

**Sri Sivasubramaniya Nadar College of Engineering
Kalavakkam – 603 110
(An Autonomous Institution, Affiliated to Anna University, Chennai)**

Department of Information Technology

UIT1312 – Database Management Systems and Applications Lab

Mini Project – 2021-2022

III SEMESTER IT – INSTANCE A

BLOOD DONATION MANAGEMENT SYSTEM

Register number 205002059 – Parthiban D

1. Project Title: Blood Donation Management System

2. Problem Statement:

Develop a Blood Donation Management System to manage the blood donation details and to provide the blood receiver with the required details. Assign unique IDs to the donors and receivers and store the relevant information under the same. You'll have to add the donor's name, age, address, blood group and diseases if infected. You'll also have to mention which blood bank the required blood group is available. The blood bank obtains and provides information about the blood group to the receivers.

3. Project Overview:

Blood Donation Management System (BDMS) is a database system to link between the donors and blood banks and act as an interface for the patient to find his/her desired blood in a fast and efficient way. It will make the blood transfusion service and its management more reliable and efficient than the conventional system.

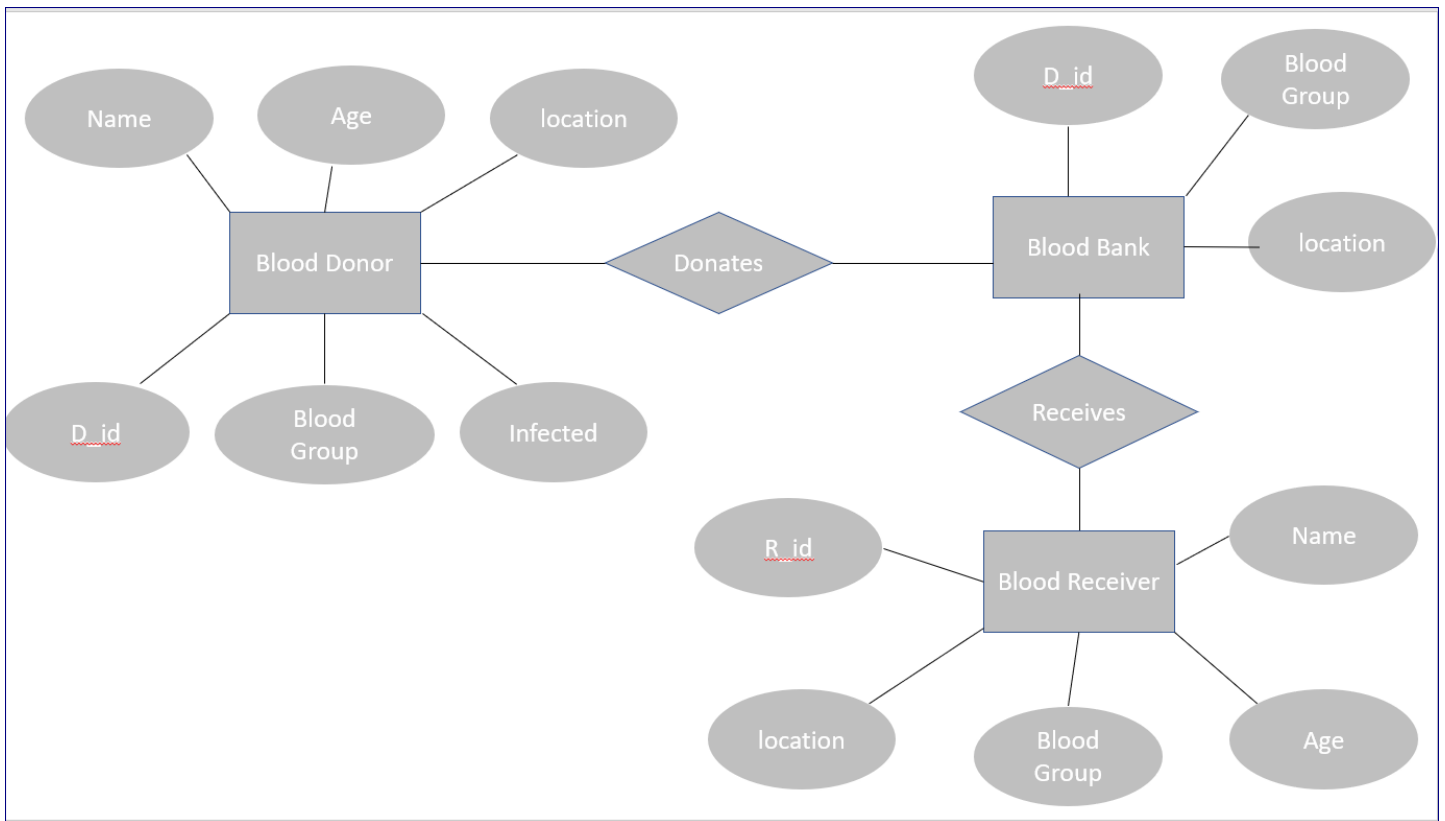
Transfusion of blood and blood components is an established standard way of treating patients who are deficient in one or more blood constituents and is therefore an essential part of health care. A blood transfusion service is a complex organization requiring careful design and management. Essential functions of a blood transfusion service are donor recruitment, blood collection, testing of donor blood, component preparation and supply of these components to the patients.

