
            else:
                st.write("No rooms assigned.")

        # Display Ambulance Requests Table
        if st.button("View all Ambulance Requests"):
            ambulance_requests = get_ambulance_requests()
            if ambulance_requests:
                ambulance_df = pd.DataFrame(ambulance_requests, columns=["Request
ID", "Patient ID", "Driver Name", "Contact"])
                st.dataframe(ambulance_df)
            else:
                st.write("No ambulance requests found.")

# Manager Actions
elif role == "Manager":
    action = st.sidebar.selectbox("Choose Manager Action", ["Remove Doctor", "View
Records"])
```

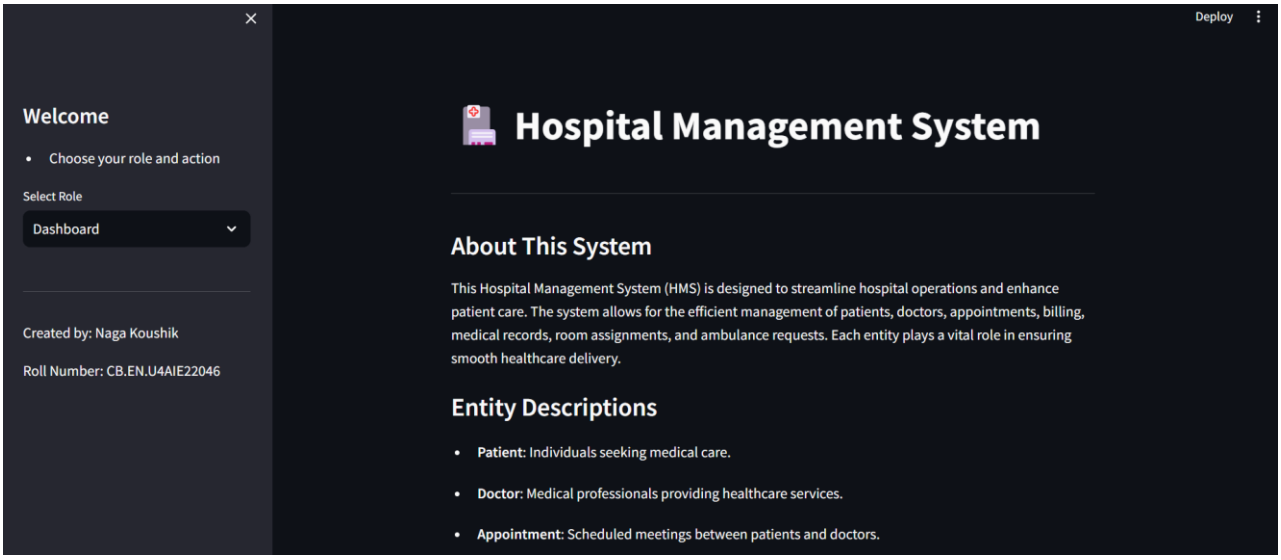
```
    # Remove Doctor Action
    if action == "Remove Doctor":
        st.subheader("🩺 Remove Doctor")
        with st.form("remove_doctor_form"):
            doctor_id = st.number_input("Doctor ID", min_value=1)
            remove_submitted = st.form_submit_button("Remove Doctor")
            if remove_submitted:
                out = remove_doctor(doctor_id)
                if out:
                    st.success(f"Doctor with ID {doctor_id} has been removed
successfully!")
                else:
                    st.error(f"Doctor with ID {doctor_id} not found!")
        if st.button("Current Doctors list"):
            doctors = get_doctor_info()
            if doctors:

                doctors_df = pd.DataFrame(doctors, columns=["Doctor ID", "Name",
"Specialization", "Available"])
                st.dataframe(doctors_df)
            else:
                st.write("No doctors found.")

    # View Manager Records
    if action == "View Records":
        st.subheader("📊 View All Records")

        # Display Patients Table
        if st.button("View all Patients"):
            patients = get_patient_info()
            if patients:
                patients_df = pd.DataFrame(patients, columns=["Patient ID",
"Name", "Age", "Gender", "Contact"])
                st.dataframe(patients_df)
            else:
                st.write("No patients found.")

        # Display Doctors Table
        if st.button("View all Doctors"):
            doctors = get_doctor_info()
            if doctors:
                doctors_df = pd.DataFrame(doctors, columns=["Doctor ID", "Name",
"Specialization", "Available"])
                st.dataframe(doctors_df)
            else:
                st.write("No doctors found.")

