



Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES

(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)

MoE, UGC & AICTE Approved; NAAC Accredited A++

Karunya Nagar, Coimbatore - 641 114, Tamil Nadu, India.

DOCTOR-PATIENT DATABASE

A Mini Project report submitted by

JONATHAN THOMAS MATHEWS - URK21CS2036

in partial fulfillment for the award of the degree of

**BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING**

under the supervision of

**Dr. E. Grace Mary Kanaga (M.Tech. Ph.D)
Professor**



**DIVISION OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTER SCIENCE AND TECHNOLOGY
KARUNYA INSTITUTE OF TECHNOLOGY AND SCIENCES**

(Declared as Deemed to be University under Sec-3 of the UGC Act, 1956)

Karunya Nagar, Coimbatore - 641 114. INDIA



Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES

(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)

MoE, UGC & AICTE Approved; NAAC Accredited A++

Karunya Nagar, Coimbatore - 641 114, Tamil Nadu, India.

DIVISION OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that the project report entitled, “**DOCTOR-PATIENT DATABASE**” is a bonafide record of Mini Project work done for the subject **20CS2016 – DATABASE MANAGEMENT SYSTEMS** during the academic year 2023-2024 by

JONATHAN THOMAS MATHEWS (Reg. No: URK21CS2036)

in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering of Karunya Institute of Technology and Sciences.

Guide Signature

Mrs. E. GRACE MARY KANAGA (M.Tech, Ph.D)

Professor

INDEX

1. Acknowledgement	4
2. Introduction	5-6
2.1 Abstract	5
2.2 Scope and Objectives	6
3. Analysis and Design	7-8
4. Implementation	9-27
4.1 Theory	9-10
4.2 Source Code	11-24
4.3 Explanation	25-27
5. Inputs and Outputs	28-32
6. Conclusion	33-34

1. ACKNOWLEDGEMENT

First and foremost, I praise and thank ALMIGHTY GOD whose blessings have bestowed in me the will power and confidence to carry out my Mini Project.

I am grateful to our beloved founders **Late. Dr. D.G.S. Dhinakaran, C.A.I.I.B, Ph.D** and **Dr. Paul Dhinakaran, M.B.A, Ph.D**, for their love and always remembering us in their prayers.

We extend Our thanks to **Dr. Dr. Prince Arulraj, Ph.D.**, our honorable Vice Chancellor, **Dr. E. J. James, Ph.D.**, and **Dr. Ridling Margaret Waller, Ph.D.**, our honorable Pro-Vice Chancellor(s) and **Dr. R. Elijah Blessing, Ph.D.**, our respected Registrar for giving me this opportunity to do the Mini Project.

I would like to thank **Dr. Ciza Thomas, Ph.D.**, Dean, School of Engineering and Technology for her direction and invaluable support to complete the same.

I would like to place my heart-felt thanks and gratitude to **Dr. J. Immanuel John Raja, M.E., Ph.D.**, Head of the Division, Computer Science and Engineering for his encouragement and guidance.

I feel it a pleasure to be indebted to, **Mrs E. Grace Mary Kanaga** for the invaluable support, advice and encouragement.

I would also like to thank all my friends and my parents who have prayed and helped me during the Mini Project.

2. INTRODUCTION

2.1 ABSTRACT

Modern healthcare systems rely on efficient and secure information management. This paper presents the design and implementation of a comprehensive doctor-patient application utilizing an SQLite database to facilitate interactions and data storage within the healthcare domain. The application comprises four main components: patient management, doctor management, medical records storage, and login credentials management. A robust database schema captures patient and doctor information, forging relationships between them and ensuring data consistency through various integrity triggers.

The "patients" table stores essential patient details such as name, date of birth, gender, and their associated doctors. The application's architecture allows for the seamless addition of new patients and doctors. A companion "doctors" table is dedicated to managing healthcare providers, recording their names and specializations.

Security and privacy are paramount in healthcare applications. The "patient_credentials" and "doctor_credentials" tables enable secure authentication for both patients and doctors, safeguarding sensitive patient data.

To maintain data integrity, this application employs triggers. These triggers validate input data, checking the age, doctor IDs, and gender provided, ensuring they conform to predefined constraints. This safeguards the database from incorrect or malicious data.

The "doctor_reports" table provides a framework for recording medical reports, linking patients and doctors. The structure allows healthcare providers to maintain patient medical records, including vital signs and diagnostic information.

The doctor-patient application presented herein demonstrates a well-structured database schema, secure authentication, and data integrity enforcement, crucial for the successful operation of modern healthcare systems. This application can be used as a foundation for healthcare providers to improve patient care, streamline processes, and maintain the security and privacy of sensitive medical data.

The doctor-patient application stands as a practical solution to the challenges of managing healthcare information in an era of digital transformation. By combining structured database management, security measures, and data integrity enforcement, it provides a solid foundation for healthcare institutions aiming to deliver quality care, streamline operations, and uphold the privacy and confidentiality of patient records. This application serves as a model for the development of advanced healthcare information systems that contribute to improved patient outcomes and enhanced doctor-patient relationships.

2.2 SCOPE AND OBJECTIVES

The doctor-patient application project represents a holistic approach to healthcare management, encompassing several key aspects:

- 1) **Patient Registration and Management:** The application facilitates the seamless registration and management of patients, including their personal information, medical history, and assigned healthcare providers (doctors).
- 2) **Doctor Management:** The system also provides a comprehensive database for doctors, including their credentials and areas of specialization. This ensures that patients are matched with the most appropriate healthcare providers.
- 3) **Security and Access Control:** Robust security measures, including user authentication, ensure that patient data remains confidential and is accessible only to authorized individuals.
- 4) **Patient-Doctor Interaction:** The application promotes effective communication between patients and their doctors, allowing for the sharing of medical reports and updates.
- 5) **Medical Records:** The system securely stores and manages detailed medical records, including diagnosis, assessments, and treatment plans.

Project Objectives:

- 1) To design and implement a database that efficiently handles patient and doctor information, establishing a robust foundation for the application.
- 2) To create user-friendly interfaces for both patients and doctors, offering intuitive access to relevant information.
- 3) To enforce stringent security measures, safeguarding sensitive medical data in compliance with privacy regulations.
- 4) To establish communication channels between patients and doctors, enabling remote monitoring and consultations.
- 5) To provide a centralized repository for medical records, ensuring easy retrieval of patient history and reports.

3. ANALYSIS AND DESIGN

The doctor-patient application is a comprehensive healthcare management system designed to streamline the interactions between healthcare providers and their patients. The analysis and design of this system encompass various critical aspects, from data storage and management to user interfaces and security features.

1. Database Design:

Analysis: The foundation of the doctor-patient application is the database, which stores and manages patient and doctor information. The database schema consists of three primary tables: patients, doctors, and patient_credentials. The patients table includes essential patient details such as name, date of birth, gender, and assigned doctor. The doctors table stores doctor information, including their name and specialization. The patient_credentials table contains login credentials for patients.

Design: The database design ensures data integrity and relationships between patients, doctors, and their respective credentials. Foreign key constraints maintain the association between patients and their assigned doctors. Additionally, triggers are implemented to enforce data validation rules, such as checking for valid ages, doctor IDs, and gender values. These measures contribute to the reliability and accuracy of the stored data.

2. User Interfaces:

Analysis: User interfaces play a pivotal role in ensuring a user-friendly experience for both patients and doctors. Patients can access and update their personal information, while doctors can view their profiles and specialization details. This analysis involves understanding the specific data fields and functionalities needed for each user type.

Design: The user interface design involves creating forms and views that allow patients to manage their profiles and communicate with their doctors. Doctors, on the other hand, can access their profiles and specialization details through their interfaces. The design ensures that data is presented in an organized and intuitive manner, enhancing user experience.

3. Security Measures:

Analysis: Security is a paramount concern, given the sensitive medical data stored in the application. Unauthorized access to patient information can lead to privacy breaches and legal consequences.

Design: To address this concern, the application incorporates robust security measures. The patient_credentials and doctor_credentials tables store login information securely. User authentication is required to access the application. Additionally, foreign key constraints are used to maintain data integrity, and triggers ensure the validation of data before insertion. All of these security features collectively safeguard patient information from unauthorized access.

4. Patient-Doctor Interaction:

Analysis: Effective communication between patients and doctors is a fundamental requirement of the application, allowing patients to share their medical reports and updates with their healthcare providers.

Design: To facilitate this interaction, the `doctor_reports` table is designed to store medical reports, including vital signs, examinations, assessments, and diagnoses. This information can be accessed and updated by both patients and doctors through their respective interfaces, enabling remote monitoring and consultations.

5. Extensibility:

Analysis: An important aspect of the design is the potential for future growth and extension of the application. Healthcare is a dynamic field with evolving needs, and the system should be adaptable to accommodate these changes.

