# <CARE CONNECT>

*A*

*Mini Project Report*

*Submitted in partial fulfilment of the Requirements for the award of the Degree of*

# BACHELOR OF ENGINEERING

IN

# INFORMATION TECHNOLOGY

By

**<K.HARITHA><1602-23-737-084>**

**<R.VARSHINI><1602-23-737-128>**

**<TAGORE><1602-23-737-127>**



**Department of Information Technology Vasavi College of Engineering (Autonomous)**

# ACCREDITED BY NAAC WITH 'A++' GRADE

**(Affiliated to Osmania University and Approved by AICTE) Ibrahimbagh, Hyderabad-31**

**2024**

**Vasavi College of Engineering (Autonomous)**

***ACCREDITED BY NAAC WITH 'A++' GRADE***

**(Affiliated to Osmania University and Approved by AICTE) Hyderabad-500 031**

**Department of Information Technology**



# DECLARATION BY THE CANDIDATE

We, **K.Haritha, R.Varshini,** and **Tagore**

bearing hall ticket numbers, **1602-23-737-084, 1602-23-737-128** and **1602-23-737-127**, hereby declare that the project report entitled **“Care Connect”** is submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Information Technology**

This is a record of bonafide work carried out by us and the results embodied in this project report havenot been submitted to any other university or institute for the award of any other degree or diploma.

**K.Haritha**

**1602-23-737-084**

**R.Varshini**

**1602-23-737-128**

**B.Tagore**

**1602-23-737-127**

(Faculty In-Charge) (External Examiner) (Head,Dept of IT)

**ACKNOWLEDGMENT**

We extend our sincere thanks to Dr. S. V. Ramana, Principal, Vasavi College of Engineering, for his encouragement.

We express our sincere gratitude to Dr. K. Ram Mohan Rao, Professor & Head, Department of Information Technology, Vasavi College of Engineering, for introducing the Mini-Project module in our curriculum, and for his suggestions, motivation, and co-operation for the successful completion of our Mini Project.

We also want to thank and convey our gratitude towards our mini project coordinators Divya Madam, Anjani Devi Madam and Nanda Kumari Madam and our reviewer Mr. Krushi Raj Tula for guiding us in understanding the process of project development & giving us timely suggestions at every phase.

We would also like to sincerely thank the project reviewers for their valuable input and suggestions.

# Abstract

The **Care Connect** is a dynamic and user-centric application designed to enhance the efficiency and reliability of healthcare administration. This project aims to address common challenges in managing healthcare operations by digitizing processes such as patient registration, appointment scheduling, and healthcare service management. It offers a streamlined platform for managing critical information while ensuring accessibility and ease of use for both staff and patients.

Key features of the system include a robust patient registration module for securely storing personal and medical details, an intuitive appointment management system for real-time scheduling, and a categorized list of doctors based on specialization to help patients connect with the right healthcare professionals. Additionally, the application showcases a detailed overview of available healthcare services, ensuring transparency and informed decision-making for patients. The system also includes functionality to modify existing data, such as patient details or appointment schedules, providing flexibility and maintaining data accuracy over time.

Developed using Python with Tkinter for the graphical user interface, the project employs a relational database for secure and structured data management. The modular design ensures scalability and maintainability, while the user-friendly interface minimizes the learning curve for non-technical users. This project integrates fundamental programming principles with practical problem-solving, offering a comprehensive solution that reflects the evolving needs of the healthcare sector.

The project highlights the potential of technology to modernize healthcare systems, reduce manual errors, and enhance overall productivity. It serves as a significant step toward bridging the gap between traditional healthcare practices and digital transformation, ultimately contributing to better patient outcomes and streamlined healthcare delivery.

### ****Table of Contents****

1. Cover Sheet – Page 1
2. Declaration – Page 2
3. Acknowledgements – Page 3
4. Abstract – Page 4
5. Table of Contents – Page 5
6. Abstract & Introduction – Pages 6-10
   * Project Domain Information
   * Features and Priorities
7. Technology – Pages 10-12
   * Software Requirements
   * Hardware Requirements
8. Proposed Work – Pages 13-40
   * Design and Workflow
   * Implementation (Module-wise Code and Logic)
   * Testing (Test Cases)
9. Results – Pages 41-53
   * Screenshots and Test Case Validation
10. Additional Knowledge Gained – Page 54
11. Conclusion and Future Work – Page 55
12. References – Page 56

# Abstract &Introduction

# Project Domain Overview

The domain of **Care Connect** is rooted in the integration of technology to streamline and optimize the essential administrative, clinical, and operational functions in healthcare facilities. Care Connect addresses the growing demand for efficient, user-friendly solutions in the management of healthcare services, catering to the needs of both patients and healthcare providers. he increasing complexity of managing patient records, scheduling appointments, maintaining accurate service data, and handling prescriptions underscores the critical need for systems like **Care Connect**.

Historically, healthcare operations relied heavily on manual processes, leading to inefficiencies, data errors, and communication gaps. Traditional approaches, such as paper-based appointment scheduling or physical patient recordkeeping, often resulted in delays, mismanagement, and compromised patient satisfaction. These challenges emphasized the need for a comprehensive system that centralizes and automates healthcare operations while maintaining a strong focus on accuracy and user experience.

Care Connect serves as a bridge between healthcare providers and patients by offering an integrated platform to handle key aspects of healthcare management. Its purpose is to modernize healthcare facilities by providing tools for seamless patient registration, streamlined appointment scheduling, an updated directory of available doctors and services, and efficient data modification capabilities. By centralizing these functions, Care Connect reduces administrative burdens and empowers healthcare providers to focus on delivering quality care.

One of the distinguishing features of Care Connect is its emphasis on user accessibility and engagement. Patients can effortlessly register their details, view and book appointments, and access an updated list of services and doctors. On the other hand, healthcare staff can manage records, modify data, and monitor operations in real-time. This dual-focus approach ensures that both patients and providers benefit from the system, fostering a more collaborative and efficient healthcare environment.

# Project Objectives

The primary objective of this project is to develop a **Healthcare Management System** that simplifies and automates essential administrative tasks. The system aims to provide:

1. **Efficient Data Management**: Centralized storage of patient, doctor, and service data with quick access and retrieval mechanisms.
2. **Streamlined Appointment Scheduling**: Real-time management of patient appointments, reducing conflicts and ensuring optimal utilization of healthcare resources.
3. **Comprehensive Service Listing**: Clear and accessible information about the services offered, enhancing transparency for patients.
4. **User-Friendly Interface**: An intuitive design to cater to non-technical users such as healthcare staff and patients.
5. **Data Modification and Accuracy**: Secure methods for updating existing records to maintain data accuracy over time.

# Project-Specific Features and Priorities

The **Care Connect** system has been designed with several key features that are essential for improving the overall management of healthcare services. Each feature is prioritized to ensure that the system efficiently meets the needs of both healthcare providers and patients. The following sections highlight the key features included in the project, with a focus on their functionality, priority, and the goals we aim to achieve.

# ****1. Patient Registration and Management****

**Priority:** High  
The **Patient Registration** feature is a foundational aspect of the Care Connect system. This module allows new patients to register by entering essential details such as their name, age, gender, contact information, medical history, and allergies, if any. For existing patients, the system facilitates updating personal data, ensuring that the records are always up to date. The importance of this feature lies in the ability to maintain accurate patient profiles, reducing errors and enhancing the overall efficiency of the healthcare services. By automating the registration process, the system eliminates the need for physical paperwork, making it easier for both patients and healthcare staff to manage patient information.

# ****2**. Appointment Scheduling and Management**

**Priority:** High  
The **Appointment Scheduling** feature is a critical element for ensuring efficient operation within healthcare facilities. Care Connect allows patients to book appointments with their desired healthcare provider by checking their available time slots. The system supports the rescheduling and cancellation of appointments as well. From an administrative perspective, staff members can view and manage all appointments, preventing conflicts in doctor schedules and ensuring a balanced workload. Real-time updates regarding available time slots and appointment statuses ensure that both patients and staff are always on the same page..

# ****3.** List of Doctors**

**Priority:** Medium  
Care Connect features a **Doctor Directory** that allows patients to search and view a list of available doctors based on specialization, availability, and qualifications. The system allows users to filter doctors by their specialties (e.g., cardiologists, dermatologists, general practitioners), making it easier for patients to find a doctor suited to their healthcare needs. Each doctor’s profile includes relevant information such as their experience, specialties, and working hours. This feature empowers patients to make informed decisions and helps healthcare administrators maintain an organized database of medical staff.

# ****4****. Service Listings and Availability

**Priority:** Medium  
The **Service Listings** feature provides a comprehensive overview of the medical services offered at the healthcare facility. It includes information on diagnostic tests, treatments, consultations, and other healthcare services. Each service entry includes details such as prerequisites, cost, and description, helping patients understand what services they can access. This feature is particularly beneficial for ensuring transparency and clarity about what is available, aiding patients in preparing for their visits. It also helps administrative staff manage and track the availability of various healthcare services.

# ****5. Data Modification and Management****

**Priority:** High  
Care Connect allows authorized healthcare administrators to **modify existing data** within the system. This includes updating patient details, appointment schedules, doctor availability, and healthcare services. The **data modification** feature is crucial for maintaining accurate and up-to-date records. For example, if a patient’s contact information changes or if a doctor’s schedule needs to be adjusted, the system provides an interface for easy data modifications.

# ****6. View and Manage Prescriptions****

**Priority:** Medium  
Patients can view their prescription history, including medication details, dosage instructions, and dates of issuance. This feature also allows healthcare providers to review and update prescriptions as needed. It provides an additional layer of convenience and safety, ensuring that prescriptions are accurate and easily accessible.

# ****7. User-Friendly Interface****

**Priority:** High  
A **user-friendly interface** is essential to ensure that the system is easy to use for both healthcare professionals and patients. The Care Connect system has been designed with a clean and intuitive layout, making it simple for users to navigate through various features such as registration, appointments, doctor listings, and service details. The goal of this feature is to ensure that even users with limited technological experience can use the system with minimal training, promoting widespread adoption within healthcare settings.

# ****8. Security and Privacy****

**Priority:** High  
Given the sensitive nature of healthcare data, **security and privacy** are top priorities for Care Connect. The system employs strong encryption techniques to protect patient data and follows industry standards for data security. Access to patient information and system features is strictly controlled through **role-based access**, ensuring that only authorized personnel can view or modify sensitive data. This feature complies with healthcare data protection regulations, such as HIPAA (Health Insurance Portability and Accountability Act), to safeguard patients' privacy and maintain trust in the system..

# ****Features and Priority****

The features of Care Connect have been prioritized based on their impact on healthcare operations and user experience. The most critical features—such as **patient registration**, **appointment scheduling**, **doctor directory**, and **data modification**—have been given the highest priority to ensure the system’s core functionalities are operational from the outset. Features like **service listings** and **reporting** are important but were assigned a lower priority to allow the development team to focus on the most essential aspects of the system first.

# Technology

# a. Software Requirements

The **Care Connect** system relies on a robust set of software tools and platforms to ensure seamless functionality, user-friendly interfaces, and secure data handling. Below are the primary software requirements:

1. **Programming Language**:
   * Python (Version 3.x) - Used for the backend logic and overall implementation.
2. **Front-End:**
   * Tkinter - For building the graphical user interface (GUI).
3. **Database Management**:
   * SQLite/MySQL - To store and manage patient records, appointments, doctor lists, services, and prescriptions.
4. **Integrated Development Environment (IDE)**:
   * VS Code, or any Python-compatible IDE for writing and debugging code.
5. **Libraries and Frameworks**:
   * Pillow - For handling images, such as doctor photos or service icons.
6. **Other Utilities**:
   * Email/Notification API - To send appointment confirmations and reminders (if implemented).
   * PDF Library - To generate digital prescriptions.
7. **Operating System**:
   * Windows, macOS, or Linux - Compatible with Python and the chosen database system.

# b. Hardware Requirements

To run the **Care Connect** system effectively, the following hardware specifications are recommended:

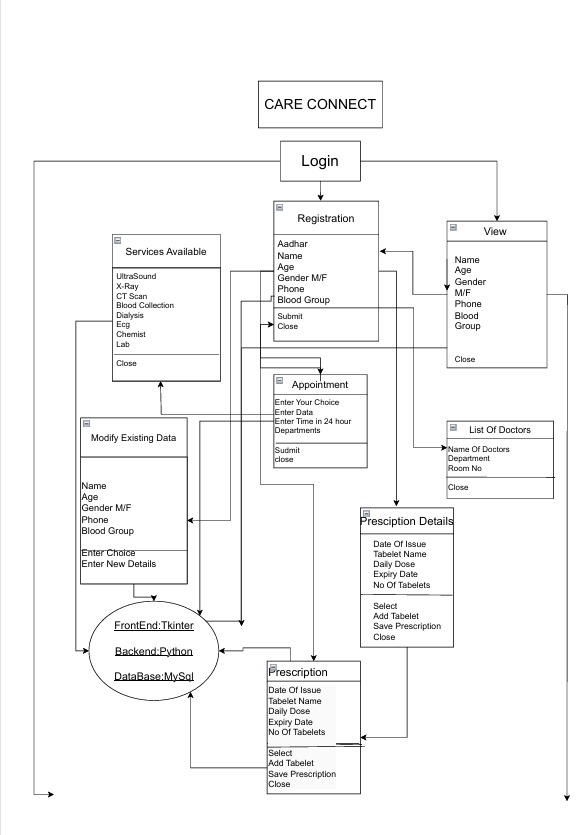
1. **Server Requirements**:
   * Processor: Dual-core or higher.
   * RAM: Minimum 4 GB (8 GB recommended for better performance).
   * Storage: 256 GB or higher to store patient and operational data.
2. **Client-Side Requirements**:
   * Processor: Any modern processor (e.g., Intel i3 or equivalent).
   * RAM: At least 2 GB.
   * Display: A monitor with a minimum resolution of 1366x768 for clear viewing of the GUI.
3. **Network Requirements**:
   * Reliable Internet connection for cloud-based data synchronization or appointment notifications (if applicable).
4. **Backup Solution**:
   * External hard drive or cloud storage service to ensure data redundancy and security.

These requirements ensure that the **Care Connect** system operates smoothly, providing a scalable and efficient platform for healthcare management.

