Blood bank management system

Project members:

- Ganna Eslam **(sec4)**
- Ganna Mahmoud **(sec4)**
- Habiba Mohammed Ahmed **(sec5)**
- Ayat Bahii Ahmed **(sec4)**
- Omnia Magdy Ibrahim **(sec3)**
- Habiba Alrehman Ayman **(sec5)**
- Basmala Ahmed Atta **(sec4)**

Table of Contents:

Abstract

With the increasing of the population of and revolution of the new technologies, Blood Bank Management System plays an important role in the blood bank as blood is a necessity to everyone.

This proposed system of the Blood Bank Management System intends to simplify and automate the process of searching for blood in case of emergency and maintain the records of blood donors, and blood stocks in the bank.

Introduction:

The blood bank management system is important for users to store and update donor data (name, id, father name, mother name, telephone number, gender, E-mail, address, date of birth) and know the amount of blood available for each type.

We created this system to solve many problems.

Problems:

- Routine procedures take a long time.
- Difficulty in reaching a suitable donor in cases of emergencies and accidents.
- Continuous changes in blood quantities are difficult to document on paper, especially at times when donations and withdrawals increase.
- Changing donor data may cause some errors and confusion between old and new data.

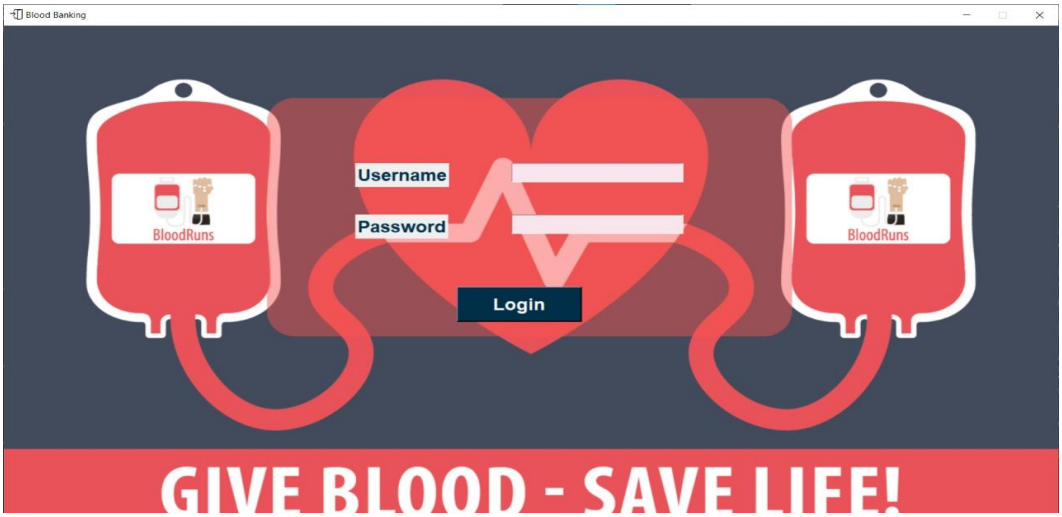
GUI Frames:

We have created several frames in our system.

Frame 1:

First, we created a **login** framework:

In this frame, the owner of the system is asked for the username and password, and if the password is correct, he is transferred to the next frame. If he is wrong, an error message appears in the password or username.



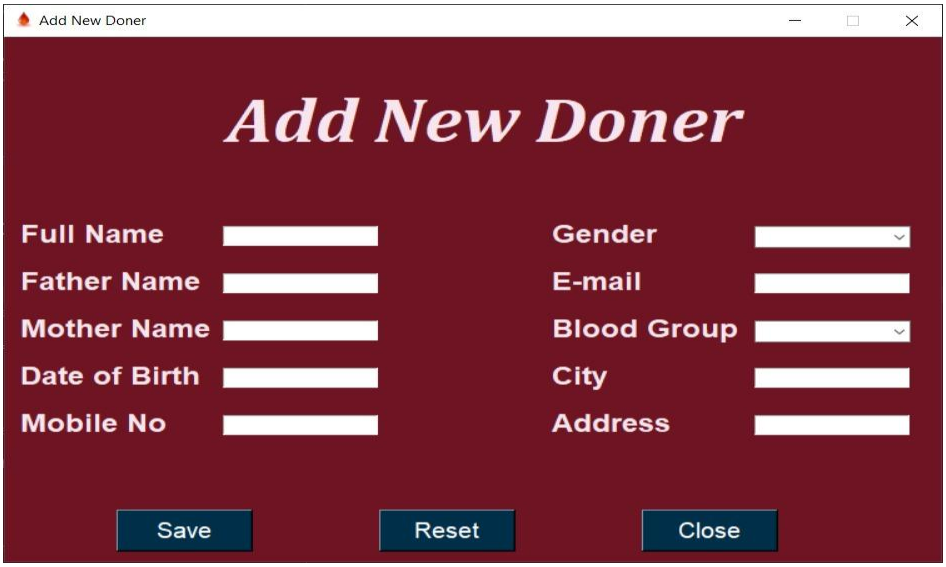
Frame 2:

Then we transfer to the **home** framework: this frame contains a list of each of its items (donor, search blood donor, stock, delete donor, exit) has many options.

In the first item donor there are 3 choices (add new, update details, all donor details).