Design: The system is designed to be extensible, allowing for the addition of new features, modules, or data fields in response to changing requirements. For example, the application can easily incorporate telemedicine capabilities, electronic health records (EHR) integration, and data analytics for improved patient care. This adaptability ensures that the application remains relevant and useful over time.

6. Scalability:

Analysis: Scalability is crucial to handle a growing number of patients and doctors who may use the application. The system must efficiently manage increased data and user loads.

Design: The database design is structured to handle scalability by optimizing queries and indexing key fields. The use of triggers for data validation helps maintain system performance even with a growing dataset. Additionally, the application's architecture can be hosted on scalable cloud infrastructure to handle increased traffic and data storage requirements.

7. User Training and Support:

Analysis: Effective user training and support are essential to ensure that patients and doctors can use the application confidently and efficiently.

Design: The design includes provisions for user training materials, guides, and a support system. Video tutorials, FAQs, and a responsive helpdesk are planned to assist users in navigating the application and addressing any queries or issues they may encounter.

4. IMPLEMENTATION

4.1 THEORY

The implementation phase of the doctor-patient application involves translating the designed system into functional code. It covers the creation and deployment of the two main components: the patient management system and the doctor's reporting system. Let's delve into the details of their implementation.

1. Patient Management System:

The patient management system is designed to handle patient records, appointments, and user authentication.

Implementation Steps:

Database Creation: SQLite is used as the relational database management system. The code includes SQL statements to create tables for patients, doctors, patient credentials, doctor reports, and doctor credentials. The schema ensures data integrity and supports relationships between patients and doctors.

Data Population: The code inserts sample data into the patient and doctor tables using the `executemany` method. This populates the database with initial patient and doctor information, including names, genders, and specialization.

User Authentication: The system provides a basic form of user authentication by creating a patient credentials table to store usernames and passwords. The `executemany` method inserts sample login data.

Triggers for Data Validation: Triggers are created to enforce data validation rules before inserting patient information. These triggers ensure that data such as age, doctor IDs, and gender conform to expected formats.

2. Doctor's Reporting System:

The doctor's reporting system focuses on storing and managing medical reports generated by doctors.

Implementation Steps:

Report Table Creation: A dedicated table, 'doctor_reports,' is created to store medical reports. This table captures essential data, including patient details, report date, vital signs, and medical assessments.

Database Relationships: Foreign key constraints are applied to establish relationships between the 'doctor_reports' table and the 'patients' and 'doctors' tables. This ensures that reports are associated with the correct patients and doctors.

Database Triggers: A trigger is used to disable foreign key constraints temporarily while inserting data to avoid conflicts, and then re-enables them afterward.

Doctor Credential Table: A table is created to manage doctor credentials, providing a secure login mechanism for doctors. Sample login data is inserted.

Implementation of Extensibility and Scalability:

The design ensures extensibility and scalability. The application structure allows for future additions of features, such as telemedicine capabilities or electronic health records integration, without major architectural changes. Additionally, the use of SQLite as a backend database facilitates scalability, and the system can be hosted on scalable cloud infrastructure to manage increased user loads efficiently.

Implementation of User Interface:

The doctor-patient application's user interface is a critical aspect of the implementation. The GUI would allow doctors to access patient records, create medical reports, and manage patients. Patients would use the interface to access their medical history and doctor details so that they may book their next appointment.

Implementation of Report Generation:

For the doctor's reporting system, the code establishes a 'doctor_reports' table for storing medical reports. The actual implementation would involve a user-friendly interface for doctors to generate, edit, and save medical reports. These reports may include structured forms for entering patient information and medical findings. Implementation would include generating PDF reports for printing or digital sharing, ensuring the records are easily accessible and can be shared securely.

Implementation of Data Security:

While the code initializes tables and triggers for data validation, a complete implementation would include comprehensive data security measures. This involves encryption of sensitive data, securing communication channels, and regular data backups to prevent data loss. It's essential to protect patient information, comply with data protection laws, and maintain the confidentiality of medical records.

4.2 SOURCE CODE

1) **medical_app.py**

```
import sqlite3
from datetime import date
conn = sqlite3.connect('medical_app.db')
cursor = conn.cursor()

cursor.execute("""
    CREATE TABLE IF NOT EXISTS patients (
        patient_id INTEGER PRIMARY KEY,
        name TEXT NOT NULL,
        dob DATE NOT NULL,
        gender TEXT NOT NULL,
        doctor_id INTEGER,
        FOREIGN KEY (doctor_id) REFERENCES doctors(doctor_id)
    );
""")

patients_data = [
    (1, 'John Doe', date(1988, 5, 12), 'Male', 1),
    (2, 'Jane Smith', date(1995, 9, 23), 'Female', 2),
    (3, 'Bob Johnson', date(1978, 11, 3), 'Male', 1),
    (4, 'Alice Brown', date(1990, 7, 15), 'Female', 3),
    (5, 'Michael Wilson', date(1973, 3, 30), 'Male', 2),
    (6, 'Sarah Davis', date(1981, 12, 8), 'Female', 1),
    (7, 'David Lee', date(1963, 8, 21), 'Male', 2),
    (8, 'Emily White', date(1996, 2, 14), 'Female', 3),
    (9, 'James Clark', date(1975, 6, 6), 'Male', 1),
    (10, 'Olivia Taylor', date(1990, 10, 7), 'Female', 2)
]

cursor.executemany("INSERT INTO patients (patient_id, name, dob, gender, doctor_id)
VALUES (?, ?, ?, ?, ?)", patients_data)

cursor.execute("""
    CREATE TABLE IF NOT EXISTS doctors (
        doctor_id INTEGER PRIMARY KEY,
        name TEXT,
        specialization TEXT
    )
""")

doctors_data = [
    (1, "Dr. John Doe", "Cardiologist"),
    (2, "Dr. Jane Smith", "Pediatrician"),
    (3, "Dr. Robert Johnson", "Dermatologist"),
    (4, "Dr. Rita Suresh", "General Practioner")
]
```

```

]

cursor.executemany("INSERT INTO doctors (doctor_id, name, specialization) VALUES (?,
?, ?)", doctors_data)

cursor.execute("""
    CREATE TABLE IF NOT EXISTS patient_credentials (
        patient_id INTEGER PRIMARY KEY,
        username TEXT,
        password TEXT NOT NULL
    );
""")

login_data = [
    (1, "jeevan", "jk123"),
    (2, "jobin", "joby746"),
    (3, "sharon", "sherry1234"),
    (4, "harold", "cooper1968")
]

cursor.executemany("INSERT INTO patient_credentials (patient_id, username, password)
VALUES (?, ?, ?)", login_data)

cursor.execute("""
CREATE TABLE IF NOT EXISTS doctor_reports (
    report_id INTEGER PRIMARY KEY,
    patient_id INTEGER,
    patient_name TEXT,
    doctor_id INTEGER,
    report_date TEXT,
    blood_pressure TEXT,
    pulse_rate TEXT,
    respiratory_rate TEXT,
    body_temperature TEXT,
    oxygen_saturation TEXT,
    head_exam TEXT,
    chest_exam TEXT,
    abdominal_exam TEXT,
    extremities_exam TEXT,
    assessment TEXT,
    diagnosis TEXT,
    FOREIGN KEY (patient_id) REFERENCES patients (patient_id),
    FOREIGN KEY (doctor_id) REFERENCES doctors (doctor_id)
)
""")

cursor.execute("""
    CREATE TABLE IF NOT EXISTS doctor_credentials (
        doctor_id INTEGER PRIMARY KEY,
        username TEXT,

```

```

        password TEXT NOT NULL
    );
    ")

login1_data = [
    (1, "johndoe", "jd123"),
    (2, "janesmith", "janes123"),
    (3, "robertjohnson", "robert123"),
    (4, "ritasuresh", "rita123")
]

cursor.executemany("INSERT INTO doctor_credentials (doctor_id, username, password)
VALUES (?, ?, ?)", login1_data)

cursor.execute("PRAGMA foreign_keys = OFF;")
cursor.execute("CREATE TRIGGER check_age
BEFORE INSERT ON patients
BEGIN
    SELECT CASE
        WHEN (NEW.age NOT NULL AND NEW.age NOT LIKE '%[^0-9]%') THEN
            RAISE(IGNORE)
        ELSE
            RAISE(ROLLBACK, 'Age must be a valid integer')
        END;
END;")

cursor.execute("CREATE TRIGGER check_doctor_id
BEFORE INSERT ON patients
BEGIN
    SELECT CASE
        WHEN (NEW.doctor_id NOT NULL AND NEW.doctor_id NOT LIKE '%[^0-9]%')
THEN
            RAISE(IGNORE)
        ELSE
            RAISE(ROLLBACK, 'Doctor ID must be a valid integer')
        END;
END;")

cursor.execute("CREATE TRIGGER check_gender
BEFORE INSERT ON patients
BEGIN
    SELECT CASE
        WHEN (NEW.gender IS NULL OR NEW.gender IN ('Male', 'Female', 'Other')) THEN
            RAISE(IGNORE)
        ELSE
            RAISE(ROLLBACK, 'Invalid gender')
        END;
END;")

cursor.execute("PRAGMA foreign_keys = ON;")

