# BLOOD DONATION MANAGEMENT SYSTEM

**A MINI-PROJECT REPORT**

**Submitted by**


| | |
|---|---|
| **SHINY A** | **220701267** |
| **SNEKHA R** | **220701282** |


**in partial fulfillment of the award of the degree**

**of**

## BACHELOR OF ENGINEERING

## IN

**COMPUTER SCIENCE AND ENGINEERING**


**RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI**

**An Autonomous Institute**

**CHENNAI-602105**


**JUNE 2024**

# BONAFIDE CERTIFICATE

Certified that this project report "**BLOOD DONATION MANAGEMENT SYSTEM**" is the bonafide work of **"SHINY A (220701267), SNEKHA R (220701282) "** who carried out the project work under my supervision.

**SIGNATURE**

**Dr.R.SABITHA**
**Professor and II Year Academic Head,**
**Computer Science and Engineering,**
**Engineering, Rajalakshmi Engineering College**
**(Autonomous),**
**Thandalam, Chennai - 602 105**

**SIGNATURE**

**Mrs.V.JANANEE**
**Assistant Professor (SG),**
**Computer Science and**
**Rajalakshmi Engineering College,**
**(Autonomous),**
**Thandalam, Chennai - 602 105**

**Submitted for the Practical Examination held on _____**

**INTERNAL EXAMINER** **EXTERNAL EXAMINER**

# ABSTRACT

The blood donation management system built with Python to enhance blood bank efficiency and data management. The user-friendly interface, developed using Tkinter, provides a clear visual experience for staff. Tkinter facilitates functionalities like registering new donors and recording donation details, managing blood inventory for various blood groups, searching for available blood based on specific types, and processing blood requests from hospitals or patients. The system's robust backend leverages an SQL database, chosen for its data integrity and efficient retrieval capabilities. The database creates and manages dedicated tables for storing information about donors, blood stock levels, and blood requests. This centralized data storage ensures consistency and simplifies data retrieval for effective blood bank operations. Implementing this system brings several significant benefits. Firstly, it automates repetitive tasks such as data entry and updates, leading to significant improvements in overall efficiency and reducing processing time for staff. Secondly, the system provides a centralized and organized platform for managing all blood bank data, eliminating the need for scattered spreadsheets or paper records. This enhances data integrity and simplifies data retrieval for informed decision-making. Thirdly, the user-friendly Tkinter interface ensures accessibility for staff with varying technical expertise. Finally, the system facilitates real-time tracking of blood stock levels for different blood groups. This vital information allows staff to make informed decisions regarding blood collection drives and resource allocation, ultimately ensuring the timely availability of life-saving blood for those in need.

# TABLE OF CONTENTS

# Ch 1. INTRODUCTION

## 1.1 INTRODUCTION

In this blood donation management system, users can perform essential operations such as registering new donors, recording donation details, managing blood inventory across various blood groups, and processing blood requests from hospitals or patients. The system, developed using Python and featuring a user-friendly Tkinter interface, allows staff to easily navigate and perform tasks with minimal technical expertise. By leveraging an SQL database for its backend, the system ensures robust data integrity and efficient retrieval, storing detailed information about donors, blood stock levels, and blood requests. This centralized data management solution automates repetitive tasks, improves overall efficiency, and provides real-time tracking of blood stock levels, ultimately ensuring the timely availability of life-saving blood.

## 1.2 OBJECTIVE

The main objective of the Blood Donation Management System is to manage the details of Donors, Donations, Blood Inventory, and Blood Requests. It manages all the information about Donors, Blood Stock Levels, and Blood Requests. The project is totally built at the administrative end, and thus only the staff is guaranteed access.

## 1.3 MODULE

- Add donors
- View Donors
- Update Donors
- Delete Donor Details
- Add receiver details
- View receiver details
- Update details
- Delete details

# Ch 2. SURVEY OF TECHNOLOGY

## 2.1 SOFTWARE DESCRIPTION

## Visual studio Code

Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging.
First and foremost, it is an editor that gets out of your way. The delightfully frictionless edit-build-debug cycle means less time fiddling with your environment, and more time executing on your ideas.

## 2.2 LANGUAGES

## 2.2.1 Python

**Python Tkinter**

**Python**: a high-level, interpreted programming language known for its readability and versatility.

**Tkinter**: a standard GUI (Graphical User Interface) library in Python that provides tools for creating graphical interfaces.

**Interface**: a user-friendly environment that allows users to interact with the application through graphical elements like buttons, text fields, and menus.

Tkinter determines the structure and functionality of the application's GUI. This structure alone is not enough to make an application visually appealing or highly interactive. So you'll use assisted technologies such as CSS for styling the GUI components and additional Python libraries for advanced functionalities to enhance your Tkinter applications.

## 2.2.1 SQL

Many of the world's largest and fastest-growing organizations, such as Microsoft, IBM, Oracle, SAP, and Amazon, rely on SQL for managing their high-volume databases and business-critical systems. The performance, scalability, reliability, and ease of use of SQL have made it the top choice for relational database management and querying, continuously improving to meet the demands of modern data-driven applications.

# Ch 3. REQUIREMENT AND ANALYSIS

## 3.1 REQUIREMENTS SPECIFICATION

### User Requirements

The system requirement in the blood donation management system focuses on the ability to search for available blood by blood type, donor details, or receiver details by the staff.

### System Requirements

There should be a database backup of the library management system. Operating system should be WindowsXP or a higher version of windows

## 3.2 HARDWARE AND SOFTWARE REQUIREMENTS

### Software Requirements

• Operating System Windows 10
• Front End Python
• Back End SQL

### Hardware Requirements

● Desktop PC or a Laptop
● Printer
● Operating System – Windows 10
● Intel® CoreTM i3-6006U CPU @ 2.00GHz
● 4.00 GB RAM
● 64-bit operating system, x64 based processor
● 1024 x 768 monitor resolution
● Keyboard and Mouse

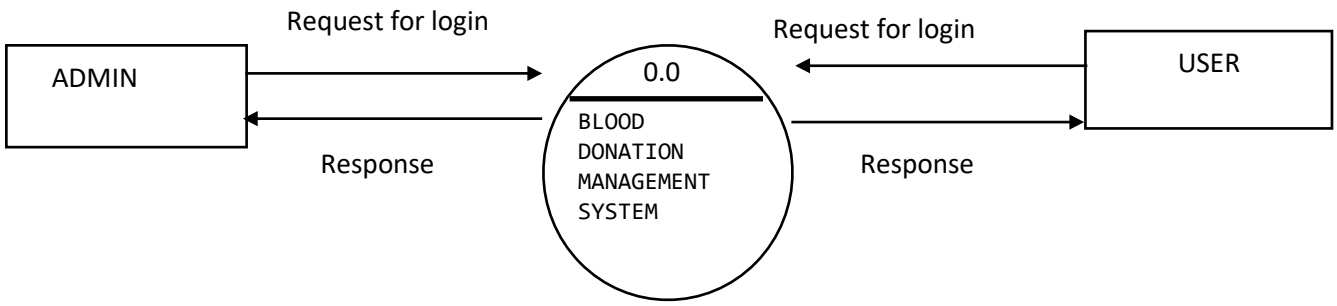
## 3.3 DATA FLOW DIAGRAM

DFD is an important tool used by system analysis. A data flow diagram model, a system using external entities from which data flows through a process which transforms the data and creates output data transforms which go to other processes external entities such as files. The main merit of DFD is that it can provide an overview of what data a system would process.