        # Display Appointments Table
        if st.button("View all Appointments"):
            appointments = get_appointments()
            if appointments:
```

```
                appointments_df = pd.DataFrame(appointments, columns=["Appointment
ID", "Patient ID", "Doctor ID", "Date", "Time", "Status"])
                st.dataframe(appointments_df)
            else:
                st.write("No appointments found.")

        # Display Medical Records Table
        if st.button("View all Medical Records"):
            records = get_medical_records()
            if records:
                medical_records_df = pd.DataFrame(records, columns=["Record ID",
"Patient ID", "Doctor ID", "Diagnosis", "Treatment", "Date"])
                st.dataframe(medical_records_df)
            else:
                st.write("No medical records found.")
        # Display Billing Table
        if st.button("View all Billing"):
            bills = get_bills()
            if bills:
                bills_df = pd.DataFrame(bills, columns=["Bill ID", "Patient ID",
"Total Amount", "Paid Amount", "Date"])
                st.dataframe(bills_df)
            else:
                st.write("No bills found.")

        # Display Rooms Table
        if st.button("View all Rooms"):
            rooms = get_rooms()
            if rooms:
                rooms_df = pd.DataFrame(rooms, columns=["Room ID", "Room Number",
"Patient ID", "Status"])
                st.dataframe(rooms_df)
            else:
                st.write("No rooms assigned.")

        # Display Ambulance Requests Table
        if st.button("View all Ambulance Requests"):
            ambulance_requests = get_ambulance_requests()
            if ambulance_requests:
                ambulance_df = pd.DataFrame(ambulance_requests, columns=["Request
ID", "Patient ID", "Driver Name", "Contact"])
                st.dataframe(ambulance_df)
            else:
                st.write("No ambulance requests found.")