```

```
conn.commit()
conn.close()
```

2) login.py

```
import tkinter as tk
from tkinter import ttk, messagebox
import sqlite3
from datetime import datetime
from tkcalendar import DateEntry
```

```
current_doctor_id = None
current_patient_id = None
```

```
def clear_entry_fields():
    doctor_username_entry.delete(0, "end")
    doctor_password_entry.delete(0, "end")
    patient_username_entry.delete(0, "end")
    patient_password_entry.delete(0, "end")
```

```
def doctor_login():
    global current_doctor_id
    username = doctor_username_entry.get()
    password = doctor_password_entry.get()
    conn = sqlite3.connect('medical_app.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM doctor_credentials WHERE username = ? AND
password = ?", (username, password))
    doctor = cursor.fetchone()
    conn.close()
    if doctor:
        current_doctor_id = doctor[0]
        clear_entry_fields()
        open_doctor_app()
    else:
        messagebox.showerror("Login Error", "Invalid credentials for doctor.")
        clear_entry_fields()
```

```
def patient_login():
    global current_patient_id
    username = patient_username_entry.get()
    password = patient_password_entry.get()
    conn = sqlite3.connect('medical_app.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM patient_credentials WHERE username = ? AND
password = ?", (username, password))
    patient = cursor.fetchone()
    conn.close()
    if patient:
```

```

        current_patient_id = patient[0]
        clear_entry_fields()
        open_patient_app()
    else:
        messagebox.showerror("Login Error", "Invalid credentials for patient.")
        clear_entry_fields()

def open_doctor_app():
    doctor_window = tk.Toplevel(root)
    doctor_window.title("Doctor Application")
    doctor_window.geometry("800x600")
    notebook = ttk.Notebook(doctor_window)
    notebook.pack(fill="both", expand=True)

def add_patient():
    name = add_name_entry.get()
    dob = add_dob_entry.get_date().strftime('%Y-%m-%d')
    gender = add_gender_var.get()
    doctor_id = add_doctorid_entry.get()
    try:
        conn = sqlite3.connect('medical_app.db')
        cursor = conn.cursor()
        cursor.execute("INSERT INTO patients (name, dob, gender, doctor_id) VALUES (?, ?, ?), (name, dob, gender, doctor_id)", (name, dob, gender, doctor_id))
        conn.commit()
        conn.close()
        messagebox.showinfo("Success", "Patient details added successfully!")
    except sqlite3.Error as e:
        messagebox.showerror("Error", str(e))

def view_patients():
    if current_doctor_id is not None:
        view_window = tk.Toplevel(root)
        view_window.title("View Patients")
        view_window.configure(bg="lightgray")
        try:
            conn = sqlite3.connect('medical_app.db')
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM patients WHERE doctor_id=?",
(current_doctor_id,))
            patients = cursor.fetchall()
            text_widget = tk.Text(view_window)
            text_widget.config(borderwidth=1, relief="solid")
            text_widget.pack()
            for patient in patients:
                text_widget.insert(tk.END, f"Patient ID: {patient[0]}\n")
                text_widget.insert(tk.END, f"Name: {patient[1]}\n")
                text_widget.insert(tk.END, f"Age: {patient[2]}\n")
                text_widget.insert(tk.END, f"Gender: {patient[3]}\n")
                text_widget.insert(tk.END, f"Doctor ID: {patient[4]}\n")

```

```

        text_widget.insert(tk.END, "\n")
    conn.close()
except sqlite3.Error as e:
    messagebox.showerror("Error", str(e))
else:
    messagebox.showerror("Error", "No doctor is currently logged in.")

def search_patients():
    search_term = search_entry.get()
    conn = sqlite3.connect('medical_app.db')
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT * FROM patients WHERE name LIKE ? OR patient_id = ? AND doctor_id = ?", ('%' + search_term + '%', search_term, current_doctor_id))
        patients = cursor.fetchall()
        if patients:
            search_results_window = tk.Toplevel(root)
            search_results_window.title("Search Results")
            text_widget = tk.Text(search_results_window)
            text_widget.pack()
            for patient in patients:
                text_widget.insert(tk.END, f"Patient ID: {patient[0]}\n")
                text_widget.insert(tk.END, f"Name: {patient[1]}\n")
                text_widget.insert(tk.END, f"Age: {patient[2]}\n")
                text_widget.insert(tk.END, f"Gender: {patient[3]}\n")
                text_widget.insert(tk.END, f"Doctor ID: {patient[4]}\n")
                text_widget.insert(tk.END, "\n")
        except sqlite3.Error as e:
            messagebox.showerror("Error", str(e))
    finally:
        conn.close()

def update_patient():
    try:
        patient_id = new_patient_id_entry.get()
        new_name = new_name_entry.get()
        new_dob = new_dob_entry.get_date().strftime('%Y-%m-%d')
        new_gender = new_gender_var.get()
        try:
            new_doctor_id = int(new_doctor_id_entry.get())
        except ValueError:
            messagebox.showerror("Error", "Invalid doctor ID. Please enter valid ID.")
        conn = sqlite3.connect('medical_app.db')
        cursor = conn.cursor()
        cursor.execute("UPDATE patients SET name=?, dob=?, gender=?, doctor_id=? WHERE patient_id=?", (new_name, new_dob, new_gender, new_doctor_id, patient_id))
        conn.commit()
        conn.close()
        messagebox.showinfo("Success", "Patient details updated successfully!")

```



```

except ValueError:
    messagebox.showerror("Error", "Invalid patient ID or doctor ID. Please enter valid
IDs.")
except sqlite3.Error as e:
    messagebox.showerror("Error", str(e))

def view_doctors():
    conn = sqlite3.connect('medical_app.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM doctors")
    doctors = cursor.fetchall()
    text_widget = tk.Text(view_doctor)
    text_widget.pack()
    for doctor in doctors:
        text_widget.insert(tk.END, f"Doctor ID: {doctor[0]}\n")
        text_widget.insert(tk.END, f"Name: {doctor[1]}\n")
        text_widget.insert(tk.END, f"Specialization: {doctor[2]}\n")
        text_widget.insert(tk.END, "\n")
    conn.close()

def submit_report():
    conn = sqlite3.connect('medical_app.db')
    cursor = conn.cursor()
    patient_id = patient_id_entry.get()
    doctor_id = doctor_id_entry.get()
    patient_name = patient_name_entry.get()
    report_date = report_date_entry.get()
    blood_pressure = blood_pressure_entry.get()
    pulse_rate = pulse_rate_entry.get()
    respiratory_rate = respiratory_rate_entry.get()
    body_temperature =
body_temperature_entry.get()
    oxygen_saturation = oxygen_saturation_entry.get()
    head_exam = head_exam_entry.get("1.0", tk.END)
    chest_exam = chest_exam_entry.get("1.0", tk.END)
    abdominal_exam = abdominal_exam_entry.get("1.0", tk.END)
    extremities_exam = extremities_exam_entry.get("1.0", tk.END)
    assessment = assessment_text.get("1.0", tk.END)
    diagnosis = diagnosis_entry.get("1.0", tk.END)
    cursor.execute("INSERT INTO doctor_reports (patient_id, doctor_id, patient_name,
report_date, blood_pressure, pulse_rate, respiratory_rate, body_temperature,
oxygen_saturation, head_exam, chest_exam, abdominal_exam, extremities_exam,
assessment, diagnosis) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", (patient_id,
doctor_id, patient_name, report_date, blood_pressure, pulse_rate, respiratory_rate,
body_temperature, oxygen_saturation, head_exam, chest_exam, abdominal_exam,
extremities_exam, assessment, diagnosis))
    conn.commit()
    conn.close()
    messagebox.showinfo("Success", "Patient Report submitted successfully!")