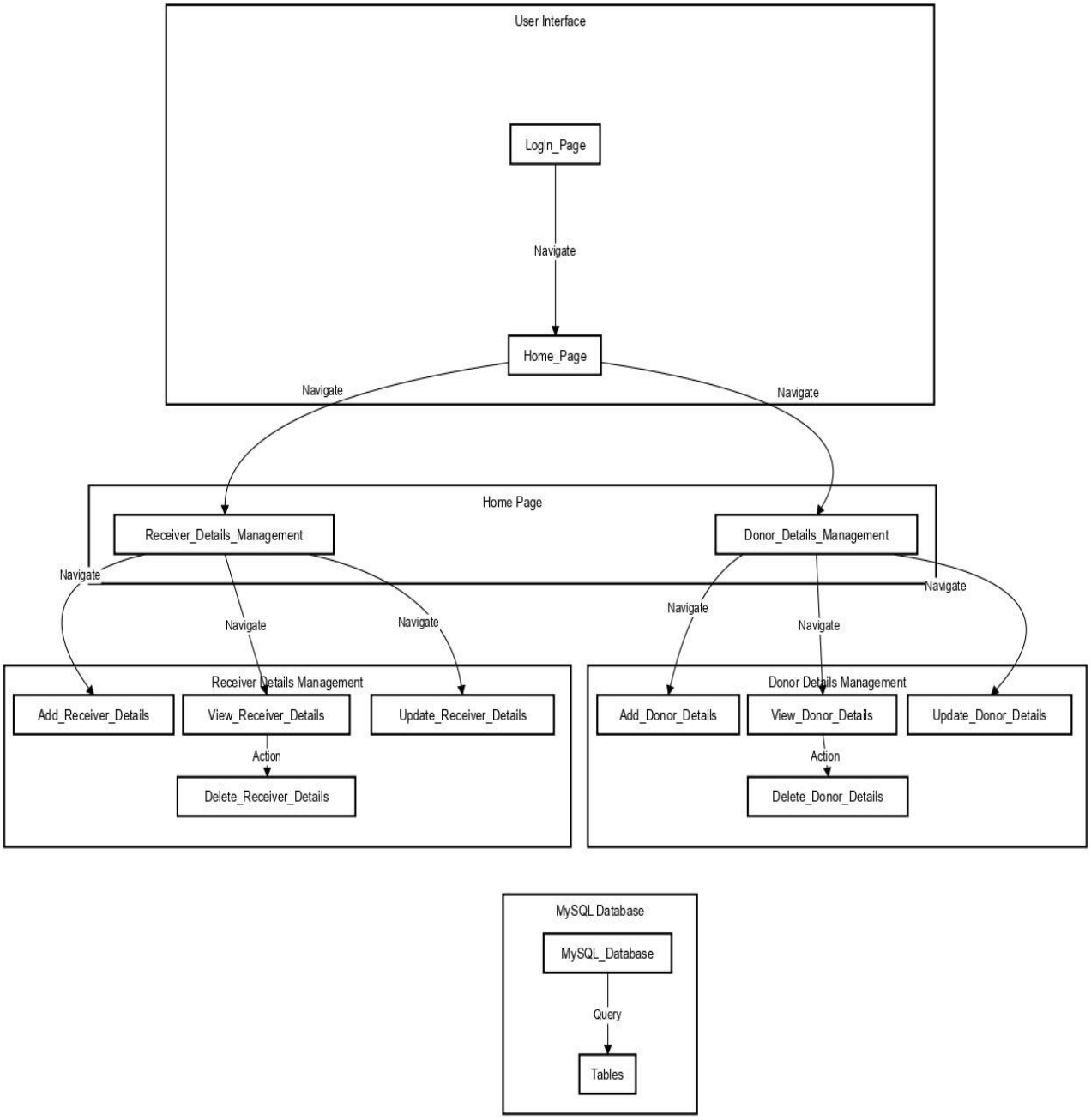
• A data-flow diagram is a way of representing a flow of data through a process or a system.

• The DFD also provides information about the outputs and inputs of each entity and the process itself.