Alerts for blood requirement from registered donors. The main goal of the Blood Bank and Donor Management System project is to monitor Blood Bank data, Donor List.

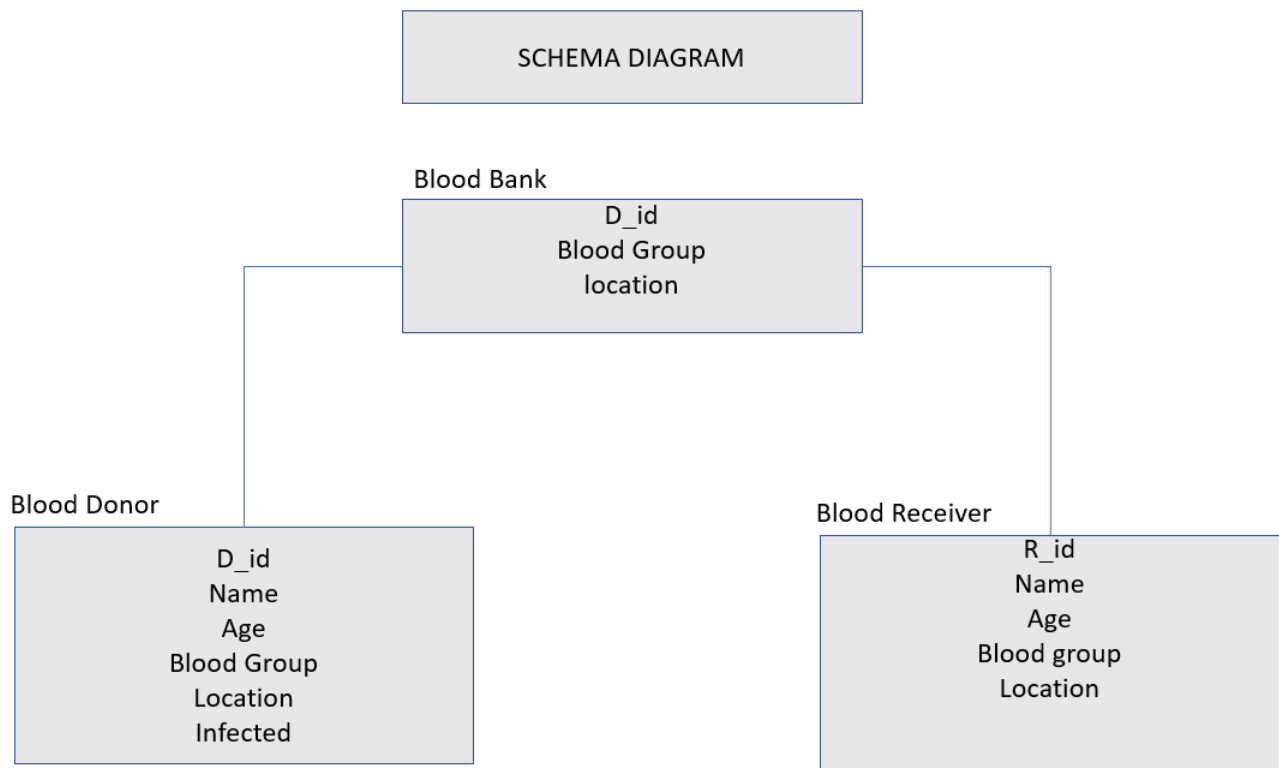
The organization of a blood transfusion service should receive utmost attention and care for smooth functioning of various components of the service. The goal of

blood transfusion service is to provide effective blood and blood components which are as safe as possible and adequate to meet the patients' needs.

2. ER diagram



3. Database schema



4. Database Normalization

Blood donor

D_id	Name	Age	Blood Group	location	Infected
------	------	-----	-------------	----------	----------

1NF:

There is no multivalued attributes in the table. So there is no need to do 1NF.

2NF:

There is no partial dependency. So there is no need to do 2NF.

3NF:

There is no Transitive dependency. So there is no need to do 3NF.

Blood Bank

D_id	Blood Group	location
------	-------------	----------

1NF:

There is no multivalued attributes in the table. So there is no need to do 1NF.

2NF:

There is no partial dependency. So there is no need to do 2NF.

3NF:

There is no Transitive dependency. So there is no need to do 3NF.

Blood Receiver

R_id	Name	Age	Blood Group	location
------	------	-----	-------------	----------

1NF:

There is no multivalued attributes in the table. So there is no need to do 1NF

2NF:

There is no partial dependency. So there is no need to do 2NF.

3NF:

There is no Transitive dependency. So there is no need to do 3NF.