```

```

patient_name_entry.delete(0, tk.END)
doctor_id_entry.delete(0, tk.END)
report_date_entry.delete(0, tk.END)
blood_pressure_entry.delete(0, tk.END)
pulse_rate_entry.delete(0, tk.END)
respiratory_rate_entry.delete(0, tk.END)
body_temperature_entry.delete(0, tk.END)
oxygen_saturation_entry.delete(0, tk.END)
head_exam_entry.delete("1.0", tk.END)
chest_exam_entry.delete("1.0", tk.END)
abdominal_exam_entry.delete("1.0", tk.END)
extremities_exam_entry.delete("1.0", tk.END)
assessment_text.delete("1.0", tk.END)
diagnosis_entry.delete("1.0", tk.END)

# Tab 1: Add Patient
add_patient_frame = ttk.Frame(notebook)
notebook.add(add_patient_frame, text="Add Patient")
entry_style = {"width": 22, "font": font_style}
label_style = {"font": font_style}
add_name_label = tk.Label(add_patient_frame, text="Name:", **label_style)
add_name_label.grid(row=0, column=0, padx=10, pady=5)
add_name_entry = tk.Entry(add_patient_frame, **entry_style)
add_name_entry.grid(row=0, column=1, padx=10, pady=5)
add_dob_label = tk.Label(add_patient_frame, text="DOB:", **label_style)
add_dob_label.grid(row=1, column=0, padx=10, pady=5)
add_dob_entry = DateEntry(add_patient_frame, background="darkblue",
foreground="white", width = 21, font = font_style)
add_dob_entry.grid(row=1, column=1, padx=10, pady=5)
add_gender_label = tk.Label(add_patient_frame, text="Gender:", **label_style)
add_gender_label.grid(row=2, column=0, padx=10, pady=5)
add_gender_var = tk.StringVar()
add_gender_var.set("Male")
add_gender_combobox = ttk.Combobox(add_patient_frame, textvariable=add_gender_var,
values=["Male", "Female", "Other"], state="readonly", width = 21, font = font_style)
add_gender_combobox.grid(row=2, column=1, padx=10, pady=5)
add_doctorid_label = tk.Label(add_patient_frame, text="Doctor ID:", **label_style)
add_doctorid_label.grid(row=3, column=0, padx=10, pady=5)
add_doctorid_entry = tk.Entry(add_patient_frame, **entry_style)
add_doctorid_entry.grid(row=3, column=1, padx=10, pady=5)
add_button = tk.Button(add_patient_frame, text="Add Patient", command=add_patient,
bg="blue", fg="white", font=font_style)
add_button.grid(row=4, columnspan=2, padx=10, pady=10)

# Tab 2: Search Patient
search_patient = ttk.Frame(notebook)
notebook.add(search_patient, text="Search Patients")
search_label = tk.Label(search_patient, text="Search:", font=font_style)
search_label.grid(row=0, column=0)
search_entry = tk.Entry(search_patient, width=22, font=font_style)

```

```

search_entry.grid(row=0, column=1)
search_button = tk.Button(search_patient, text="Search", command=search_patients,
bg="blue", fg="white", font=font_style)
search_button.grid(row=0, column=2)

# Tab 3: Update Patient
update_patient_frame = ttk.Frame(notebook)
notebook.add(update_patient_frame, text="Update Patient")
new_patient_id_label = tk.Label(update_patient_frame, text="Patient ID:", **label_style)
new_patient_id_label.grid(row=0, column=0, padx=10, pady=5)
new_patient_id_entry = tk.Entry(update_patient_frame, **entry_style)
new_patient_id_entry.grid(row=0, column=1, padx=10, pady=5)
new_name_label = tk.Label(update_patient_frame, text="Name:", **label_style)
new_name_label.grid(row=0, column=3, padx=10, pady=5)
new_name_entry = tk.Entry(update_patient_frame, **entry_style)
new_name_entry.grid(row=0, column=4, padx=10, pady=5)
new_dob_label = tk.Label(update_patient_frame, text="DOB:", **label_style)
new_dob_label.grid(row=1, column=0, padx=10, pady=5)
new_dob_entry = DateEntry(update_patient_frame, background="darkblue",
foreground="white", width = 21, font = font_style)
new_dob_entry.grid(row=1, column=1, padx=10, pady=5)
new_gender_label = tk.Label(update_patient_frame, text="Gender:", **label_style)
new_gender_label.grid(row=1, column=3, padx=10, pady=5)
new_gender_var = tk.StringVar()
new_gender_var.set("Male")
new_gender_combobox = ttk.Combobox(update_patient_frame,
textvariable=new_gender_var, values=["Male", "Female", "Other"], state="readonly", width
= 21, font = font_style)
new_gender_combobox.grid(row=1, column=4, padx=10, pady=5)
new_doctor_id_label = tk.Label(update_patient_frame, text="Doctor ID:", **label_style)
new_doctor_id_label.grid(row=2, column=0, padx=10, pady=5)
new_doctor_id_entry = tk.Entry(update_patient_frame, **entry_style)
new_doctor_id_entry.grid(row=2, column=1, padx=10, pady=5)
update_button = tk.Button(update_patient_frame, text="Update Patient",
command=update_patient, bg="blue", fg="white", font=font_style)
update_button.grid(row=4, column=4, padx=10, pady=10)

# Tab 4: View Doctors
view_doctor = tk.Frame(notebook)
notebook.add(view_doctor, text="View Doctor")
view_doctors_button = tk.Button(view_doctor, text="View Doctors",
command=view_doctors, bg="blue", fg="white", font=font_style)
view_doctors_button.pack()

# Tab 5: Add Patient Report
report_doctors = ttk.Frame(notebook)
notebook.add(report_doctors, text="Add Patient Report")
patient_id_label = tk.Label(report_doctors, text="Patient ID:", font=font_style)
patient_id_label.grid(row=0, column=0, padx=10, pady=5)
patient_id_entry = tk.Entry(report_doctors, width=22, font=font_style)

```

```

patient_id_entry.grid(row=0, column=1, padx=10, pady=5)
doctor_id_label = tk.Label(report_doctors, text="Doctor ID:", font=font_style)
doctor_id_label.grid(row=0, column=2, padx=10, pady=5)
doctor_id_entry = tk.Entry(report_doctors, width=22, font=font_style)
doctor_id_entry.grid(row=0, column=3, padx=10, pady=5)
patient_name_label = tk.Label(report_doctors, text="Patient Name:", font=font_style)
patient_name_label.grid(row=0, column=4, padx=10, pady=5)
patient_name_entry = tk.Entry(report_doctors, width=22, font=font_style)
patient_name_entry.grid(row=0, column=5, padx=10, pady=5)
report_date_label = tk.Label(report_doctors, text="Report Date:", font=font_style)
report_date_label.grid(row=1, column=0, padx=10, pady=5)
report_date_entry = tk.Entry(report_doctors, width=22, font=font_style)
report_date_entry.grid(row=1, column=1, padx=10, pady=5)
blood_pressure_label = tk.Label(report_doctors, text="Blood Pressure:", font=font_style)
blood_pressure_label.grid(row=1, column=2, padx=10, pady=5)
blood_pressure_entry = tk.Entry(report_doctors, width=22, font=font_style)
blood_pressure_entry.grid(row=1, column=3, padx=10, pady=5)
pulse_rate_label = tk.Label(report_doctors, text="Pulse Rate:", font=font_style)
pulse_rate_label.grid(row=1, column=4, padx=10, pady=5)
pulse_rate_entry = tk.Entry(report_doctors, width=22, font=font_style)
pulse_rate_entry.grid(row=1, column=5, padx=10, pady=5)
respiratory_rate_label = tk.Label(report_doctors, text="Respiratory Rate:",
font=font_style)
respiratory_rate_label.grid(row=2, column=0, padx=10, pady=5)
respiratory_rate_entry = tk.Entry(report_doctors, width=22, font=font_style)
respiratory_rate_entry.grid(row=2, column=1, padx=10, pady=5)
body_temperature_label = tk.Label(report_doctors, text="Body Temperature:",
font=font_style)
body_temperature_label.grid(row=2, column=2, padx=10, pady=5)
body_temperature_entry = tk.Entry(report_doctors, width=22, font=font_style)
body_temperature_entry.grid(row=2, column=3, padx=10, pady=5)
oxygen_saturation_label = tk.Label(report_doctors, text="Oxygen Saturation:",
font=font_style)
oxygen_saturation_label.grid(row=2, column=4, padx=10, pady=5)
oxygen_saturation_entry = tk.Entry(report_doctors, width=22, font=font_style)
oxygen_saturation_entry.grid(row=2, column=5, padx=10, pady=5)
head_exam_label = tk.Label(report_doctors, text="Head Exam:", font=font_style)
head_exam_label.grid(row=3, column=0, padx=10, pady=5)
head_exam_entry = tk.Text(report_doctors, height=4, width=30)
head_exam_entry.grid(row=3, column=1, padx=10, pady=5)
chest_exam_label = tk.Label(report_doctors, text="Chest Exam:", font=font_style)
chest_exam_label.grid(row=3, column=2, padx=10, pady=5)
chest_exam_entry = tk.Text(report_doctors, height=4, width=30)
chest_exam_entry.grid(row=3, column=3, padx=10, pady=5)
abdominal_exam_label = tk.Label(report_doctors, text="Abdominal Exam:",
font=font_style)
abdominal_exam_label.grid(row=3, column=4, padx=10, pady=5)
abdominal_exam_entry = tk.Text(report_doctors, height=4, width=30)
abdominal_exam_entry.grid(row=3, column=5, padx=10, pady=5)