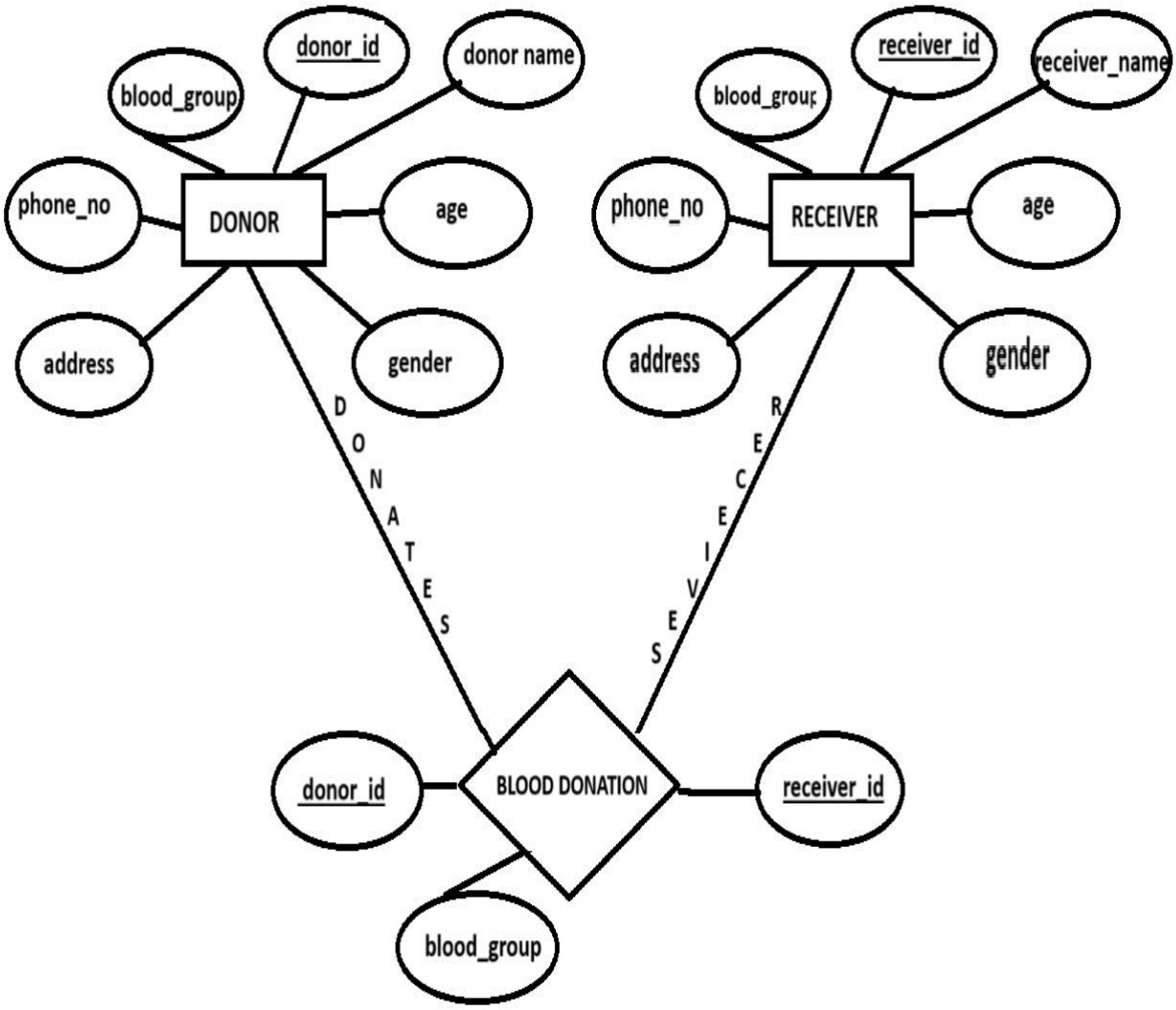
## CONTEXT LEVEL DFD :



# 3.3 ARCHITECTURAL DIAGRAM



## 3.4 ER DIAGRAM



# 3.5 NORMALIZATION

## 1NF TABLES:

### Donors:

| donor_id | donor_name | age | gender | phone_no | address | blood_ group | other_medical_ details |
|---|---|---|---|---|---|---|---|
| 1 | Snekha | 19 | F | 9444689216 | Egmore | A1+ve | none |
| 2 | Shiny | 20 | F | 7448848591 | Chennai | B+ve | none |

### Receivers:

| receiver_i d | receiver_name | age | gender | phone_no | address | blood_ group | other_medical_ details |
|---|---|---|---|---|---|---|---|
| 1 | Bharathi | 21 | M | 6383252593 | kanchipur am | O+ve | none |
| 2 | Swetha | 19 | F | 9790266947 | Trichy | A1+ve | none |

## 2NF TABLES :

### Donors

| donor_id | donor_name | age | gender |
|---|---|---|---|
| 1 | snekha | 19 | F |
| 2 | shiny | 20 | F |

### Receivers

| receiver_id | receiver_name | age | gender |
|---|---|---|---|
| 1 | Bharathi | 21 | M |
| 2 | Swetha | 19 | F |

## DonorsContactDetails

| donor_id | phone_no | address |
|---|---|---|
| 1 | 9444568216 | egmore |
| 2 | 7448848591 | chennai |

## ReceiverContactDetails

| receiver_id | phone_no | address |
|---|---|---|
| 1 | 9444568216 | egmore |
| 2 | 7448848591 | chennai |

## DonorMedicalDetails

| donor_id | bloodgroup | other_medical_details |
|---|---|---|
| 1 | A1+ve | none |
| 2 | B+ve | none |

## ReceiverMedicalDetails

| receiver_id | bloodgroup | other_medical_details |
|---|---|---|
| 1 | O+ve | none |
| 2 | A1+ve | none |