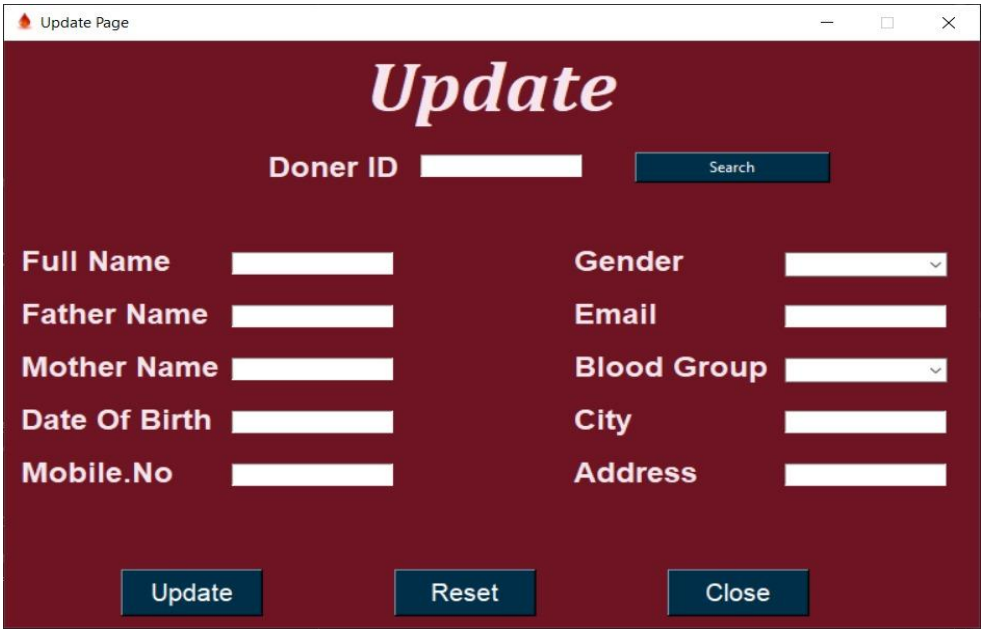
Frame 3:

If the first choice (add new) is selected, we transfer to **add new Donor** framework: this frame allows the users to store the new donor details (name,, father name, mother name, telephone number, gender, E-mail, address, date of birth). As soon as a donor is registered, a unique identification number is assigned to him/her automatically.



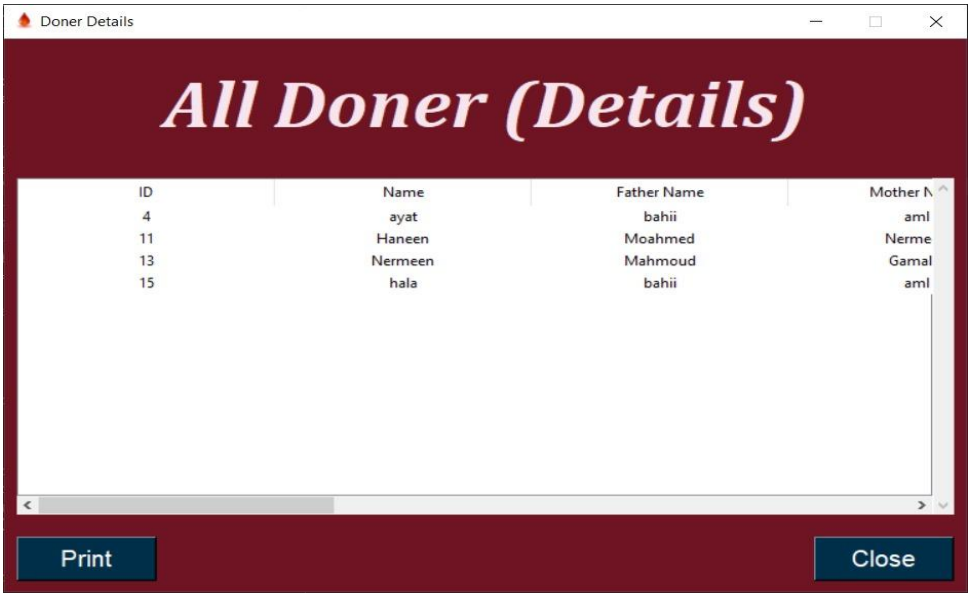
Frame 4:

If the second choice (update details) is selected, we transfer to **update details** of donor framework: If you just choice the donor's ID and then search, all the details will appear, this frame allows the users to update the donor details.