```

```

    extremities_exam_label = tk.Label(report_doctors, text="Extremities Exam:",
font=font_style)
    extremities_exam_label.grid(row=4, column=0, padx=10, pady=5)
    extremities_exam_entry = tk.Text(report_doctors, height=4, width=30)
    extremities_exam_entry.grid(row=4, column=1, padx=10, pady=5)
    assessment_text_label = tk.Label(report_doctors, text="Assessment:", font=font_style)
    assessment_text_label.grid(row=4, column=2, padx=10, pady=5)
    assessment_text = tk.Text(report_doctors, height=4, width=30)
    assessment_text.grid(row=4, column=3, padx=10, pady=5)
    diagnosis_entry_label = tk.Label(report_doctors, text="Diagnosis:", font=font_style)
    diagnosis_entry_label.grid(row=4, column=4, padx=10, pady=5)
    diagnosis_entry = tk.Text(report_doctors, height=4, width=30)
    diagnosis_entry.grid(row=4, column=5, padx=10, pady=5)
    submit_button = tk.Button(report_doctors, text="Submit Report",
command=submit_report, bg="blue", fg="white", font=font_style)
    submit_button.grid(row=7, column=3, padx=10, pady=10)

def open_patient_app():
    patient_window = tk.Toplevel(root)
    patient_window.title("Patient Application")
    patient_window.geometry("800x600")
    notebook = ttk.Notebook(patient_window)
    notebook.pack(fill="both", expand=True)

def display_patient_details():
    patient_id = current_patient_id
    db_file = 'medical_app.db'
    try:
        conn = sqlite3.connect(db_file)
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM patients WHERE patient_id = ?", (patient_id,))
        patient = cursor.fetchone()
        if patient:
            dob = datetime.strptime(patient[2], '%Y-%m-%d').date()
            age = (datetime.now().date() - dob).days // 365
            patient_details_text.config(state='normal')
            patient_details_text.delete('1.0', 'end')
            patient_details_text.insert('1.0',
                f"Patient ID: {patient[0]}\n"
                f"Name: {patient[1]}\n"
                f"Date of Birth: {dob}\n"
                f"Age: {age} years\n"
                f"Gender: {patient[3]}\n"
                f"Doctor ID: {patient[4]}\n")
            patient_details_text.config(state='disabled')
        else:
            patient_details_text.config(state='normal')
            patient_details_text.delete('1.0', 'end')
            patient_details_text.insert('1.0', "Patient details not found.")
            patient_details_text.config(state='disabled')

```

```

        conn.close()
except sqlite3.Error as e:
    patient_details_text.config(state='normal')
    patient_details_text.delete('1.0', 'end')
    patient_details_text.insert('1.0', "An error occurred while fetching patient details.")
    patient_details_text.config(state='disabled')
    print("SQLite error:", e)

def view_doctors():
    conn = sqlite3.connect('medical_app.db')
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM doctors")
    doctors = cursor.fetchall()
    text_widget = tk.Text(view_doctor)
    text_widget.pack()
    for doctor in doctors:
        text_widget.insert(tk.END, f"Doctor ID: {doctor[0]}\n")
        text_widget.insert(tk.END, f"Name: {doctor[1]}\n")
        text_widget.insert(tk.END, f"Specialization: {doctor[2]}\n")
        text_widget.insert(tk.END, "\n")
    conn.close()

def retrieve_doctor_reports(db_file, current_patient_id):
    try:
        conn = sqlite3.connect(db_file)
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM doctor_reports WHERE patient_id=?",
(current_patient_id,))
        reports = cursor.fetchall()
        conn.close()
        return reports
    except sqlite3.Error as e:
        print("SQLite error:", e)
        return []

def display_reports():
    patient_id = current_patient_id
    db_file = 'medical_app.db'
    doctor_reports = retrieve_doctor_reports(db_file, patient_id)
    if not doctor_reports:
        report_text.set("No reports found for Patient ID: " + str(patient_id))
        return
    report_text.set("Patient Reports for Patient ID " + str(patient_id) + ":\n\n")
    for index, report in enumerate(doctor_reports, start=1):
        report_text.set(report_text.get() +
            f"Report ID: {report[0]}\n"
            f"Patient ID: {report[1]}\n"
            f"Patient Name: {report[2]}\n"
            f"Doctor ID: {report[3]}\n"
            f"Report Date: {report[4]}\n")

```

```

f"Blood Pressure: {report[5]}\n"
f"Pulse Rate: {report[6]}\n"
f"Respiratory Rate: {report[7]}\n"
f"Body Temperature: {report[8]}\n"
f"Oxygen Saturation: {report[9]}\n"
f"Head Exam:\n{report[10]}\n"
f"Chest Exam:\n{report[11]}\n"
f"Abdominal Exam:\n{report[12]}\n"
f"Extremities Exam:\n{report[13]}\n"
f"Assessment:\n{report[14]}\n"
f"Diagnosis:\n{report[15]}\n\n")

patient_details = tk.Frame(notebook, bg="#f5f5f5")
notebook.add(patient_details, text="Your Info")
patient_details_text = tk.Text(patient_details, height=10, width=40)
patient_details_text.pack(pady=20)
patient_details_text.config(state='disabled')
notebook.bind("<<NotebookTabChanged>>", lambda event: display_patient_details())
display_details_button = tk.Button(patient_details, text="Refresh Details",
command=display_patient_details, bg="blue", fg="white", font=font_style)
display_details_button.pack()

view_doctor = tk.Frame(notebook)
notebook.add(view_doctor, text="View Doctor")
view_doctors_button = tk.Button(view_doctor, text="View Doctors",
command=view_doctors, bg="blue", fg="white", font=font_style)
view_doctors_button.pack()

generate_report = ttk.Frame(notebook)
notebook.add(generate_report, text="Generate Report")
report_text = tk.StringVar()
report_label = tk.Label(generate_report, textvariable=report_text, justify=tk.LEFT)
report_label.pack()
retrieve_button = tk.Button(generate_report, text="Retrieve Reports",
command=display_reports, bg="blue", fg="white", font=font_style)
retrieve_button.pack()

root = tk.Tk()
root.title("Login Page")
root.geometry("800x600")

style = ttk.Style()
style.configure("TNotebook", background="#D3D3D3")
style.configure("TNotebook.Tab", font=("Helvetica", 12), padding=[10, 5])
root.rowconfigure(5, minsize=20)
style.configure("TEntry", padding=5, relief="flat", font=("Helvetica", 12))
style.map("TCombobox", fieldbackground=[("readonly", "white")])

font_style = ("Helvetica", 12)

```

```

# Doctor Login Form
doctor_frame = ttk.LabelFrame(root, text="Doctor Login")
doctor_frame.grid(row=0, column=0, padx=20, pady=20)

doctor_username_label = tk.Label(doctor_frame, text="Username:", font=font_style)
doctor_username_label.grid(row=0, column=0, padx=10, pady=5)

doctor_username_entry = tk.Entry(doctor_frame, font=font_style)
doctor_username_entry.grid(row=0, column=1, padx=10, pady=5)

doctor_password_label = tk.Label(doctor_frame, text="Password:", font=font_style)
doctor_password_label.grid(row=1, column=0, padx=10, pady=5)

doctor_password_entry = tk.Entry(doctor_frame, show="*", font=font_style)
doctor_password_entry.grid(row=1, column=1, padx=10, pady=5)

doctor_login_button = tk.Button(doctor_frame, text="Login", command=doctor_login,
bg="#4CAF50", fg="white", font=font_style)
doctor_login_button.grid(row=2, columnspan=2, padx=10, pady=10)

# Patient Login Form
patient_frame = ttk.LabelFrame(root, text="Patient Login")
patient_frame.grid(row=0, column=1, padx=20, pady=20)

patient_username_label = tk.Label(patient_frame, text="Username:", font=font_style)
patient_username_label.grid(row=0, column=0, padx=10, pady=5)

patient_username_entry = tk.Entry(patient_frame, font=font_style)
patient_username_entry.grid(row=0, column=1, padx=10, pady=5)

patient_password_label = tk.Label(patient_frame, text="Password:", font=font_style)
patient_password_label.grid(row=1, column=0, padx=10, pady=5)

patient_password_entry = tk.Entry(patient_frame, show="*", font=font_style)
patient_password_entry.grid(row=1, column=1, padx=10, pady=5)

patient_login_button = tk.Button(patient_frame, text="Login", command=patient_login,
bg="#2196F3", fg="white", font=font_style)
patient_login_button.grid(row=2, columnspan=2, padx=10, pady=10)

root.mainloop()

```


4.3 EXPLANATION

I) **medical_app.py (for creating and maintaining the schema)**

Database Setup for a Doctor-Patient Application

This code snippet focuses on setting up the database for a doctor-patient application. It uses SQLite, a lightweight, embedded relational database management system. Let's break down the key components:

- 1) **Database Connection:** The code starts by importing the `sqlite3` library and establishes a connection to an SQLite database file named 'medical_app.db' using `sqlite3.connect('medical_app.db')`.
- 2) **Creating Tables:** The code creates several tables to store different types of data:
 - a) **patients table:** Stores patient information such as patient ID, name, date of birth, gender, and a reference to the doctor they are associated with.
 - b) **doctors table:** Contains information about doctors, including their ID, name, and specialization.
 - c) **patient_credentials table:** Stores patient login credentials with fields for patient ID, username, and password.
 - d) **doctor_reports table:** Designed for storing medical reports generated by doctors, including patient information, report date, and various medical parameters.
 - e) **doctor_credentials table:** Stores login credentials for doctors, including their ID, username, and password.
 - f) **Data Insertion:** The code uses the `cursor.executemany()` method to insert sample data into the created tables. This includes sample patient, doctor, and login data, which is essential for testing and demonstrating the application's functionalities.
- 3) **Triggers:** Three triggers are created to enforce data integrity:
 - a) **check_age:** Ensures that the 'age' field in the 'patients' table contains valid integer values.
 - b) **check_doctor_id:** Ensures that the 'doctor_id' field in the 'patients' table contains valid integer values.
 - c) **check_gender:** Validates the 'gender' field in the 'patients' table, allowing only 'Male,' 'Female,' or 'Other' as valid values.
 - d) **Foreign Key Constraints:** Foreign key constraints are enabled by executing `PRAGMA foreign_keys = ON;`. These constraints maintain referential integrity between tables by enforcing that foreign key values in the 'patients' and 'doctor_reports' tables correspond to primary key values in the 'doctors' table.
- 4) **Commit and Close:** After setting up the tables and inserting sample data, the code commits the changes to the database and closes the database connection using `conn.commit()` and `conn.close()`.

Database Access Control for a Doctor-Patient Application

This code snippet focuses on access control and user authentication for the doctor-patient application. It extends the functionality of the previous code. Here are the key elements:

- 1) Database Connection: It starts by connecting to the same 'medical_app.db' database.
- 2) Creating the patient_credentials Table: Similar to the first code, this code creates the patient_credentials table to store patient login credentials (username and password).
- 3) Inserting Sample Login Data: Sample patient login data is inserted into the 'patient_credentials' table using the cursor.executemany() method.
- 4) Creating the doctor_credentials Table: This table is designed to store login credentials for doctors (doctor ID, username, and password).
- 5) Inserting Sample Doctor Login Data: Similar to the patient data, sample doctor login data is inserted into the 'doctor_credentials' table.
- 6) Triggers: The code creates triggers for patient and doctor login data to ensure that valid username and password formats are maintained, enhancing security.
- 7) Enabling and Disabling Foreign Key Constraints: The code includes statements to temporarily disable foreign key constraints using PRAGMA foreign_keys = OFF; and then re-enable them using PRAGMA foreign_keys = ON;. This is done during trigger creation to prevent foreign key constraint violations while the triggers are created.
- 8) Commit and Close: As in the first code, the changes are committed to the database, and the connection is closed.

II) Login.py (for creating the GUI and applying the schema for the doctor-patient applications)

This Python code is for a simple application that uses the Tkinter library to create a graphical user interface (GUI) for a doctor-patient management system. The application allows doctors and patients to log in and access relevant information and features. Let's break down the code into its components:

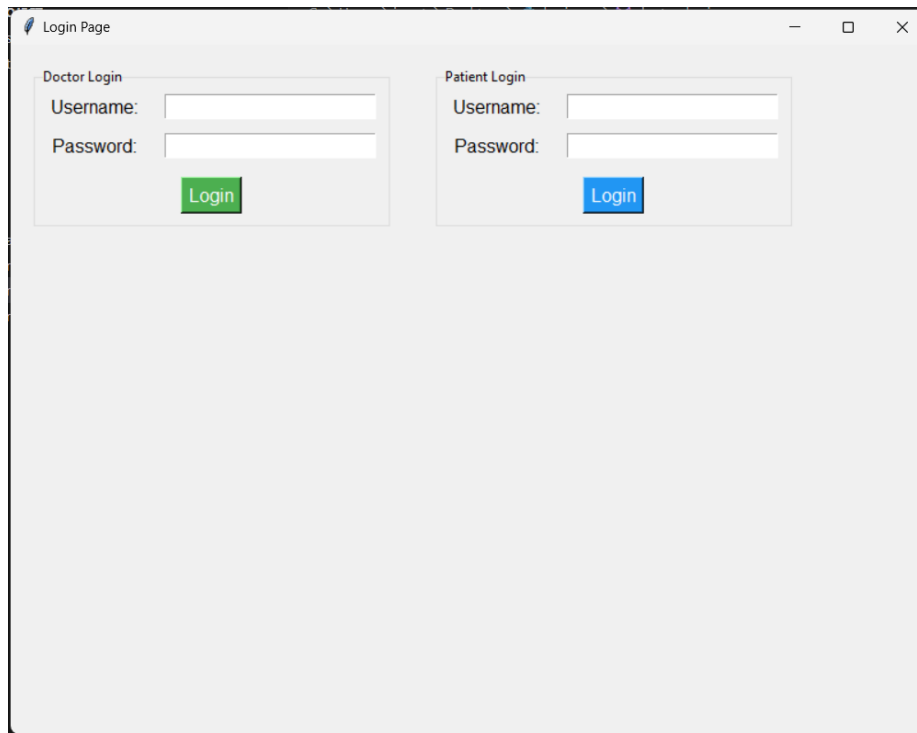
- 1) Importing Libraries: The code begins by importing the necessary libraries, including tkinter for creating the GUI, sqlite3 for interacting with a SQLite database, datetime for handling date and time information, and ttk for additional widgets.
- 2) Global Variables: Two global variables, current_doctor_id and current_patient_id, are initialized as None. These variables will be used to keep track of the currently logged-in doctor and patient.
- 3) Clear Entry Fields Function: clear_entry_fields is a function that clears the input fields for doctor and patient usernames and passwords.

- 4) Doctor Login Function: The `doctor_login` function is called when the doctor presses the "Login" button. It retrieves the doctor's input username and password, validates them against the database, and logs in if the credentials are correct.
- 5) Patient Login Function: The `patient_login` function is similar to the doctor login function, but it validates the credentials for patients.
- 6) Open Doctor Application and Patient Application Functions: These functions are called when a doctor or patient successfully logs in. They open a new window for the respective user with different tabs and functionality.
- 7) Tabs for Doctor Application: In the doctor's application window, multiple tabs are created using the `tk.Notebook` widget. These tabs include:
 - a) Add Patient: Allows the doctor to add patient details.
 - b) Search Patients: Lets the doctor search for patients.
 - c) Update Patient: Permits updating patient information.
 - d) View Doctors: Displays a list of doctors.
 - e) Add Patient Report: Allows doctors to submit medical reports for patients.
- 8) Tabs for Patient Application: In the patient's application window, there are also multiple tabs, such as:
 - a) Your Info: Displays patient details.
 - b) View Doctor: Allows patients to view a list of doctors.
 - c) Generate Report: Lets patients retrieve and display medical reports.
 - d) Main Application Initialization: The main application window is created using `tkinter`. It includes login forms for both doctors and patients, and the `tk.Style` is used to customize the appearance of the tabs.

The code also includes various user interface elements, such as labels, entry fields, buttons, and text widgets, to provide input and display information to the users.

5. INPUTS AND OUTPUTS

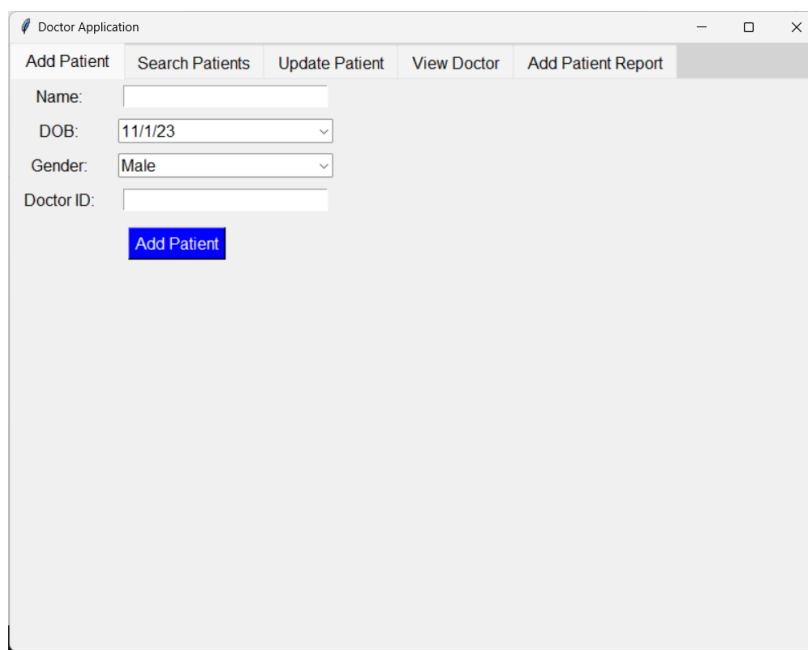
Login Page:



The screenshot shows a web application window titled "Login Page". It contains two login forms side-by-side. The "Doctor Login" form on the left has a green "Login" button. The "Patient Login" form on the right has a blue "Login" button. Both forms have "Username:" and "Password:" labels followed by input fields.

