# E-HEALTHCARE

# MANAGEMENT SYSTEM (PYTHON)

MINOR PROJECT REPORT

By

**NITHIN VIKNESH (RA221030010181)**

**TADEPALLY DATTA MANI ANURAG (RA2211030010178)**

Under the guidance of   
**Dr. S. Thenmalar**  
*In partial fulfilment for the Course*

of

**21CSC203P – ADVANCED PROGRAMMING PRACTICE**

in **NETWORKING AND COMMUNICTION**



**FACULTY OF ENGINEERING AND TECHNOLOGY**

**SCHOOL OF COMPUTING**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR**

**NOVEMBER 2023**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**(Under Section 3 of UGC Act, 1956)**

**BONAFIDE CERTIFICATE**

Certified that this minor project report for the course **21CSC203P** - **ADVANCED PROGRAMMING PRACTICE** entitled in "**E-HEALTHCARE** **MANAGEMENT SYSTEM**" is the bonafide work of **NITHIN VIKNESH (RA221030010181)** and **TADEPALLY DATTA MANI ANURAG (RA2211030010178)** who carried out the work under my supervision.

**FACULTY IN-CHARGE**

Dr. S. Thenmalar

Associate Professor

Department of Networking and

Communications

SRM Institute of Science and Technology

Kattankulathur

# 

**ABSTRACT**

The Hospital Management System is a user-friendly tool that helps healthcare places run better. It connects well with a strong database, managing patient info, appointments, and payments smoothly. The design is easy, making it simple for doctors and staff to handle patient records. It's versatile and handles different details, making patient record management complete.

Apart from patient records, it's great at scheduling appointments. The billing feature is strong, ensuring clear financial transactions and helping with efficient financial management and reporting.

The 'View Patients' tool is user-friendly, giving quick access to all patient records. The design is easy to understand, reducing the learning curve for users, even for staff not familiar with technology.

This project shows how combining a user-friendly design and a strong database can improve hospital record management. Overall, the Hospital Management System is an efficient solution for healthcare places, making patient care better, administrative processes smoother, and healthcare operations more efficient. It proves how using technology positively changes healthcare management.

# ACKNOWLEDGEMENT

We express our heartfelt thanks to our honorable **Vice Chancellor Dr. C. MUTHAMIZHCHELVAN**, for being the beacon in all our endeavors.

We would like to express my warmth of gratitude to our **Registrar Dr. S. Ponnusamy,** for his encouragement.

We express our profound gratitude to our **Dean (College of Engineering and Technology) Dr. T. V. Gopal,** for bringing out novelty in all executions.

We would like to express my heartfelt thanks to Chairperson, School of Computing **Dr. Revathi Venkataraman,** for imparting confidence to complete my course project

We wish to express my sincere thanks to **Course Audit Professors Dr. Vadivu. G, Professor, Department of Data Science and Business Systems and Dr. Sasikala. E Professor, Department of Data Science and Business Systems** and **Course Coordinators** for their constant encouragement and support.

We are highly thankful to our Course project Faculty **Dr. S. Thenmalar, Associate Professor, Department of Networking and Communications,** for herassistance, timely suggestion and guidance throughout the duration of this course project.

We extend my gratitude to our **HoD Dr. Annapurani. K Professor and Head, Department of Networking and Communications** and my Departmental colleagues for their Support.

Finally, we thank our parents and friends near and dear ones who directly and indirectly contributed to the successful completion of our project. Above all, I thank the almighty for showering his blessings on me to complete my Course project.

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **CHAPTER NO** | | **CONTENTS** | **PAGE NO** |
| **1** | **INTRODUCTION** | |  |
|  | 1.1 Motivation | |  |
|  | 1.2 Objective | |  |
|  | 1.3 Problem Statement | |  |
|  | 1.4 Challenges | |  |
| **2** | **LITERATURE SURVEY** | |  |
| **3** | **REQUIREMENT ANALYSIS** | |  |
| **4** | **ARCHITECTURE & DESIGN** | |  |
| **5** | **IMPLEMENTATION** | |  |
| **6** | **EXPERIMENT RESULTS & ANALYSIS** | |  |
| **7** | **CONCLUSION** | |  |
| **8** | **REFERENCES** | |  |

1. **INTRODUCTION**
   1. **Motivation:**

The motivation behind developing a Hospital Management System (HMS) using Python lies in the need for an efficient and streamlined approach to healthcare administration. A well-designed HMS can significantly enhance the overall quality of patient care, optimize resource utilization, and improve the decision-making process within a hospital. By leveraging Python's versatility, we aim to create a robust system that automates various tasks, reduces manual errors, and ultimately contributes to the enhancement of healthcare services.

* 1. **Objective:**

The primary objective of this project is to develop a Hospital Management System using Python that addresses the complexities of healthcare administration. The system aims to streamline processes such as patient registration, appointment scheduling, billing, and inventory management, fostering a more organized and efficient healthcare environment. By leveraging Python's capabilities, we seek to create a user-friendly and scalable solution that caters to the diverse needs of hospital staff, administrators, and patients.

* 1. **Problem Statemen**t:

The current state of hospital management often involves manual and paper-based processes, leading to inefficiencies, errors, and delays. Inaccurate record-keeping, difficulties in appointment scheduling, and challenges in managing resources can hinder the smooth operation of healthcare facilities. This project aims to tackle these issues by developing a comprehensive Hospital Management System using Python, providing a centralized platform for data management, communication, and decision support.

**1.4 Challenges:**

* Integration of Existing Systems: Integrating the new HMS with existing hospital systems (such as Electronic Health Records) poses a challenge, requiring careful planning and execution to ensure seamless data flow and interoperability.
* User Adoption: Ensuring that the hospital staff is comfortable with the new system is crucial. User training and change management strategies must be implemented to facilitate a smooth transition and promote user acceptance.
* Data Security and Privacy: Healthcare systems deal with sensitive patient information, making data security and privacy paramount. Implementing robust security measures to protect patient data and comply with relevant regulations is a critical challenge.
* Scalability: As the hospital grows, the system should be able to scale accordingly. Designing the HMS to accommodate an increasing number of patients, staff, and services while maintaining performance is a key challenge.
* Real-time Updates: Providing real-time updates on patient statuses, bed availability, and resource utilization requires efficient data processing and communication mechanisms. Achieving low-latency and reliable updates is a technical challenge to be addressed.
* Customization and Flexibility: Hospitals have diverse workflows and requirements. Designing the system to be customizable and flexible enough to adapt to different hospital settings and workflows is a challenge that needs careful consideration.

By addressing these challenges, the Hospital Management System developed in Python aims to contribute to the overall efficiency, accuracy, and quality of healthcare services provided by the hospital.

1. **REQUIREMENTS**

**2.1 Software Requirements**

* Operating System Compatibility The application should be compatible with popular operating systems, including Windows, macOS, and Linux.
* Python interpreter, Tkinter
* GUI Framework Utilize Tkinter to create an intuitive and responsive graphical user interface (GUI) for E healthcare management system
* Database Management for database connectivity to support various relational databases (e.g., MySQL, PostgreSQL, SQLite).
* User Authentication Optional but recommended: Provide user authentication mechanisms to safeguard data access.
* Data Storage Define data structures and schema for efficient storage and retrieval of expense records.