# Proposed Work

# Design

# I . High-Level Design



# b.Implementation

I.Module Wise Code

# Module 1: User Authentication

from tkinter import \*

from PIL import Image, ImageTk

# Constants for default login

DEFAULT\_USERNAME = "admin"

DEFAULT\_PASSWORD = "password123"

# def show\_main\_window():

global root

username = username\_entry.get()

password = password\_entry.get()

if username == DEFAULT\_USERNAME and password == DEFAULT\_PASSWORD:

login\_window.destroy()

main\_dashboard()

else:

error\_label.config(text="Invalid username or password!", fg="red")

# Create the login window

login\_window = Tk()

login\_window.attributes('-fullscreen', True)

login\_window.title("Login - Care Connect")

login\_window.configure(bg='#e3f2fd')

# Login frame

login\_frame = Frame(login\_window, bg='#e3f2fd')

login\_frame.place(relx=0.5, rely=0.5, anchor=CENTER)

# Username and password labels

username\_label = Label(login\_frame, text="Username", font="arial 16", bg='#e3f2fd')

username\_label.grid(row=0, column=0, pady=10, padx=10, sticky="w")

username\_entry = Entry(login\_frame, font="arial 16", width=25)

username\_entry.grid(row=0, column=1, pady=10, padx=10)

password\_label = Label(login\_frame, text="Password", font="arial 16", bg='#e3f2fd')

password\_label.grid(row=1, column=0, pady=10, padx=10, sticky="w")

password\_entry = Entry(login\_frame, font="arial 16", width=25, show="\*")

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

# Error message label

error\_label = Label(login\_frame, text="", font="arial 12", bg='#e3f2fd')

error\_label.grid(row=2, column=0, columnspan=2)

# Buttons

login\_button = Button(

login\_frame, text="Login", font="arial 16 bold", bg='#a40182', fg='white', command=show\_main\_window)

login\_button.grid(row=3, column=0, pady=20)

close\_button = Button(

login\_frame, text="Close", font="arial 16 bold", bg='red', fg='white', command=login\_window.destroy)

close\_button.grid(row=3, column=1, pady=20)

login\_window.mainloop()

# Module 2: Database Setup

import mysql.connector as sqlcon

# Database connection setup

con = sqlcon.connect(host="localhost", user="root", password="Haritha@07")

cur = con.cursor(buffered=True)

if con.is\_connected():

print("Connection successful")

else:

print("Connection unsuccessful")

# Create database and tables

cur.execute("CREATE DATABASE IF NOT EXISTS Hospital")

cur.execute("USE Hospital")

cur.execute("""

CREATE TABLE IF NOT EXISTS appointment (

idno VARCHAR(12) PRIMARY KEY,

name CHAR(50),

age CHAR(3),

gender VARCHAR(20),

phone VARCHAR(10),

bg VARCHAR(3)

)

""")

cur.execute("""

CREATE TABLE IF NOT EXISTS appointment\_details (

appointment\_id INT AUTO\_INCREMENT PRIMARY KEY,

idno VARCHAR(12),

department VARCHAR(50),

doctor VARCHAR(50),

date DATE,

time VARCHAR(20),

FOREIGN KEY (idno) REFERENCES appointment(idno)

)

""")

cur.execute("""

CREATE TABLE IF NOT EXISTS prescription (

prescription\_id INT AUTO\_INCREMENT PRIMARY KEY,

idno VARCHAR(12),

date\_of\_issue DATE,

tablet\_name VARCHAR(100),

daily\_dose VARCHAR(50),

expiry\_date DATE,

no\_of\_tablets INT,

FOREIGN KEY (idno) REFERENCES appointment(idno)

)

""")

# Module 3:User Registration

# def register():

root1 = tk.Toplevel() # Use Toplevel for a new window

root1.title("Registration")

# Set the window to full screen

root1.attributes('-fullscreen', True)

# Set the background color

root1.configure(bg='#d8e2dc') # Light green background

# Main title label

label = tk.Label(root1, text="REGISTER YOURSELF", font='Arial 40 bold', bg='#598392', fg='white')

label.grid(row=0, column=0, columnspan=2, pady=30, padx=20)

# Font style for labels and entries

label\_font = ('Arial', 20, 'bold') # Font style for labels

entry\_width = 30 # Width for entry fields

# Create a frame for better layout of input fields

frame = tk.Frame(root1, bg='#f0f0f0')

frame.grid(row=1, column=0, columnspan=2, pady=20, padx=20)

# Labels and Entry widgets for registration fields using grid

l1 = tk.Label(frame, text="AADHAR CARD NO.", bg='#f0f0f0', font=label\_font)

l1.grid(row=0, column=0, padx=10, pady=10, sticky='e')

e1 = tk.Entry(frame, width=entry\_width, font=('Arial', 18, 'bold'), highlightthickness=1, borderwidth=2)

e1.grid(row=0, column=1, padx=10, pady=10)

l2 = tk.Label(frame, text="NAME", bg='#f0f0f0', font=label\_font)

l2.grid(row=1, column=0, padx=10, pady=10, sticky='e')

e2 = tk.Entry(frame, width=entry\_width, font=('Arial', 18, 'bold'), highlightthickness=1, borderwidth=2)

e2.grid(row=1, column=1, padx=10, pady=10)

l3 = tk.Label(frame, text="AGE", bg='#f0f0f0', font=label\_font)

l3.grid(row=2, column=0, padx=10, pady=10, sticky='e')

e3 = tk.Entry(frame, width=entry\_width, font=('Arial', 18, 'bold'), highlightthickness=1, borderwidth=2)

e3.grid(row=2, column=1, padx=10, pady=10)

# Gender Label

l4 = tk.Label(frame, text="GENDER", bg='#f0f0f0', font=label\_font)

l4.grid(row=3, column=0, padx=10, pady=10, sticky='e')

# Gender Radio Buttons

gender\_var = tk.StringVar(value="Male") # Default value is "Male"

# Frame to group the radio buttons for better alignment

gender\_frame = tk.Frame(frame, bg='#f0f0f0')

gender\_frame.grid(row=3, column=1, padx=10, pady=10, sticky='w')

r1 = tk.Radiobutton(gender\_frame, text="Male", variable=gender\_var, value="Male",

font=('Arial', 18), bg='#f0f0f0', padx=10, indicatoron=True)

r1.pack(side='left', padx=10)

r2 = tk.Radiobutton(gender\_frame, text="Female", variable=gender\_var, value="Female",

font=('Arial', 18), bg='#f0f0f0', padx=10, indicatoron=True)

r2.pack(side='left', padx=10)

r3 = tk.Radiobutton(gender\_frame, text="Other", variable=gender\_var, value="Other",

font=('Arial', 18), bg='#f0f0f0', padx=10, indicatoron=True)

r3.pack(side='left', padx=10)

l5 = tk.Label(frame, text="PHONE", bg='#f0f0f0', font=label\_font)

l5.grid(row=4, column=0, padx=10, pady=10, sticky='e')

e5 = tk.Entry(frame, width=entry\_width, font=('Arial', 18, 'bold'), highlightthickness=1, borderwidth=2)

e5.grid(row=4, column=1, padx=10, pady=10)

from tkinter import ttk # Import ttk for Combobox

# Replace the blood group Entry widget

l6 = tk.Label(frame, text="BLOOD GROUP", bg='#f0f0f0', font=label\_font)

l6.grid(row=5, column=0, padx=10, pady=10, sticky='e')

# Dropdown for blood group

blood\_groups = ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]

e6 = ttk.Combobox(frame, values=blood\_groups, width=28, font=('Arial', 18, 'bold'))

e6.grid(row=5, column=1, padx=10, pady=10)

e6.set("Select") # Set default placeholder value

# Confirmation message label

confirmation\_label = tk.Label(root1, text="", font=('Arial', 18, 'bold'), bg='#d8e2dc', fg='green')

confirmation\_label.grid(row=3, column=0, columnspan=2, pady=10)

# Submit button

b1 = tk.Button(root1, text="SUBMIT", command=lambda: entry(e1, e2, e3, gender\_var, e5, e6, confirmation\_label, root1),

font=('Arial', 18, 'bold'), width=15, bg='#4CAF50', fg='white')

b1.grid(row=2, column=0, padx=10, pady=20)

# Close button

b2 = tk.Button(root1, text="CLOSE", command=root1.destroy, font=('Arial', 18, 'bold'), width=15, bg='#f44336', fg='white')

b2.grid(row=2, column=1, padx=10, pady=20)

# Centering buttons in the grid

root1.grid\_columnconfigure(0, weight=1)

root1.grid\_columnconfigure(1, weight=1)

root1.resizable(False, False)

root1.mainloop()

# Module 4: Appointment Management

# def apo\_details():

global x2, selected\_date\_field, x4

department = x2.get() # Department input

selected\_date = selected\_date\_field.get() # Selected date

time\_slot = x4.get() # Entered time

# Input validation

if not department or not selected\_date or not time\_slot:

messagebox.showwarning("Input Error", "All fields are required!")

return

try:

department = int(department) # Ensure department is a number

except ValueError:

messagebox.showwarning("Input Error", "Invalid department input! Please enter a number.")

return

# Assign doctors based on department

doctors = {

1: [("Dr. Sharma", "Room 10"), ("Dr. Verma", "Room 11")],

2: [("Dr. Kumar", "Room 12"), ("Dr. Khan", "Room 13")],

3: [("Dr. Anush", "Room 14"), ("Dr. Singh", "Room 15")],

4: [("Dr. Siddharth", "Room 16"), ("Dr.Yuv", "Room 17")],

5: [("Dr. Virat", "Room 18"), ("Dr. Leo", "Room 19")],

6: [("Dr. Irfan", "Room 20"), ("Dr. Srividya", "Room 21")],

7: [("Dr. Anjali", "Room 22"), ("Dr. Eesha", "Room 23")]

# Add other departments here...

}

if department not in doctors:

messagebox.showwarning("Input Error", "Invalid department number!")

return

# Randomly assign a doctor

doctor, room = rd.choice(doctors[department])

try:

# Insert appointment details into the database

query = """

INSERT INTO appointment\_details (department, doctor, date, time)

VALUES (%s, %s, %s, %s)

"""

cur.execute(query, (department, doctor, selected\_date, time\_slot))

con.commit()

# Retrieve the unique ordered appointment number

cur.execute("SELECT LAST\_INSERT\_ID()")

appointment\_no = cur.fetchone()[0]

# Display success message with appointment details

messagebox.showinfo(

"Appointment Scheduled",

f"Appointment Scheduled Successfully!\n\n"

f"Appointment Number: {appointment\_no}\n"

f"Doctor: {doctor}\n"

f"Room: {room}\n"

f"Date: {selected\_date}\n"

f"Time: {time\_slot}"

)

except sqlcon.Error as e:

messagebox.showerror("Database Error", f"An error occurred: {e}")

# def get\_apoint(aadhaar\_number):

global x2, selected\_date\_field, x4

cur.execute('SELECT \* FROM appointment WHERE idno = %s', (aadhaar\_number,))

dat = cur.fetchall()

if not dat:

messagebox.showwarning("ERROR", "NO DATA FOUND!!")

return

root3 = tk.Toplevel() # Using Toplevel for dialog behavior

root3.title("Appointment Details")

root3.configure(bg='#d8e2dc')

root3.attributes('-fullscreen', True)

label = tk.Label(root3, text="APPOINTMENT DETAILS", font='Arial 30 bold', bg='#598392', fg='white')

label.grid(row=0, column=0, columnspan=2, pady=20) # Moved up from previous position

data\_frame = tk.Frame(root3, bg='#d8e2dc')

data\_frame.grid(row=1, column=0, columnspan=2, pady=20) # Moved up from previous position

for i in dat:

title = "Mr." if i[3].upper() == 'M' else "Mrs/Ms."

tk.Label(data\_frame, text=f'WELCOME {title} {i[1]}', font='Arial 18', bg='#d8e2dc').grid(row=0, column=0, padx=10, pady=10, sticky='w')

tk.Label(data\_frame, text=f'AGE: {i[2]}', font='Arial 14', bg='#d8e2dc').grid(row=1, column=0, padx=10, pady=5, sticky='w')

tk.Label(data\_frame, text=f'PHONE: {i[4]}', font='Arial 14', bg='#d8e2dc').grid(row=2, column=0, padx=10, pady=5, sticky='w')

tk.Label(data\_frame, text=f'BLOOD GROUP: {i[5]}', font='Arial 14', bg='#d8e2dc').grid(row=3, column=0, padx=10, pady=5, sticky='w')

# Moved departments section higher

tk.Label(data\_frame, text='DEPARTMENTS', font='Arial 26 bold', bg='#598392', fg='white').grid(row=4, column=0, padx=10, pady=10, sticky='w')

departments = ["1. Orthopaedic Surgeon", "2. Physician", "3. Nephrologist",

"4. Neurologist", "5. Gynaecologist","6. Cardiologist","7. ENT Specialist"]

for index, dept in enumerate(departments):

tk.Label(data\_frame, text=dept, font='Arial 14', bg='#d8e2dc').grid(row=5 + index, column=0, padx=10, pady=5, sticky='w')

input\_frame = tk.Frame(data\_frame, bg='#d8e2dc')

input\_frame.grid(row=0, column=1, padx=20, pady=10, sticky='n')

tk.Label(input\_frame, text='Enter your choice', font='Arial 16 bold', bg='#f0f0f0').grid(row=0, column=0, padx=10, pady=10, sticky='w')

x2 = tk.Entry(input\_frame, font='Arial 14', bd=2, relief='solid', width=20)

x2.grid(row=0, column=1, padx=10, pady=10)

tk.Label(input\_frame, text='Choose date', font='Arial 16 bold', bg='#f0f0f0').grid(row=1, column=0, padx=10, pady=10, sticky='w')

selected\_date\_field = tk.Entry(input\_frame, font='Arial 14', bd=2, relief='solid', width=20)

selected\_date\_field.grid(row=1, column=1, padx=10, pady=10)

# Calendar Frame (placed directly below the input field)

calendar\_frame = tk.Frame(input\_frame, bg='#f0f0f0')

today = datetime.date.today()

# Calculate the date 6 months from today

six\_months\_later = today + datetime.timedelta(days=180) # Approx. 6 months (may vary slightly depending on the months)

calendar = Calendar(calendar\_frame, font='Arial 10', selectmode='day', year=today.year, month=today.month, day=today.day, mindate=today, maxdate=six\_months\_later)

calendar.pack(padx=5, pady=5)

calendar\_frame.grid\_remove() # Initially hide the calendar

def toggle\_calendar():

if calendar\_frame.winfo\_viewable():

calendar\_frame.grid\_remove()

else:

calendar\_frame.grid(row=2, column=1, padx=10, pady=10) # Place the calendar below the Select Date input field

# def get\_selected\_date():

selected\_date\_field.config(state='normal') # Enable writing

selected\_date\_field.delete(0, tk.END) # Clear current content

selected\_date\_field.insert(0, calendar.get\_date()) # Add the selected date

selected\_date\_field.config(state='readonly') # Disable editing

calendar\_frame.grid\_remove() # Hide the calendar after selection

tk.Button(input\_frame, text='Select Date', command=toggle\_calendar, font='Arial 14 bold', bg='#007ACC', fg='white').grid(row=1, column=2, padx=10, pady=10)

tk.Button(calendar\_frame, text='OK', command=get\_selected\_date, font='Arial 12', bg='green', fg='white').pack(pady=5)

# Time Input Field

tk.Label(input\_frame, text='Enter time (24-hour format)', font='Arial 16 bold', bg='#f0f0f0').grid(row=3, column=0, padx=10, pady=10, sticky='w')

x4 = tk.Entry(input\_frame, font='Arial 14', bd=2, relief='solid', width=20)

x4.grid(row=3, column=1, padx=10, pady=10)

button\_frame = tk.Frame(root3, bg='#d8e2dc')

button\_frame.grid(row=3, column=0, columnspan=2, pady=20) # Placed below the form input

tk.Button(button\_frame, text='Submit', command=apo\_details, font='Arial 14 bold', bg='green', fg='white').pack(side='left', padx=7)

tk.Button(button\_frame, text='Close', command=root3.destroy, font='Arial 14 bold', bg='red', fg='white').pack(side='right', padx=7)

root3.mainloop()

# def submit\_and\_close():

aadhaar\_number = x1.get()

root2.destroy()

get\_apoint(aadhaar\_number)

