Mini Project: Doctor-Patient Application

20CS2016 – Database Management Systems

By: Jonathan Thomas Mathews URK21CS2036

Database Management Systems

- A Database Management System (DBMS) is a software system that enables the creation, maintenance, and management of databases. It provides a structured way to store, retrieve, and manipulate data efficiently. DBMSs serve as intermediaries between users and the database, allowing users to interact with data without needing to understand the underlying data structures. Key features of DBMSs include data storage, retrieval, indexing, security, and support for querying and transactions.

- DBMSs can be broadly categorized into two types:
- Relational DBMS (RDBMS): These systems organize data into tables with rows and columns, providing a structured way to store and query data. Popular RDBMSs include MySQL, PostgreSQL, Oracle, and Microsoft SQL Server.
- NoSQL Databases: NoSQL databases are more flexible and suitable for handling unstructured or semi-structured data. They are commonly used in modern web applications for their ability to scale and adapt to changing data requirements. Examples include MongoDB, Cassandra, and Redis.

Motivation

The motivation behind developing the Doctor-Patient Database project is to streamline and enhance the management of medical records and improve patient care. Here are some key motivations:

Efficient Data Management: Doctors and healthcare providers can quickly access patient information, reducing administrative overhead.

Data Security: Storing medical records in a centralized and secure database helps ensure the confidentiality and integrity of patient data.

Reporting and Analysis: The system allows doctors to generate detailed medical reports for patients. These reports provide valuable insights into a patient's health and can be crucial for diagnosis and treatment planning.

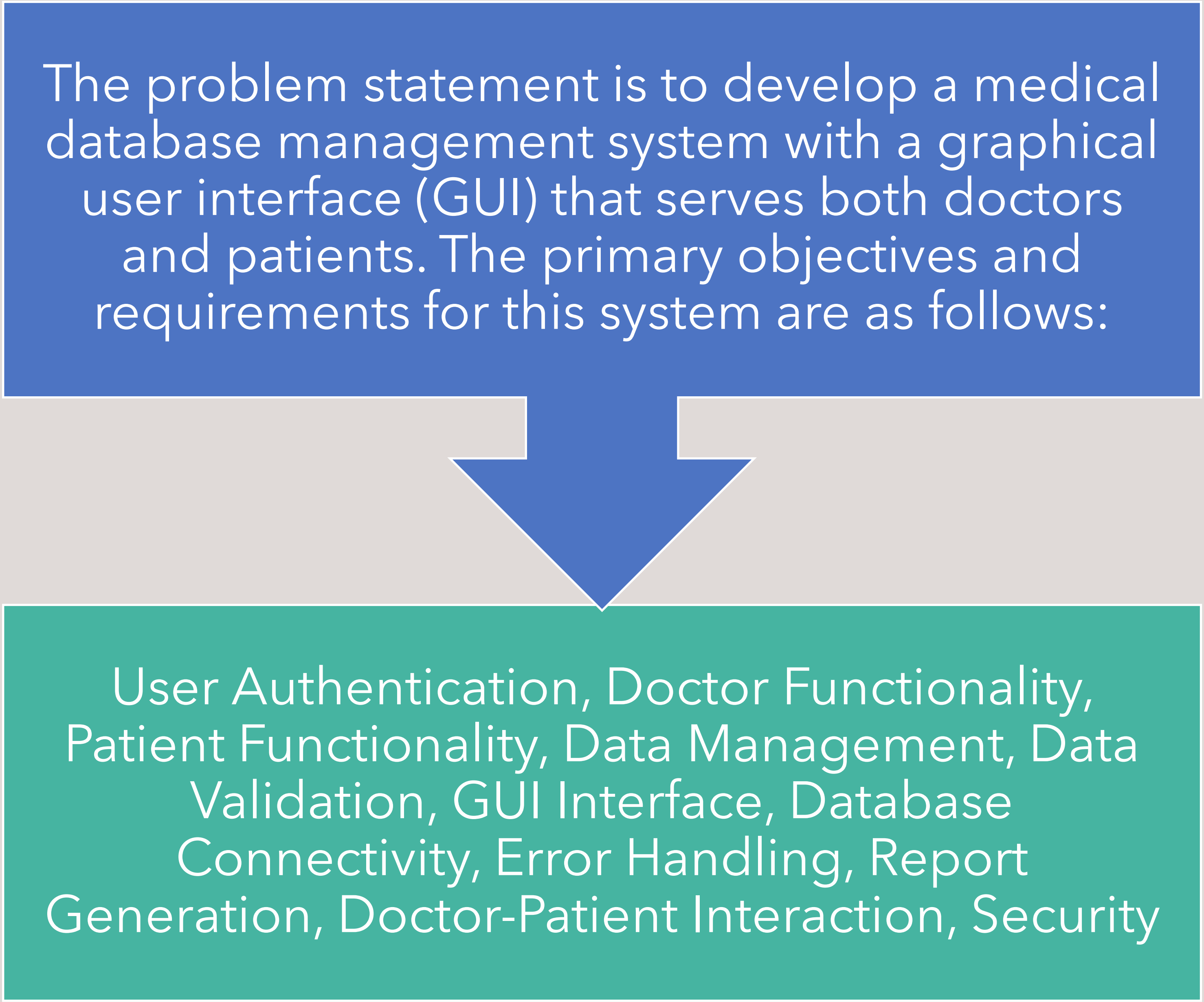
Patient Convenience: Patients can access their own records, reducing the need for physical paperwork. This convenience can improve patient engagement and satisfaction.