# Ch 4. PROGRAM CODE

## 4.1. CODE DETAILS AND CODE EFFICIENCY

Index

```
from tkinter import *
from tkinter import messagebox
import mysql.connector
from tkinter import ttk,simpledialog
from PIL import Image, ImageTk  # Import Pillow modules


db_host = "localhost"
db_name = "blooddonation"
db_user = "root"
db_password = "Snekha@2402"
def connect_to_database():
    try:
        connection = mysql.connector.connect(host=db_host, database=db_name,
user=db_user, password=db_password)
        return connection
    except mysql.connector.Error as err:
        messagebox.showerror(title="Database Error", message=err)
        return None


# Function to check login credentials
def login():
    username = username_entry.get()
    password = password_entry.get()

    # Replace these with your actual credentials (or a mechanism to store them
securely)
    correct_username = "admin"
    correct_password = "123"

    if username == correct_username and password == correct_password:
        messagebox.showinfo(title="Login Successful!", message="Welcome to
DONATE4LIFE!")
```

```
        destroy_login_page()
        create_home_page()
    else:
        messagebox.showerror(title="Error", message="Invalid username or
password.")

# Function to destroy the login page
def destroy_login_page():
    login_frame.destroy()

# Function to create the home page
def create_home_page():
    # Create a new window for the home page
    home_window = Toplevel()
    home_window.title("DONATE4LIFE - Home Page")

    # Get screen width and height
    screen_width = home_window.winfo_screenwidth()
    screen_height = home_window.winfo_screenheight()

    # Set the geometry of the window to cover the entire screen
    home_window.geometry(f"{screen_width}x{screen_height}")

    # Load background image for home page
    home_background_image = Image.open("home.jpg")
    home_background_image = home_background_image.resize((screen_width,
screen_height))
    home_background_photo = ImageTk.PhotoImage(home_background_image)

    # Create a label to display the background image
    home_background_label = Label(home_window,
image=home_background_photo)
    home_background_label.place(x=0, y=0, relwidth=1, relheight=1)

    # Create buttons for blood donation management system project
    # Example buttons, you can customize as needed
    donors_button = Button(home_window, text="DONATER
DETAILS",bg="lightgrey" ,font=("Times New Roman", 12,"bold"),height=3,
width=50,command=donors)
    donors_button.place(x=150, y=150)

    receiver_button = Button(home_window, text="RECEIVER DETAILS",
bg="lightgrey",font=("Times New Roman", 12,"bold"),height=3,
width=50,command=receiver)
    receiver_button.place(x=150, y=230)
```

```
    logout_button = Button(home_window, text="LOGOUT",bg="lightgrey"
,font=("Times New Roman", 12,"bold"),height=3, width=50,command=logout)
    logout_button.place(x=150,y=310)



    # Add more buttons or options as needed

    # Run the home window's main loop
    home_window.mainloop()

# Functions to handle button commands (example functions, customize as needed)
# Function to create the Donater Details page
def create_donater_details_page():
    # Create a new window for the Donater Details page

    donater_details_window = Toplevel()
    donater_details_window.title("DONATE4LIFE - Donater Details")

    # Get screen width and height
    screen_width = donater_details_window.winfo_screenwidth()
    screen_height = donater_details_window.winfo_screenheight()

    # Set the geometry of the window to cover the entire screen
    donater_details_window.geometry(f"{screen_width}x{screen_height}")
    background_image = Image.open("bg1.jpg")
    background_image = background_image.resize((screen_width, screen_height))
    background_photo = ImageTk.PhotoImage(background_image)

        # Create a label to display the background image
    background_label = Label(donater_details_window, image=background_photo)
    background_label.place(x=0, y=0, relwidth=1, relheight=1)

        # Ensure that the image is not garbage collected
    background_label.image = background_photo

    # Create a label to display the background image


    # Function to view the donors table
    # Function to view the donors table
    def view_donors_table():
```

```
    # Create a new window for displaying donor details
    donors_table_window = Toplevel()
    donors_table_window.title("DONATE4LIFE - Donors Table")

    # Get screen width and height
    screen_width = donors_table_window.winfo_screenwidth()
    screen_height = donors_table_window.winfo_screenheight()

    # Set the geometry of the window (optional for full screen)
    donors_table_window.geometry(f"{screen_width}x{screen_height}")

    connection = connect_to_database()
    if not connection:
        return

    try:
        cursor = connection.cursor()
        cursor.execute("SELECT * FROM donors")
        donor_data = cursor.fetchall()

        # Create a Treeview widget to display donor data
        donor_table = ttk.Treeview(donors_table_window, columns=("donor_id",
"donor_name", "age", "gender", "address", "phone_number", "blood_group",
"other_medical_details"), show="headings")
        donor_table.heading("donor_id", text="ID")
        donor_table.heading("donor_name", text="Donor Name")
        donor_table.heading("age", text="Age")
        donor_table.heading("gender", text="Gender")
        donor_table.heading("address", text="Address")
        donor_table.heading("phone_number", text="Phone Number")
        donor_table.heading("blood_group", text="Blood Group")
        donor_table.heading("other_medical_details", text="Other Medical
Details")

        donor_table.grid(row=0, columnspan=1, padx=5, pady=5)

        # Insert data into the Treeview
        for donor in donor_data:
            donor_table.insert("", "end", values=donor)

        # Adjusting column widths
        donor_table.column("donor_id", width=50)
        donor_table.column("donor_name", width=150)
        donor_table.column("age", width=50)
        donor_table.column("gender", width=80)
```

```
            donor_table.column("address", width=200)
            donor_table.column("phone_number", width=120)
            donor_table.column("blood_group", width=80)
            donor_table.column("other_medical_details", width=200)


            # Function to handle delete button clicks


        except mysql.connector.Error as err:
            messagebox.showerror(title="Database Error", message=err)
        finally:
            if connection:
                connection.close()

        # Run the window's main loop
        donors_table_window.mainloop()









    # Function to handle saving donor details
    def save_donor_details():
        connection = connect_to_database()
        if not connection:
            return
        donor_name = donor_name_entry.get()
        age=age_entry.get()
        gender = gender_var.get()
        address = address_entry.get()
        phone_number = phone_number_entry.get()
        blood_group = blood_group_entry.get()
        other_medical_details = other_medical_details_entry.get("1.0", END)  #