# Footer with Creator Information
st.sidebar.write("---")
st.sidebar.write("Created by: Naga Koushik")
st.sidebar.write("Roll Number: CB.EN.U4AIE22046")
```

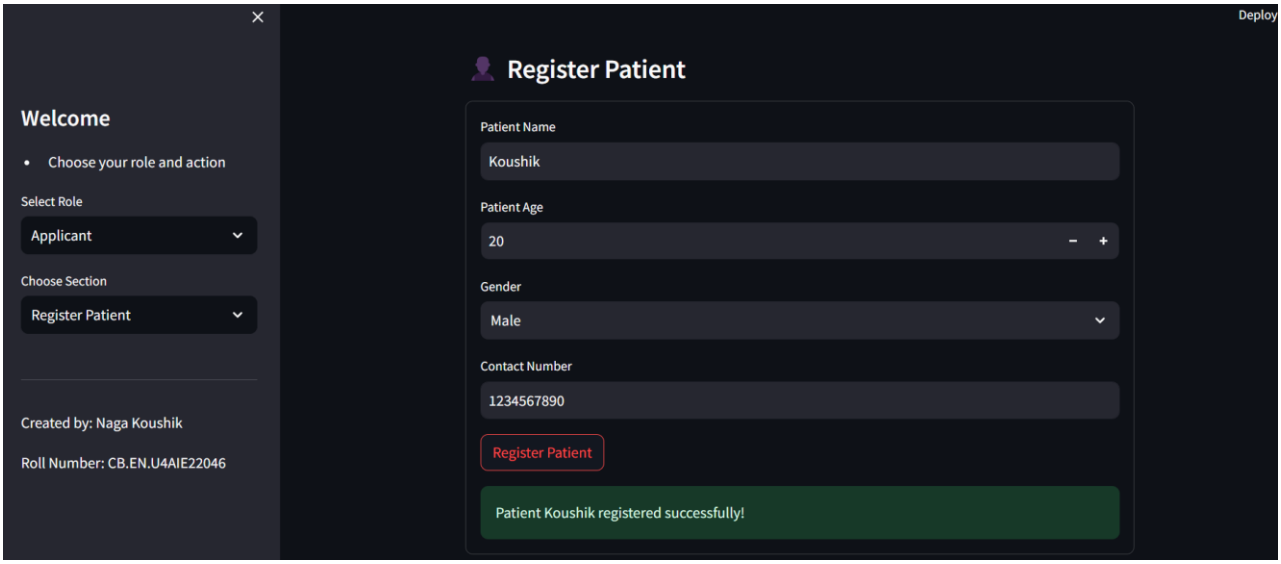
Front-end(GUI):

The frontend of the Hospital Management System is built using Streamlit, offering a clean and interactive GUI. It allows users to interact with the system through intuitive forms, buttons, and tables. Forms are used for tasks like registering patients, booking appointments, and assigning rooms, while interactive widgets such as dropdowns, text inputs, and checkboxes are utilized to gather user inputs. The interface also displays data in an organized manner, showing patient records, bills, and appointments in tables. A sidebar provides easy navigation, allowing users to select their roles (e.g., applicant or manager) and access the relevant operations, ensuring a seamless user experience.

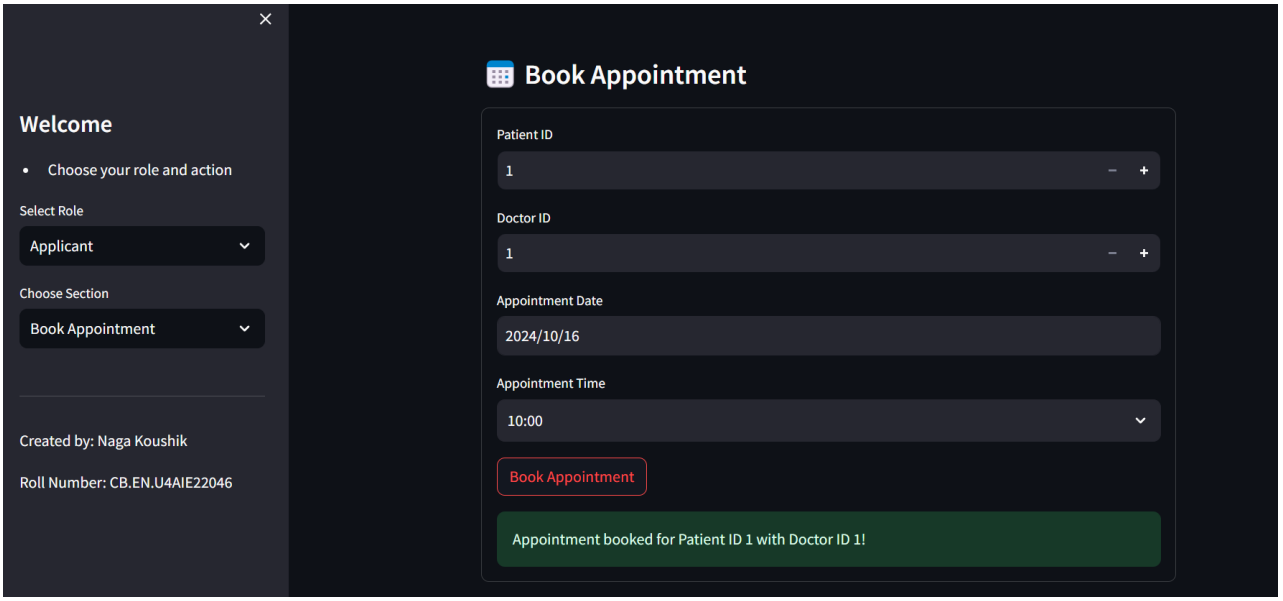
Dashboard