Data Accuracy: A well-maintained medical database can help reduce errors in patient records. Accurate and up-to-date information is critical for delivering quality healthcare.

Scalability: This project can be a foundation for expanding the database to handle a larger number of patients and healthcare providers.

Research and Analysis: The data collected in the database can be a valuable resource for medical research, clinical trials, and epidemiological studies.

Problem Statement



The problem statement is to develop a medical database management system with a graphical user interface (GUI) that serves both doctors and patients. The primary objectives and requirements for this system are as follows:

User Authentication, Doctor Functionality, Patient Functionality, Data Management, Data Validation, GUI Interface, Database Connectivity, Error Handling, Report Generation, Doctor-Patient Interaction, Security

Front-end of the Project

- Tkinter: Tkinter is a Python library for building graphical user interfaces. It provides a range of widgets (such as buttons, labels, text entry fields, and frames) to create the application's interface. In this project, Tkinter is used to create the login screens, tabbed interfaces, input fields, buttons, and display areas for patient and doctor interactions.
- Ttk (Themed Tkinter): Ttk is a module within Tkinter that provides themed widget sets. It is used to enhance the look and feel of the application by providing a more modern and attractive appearance to the GUI elements.
- Notebook: The ttk.Notebook widget is used to create a tabbed interface, allowing the user to switch between different sections of the application (such as "Add Patient," "Search Patients," "Generate Report," etc.) with ease.
- Buttons and Entry Fields: Various buttons are used to trigger actions like logging in and submitting data. Entry fields are provided for users to input data such as usernames, passwords, patient details, and medical reports.
- Text and Text Widgets: Text widgets are used to display detailed information, such as patient details and medical reports.

Back-end of the Project

SQLite Database: The backend uses an SQLite database named "medical_app.db" to store and manage data related to doctors, patients, and medical reports. SQLite is a lightweight, serverless, and self-contained database engine, making it suitable for small to medium-scale applications.

Database Operations: The code includes functions for database operations, such as inserting patient records, searching for patients, updating patient information, and submitting medical reports. These functions interact with the database to perform CRUD (Create, Read, Update, Delete) operations.

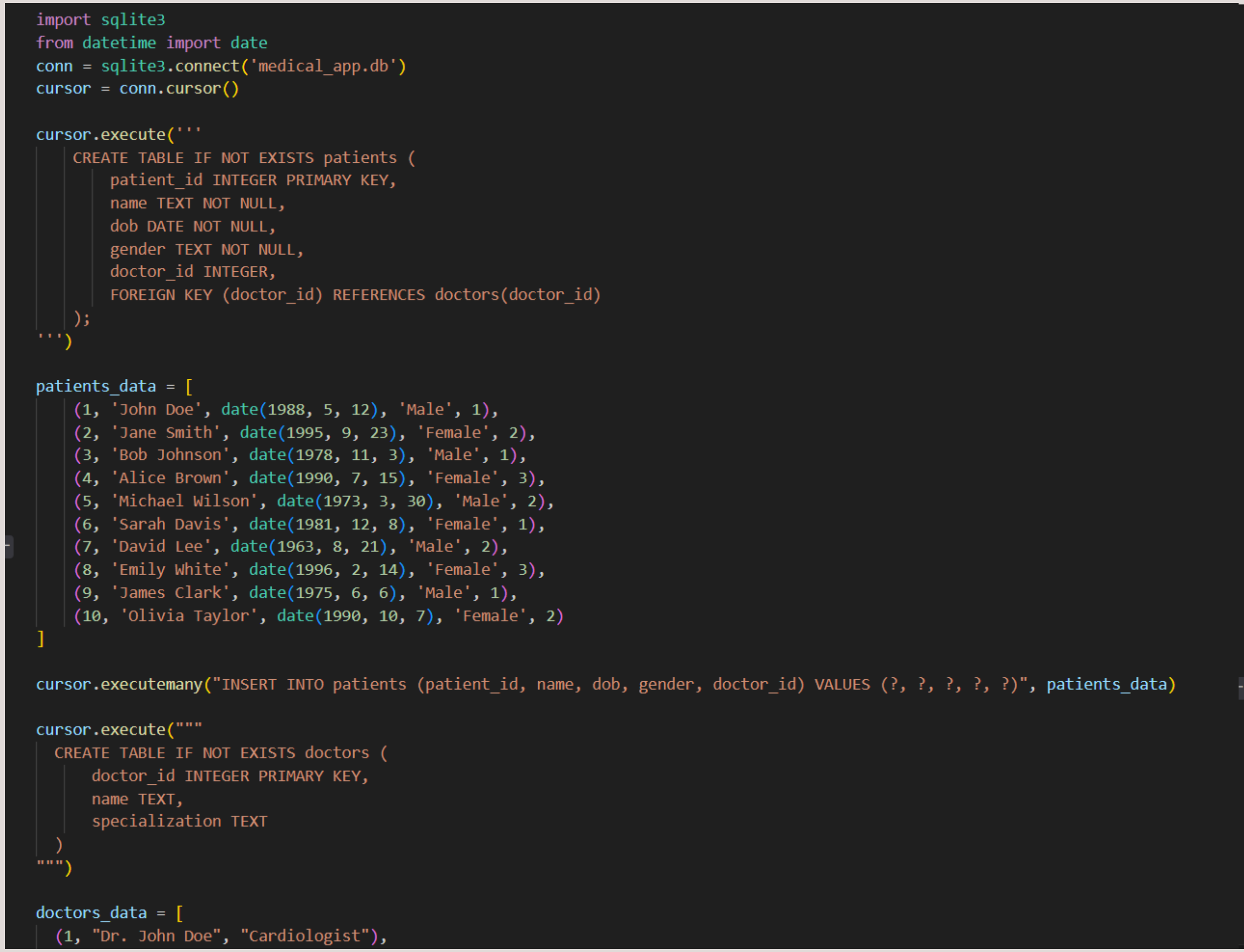
Data Validation: The backend includes data validation to ensure that data entered by users is accurate and appropriate. For example, it checks for valid doctor or patient IDs before updating records.