Retrieve text from Text widget


        try:
            cursor = connection.cursor()
            sql = ("INSERT INTO donors (donor_name, age, gender, address,
phone_number, blood_group, other_medical_details) "
            "VALUES (%s, %s, %s, %s, %s, %s, %s)")
```

```
            cursor.execute(sql, (donor_name, age, gender, address, phone_number,
blood_group, other_medical_details))
            connection.commit()
            messagebox.showinfo("Success", "Donor details saved successfully!")

        except mysql.connector.Error as err:
            messagebox.showerror(title="Database Error", message=err)
        finally:
            if connection:
                connection.close()



        # Here, you can add code to save the donor details to a database or perform
other actions

        # Example: Print the donor details
        print("Donor Name:", donor_name)
        print("Age:", age)
        print("Gender:", gender)
        print("Address:", address)
        print("Phone Number:", phone_number)
        print("Blood Group:", blood_group)
        print("Other Medical Details:", other_medical_details)

    def delete_donor_by_id():
        donor_id = simpledialog.askinteger("Input", "Enter Donor ID to delete:")
        if donor_id is None:
            return

        connection = connect_to_database()
        if not connection:
            return

        try:
            cursor = connection.cursor()
            cursor.callproc("delete_donor_id", [donor_id])
            connection.commit()
            messagebox.showinfo("Success", "Donor record deleted successfully!")
        except mysql.connector.Error as err:
            messagebox.showerror(title="Database Error", message=err)
        finally:
            if connection:
                connection.close()
```

```
    def update_donor_details():
        def save_changes():
            donor_id = donor_id_entry.get()
            new_name = entry_values[0].get()
            new_age = entry_values[1].get()
            new_gender = gender_var.get()
            new_address = entry_values[2].get()
            new_phone_number = entry_values[3].get()
            new_blood_type = entry_values[4].get()



            connection = connect_to_database()
            if not connection:
                return

            try:
                cursor = connection.cursor()
                cursor.callproc('update_donor_id', [donor_id, new_name, new_age,
new_gender, new_address, new_phone_number, new_blood_type])
                connection.commit()
                messagebox.showinfo(title="Success", message=f"Donor ID
{donor_id} updated successfully.")
                donor_details_window.destroy()  # Close the window after successful
update
            except mysql.connector.Error as err:
                messagebox.showerror(title="Error", message=str(err))
            finally:
                if connection.is_connected():
                    cursor.close()
                    connection.close()

        donor_details_window = Toplevel()
        donor_details_window.title("Update Donor Details")

        # Donor ID Label and Entry
        donor_id_label = Label(donor_details_window, text="Donor ID:",
font=("Times New Roman", 16))
        donor_id_label.grid(row=0, column=0, padx=10, pady=10)
        donor_id_entry = Entry(donor_details_window, font=("Times New Roman",
16))
        donor_id_entry.grid(row=0, column=1, padx=10, pady=10)

        # Other Donor Details Labels and Entries
```

```
        details_labels = ["Name:", "Age:", "Gender:", "Address:", "Phone Number:",
"Blood Type:"]
        entry_values = [StringVar() for _ in range(len(details_labels))]
        for i, label_text in enumerate(details_labels):
            label = Label(donor_details_window, text=label_text, font=("Times New
Roman", 16))
            label.grid(row=i+1, column=0, padx=10, pady=10)
            entry = Entry(donor_details_window, font=("Times New Roman", 16),
textvariable=entry_values[i])
            entry.grid(row=i+1, column=1, padx=10, pady=10)
            if i == 2:  # Gender dropdown
                gender_options = ["Male", "Female", "Other"]
                gender_var = StringVar(donor_details_window)
                gender_var.set(gender_options[0])  # Default value
                gender_dropdown = OptionMenu(donor_details_window, gender_var,
*gender_options)
                gender_dropdown.config(font=("Times New Roman", 12))
                gender_dropdown.grid(row=i+1, column=1, padx=10, pady=10)

        save_button = Button(donor_details_window, text="Save",
command=save_changes, font=("Times New Roman", 16))
        save_button.grid(row=len(details_labels)+1, columnspan=2, pady=10)

        donor_details_window.mainloop()


    # Optionally, show a success message

# Create input fields for donor details
    donor_name_label = Label(donater_details_window, text="Donor Name:",
font=("Times New Roman", 16))
    donor_name_label.grid(row=0, column=0, padx=10, pady=10)
    donor_name_entry = Entry(donater_details_window, font=("Times New
Roman", 16))
    donor_name_entry.grid(row=0, column=1, padx=10, pady=10)

    age_label = Label(donater_details_window, text="Age:", font=(None, 16))
    age_label.grid(row=1, column=0, padx=10, pady=10)
    age_entry = Entry(donater_details_window, font=("Times New Roman", 16))
    age_entry.grid(row=1, column=1, padx=10, pady=10)

    gender_label = Label(donater_details_window, text="Gender:", font=(None,
16))
    gender_label.grid(row=2, column=0, padx=10, pady=10)
    gender_var = StringVar(donater_details_window)
```

```
    gender_var.set("Male")
    gender_options = ["Male", "Female", "Other"]
    gender_dropdown = OptionMenu(donater_details_window, gender_var,
*gender_options)
    gender_dropdown.config(font=(None, 16))
    gender_dropdown.grid(row=2, column=1, padx=10, pady=10)

    address_label = Label(donater_details_window, text="Address:", font=("Times
New Roman", 16))
    address_label.grid(row=3, column=0, padx=10, pady=10)
    address_entry = Entry(donater_details_window, font=("Times New Roman",
16))
    address_entry.grid(row=3, column=1, padx=10, pady=10)

    phone_number_label = Label(donater_details_window, text="Phone Number:",
font=("Times New Roman", 16))
    phone_number_label.grid(row=4, column=0, padx=10, pady=10)
    phone_number_entry = Entry(donater_details_window, font=("Times New
Roman", 16))
    phone_number_entry.grid(row=4, column=1, padx=10, pady=10)

    blood_group_label = Label(donater_details_window, text="Blood Group:",
font=("Times New Roman", 16))
    blood_group_label.grid(row=5, column=0, padx=10, pady=10)
    blood_group_entry = Entry(donater_details_window, font=("Times New
Roman", 16))
    blood_group_entry.grid(row=5, column=1, padx=10, pady=10)

    other_medical_details_label = Label(donater_details_window, text="Other
Medical Details:", font=("Times New Roman", 16))
    other_medical_details_label.grid(row=6, column=0, padx=10, pady=10)
    other_medical_details_entry = Text(donater_details_window, font=("Times
New Roman", 16), height=5, width=30)