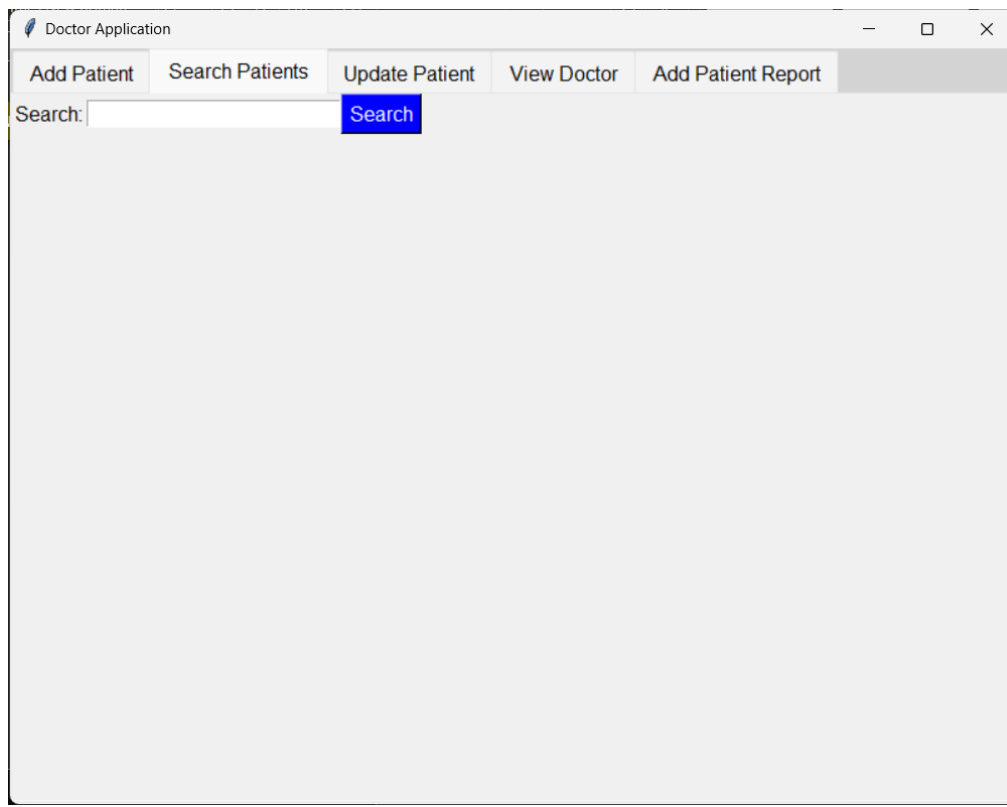
Doctor Application:

1. Add Patient:

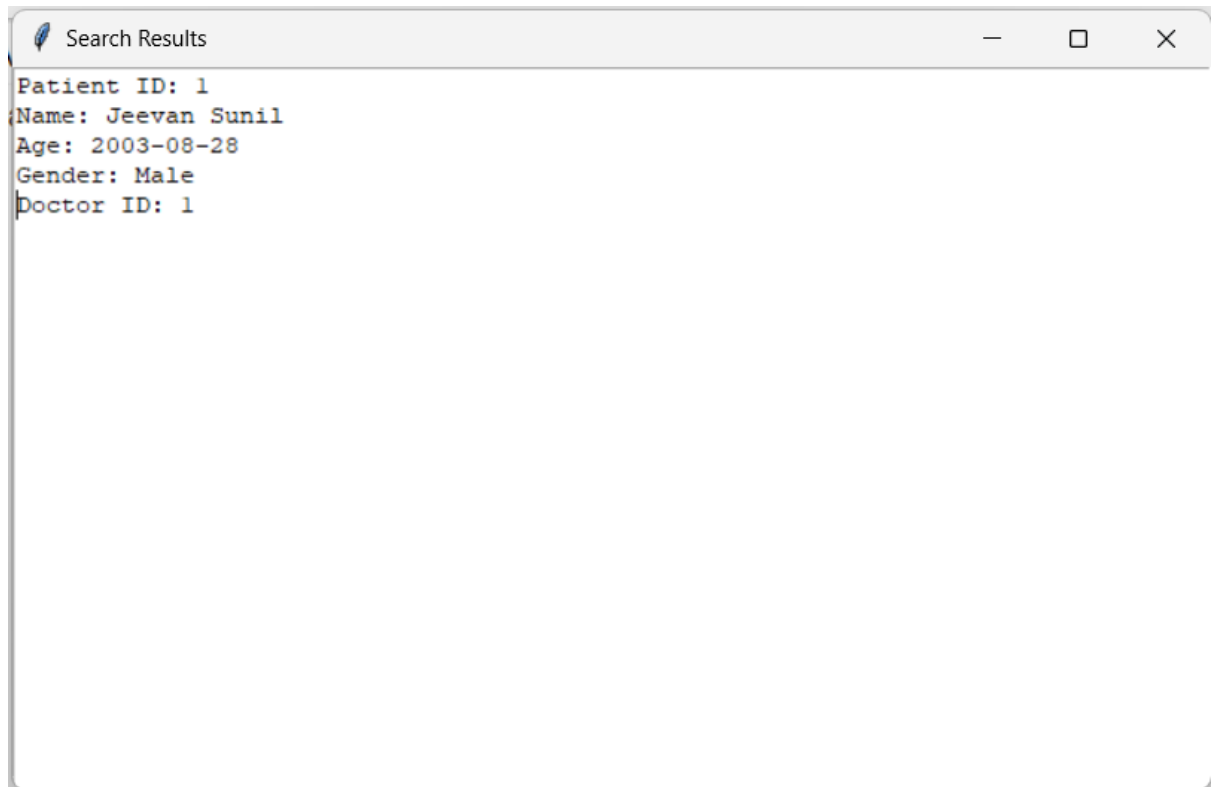


The screenshot shows a web application window titled "Doctor Application". It has a tabbed interface with five tabs: "Add Patient", "Search Patients", "Update Patient", "View Doctor", and "Add Patient Report". The "Add Patient" tab is currently selected. The form contains the following fields: "Name:" (text input), "DOB:" (date picker showing "11/1/23"), "Gender:" (dropdown menu showing "Male"), and "Doctor ID:" (text input). A blue "Add Patient" button is located below the "Doctor ID" field.

2. Search Patients:



The screenshot shows a window titled "Doctor Application" with a menu bar containing "Add Patient", "Search Patients", "Update Patient", "View Doctor", and "Add Patient Report". The "Search Patients" tab is selected. Below the menu bar, there is a "Search:" label followed by a text input field and a blue "Search" button. The main area of the window is empty.



The screenshot shows a window titled "Search Results" displaying the following patient information:

```
Patient ID: 1
Name: Jeevan Sunil
Age: 2003-08-28
Gender: Male
Doctor ID: 1
```

3. Update Patient:

The screenshot shows a web application window titled "Doctor Application". It has five tabs: "Add Patient", "Search Patients", "Update Patient", "View Doctor", and "Add Patient Report". The "Update Patient" tab is active. The form contains the following fields:

- Patient ID:
- Name:
- DOB:
- Gender:
- Doctor ID:

Below the fields is a blue button labeled "Update Patient".

4. View Doctor:

The screenshot shows the same "Doctor Application" window, but with the "View Doctor" tab active. A blue button labeled "View Doctors" is highlighted. Below the button, a text area displays the following information:

```
Doctor ID: 1  
Name: Dr. John Doe  
Specialization: Cardiologist  
  
Doctor ID: 2  
Name: Dr. Jane Smith  
Specialization: Pediatrician  
  
Doctor ID: 3  
Name: Dr. Robert Johnson  
Specialization: Dermatologist  
  
Doctor ID: 4  
Name: Dr. Rita Suresh  
Specialization: General Practitioner
```

5. Add Patient Report:

The screenshot shows a web application window titled "Doctor Application". It has a tabbed interface with the following tabs: "Add Patient", "Search Patients", "Update Patient", "View Doctor", and "Add Patient Report". The "Add Patient Report" tab is currently selected. The form contains the following fields:

Patient ID:	<input type="text"/>	Doctor ID:	<input type="text"/>	Patient Name:	<input type="text"/>
Report Date:	<input type="text"/>	Blood Pressure:	<input type="text"/>	Pulse Rate:	<input type="text"/>
Respiratory Rate:	<input type="text"/>	Body Temperature:	<input type="text"/>	Oxygen Saturation:	<input type="text"/>
Head Exam:	<input type="text"/>	Chest Exam:	<input type="text"/>	Abdominal Exam:	<input type="text"/>
Extremities Exam:	<input type="text"/>	Assessment:	<input type="text"/>	Diagnosis:	<input type="text"/>

Below the form fields is a blue button labeled "Submit Report".