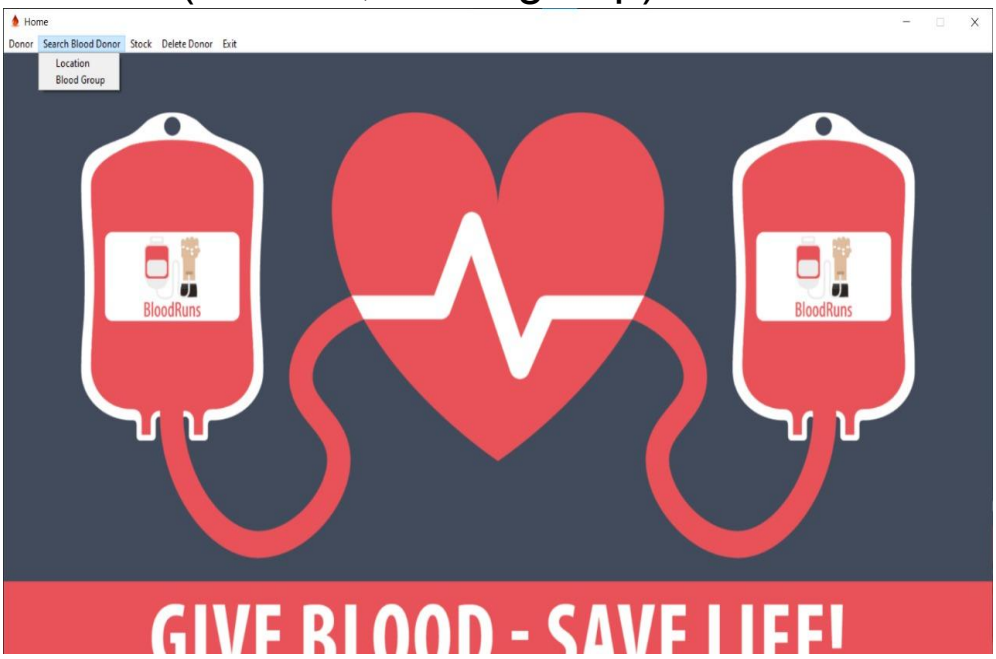


Frame 5:

If the last choice (all donor details) is selected, we transfer to **all donor details** framework: this frame shows all donors details in a table.



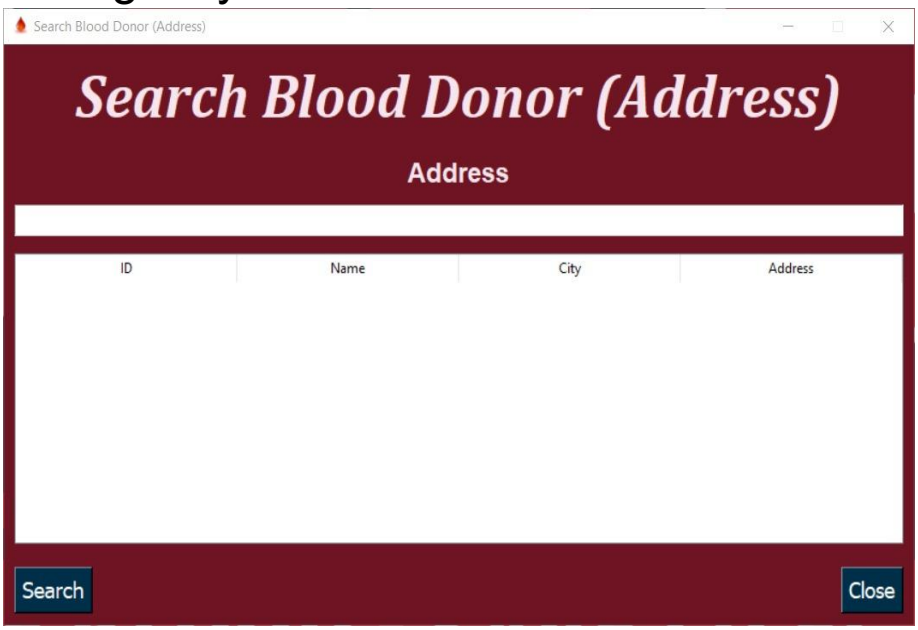
In the second item (search blood donor) there are 2 choices (location, blood group).



Frame 6:

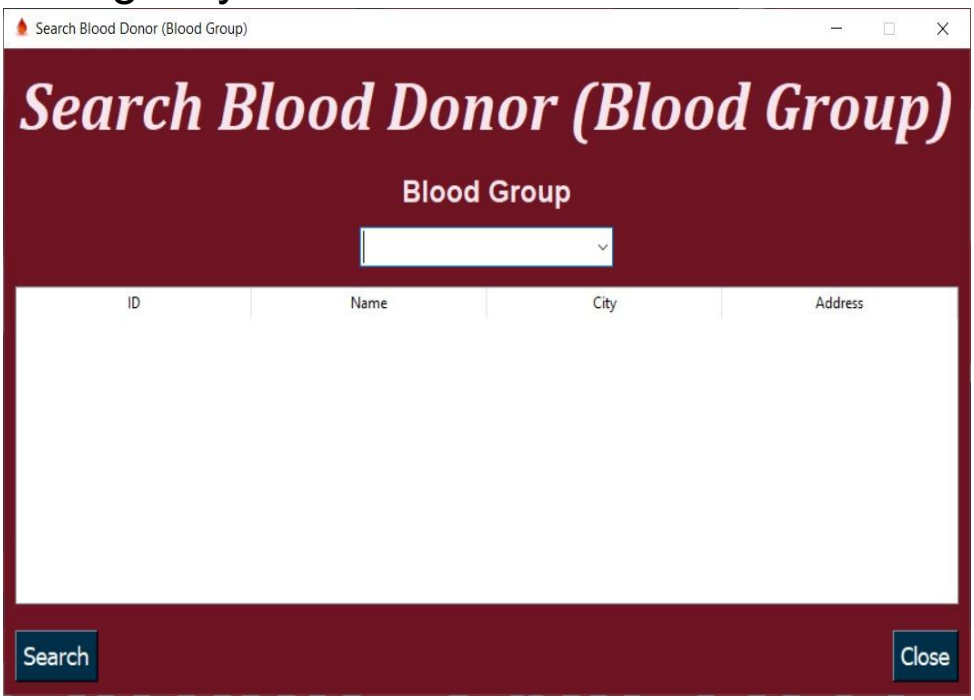
If the first choice (location) is selected, we transfer to **location** framework:

In this frame, it is allowed to search for donors by writing the city or address or part of it only, it shows all data of suitable donors in a table, and this facilitates the search for the appropriate donor in emergency situations.