**2.2 Hardware Requirements**

* Processor (1 GHz+)
* Memory (RAM, 2 GB+)
* Storage Space (Variable)
* Screen Resolution (1280x720+)
* Input Devices (Keyboard, Mouse)
* Network Connection (For remote database)

1. **ARCHITECTURE AND DESIGN**
   1. **Flow Chart**

Login Page

Patient Tab

Doctor Tab

Appointments Tab

Pharmacy Tab

Report Tab

1. **IMPLEMENTATION**

**5.1 Source Code**

import tkinter as tk

from tkinter import ttk

import mysql.connector

from datetime import datetime

# Connect to MySQL

db = mysql.connector.connect(

host="localhost",

user="root",

password="\*\*\*\*\*\*\*\*\*",

database="Hospital"

)

cursor = db.cursor()

cursor.execute("""

CREATE TABLE IF NOT EXISTS patients (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255),

age INT,

gender VARCHAR(10),

phone\_number VARCHAR(15),

bill\_due DECIMAL(10, 2)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS doctors (

id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(255),

specialization VARCHAR(255),

salary DECIMAL(10, 2)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS appointments (

id INT AUTO\_INCREMENT PRIMARY KEY,

patient\_id INT,

doctor\_id INT,

time VARCHAR(20),

date DATE,

FOREIGN KEY (patient\_id) REFERENCES patients(id),

FOREIGN KEY (doctor\_id) REFERENCES doctors(id)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS pharmacy (

id INT AUTO\_INCREMENT PRIMARY KEY,

medicine\_name VARCHAR(255),

quantity INT,

price DECIMAL(10, 2),

expiry\_date DATE

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS reports (

id INT AUTO\_INCREMENT PRIMARY KEY,

patient\_id INT,

doctor\_id INT,

diagnosis TEXT,

medicine\_prescribed TEXT,

bill\_amount DECIMAL(10, 2),

FOREIGN KEY (patient\_id) REFERENCES patients(id),

FOREIGN KEY (doctor\_id) REFERENCES doctors(id)

)

""")

cursor.execute("""

CREATE TABLE IF NOT EXISTS users (

id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255),

password VARCHAR(255)

)

""")

db.commit()

# GUI

class HospitalManagementSystem:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("E-Healthcare Management System")

# Hospital Name

self.hospital\_name\_label = tk.Label(self.root, text="SRM Hospitals", font=('Helvetica', 26, 'bold'))

self.hospital\_name\_label.pack(pady=10)

# Live Date and Time

self.date\_time\_label = tk.Label(self.root, font=('Helvetica', 12))

self.date\_time\_label.pack(pady=5)

# Tabs

self.notebook = ttk.Notebook(root)

self.login\_tab = ttk.Frame(self.notebook)

self.patient\_tab = ttk.Frame(self.notebook)

self.doctor\_tab = ttk.Frame(self.notebook)

self.appointment\_tab = ttk.Frame(self.notebook)

self.pharmacy\_tab = ttk.Frame(self.notebook)

self.report\_tab = ttk.Frame(self.notebook)

self.notebook.add(self.login\_tab, text='Login')

self.notebook.add(self.patient\_tab, text='Patients', state='disabled')

self.notebook.add(self.doctor\_tab, text='Doctors', state='disabled')

self.notebook.add(self.appointment\_tab, text='Appointments', state='disabled')

self.notebook.add(self.pharmacy\_tab, text='Pharmacy', state='disabled')

self.notebook.add(self.report\_tab, text='Report', state='disabled')

# Login Tab

self.create\_login\_tab()

# Patient Tab

self.create\_patient\_tab()

# Doctor Tab

self.create\_doctor\_tab()

# Appointment Tab

self.create\_appointment\_tab()

# Pharmacy Tab

self.create\_pharmacy\_tab()

#Report tab

self.create\_report\_tab()

self.notebook.pack()

# Update date and time every second

self.update\_date\_time()

def update\_date\_time(self):

current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

self.date\_time\_label.config(text=f"Current Date and Time: {current\_time}")

self.root.after(1000, self.update\_date\_time)

def create\_login\_tab(self):

# User ID Entry

self.user\_id\_label = tk.Label(self.login\_tab, text="User ID:")

self.user\_id\_entry = tk.Entry(self.login\_tab)

# Password Entry

self.password\_label = tk.Label(self.login\_tab, text="Password:")

self.password\_entry = tk.Entry(self.login\_tab, show="\*")

# Login Button

self.login\_button = tk.Button(self.login\_tab, text="Login", command=self.login)

# Grid

self.user\_id\_label.grid(row=0, column=0, pady=10)

self.user\_id\_entry.grid(row=0, column=1, pady=10)

self.password\_label.grid(row=1, column=0, pady=10)

self.password\_entry.grid(row=1, column=1, pady=10)

self.login\_button.grid(row=2, column=0, columnspan=2, pady=10)

def login(self):

username = self.user\_id\_entry.get()

password = self.password\_entry.get()

# Check the credentials in the 'users' table

cursor.execute("SELECT \* FROM users WHERE username = %s AND password = %s", (username, password))

result = cursor.fetchone()

if result:

# If the login is successful, enable all tabs and switch to the Patients tab

for tab in (self.patient\_tab, self.doctor\_tab, self.appointment\_tab, self.pharmacy\_tab, self.report\_tab):

self.notebook.tab(tab, state='normal')

self.notebook.select(self.patient\_tab)

else:

# Display an error message for unsuccessful login

tk.messagebox.showerror("Login Failed", "Invalid username or password")

def create\_patient\_tab(self):

# Patient Entry

self.patient\_name\_label = tk.Label(self.patient\_tab, text="Name:")

self.patient\_name\_entry = tk.Entry(self.patient\_tab)

self.patient\_age\_label = tk.Label(self.patient\_tab, text="Age:")

self.patient\_age\_entry = tk.Entry(self.patient\_tab)

self.patient\_gender\_label = tk.Label(self.patient\_tab, text="Gender:")

self.patient\_gender\_var = tk.StringVar(self.patient\_tab)

self.patient\_gender\_combobox = ttk.Combobox(self.patient\_tab, textvariable=self.patient\_gender\_var,

values=["Male", "Female", "Other"])

self.phone\_number\_label = tk.Label(self.patient\_tab, text="Phone Number:")

self.phone\_number\_entry = tk.Entry(self.patient\_tab)

self.bill\_due\_label = tk.Label(self.patient\_tab, text="Bill Due:")

self.bill\_due\_entry = tk.Entry(self.patient\_tab)

self.add\_patient\_button = tk.Button(self.patient\_tab, text="Add Patient", command=self.add\_patient)

self.edit\_patient\_button = tk.Button(self.patient\_tab, text="Edit Patient", command=self.edit\_patient)

self.update\_patient\_button = tk.Button(self.patient\_tab, text="Update Patient", command=self.update\_patient)

self.delete\_patient\_button = tk.Button(self.patient\_tab, text="Delete Patient", command=self.delete\_patient)

# Patient Treeview

self.patients\_tree = ttk.Treeview(self.patient\_tab, columns=('ID', 'Name', 'Age', 'Gender', 'Phone Number', 'Bill Due'))

self.patients\_tree.heading('ID', text='ID')

self.patients\_tree.heading('Name', text='Name')

self.patients\_tree.heading('Age', text='Age')

self.patients\_tree.heading('Gender', text='Gender')

self.patients\_tree.heading('Phone Number', text='Phone Number')

self.patients\_tree.heading('Bill Due', text='Bill Due')

# Grid

self.patient\_name\_label.grid(row=0, column=0)

self.patient\_name\_entry.grid(row=0, column=1)

self.patient\_age\_label.grid(row=1, column=0)

self.patient\_age\_entry.grid(row=1, column=1)

self.patient\_gender\_label.grid(row=2, column=0)

self.patient\_gender\_combobox.grid(row=2, column=1)

self.phone\_number\_label.grid(row=3, column=0)

self.phone\_number\_entry.grid(row=3, column=1)

self.bill\_due\_label.grid(row=4, column=0)

self.bill\_due\_entry.grid(row=4, column=1)

self.add\_patient\_button.grid(row=5, column=0, columnspan=2)

self.edit\_patient\_button.grid(row=5, column=2, columnspan=2)

self.update\_patient\_button.grid(row=5, column=4, columnspan=2)

self.delete\_patient\_button.grid(row=5, column=6, columnspan=2)

self.patients\_tree.grid(row=6, column=0, columnspan=8)