Exception Handling: The backend includes exception handling to manage errors, such as database connection issues, and provide appropriate error messages to the user.

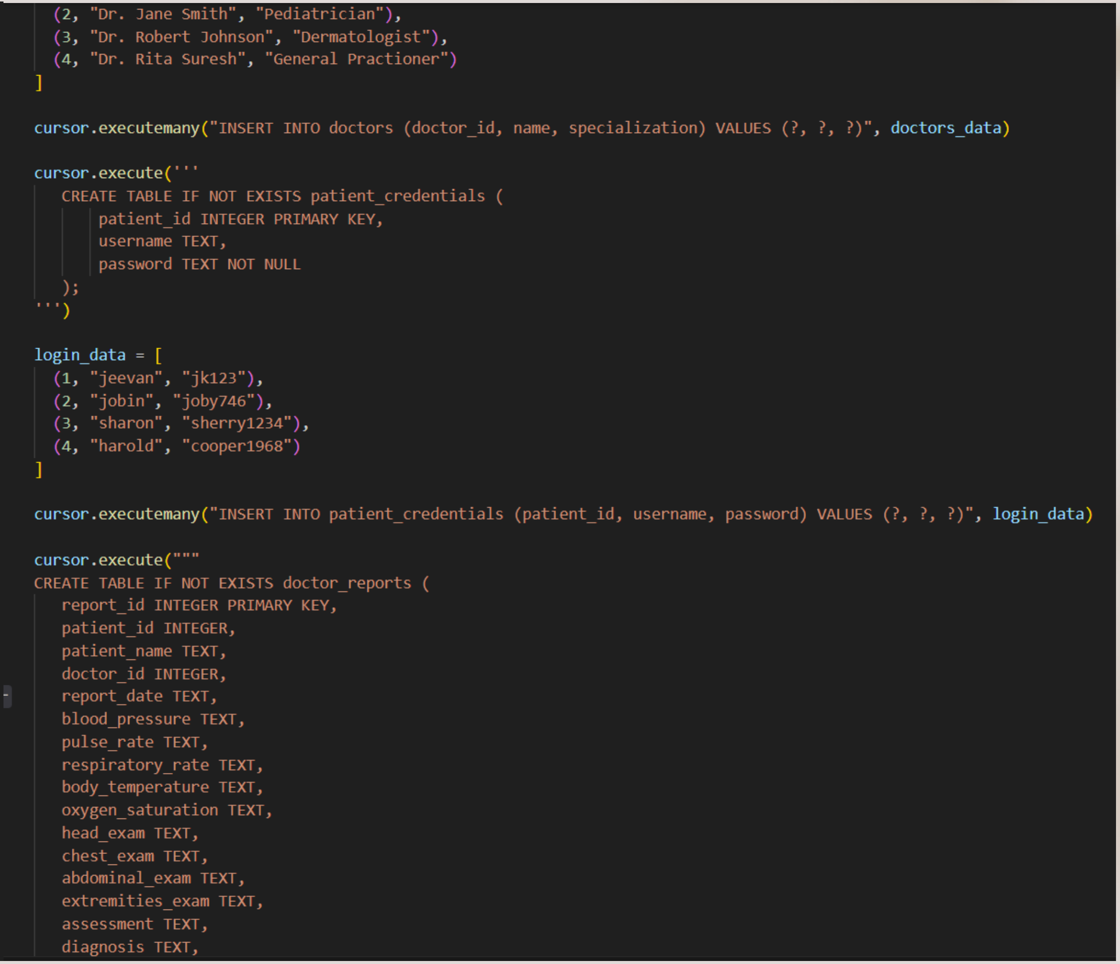
Database Retrieval: The backend retrieves and processes data from the database to display patient details and generate patient reports.

User Authentication: The backend is responsible for user authentication by verifying the entered username and password against the stored credentials in the database.

