REC HOSPITAL MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted By

SHANMUGA PRIYA RAANJANI S H (220701262)

SAKTHI M S (220701240)

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023-2024

BONAFIDE CERTIFICATE

Certified that this project report "**HOSPITAL MANAGEMENT SYSTEM**" is the bonafide work of "**SHANMUGA PRIYA RAANJANI S H (220701262) AND SAKTHI M S (220701240)**" who carried out the project work under my supervision.

Submitted for practical examination held on _____

INTERNAL EXAMINER EXTERNAL EXAMINER

TABLE OF CONTENTS

Hospital Management System

1. INTRODUCTION

1.1 Introduction

In the rapidly evolving landscape of healthcare, the integration of technology is crucial for enhancing the efficiency and quality of medical services. The Hospital Management System (HMS) is designed to address this need by providing a comprehensive, centralized platform for managing various aspects of hospital operations. As healthcare facilities grow in size and complexity, managing patient records, doctor schedules, administrative tasks, and other critical functions becomes increasingly challenging. The HMS aims to simplify these processes through automation and integration, ensuring that healthcare providers can deliver timely and accurate care to patients.

The core functionality of the HMS revolves around the seamless management of patient and doctor data, streamlined administrative processes, and improved communication within the hospital. By centralizing data and providing easy access to information, the system helps reduce errors, improve decision-making, and enhance the overall patient experience. Furthermore, the HMS is designed to be user-friendly, ensuring that users with varying levels of technical expertise can navigate the system with ease.

One of the key features of the HMS is its ability to provide personalized dashboards for different user roles, including doctors, patients, and administrators. Doctors can access their schedules, view patient details, and search for patient information, enabling them to manage their workload more efficiently. Patients, on the other hand, can view their medical records, manage appointments, and search for doctors, thereby enhancing their engagement in their own healthcare journey. Administrators have access to powerful tools for managing patient and doctor data, updating records, and generating reports, which helps streamline hospital operations and improve resource allocation.

Security is a paramount concern in the healthcare industry, and the HMS addresses this through robust security measures that protect sensitive patient and doctor information. The system employs encryption, secure authentication mechanisms, and role-based access control to ensure that data remains confidential and is accessed only by authorized personnel. Additionally, the HMS is built to be scalable and adaptable, allowing it to grow with the hospital and accommodate new functionalities as needed.

The HMS also facilitates better communication between patients and healthcare providers. Integrated messaging and notification systems ensure that important information is conveyed promptly, reducing delays and improving patient outcomes. By leveraging modern software technologies such as Python for backend development and Streamlit for the web interface, along with MySQL for reliable database management, the HMS combines performance, reliability, and ease of use.

In summary, the Hospital Management System is a comprehensive solution designed to meet the needs of modern healthcare facilities. By automating routine tasks, integrating critical processes, and providing a secure and user-friendly platform, the HMS enhances the efficiency and quality of healthcare services, ultimately benefiting both healthcare providers and patients.

1.2 Objectives

The key objectives of the Hospital Management System are:

1. Enhance Operational Efficiency:

- Streamline hospital workflows by automating routine administrative tasks.
- Reduce paperwork and manual data entry to save time and minimize errors.

2. Improve Patient Care:

- Provide healthcare providers with quick access to accurate patient records.
- Facilitate better decision-making through comprehensive data analytics.

3. Ensure Data Security:

- Implement robust security measures to protect sensitive patient and doctor information.
- Use encryption, secure authentication, and access controls to ensure data privacy.

4. Facilitate Communication:

- Enhance communication between doctors, patients, and administrative staff.
- Implement notification systems for appointments, reminders, and updates.

5. Support Scalability and Flexibility:

- Design the system to handle growing volumes of data and users.
- Allow for easy integration of new features and functionalities as needed.

6. User-Friendly Interface:

- Develop intuitive and accessible interfaces for doctors, patients, and admins.
- Ensure ease of use across different devices and platforms.

7. Centralized Data Management:

- Consolidate all hospital data into a single, centralized system for easy access and management.
- Enable seamless data sharing and coordination across departments.

8. Regulatory Compliance:

- Ensure the system complies with healthcare regulations and standards.
- Implement features to support data audit trails and compliance reporting.

9. Enhance Resource Management:

- Optimize the allocation and utilization of hospital resources, including rooms and equipment.
- Provide real-time insights into resource availability and usage.

10. Customizable Reporting:

- Enable the generation of customizable reports for various stakeholders.
- Provide data visualization tools to support hospital management and planning.

1.3 Modules

1.3.1 User Management Module

Description: This module handles the registration, login, and management of users, including doctors, patients, and administrators.

Key Features:

Registration: Allows new users (doctors, patients, and admins) to register by providing personal details and creating credentials.

Login: Provides secure login functionality with email and password authentication.

Password Management: Facilitates password creation and updating, especially for users with null passwords.

Role-Based Access Control: Ensures that different users have access to different features based on their role.

Functionality:

Doctors can access their schedules and patient information.
Patients can view their medical records and appointment details.
Admins can manage hospital resources, patient records, and doctor details.

1.3.2 Patient Management Module

Description: This module focuses on managing patient data and interactions within the hospital.

Key Features:

Patient Records: Maintains detailed records for each patient, including personal information, medical history, appointments, and prescriptions.

Appointment Scheduling: Allows patients to schedule, reschedule, or cancel appointments with doctors.

Medical History Tracking: Records patient visits, diagnoses, treatments, and medication history.

Search and Filter: Provides search functionality for doctors to quickly find patient records.

Functionality:

Patients can view and update their personal details and appointment schedules.
Doctors can access patient medical records and update treatment information.

1.3.3 Doctor Management Module

Description: This module is designed to manage doctor-related activities and data.

Key Features:

Doctor Profiles: Maintains detailed profiles of doctors, including specialties, department affiliations, contact information, and schedules.

Schedule Management: Allows doctors to view and manage their appointment schedules.

Patient Interaction: Provides tools for doctors to search patient records, update medical information, and track treatment plans.

Notifications: Sends reminders and notifications to doctors about appointments and patient updates.

Functionality:

- Doctors can update their profiles and manage their schedules.
- Doctors can search for patients and update medical records.

1.3.4 Appointment Management Module

Description: This module manages all aspects related to appointments between patients and doctors.

Key Features:

Booking System: Enables patients to book, view, and cancel appointments with doctors.

Schedule Viewing: Allows doctors to view their upcoming appointments and manage their schedules.

Reminders and Notifications: Sends automated reminders to patients and doctors about upcoming appointments.

Conflict Management: Prevents double-booking and manages appointment conflicts.

Functionality:

- Patients can easily book appointments through a user-friendly interface.
- Doctors can view and manage their daily, weekly, and monthly schedules.

1.3.5 Administration Module

Description: This module provides administrative tools for managing the overall operations of the hospital.

Key Features:

Resource Management: Manages hospital resources, including rooms, equipment, and staff.

Data Update and Management: Allows admins to update patient and doctor information.

Reporting and Analytics: Generates reports on hospital performance, resource utilization, and other key metrics.

User Management: Provides tools for managing user roles, access levels, and permissions.

Functionality:

- Admins can view, update, and manage patient and doctor records.
- Admins can generate reports and analyze data for informed decision-making.

1.3.6 Room Management Module

Description: This module manages the allocation and availability of hospital rooms and equipment.

Key Features:

Room Allocation: Tracks the availability and occupancy of hospital rooms.

Equipment Inventory: Manages the inventory of medical equipment and tracks usage and maintenance schedules.

Booking and Scheduling: Allows for the booking and scheduling of rooms and equipment for various purposes, including surgeries and consultations.

Resource Tracking: Provides real-time tracking of resource usage and availability.

Functionality:

- Admins can allocate rooms and equipment based on availability and need.
- Doctors can book rooms and equipment for procedures and consultations.

1.3.7 Notification and Communication Module

Description: This module facilitates communication between patients, doctors, and administrators through automated notifications and messaging.

Key Features:

Automated Notifications: Sends email and SMS notifications for appointment reminders, updates, and important announcements.

Internal Messaging: Allows users to send and receive messages within the system for better coordination.

Alert System: Provides alerts for critical updates, such as changes in schedules or urgent medical information.

Functionality:

- Patients receive notifications about their appointments and medical updates.
- Doctors receive alerts and reminders about their schedules and patient updates.
- Admins can send out important announcements and updates to all users.

1.3.8 Reporting and Analytics Module

Description: This module provides comprehensive reporting and analytics tools to support hospital management and decision-making.

Key Features:

Customizable Reports: Allows users to generate customizable reports on various aspects of hospital operations, including patient demographics, treatment outcomes, and resource utilization.