    other_medical_details_entry.grid(row=6, column=1, padx=10, pady=10)

    # Create a button to save donor details
    save_button = Button(donater_details_window, text="Save",
command=save_donor_details, font=("Times New Roman", 16))
    save_button.grid(row=7, columnspan=2, pady=10)

    view_donors_button = Button(donater_details_window, text="View Donors",
command=view_donors_table, font=("Times New Roman", 16))
    view_donors_button.grid(row=9, columnspan=2, pady=10)
```

```
    delete_button = Button(donater_details_window, text="Delete",
command=delete_donor_by_id, font=("Times New Roman", 16))
    delete_button.grid(row=10, columnspan=2, pady=10)

    update_button = Button(donater_details_window, text="Modify",
command=update_donor_details, font=("Times New Roman", 16))
    update_button.grid(row=11, columnspan=2, pady=10)


    # Run the Donater Details window's main loop
    donater_details_window.mainloop()

# Modify the donors() function to open the Donater Details page
def donors():
    create_donater_details_page()

def create_receiver_details_page():
    # Create a new window for the Donater Details page

    receiver_details_window = Toplevel()
    receiver_details_window.title("DONATE4LIFE - Donater Details")

    # Get screen width and height
    screen_width = receiver_details_window.winfo_screenwidth()
    screen_height = receiver_details_window.winfo_screenheight()

    # Set the geometry of the window to cover the entire screen
    receiver_details_window.geometry(f"{screen_width}x{screen_height}")

    # Function to handle saving donor details
    def save_receiver_details():
        receiver_name = receiver_name_entry.get()
        age=age_entry.get()
        gender = gender_var.get()
        address = address_entry.get()
        phone_number = phone_number_entry.get()
        blood_group = blood_group_entry.get()
        other_medical_details = other_medical_details_entry.get("1.0", END)  #
Retrieve text from Text widget

        # Here, you can add code to save the donor details to a database or perform
other actions

        # Example: Print the donor details
        print("Receiver Name:", receiver_name)
```

```
        print("Age:", age)
        print("Gender:", gender)
        print("Address:", address)
        print("Phone Number:", phone_number)
        print("Blood Group:", blood_group)
        print("Other Medical Details:", other_medical_details)

        # Optionally, you can show a message confirming that the details are saved
        messagebox.showinfo("Success", "Receiver details saved successfully!")

    # Create input fields for donor details
    receiver_name_label = Label(receiver_details_window, text="Receiver
Name:", font=("Times New Roman", 16))
    receiver_name_label.grid(row=0, column=0, padx=10, pady=10)
    receiver_name_entry = Entry(receiver_details_window, font=("Times New
Roman", 16))
    receiver_name_entry.grid(row=0, column=1, padx=10, pady=10)

    age_label = Label(receiver_details_window, text="Age:", font=(None, 16))
    age_label.grid(row=1, column=0, padx=10, pady=10)
    age_entry = Entry(receiver_details_window, font=("Times New Roman", 16))
    age_entry.grid(row=1, column=1, padx=10, pady=10)

    gender_label = Label(receiver_details_window, text="Gender:", font=(None,
16))
    gender_label.grid(row=2, column=0, padx=10, pady=10)
    gender_var = StringVar(receiver_details_window)
    gender_var.set("Male")
    gender_options = ["Male", "Female", "Other"]
    gender_dropdown = OptionMenu(receiver_details_window, gender_var,
*gender_options)
    gender_dropdown.config(font=(None, 16))
    gender_dropdown.grid(row=2, column=1, padx=10, pady=10)

    address_label = Label(receiver_details_window, text="Address:", font=("Times
New Roman", 16))
    address_label.grid(row=3, column=0, padx=10, pady=10)
    address_entry = Entry(receiver_details_window, font=("Times New Roman",
16))
    address_entry.grid(row=3, column=1, padx=10, pady=10)

    phone_number_label = Label(receiver_details_window, text="Phone
Number:", font=("Times New Roman", 16))
    phone_number_label.grid(row=4, column=0, padx=10, pady=10)
```

```
    phone_number_entry = Entry(receiver_details_window, font=("Times New
Roman", 16))
    phone_number_entry.grid(row=4, column=1, padx=10, pady=10)

    blood_group_label = Label(receiver_details_window, text="Blood Group:",
font=("Times New Roman", 16))
    blood_group_label.grid(row=5, column=0, padx=10, pady=10)
    blood_group_entry = Entry(receiver_details_window, font=("Times New
Roman", 16))
    blood_group_entry.grid(row=5, column=1, padx=10, pady=10)

    other_medical_details_label = Label(receiver_details_window, text="Other
Medical Details:", font=("Times New Roman", 16))
    other_medical_details_label.grid(row=6, column=0, padx=10, pady=10)
    other_medical_details_entry = Text(receiver_details_window, font=("Times
New Roman", 16), height=5, width=30)
    other_medical_details_entry.grid(row=6, column=1, padx=10, pady=10)

    # Create a button to save donor details
    save_button = Button(receiver_details_window, text="Save",
command=save_receiver_details, font=("Times New Roman", 16))
    save_button.grid(row=7, columnspan=2, pady=10)

    # Run the Donater Details window's main loop
    receiver_details_window.mainloop()

def receiver():
    create_receiver_details_page()


def logout():
    messagebox.showinfo(title="Logout", message="You have been logged out.")

# Create the main window for the login page
root = Tk()
root.title("Login Page")
# Get screen width and height
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()

# Set the geometry of the window to cover the entire screen
root.geometry(f"{screen_width}x{screen_height}")

# Load background image for login page
login_background_image = Image.open("bg2.jpg")
```

```
login_background_image = login_background_image.resize((screen_width,
screen_height))
login_background_photo = ImageTk.PhotoImage(login_background_image)
# Create a label for the login page title
# Create a label to display the background image
login_background_label = Label(root, image=login_background_photo)
login_background_label.place(x=0, y=0, relwidth=1, relheight=1)

# Create a login frame
login_frame = Frame(root, bg="white")
login_frame.place(relx=0.65, rely=0.5, anchor=CENTER)
login_title_label = Label(root, text=" DONATE4LIFE", font=("Helvetica", 30,),
fg="red")
login_title_label.pack(side=TOP, fill=X)
# Create elements inside the login frame
username_label = Label(login_frame, text="Username:", font=(None, 16))
username_label.grid(row=0, column=0)

username_entry = Entry(login_frame, font=(None, 16), width=30)
username_entry.grid(row=0, column=1, padx=5, pady=5)

password_label = Label(login_frame, text="Password:", font=(None, 16))
password_label.grid(row=1, column=0)

password_entry = Entry(login_frame, font=(None, 16), show="*", width=30)
password_entry.grid(row=1, column=1, padx=5, pady=5)

login_button = Button(login_frame, text="Login", command=login,
font=("Helvetica", 16))
login_button.grid(row=2, columnspan=2, pady=10)

# Run the main loop for the login page
root.mainloop()
```