5. Coding & Implementation

```
from tkinter import *
from tkinter import ttk
import tkinter.font as font
from tkinter import messagebox
from PIL import ImageTk, Image

import sqlite3
class DataBase:

    def __init__(self,db):
        self.con=sqlite3.connect(db)
        self.curr=self.con.cursor()
        table1=""" create table if not exists
BloodDonor(D_ID integer primary key autoincrement,
            DonorName text,Age int,BloodGroup text,Addr
text,Infected text)"""
        self.curr.execute(table1)
        table2= """ create table if not exists
BBank(D_ID integer primary key autoincrement,
            BloodGroup text,Addr text)"""
        self.curr.execute(table2)
        table3= """ create table if not exists
BloodReceiver(R_ID integer primary key autoincrement,
            DonorName text,Age int,BloodGroup
text,Addr text)"""
        self.curr.execute(table3)
        self.con.commit()

    def insertDonor(self,Dname,age,Bgrp,adr,Dis):
        if(str.upper(Dis)=='NO'):
            self.curr.execute("insert into BloodDonor
values(null,?,?,?,?,?)",
                               (Dname, age, Bgrp, adr,
```

```

Dis))

        self.curr.execute("insert into BBank
values(null,?,?)",

                                (Bgrp, adr))

        self.con.commit()
        return True
    self.con.commit()
    return False

def BloodB(self,BGroup,adr):
    self.curr.execute("insert into BBank
values(?,?)",

                                (BGroup,adr))
    self.con.commit()

def insertReceive(self,Rname,age,Bgroup,ad):
    Bgr = str.upper(Bgroup)
    adr = str.upper(ad)
    self.curr.execute("select count(*) from BBank
where upper(BloodGroup)=? and upper(Addr)=?", (Bgr,adr))
    cur_result =self.curr.fetchall()
    if(cur_result[0][0]==0):
        self.con.commit()
        return False
    else:
        self.curr.execute("insert into
BloodReceiver values(NULL,?,?,?,?)",

                                (Rname,age,Bgroup,ad))
        self.con.commit()
        return cur_result[0][0]

def fetchdonor(self):
    self.curr.execute("select * from BloodDonor")
    rows1=self.curr.fetchall()
    return rows1

```

```

def fetchBank(self):
    self.curr.execute("select * from BBank")
    rows2= self.curr.fetchall()
    return rows2

def fetchReciever(self):
    self.curr.execute("select * from
BloodReceiver")
    row = self.curr.fetchall()
    return row

def DelDonor(self,DoId):
    self.curr.execute("select count(*) from
BloodDonor where D_ID=?", (DoId))
    cur_result = self.curr.fetchall()
    if (cur_result[0][0] == 1):
        self.curr.execute("Delete from BloodDonor
where D_ID=?", (DoId))
        self.curr.execute("Delete from BBank where
D_ID=?", (DoId))
        self.con.commit()
        return True
    else:
        self.con.commit()
        return False

def DelReceiver(self,RoId):
    self.curr.execute("select count(*) from
BloodReceiver where R_ID=?", (RoId))
    cur_result = self.curr.fetchall()
    if (cur_result[0][0] == 1):
        self.curr.execute("Delete from
BloodReceiver where R_ID=?", (RoId))
        self.con.commit()
        return True
    else:

```



```
        self.con.commit()
        return False
```

```
def InsertDonor():
    if (Etd1.get() == "" or Etd2.get()==" " or
Etd3.get()==" " or Etd4.get()==" " or Etd5.get()==" "):
        messagebox.showerror("Error","Please fill all
the details")
        return
```

```
res=db.insertDonor(Etd1.get(),Etd2.get(),Etd3.get(),Etd
4.get(),Etd5.get())
    if(res==False):
        messagebox.showerror("Error", "Not allowed to
Donate Blood")
        return
    messagebox.showinfo(screen1,"Blood Donated")
```

```
def BloodDonarDetails():
    global Etd1,Etd2,Etd3,Etd4,screen1,Etd5,imgg
    screen1=Toplevel(root)
    screen1.title("Enter Details")
    screen1.geometry("1920x1080+0+0")
    labl2 = Label(screen1, image=imgg3)
    labl2.place(x=0, y=0, relwidth=1, relheight=1)
    lbs1 = Label(screen1, text="ENTER THE DETAILS",
width=600, height=2, background='Black', fg='White',
    font=("Times New Roman", 22, 'bold'))
    lbs1.pack()
    name=StringVar()
    Age=StringVar()
    Bgroup=StringVar()
    Addr=StringVar()
```

```

dis=StringVar()

lbd1=Label(screen1,text="NAME",background='black',fg='White', font=("Times New Roman",16,'bold'))
lbd1.place(x="240", y="120")
Etd1 = Entry(screen1,textvariable=name,
width=30,font='Helvetica 13')
Etd1.place(x="320", y="124", height="24")
lbd2 =
Label(screen1,text="AGE",background='black',fg='White',
font=("Times New Roman",16,'bold'))
lbd2.place(x="680", y="120")
Etd2 = Entry(screen1,textvariable=Age,
width=26,font='Helvetica 13')
Etd2.place(x="740", y="124", height="24")
lbd3 = Label(screen1, text="BLOOD
GROUP",background='black',fg='White', font=("Times New
Roman",16,'bold'))
lbd3.place(x="512", y="220")
Etd3 = Entry(screen1, textvariable=Bgroup,
width=26,font='Helvetica 13')
Etd3.place(x="480", y="262", height="24")
lbd4 = Label(screen1,
text="LOCATION",background='black',fg='White',
font=("Times New Roman",16,'bold'))
lbd4.place(x="534", y="310")
Etd4 = Entry(screen1, textvariable=Addr,
width=44,font='Helvetica 13')
Etd4.place(x="400", y="350", height="40")
lbd5 = Label(screen1,
text="INFECTED",background='black',fg='White',
font=("Times New Roman",16,'bold'))
lbd5.place(x="540", y="410")
Etd5 = Entry(screen1, textvariable=dis,
width=26,font='Helvetica 13')