# def apoint():

global x1, root2

root2 = tk.Tk()

root2.title("Appointment")

root2.configure(bg='#d8e2dc')

screen\_width = root2.winfo\_screenwidth()

screen\_height = root2.winfo\_screenheight()

window\_width, window\_height = 400, 300

x = (screen\_width // 2) - (window\_width // 2)

y = (screen\_height // 2) - (window\_height // 2)

root2.geometry(f"{window\_width}x{window\_height}+{x}+{y}")

tk.Label(root2, text="APPOINTMENT", font='Arial 30 bold', bg='#598392', fg='white').pack(pady=10)

box\_frame = tk.Frame(root2, bg='#ffffff', bd=2, relief='groove', padx=20, pady=20)

box\_frame.pack(padx=20, pady=10, fill='both', expand=True)

tk.Label(box\_frame, text="AADHAAR NO.", font='Arial 14 bold', bg='#ffffff').grid(row=0, column=0, padx=10, pady=10, sticky='e')

x1 = tk.Entry(box\_frame, font='Arial 14', width=14, bd=2, relief='solid')

x1.grid(row=0, column=1, padx=10, pady=10)

tk.Button(box\_frame, text='Submit', command=submit\_and\_close, font='Arial 14 bold', bg='green', fg='white', bd=2, relief='solid').grid(row=1, column=0, columnspan=2, pady=10)

root2.resizable(False, False)

root2.mainloop()

# Module 5:List Of Doctors

# def lst\_doc():

root4 = Toplevel() # Use Toplevel if this is opened from another main window

root4.attributes("-fullscreen", True) # Set fullscreen mode

root4.configure(bg="#d8e2dc") # Set background color

# Doctor details

l = ["Dr. Sharma", "Dr. Verma", "Dr.Kumar","Dr. Khan", "Dr. Anusha", "Dr. Singh", "Dr. Siddharth",

"Dr. Yuv", "Dr. Virat", "Dr.Leo", "Dr. Irfan",

"Dr. Srividya", "Dr. Anjali", "Dr. Eesha", "Dr. Shahid"]

m = ["Orthopaedic surgeon", "Orthopaedic surgeon", "Nephrologist", "Nephrologist",

"Gynaecologist", "Gynaecologist", "Physician", "Physician", "Neurologist",

"Neurologist", "Cardiologist", "Cardiologist", "ENT Specialist", "ENT Specialist"]

n = [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]

# Create a Frame for the content

frame = Frame(root4, bg="#d8e2dc")

frame.place(relx=0.5, rely=0.5, anchor='center') # Center the frame

# Configure grid columns

frame.columnconfigure(0, weight=1, minsize=200) # Name column

frame.columnconfigure(1, weight=1, minsize=200) # Department column

frame.columnconfigure(2, weight=1, minsize=200) # Room No column

# Column headings with increased font size and colors

heading\_font = "Arial 20 bold"

header\_bg\_color = "#598392"

header\_height = 2 # Height of header boxes (in terms of grid rows)

# Create headers with borders

Label(frame, text='NAME OF DOCTORS', font=heading\_font, bg=header\_bg\_color, fg="white",

borderwidth=2, relief="groove", height=header\_height, padx=10, pady=10).grid(row=0, column=0, padx=15, pady=10, sticky='nsew')

Label(frame, text='DEPARTMENT', font=heading\_font, bg=header\_bg\_color, fg="white",

borderwidth=2, relief="groove", height=header\_height, padx=10, pady=10).grid(row=0, column=1, padx=15, pady=10, sticky='nsew')

Label(frame, text='ROOM NO', font=heading\_font, bg=header\_bg\_color, fg="white",

borderwidth=2, relief="groove", height=header\_height, padx=10, pady=10).grid(row=0, column=2, padx=15, pady=10, sticky='nsew')

# Doctor information list with increased font size

info\_font = "Arial 16"

for i, (name, dept, room) in enumerate(zip(l, m, n), start=1):

Label(frame, text=name, font=info\_font, bg="#d8e2dc", borderwidth=2, relief="groove").grid(row=i, column=0, padx=10, pady=5, sticky='nsew')

Label(frame, text=dept, font=info\_font, bg="#d8e2dc", borderwidth=2, relief="groove").grid(row=i, column=1, padx=10, pady=5, sticky='nsew')

Label(frame, text=room, font=info\_font, bg="#d8e2dc", borderwidth=2, relief="groove").grid(row=i, column=2, padx=10, pady=5, sticky='nsew')

# Add a "Close" button to close the window, centered below the table

close\_button = Button(root4, text="Close", command=root4.destroy, font="Arial 16", bg="red", fg="white")

close\_button.place(relx=0.5, rely=0.95, anchor='center') # Center button below the table

root4.resizable(False, False)

root4.mainloop()

# Module 6 :Services Available

# def ser\_avail():

root5 = Toplevel() # Use Toplevel to create a new window

root5.attributes("-fullscreen", True) # Set fullscreen mode

root5.configure(bg="#d8e2dc") # Set background color

# Create a Frame for the content

frame = Frame(root5, bg="#d8e2dc")

frame.place(relx=0.5, rely=0.5, anchor='center') # Center the frame

# Configure grid columns

frame.columnconfigure(0, weight=1, minsize=200) # Services column

frame.columnconfigure(1, weight=1, minsize=200) # Room No column

# Title label with a box around it

title\_font = "Arial 30 bold" # Increased font size

title\_bg\_color = "#598392" # Box background color

title\_label = Label(frame, text='SERVICES AVAILABLE', font=title\_font, bg=title\_bg\_color, fg="white",

borderwidth=2, relief="groove", width=30, height=2) # Increased width and height

title\_label.grid(row=0, column=0, columnspan=2, padx=15, pady=10)

# Services list

services = ["ULTRASOUND", "X-RAY", "CT Scan", "MRI", "BLOOD COLLECTION",

"DIALYSIS", "ECG", "CHEMIST", "LAB"]

# Room numbers

room\_numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Populate the services and room numbers

for i, service in enumerate(services):

Label(frame, text=service, font="Arial 16", bg="#d8e2dc", borderwidth=2, relief="groove").grid(row=i + 1, column=0, padx=10, pady=5, sticky='nsew')

Label(frame, text=room\_numbers[i], font="Arial 16", bg="#d8e2dc", borderwidth=2, relief="groove").grid(row=i + 1, column=1, padx=10, pady=5, sticky='nsew')

# Contact information label

contact\_label = Label(frame, text='To avail any of these please contact on our no.:- 7042\*\*\*\*55',

font="Arial 14", bg="#d8e2dc")

contact\_label.grid(row=len(services) + 1, column=0, columnspan=2, padx=15, pady=20)

# Add a "Close" button to close the window, centered below the table

close\_button = Button(root5, text="Close", command=root5.destroy, font="Arial 16", bg="red", fg="white")

close\_button.place(relx=0.5, rely=0.9, anchor='center') # Center button below the content

root5.resizable(False, False)

root5.mainloop()

# def modify(aadhaar\_no):

global x4, x5, choice, root6, success\_label

p1 = aadhaar\_no

cur.execute('select \* from appointment where idno=(%s)', (p1,))

dat = cur.fetchall()

a = [i for i in dat]

if len(a) == 0:

messagebox.showwarning("ERROR", "NO DATA FOUND!!")

else:

# Open the modification window in fullscreen

root6 = tk.Toplevel(root)

root6.title("Modify Details")

root6.attributes('-fullscreen', True)

root6.configure(bg='#d8e2dc')

# Title styling to match "Modify Appointment Details"

title\_label = tk.Label(root6, text='MODIFY DETAILS', font="Arial 25 bold", bg='#598392', fg='white')

title\_label.pack(pady=20)

# Current Details frame

details\_frame = tk.Frame(root6, bg='#ffffff', bd=2, relief='groove', padx=20, pady=20)

details\_frame.pack(padx=20, pady=10, fill='both', expand=True)

# Display current details

for i in dat:

labels = {

'NAME': i[1],

'AGE': i[2],

'GENDER': i[3],

'PHONE': i[4],

'BLOOD GROUP': i[5]

}

y\_pos = 20

for key, value in labels.items():

tk.Label(details\_frame, text=f'{key}:', font="Arial 14 bold", bg='#ffffff').grid(row=y\_pos, column=0, padx=10, pady=5, sticky='w')

tk.Label(details\_frame, text=value, font="Arial 14", bg='#ffffff').grid(row=y\_pos, column=1, padx=10, pady=5, sticky='w')

y\_pos += 1

# Modify options

options = ["1. NAME", "2. AGE", "3. GENDER", "4. PHONE", "5. BLOOD GROUP"]

y\_pos += 2

for option in options:

tk.Label(details\_frame, text=option, font="Arial 14", bg='#ffffff').grid(row=y\_pos, column=0, sticky='w')

y\_pos += 1

# Choice input

tk.Label(details\_frame, text='Enter Choice:', font="Arial 14 bold", bg='#ffffff').grid(row=y\_pos, column=0, pady=10, sticky='w')

x4 = tk.Entry(details\_frame, font="Arial 14", bd=2, relief='solid')

x4.grid(row=y\_pos, column=1, padx=10, pady=10)

# New value input

tk.Label(details\_frame, text='Enter New Detail:', font="Arial 14 bold", bg='#ffffff').grid(row=y\_pos + 1, column=0, pady=10, sticky='w')

x5 = tk.Entry(details\_frame, font="Arial 14", bd=2, relief='solid')

x5.grid(row=y\_pos + 1, column=1, padx=10, pady=10)

# Success message label

success\_label = tk.Label(root6, text="", font="Arial 14", bg='#d8e2dc', fg='green')

success\_label.pack(pady=20)

# Submit and Close buttons

button\_frame = tk.Frame(root6, bg='#d8e2dc')

button\_frame.pack(pady=10)

tk.Button(button\_frame, text='Submit', command=do\_modify, font='Arial 14 bold', bg='green', fg='white').pack(side='left', padx=10)

tk.Button(button\_frame, text='Close', command=root6.destroy, font='Arial 14 bold', bg='red', fg='white').pack(side='left')

root6.mainloop()

# Module 7 :Modify Existing Data

def do\_modify():

ad = x3.get()

choice = x4.get()

new = x5.get()

# Update the database based on the choice

field\_mapping = {

'1': 'name',

'2': 'age',

'3': 'gender',

'4': 'phone',

'5': 'bg'

}

if choice in field\_mapping:

cur.execute(f'UPDATE appointment SET {field\_mapping[choice]} = %s WHERE idno = %s', (new, ad))

con.commit()

# Update the success message in the same window

success\_label.config(text="The data got modified successfully!")

else:

success\_label.config(text="Invalid choice. Please try again.", fg='red')

# def mod\_sub():

global x3,root

root7 = tk.Toplevel(root)

root7.title("Modification - Aadhaar Entry")

root7.configure(bg='#d8e2dc') # Match the background color

# Center window on screen

screen\_width = root7.winfo\_screenwidth()

screen\_height = root7.winfo\_screenheight()

window\_width, window\_height = 400, 300

x = (screen\_width // 2) - (window\_width // 2)

y = (screen\_height // 2) - (window\_height // 2)

root7.geometry(f"{window\_width}x{window\_height}+{x}+{y}")

# Title with similar styling as the appointment Aadhaar window

tk.Label(root7, text="MODIFY DETAILS", font='Arial 30 bold', bg='#598392', fg='white').pack(pady=10)

# Input frame styled like appointment Aadhaar entry

box\_frame = tk.Frame(root7, bg='#ffffff', bd=2, relief='groove', padx=20, pady=20)

box\_frame.pack(padx=20, pady=10, fill='both', expand=True)

# Aadhaar input label and entry

tk.Label(box\_frame, text="AADHAAR NO.", font='Arial 14 bold', bg='#ffffff').grid(row=0, column=0, padx=10, pady=10, sticky='e')

x3 = tk.Entry(box\_frame, font='Arial 14', width=14, bd=2, relief='solid')

x3.grid(row=0, column=1, padx=10, pady=10)

# Submit button with lambda to close root7 and open modify function in fullscreen

tk.Button(box\_frame, text='Submit', command=lambda: submit\_aadhaar(root7), font='Arial 14 bold', bg='green', fg='white', bd=2, relief='solid').grid(row=1, column=0, columnspan=2, pady=10)

root7.resizable(False, False)

root7.mainloop()

# def submit\_aadhaar(window):

aadhaar\_number = x3.get() # Get Aadhaar number

if aadhaar\_number: # Ensure the Aadhaar number is not empty

modify(aadhaar\_number) # Call modify function with Aadhaar number

window.destroy() # Close the Aadhaar entry window

else:

messagebox.showwarning("Input Error", "Please enter a valid Aadhaar number.") # Show a warning if empty

# Module 8:View Data

def search\_data():

global x3,root7,root

root7 = tk.Toplevel(root)

root7.title("Search Data")

root7.configure(bg='#d8e2dc') # Set background color

# Center window on screen

screen\_width = root7.winfo\_screenwidth()

screen\_height = root7.winfo\_screenheight()

window\_width, window\_height = 400, 300

x = (screen\_width // 2) - (window\_width // 2)

y = (screen\_height // 2) - (window\_height // 2)

root7.geometry(f"{window\_width}x{window\_height}+{x}+{y}")

# Title with adjusted styling

tk.Label(root7, text="SEARCH DATA", font='Arial 30 bold', bg='#598392', fg='white').pack(pady=10)

# Input frame

box\_frame = tk.Frame(root7, bg='#ffffff', bd=2, relief='groove', padx=20, pady=20)

box\_frame.pack(padx=20, pady=10, fill='both', expand=True)

# Aadhaar input label and entry

tk.Label(box\_frame, text="AADHAAR NO.", font='Arial 14 bold', bg='#ffffff').grid(row=0, column=0, padx=10, pady=10, sticky='e')

x3 = tk.Entry(box\_frame, font='Arial 14', width=14, bd=2, relief='solid')

x3.grid(row=0, column=1, padx=10, pady=10)

# Submit button

tk.Button(box\_frame, text='Submit', command=view\_data, font='Arial 14 bold', bg='green', fg='white', bd=2, relief='solid').grid(row=1, column=0, columnspan=2, pady=10)

root7.resizable(False, False)

root7.mainloop()

# def view\_data():

global p1,root

p1 = x3.get()

cur.execute('SELECT \* FROM appointment WHERE idno = %s', (p1,))

dat = cur.fetchall()

if not dat:

tkinter.messagebox.showwarning("ERROR", "NO DATA FOUND!!")

else:

# Open a fullscreen window to display the details

root\_fullscreen = tk.Toplevel(root)

root\_fullscreen.title("Patient Details")

root\_fullscreen.attributes('-fullscreen', True)

root\_fullscreen.configure(bg='#d8e2dc')

# Title label

tk.Label(root\_fullscreen, text="PATIENT DETAILS", font='Arial 30 bold', bg='#598392', fg='white').pack(pady=20)

# Create a frame for patient details

detail\_frame = tk.Frame(root\_fullscreen, bg='#d8e2dc')

detail\_frame.pack(pady=20)

# Create labels for patient details with boxes

details = [('NAME OF THE PATIENT', f'Mr. {dat[0][1]}' if dat[0][3].upper() == 'M' else f'Mrs/Ms. {dat[0][1]}'),

('AGE', dat[0][2]),

('PHONE', dat[0][4]),

('BLOOD GROUP', dat[0][5])]

for index, (heading, value) in enumerate(details):

# Heading label

tk.Label(detail\_frame, text=heading, font='Arial 14 bold', bg='#d8e2dc').grid(row=index, column=0, padx=10, pady=5, sticky='w')

# Value label inside a box

value\_label = tk.Label(detail\_frame, text=value, font='Arial 14', bg='white', bd=2, relief='solid', width=30)

value\_label.grid(row=index, column=1, padx=10, pady=5, sticky='w')