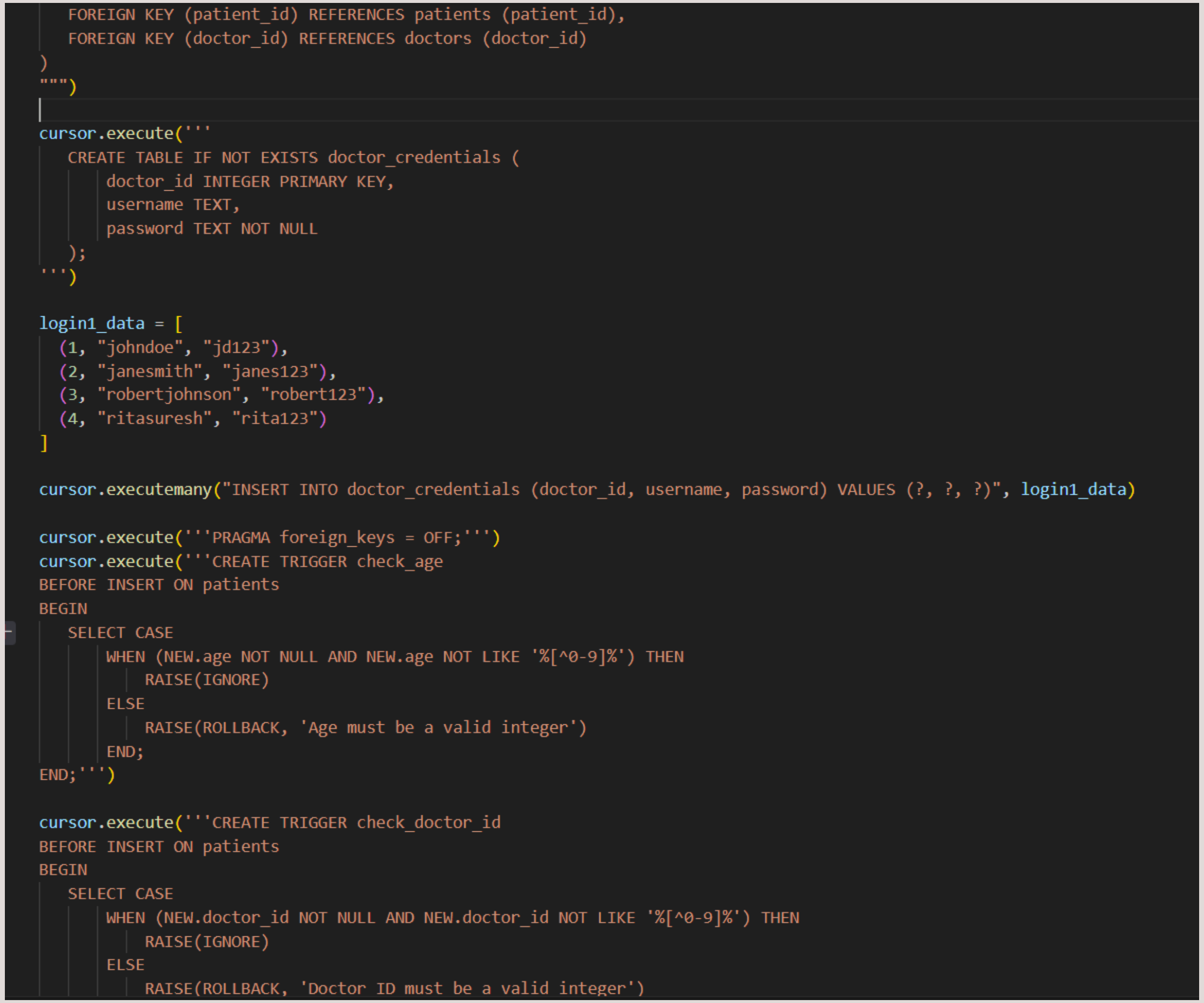
Source Code



```
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"),
```

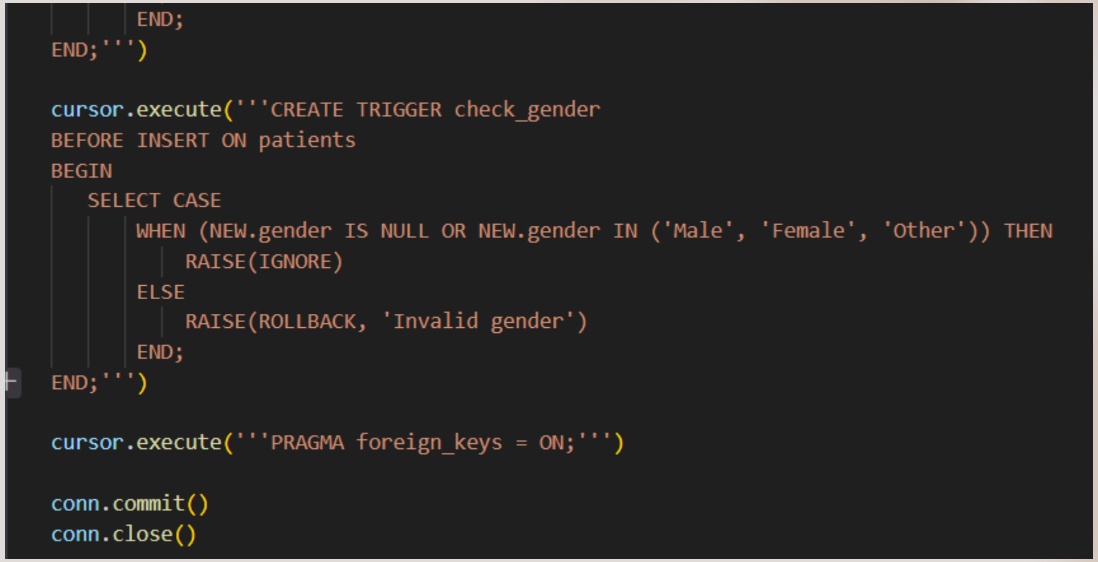


```
  (2, "Dr. Jane Smith", "Pediatrician"),
  (3, "Dr. Robert Johnson", "Dermatologist"),
  (4, "Dr. Rita Suresh", "General Practioner")
]

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,
```