Frame 7:

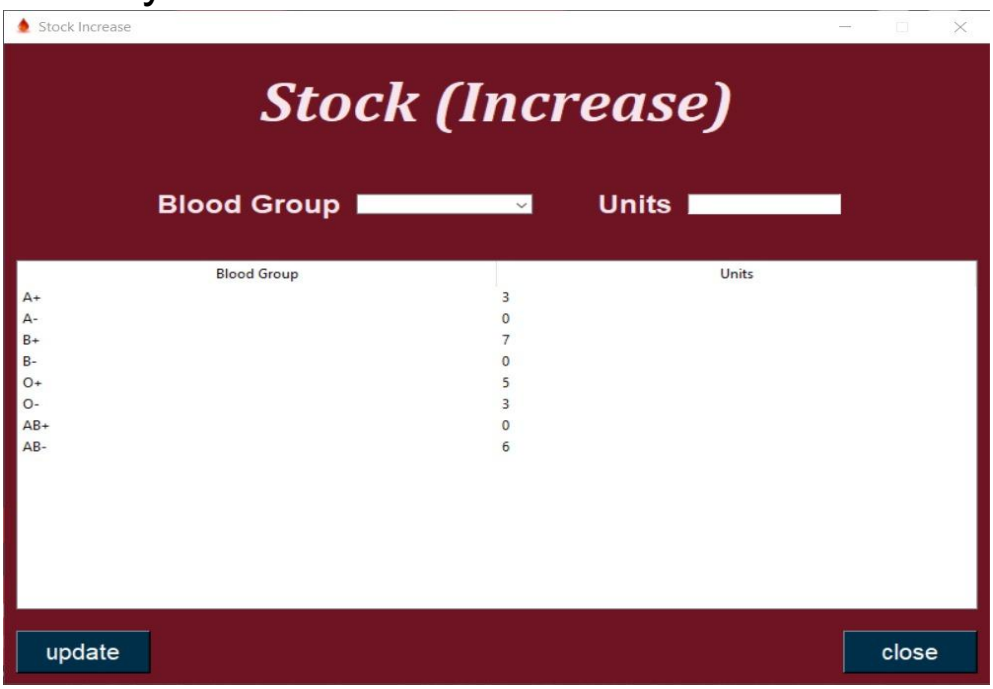
If the second choice (blood group) is selected, we transfer to **blood group** framework: In this frame, it is allowed to search for donors by writing the blood type (A+, A-, B+, B-, AB+, AB-, O+, O-), and this facilitates the search for the appropriate donor in emergency situations.



In the third item (stock) there are 3 choices (increase, decrease, details).

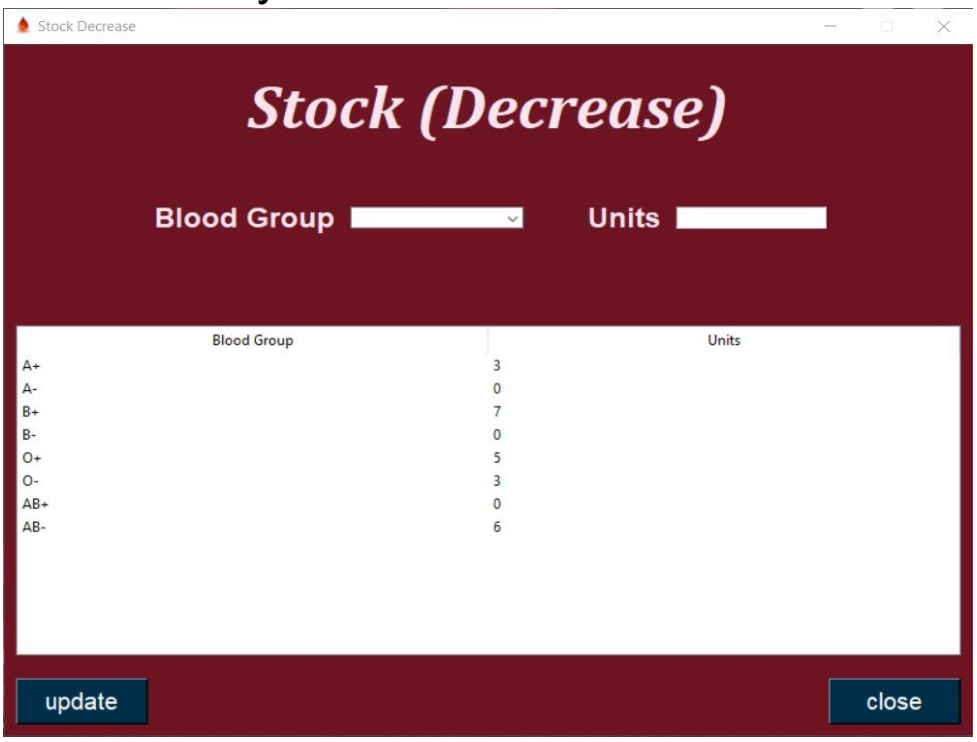
Frame 8:

If the first choice (increase) is selected, we transfer to **Stock Increase** framework: In this frame, it is allowed to increase the number of liters of blood by choosing the blood type and writing the number of liters that were added, thus increasing the number of liters actually stored in the table.