# Load Patients on startup

self.load\_patients()

def add\_patient(self):

name = self.patient\_name\_entry.get()

age = self.patient\_age\_entry.get()

gender = self.patient\_gender\_var.get()

phone\_number = self.phone\_number\_entry.get()

bill\_due = self.bill\_due\_entry.get()

# Inserting data into the 'patients' table

cursor.execute("INSERT INTO patients (name, age, gender, phone\_number, bill\_due) VALUES (%s, %s, %s, %s, %s)",

(name, age, gender, phone\_number, bill\_due))

db.commit()

# Update the patient list

self.load\_patients()

def edit\_patient(self):

selected\_item = self.patients\_tree.selection()

if selected\_item:

patient\_id, name, age, gender, phone\_number, bill\_due = self.patients\_tree.item(selected\_item, 'values')

self.patient\_name\_entry.delete(0, tk.END)

self.patient\_age\_entry.delete(0, tk.END)

self.patient\_name\_entry.insert(0, name)

self.patient\_age\_entry.insert(0, age)

self.patient\_gender\_var.set(gender)

self.phone\_number\_entry.delete(0, tk.END)

self.phone\_number\_entry.insert(0, phone\_number)

self.bill\_due\_entry.delete(0, tk.END)

self.bill\_due\_entry.insert(0, bill\_due)

def update\_patient(self):

selected\_item = self.patients\_tree.selection()

if selected\_item:

patient\_id = self.patients\_tree.item(selected\_item, 'values')[0]

name = self.patient\_name\_entry.get()

age = self.patient\_age\_entry.get()

gender = self.patient\_gender\_var.get()

phone\_number = self.phone\_number\_entry.get()

bill\_due = self.bill\_due\_entry.get()

# Updating data in the 'patients' table

cursor.execute("UPDATE patients SET name=%s, age=%s, gender=%s, phone\_number=%s, bill\_due=%s WHERE id=%s",

(name, age, gender, phone\_number, bill\_due, patient\_id))

db.commit()

# Update the patient list

self.load\_patients()

def delete\_patient(self):

selected\_item = self.patients\_tree.selection()

if selected\_item:

patient\_id = self.patients\_tree.item(selected\_item, 'values')[0]

# Deleting data from the 'patients' table

cursor.execute("DELETE FROM patients WHERE id=%s", (patient\_id,))

db.commit()

# Update the patient list

self.load\_patients()

def load\_patients(self):

# Clearing the existing data in the treeview

for item in self.patients\_tree.get\_children():

self.patients\_tree.delete(item)

# Fetching patients from the database

cursor.execute("SELECT \* FROM patients")

patients = cursor.fetchall()

# Inserting the updated data

for patient in patients:

self.patients\_tree.insert('', 'end', values=patient)

def create\_doctor\_tab(self):

# Doctor Entry

self.doctor\_name\_label = tk.Label(self.doctor\_tab, text="Name:")

self.doctor\_name\_entry = tk.Entry(self.doctor\_tab)

self.specialization\_label = tk.Label(self.doctor\_tab, text="Specialization:")

self.specialization\_entry = tk.Entry(self.doctor\_tab)

self.salary\_label = tk.Label(self.doctor\_tab, text="Salary:")

self.salary\_entry = tk.Entry(self.doctor\_tab)

self.add\_doctor\_button = tk.Button(self.doctor\_tab, text="Add Doctor", command=self.add\_doctor)

self.edit\_doctor\_button = tk.Button(self.doctor\_tab, text="Edit Doctor", command=self.edit\_doctor)

self.update\_doctor\_button = tk.Button(self.doctor\_tab, text="Update Doctor", command=self.update\_doctor)

self.delete\_doctor\_button = tk.Button(self.doctor\_tab, text="Delete Doctor", command=self.delete\_doctor)

# Doctor Treeview

self.doctors\_tree = ttk.Treeview(self.doctor\_tab, columns=('ID', 'Name', 'Specialization', 'Salary'))

self.doctors\_tree.heading('ID', text='ID')

self.doctors\_tree.heading('Name', text='Name')

self.doctors\_tree.heading('Specialization', text='Specialization')

self.doctors\_tree.heading('Salary', text='Salary')

# Grid

self.doctor\_name\_label.grid(row=0, column=0)

self.doctor\_name\_entry.grid(row=0, column=1)

self.specialization\_label.grid(row=1, column=0)

self.specialization\_entry.grid(row=1, column=1)

self.salary\_label.grid(row=2, column=0)

self.salary\_entry.grid(row=2, column=1)

self.add\_doctor\_button.grid(row=5, column=0, columnspan=2)

self.edit\_doctor\_button.grid(row=5, column=2, columnspan=2)

self.update\_doctor\_button.grid(row=5, column=4, columnspan=2)

self.delete\_doctor\_button.grid(row=5, column=6, columnspan=2)

self.doctors\_tree.grid(row=6, column=0, columnspan=8)

# Load Doctors on startup

self.load\_doctors()

def add\_doctor(self):

name = self.doctor\_name\_entry.get()

specialization = self.specialization\_entry.get()

salary = self.salary\_entry.get()

# Inserting data into the 'doctors' table

cursor.execute("INSERT INTO doctors (name, specialization, salary) VALUES (%s, %s, %s)", (name, specialization, salary))

db.commit()

# Update the doctor list

self.load\_doctors()

def edit\_doctor(self):

selected\_item = self.doctors\_tree.selection()

if selected\_item:

doctor\_id, name, specialization, salary = self.doctors\_tree.item(selected\_item, 'values')

self.doctor\_name\_entry.delete(0, tk.END)

self.specialization\_entry.delete(0, tk.END)

self.salary\_entry.delete(0, tk.END)

self.doctor\_name\_entry.insert(0, name)

self.specialization\_entry.insert(0, specialization)

self.salary\_entry.insert(0, salary)

def update\_doctor(self):

selected\_item = self.doctors\_tree.selection()

if selected\_item:

doctor\_id = self.doctors\_tree.item(selected\_item, 'values')[0]

name = self.doctor\_name\_entry.get()

specialization = self.specialization\_entry.get()

salary = self.salary\_entry.get()

# Updating data in the 'doctors' table

cursor.execute("UPDATE doctors SET name=%s, specialization=%s, salary=%s WHERE id=%s", (name, specialization, salary, doctor\_id))

db.commit()

# Update the doctor list

self.load\_doctors()

def delete\_doctor(self):

selected\_item = self.doctors\_tree.selection()

if selected\_item:

doctor\_id = self.doctors\_tree.item(selected\_item, 'values')[0]

# Deleting data from the 'doctors' table

cursor.execute("DELETE FROM doctors WHERE id=%s", (doctor\_id,))

db.commit()

# Update the doctor list

self.load\_doctors()

def load\_doctors(self):

# Clearing the existing data in the treeview

for item in self.doctors\_tree.get\_children():

self.doctors\_tree.delete(item)

# Fetching doctors from the database

cursor.execute("SELECT \* FROM doctors")

doctors = cursor.fetchall()

# Inserting the updated data

for doctor in doctors:

self.doctors\_tree.insert('', 'end', values=doctor)

def create\_appointment\_tab(self):