```

```

Etd5.place(x="480", y="460", height="24")

btd=Button(screen1,text="SUBMIT",width=12,height=2,bg="#808080",fg="#fdfff5",command=InsertDonor)
    btd.place(x="524", y="530")
    BFont = font.Font(family="Times New Roman",
size=14, weight='bold')
    btd['font']=BFont

def InsertReciever():
    if (Etr1.get() == "" or Etr2.get()==" " or
Etr3.get()==" " or Etr4.get()==" "):
        messagebox.showerror("Error","Please fill all
the details")
        return

r=db.insertReceive(Etr1.get(),Etr2.get(),Etr3.get(),Etr
4.get())
    if(r==False):
        messagebox.showinfo(screen2, "Blood Not
available")
    else:
        messagebox.showinfo(screen2,f"Blood Request
Accepted, No of people with this blood group {r}")

def BloodRequestDetails():
    global Etr1, Etr2, Etr3, Etr4, screen2
    screen2 = Toplevel(root)
    screen2.title("Enter Details")
    screen2.geometry("1920x1080+0+0")
    labl2 = Label(screen2, image=imgg3)
    labl2.place(x=0, y=0, relwidth=1, relheight=1)
    lbs1 = Label(screen2, text="ENTER THE DETAILS",
width=600, height=2, background='Black',

```

```

fg='White',font=("Times New Roman", 22, 'bold'))
    lbs1.pack()
    name = StringVar()
    Age = StringVar()
    Bgroup = StringVar()
    Addr = StringVar()
    lbr1 = Label(screen2,
text="NAME",background='black',fg='White', font=("Times
New Roman",16,'bold'))
    lbr1.place(x="240", y="120")
    Etr1 = Entry(screen2, textvariable=name,
width=26,font='Helvetica 13')
    Etr1.place(x="320", y="124", height="24")
    lbr2 = Label(screen2,
text="AGE",background='black',fg='White', font=("Times
New Roman",16,'bold'))
    lbr2.place(x="680", y="120")
    Etr2 = Entry(screen2, textvariable=Age,
width=26,font='Helvetica 13')
    Etr2.place(x="740", y="125", height="24")
    lbr3 = Label(screen2, text="BLOOD
GROUP",background='black',fg='White', font=("Times New
Roman",16,'bold'))
    lbr3.place(x="520", y="220")
    Etr3 = Entry(screen2, textvariable=Bgroup,
width=26,font='Helvetica 13')
    Etr3.place(x="480", y="262", height="24")
    lbr4 = Label(screen2,
text="LOCATION",background='black',fg='White',
font=("Times New Roman",16,'bold'))
    lbr4.place(x="545", y="310")
    Etr4 = Entry(screen2, textvariable=Addr,
width=46,font='Helvetica 13')
    Etr4.place(x="400", y="350", height="40")
    btd = Button(screen2, text="SUBMIT", width=12,