# Close button for the fullscreen window

close\_button = tk.Button(root\_fullscreen, text='Close', command=root\_fullscreen.destroy, font='Arial 14 bold', bg='red', fg='white')

close\_button.pack(pady=20)

root7.destroy() # Close the search window after submission

# Main window setup

# Module 9: Prescription Details

# def prescription\_details():

global root

global x\_aadhaar # Global variable for Aadhaar input

prescription\_window = tk.Toplevel(root)

prescription\_window.title("Prescription Details")

prescription\_window.configure(bg='#d8e2dc')

# Set window size (same size as Modify Details)

window\_width, window\_height = 400, 300

screen\_width = prescription\_window.winfo\_screenwidth()

screen\_height = prescription\_window.winfo\_screenheight()

x = (screen\_width // 2) - (window\_width // 2)

y = (screen\_height // 2) - (window\_height // 2)

prescription\_window.geometry(f"{window\_width}x{window\_height}+{x}+{y}")

# Title with similar styling as the Modify Details Aadhaar window

tk.Label(prescription\_window, text="PRESCRIPTION DETAILS", font='Arial 23 bold', bg='#598392', fg='white').pack(pady=10)

# Input frame styled like Modify Details Aadhaar entry

box\_frame = tk.Frame(prescription\_window, bg='#ffffff', bd=2, relief='groove', padx=20, pady=20)

box\_frame.pack(padx=20, pady=10, fill='both', expand=True)

# Aadhaar input label and entry (same as Modify Details)

tk.Label(box\_frame, text="AADHAAR NO.", font='Arial 14 bold', bg='#ffffff').grid(row=0, column=0, padx=10, pady=10, sticky='e')

# Aadhaar Entry box with the same style as Modify Details

x\_aadhaar = tk.Entry(box\_frame, font='Arial 14', width=14, bd=2, relief='solid')

x\_aadhaar.grid(row=0, column=1, padx=10, pady=10)

# Label for displaying messages

message\_label = tk.Label(box\_frame, text="", font='Arial 12', bg='#ffffff', fg='red')

message\_label.grid(row=2, column=0, columnspan=2, pady=10)

# Submit button that calls the submit\_aadhaar function and updates the message label

tk.Button(prescription\_window, text='Submit', command=lambda: open\_prescription\_window(prescription\_window), font='Arial 14 bold', bg='green', fg='white').pack(pady=10)

prescription\_window.resizable(False, False)

prescription\_window.mainloop()

# Function to open the prescription details window

# def open\_prescription\_window(aadhaar\_window):

aadhaar\_number = x\_aadhaar.get()

if not aadhaar\_number:

messagebox.showwarning("Input Error", "Please enter an Aadhaar ID.")

return

# Fetch patient details from the database

cur.execute('SELECT \* FROM appointment WHERE idno = %s', (aadhaar\_number,))

patient\_data = cur.fetchall()

if not patient\_data:

messagebox.showwarning("Error", "No patient data found for this Aadhaar ID.")

return

# Close the Aadhaar window before opening the prescription details window

aadhaar\_window.destroy()

# Create the fullscreen window for prescription

prescription\_window = tk.Tk()

prescription\_window.title("Prescription Details")

prescription\_window.attributes('-fullscreen', True)

tk.Label(prescription\_window, text="Prescription Details", font='Arial 30 bold', bg='#598392', fg='white').pack(pady=20)

patient\_frame = tk.Frame(prescription\_window, bg='#d8e2dc')

patient\_frame.pack(pady=20)

for patient in patient\_data:

title = "Mr." if patient[3].upper() == 'M' else "Mrs/Ms."

tk.Label(patient\_frame, text=f'WELCOME {title} {patient[1]}', font='Arial 18', bg='#d8e2dc').pack()

# Input fields for prescription

input\_frame = tk.Frame(prescription\_window, bg='#d8e2dc')

input\_frame.pack(pady=20)

# Date of Issue Label and Entry aligned left and right

tk.Label(input\_frame, text='Date of Issue', font='Arial 16 bold', bg='#d8e2dc').grid(row=0, column=0, padx=10, pady=10, sticky="w")

date\_of\_issue\_entry = tk.Entry(input\_frame, font='Arial 14', bd=2, relief='solid', width=30)

date\_of\_issue\_entry.grid(row=0, column=1, padx=10, pady=10, sticky="e")

# Select button to toggle calendar visibility

# 

# def toggle\_calendar(calendar\_frame):

# Toggle visibility of the calendar frame

if calendar\_frame.winfo\_ismapped():

calendar\_frame.grid\_forget()

else:

calendar\_frame.grid(row=1, column=0, columnspan=2, pady=10)

# Button to show the calendar

select\_button = tk.Button(input\_frame, text="Select", font='Arial 14', bg='#7cb5e0', fg='white', command=lambda: toggle\_calendar(calendar\_frame))

select\_button.grid(row=0, column=2, padx=10, pady=10)

# Calendar frame (initially hidden)

calendar\_frame = tk.Frame(input\_frame, bg='#ffffff')

cal = Calendar(calendar\_frame, selectmode='day', date\_pattern='yyyy-mm-dd')

cal.pack(pady=10)

cal.bind('<<CalendarSelected>>', lambda event: date\_of\_issue\_entry.insert(0, cal.get\_date()) or toggle\_calendar(calendar\_frame)) # Insert selected date and hide calendar

# List to hold tablet entries

tablet\_entries = []

# def add\_tablet\_entry():

row = len(tablet\_entries) + 1 # Start from row 1 for tablets

# Set equal column weight to make sure everything stretches evenly

for col in range(8): # Adjust the number of columns if necessary

input\_frame.grid\_columnconfigure(col, weight=1, uniform="equal")

# Label for Tablet Name

tk.Label(input\_frame, text='Tablet Name', font='Arial 16 bold', bg='#d8e2dc').grid(row=row, column=0, padx=10, pady=10, sticky='w')

# Tablet Name Entry

tablet\_name\_entry = tk.Entry(input\_frame, font='Arial 12', bd=2, relief='solid', width=25)

tablet\_name\_entry.grid(row=row, column=1, padx=10, pady=10, sticky='w')

# Label for Daily Dose

tk.Label(input\_frame, text='Daily Dose', font='Arial 16 bold', bg='#d8e2dc').grid(row=row, column=2, padx=10, pady=10, sticky='w')

# Daily Dose Entry

daily\_dose\_entry = tk.Entry(input\_frame, font='Arial 14', bd=2, relief='solid', width=25)

daily\_dose\_entry.grid(row=row, column=3, padx=10, pady=10, sticky='w')

# Label for Expiry Date

tk.Label(input\_frame, text='Expiry Date', font='Arial 16 bold', bg='#d8e2dc').grid(row=row, column=4, padx=10, pady=10, sticky='w')

# Expiry Date Entry

expiry\_date\_entry = tk.Entry(input\_frame, font='Arial 14', bd=2, relief='solid', width=25)

expiry\_date\_entry.grid(row=row, column=5, padx=10, pady=10, sticky='w')

# Label for No of Tablets

tk.Label(input\_frame, text='No of Tablets', font='Arial 16 bold', bg='#d8e2dc').grid(row=row, column=6, padx=10, pady=10, sticky='w')

# No of Tablets Entry

no\_of\_tablets\_entry = tk.Entry(input\_frame, font='Arial 14', bd=2, relief='solid', width=25)

no\_of\_tablets\_entry.grid(row=row, column=7, padx=10, pady=10, sticky='w')

# Add the entries to the list for saving later

tablet\_entries.append((tablet\_name\_entry, daily\_dose\_entry, expiry\_date\_entry, no\_of\_tablets\_entry))

# Button to add tablet entries

tk.Button(prescription\_window, text='Add Tablet', command=add\_tablet\_entry, font='Arial 14 bold', bg='blue', fg='white').pack(pady=10)

# Save prescription button

tk.Button(prescription\_window, text='Save Prescription', command=lambda: save\_prescription(aadhaar\_number, date\_of\_issue\_entry.get(), tablet\_entries, prescription\_window), font='Arial 14 bold', bg='green', fg='white').pack(pady=20)

# Close button to exit the fullscreen window

tk.Button(prescription\_window, text='Close', command=prescription\_window.destroy, font='Arial 14 bold', bg='red', fg='white').pack(pady=20)

prescription\_window.resizable(False, False)

prescription\_window.mainloop()

import datetime

# Display the success message in the prescription window

# def save\_prescription(aadhaar\_number, date\_of\_issue, tablet\_entries, prescription\_window):

# Validate that the date\_of\_issue is in the correct string format (YYYY-MM-DD)

try:

datetime.datetime.strptime(date\_of\_issue, '%Y-%m-%d')

except ValueError:

messagebox.showwarning("Input Error", "Please enter the Date of Issue in the correct format (YYYY-MM-DD).")

return

# Validate all tablet fields

for tablet in tablet\_entries:

tablet\_name, daily\_dose, expiry\_date, no\_of\_tablets = tablet

if not (tablet\_name.get() and daily\_dose.get() and expiry\_date.get() and no\_of\_tablets.get()):

messagebox.showwarning("Input Error", "Please fill in all fields for each tablet.")

return

# Validate expiry date format

try:

# Attempt to parse the expiry date to ensure it's in the correct format

datetime.datetime.strptime(expiry\_date.get(), '%Y-%m-%d')

except ValueError:

messagebox.showwarning("Input Error", "Please enter the Expiry Date in the correct format (YYYY-MM-DD).")

return

# Insert prescription details into the database

query = '''INSERT INTO prescription (idno, date\_of\_issue, tablet\_name, daily\_dose, expiry\_date, no\_of\_tablets)

VALUES (%s, %s, %s, %s, %s, %s)'''

values = (aadhaar\_number, date\_of\_issue, tablet\_name.get(), daily\_dose.get(), expiry\_date.get(), no\_of\_tablets.get())

try:

cur.execute(query, values)

con.commit() # Commit the transaction

except sqlcon.Error as err:

messagebox.showerror("Database Error", f"Error: {err}")

return

# If everything is successful, show the success message in the prescription window

success\_label = tk.Label(

prescription\_window, # Use prescription\_window for displaying the success message

text="Prescription saved successfully!",

font='Arial 16 bold',

bg='#d8e2dc',

fg='green'

)

success\_label.pack(pady=10)

# Module 10 :View Precription

# def view\_prescription():

global root

global x\_aadhaar # Global variable for Aadhaar input

view\_window = tk.Toplevel(root)

view\_window.title("View Prescription")

view\_window.configure(bg='#d8e2dc')

# Set window size (same size as Modify Details)

window\_width, window\_height = 400, 300

screen\_width = view\_window.winfo\_screenwidth()

screen\_height = view\_window.winfo\_screenheight()

x = (screen\_width // 2) - (window\_width // 2)

y = (screen\_height // 2) - (window\_height // 2)

view\_window.geometry(f"{window\_width}x{window\_height}+{x}+{y}")

# Title with similar styling as the Modify Details Aadhaar window

tk.Label(view\_window, text="VIEW PRESCRIPTION", font='Arial 23 bold', bg='#598392', fg='white').pack(pady=10)

# Input frame styled like Modify Details Aadhaar entry

box\_frame = tk.Frame(view\_window, bg='#ffffff', bd=2, relief='groove', padx=20, pady=20)

box\_frame.pack(padx=20, pady=10, fill='both', expand=True)

# Aadhaar input label and entry (same as Modify Details)

tk.Label(box\_frame, text="AADHAAR NO.", font='Arial 14 bold', bg='#ffffff').grid(row=0, column=0, padx=10, pady=10, sticky='e')

# Aadhaar Entry box with the same style as Modify Details

x\_aadhaar = tk.Entry(box\_frame, font='Arial 14', width=14, bd=2, relief='solid')

x\_aadhaar.grid(row=0, column=1, padx=10, pady=10)

# Label for displaying messages

message\_label = tk.Label(box\_frame, text="", font='Arial 12', bg='#ffffff', fg='red')

message\_label.grid(row=2, column=0, columnspan=2, pady=10)

# Submit button that calls the submit\_aadhaar function and updates the message label

tk.Button(view\_window, text='Submit', command=lambda: open\_view\_prescription\_window(view\_window), font='Arial 14 bold', bg='green', fg='white').pack(pady=10)

view\_window.resizable(False, False)

view\_window.mainloop()

# Function to open the prescription details window and fetch prescription data

# def open\_view\_prescription\_window(aadhaar\_window):

aadhaar\_number = x\_aadhaar.get()

if not aadhaar\_number:

messagebox.showwarning("Input Error", "Please enter an Aadhaar ID.")

return

# Fetch prescription details from the database

cur.execute('''SELECT tablet\_name, daily\_dose, expiry\_date, no\_of\_tablets

FROM prescription

WHERE idno = %s

ORDER BY modification\_date DESC LIMIT 1''', (aadhaar\_number,))

prescription\_data = cur.fetchall()

if not prescription\_data:

messagebox.showwarning("Error", "No prescription data found for this Aadhaar ID.")

return

# Close the Aadhaar window before opening the prescription details window

aadhaar\_window.destroy()

# Create the fullscreen window for prescription

prescription\_window = tk.Tk()

prescription\_window.title("Prescription Details")

prescription\_window.attributes('-fullscreen', True)

tk.Label(prescription\_window, text="Prescription Details", font='Arial 30 bold', bg='#598392', fg='white').pack(pady=20)

# Prescription display frame

prescription\_frame = tk.Frame(prescription\_window, bg='#d8e2dc')

prescription\_frame.pack(pady=20)

# Headers for the prescription details

tk.Label(prescription\_frame, text="Tablet Name", font='Arial 14 bold', bg='#d8e2dc', relief="solid", width=20, height=2).grid(row=0, column=0, padx=5, pady=5)

tk.Label(prescription\_frame, text="Daily Dose", font='Arial 14 bold', bg='#d8e2dc', relief="solid", width=20, height=2).grid(row=0, column=1, padx=5, pady=5)

tk.Label(prescription\_frame, text="Expiry Date", font='Arial 14 bold', bg='#d8e2dc', relief="solid", width=20, height=2).grid(row=0, column=2, padx=5, pady=5)

tk.Label(prescription\_frame, text="No of Tablets", font='Arial 14 bold', bg='#d8e2dc', relief="solid", width=20, height=2).grid(row=0, column=3, padx=5, pady=5)

# Display the prescription details in the rows

row = 1

for tablet in prescription\_data:

tk.Label(prescription\_frame, text=tablet[0], font='Arial 12', bg='#d8e2dc', relief="solid", width=20, height=2).grid(row=row, column=0, padx=5, pady=5)

tk.Label(prescription\_frame, text=tablet[1], font='Arial 12', bg='#d8e2dc', relief="solid", width=20, height=2).grid(row=row, column=1, padx=5, pady=5)

tk.Label(prescription\_frame, text=tablet[2], font='Arial 12', bg='#d8e2dc', relief="solid", width=20, height=2).grid(row=row, column=2, padx=5, pady=5)

tk.Label(prescription\_frame, text=tablet[3], font='Arial 12', bg='#d8e2dc', relief="solid", width=20, height=2).grid(row=row, column=3, padx=5, pady=5)

row += 1

# Close button to exit the fullscreen window

tk.Button(prescription\_window, text='Close', command=prescription\_window.destroy, font='Arial 14 bold', bg='red', fg='white').pack(pady=20)

prescription\_window.resizable(False, False)

# Module 11 : Main Dashboard

# Main window code

# Initialize the main window

DEFAULT\_USERNAME = "admin"

DEFAULT\_PASSWORD = "password123"

root = None

# def show\_main\_window():

global root

# Validate username and password

username = username\_entry.get()

password = password\_entry.get()

if username == DEFAULT\_USERNAME and password == DEFAULT\_PASSWORD:

login\_window.destroy() # Close the login window

root = Tk()

root.attributes('-fullscreen', True) # Set the window to fullscreen mode

root.title("Care Connect")