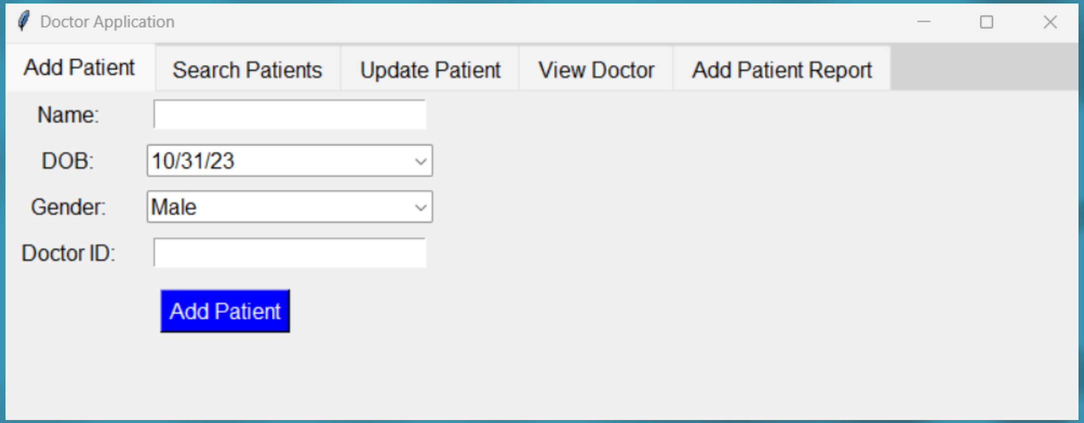
```
   FOREIGN KEY (patient_id) REFERENCES patients (patient_id),
   FOREIGN KEY (doctor_id) REFERENCES doctors (doctor_id)
)
""")

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')
```

This creates the database schema



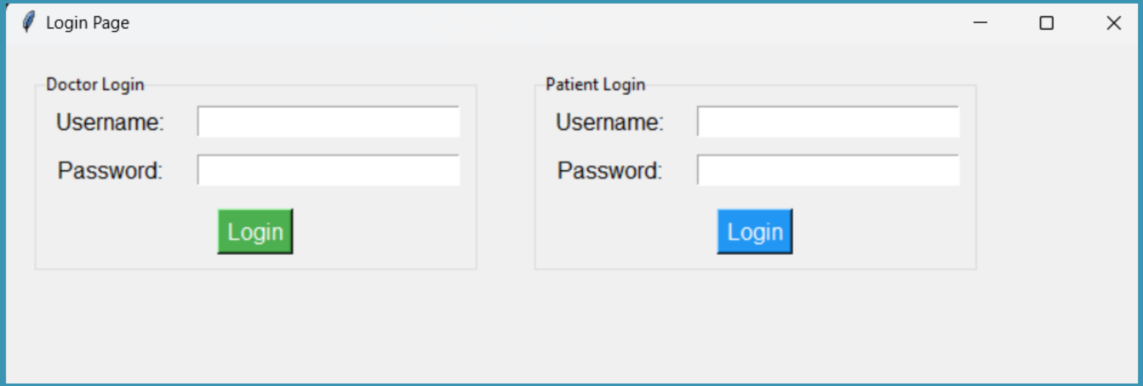
```
       END;
END;''')

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()
```