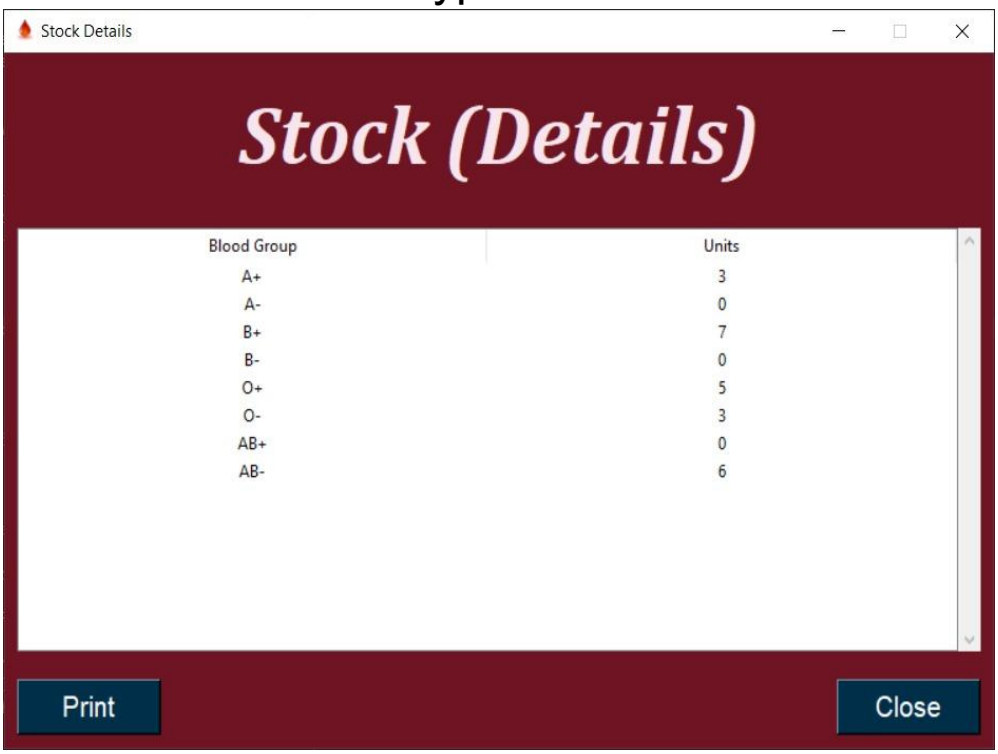
Frame 9:

If the second choice (decrease) is selected, we transfer to **Stock Decrease** framework: In this frame, it is allowed to decrease the number of liters of blood by choosing the blood type and writing the number of liters that were taken, thus decreasing the number of liters actually stored in the table.



Frame 10:

If the last choice (details) is selected, we transfer to **stock Details** framework: In this frame, a table appears containing the blood type and the number of liters of each blood type.

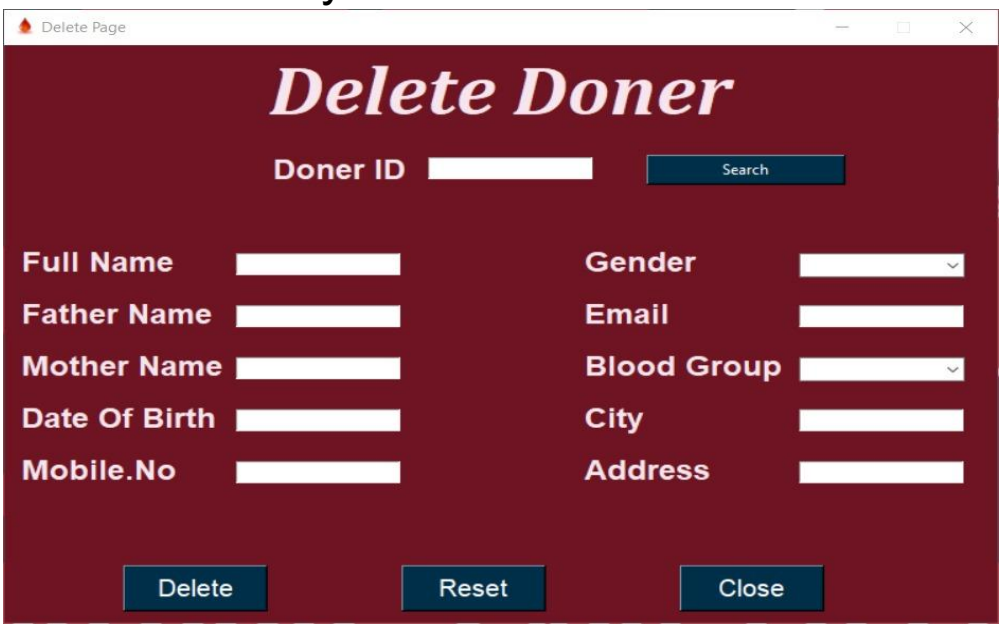


In the fourth item (delete donor) there is only 1choice (Delete donor).