# Create a frame for the background color

background\_frame = Frame(root, bg='#e3f2fd') # Light blue background

background\_frame.pack(fill=BOTH, expand=True)

# Load the image

image\_path = "C:\\Users\\Dell\\Downloads\\Screenshot 2024-11-03 202101.png" # Update this path if necessary

header\_image = Image.open(image\_path)

header\_image = header\_image.resize((800, 200), Image.LANCZOS) # Resize if needed

header\_photo = ImageTk.PhotoImage(header\_image)

# Display the image as a label

header\_label = Label(background\_frame, image=header\_photo, bg='#e3f2fd') # Set the same background color

header\_label.image = header\_photo # Keep a reference to avoid garbage collection

header\_label.pack(pady=20)

# Create a frame for buttons with the same background color

button\_frame = Frame(background\_frame, bg='#e3f2fd') # Ensure the button frame matches the background

button\_frame.pack(pady=20, fill=BOTH, expand=True)

# Button setup

buttons = [

("Registration", register),

("Appointment", apoint),

("List of Doctors", lst\_doc),

("Services Available", ser\_avail),

("Modify Existing Data", mod\_sub),

("View Data", search\_data),

("Prescription Details", prescription\_details),

("View Prescription", view\_prescription),

("Exit", root.destroy)

]

layout = [

(0, 0), (0, 1), (0, 2),

(1, 0), (1, 1), (1, 2),

(2, 0), (2, 1), (2, 2)

]

for (i, (text, command)) in enumerate(buttons):

row, col = layout[i]

button = Button(

button\_frame,

text=text,

font="arial 22 bold",

bg='#ffe5ec',

fg='#000000',

width=18,

height=2,

command=command,

bd=4,

highlightbackground='#a40182',

highlightthickness=2

)

button.grid(row=row, column=col, padx=20, pady=20, sticky="nsew")

for i in range(3):

button\_frame.grid\_rowconfigure(i, weight=1)

for j in range(3):

button\_frame.grid\_columnconfigure(j, weight=1)

root.resizable(False, False)

root.mainloop()

else:

# Show error message

error\_label.config(text="Invalid username or password!", fg="red")

login\_window = Tk()

login\_window.attributes('-fullscreen', True) # Set to fullscreen

login\_window.title("Login - Care Connect")

login\_window.configure(bg='#e3f2fd') # Match the main window background

# Login frame

login\_frame = Frame(login\_window, bg='#e3f2fd')

login\_frame.place(relx=0.5, rely=0.5, anchor=CENTER)

# Username label and entry

username\_label = Label(login\_frame, text="Username", font="arial 16", bg='#e3f2fd', fg='black')

username\_label.grid(row=0, column=0, pady=10, padx=10, sticky="w")

username\_entry = Entry(login\_frame, font="arial 16", width=25)

username\_entry.grid(row=0, column=1, pady=10, padx=10)

# Password label and entry

password\_label = Label(login\_frame, text="Password", font="arial 16", bg='#e3f2fd', fg='black')

password\_label.grid(row=1, column=0, pady=10, padx=10, sticky="w")

password\_entry = Entry(login\_frame, font="arial 16", width=25, show="\*")

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

error\_label = Label(login\_frame, text="", font="arial 12", bg='#e3f2fd', fg='red')

error\_label.grid(row=2, column=0, columnspan=2)

buttons\_frame = Frame(login\_frame, bg='#e3f2fd')

buttons\_frame.grid(row=3, column=0, columnspan=2, pady=20)

login\_button = Button(

buttons\_frame,

text="Login",

font="arial 16 bold",

bg='#a40182',

fg='white',

command=show\_main\_window # Trigger the main window

)

login\_button.pack(side=LEFT, padx=10)

close\_button = Button(

buttons\_frame,

text="Close",

font="arial 16 bold",

bg='red',

fg='white',

command=login\_window.destroy # Close the login window

)

close\_button.pack(side=LEFT, padx=10)

login\_window.mainloop()

# II.Specific Algorithms/Logic to Highlight

# Aadhaar and Phone Validation Logic

Logic to ensure the Aadhaar number is exactly 12 digits and the phone number is 10 digits, preventing invalid entries.

# Doctor Assignment Algorithm

A random assignment algorithm that assigns doctors based on the selected department using a predefined list of doctors.

# Input Validation Workflow

A systematic validation process for fields like age, blood group, and gender to ensure all entered data is accurate and standardized.

# Appointment Scheduling Logic

Ensures selected appointment dates and times are valid and within a permissible range (e.g., no past dates).

# Database Integration Logic

Uses MySQL queries to synchronize GUI actions with the database, ensuring real-time data storage and retrieval.

# Prescription Management Logic

# 

Captures details like tablet name, dose, and expiry date, and allows staff to view or update prescriptions seamlessly.

View Data Workflow

Combines patient and appointment details in a structured format using SQL JOIN queries and displays them in the GUI.

# Error Handling and Alerts

Includes logic for displaying error messages for invalid input and success pop-ups for completed operations.

GitHub link :

<https://github.com/Haritha-7172/CareConnect>

Folder Structure:

**CareConnect**/

├── main.py # Entry point for the application (starts login and main window)

├── database/

│ ├── connection.py # Handles database connection

│ ├── tables.py # Creates required tables if not present

│ ├── queries.py # Contains SQL query functions

├── modules/

│ ├── registration.py # Code for registration functionality

│ ├── appointments.py # Code for appointment-related operations

│ ├── view\_modify.py # Code for viewing and modifying data

│ ├── prescription.py # Code for prescription-related functionalities

│ ├── list\_services.py # Displays doctors and services available

├── utils/

│ ├── validations.py # Validates inputs (e.g., Aadhaar, phone, etc.)

│ ├── gui\_helpers.py # Helper functions for creating GUI components

│ ├── styles.py # Centralized styling configurations

├── static/

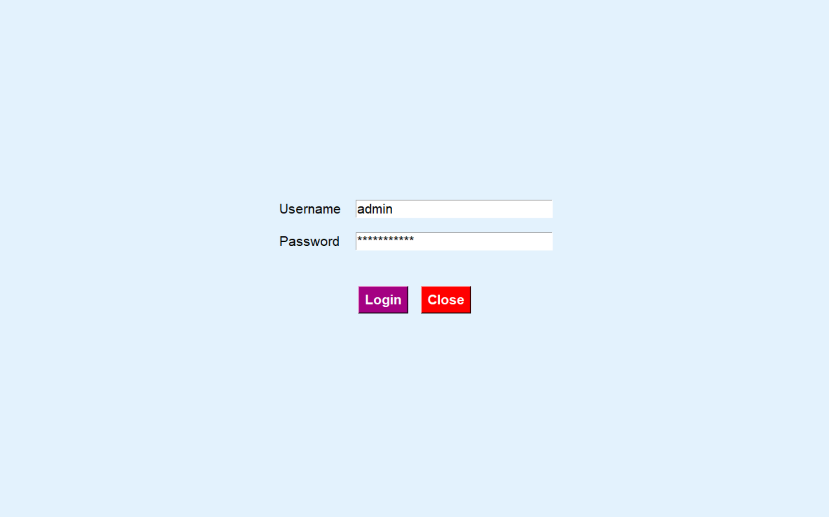
│ ├── images/ # Placeholder for all images