MYSQL:

```
create database blooddonation;
use blooddonation;
CREATE TABLE donors (
  donor_id INT AUTO_INCREMENT PRIMARY KEY,
  donor_name VARCHAR(255) NOT NULL,
  age INT,  -- Adjust data type if needed (e.g., TINYINT for ages 0-127)
  gender VARCHAR(10),
  address VARCHAR(255) NOT NULL,
  phone_number VARCHAR(20) NOT NULL,
  blood_group VARCHAR(10) NOT NULL,
```

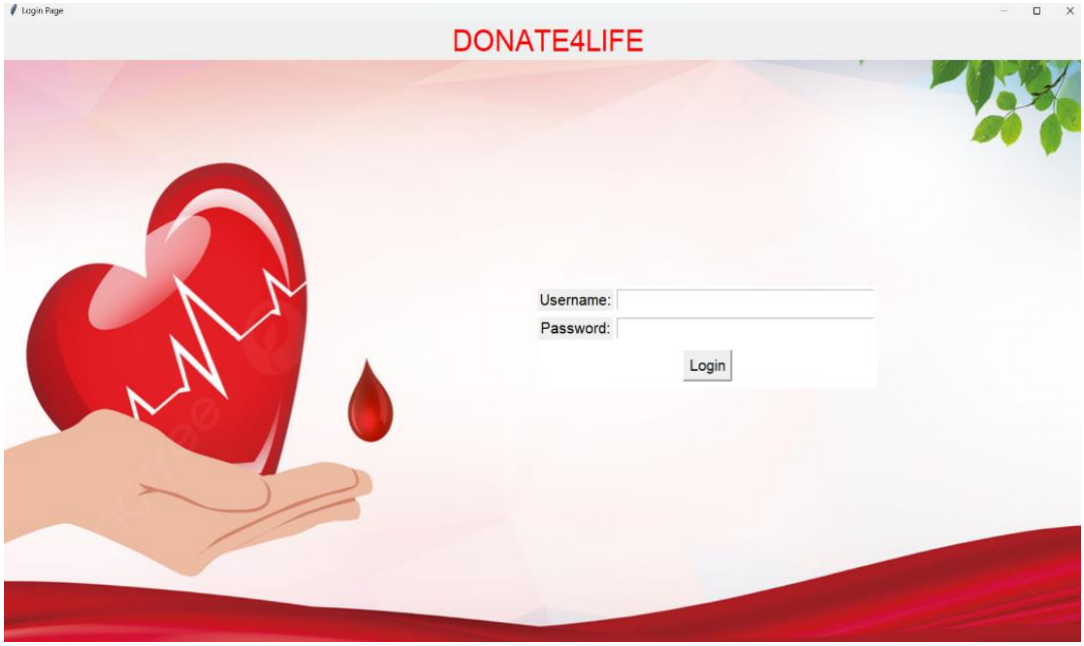
```
  other_medical_details TEXT
);



DELIMITER //
CREATE PROCEDURE delete_donor_id(IN donor_id INT)
BEGIN
    DELETE FROM donors WHERE donors.donor_id = donor_id;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE update_donor_id(
    IN p_donor_id INT,
    IN p_donor_name VARCHAR(255),
    IN p_age INT,
    IN p_gender VARCHAR(50),
    IN p_address VARCHAR(255),
    IN p_phone_number VARCHAR(20),
    IN p_blood_group VARCHAR(10)

)
BEGIN
    UPDATE donors
    SET
        donor_name = p_donor_name,
        age = p_age,
        gender = p_gender,
        address = p_address,
        phone_number = p_phone_number,
        blood_group = p_blood_group

    WHERE donor_id = p_donor_id;
END //
DELIMITER ;

CREATE TABLE receiver (
  receiver_id INT AUTO_INCREMENT PRIMARY KEY,
  receiver_name VARCHAR(255) NOT NULL,
  age INT,  -- Adjust data type if needed (e.g., TINYINT for ages 0-127)
  gender VARCHAR(10),
  address VARCHAR(255) NOT NULL,
  phone_number VARCHAR(20) NOT NULL,
  blood_group VARCHAR(10) NOT NULL,
```

```
  other_medical_details TEXT
);

DELIMITER //
CREATE PROCEDURE delete_receiver_id(IN receiver_id INT)
BEGIN
   DELETE FROM receiver WHERE receiver.receiver_id = receiver_id;
END //
DELIMITER ;

DELIMITER //
CREATE PROCEDURE update_receiver_id(
   IN p_receiver_id INT,
   IN p_receiver_name VARCHAR(255),
   IN p_age INT,
   IN p_gender VARCHAR(50),
   IN p_address VARCHAR(255),
   IN p_phone_number VARCHAR(20),
   IN p_blood_group VARCHAR(10)

)
BEGIN
   UPDATE receiver
   SET
      receiver_name = p_receiver_name,
      age = p_age,
      gender = p_gender,
      address = p_address,
      phone_number = p_phone_number,
      blood_group = p_blood_group

   WHERE receiver_id = p_receiver_id;
END //
DELIMITER ;

select * from receiver;
```