Data Visualization: Provides data visualization tools, such as charts and graphs, for easy interpretation of data.

Performance Metrics: Tracks key performance indicators (KPIs) to monitor hospital performance and identify areas for improvement.

Export Options: Offers options to export reports in various formats, such as PDF, Excel, and CSV.

Functionality:

- Admins can generate and view reports to gain insights into hospital operations.
- Doctors can analyze patient data to improve treatment outcomes.

1.3.9 Security and Compliance Module

Description: This module ensures the security of the HMS and compliance with healthcare regulations.

Key Features:

Data Encryption: Encrypts sensitive data to protect it from unauthorized access.

Authentication and Authorization: Implements secure authentication and role-based access control to ensure that only authorized users can access specific functionalities.

Audit Trails: Maintains detailed audit trails of user activities for accountability and compliance purposes.

Regulatory Compliance: Ensures that the system complies with healthcare regulations, such as HIPAA and GDPR.

Functionality:

- All user data is securely stored and accessed.
- Compliance with regulatory standards is maintained, ensuring the confidentiality and integrity of patient and doctor information.

2. SURVEY OF TECHNOLOGIES

2.1 Software Description

The Hospital Management System (HMS) is a comprehensive software solution designed to streamline and automate various hospital operations. Developed using Python for its robust backend capabilities and Streamlit for a user-friendly web interface, the system ensures efficient data management and accessibility. MySQL is employed as the database management system, providing reliable and scalable data storage. The HMS integrates essential functionalities, including patient management, doctor scheduling, and administrative tasks, into a single, cohesive platform. With a focus on security, usability, and performance, the HMS facilitates improved healthcare delivery and operational efficiency in modern healthcare facilities.

2.2 Languages

2.2.1 Python

Python is a high-level programming language known for its readability and simplicity. It is widely used in various domains, including web development, data analysis, machine learning, and more. Python's extensive libraries and frameworks, such as Flask and Django, make it an ideal choice for backend development.

Integration with Streamlit

Python’s integration with Streamlit, a popular framework for building interactive web applications, is crucial for the HMS’s frontend development.

1. User Interface Development:
   Streamlit allows developers to create intuitive and interactive user interfaces using simple Python scripts. This is particularly useful for developing dashboards and data visualization tools for doctors, patients, and admins.
   Python functions are directly used to generate web elements like buttons, forms, tables, and charts, enabling rapid prototyping and iteration.

2. Real-Time Data Interaction:

Streamlit's ability to handle real-time data updates ensures that users can interact with the system and see immediate feedback. This is essential for dynamic features like search functionalities and data filtering.

Key Features:

- Readable and maintainable code
- Extensive standard library
- Strong community support
- Support for multiple programming paradigms
- High compatibility with other technologies
- Rapid development and prototyping

2.2.2 SQL

SQL (Structured Query Language) is the standard language for managing relational databases. MySQL, an open-source RDBMS, is used in this project for its robustness and ease of use. SQL allows for efficient querying, updating, and management of data.

Key Features:

- Efficient data manipulation
- Support for complex queries
- Transaction control
- Data integrity and security features
- Scalability and high performance
- Extensive community and documentation

3. REQUIREMENTS AND ANALYSIS

3.1 Requirement Specification

Functional Requirements

User Authentication and Authorization:

- Secure login and registration for doctors, patients, and admins.

- Role-based access control to ensure users access only permitted functionalities.

Data Management:

- CRUD (Create, Read, Update, Delete) operations for patient, doctor, and admin data.
- Viewing and searching patient and doctor records.

Dashboard Functionality:

- Personalized dashboards for doctors, patients, and admins.
- Search functionality to find patient and doctor records based on various criteria.
- Options for admins to update patient and doctor details.

Non-Functional Requirements

Security:

- Implement encryption for sensitive data.
- Use secure authentication mechanisms.

Usability:

- Ensure a user-friendly interface with intuitive navigation.
- Provide clear feedback to users on their actions.

Performance:

- Optimize database queries for fast data retrieval.
- Ensure the system can handle concurrent users without performance degradation.

3.2 Hardware and Software Requirements

Hardware Requirements:

Processor: Intel i5 or higher
RAM: 8 GB or more
Storage: 500 GB HDD or SSD
Network: Reliable internet connection

Software Requirements

Operating System: Windows 10 / Linux / macOS
Python: Version 3.8 or above
Streamlit: Latest version
MySQL: Version 5.7 or above
IDE: PyCharm or any preferred Python IDE
Browser: Google Chrome, Firefox, or any modern web browser

4. SYSTEM DESIGN

4.1 Architectural Design

Client-Server Architecture:

The system follows a client-server architecture where the client (frontend) interacts with the server (backend) through API calls.

Modular Design:

The system is designed in a modular fashion, separating different functionalities into distinct modules for better maintainability and scalability.

4.1.1 Database Schema

The database schema is designed to store and manage data efficiently.

The key tables include:

Doctors: Stores doctor details such as ID, name, email, phone, specialty, department ID, and password.

Patients: Stores patient details such as ID, name, email, phone, address, DOB, doctor ID, and password.

Admins: Stores admin details such as ID, name, email, phone, role, department ID, and password.

Departments: Stores department information such as ID, name, head, phone, location, and capacity.

Rooms: Stores room availability information such as room ID, type, capacity, and status.

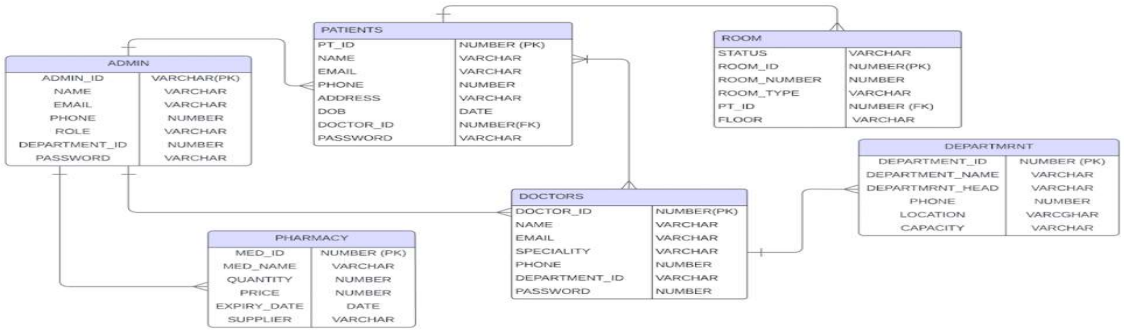
4.1.2 Entity-Relationship Diagram (ERD)

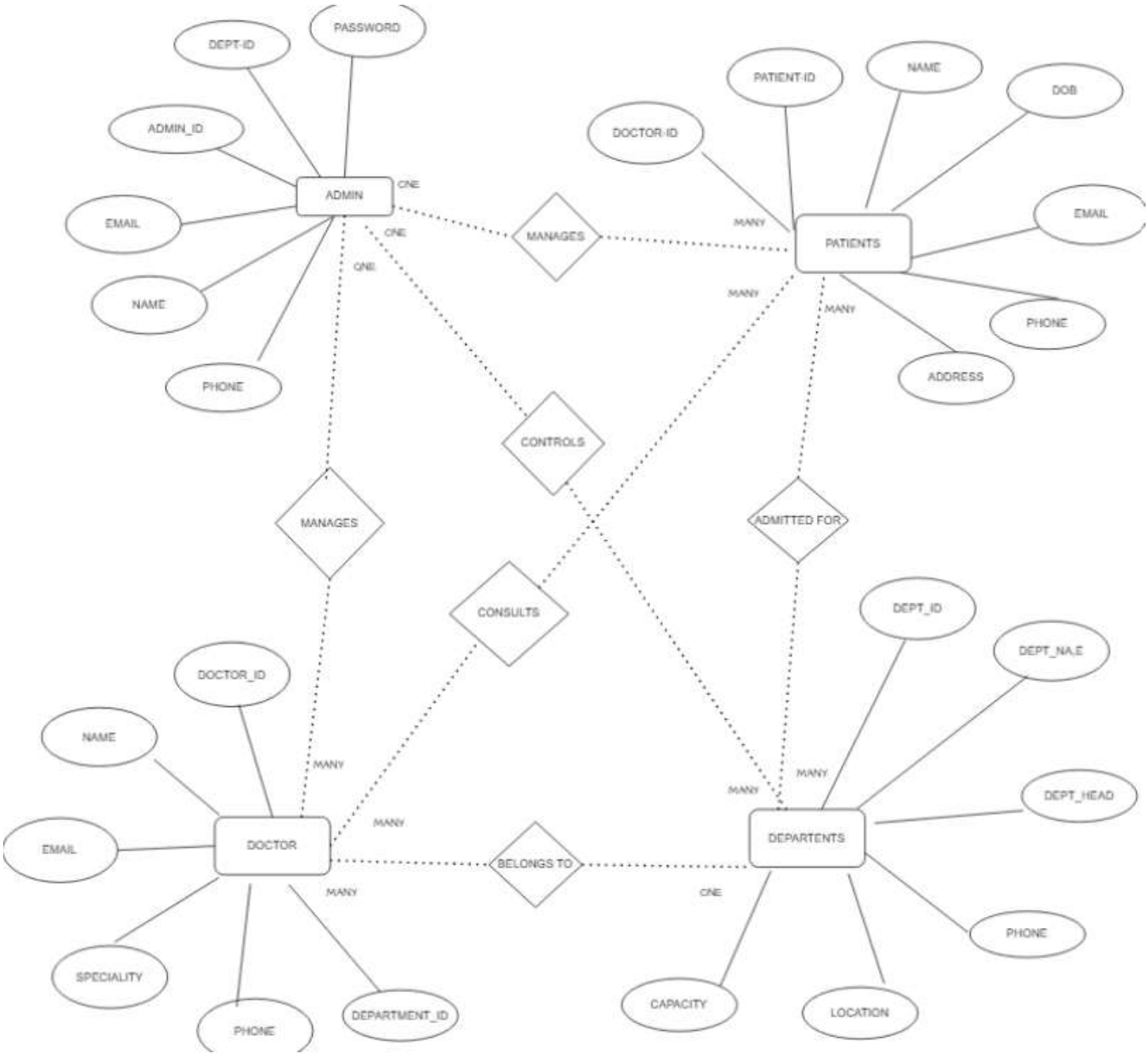
The ERD represents the relationships between different entities in the system. Key relationships include:

Doctor works in a Department: A doctor is associated with a specific department.

Patient is assigned to a Doctor: A patient is assigned to a specific doctor for treatment.

Admin management: An admin oversees the management of departments and rooms in the hospital.


PATIENTS
PT_ID NUMBER (PK)
NAME VARCHAR
EMAIL VARCHAR
PHONE NUMBER
ADDRESS VARCHAR
DOB DATE
DOCTOR_ID NUMBER(FK)
PASSWORD VARCHAR
ROOM
STATUS VARCHAR
ROOM_ID NUMBER(PK)
ROOM_NUMBER NUMBER
ROOM_TYPE VARCHAR
PT_ID NUMBER (FK)
FLOOR VARCHAR
ADMIN
ADMIN_ID VARCHAR(PK)
NAME VARCHAR
EMAIL VARCHAR
PHONE NUMBER
ROLE VARCHAR
DEPARTMENT_ID NUMBER
PASSWORD VARCHAR
DEPARTMRNT
DEPARTMENT_ID NUMBER (PK)
DEPARTMENT_NAME VARCHAR
DEPARTMRNT_HEAD VARCHAR
PHONE NUMBER
LOCATION VARCGHAR
CAPACITY VARCHAR
DOCTORS
DOCTOR_ID NUMBER(PK)
NAME VARCHAR
EMAIL VARCHAR
SPECIALITY VARCHAR
PHONE NUMBER
DEPARTMENT_ID VARCHAR
PASSWORD NUMBER
PHARMACY
MED_ID NUMBER (PK)
MED_NAME VARCHAR
QUANTITY NUMBER
PRICE NUMBER
EXPIRY_DATE DATE
SUPPLIER VARCHAR


PASSWORD
DEPT ID
ADMIN_ID
ADMIN
EMAIL
NAME
PHONE
ONE
ONE
ONE
MANAGES
PATIENT ID
NAME
DOB
DOCTOR ID
EMAIL
PATIENTS
PHONE
ADDRESS
MANY
MANY
MANY
CONTROLS
MANAGES
ADMITTED FOR
CONSULTS
DOCTOR_ID
NAME
EMAIL
DOCTOR
SPECIALITY
PHONE
DEPARTMENT_ID
MANY
MANY
MANY
BELONGS TO
ONE
DEPT_ID
DEPT_NAME
DEPT_HEAD
DEPARTENTS
PHONE
CAPACITY
LOCATION

5.PROGRAM CODE

 # schema.py

```
import mysql.connector

def create_connection():
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="Varsha7279@",
        database="hospital_management"
    )
    return conn


def create_tables(conn):
    cursor = conn.cursor()

    cursor.execute('''CREATE TABLE IF NOT EXISTS Departments (
                        department_id INT AUTO_INCREMENT PRIMARY KEY,
                        dept_name VARCHAR(255) NOT NULL,
                        dept_head VARCHAR(255),
                        phone VARCHAR(20),
                        location VARCHAR(255),
                        capacity INT
                    );''')

    cursor.execute('''CREATE TABLE IF NOT EXISTS Doctors (
                        doctor_id INT AUTO_INCREMENT PRIMARY KEY,
                        name VARCHAR(255) NOT NULL,
                        email VARCHAR(255) UNIQUE,
                        specialty VARCHAR(255),
                        phone VARCHAR(20),
                        department_id INT,
                        password VARCHAR(255),
                        FOREIGN KEY (department_id) REFERENCES Departments(department_id)
```

```
        );"")

cursor.execute("'CREATE TABLE IF NOT EXISTS Patients (
            patient_id INT AUTO_INCREMENT PRIMARY KEY,
            name VARCHAR(255) NOT NULL,
            email VARCHAR(255) UNIQUE,
            phone VARCHAR(20),
            address VARCHAR(255),
            dob DATE,
            doctor_id INT,
            password VARCHAR(255),
            FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
        );"")

cursor.execute("'CREATE TABLE IF NOT EXISTS Admins (
            admin_id INT AUTO_INCREMENT PRIMARY KEY,
            name VARCHAR(255) NOT NULL,
            email VARCHAR(255) UNIQUE,
            phone VARCHAR(20),
            role VARCHAR(255),
            department_id INT,
            password VARCHAR(255),
            FOREIGN KEY (department_id) REFERENCES Departments(department_id)
        );"")

cursor.execute("'CREATE TABLE IF NOT EXISTS Pharmacy (
            med_id INT AUTO_INCREMENT PRIMARY KEY,
            med_name VARCHAR(255) NOT NULL,
            quantity INT,
            price DECIMAL(10, 2),
            expiry_date DATE,
            supplier VARCHAR(255)
        );"")

cursor.execute("'CREATE TABLE IF NOT EXISTS Rooms (
            room_id INT AUTO_INCREMENT PRIMARY KEY,
            room_number VARCHAR(50) NOT NULL,
            room_type VARCHAR(255),
            status VARCHAR(50),
            patient_id INT,
```

```
                floor INT,
                FOREIGN KEY (patient_id) REFERENCES Patients(patient_id)
            );"")

    conn.commit()


if _name_ == "_main_":
    conn = create_connection()
    create_tables(conn)
    conn.close()
```

#login.py

```
from database.schema import create_connection

def login_user(table, email, password):
    conn = create_connection()
    cursor = conn.cursor()
    query = f"SELECT * FROM {table} WHERE email=%s AND password=%s"
    cursor.execute(query, (email, password))
    row = cursor.fetchone()
    conn.close()
    return row

def check_password_null(table, email):
    conn = create_connection()
    cursor = conn.cursor()
    query = f"SELECT password FROM {table} WHERE email=%s"
    cursor.execute(query, (email,))
    result = cursor.fetchone()
    conn.close()
    if result and result[0] is None:
        return True
    return False

def set_password(table, email, new_password):
    conn = create_connection()
    cursor = conn.cursor()
```

```
    query = f"UPDATE {table} SET password=%s WHERE email=%s"
    cursor.execute(query, (new_password, email))
    conn.commit()
    conn.close()
```

signup.py

```
from database.schema import create_connection

def insert_user(table, name, email, phone, password, **kwargs):
    conn = create_connection()
    cursor = conn.cursor()
    if table == 'Doctors':
        specialty = kwargs.get('specialty')
        department_id = kwargs.get('department_id')
        cursor.execute('INSERT INTO Doctors (name, email, specialty, phone, department_id,
password) VALUES (%s, %s, %s, %s, %s, %s)',
                       (name, email, specialty, phone, department_id, password))
    elif table == 'Patients':
        address = kwargs.get('address')
        dob = kwargs.get('dob')
        doctor_id = kwargs.get('doctor_id')
        cursor.execute('INSERT INTO Patients (name, email, phone, address, dob, doctor_id,
password) VALUES (%s, %s, %s, %s, %s, %s, %s)',
                       (name, email, phone, address, dob, doctor_id, password))
    elif table == 'Admins':
        role = kwargs.get('role')
        department_id = kwargs.get('department_id')
        cursor.execute('INSERT INTO Admins (name, email, phone, role, department_id, password)
VALUES (%s, %s, %s, %s, %s, %s)',
                       (name, email, phone, role, department_id, password))
    conn.commit()
    conn.close()
```