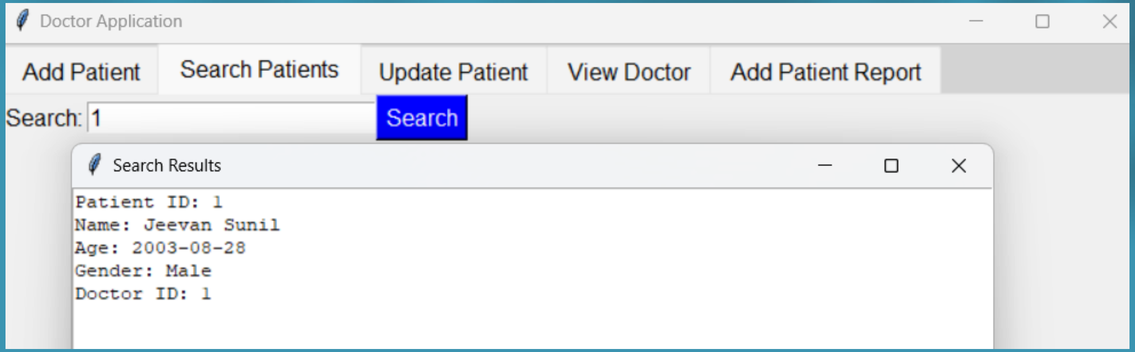
Output



DOB: 10/31/23

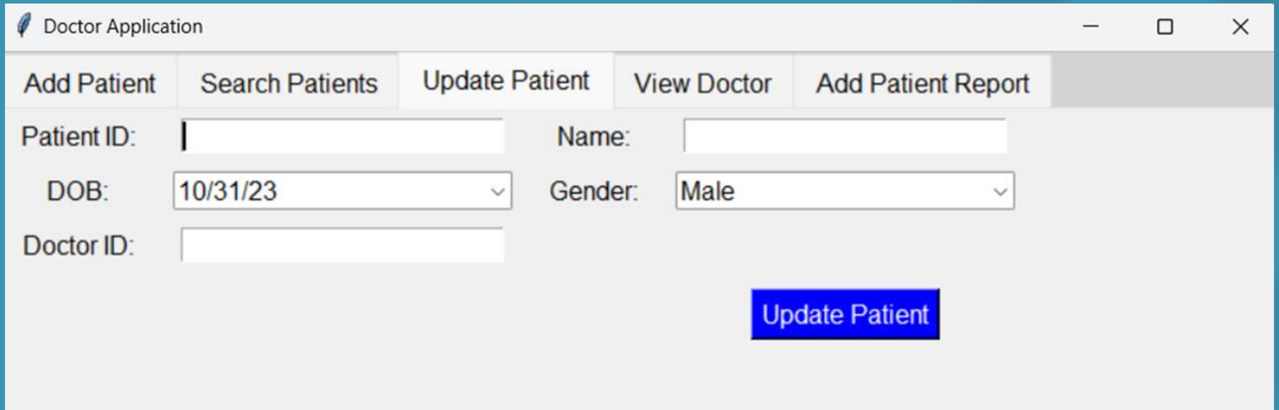
Gender: Male





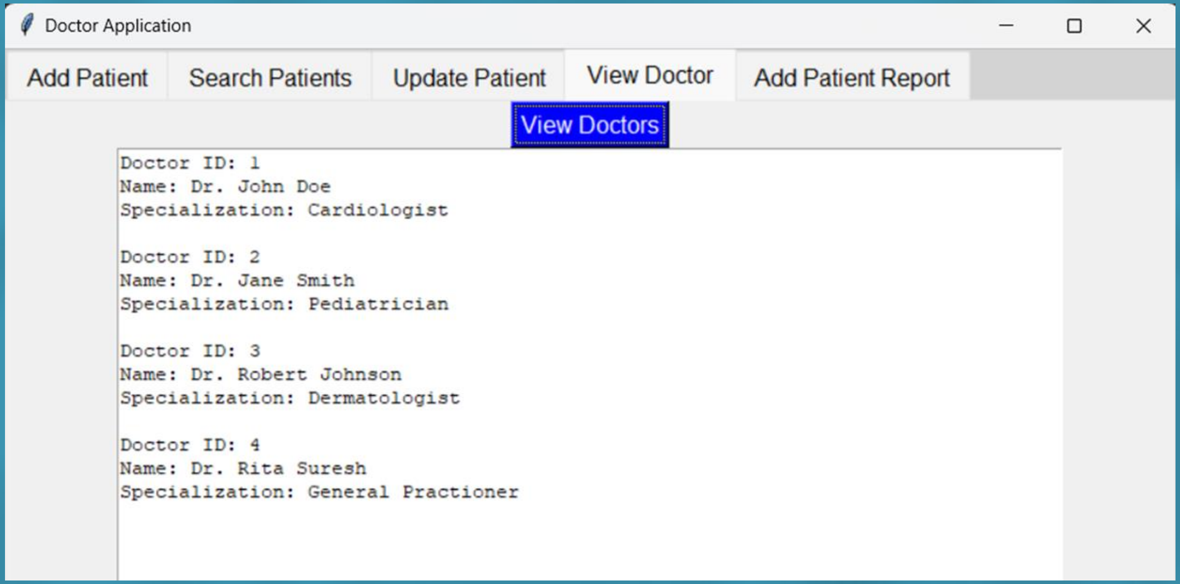
Search: 1

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

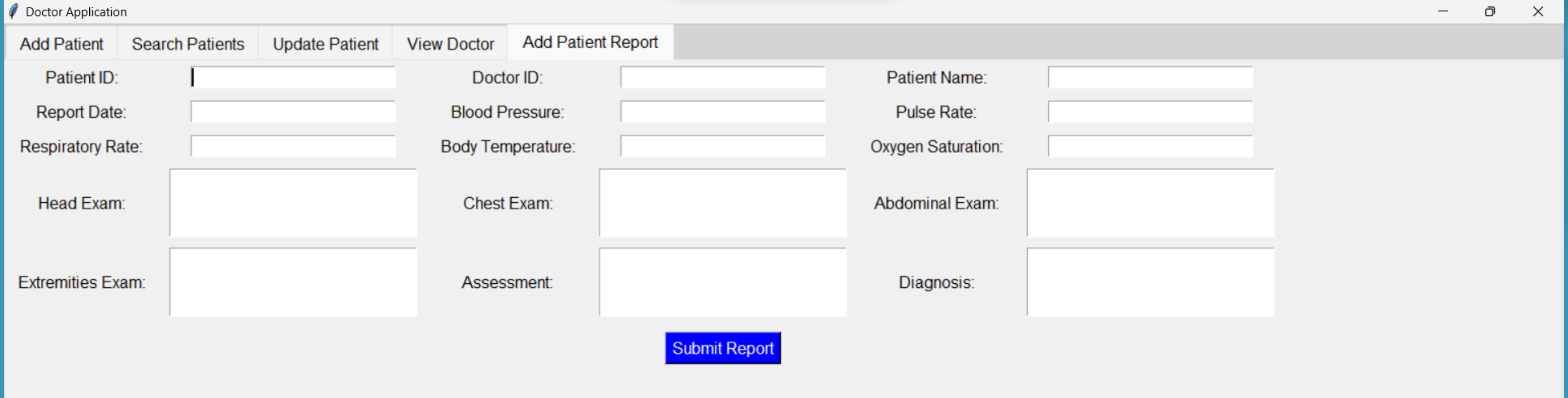