# Appointment Entry

self.appointment\_patient\_label = tk.Label(self.appointment\_tab, text="Patient:")

self.appointment\_patient\_var = tk.StringVar(self.appointment\_tab)

self.appointment\_patient\_combobox = ttk.Combobox(self.appointment\_tab, textvariable=self.appointment\_patient\_var,

values=self.get\_patient\_names())

self.appointment\_doctor\_label = tk.Label(self.appointment\_tab, text="Doctor:")

self.appointment\_doctor\_var = tk.StringVar(self.appointment\_tab)

self.appointment\_doctor\_combobox = ttk.Combobox(self.appointment\_tab, textvariable=self.appointment\_doctor\_var,

values=self.get\_doctor\_names())

self.appointment\_time\_label = tk.Label(self.appointment\_tab, text="Time:")

self.appointment\_time\_entry = tk.Entry(self.appointment\_tab)

self.appointment\_date\_label = tk.Label(self.appointment\_tab, text="Date:")

self.appointment\_date\_entry = tk.Entry(self.appointment\_tab)

self.add\_appointment\_button = tk.Button(self.appointment\_tab, text="Add Appointment", command=self.add\_appointment)

self.edit\_appointment\_button = tk.Button(self.appointment\_tab, text="Edit Appointment", command=self.edit\_appointment)

self.update\_appointment\_button = tk.Button(self.appointment\_tab, text="Update Appointment", command=self.update\_appointment)

self.delete\_appointment\_button = tk.Button(self.appointment\_tab, text="Delete Appointment", command=self.delete\_appointment)

# Appointment Treeview

self.appointments\_tree = ttk.Treeview(self.appointment\_tab, columns=('ID', 'Patient', 'Doctor', 'Time', 'Date'))

self.appointments\_tree.heading('ID', text='ID')

self.appointments\_tree.heading('Patient', text='Patient')

self.appointments\_tree.heading('Doctor', text='Doctor')

self.appointments\_tree.heading('Time', text='Time')

self.appointments\_tree.heading('Date', text='Date')

# Grid

self.appointment\_patient\_label.grid(row=0, column=0)

self.appointment\_patient\_combobox.grid(row=0, column=1)

self.appointment\_doctor\_label.grid(row=1, column=0)

self.appointment\_doctor\_combobox.grid(row=1, column=1)

self.appointment\_time\_label.grid(row=2, column=0)

self.appointment\_time\_entry.grid(row=2, column=1)

self.appointment\_date\_label.grid(row=3, column=0)

self.appointment\_date\_entry.grid(row=3, column=1)

self.add\_appointment\_button.grid(row=4, column=0, columnspan=2)

self.edit\_appointment\_button.grid(row=4, column=2, columnspan=2)

self.update\_appointment\_button.grid(row=4, column=4, columnspan=2)

self.delete\_appointment\_button.grid(row=4, column=6, columnspan=2)

self.appointments\_tree.grid(row=12, column=0, columnspan=8)

# Load Appointments on startup

self.load\_appointments()

def add\_appointment(self):

patient\_name = self.appointment\_patient\_var.get()

doctor\_name = self.appointment\_doctor\_var.get()

time = self.appointment\_time\_entry.get()

date = self.appointment\_date\_entry.get()

# Get patient and doctor IDs

patient\_id = self.get\_patient\_id\_by\_name(patient\_name)

doctor\_id = self.get\_doctor\_id\_by\_name(doctor\_name)

if patient\_id and doctor\_id:

# Inserting data into the 'appointments' table

cursor.execute("INSERT INTO appointments (patient\_id, doctor\_id, time, date) VALUES (%s, %s, %s, %s)",

(patient\_id, doctor\_id, time, date))

db.commit()

# Update the appointment list

self.load\_appointments()

def edit\_appointment(self):

selected\_item = self.appointments\_tree.selection()

if selected\_item:

appointment\_id, patient\_name, doctor\_name, time = self.appointments\_tree.item(selected\_item, 'values')

self.appointment\_patient\_var.set(patient\_name)

self.appointment\_doctor\_var.set(doctor\_name)

self.appointment\_time\_entry.delete(0, tk.END)

self.appointment\_time\_entry.insert(0, time)

def update\_appointment(self):

selected\_item = self.appointments\_tree.selection()

if selected\_item:

appointment\_id = self.appointments\_tree.item(selected\_item, 'values')[0]

patient\_name = self.appointment\_patient\_var.get()

doctor\_name = self.appointment\_doctor\_var.get()

time = self.appointment\_time\_entry.get()

# Get patient and doctor IDs

patient\_id = self.get\_patient\_id\_by\_name(patient\_name)

doctor\_id = self.get\_doctor\_id\_by\_name(doctor\_name)

if patient\_id and doctor\_id:

# Updating data in the 'appointments' table

cursor.execute("UPDATE appointments SET patient\_id=%s, doctor\_id=%s, time=%s WHERE id=%s",

(patient\_id, doctor\_id, time, appointment\_id))

db.commit()

# Update the appointment list

self.load\_appointments()

def delete\_appointment(self):

selected\_item = self.appointments\_tree.selection()

if selected\_item:

appointment\_id = self.appointments\_tree.item(selected\_item, 'values')[0]

# Deleting data from the 'appointments' table

cursor.execute("DELETE FROM appointments WHERE id=%s", (appointment\_id,))

db.commit()

# Update the appointment list

self.load\_appointments()

def load\_appointments(self):

# Clearing the existing data in the treeview

for item in self.appointments\_tree.get\_children():

self.appointments\_tree.delete(item)

# Fetching appointments from the database

cursor.execute("SELECT \* FROM appointments")

appointments = cursor.fetchall()

# Inserting the updated data into the treeview

for appointment in appointments:

# Format the date before inserting into the Treeview

formatted\_date = datetime.strftime(appointment[4], "%Y-%m-%d")

self.appointments\_tree.insert('', 'end', values=(appointment[0], self.get\_patient\_name\_by\_id(appointment[1]),

self.get\_doctor\_name\_by\_id(appointment[2]), appointment[3],

formatted\_date))

def create\_pharmacy\_tab(self):

# Medicine Entry

self.medicine\_id\_label = tk.Label(self.pharmacy\_tab, text="Medicine ID:")

self.medicine\_id\_entry = tk.Entry(self.pharmacy\_tab, state='disabled')

self.medicine\_name\_label = tk.Label(self.pharmacy\_tab, text="Medicine Name:")

self.medicine\_name\_entry = tk.Entry(self.pharmacy\_tab)

self.medicine\_quantity\_label = tk.Label(self.pharmacy\_tab, text="Quantity:")

self.medicine\_quantity\_entry = tk.Entry(self.pharmacy\_tab)

self.medicine\_cost\_label = tk.Label(self.pharmacy\_tab, text="Medicine Cost:")

self.medicine\_cost\_entry = tk.Entry(self.pharmacy\_tab)

self.medicine\_expiry\_label = tk.Label(self.pharmacy\_tab, text="Expiry Date (YYYY-MM-DD):")

self.medicine\_expiry\_entry = tk.Entry(self.pharmacy\_tab)

self.add\_medicine\_button = tk.Button(self.pharmacy\_tab, text="Add Medicine", c o command=self.add\_medicine)

selfco.edit\_medicine\_button = tk.Button(self.pharmacy\_tab, text="Edit Medicine", c o mmand=self.edit\_medicine)

self.update\_medicine\_button = tk.Button(self.pharmacy\_tab, text="Update Medicine", command=self.update\_medicine)

self.delete\_medicine\_button = tk.Button(self.pharmacy\_tab, text="Delete Medicine", command=self.delete\_medicine)

# Medicine Treeview

self.medicines\_tree = ttk.Treeview(self.pharmacy\_tab, columns=('ID', 'Medicine Name', 'Quantity', 'Cost', 'Expiry Date'))

self.medicines\_tree.heading('ID', text='ID')

self.medicines\_tree.heading('Medicine Name', text='Medicine Name')

self.medicines\_tree.heading('Quantity', text='Quantity')

self.medicines\_tree.heading('Cost', text='Cost')

self.medicines\_tree.heading('Expiry Date', text='Expiry Date')