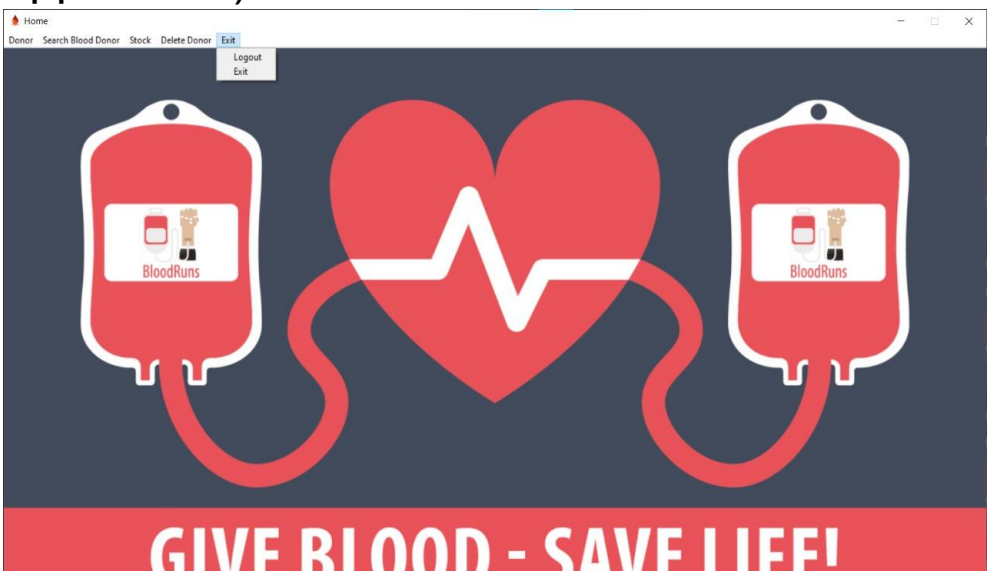
Frame 11:

When the only choice is selected, we transfer to **delete Donor** framework:

In this frame, we only search for the donor who wants to be deleted by ID and delete it.



In the last item (exit) there are 2 choices (Logout, Exit application).



If you click on exiting the program, a message appears stating the right to exit the program, and two options appear: Yes or No. If Yes is pressed, you exit the entire program, and if No option is chosen, it will not remain in the same window.

If the first choice (Logout) is selected, we transfer to login frame.

If the last choice (Exit application) is selected, a message appears: Do you really want to exit the program, and two options appear: Yes and No. If Yes is chosen, you exit the entire program, and if No is chosen, it will not remain in the same frame.

Data base:

Two tables have been created,

the first for the **donor** data (ID , Name, Father name, Mother name, Mobile NO, Gender, E-mail, City , Address, Date Of Birth)

,And the second for **stock** (blood Group, units).

ERD:

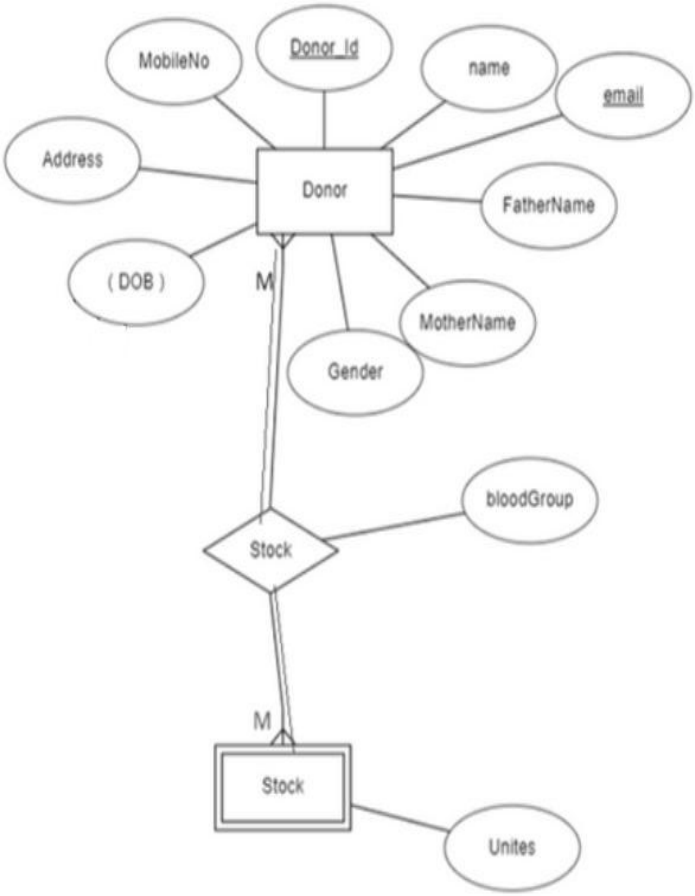


Table in database:

Tables (3)		
Stock		CREATE TABLE "Stock" ("BloodGroup" TEXT, "Units" INTEGER)
BloodGroup	TEXT	"BloodGroup" TEXT
Units	INTEGER	"Units" INTEGER
doner		CREATE TABLE "doner" ("DonerID" INTEGER, "Name" TEXT, "FatherName" TEXT, "MotherName" TEXT, "DOB" TEXT
DonerID	INTEGER	"DonerID" INTEGER
Name	TEXT	"Name" TEXT
FatherName	TEXT	"FatherName" TEXT
MotherName	TEXT	"MotherName" TEXT
DOB	TEXT	"DOB" TEXT
MobileNo	TEXT	"MobileNo" TEXT
Gender	TEXT	"Gender" TEXT
Email	TEXT	"Email" TEXT
BloodGroup	TEXT	"BloodGroup" TEXT
City	TEXT	"City" TEXT
Address	TEXT	"Address" TEXT
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
Indices (0)		
Views (0)		
Triggers (0)		

Connection:

In each frame we created function to connect between GUI and database

In **third frame** we create a save function to connect with database to can insert new doners with all him/her details

```
def save_doner():
    full_name = str(name_ent.get())
    father_name = str(father_ent.get())
    mother_name = str(mother_ent.get())
    date_birth = str(date_ent.get())
    mobile = str(mobile_ent.get())
    gender = str(gender_ent.get())
    mail = str(mail_ent.get())
    blood_g = str(blood.get())
    city = str(city_ent.get())
    address = str(address_txt.get())


    conn = sqlite3.connect("BloodBank.db")
    cur = conn.cursor()
    cur.execute("INSERT INTO doner (name , FatherName , MotherName , dob , MobileNo , Gender , Email , BloodGroup , city , Address) VALUES (?
    conn.commit()
    conn.close()
    messagebox.showinfo("Success", "Successfully Added")
    reset_form()
```