#onboarding.py

```
import streamlit as st
import login
```

```
def main():
    st.sidebar.title("Welcome to Hospital Management System")

    if "page" in st.experimental_get_query_params():
        page = st.experimental_get_query_params()["page"][0]
        if page == "doctor_dashboard":
            import doctor_dashboard
            doctor_dashboard.doctor_dashboard()
        # Add similar imports and calls for patient and admin dashboards
    else:
        login.main()
```

app.py

```
import streamlit as st
from login import login_user, check_password_null, set_password
from signup import insert_user
from database.schema import create_connection
import doctor_dashboard
import patient_dashboard
import admin_dashboard


def view_data(table):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute(f'SELECT * FROM {table}')
    rows = cursor.fetchall()
    conn.close()
    return rows


st.title('Hospital Management System')

if 'authenticated' not in st.session_state:
    st.session_state.authenticated = False
    st.session_state.user_role = None
```

```
def show_login(user_type):
    email = st.text_input('Email', key=f"{user_type}_email")
    password = st.text_input('Password', type='password', key=f"{user_type}_password")

    form = st.form(key=f"{user_type}_form")
    new_password = form.text_input("New Password", type="password",
key=f"{user_type}_new_password")
    confirm_password = form.text_input("Confirm Password", type="password",
key=f"{user_type}_confirm_password")

    submit_button = form.form_submit_button("Set Password")

    if submit_button:
        if new_password == confirm_password:
            table = user_type + 's'
            set_password(table, email, new_password)
            st.success("Password set successfully. You can now login.")
            st.experimental_rerun()
        else:
            st.error("Passwords do not match.")

    if st.button('Login', key=f"{user_type}_login_button"):
        table = user_type + 's'
        if check_password_null(table, email):
            st.warning("Password not set. Please set your password.")
            form  # Display the form for setting the password
        else:
            user = login_user(table, email, password)
            if user:
                st.session_state.authenticated = True
                st.session_state.user_role = user_type
                st.session_state.user_id = user[0]
                st.success(f"Welcome, {user[1]}!")
            else:
                st.error("Invalid Credentials")


def show_register(user_type):
```

```
    st.subheader(f'{user_type} Register')
    name = st.text_input('Name', key=f"{user_type}_name")
    email = st.text_input('Email', key=f"{user_type}_email_reg")
    phone = st.text_input('Phone', key=f"{user_type}_phone")
    password = st.text_input('Password', type='password', key=f"{user_type}_password_reg")
    confirm_password = st.text_input('Confirm Password', type='password',
key=f"{user_type}_confirm_password")

    if password != confirm_password:
        st.error('Passwords do not match')
    else:
        if user_type == 'Doctor':
            specialty = st.text_input('Specialty', key=f"{user_type}_specialty")
            department_id = st.number_input('Department ID', min_value=1,
key=f"{user_type}_department_id")
            if st.button('Register', key=f"{user_type}_register_button"):
                insert_user('Doctors', name, email, phone, password, specialty=specialty,
department_id=department_id)
                st.success('Doctor registered successfully!')

        elif user_type == 'Patient':
            address = st.text_input('Address', key=f"{user_type}_address")
            dob = st.date_input('Date of Birth', key=f"{user_type}_dob")
            doctor_id = st.number_input('Doctor ID', min_value=1, key=f"{user_type}_doctor_id")
            if st.button('Register', key=f"{user_type}_register_button"):
                insert_user('Patients', name, email, phone, password, address=address, dob=dob,
doctor_id=doctor_id)
                st.success('Patient registered successfully!')

        elif user_type == 'Admin':
            role_desc = st.text_input('Role Description', key=f"{user_type}_role_desc")
            department_id = st.number_input('Department ID', min_value=1,
key=f"{user_type}_department_id")
            if st.button('Register', key=f"{user_type}_register_button"):
                insert_user('Admins', name, email, phone, password, role=role_desc,
department_id=department_id)
                st.success('Admin registered successfully!')


if not st.session_state.authenticated:
```

```
    st.sidebar.header('User Login')
    user_type = st.sidebar.selectbox('Login as:', ['Doctor', 'Patient', 'Admin'])

    if user_type:
        st.subheader(f"Login as {user_type}")
        show_login(user_type)

if st.session_state.authenticated:
    st.header("Dashboard")

    if st.session_state.user_role == 'Doctor':
        doctor_dashboard.doctor_dashboard()
        # Add more doctor-specific functionality here

    elif st.session_state.user_role == 'Patient':
        patient_dashboard.patient_dashboard()
        # Add more patient-specific functionality here

    elif st.session_state.user_role == 'Admin':
        admin_dashboard.admin_dashboard()
        # Add more admin-specific functionality here

    st.sidebar.button('Logout', on_click=lambda: st.session_state.update(
        {'authenticated': False, 'user_role': None, 'user_id': None}))
```

#admin_dashboard.py

```
import streamlit as st
import mysql.connector
from database.schema import create_connection

def load_admin_details(admin_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM admins WHERE admin_id = %s", (admin_id,))
    admin_details = cursor.fetchone()
    conn.close()
    return admin_details
```

```
def search_records(table, sort_by, search_term):
    conn = create_connection()
    cursor = conn.cursor()
    query = f"SELECT * FROM {table} WHERE {sort_by} LIKE %s"
    cursor.execute(query, (f"%{search_term}%",))
    results = cursor.fetchall()
    conn.close()
    return results

def update_patient_data(patient_id, name, email, phone, address, dob, doctor_id):
    conn = create_connection()
    cursor = conn.cursor()
    #query = """
      # UPDATE patients
      # SET name = %s, email = %s, phone = %s, address = %s, dob = %s, doctor_id = %s
       #WHERE patient_id = %s
    #"""
    cursor.execute(
        'INSERT INTO Patients (name, email, phone, address, dob, doctor_id) VALUES (%s, %s, %s,
%s, %s, %s)',
        (name, email, phone, address, dob, doctor_id))
    #cursor.execute(query, (name, email, phone, address, dob, doctor_id, patient_id))
    conn.commit()
    conn.close()

def update_doctor_data(doctor_id, name, email, specialty, phone, department_id):
    conn = create_connection()
    cursor = conn.cursor()
    #query = """
     #  UPDATE doctors
      # SET name = %s, email = %s, specialization = %s, phone = %s, department = %s
      # WHERE doctor_id = %s
   # """

    cursor.execute(
        'INSERT INTO Doctors (doctor_id,name, email, specialty, phone, department_id) VALUES
(%s,%s, %s, %s, %s, %s)',
        (doctor_id,name, email, specialty, phone, department_id))
    #cursor.execute(query, (name, email, specialization, phone, department, doctor_id))
    conn.commit()
```

```
    conn.close()

def admin_dashboard():
    if "user_id" not in st.session_state:
        st.warning("Please log in first.")
        st.stop()

    admin_id = st.session_state.user_id
    admin_details = load_admin_details(admin_id)

    if admin_details:
        st.title("Admin Dashboard")
        st.subheader("Personal Details")
        st.write(f"Name: {admin_details[1]}")
        st.write(f"Email: {admin_details[2]}")

        st.subheader("Search Records")
        entity = st.selectbox("Entity", ["patients", "doctors"])
        sort_by = st.selectbox("Sort by", ["name", "email", "phone"])
        search_term = st.text_input("Search Term")
        search_button = st.button("Search")

        if search_button:
            results = search_records(entity, sort_by, search_term)
            if results:
                st.write(f"Found {len(results)} records:")
                for record in results:
                    st.write(record)
            else:
                st.write("No records found.")

        st.subheader("Update Data")
        update_option = st.selectbox("Update", ["Patient Data", "Doctor Data"])

        if update_option == "Patient Data":
            patient_id = st.text_input("Patient ID")
            name = st.text_input("Name")
            email = st.text_input("Email")
            phone = st.text_input("Phone")
            address = st.text_input("Address")
```

```
        date_of_birth = st.date_input("Dob")
        doctor_id = st.text_input("Doctor ID")
        update_patient_button = st.button("Update Patient Data")

        if update_patient_button:
            update_patient_data(patient_id, name, email, phone, address, date_of_birth, doctor_id)
            st.success("Patient data updated successfully.")

    elif update_option == "Doctor Data":
        doctor_id = st.text_input("Doctor ID")
        name = st.text_input("Name")
        email = st.text_input("Email")
        specialization = st.text_input("Specialty")
        phone = st.text_input("Phone")
        department = st.text_input("Department")
        update_doctor_button = st.button("Update Doctor Data")

        if update_doctor_button:
            update_doctor_data(doctor_id, name, email, specialization, phone, department)
            st.success("Doctor data updated successfully.")
```