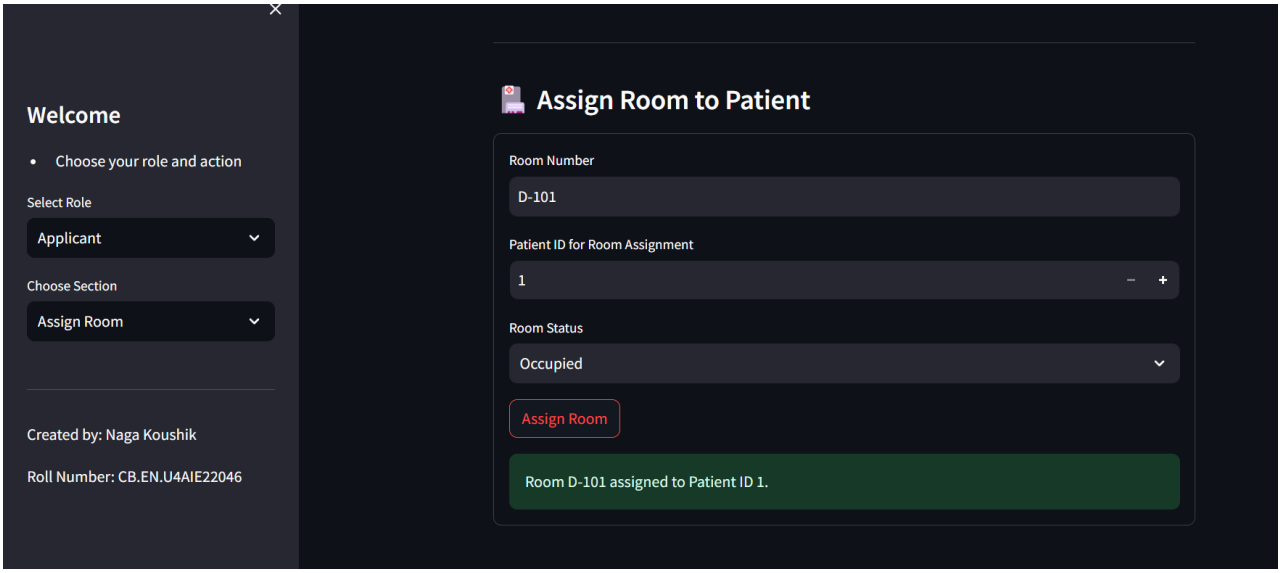


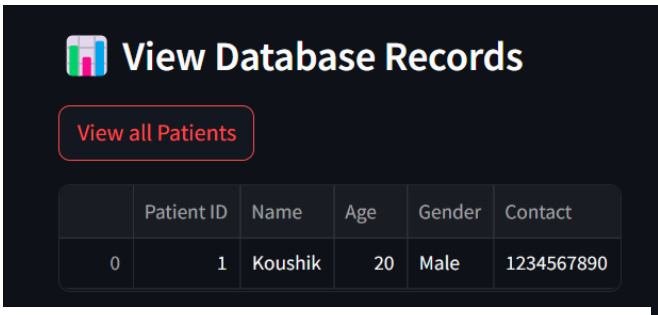
Applicant Portal

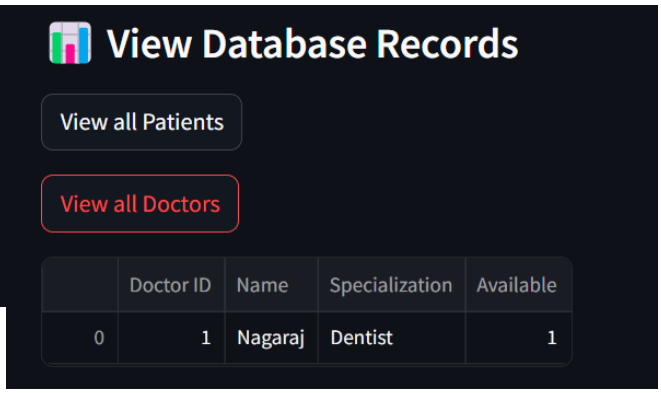


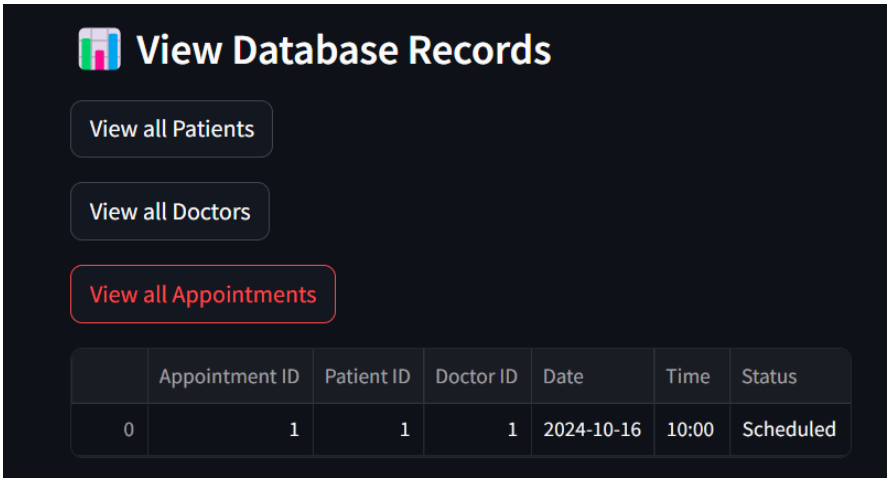

Welcome
Choose your role and action
Select Role
Applicant
Choose Section
Register Doctor
Created by: Naga Koushik
Roll Number: CB.EN.U4AIE22046
Hospital Management System
Register Doctor
Doctor Name
Nagaraj
Specialization
Dentist
Available
Register Doctor
Doctor Nagaraj registered successfully with Doctor ID [1]!


Welcome
Choose your role and action
Select Role
Applicant
Choose Section
Book Appointment
Created by: Naga Koushik
Roll Number: CB.EN.U4AIE22046
Book Appointment
Patient ID
1
Doctor ID
1
Appointment Date
2024/10/16
Appointment Time
10:00
Book Appointment
Appointment booked for Patient ID 1 with Doctor ID 1!

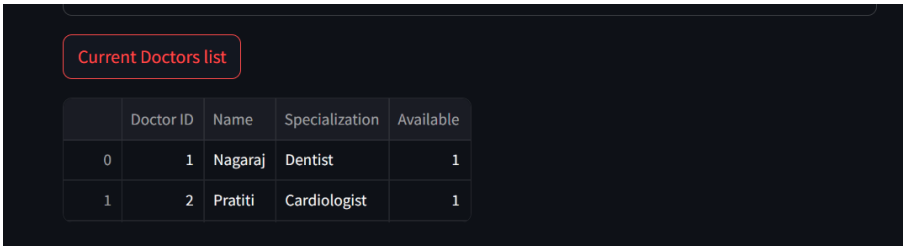

Welcome
Choose your role and action
Select Role
Applicant
Choose Section
Assign Room
Created by: Naga Koushik
Roll Number: CB.EN.U4AIE22046