```

```
height=2,bg="#808080",fg="#fdfff5",command=InsertReceiver)
```

```
    btd.place(x="532", y="436")
    BFont = font.Font(family="Times New Roman",
size=14, weight='bold')
    btd['font'] = BFont
```

```
def DispBloodDonor():
    screen1=Toplevel(root)
    screen1.title("DonorDatabase")
    screen1.geometry("1920x1080+0+0")
    style=ttk.Style()
    style.configure("mystyle.Treeview",font=('Times New
Roman',18),rowheight=100)
    style.configure("mystyle.Treeview.Heading",
font=('Times New Roman', 18))
    tv =
ttk.Treeview(screen1,columns=(1,2,3,4,5,6),style="mysty
le.Treeview")
    treeScroll = ttk.Scrollbar(screen1)
    treeScroll.configure(command=tv.yview)
    tv.configure(yscrollcommand=treeScroll.set)
    treeScroll.pack(side=RIGHT, fill=BOTH)
    tv.pack()
    tv.heading("1", text="D_ID")
    tv.heading("2",text="Name")
    tv.heading("3", text="Age")
    tv.heading("4", text="BloodGroup")
    tv.heading("5", text="Location")
    tv.heading("6",text="Infected")
    for row in db.fetchdonor():
        tv.insert("",END,values=row)
    tv['show']='headings'
    tv.pack(fill=X)
```

```

def DispBloodBank():
    screen1 = Toplevel(root)
    screen1.title("Blood Bank Database")
    screen1.geometry("1920x1080+0+0")
    style = ttk.Style()
    style.configure("mystyle.Treeview", font=('Times
New Roman', 18), rowheight=100)
    style.configure("mystyle.Treeview.Heading",
font=('Times New Roman', 18))
    tv1 = ttk.Treeview(screen1, columns=(1,2,3),
style="mystyle.Treeview")
    treeScroll = ttk.Scrollbar(screen1)
    treeScroll.configure(command=tv1.yview)
    tv1.configure(yscrollcommand=treeScroll.set)
    treeScroll.pack(side=RIGHT, fill=BOTH)
    tv1.pack()
    tv1.heading("1", text="D_ID")
    tv1.heading("2", text="Blood Group")
    tv1.heading("3", text="Location")
    for row in db.fetchBank():
        tv1.insert("", END, values=row)
    tv1['show'] = 'headings'
    tv1.pack(fill=X)

def DispBloodReceiver():
    screen1 = Toplevel(root)
    screen1.title("ReceiverDatabase")
    screen1.geometry("1920x1080+0+0")
    style = ttk.Style()
    style.configure("mystyle.Treeview", font=('Times
New Roman', 18), rowheight=100)
    style.configure("mystyle.Treeview.Heading",
font=('Times New Roman', 18))

```

```

    tvt = ttk.Treeview(screen1, columns=(1,2,3,4,5),
style="mystyle.Treeview")
    treeScroll = ttk.Scrollbar(screen1)
    treeScroll.configure(command=tvt.yview)
    tvt.configure(yscrollcommand=treeScroll.set)
    treeScroll.pack(side=RIGHT, fill=BOTH)
    tvt.pack()
    tvt.heading("1", text="R_ID")
    tvt.heading("2", text="Name")
    tvt.heading("3", text="Age")
    tvt.heading("4", text="BloodGroup")
    tvt.heading("5", text="Location")
    for row in db.fetchReciever():
        tvt.insert("", END, values=row)
    tvt['show'] = 'headings'
    tvt.pack(fill=X)

def DeleteDdonor():
    if (Et6.get()==""):
        messagebox.showerror("Error","Enter the Donor
ID")
        return
    res=db.DelDonor(Et6.get())
    if(res==True):
        messagebox.showinfo(screen1,"Record Deleted")
    else:
        messagebox.showerror("Error", "ID not found")

def DeleteRreceiver():
    if (Et8.get()==""):
        messagebox.showerror("Error","Enter the
Receiver ID")
        return
    res=db.DelReceiver(Et8.get())

```

```

    if(res==True):
        messagebox.showinfo(screen1,"Record Deleted")
    else:
        messagebox.showerror("Error", "ID not found")

def Admin():
    username_inf=Et1.get()
    password_inf=Et2.get()
    if(username_inf=="Admin" and password_inf=="1234"):
        global Et6,Et8,screen1
        screen1 = Toplevel(root)
        screen1.title("DataBase")
        screen1.geometry("1920x1080+0+0")
        labl2 = Label(screen1, image=imgg3)
        labl2.place(x=0, y=0, relwidth=1, relheight=1)

        D_id =int
        R_id=int

        BFont = font.Font(family="Times New Roman",
size=12, weight='bold')
        Btd1 = Button(screen1,text="BLOOD DONOR
DETAILS",bg="Black",fg="White", width=23, height=2,
command=DispBloodDonor)
        Btd1.place(x="535", y="80")
        Btd1['font']=BFont
        Btd2 = Button(screen1, text="BLOOD BANK
DETAILS",bg="Black",fg="White", width=23, height=2,
command=DispBloodBank)
        Btd2.place(x="535", y="180")
        Btd2['font'] = BFont
        Btd3 = Button(screen1, text="BLOOD RECEIVER
DETAILS",bg="Black",fg="White", width=25, height=2,
command=DispBloodReciever)
        Btd3.place(x="530", y="280")