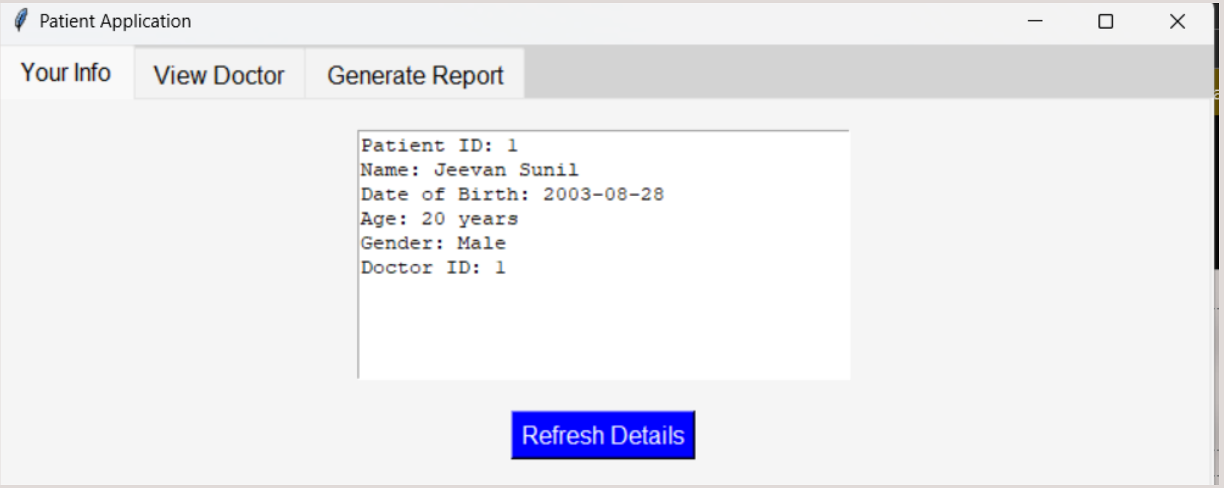
DOB: 10/31/23

Gender: Male

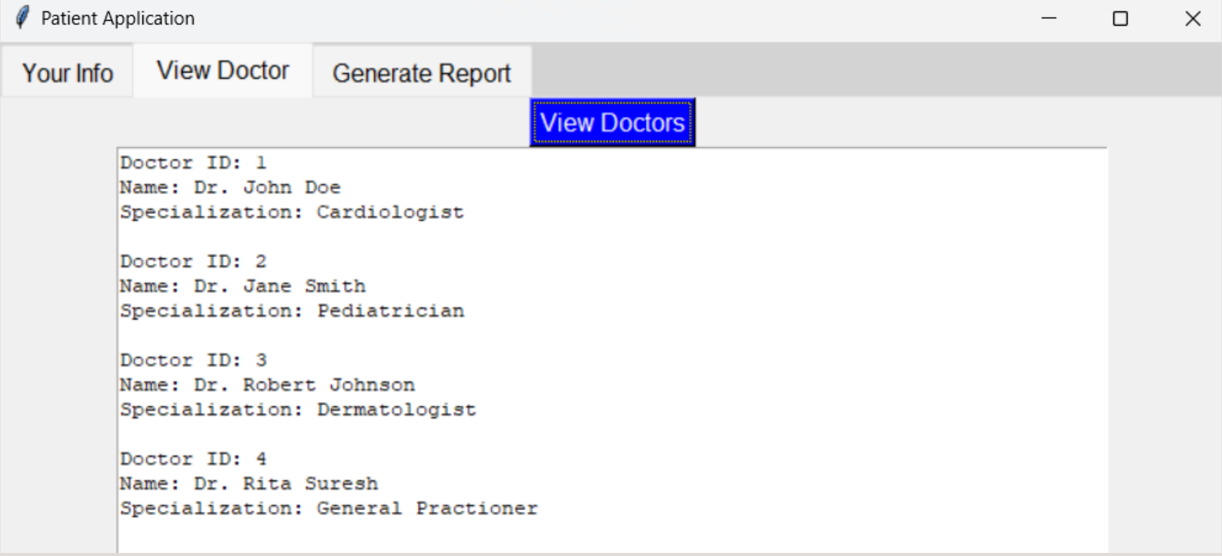


```
Doctor ID: 1
Name: Dr. John Doe
Specialization: Cardiologist

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 Practioner
```