#Doctor_dashboard.py

```
import streamlit as st
import mysql.connector
from database.schema import create_connection

def load_doctor_details(doctor_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM doctors WHERE doctor_id = %s", (doctor_id,))
    doctor_details = cursor.fetchone()
    conn.close()
    return doctor_details

def search_patients(sort_by, search_term):
    conn = create_connection()
    cursor = conn.cursor()
    query = f"SELECT * FROM patients WHERE {sort_by} LIKE %s"
```

```
    cursor.execute(query, (f"%{search_term}%",))
    patient_results = cursor.fetchall()
    conn.close()
    return patient_results

def doctor_dashboard():
    if "user_id" not in st.session_state:
        st.warning("Please log in first.")
        st.stop()

    doctor_id = st.session_state.user_id
    doctor_details = load_doctor_details(doctor_id)

    if doctor_details:
        st.title("Doctor Dashboard")
        st.subheader("Personal Details")
        st.write(f"Name: {doctor_details[1]}")
        st.write(f"Email: {doctor_details[2]}")
        st.write(f"Specialization: {doctor_details[3]}")
        st.write(f"Phone: {doctor_details[4]}")
        st.write(f"Department: {doctor_details[5]}")

        st.subheader("Search Patients")
        sort_by = st.selectbox("Sort by", ["name", "email", "age", "gender"])
        search_term = st.text_input("Search Term")
        search_button = st.button("Search")

        if search_button:
            results = search_patients(sort_by, search_term)
            if results:
                st.write(f"Found {len(results)} patients:")
                for patient in results:
                    with st.expander(str(patient[0])):
                        st.write(f"Name: {patient[1]}")
                        st.write(f"Email: {patient[2]}")
                        st.write(f"Age: {patient[3]}")
                        st.write(f"Gender: {patient[4]}")
            else:
                st.write("No patients found.")
    else:
```

```
        st.write("Doctor details not found.")

if _name_ == "_main_":
    doctor_dashboard()
```

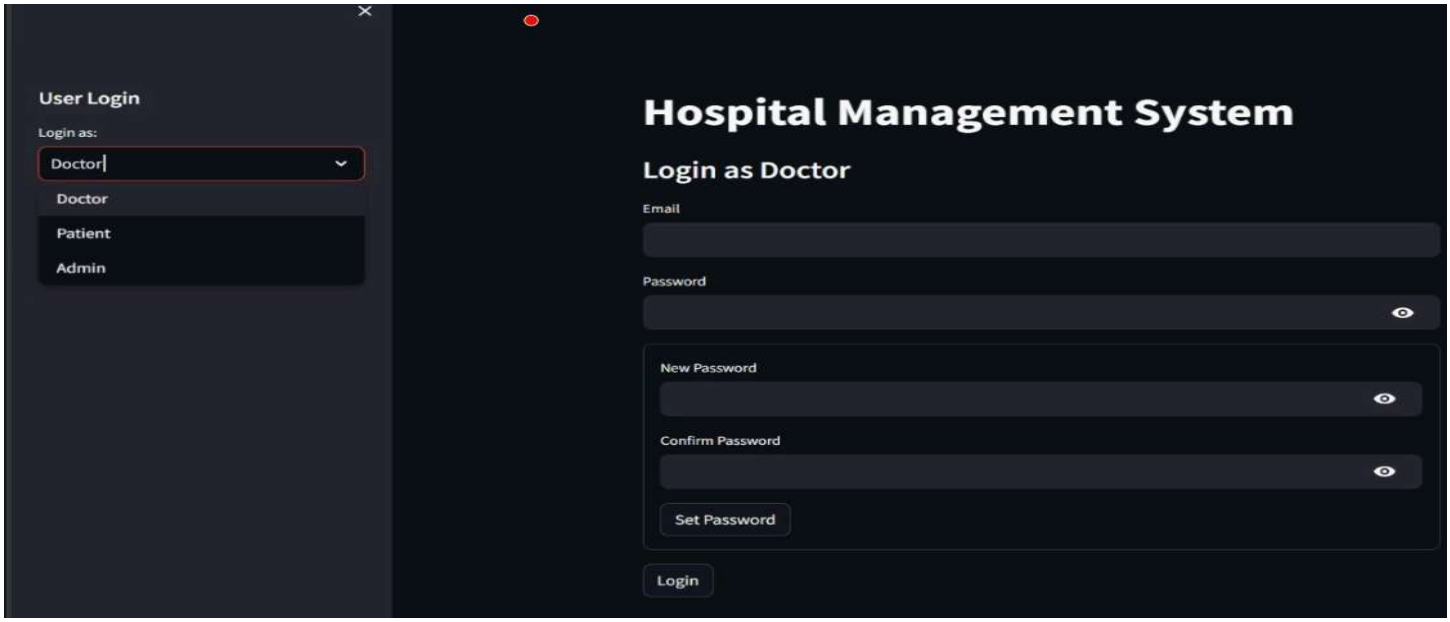
#patient_dashboard.py

```
import streamlit as st
import mysql.connector
from database.schema import create_connection

def load_patient_details(patient_id):
    conn = create_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM patients WHERE patient_id = %s", (patient_id,))
    patient_details = cursor.fetchone()
    conn.close()
    return patient_details

def search_doctors(sort_by, search_term):
    conn = create_connection()
    cursor = conn.cursor()
    query = f"SELECT * FROM doctors WHERE {sort_by} LIKE %s"
    cursor.execute(query, (f"%{search_term}%",))
    doctor_results = cursor.fetchall()
    conn.close()
    return doctor_results

def patient_dashboard():
    if "user_id" not in st.session_state:
        st.warning("Please log in first.")
        st.stop()

    patient_id = st.session_state.user_id
    patient_details = load_patient_details(patient_id)

    if patient_details:
        st.title("Patient Dashboard")
        st.subheader("Personal Details")
        st.write(f"Name: {patient_details[1]}")
```

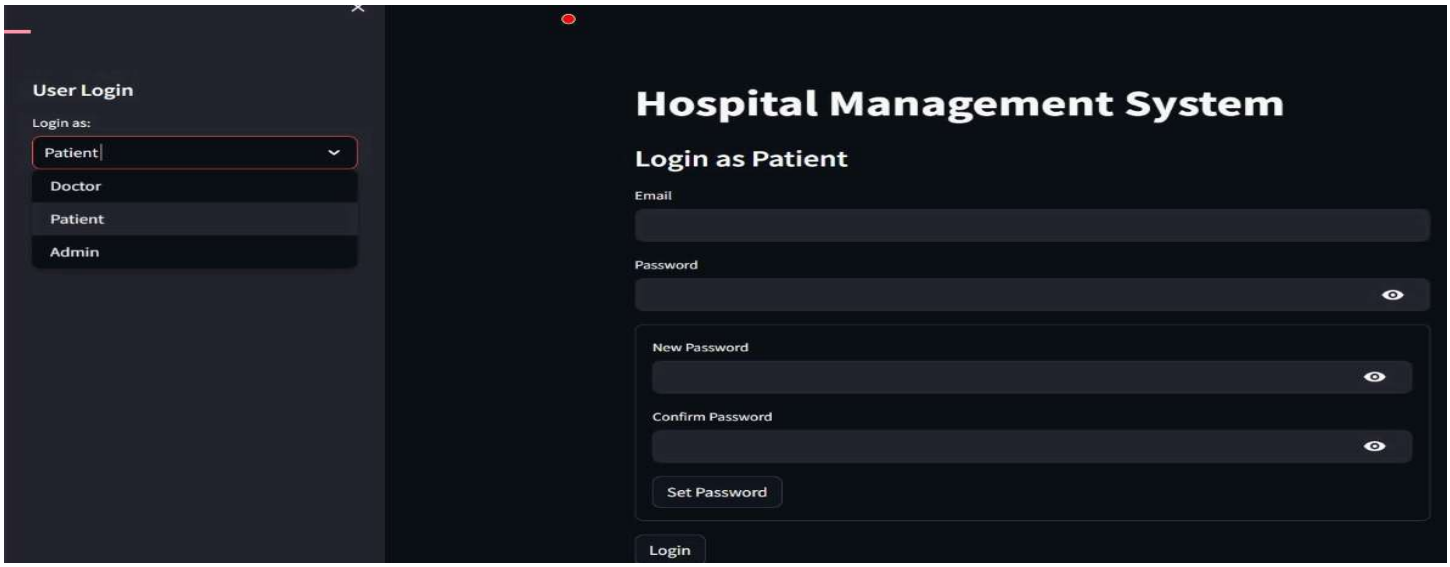
```
    st.write(f"Email: {patient_details[2]}")
    st.write(f"Age: {patient_details[3]}")
    st.write(f"Gender: {patient_details[4]}")
    st.write(f"Phone: {patient_details[5]}")
    st.write(f"Address: {patient_details[6]}")

    st.subheader("Search Doctors")
    sort_by = st.selectbox("Sort by", ["name", "email", "specialization", "phone"])
    search_term = st.text_input("Search Term")
    search_button = st.button("Search")

    if search_button:
        results = search_doctors(sort_by, search_term)
        if results:
            st.write(f"Found {len(results)} doctors:")
            for doctor in results:
              with st.expander(str(doctor[0])):
                st.write(f"Name: {doctor[1]}")
                st.write(f" Email: {doctor[2]}")
                st.write(f"Specialization: {doctor[3]}")
                st.write(f"Phone: {doctor[4]}")
        else:
            st.write("No doctors found.")
else:
    st.write("Patient details not found.")
```