```



```

        Btd3['font'] = BFont
        Btd4 = Button(screen1, text="DELETE DONOR",
bg="Black", fg="White", width=25,
height=2,command=DeleteDdonor)
        Btd4.place(x="300", y="380")
        Btd4['font'] = BFont
        Et6 = Entry(screen1,textvariable=D_id,
width=12,font='Helvetica 13')
        Et6.place(x="360", y="450", height="24")
        Btd5 = Button(screen1, text="DELETE RECEIVER",
bg="Black", fg="White", width=25,
height=2,command=DeleteRreceiver)
        Btd5.place(x="740", y="380")
        Btd5['font'] = BFont
        lbname = Label(screen1,text="DONOR ID", bd=0,
background='#FF7377', fg='Black', font=("Times New
Roman", 15, 'bold'))
        lbname.place(x="198", y="450")
        Et8= Entry(screen1, textvariable=R_id,
width=12,font='Helvetica 13')
        Et8.place(x="810", y="450", height="24")
        lBR = Label(screen1, text="RECEIVER ID", bd=0,
background='#FF7377', fg='Black',font=("Times New
Roman", 15, 'bold'))
        lBR.place(x="610", y="450")
    else:
        messagebox.showerror("Error","Invalid User")

```

```

def main_screen():
    global root,Et1,Et2,imgg3
    root=Tk()
    root.title("Blood Bank Management System")
    root.geometry("1920x1080+0+0")

```

```

imgg2=ImageTk.PhotoImage(Image.open(r"C:\Users\mithr\PycharmProjects\pythonProject3\TkinterProject\img1Project.png"))
    mylabel1=Label(root,image=imgg2)
    imgg3 =
ImageTk.PhotoImage(Image.open(r"C:\Users\mithr\PycharmProjects\pythonProject3\TkinterProject\img2Project.png"))
)
    mylabel1.place(x=0,y=0,relwidth=1,relheight=1)
    title=Label(text="BLOOD
BANK",width="600",bg='white',height="2",background='Black',fg='White',font=("Times New Roman",36,'bold'))
    title.pack()
    username=StringVar()
    password=StringVar()

lbyname=Label(text="USERNAME",bd=0,background='#FF7377',fg='Black',font=("Times New Roman",17,'bold'))
    lbyname.place(x="400",y="256")

Et1=Entry(textvariable=username,width=26,font='Helvetica 13')
    Et1.place(x="560",y="258",height="24")
    lpass =
Label(text="PASSWORD",background='#FF7377',fg='Black',font=("Times New Roman",17,'bold'))
    lpass.place(x="400",y="308")

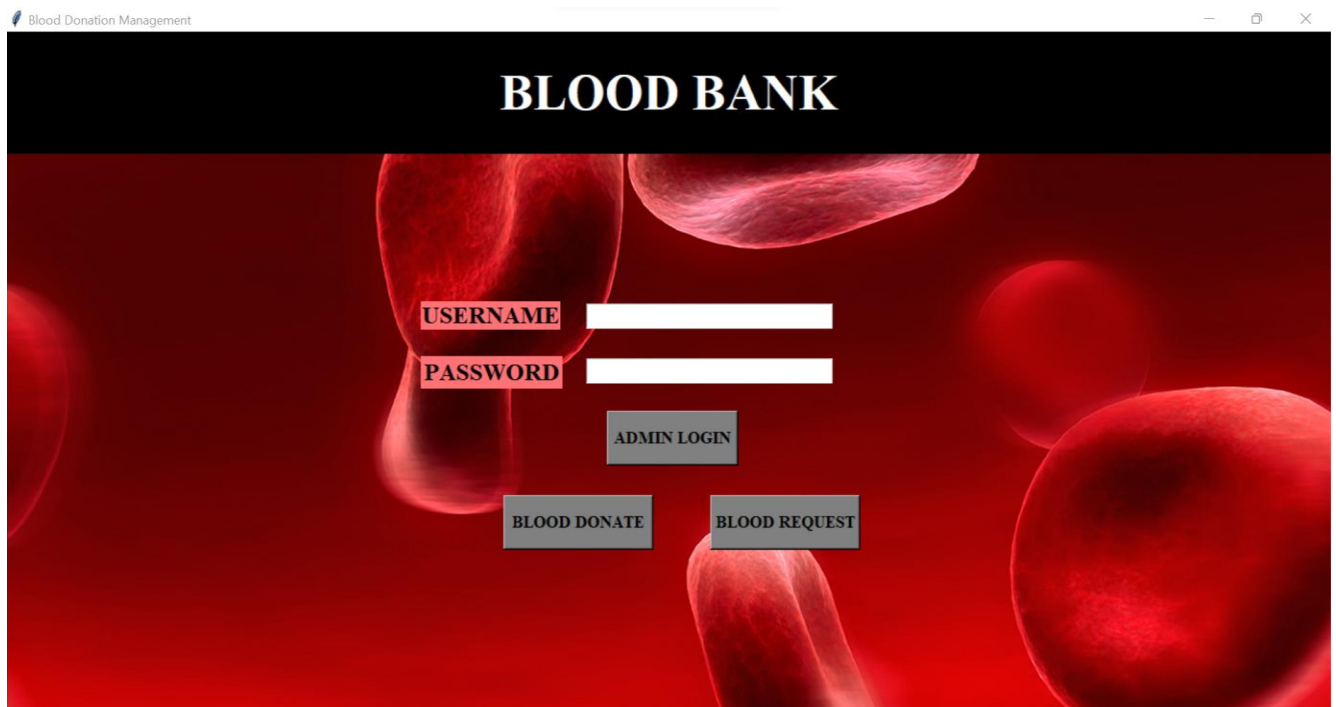
Et2=Entry(textvariable=password,width=26,font='Helvetica 13')
    Et2.place(x="560", y="310",height="24")
    BFont = font.Font(family="Times New Roman",size=12,weight='bold')
    Bt1=Button(text="ADMIN
LOGIN",bg="#808080",fg="Black",command=Admin,width=13,h