└── default.png # Example image for the header

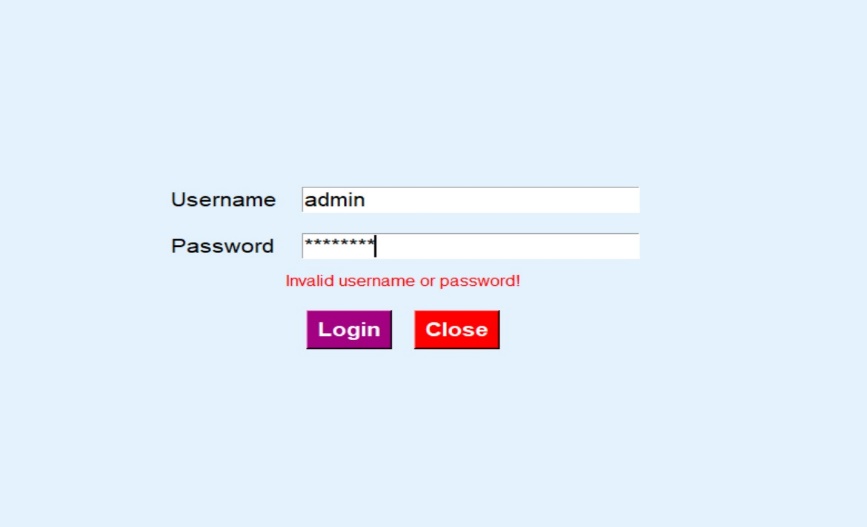
**Results:**

**Sample Outputs of the project:**

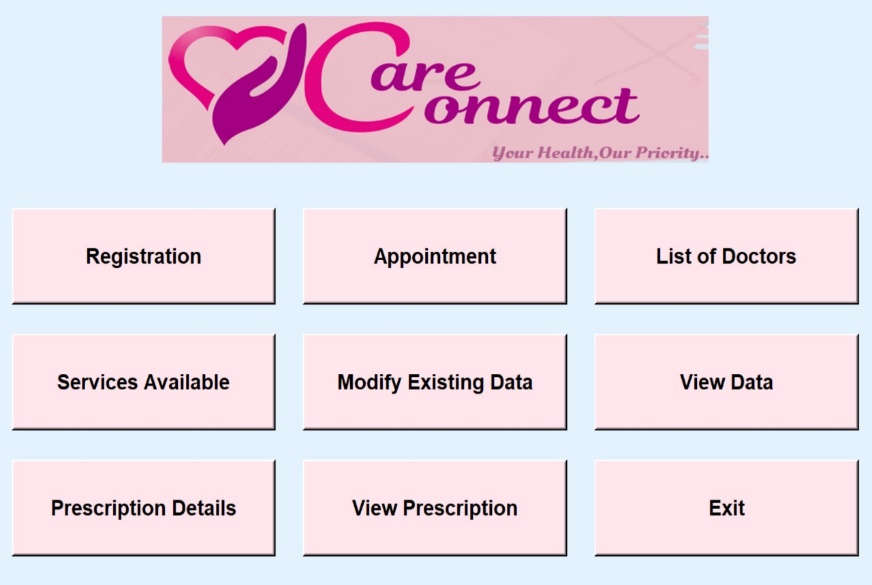
**Login window sample output:**

****

**Invalid Login:**

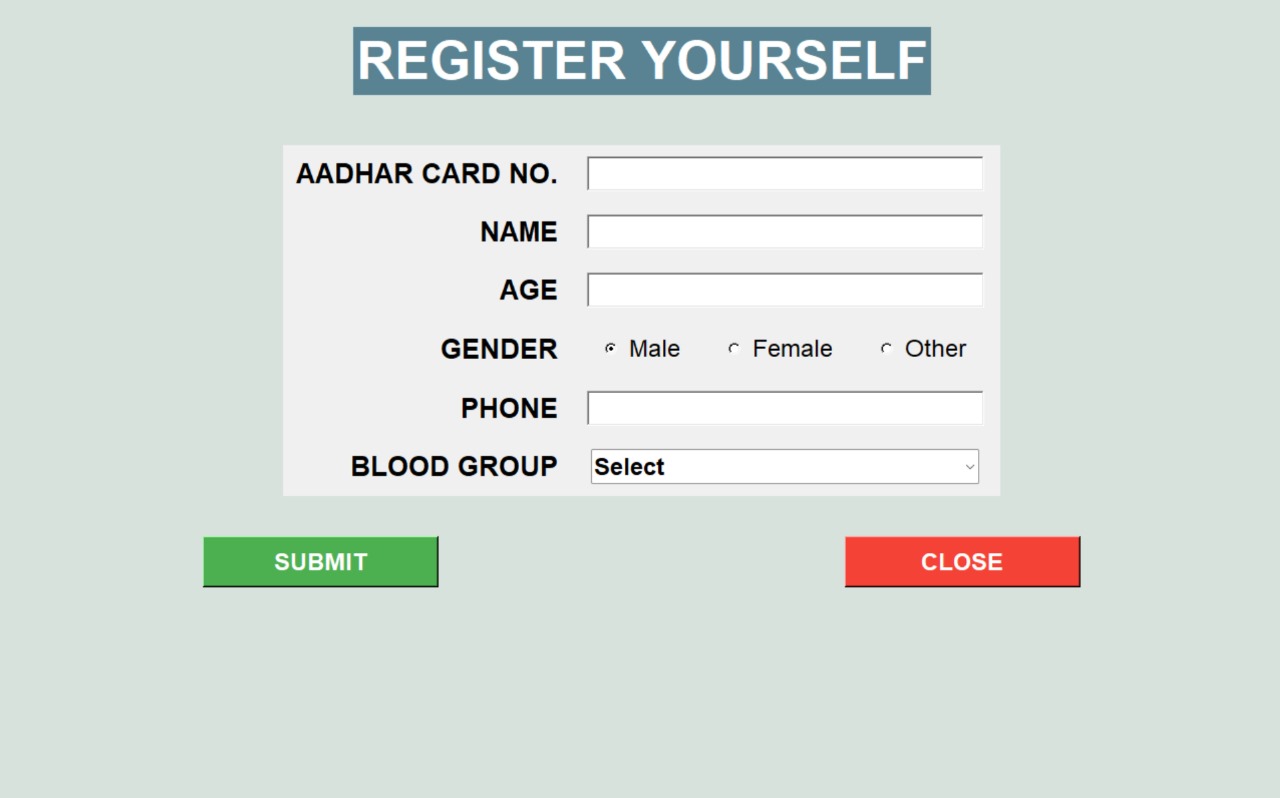


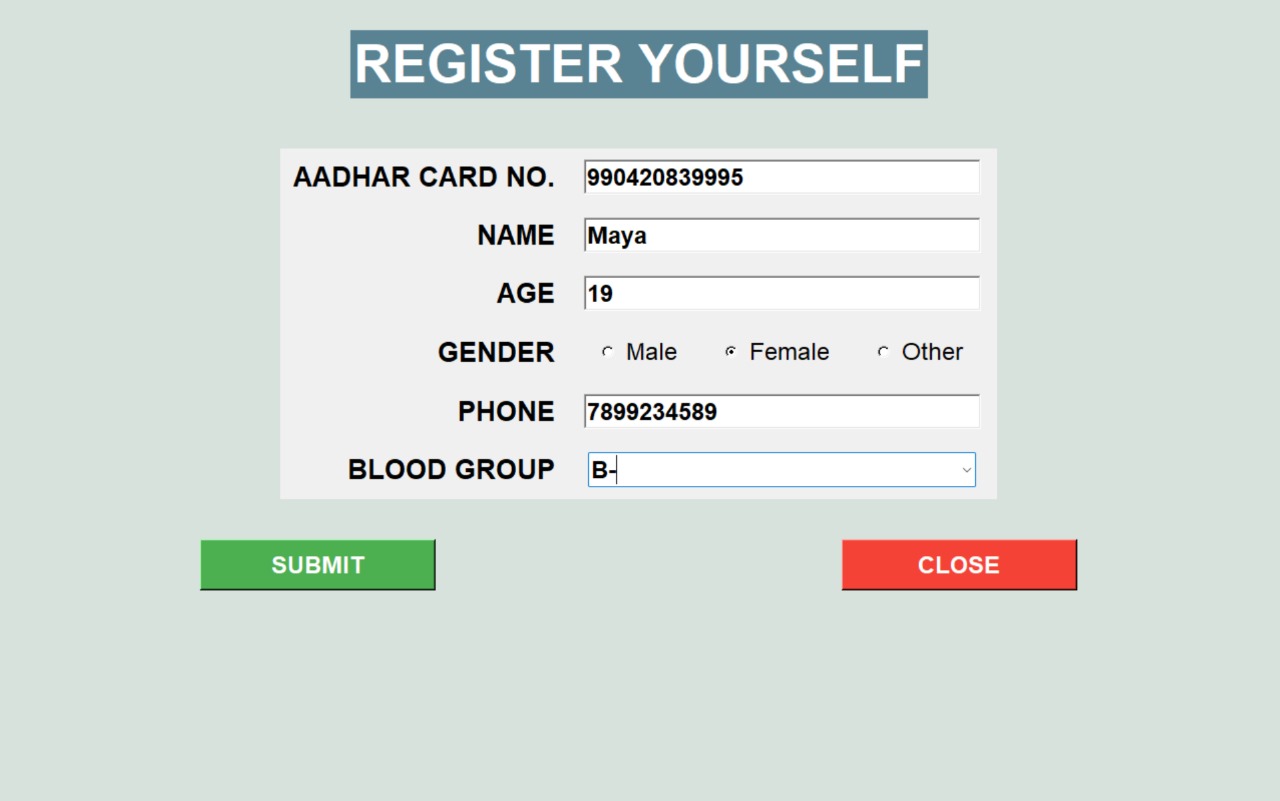
**Main Window:**



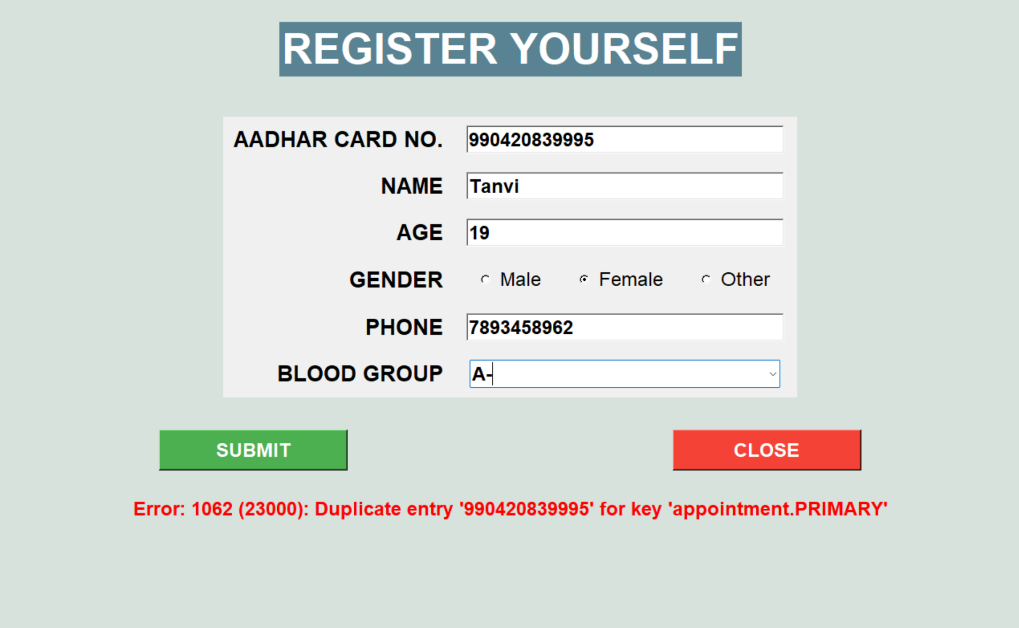
**Registration Window Sample Window:**

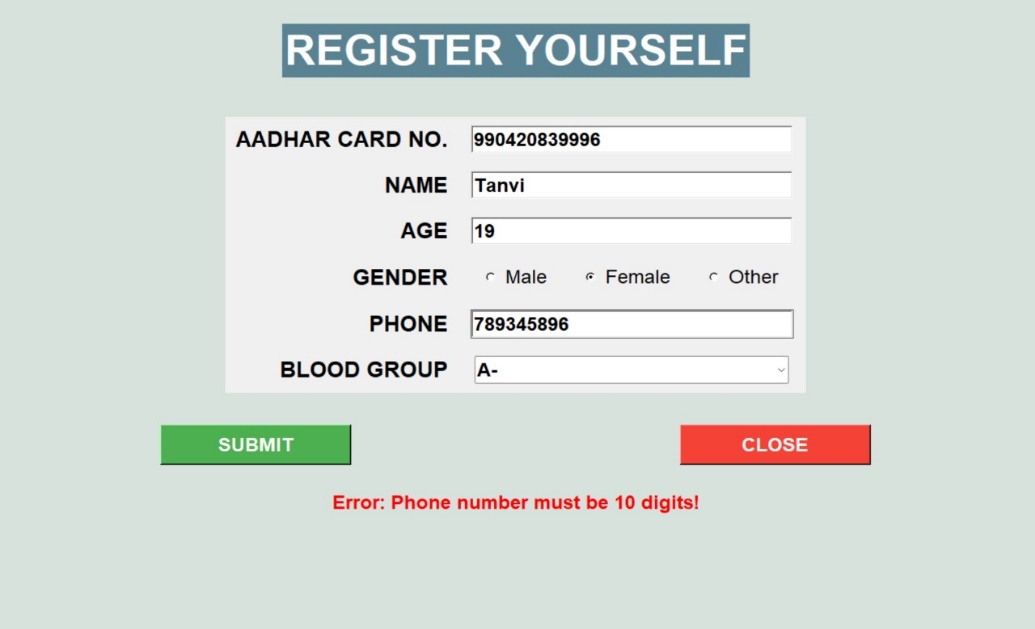
**Expected Outputs:**

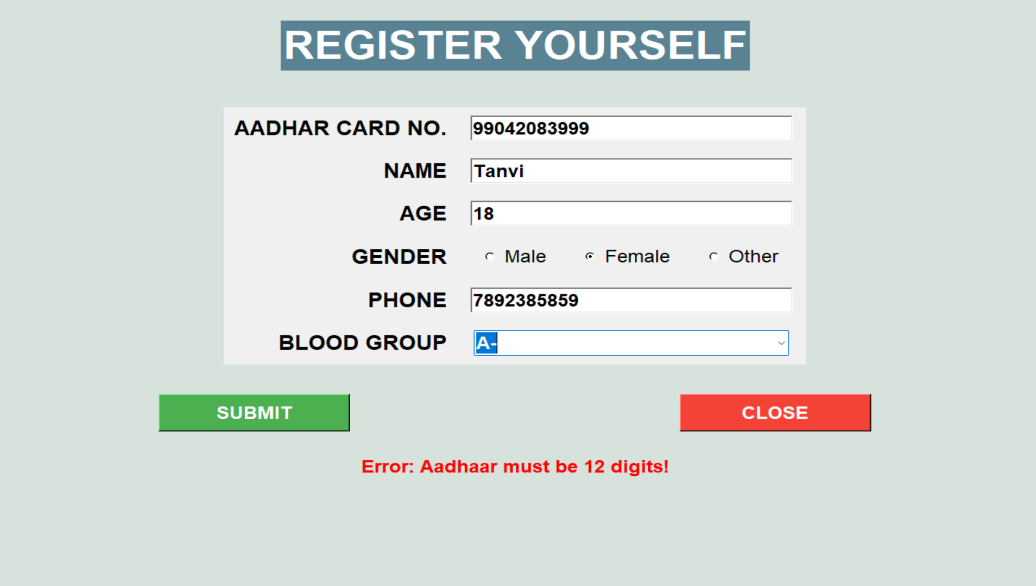




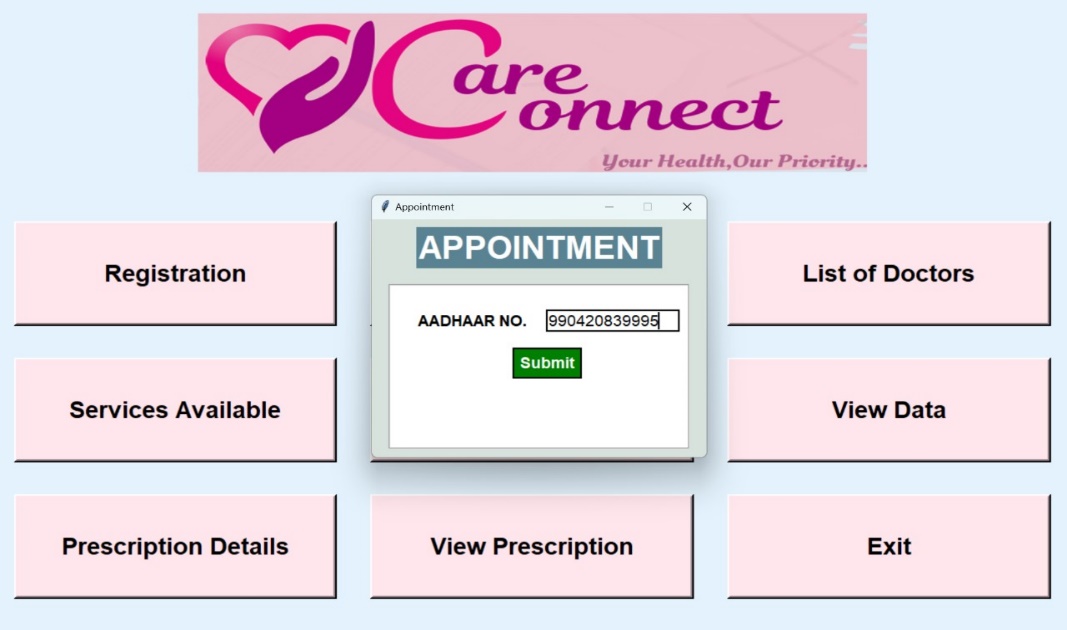
**Exception Handling in Registration**:

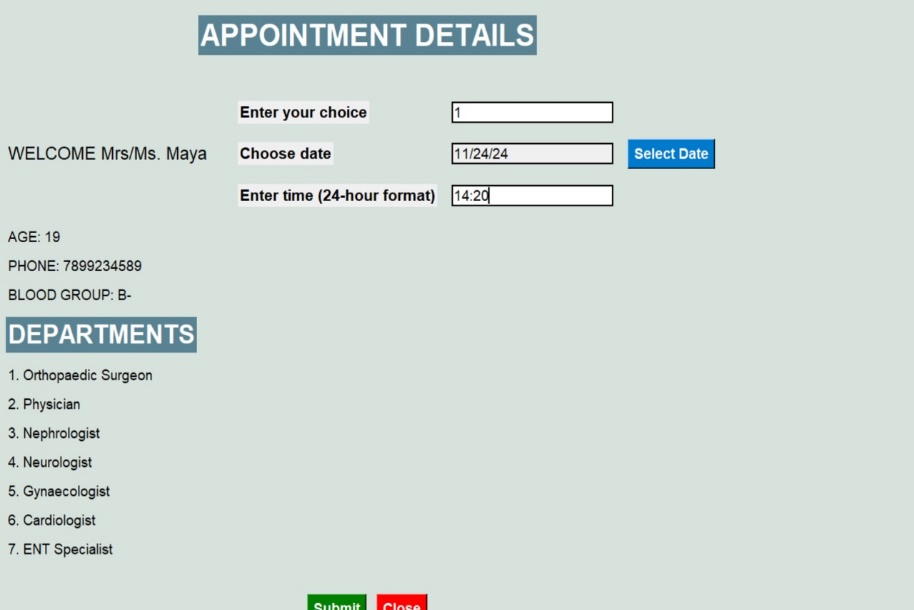


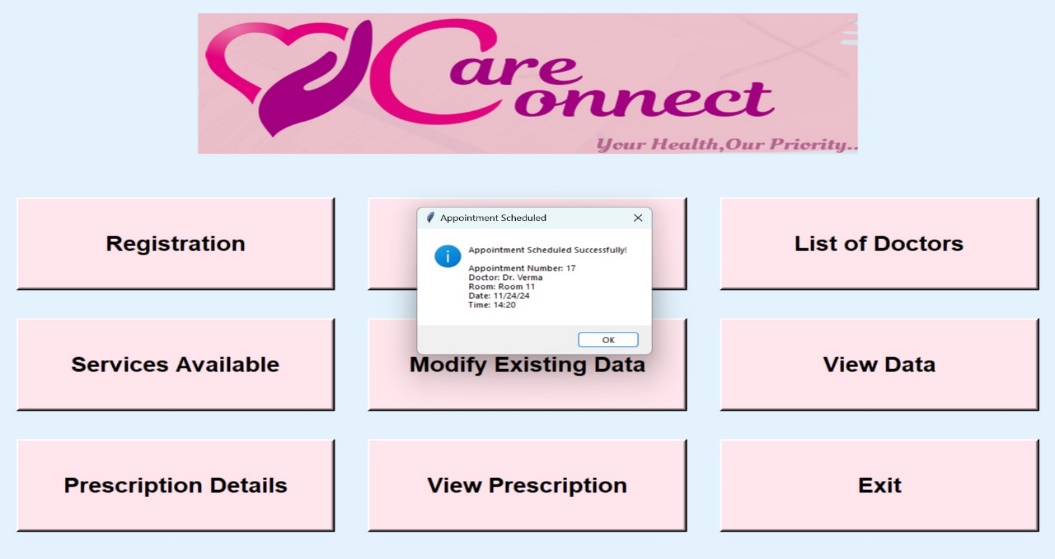




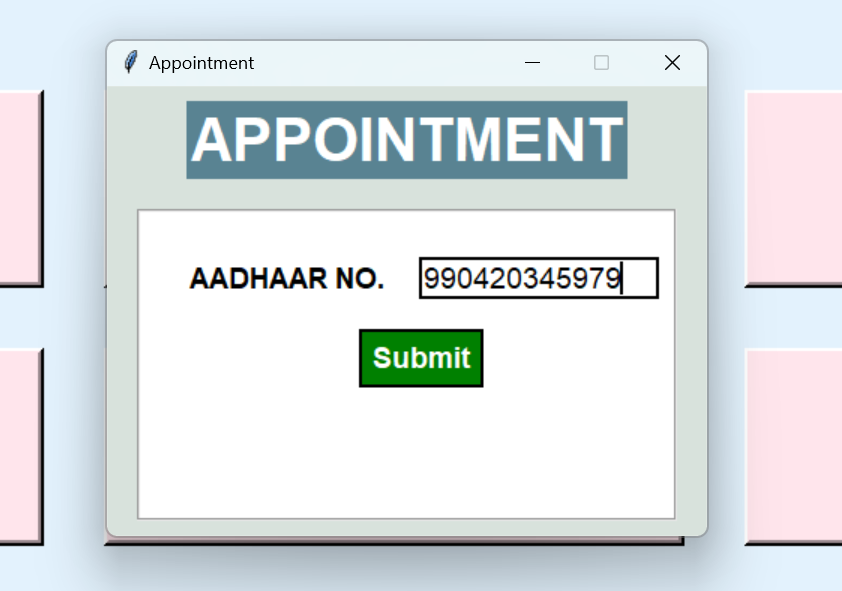
**Appointment Details Sample Output:**

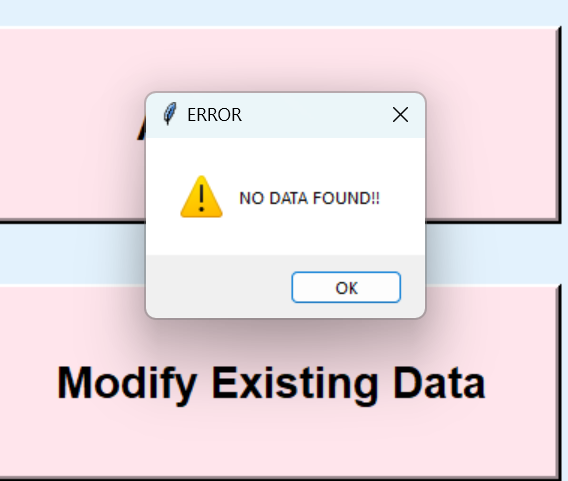




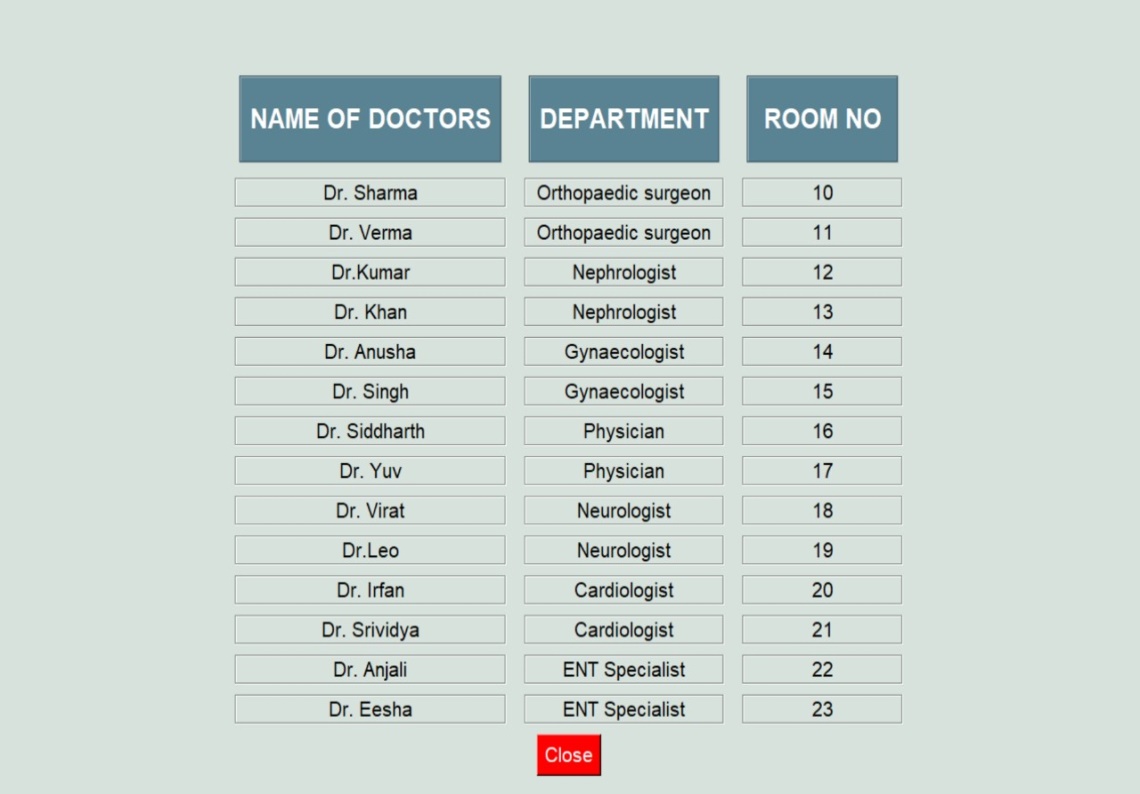


**If Aadhar number not found:**

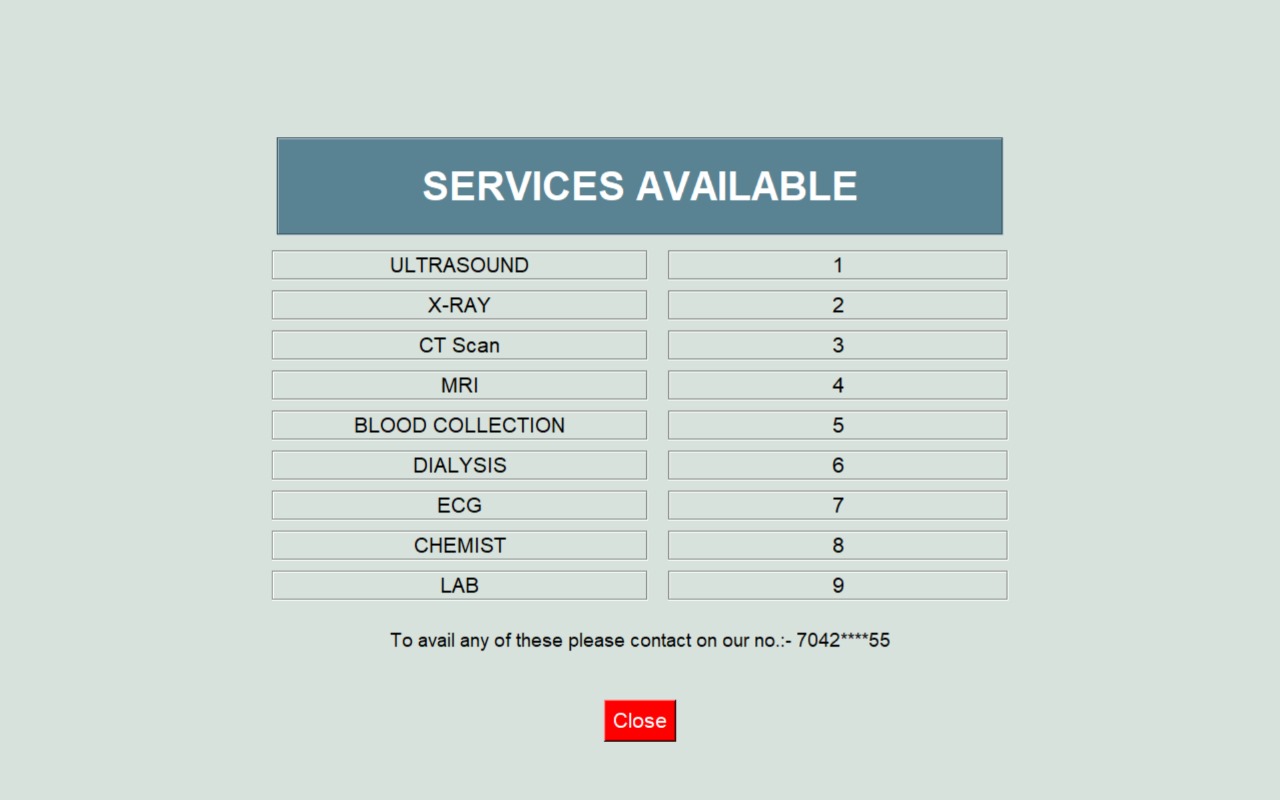




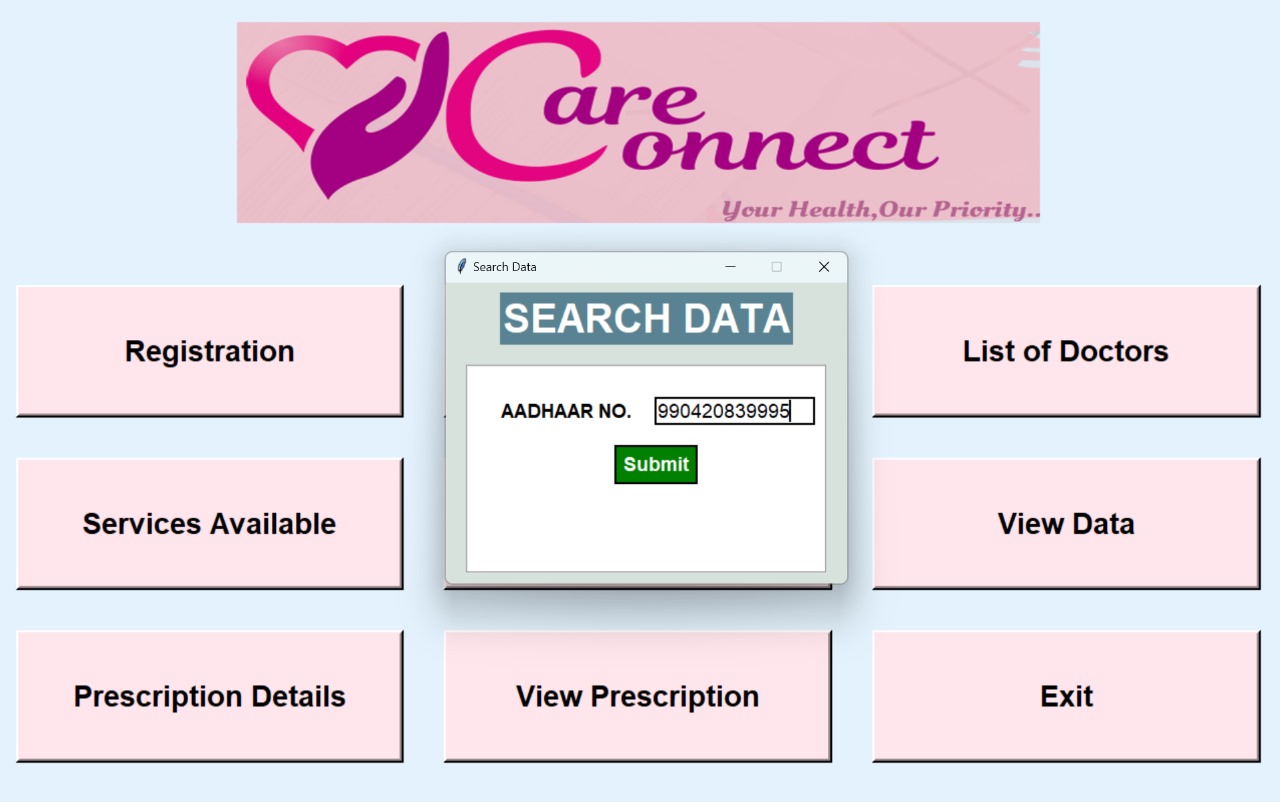
**Doctors Available Sample Output:**

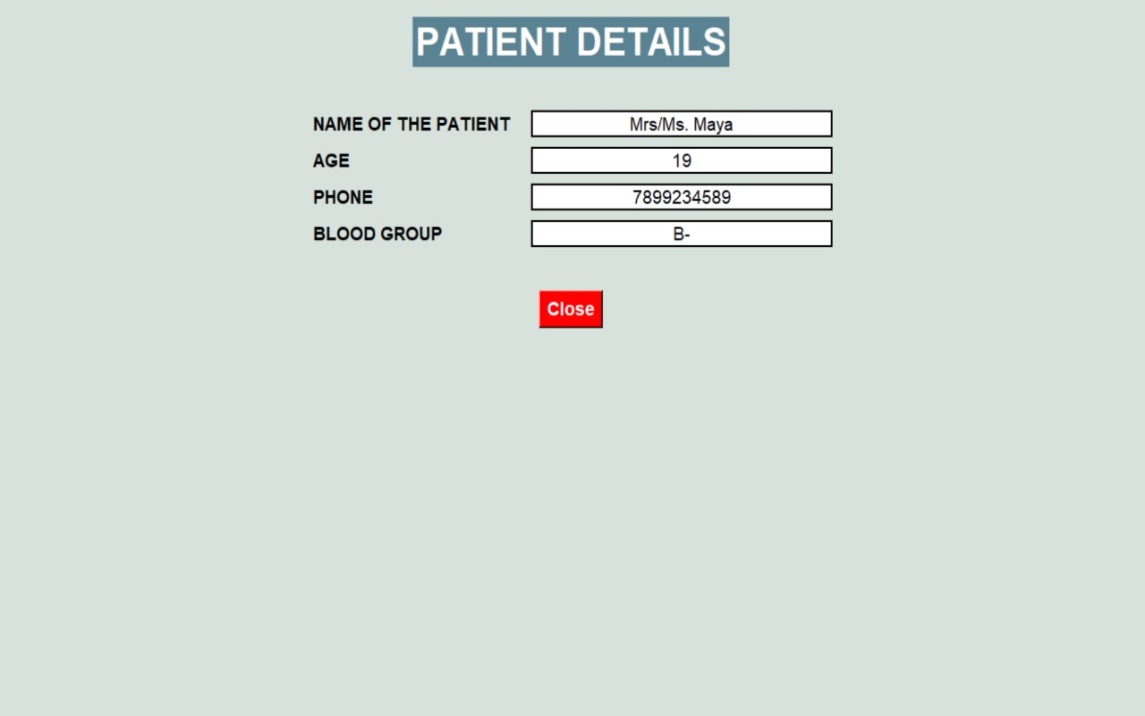


**Services Available Sample Output:**

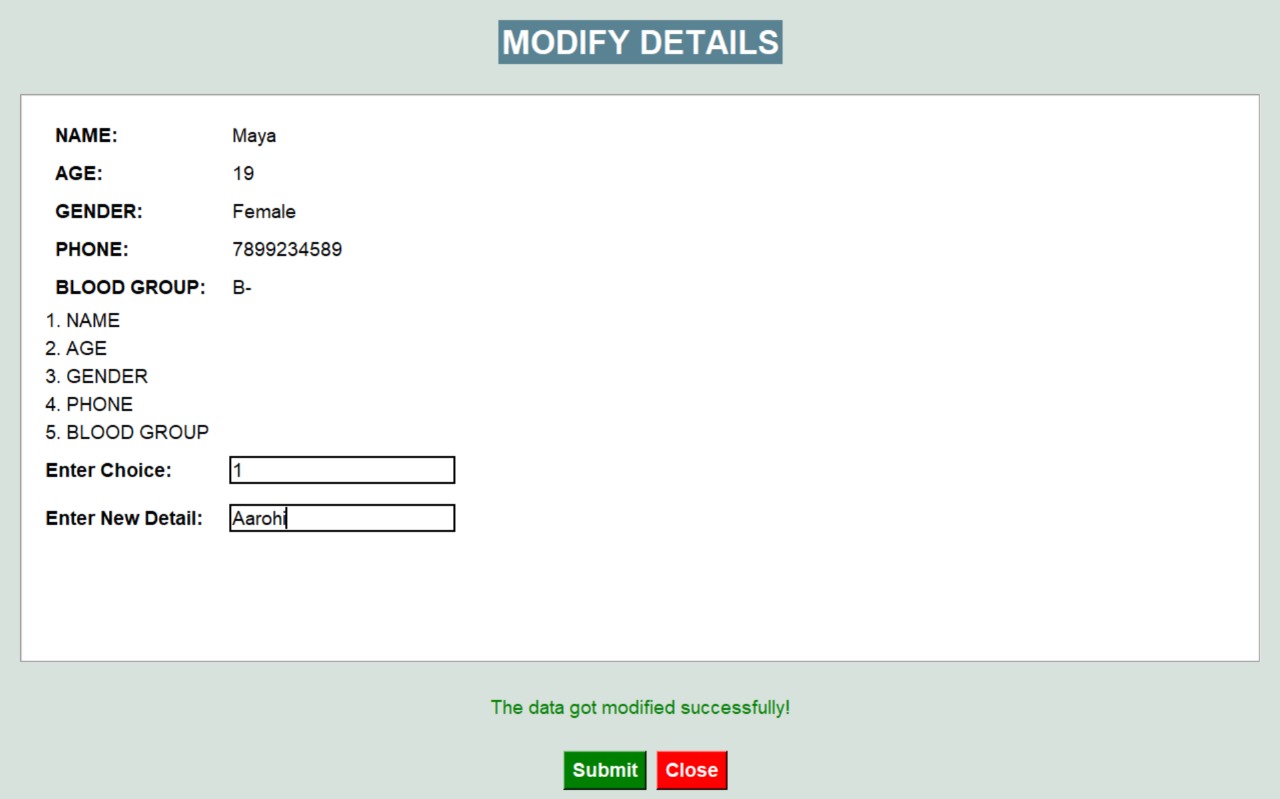


**View Data Sample Output:**

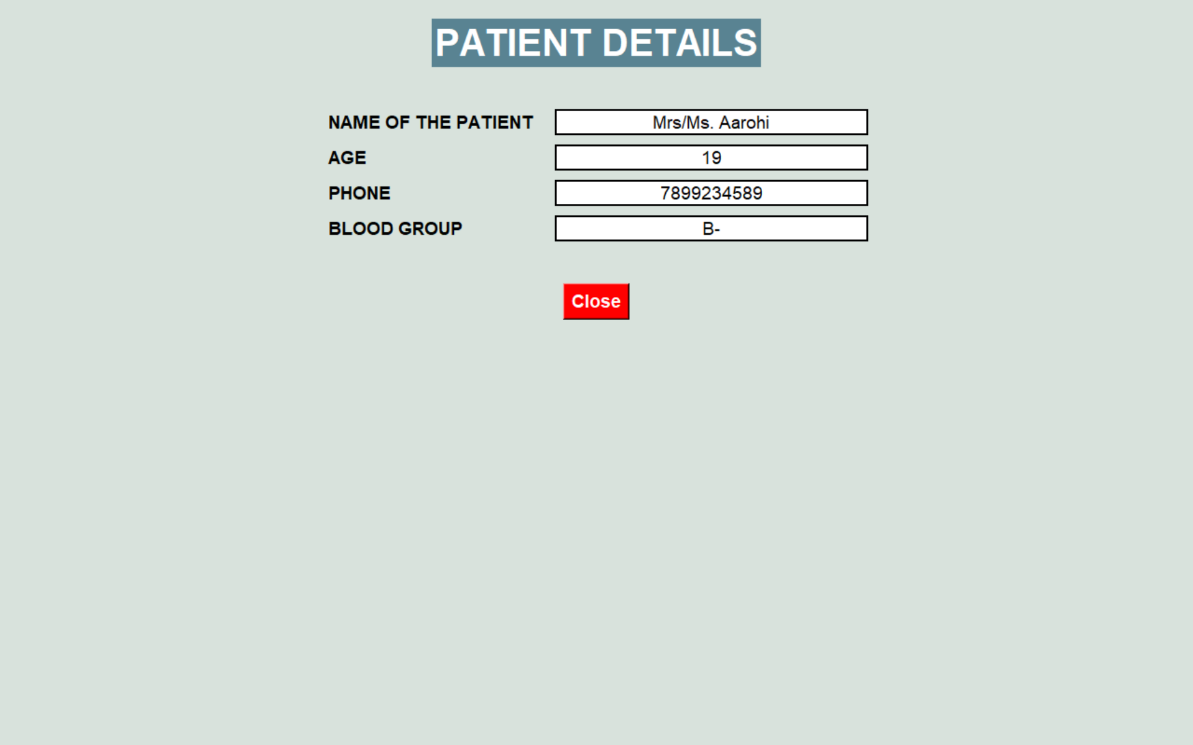




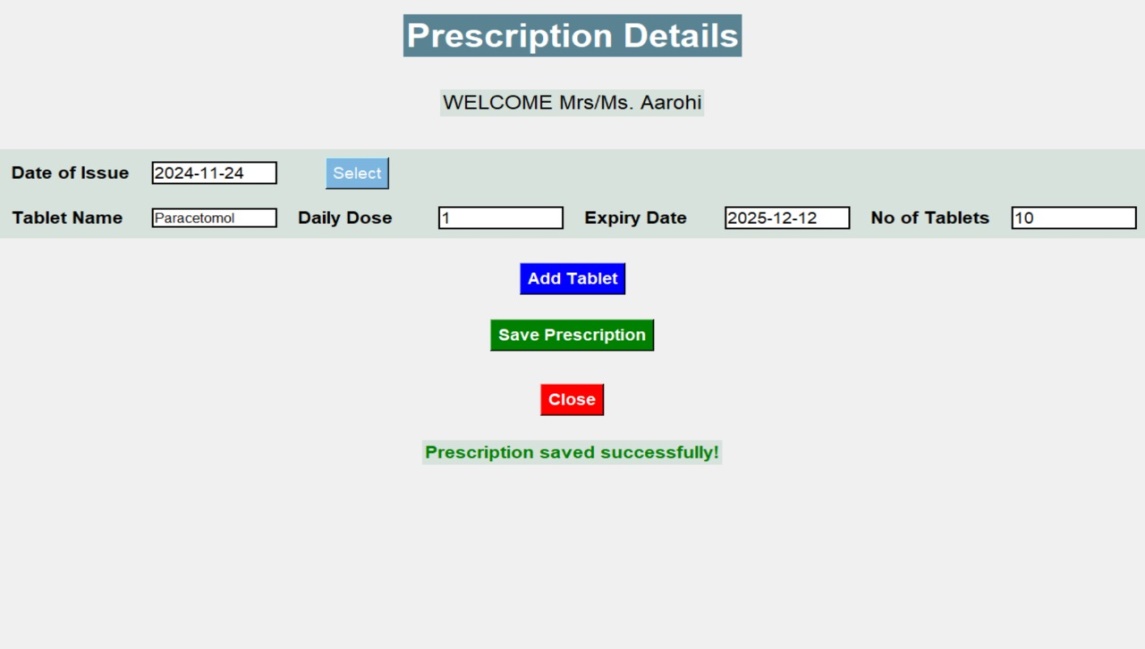
**Modify Data Sample Output:**

****

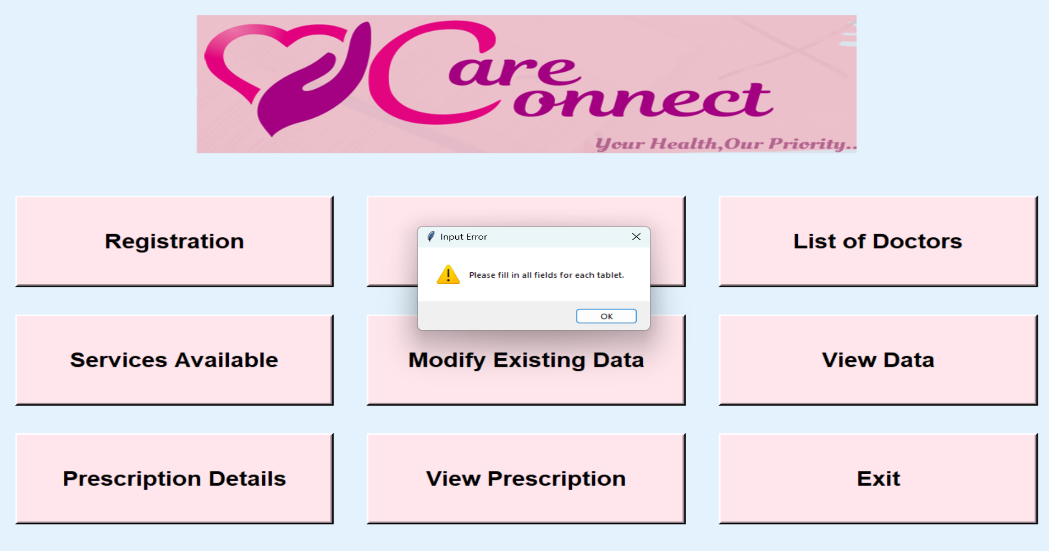
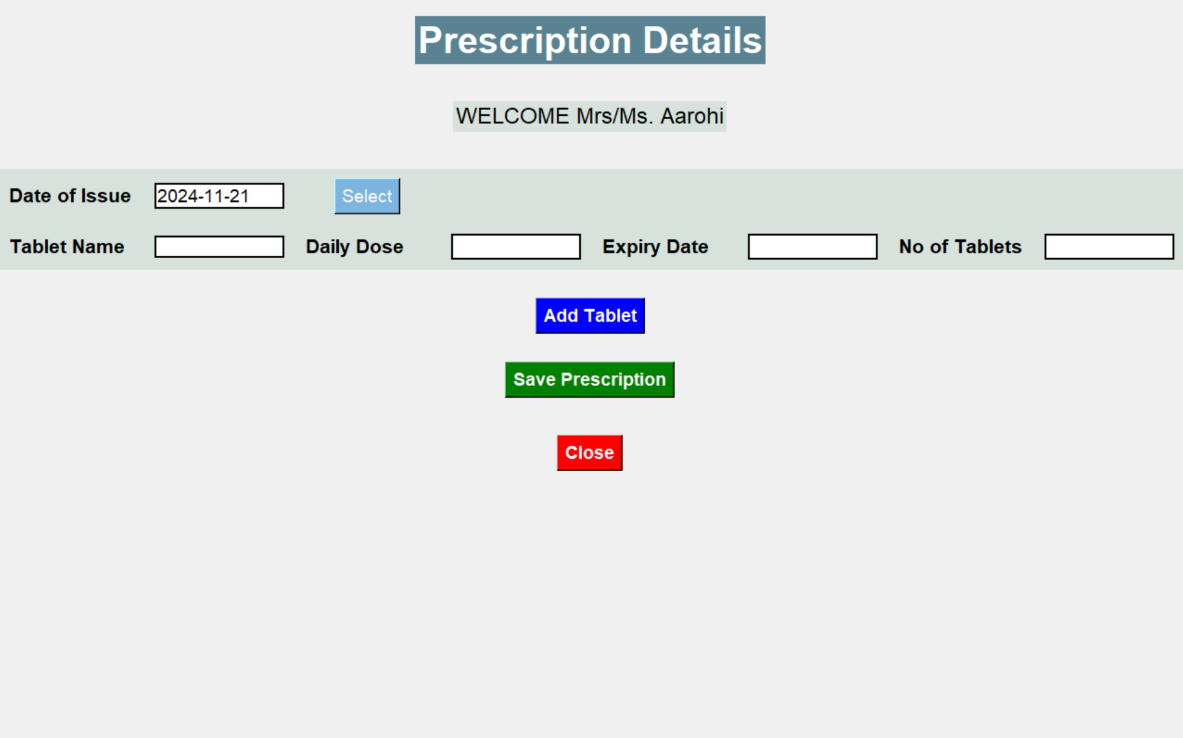
**View Data after Modification:**

****

**Prescription Details Sample Output:**



**If the tablet fields are not filled:**



**View Prescription Sample Output:**





# Additional Knowledge Gained

The development of the Care Connect system provided an opportunity to go beyond the foundational knowledge of Python programming.

# ****1. Database Design and Integration****

**Relational Database Design:** Understanding the process of designing normalized databases, including defining relationships between tables, such as those for patient data, appointment schedules, and prescription details.

**Error Handling in Databases:** Learning to handle exceptions and ensure database integrity during operations like data insertion and modification.

# ****2. Advanced GUI Design****

**User Experience Design:** Creating an intuitive, user-friendly interface using Tkinter, focusing on the placement of elements, color schemes, and ease of navigation.

**Complex Widgets:** Leveraging widgets like Radiobutton, Combobox, and Calendar to provide dynamic user input options.

**Event-Driven Programming:** Implementing button clicks and field inputs as triggers for different system operations improved my understanding of event-based systems.

# ****3. Integration of External Libraries****

**TkCalendar:** Integrating TkCalendar for date selection taught me about library-specific functionalities and customizations.

**MySQL Connector:** Learning to establish secure connections between Python and MySQL databases was invaluable for backend integration.

# 5.****Security and Validation****

**Data Validation:** Implementing checks for Aadhaar numbers, phone numbers, and other critical inputs ensured the system could handle invalid or malicious entries.

**Data Privacy:** Understanding the importance of securing sensitive patient data taught me about industry practices like encryption and access control.

# ****6.Problem Solving and Debugging****

**Practical Problem-Solving:** Real-world challenges like managing foreign key constraints and ensuring data consistency offered valuable problem-solving experiences.

**Debugging Techniques:** Learning to systematically trace and resolve errors in both database interactions and GUI components strengthened my debugging skills.

# ****Conclusion****

The Care Connect system has successfully achieved its objective of streamlining healthcare management processes by providing an intuitive and efficient platform for both patients and healthcare providers. Key functionalities such as patient registration, appointment scheduling, doctor directory, service listings, data modification, and prescription management have been implemented with a focus on usability, data security, and scalability. The project demonstrates the practical application of Python programming, database integration, and GUI development, bridging the gap between theoretical learning and real-world problem-solving.