# Grid

self.medicine\_id\_label.grid(row=0, column=0)

self.medicine\_id\_entry.grid(row=0, column=1)

self.medicine\_name\_label.grid(row=1, column=0)

self.medicine\_name\_entry.grid(row=1, column=1)

self.medicine\_quantity\_label.grid(row=2, column=0)

self.medicine\_quantity\_entry.grid(row=2, column=1)

self.medicine\_cost\_label.grid(row=3, column=0)

self.medicine\_cost\_entry.grid(row=3, column=1)

self.medicine\_expiry\_label.grid(row=4, column=0)

self.medicine\_expiry\_entry.grid(row=4, column=1)

self.add\_medicine\_button.grid(row=5, column=0, columnspan=2)

self.edit\_medicine\_button.grid(row=5, column=2, columnspan=2)

self.update\_medicine\_button.grid(row=5, column=4, columnspan=2)

self.delete\_medicine\_button.grid(row=5, column=6, columnspan=2)

self.medicines\_tree.grid(row=10, column=0, columnspan=8)

# Load Medicines on startup

self.load\_medicines()

def add\_medicine(self):

name = self.medicine\_name\_entry.get()

quantity = self.medicine\_quantity\_entry.get()

cost = self.medicine\_cost\_entry.get()

expiry\_date = self.medicine\_expiry\_entry.get()

# Inserting data into the 'pharmacy' table

cursor.execute("INSERT INTO pharmacy (medicine\_name, quantity, price, expiry\_date) VALUES (%s, %s, %s, %s)",

(name, quantity, cost, expiry\_date))

db.commit()

# Update the medicine list

self.load\_medicines()

def edit\_medicine(self):

selected\_item = self.medicines\_tree.selection()

if selected\_item:

medicine\_id, name, quantity, cost, expiry\_date = self.medicines\_tree.item(selected\_item, 'values')

self.medicine\_name\_entry.delete(0, tk.END)

self.medicine\_quantity\_entry.delete(0, tk.END)

self.medicine\_cost\_entry.delete(0, tk.END)

self.medicine\_expiry\_entry.delete(0, tk.END)

self.medicine\_name\_entry.insert(0, name)

self.medicine\_quantity\_entry.insert(0, quantity)

self.medicine\_cost\_entry.insert(0, cost)

self.medicine\_expiry\_entry.insert(0, expiry\_date)

def update\_medicine(self):

selected\_item = self.medicines\_tree.selection()

if selected\_item:

medicine\_id = self.medicines\_tree.item(selected\_item, 'values')[0]

name = self.medicine\_name\_entry.get()

quantity = self.medicine\_quantity\_entry.get()

cost = self.medicine\_cost\_entry.get()

expiry\_date = self.medicine\_expiry\_entry.get()

# Updating data in the 'pharmacy' table

cursor.execute("UPDATE pharmacy SET medicine\_name=%s, quantity=%s, price=%s, expiry\_date=%s WHERE id=%s",

(name, quantity, cost, expiry\_date, medicine\_id))

db.commit()

# Update the medicine list

self.load\_medicines()

def delete\_medicine(self):

selected\_item = self.medicines\_tree.selection()

if selected\_item:

medicine\_id = self.medicines\_tree.item(selected\_item, 'values')[0]

# Deleting data from the 'pharmacy' table

cursor.execute("DELETE FROM pharmacy WHERE id=%s", (medicine\_id,))

db.commit()

# Update the medicine list

self.load\_medicines()

def load\_medicines(self):

# Clearing the existing data in the treeview

for item in self.medicines\_tree.get\_children():

self.medicines\_tree.delete(item)

# Fetching data from the 'pharmacy' table

cursor.execute("SELECT \* FROM pharmacy")

medicines = cursor.fetchall()

# Populating the treeview with the data

for medicine in medicines:

self.medicines\_tree.insert("", "end", values=medicine)

def get\_patient\_names(self):

cursor.execute("SELECT name FROM patients")

return [row[0] for row in cursor.fetchall()]

def get\_doctor\_names(self):

cursor.execute("SELECT name FROM doctors")

return [row[0] for row in cursor.fetchall()]

def get\_patient\_id\_by\_name(self, name):

cursor.execute("SELECT id FROM patients WHERE name = %s", (name,))

result = cursor.fetchone()

return result[0] if result else None

def get\_doctor\_id\_by\_name(self, name):

cursor.execute("SELECT id FROM doctors WHERE name = %s", (name,))

result = cursor.fetchone()

return result[0] if result else None

def get\_patient\_name\_by\_id(self, patient\_id):

cursor.execute("SELECT name FROM patients WHERE id = %s", (patient\_id,))

result = cursor.fetchone()

return result[0] if result else None

def get\_doctor\_name\_by\_id(self, doctor\_id):

cursor.execute("SELECT name FROM doctors WHERE id = %s", (doctor\_id,))

result = cursor.fetchone()

return result[0] if result else None

def create\_report\_tab(self):

# Report Entry

self.report\_patient\_label = tk.Label(self.report\_tab, text="Patient:")

self.report\_patient\_var = tk.StringVar(self.report\_tab)

self.report\_patient\_combobox = ttk.Combobox(self.report\_tab, textvariable=self.report\_patient\_var,

values=self.get\_patient\_names())

self.report\_doctor\_label = tk.Label(self.report\_tab, text="Doctor:")

self.report\_doctor\_var = tk.StringVar(self.report\_tab)

self.report\_doctor\_combobox = ttk.Combobox(self.report\_tab, textvariable=self.report\_doctor\_var,

values=self.get\_doctor\_names())

self.diagnosis\_label = tk.Label(self.report\_tab, text="Diagnosis:")

self.diagnosis\_entry = tk.Entry(self.report\_tab)

self.medicine\_prescribed\_label = tk.Label(self.report\_tab, text="Medicine Prescribed:")

self.medicine\_prescribed\_entry = tk.Entry(self.report\_tab)

self.bill\_amount\_label = tk.Label(self.report\_tab, text="Bill Amount:")

self.bill\_amount\_entry = tk.Entry(self.report\_tab)

self.add\_report\_button = tk.Button(self.report\_tab, text="Add Report", command=self.add\_report)

self.edit\_report\_button = tk.Button(self.report\_tab, text="Edit Report", command=self.edit\_report)

self.update\_report\_button = tk.Button(self.report\_tab, text="Update Report", command=self.update\_report)

self.delete\_report\_button = tk.Button(self.report\_tab, text="Delete Report", command=self.delete\_report)

# Report Treeview

self.reports\_tree = ttk.Treeview(self.report\_tab, columns=('ID', 'Patient', 'Doctor', 'Diagnosis', 'Medicine Prescribed', 'Bill Amount'))

self.reports\_tree.heading('ID', text='ID')

self.reports\_tree.heading('Patient', text='Patient')

self.reports\_tree.heading('Doctor', text='Doctor')

self.reports\_tree.heading('Diagnosis', text='Diagnosis')

self.reports\_tree.heading('Medicine Prescribed', text='Medicine Prescribed')

self.reports\_tree.heading('Bill Amount', text='Bill Amount')

# Grid

self.report\_patient\_label.grid(row=0, column=0)

self.report\_patient\_combobox.grid(row=0, column=1)

self.report\_doctor\_label.grid(row=1, column=0)

self.report\_doctor\_combobox.grid(row=1, column=1)

self.diagnosis\_label.grid(row=2, column=0)

self.diagnosis\_entry.grid(row=2, column=1)

self.medicine\_prescribed\_label.grid(row=3, column=0)

self.medicine\_prescribed\_entry.grid(row=3, column=1)

self.bill\_amount\_label.grid(row=4, column=0)

self.bill\_amount\_entry.grid(row=4, column=1)

self.add\_report\_button.grid(row=5, column=0, columnspan=2)

self.edit\_report\_button.grid(row=5, column=2, columnspan=2)

self.update\_report\_button.grid(row=5, column=4, columnspan=2)

self.delete\_report\_button.grid(row=5, column=6, columnspan=2)

self.reports\_tree.grid(row=10, column=0, columnspan=8)