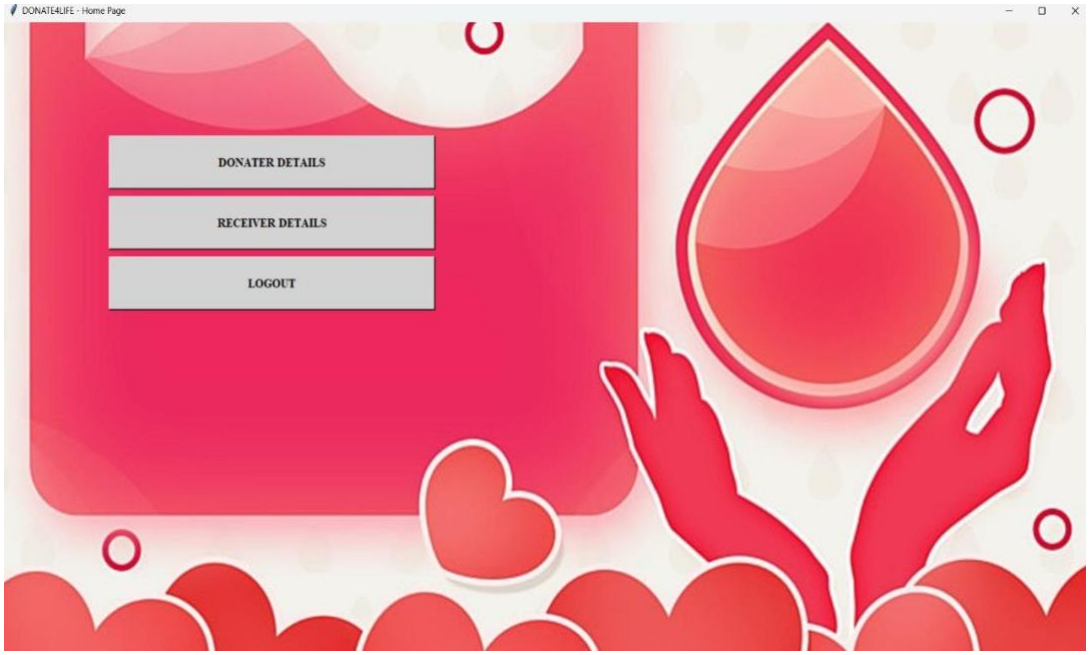
# Ch 5. RESULT AND DISCUSSION

## 5.1 USER DOCUMENTATION

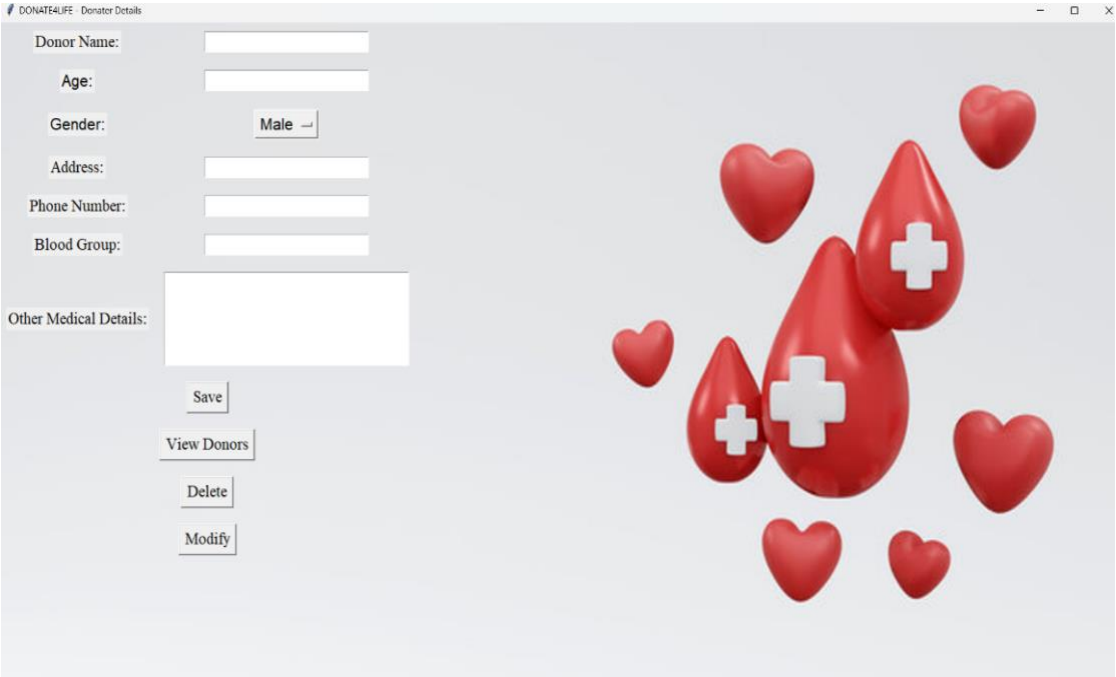
## LOGIN PAGE



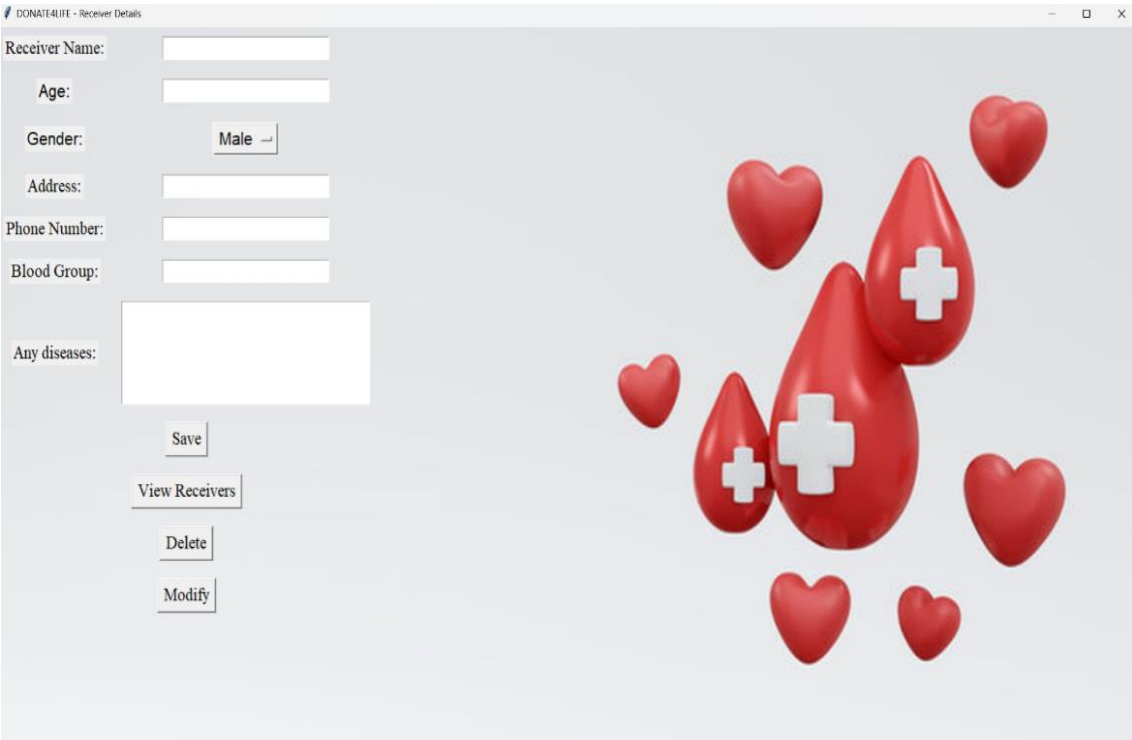

## HOME PAGE



# DONOR DETAILS




# RECEIVER DETAILS



# DONORS TABLE

DONATE4LIFE - Donors Table

| ID | Donor Name | Age | Gender | Address | Phone Number | Blood Group | Other Medical Details |
|---|---|---|---|---|---|---|---|
| 6 | Susila | 45 | Female | Perumal Street | 9488207171 | A1 | no |
| 7 | Snekha | 18 | Female | Egmore,Chennai | 9444568216 | A1 | no |

# RECEIVERS TABLE

DONATE4LIFE - Receiver Table

| ID | Receiver Name | Age | Gender | Address | Phone Number | Blood Group | Other Medical Details |
|---|---|---|---|---|---|---|---|
| 1 | Ramkumar K R | 51 | Male | Perumal Street,Egmore | 9488207171 | O+ve | NO |
| 2 | Swetha R | 21 | Female | Chennai | 9498349520 | A1 | no |

# Ch 6. TESTING

## 6.1 Unit Testing

Unit Testing is a crucial step in software development, testing individual modules like login and database connections for reliability and fitness of use. Helps identify and fix issues early.

## 6.2 Integration Testing

Integration Testing combines individually tested modules into a unified system. Focuses on seamless interaction of modules like login and database connections. Ensures smooth functioning and meets requirements.

## 6.3 System Testing

System testing evaluates the entire blood donation management system to ensure it meets requirements. The software is tested on various systems to identify and fix errors or bugs. This phase ensures smooth operation across different environments and user interactions, verifying the system's reliability before deployment.

## 6.4 Acceptance Testing

Acceptance Testing involves the client certifying that the blood donation management system meets agreed-upon requirements before deployment.

# Ch 7. CONCLUSION

## 7.1 Conclusion

After completing the blood donation management system project, we are confident that it will effectively address the challenges present in the existing system. The system has been computerized to mitigate human errors and enhance efficiency. Our primary objective is to minimize human effort throughout the blood donation process.

By storing all records in a centralized database, we have streamlined record maintenance and facilitated easy data retrieval. Navigation controls have been implemented in all forms to simplify the browsing of extensive record sets. Users can swiftly search for specific records by entering search strings, ensuring prompt access to information. Editing records has also been simplified, requiring users to input the necessary fields and update them with ease.

Each book and student in the system is assigned a unique ID for accurate and error-free access. Our project's core goal is to provide accurate information about individual donors, recipients, and available blood units, thereby supporting efficient blood management processes.