The modular architecture ensures that the system is robust and maintainable, catering to various user needs with ease. By automating critical tasks and reducing manual intervention, the system enhances operational efficiency, minimizes errors, and improves the overall patient experience. The successful implementation of the Care Connect project underscores the importance of integrating technology into healthcare to address its growing complexity and demands.

# ****Future Work****

While the current version of Care Connect meets its initial goals, there is significant scope for further development and enhancement.

**Enhanced Security Measures**

Implementation of advanced security features such as two-factor authentication and data encryption for increased data protection.

Compliance with global healthcare data regulations such as GDPR and HIPAA.

**Integration of Advanced Technologies**

Incorporating AI-based recommendation systems to suggest suitable doctors or treatments based on patient data.

Using machine learning to predict appointment trends and optimize scheduling.

**Mobile Application Development**

Developing a mobile app version of Care Connect for increased accessibility and convenience for users.

**Multi-Language Support**

Adding support for multiple languages to cater to a broader audience, particularly in regions with diverse linguistic demographics.

**Telemedicine Integration**

Enabling video consultations and remote patient monitoring to make healthcare accessible to patients in remote areas.

**Real-Time Notifications**

Adding real-time notifications via email or SMS to remind patients of upcoming appointments, prescriptions, or health checkups.

# ****References****

1. **Books and Tutorials**
   * Python Crash Course: A Hands-On, Project-Based Introduction to Programming by Eric Matthes.
   * Tkinter GUI Application Development by Example by David Love.
   * Learning SQL by Alan Beaulieu.
2. **Online Documentation**
   * Python Official Documentation: <https://docs.python.org>
   * Tkinter Documentation: <https://docs.python.org/3/library/tkinter.html>
   * MySQL Connector for Python: <https://dev.mysql.com/doc/connector-python/en/>
3. **Web Resources**
   * Stack Overflow: <https://stackoverflow.com>  
     A valuable resource for troubleshooting and understanding Python-related issues.
   * GeeksforGeeks: <https://www.geeksforgeeks.org>  
     Tutorials and examples for database management and GUI development.
   * W3Schools Python Tutorial: https://www.w3schools.com/python/  
     Reference for Python basics and Tkinter GUI concepts.
4. **Libraries and APIs**
   * MySQL Connector Python: <https://pypi.org/project/mysql-connector-python/>  
     Used for integrating MySQL with Python.
   * Tkinter Library: Part of Python Standard Library, used for GUI creation.
5. **Project-Specific References**
   * Healthcare System Workflow Design Articles:
     + <https://www.researchgate.net>  
       Articles on healthcare management systems and workflows.
   * TutorialsPoint: Comprehensive guides on healthcare applications: <https://www.tutorialspoint.com>
6. **Acknowledgments**
   * Professors and mentors for guidance on project structure.
   * Peer-reviewed healthcare management journals for insights into real-world implementations.
   * OpenAI's ChatGPT for ideation and debugging during development.