Assign Room to Patient
Room Number
D-101
Patient ID for Room Assignment
1
Room Status
Occupied
Assign Room
Room D-101 assigned to Patient ID 1.



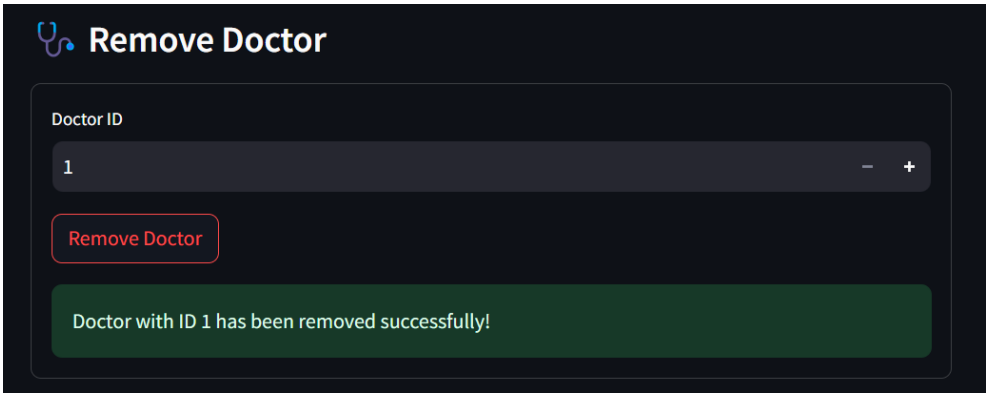


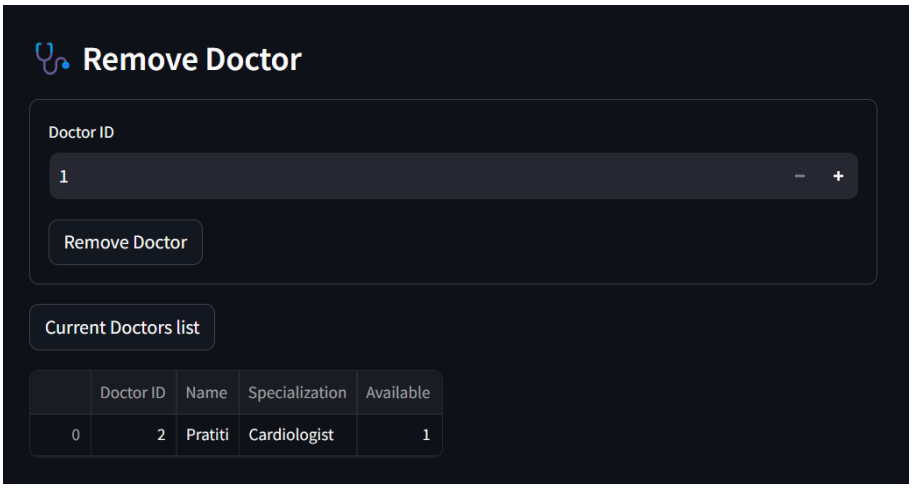


Manager Actions:



Removing Doctor with ID 1





Conclusion:

The Hospital Management System developed in Python provides a comprehensive and user-friendly solution for managing hospital operations. With its Streamlit-based frontend, the system enables smooth interaction through intuitive forms and data displays, catering to both applicants and managers. The robust backend, powered by Python modules and a well-structured database, efficiently handles various functionalities such as patient registration, doctor management, appointment scheduling, billing, and record maintenance. This project demonstrates an effective integration of technology for streamlining hospital workflows, ensuring efficient healthcare management.