In the **fourth frame** we open new connection to return all IDs in our database to show in ID combo box for search

```
conn = sqlite3.connect('BloodBank.db')
cur = conn.cursor()

cur.execute("SELECT DonerID FROM doner")

records = cur.fetchall()

conn.close()
```

And create another function in **the same frame** to know search for doners in database based on their IDs and show their all data

```
def SearchFun():
    ID = entDonerID.get()
    if ID :
        conn = sqlite3.connect("BloodBank.db")
        cur = conn.cursor()
        cur.execute("SELECT * from doner where DonerID = ?" , (ID,))
        raw = cur.fetchone()
        if raw:
            entFullName.delete(0,tk.END)
            entFatherName.delete(0,tk.END)
            entMotherName.delete(0,tk.END)
            entMobileNo.delete(0,tk.END)
            entCity.delete(0,tk.END)
            entAddr.delete(0,tk.END)
            entBG.delete(0,tk.END)
            entDOB.delete(0,tk.END)
            entEmail.delete(0,tk.END)
            entSex.delete(0,tk.END)

            entFullName.insert(0,raw[1])
            entFatherName.insert(0,raw[2])
            entMotherName.insert(0,raw[3])
            entMobileNo.insert(0,raw[5])
            entCity.insert(0,raw[9])
            entAddr.insert(0,raw[10])
            entBG.set(raw[8])
            entDOB.insert(0,raw[4])
            entEmail.insert(0,raw[7])
            entSex.set(raw[6])
        else:
            entFullName.delete(0,tk.END)
            entFatherName.delete(0,tk.END)
            entMotherName.delete(0,tk.END)
            entMobileNo.delete(0,tk.END)
            entCity.delete(0,tk.END)
            entAddr.delete(0,tk.END)
            entBG.delete(0,tk.END)
            entDOB.delete(0,tk.END)
            entEmail.delete(0,tk.END)
            entSex.delete(0,tk.END)

            tk.messagebox.showerror("Error", "Donor ID does not exist")
        conn.close()
```

The last function in this frame is Update Function , this takes all new data and update them in doner table in database

```
def UpdateFun():
    ID = entDonerID.get()
    if ID:
        fullName = entFullName.get()
        fatherName = entFatherName.get()
        motherName = entMotherName.get()
        dob = entDOB.get()
        mobileNo = entMobileNo.get()
        sex = entSex.get()
        email = entEmail.get()
        bloodGroup = entBG.get()
        city = entCity.get()
        address = entAddr.get()

        conn = sqlite3.connect("BloodBank.db")
        cur = conn.cursor()
        cur.execute("""
            UPDATE doner SET
            Name = ?,
            FatherName = ?,
            MotherName = ?,
            DOB = ?,
            MobileNo = ?,
            Gender = ?,
            Email = ?,
            BloodGroup = ?,
            City = ?,
            Address = ?
            WHERE DonerID = ?
        """, (fullName, fatherName, motherName, dob, mobileNo, sex, email, bloodGroup, city, address, ID))
        conn.commit()
        conn.close()
        tk.messagebox.showinfo("Success", "Record updated successfully")
```

In the **fifth frame** we create fetch_data function to return all doners have the same address

```
def fetch_data(address):
    conn = sqlite3.connect("bloodbank.db")
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM doner WHERE address LIKE ?",
                   ("%" + address + "%",))
    records = cursor.fetchall()
    conn.close()

    return records
```

In the **sixth frame** we create fetch_data function to return all doners have the same Blood Type

```
def fetch_data(blood_group):
    conn = sqlite3.connect('bloodbank.db')
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM doner WHERE bloodGroup LIKE ?",
                   ('%' + blood_group + '%',))
    records = cursor.fetchall()

    conn.close()

    return records
```

In the **seventh frame** we create function connect with database to can increase numbers of units of blood types

```python
def update_table():
    # Get selected values from Combobox and Entry
    blood_group = blood_group_combobox.get()
    units = units_entry.get()

    # Validate inputs
    if not blood_group or not units:
        messagebox.showerror("Input Error", "Please fill out both fields")
        return

    try:
        unit_int = int(units)
    except ValueError:
        messagebox.showerror("Input Error", "Units must be an integer")
        return

    con = sqlite3.connect("BloodBank.db")
    cursor = con.cursor()

    cursor.execute(
        "SELECT Units FROM stock WHERE BloodGroup = ?", (blood_group,))
    result = cursor.fetchone()

    if result:
        cursor.execute(
            "UPDATE stock SET Units = Units + ? WHERE BloodGroup = ?", (unit_int, blood_group))
        con.commit()
        messagebox.showinfo("Success", "Successfully updated")

        # Clear the inputs
        blood_group_combobox.set('')
        units_entry.delete(0, 'end')

        # Update the Treeview
        load_data()

    else:
        messagebox.showerror(
            "Input Error", "Blood group not found in stock")

    con.close()
```

In the **same frame** we create another function to show all data in the stock table after update in data base in tree view

```
def load_data():
    for item in StockIncrease_root.tree.get_children():
        StockIncrease_root.tree.delete(item)

    con = sqlite3.connect("BloodBank.db")
    cursor = con.cursor()
    cursor.execute("SELECT * FROM stock")
    rows = cursor.fetchall()
    con.close()

    for row in rows:
        StockIncrease_root.tree.insert("", tk.END, values=row)
```