# Load Reports on startup

self.load\_reports()

def add\_report(self):

patient\_name = self.report\_patient\_var.get()

doctor\_name = self.report\_doctor\_var.get()

diagnosis = self.diagnosis\_entry.get()

medicine\_prescribed = self.medicine\_prescribed\_entry.get()

bill\_amount = self.bill\_amount\_entry.get()

# Get patient and doctor IDs

patient\_id = self.get\_patient\_id\_by\_name(patient\_name)

doctor\_id = self.get\_doctor\_id\_by\_name(doctor\_name)

if patient\_id and doctor\_id:

# Inserting data into the 'reports' table

cursor.execute("INSERT INTO reports (patient\_id, doctor\_id, diagnosis, medicine\_prescribed, bill\_amount) VALUES (%s, %s, %s, %s, %s)",

(patient\_id, doctor\_id, diagnosis, medicine\_prescribed, bill\_amount))

db.commit()

# Update the report list

self.load\_reports()

def edit\_report(self):

selected\_item = self.reports\_tree.selection()

if selected\_item:

report\_id, patient\_name, doctor\_name, diagnosis, medicine\_prescribed, bill\_amount = self.reports\_tree.item(selected\_item, 'values')

self.report\_patient\_var.set(patient\_name)

self.report\_doctor\_var.set(doctor\_name)

self.diagnosis\_entry.delete(0, tk.END)

self.diagnosis\_entry.insert(0, diagnosis)

self.medicine\_prescribed\_entry.delete(0, tk.END)

self.medicine\_prescribed\_entry.insert(0, medicine\_prescribed)

self.bill\_amount\_entry.delete(0, tk.END)

self.bill\_amount\_entry.insert(0, bill\_amount)

def update\_report(self):

selected\_item = self.reports\_tree.selection()

if selected\_item:

report\_id = self.reports\_tree.item(selected\_item, 'values')[0]

patient\_name = self.report\_patient\_var.get()

doctor\_name = self.report\_doctor\_var.get()

diagnosis = self.diagnosis\_entry.get()

medicine\_prescribed = self.medicine\_prescribed\_entry.get()

bill\_amount = self.bill\_amount\_entry.get()

# Get patient and doctor IDs

patient\_id = self.get\_patient\_id\_by\_name(patient\_name)

doctor\_id = self.get\_doctor\_id\_by\_name(doctor\_name)

if patient\_id and doctor\_id:

# Updating data in the 'reports' table

cursor.execute("UPDATE reports SET patient\_id=%s, doctor\_id=%s, diagnosis=%s, m edicine\_prescribed=%s, bill\_amount=%s WHERE id=%s",

(patient\_id, doctor\_id, diagnosis, medicine\_prescribed, bill\_amount, report\_id))

db.commit()

# Update the report list

self.load\_reports()

def delete\_report(self):

selected\_item = self.reports\_tree.selection()

if selected\_item:

report\_id = self.reports\_tree.item(selected\_item, 'values')[0]

# Deleting data from the 'reports' table

cursor.execute("DELETE FROM reports WHERE id=%s", (report\_id,))

db.commit()

# Update the report list

self.load\_reports()

def load\_reports(self):

# Clear existing items in the Treeview

for item in self.reports\_tree.get\_children():

self.reports\_tree.delete(item)

# Fetch data from the 'reports' table

cursor.execute("""

SELECT reports.id, patients.name AS patient\_name, doctors.name AS doctor\_name, diagnosis, medicine\_prescribed, bill\_amount

FROM reports

INNER JOIN patients ON reports.patient\_id = patients.id

INNER JOIN doctors ON reports.doctor\_id = doctors.id

""")

reports = cursor.fetchall()

# Populate the Treeview with report data

for report in reports:

self.reports\_tree.insert('', 'end', values=report)

if \_\_name\_\_ == "\_\_main\_\_":

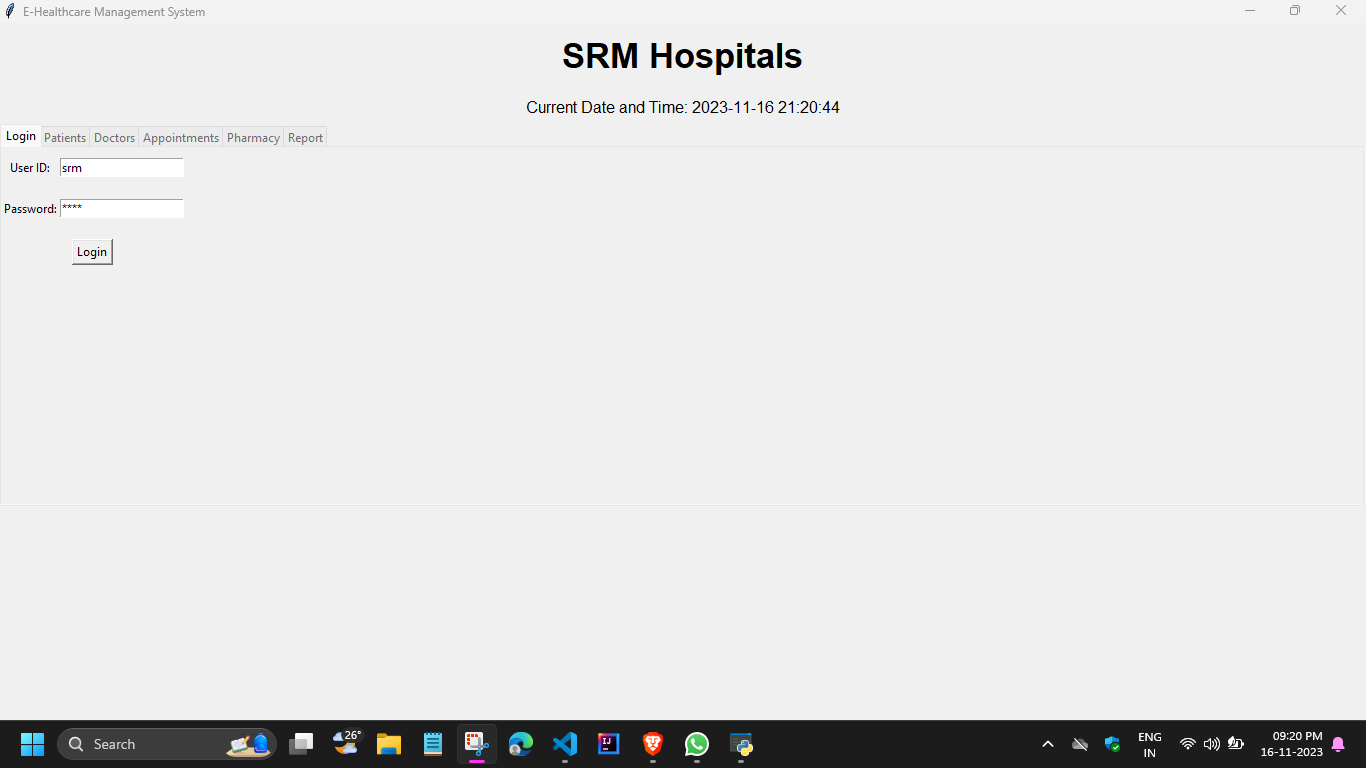
root = tk.Tk()