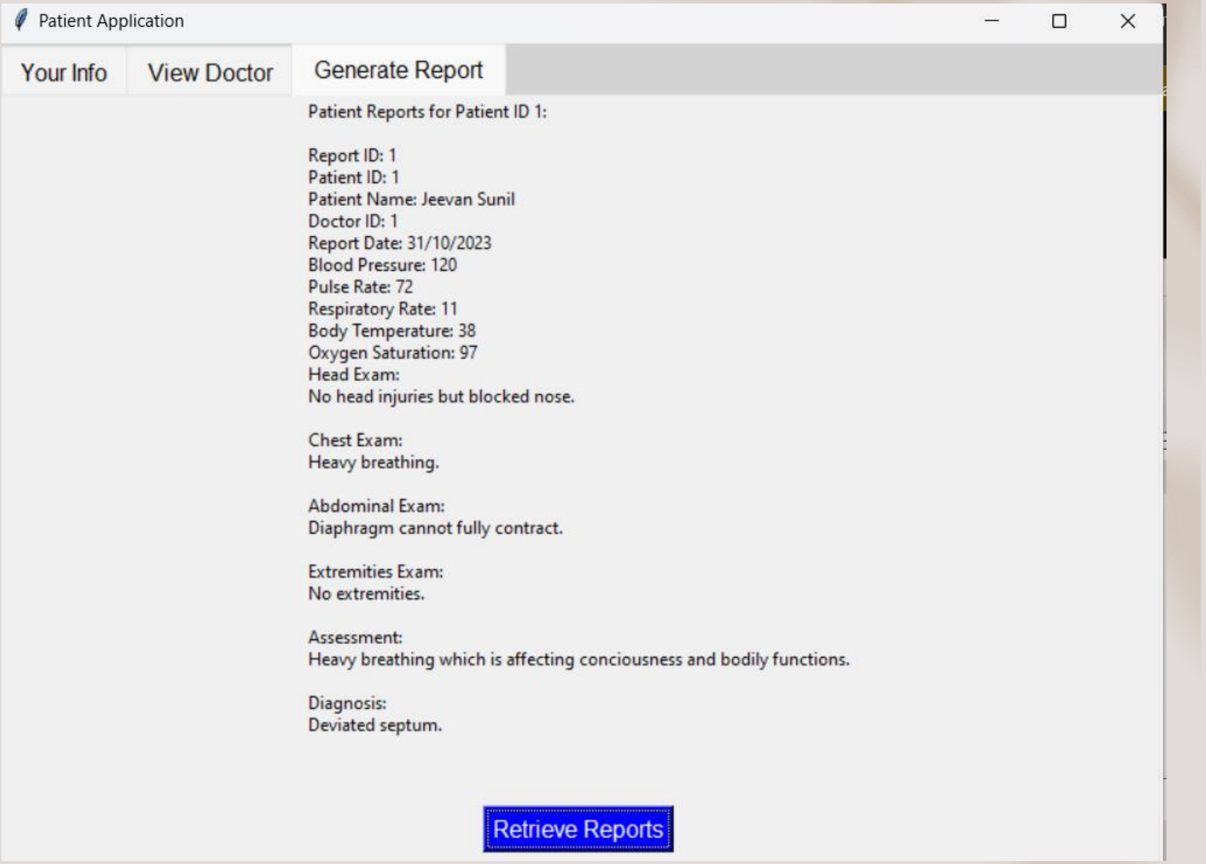




```
Patient ID: 1
Name: Jeevan Sunil
Date of Birth: 2003-08-28
Age: 20 years
Gender: Male
Doctor ID: 1
```



```
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 Practioner
```



```
Patient Reports for Patient ID 1:

Report ID: 1
Patient ID: 1
Patient Name: Jeevan Sunil
Doctor ID: 1
Report Date: 31/10/2023
Blood Pressure: 120
Pulse Rate: 72
Respiratory Rate: 11
Body Temperature: 38
Oxygen Saturation: 97
Head Exam:
No head injuries but blocked nose.

Chest Exam:
Heavy breathing.

Abdominal Exam:
Diaphragm cannot fully contract.

Extremities Exam:
No extremities.

Assessment:
Heavy breathing which is affecting conciousness and bodily functions.

Diagnosis:
Deviated septum.
```

- THANK YOU!