6.RESULTS AND DISCUSSION

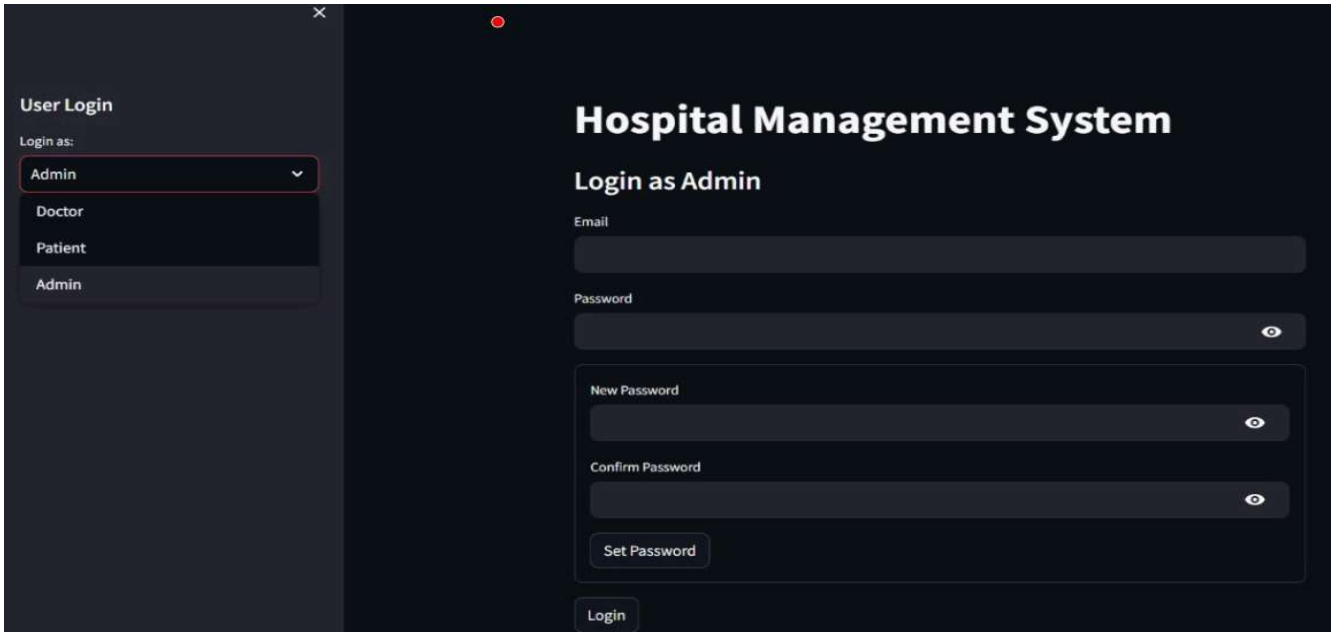
DOCTOR LOGIN PAGE:



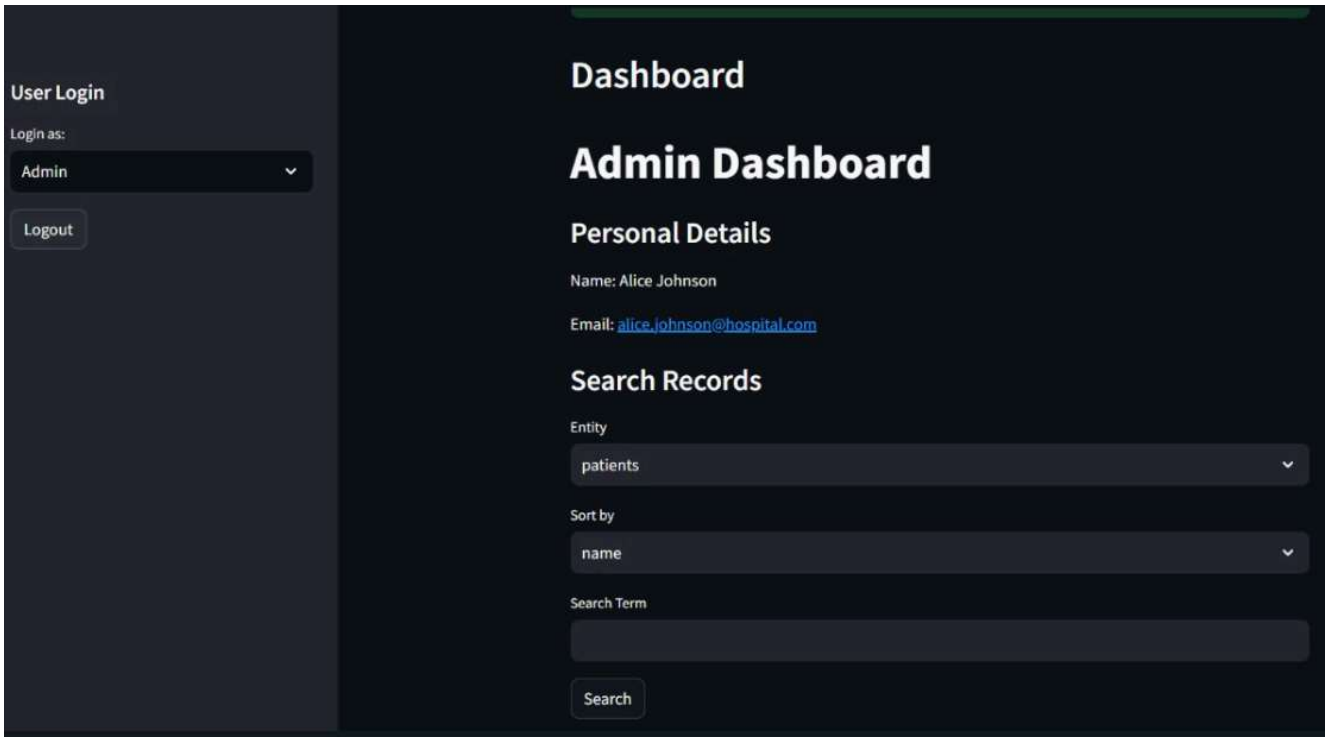
PATIENT LOGIN PAGE:



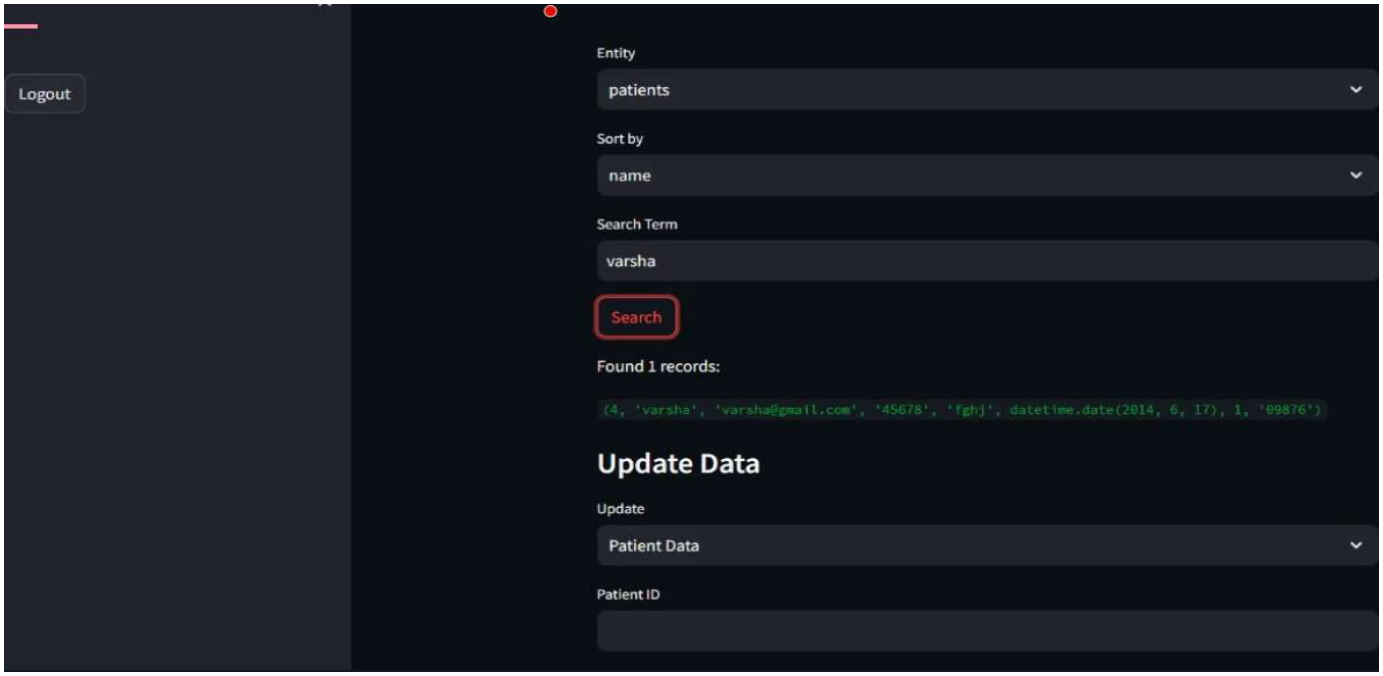
ADMIN LOGIN PAGE:



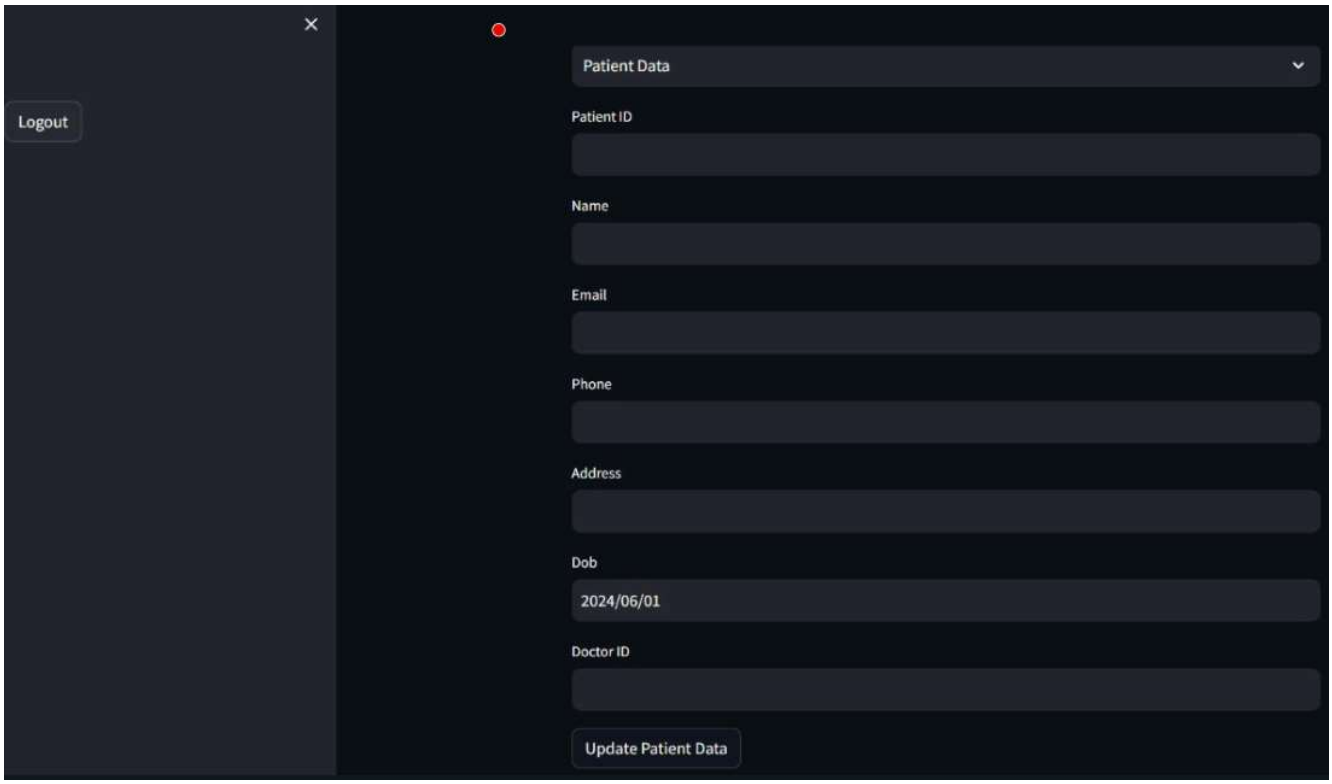
ADMIN DASHBOARD PAGE:



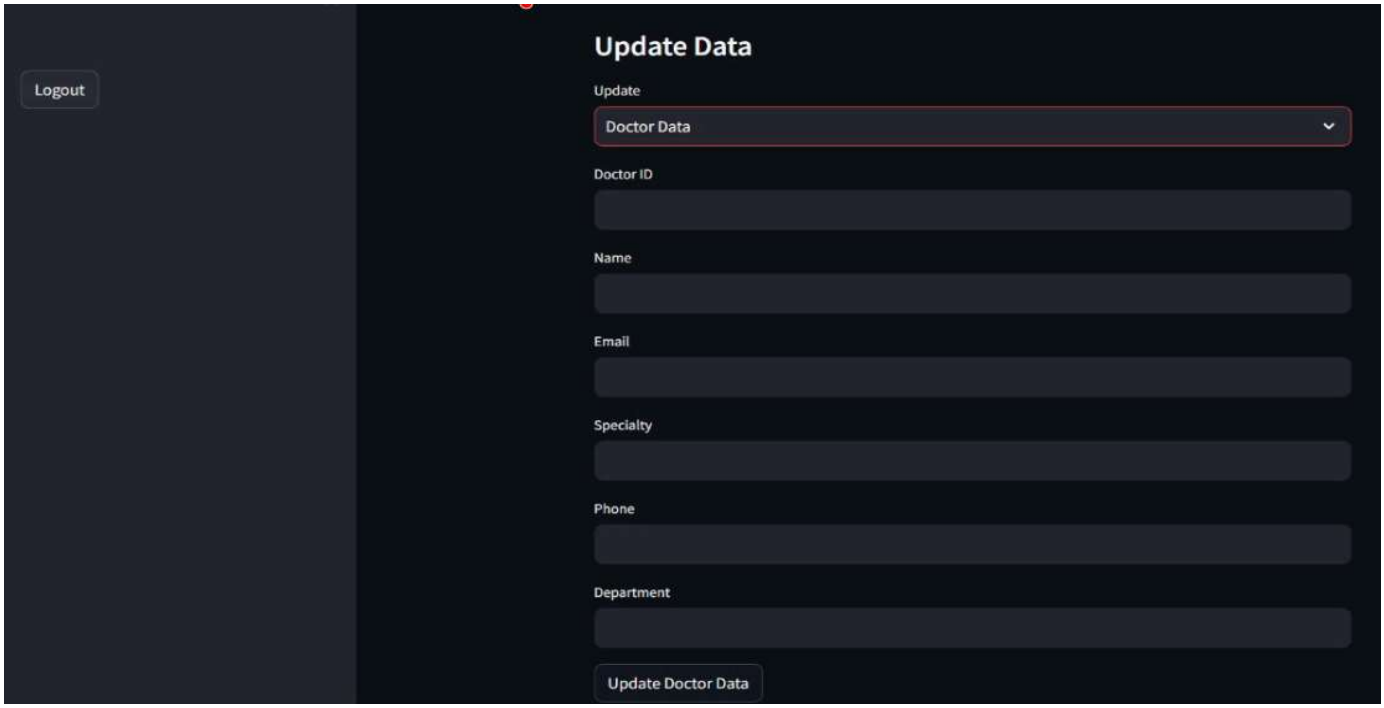
PATIENT RECORDS:



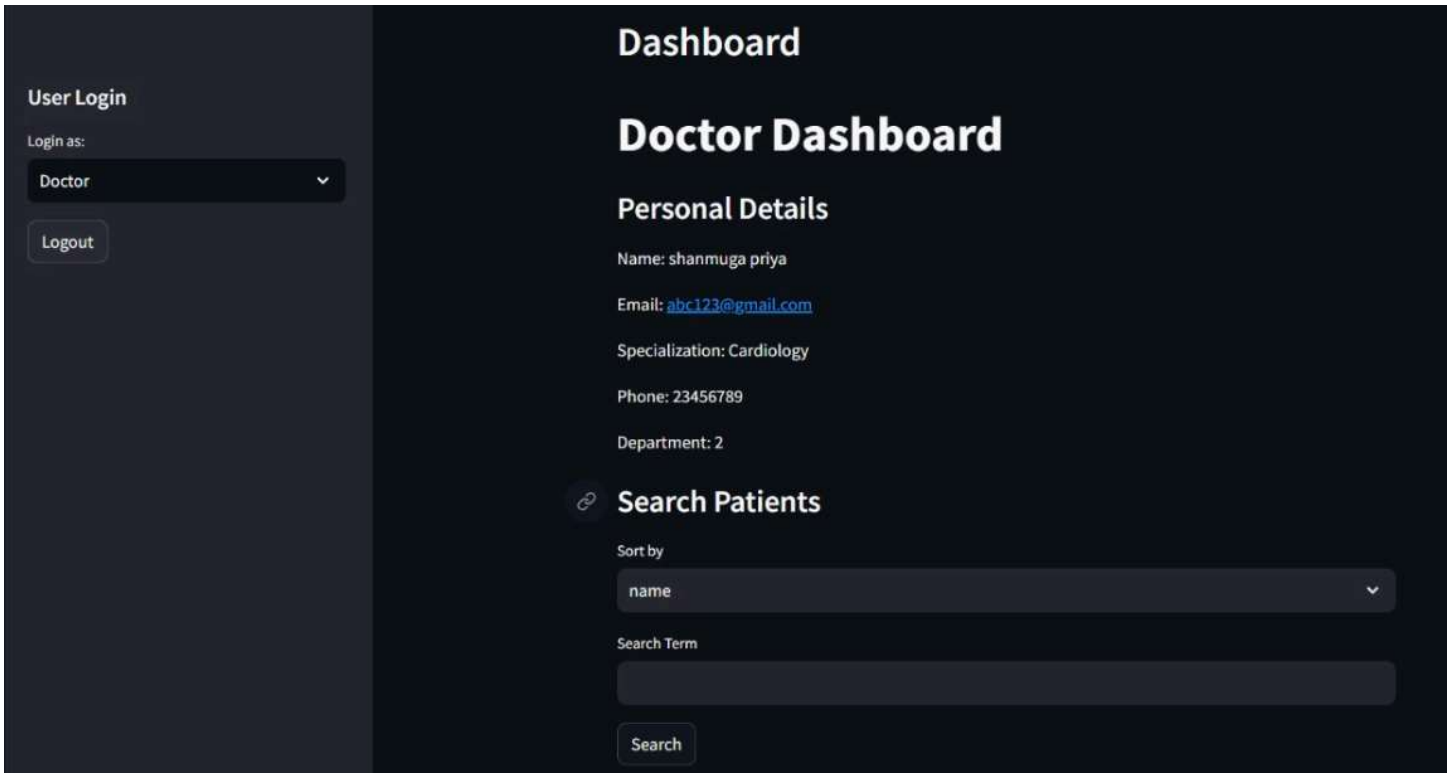
PATIENT DETAILS VIEW:



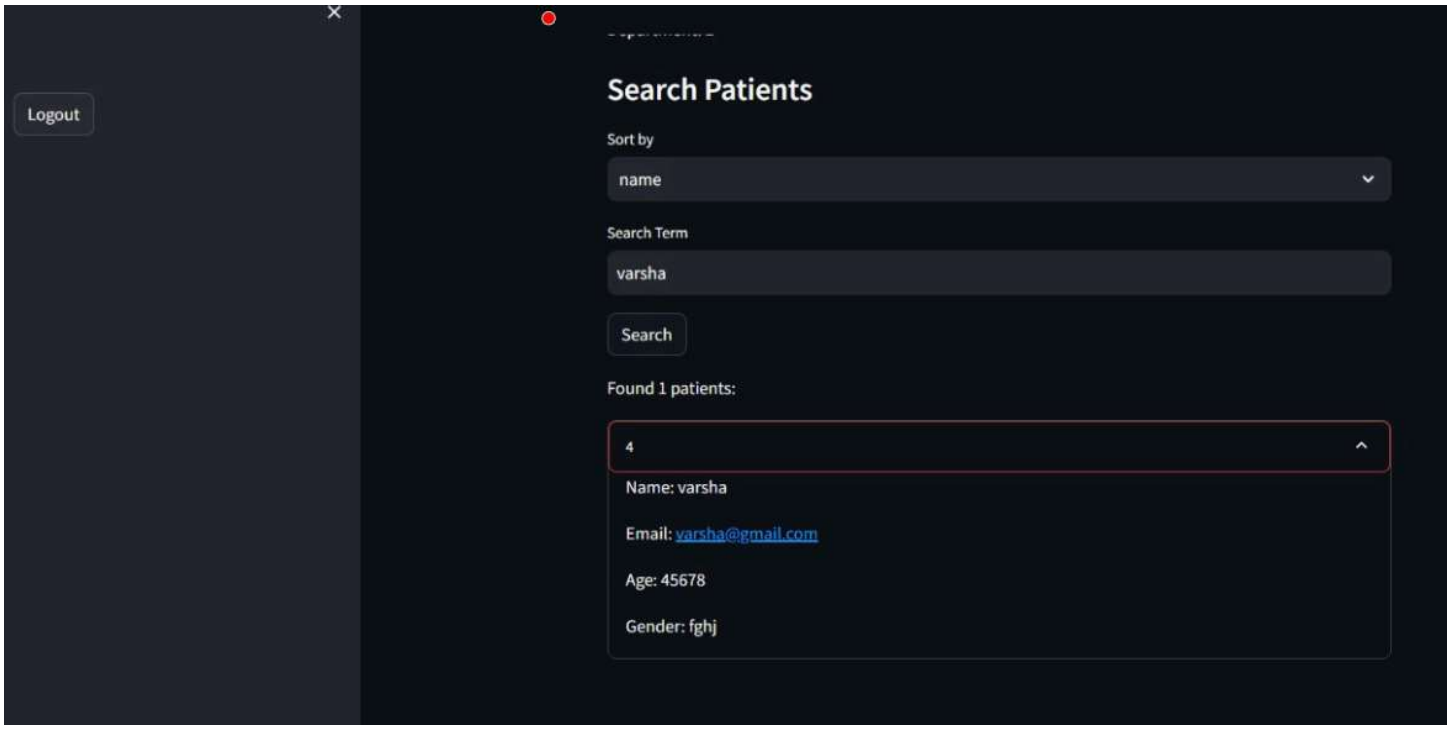
UPDATE PATIENT DETAILS:



DOCTOR DASHBOARD PAGE:



SEARCH PATIENTS:



7. CONCLUSION

The Hospital Management System (HMS) developed in Python, utilizing frameworks like Streamlit and SQL Workbench, represents a significant advancement in healthcare technology. It addresses the pressing needs of modern hospitals by streamlining administrative processes, improving patient care, and ensuring data security.

This system integrates several crucial modules, including user management, patient management, doctor management, appointment scheduling, and resource allocation. Each module is designed to enhance the efficiency and effectiveness of hospital operations. By automating routine tasks and centralizing data management, the HMS reduces administrative burdens and allows healthcare professionals to focus more on patient care.

The HMS also prioritizes data security and compliance, implementing robust authentication and authorization mechanisms, data encryption, and audit trails to ensure the confidentiality and integrity of sensitive information. This is critical in maintaining trust and meeting regulatory requirements such as HIPAA and GDPR.

Furthermore, the system's user-friendly interfaces and real-time notification features enhance communication among doctors, patients, and administrators, leading to better coordination and improved patient outcomes. The reporting and analytics module provides valuable insights into hospital performance, supporting informed decision-making and strategic planning.

In conclusion, this Hospital Management System is a comprehensive solution that leverages modern technology to address the multifaceted challenges faced by healthcare institutions. It enhances operational efficiency, improves patient care, and ensures data security, making it an indispensable tool for hospitals aiming to provide high-quality healthcare services in a rapidly evolving landscape. As the system continues to evolve, it promises to incorporate even more advanced features and functionalities, further transforming healthcare management and delivery.