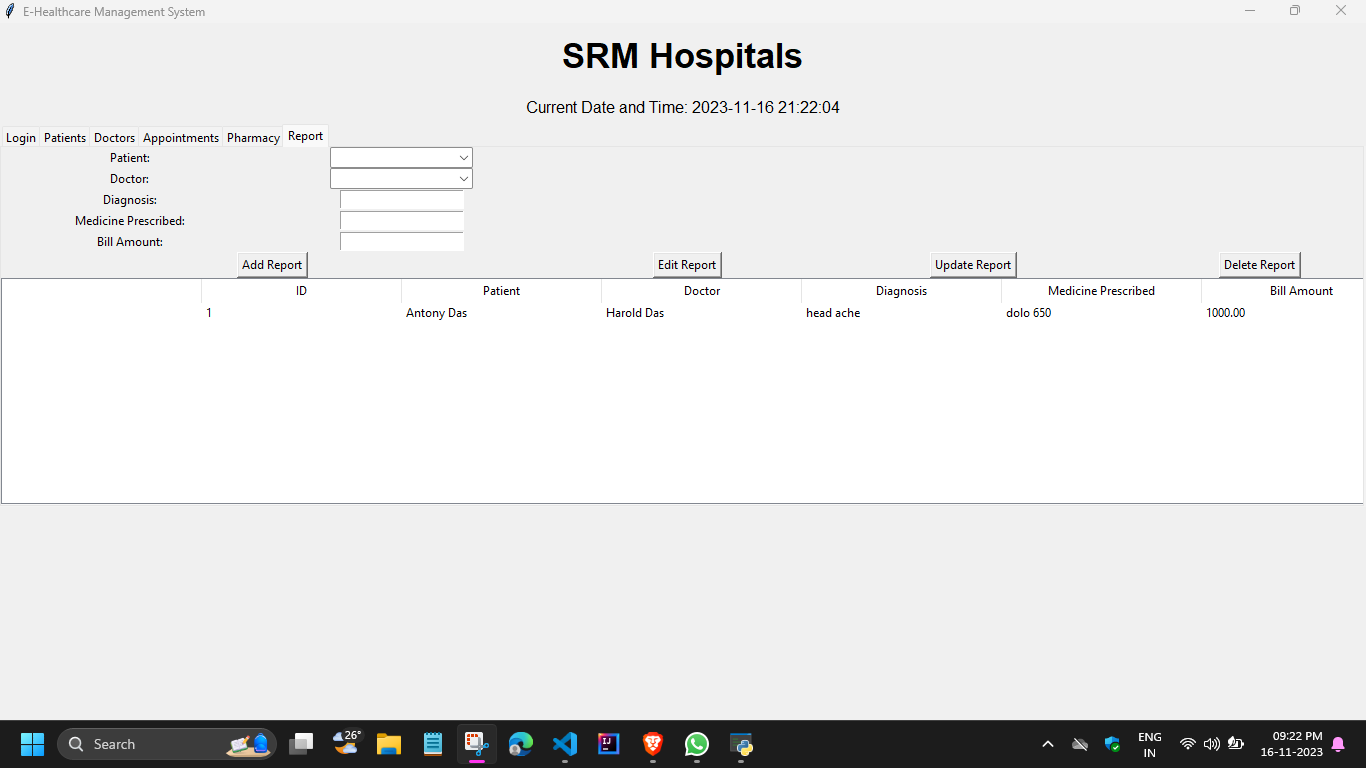
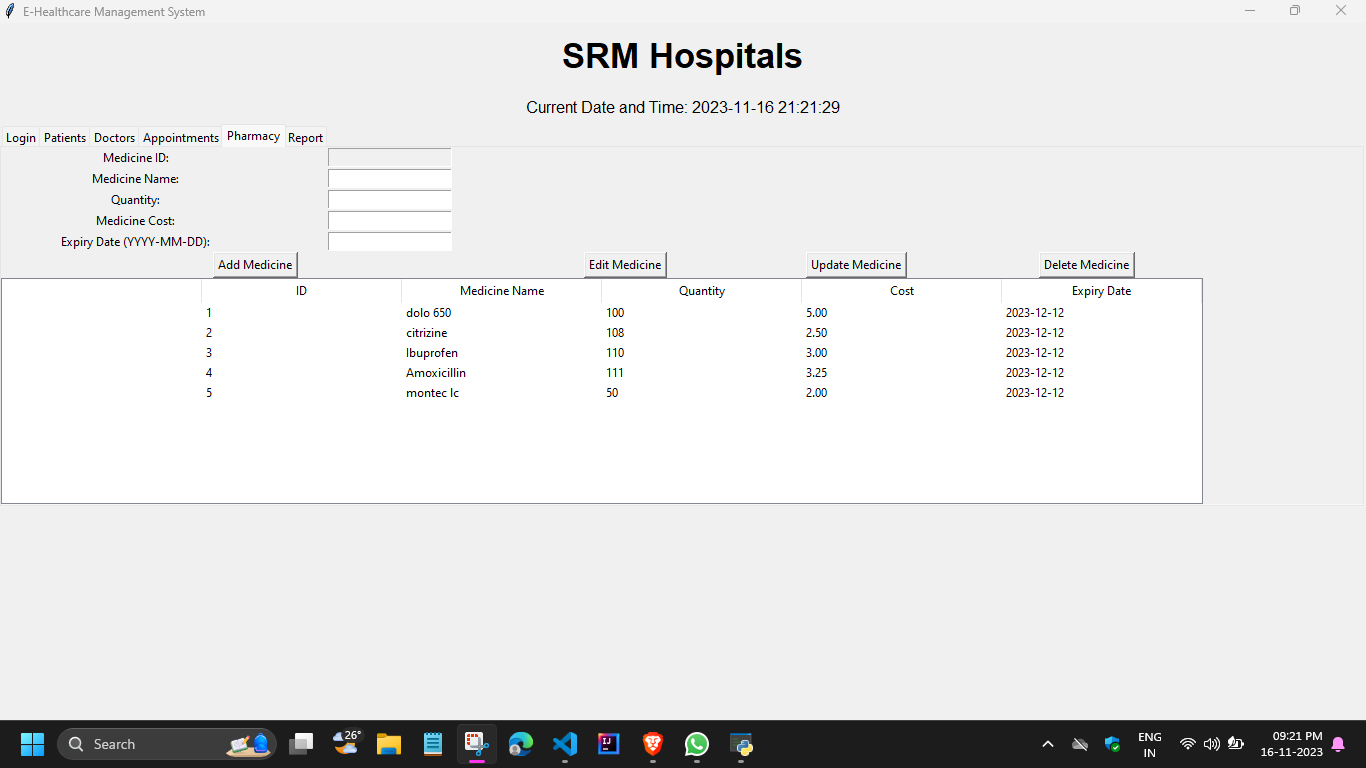
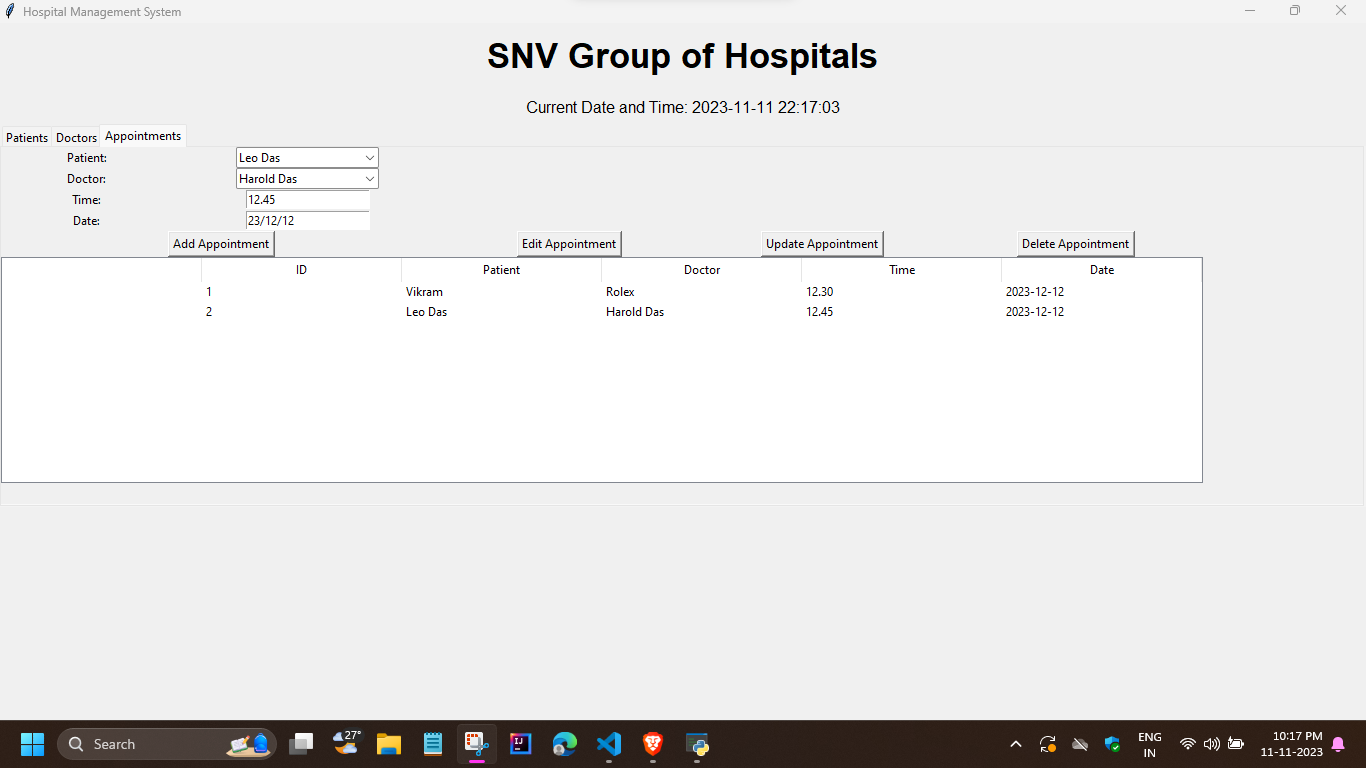
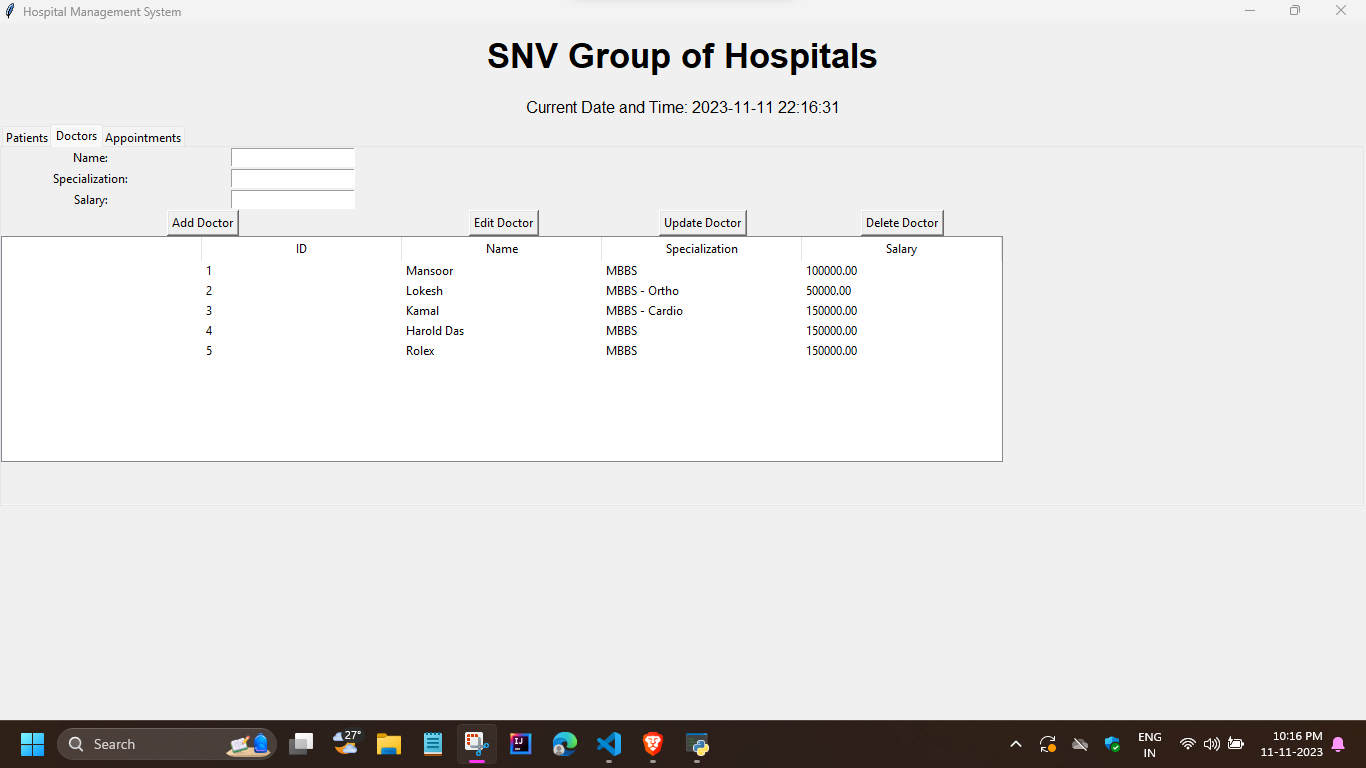
app = HospitalManagementSystem(root)