```

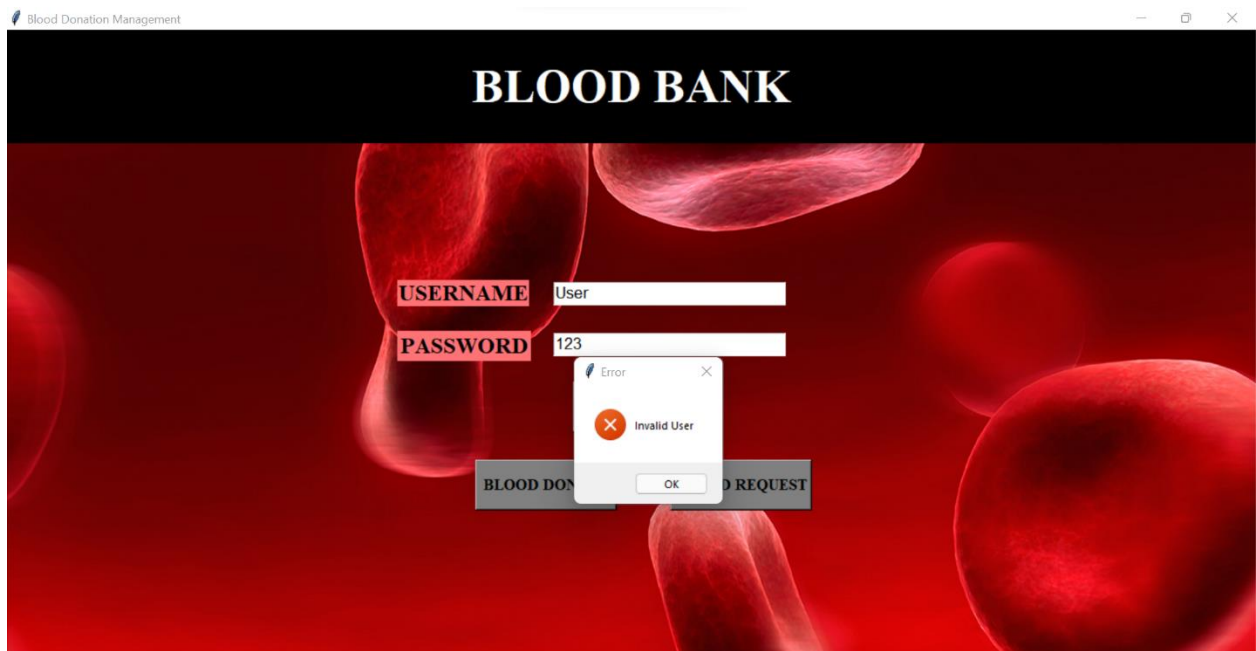
```
eight=2)
    Bt1.place(x="580",y="360")
    Bt1['font']=BFont
    Bt2= Button(text="BLOOD
DONATE",width=15,height=2,bg="#808080",fg="Black",comma
nd=BloodDonarDetails)
    Bt2.place(x="480", y="440")
    Bt2['font'] = BFont
    Bt3 = Button(text="BLOOD REQUEST", width=15,
height=2,bg="#808080",fg="Black",command=BloodRequestDe
tails)
    Bt3.place(x="680", y="440")
    Bt3['font'] = BFont
    root.mainloop()
db = DataBase("DataDb.db")
main_screen()
```