In the **eighth frame** we create function connect with database to can decrease numbers of units of blood types

```python
def update_table():
    # Get selected values from Combobox and Entry
    blood_group = blood_group_combobox.get()
    units = units_entry.get()

    # Validate inputs
    if not blood_group or not units:
        messagebox.showerror("Input Error", "Please fill out both fields")
        return

    try:
        unit_int = int(units)
    except ValueError:
        messagebox.showerror("Input Error", "Units must be an integer")
        return

    con = sqlite3.connect("BloodBank.db")
    cursor = con.cursor()

    cursor.execute(
        "SELECT Units FROM stock WHERE BloodGroup = ?", (blood_group,))
    result = cursor.fetchone()

    if result:
        current_units = result[0]
        if unit_int > current_units:
            messagebox.showerror(
                "Input Error", "Units to decrease exceed current stock")
            con.close()
            return

        cursor.execute(
            "UPDATE stock SET Units = Units - ? WHERE BloodGroup = ?", (unit_int, blood_group))
        con.commit()
        messagebox.showinfo("Success", "Successfully updated")

        # Clear the inputs
        blood_group_combobox.set('')
        units_entry.delete(0, 'end')

        # Update the Treeview
        load_data()

    else:
        messagebox.showerror(
            "Input Error", "Blood group not found in stock")

    con.close()
```

And create another function In the **same frame** to show all data in the stock table after update in data base in tree view

```
def load_data():
    for item in StockDecrease_root.tree.get_children():
        StockDecrease_root.tree.delete(item)

    con = sqlite3.connect("BloodBank.db")
    cursor = con.cursor()
    cursor.execute("SELECT * FROM stock")
    rows = cursor.fetchall()
    con.close()

    for row in rows:
        StockDecrease_root.tree.insert("", tk.END, values=row)
```

In the **ninth frame** we create Print function to show all doner details in tree view

```
def Printfun():
    for item in tree.get_children():
        tree.delete(item)
    con = sqlite3.connect("BloodBank.db")
    cursor = con.cursor()
    cursor.execute("SELECT * FROM doner")
    rows = cursor.fetchall()
    con.close()

    for row in rows:
        tree.insert("", tk.END, values=row)
```

In the **tenth frame** we create Print function to show all stock details in tree view

```
def Printfun():
    for item in tree.get_children():
        tree.delete(item)
    con = sqlite3.connect("BloodBank.db")
    cursor = con.cursor()
    cursor.execute("SELECT * FROM stock")
    rows = cursor.fetchall()
    con.close()

    for row in rows:
        tree.insert("", tk.END, values=row)
```

In the **eleventh frame** we open new connection to return all IDs in our database to show in ID combo box for search

```
conn = sqlite3.connect('BloodBank.db')
cur = conn.cursor()

cur.execute("SELECT DonerID FROM doner")

records = cur.fetchall()

conn.close()
```

And create another function in **the same frame** to know search for doners in database based on their IDs and show their all data

```
def SearchFun():
    ID = entDonerID.get()
    if ID :
        conn = sqlite3.connect("BloodBank.db")
        cur = conn.cursor()
        cur.execute("SELECT * from doner where DonerID = ?" , (ID,))
        raw = cur.fetchone()
        if raw:
            entFullName.delete(0,tk.END)
            entFatherName.delete(0,tk.END)
            entMotherName.delete(0,tk.END)
            entMobileNo.delete(0,tk.END)
            entCity.delete(0,tk.END)
            entAddr.delete(0,tk.END)
            entBG.delete(0,tk.END)
            entDOB.delete(0,tk.END)
            entEmail.delete(0,tk.END)
            entSex.delete(0,tk.END)

            entFullName.insert(0,raw[1])
            entFatherName.insert(0,raw[2])
            entMotherName.insert(0,raw[3])
            entMobileNo.insert(0,raw[5])
            entCity.insert(0,raw[9])
            entAddr.insert(0,raw[10])
            entBG.set(raw[8])
            entDOB.insert(0,raw[4])
            entEmail.insert(0,raw[7])
            entSex.set(raw[6])
        else:
            entFullName.delete(0,tk.END)
            entFatherName.delete(0,tk.END)
            entMotherName.delete(0,tk.END)
            entMobileNo.delete(0,tk.END)
            entCity.delete(0,tk.END)
            entAddr.delete(0,tk.END)
            entBG.delete(0,tk.END)
            entDOB.delete(0,tk.END)
            entEmail.delete(0,tk.END)
            entSex.delete(0,tk.END)

            tk.messagebox.showerror("Error", "Donor ID does not exist")
        conn.close()
```

The last function in this frame is Delete function , this function create to delete doners from database based on their IDs

```
def DeleteFun():
    ID = entDonerID.get()
    conn = sqlite3.connect("BloodBank.db")  # Update with your database path
    cur = conn.cursor()
    if ID:
        cur.execute("DELETE FROM doner WHERE DonerID = ?", (ID,))
        conn.commit()
        conn.close()
        messagebox.showinfo("Success", "Successfully Deleted")
        ResetFun()
    else:
        tk.messagebox.showerror("Error", "Donor ID does not exist")
```

Programs and libraries:

- Tkinter : is a standard GUI (Graphical User Interface) toolkit in Python, which provides tools for creating desktop applications with graphical interfaces.

- Sqllite3 : it is in Python provides a lightweight, disk-based database that doesn’t require a separate server process.

- PIL : is a powerful library for image processing tasks in Python.