root.mainloop()

1. **RESULTS AND DISCUSSION**
   1. **Sample Input / Output**

**Login tab:**



**Other tabs**



1. **CONCLUSION**

In conclusion, the development of a Hospital Management System (HMS) using Python represents a significant stride towards enhancing the efficiency, accuracy, and overall quality of healthcare administration. Through a comprehensive literature survey, it is evident that the current state of hospital management often involves manual processes, leading to inefficiencies and potential errors. The integration of a well-designed HMS offers a solution to address these challenges and pave the way for a more streamlined healthcare ecosystem.

The primary objective of this project is to create a robust and user-friendly system that caters to the diverse needs of hospital staff, administrators, and patients. By leveraging Python's versatility, we aim to automate critical tasks such as patient registration, appointment scheduling, billing, and inventory management. The motivation behind this endeavor is rooted in the need for a centralized platform that not only facilitates efficient data management but also contributes to informed decision-making within the hospital environment.

While the project poses certain challenges, such as the integration of existing systems, user adoption, and ensuring data security and privacy, the literature survey has provided valuable insights into existing solutions and best practices. By addressing these challenges head-on, the developed HMS can serve as a scalable and adaptable solution for healthcare facilities, accommodating the growing demands of patient care and resource management.

In essence, the Hospital Management System using Python is poised to revolutionize healthcare administration by ushering in a new era of digitization, reducing manual errors, and fostering a more organized and efficient healthcare environment. Through careful consideration of the literature and a commitment to addressing challenges, this project aspires to contribute significantly to the advancement of healthcare systems, ultimately improving the delivery of patient care and optimizing the operational workflows within hospitals.

1. **REFERENCES**

* Programming with Python – E Balagurusamy
* Class 12 – Computer Science with Python – Sumita Arora
* Class 12 – Computer Science (New) – 083 – NCERT
* APP – SRM PPT Material
* Python: A Beginner’s Guide – Herbert Schildt
* Advanced Python – Textbook
* <https://openweathermap.org/api>
* <https://www.geeksforgeeks.org/introduction-to-tkinter/>
* <https://www.w3schools.com/sql/>
* <https://www3.ntu.edu.sg/home/ehchua/programming/python/tkinter>
* <https://www.youtube.com/watch?v=dPaUazOJOBc&list=PLGwb7xZHg-oNIUlzrLpurYaNfZBnU7hby>