Patient Application:

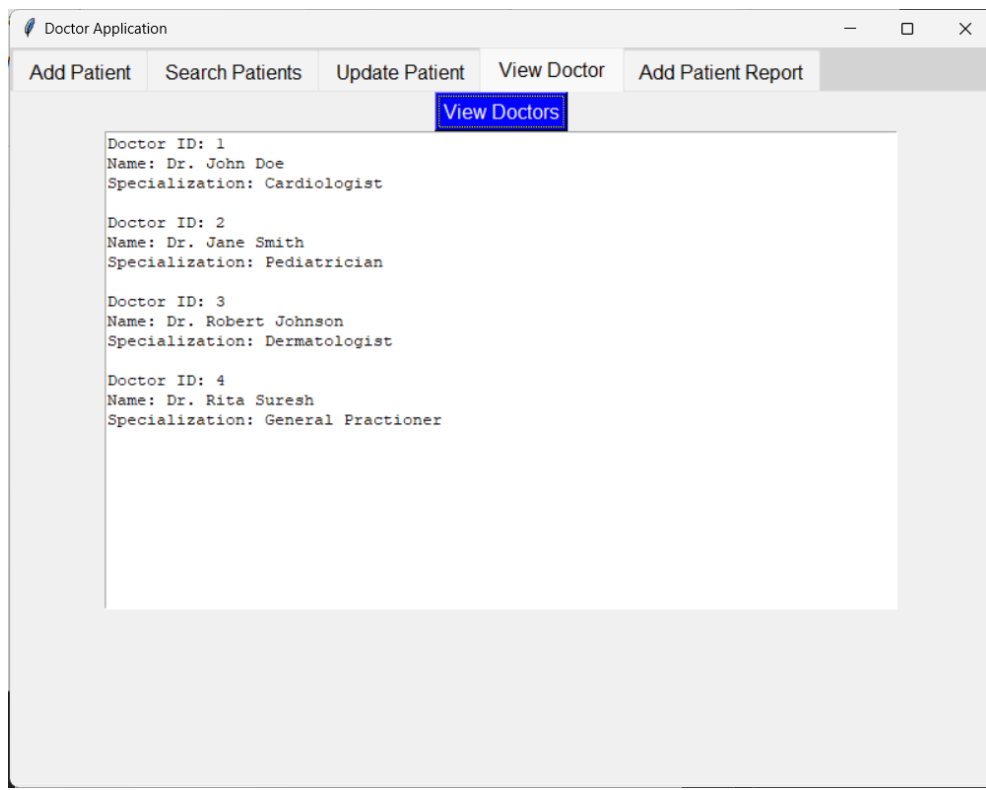
1. Your Info:

The screenshot shows a web application window titled "Patient Application". It has a tabbed interface with the following tabs: "Your Info", "View Doctor", and "Generate Report". The "Your Info" tab is currently selected. The form displays the following patient information:

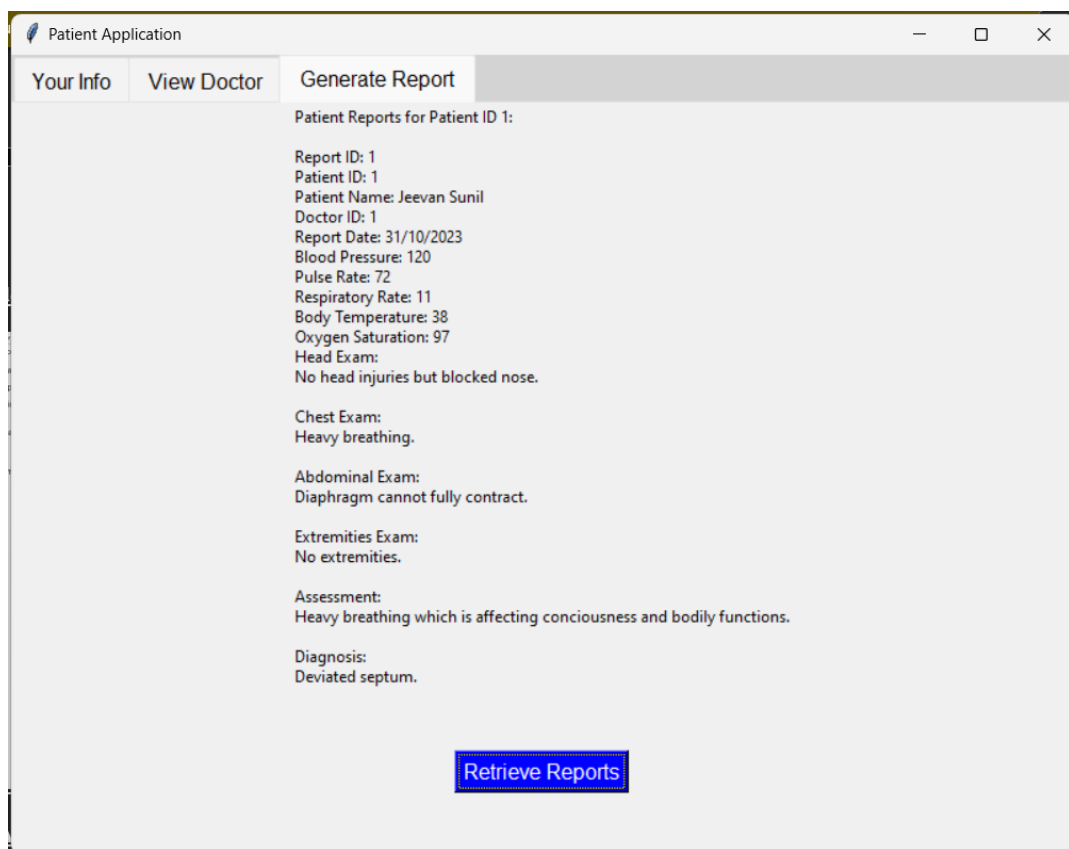
```
Patient ID: 1  
Name: Jeevan Sunil  
Date of Birth: 2003-08-28  
Age: 20 years  
Gender: Male  
Doctor ID: 1
```

Below the form is a blue button labeled "Refresh Details".

2. View Doctors:



3. Generate Report:



6. CONCLUSION

In this report, we presented the development of a doctor-patient management system using Python and the Tkinter library for creating a graphical user interface. The system is designed to facilitate the interactions between doctors and patients, enabling doctors to manage patient information and submit medical reports, and allowing patients to access their personal details and medical reports. The system is backed by a SQLite database, which stores the necessary data for user authentication and information retrieval.

The key features and functionalities of the system can be summarized as follows:

Doctor and Patient Login: The system supports secure login for both doctors and patients, ensuring that only authorized users can access their respective functionalities.

Doctor Features:

- 1) **Add Patient:** Doctors can add new patients to the system, providing details such as name, date of birth, gender, and their own doctor ID.
- 2) **Search Patients:** A search feature allows doctors to find patients based on name or patient ID.
- 3) **Update Patient Information:** Doctors can update patient details.
- 4) **View Doctors:** A list of doctors is available for reference.

Patient Features:

- 1) **Personal Details:** Patients can view their personal information, including name, date of birth, age, gender, and their assigned doctor.
- 2) **View Doctors:** Like doctors, patients can also access a list of doctors.
- 3) **Generate Medical Reports:** Patients can retrieve and view their medical reports, providing them with insights into their healthcare records.

This doctor-patient management system provides a foundation for efficient and organized healthcare data management. It ensures that doctors have the necessary tools to manage their patients' information and medical reports, while patients can conveniently access and review their own medical records. The use of a user-friendly graphical interface enhances the user experience and makes the system accessible to a wider audience.

The system's use of a SQLite database allows for data storage, retrieval, and management. However, it should be noted that for real-world use, additional security measures and data validation checks should be implemented to ensure the privacy and integrity of patient data.

As with any software system, there is always room for improvement and future development. In future iterations of this system, additional features such as appointment scheduling, prescription management, and integration with external systems could be considered to further enhance its functionality.

In conclusion, the doctor-patient management system presented in this report serves as a valuable tool for medical practitioners and patients, streamlining the process of managing

patient information and healthcare records. It is a testament to the power of Python and the Tkinter library in creating user-friendly and efficient graphical user interfaces for healthcare applications.