6. Results (Screenshots)

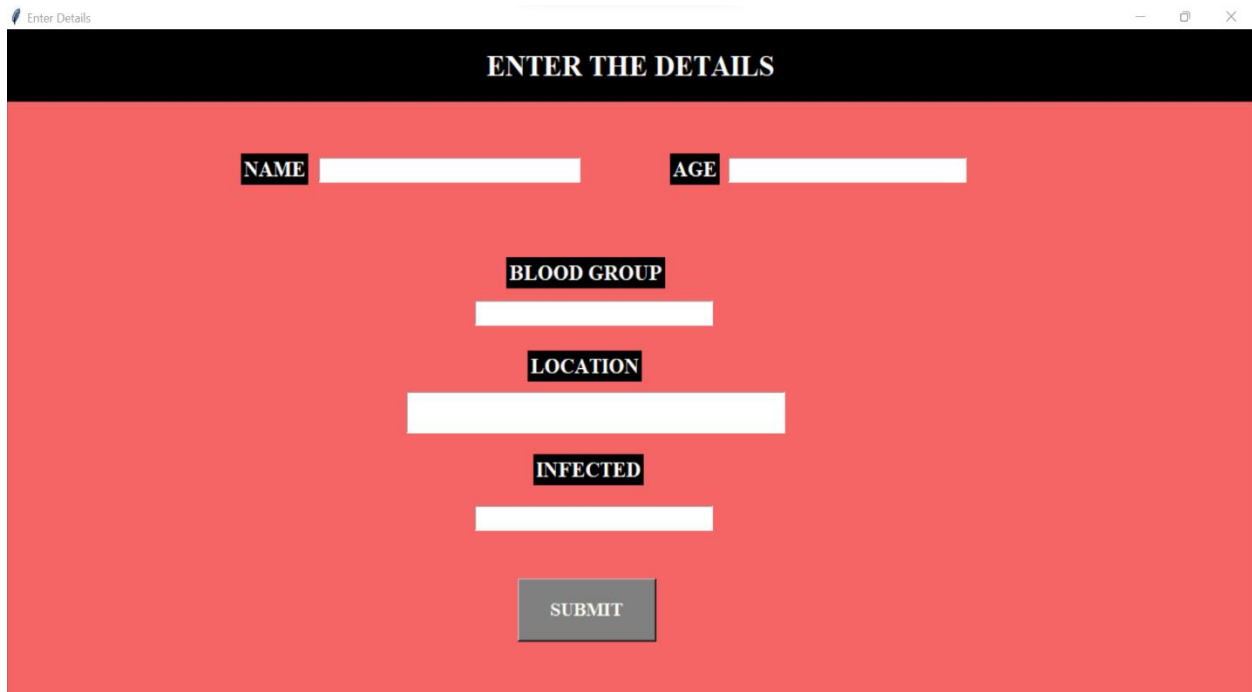
Login page and portal for Donating blood and requesting blood :



Error prompted when user tries to login through Admin portal:

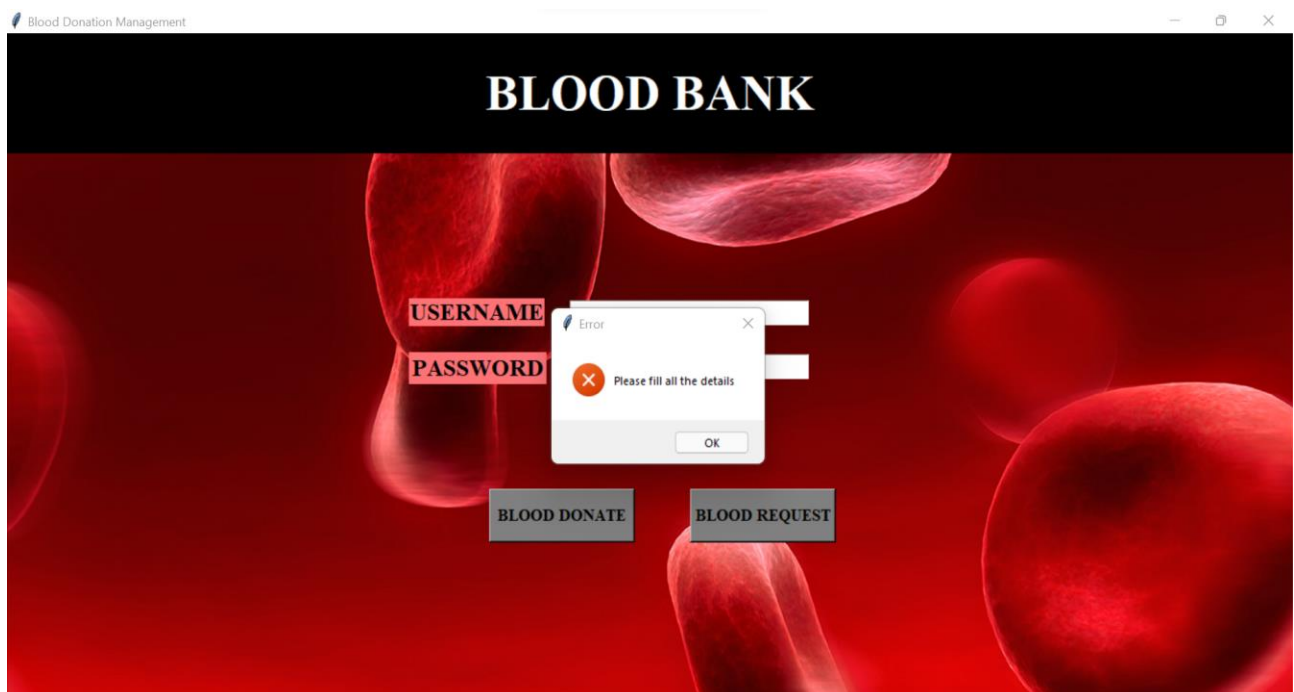


Window for getting blood donor details :



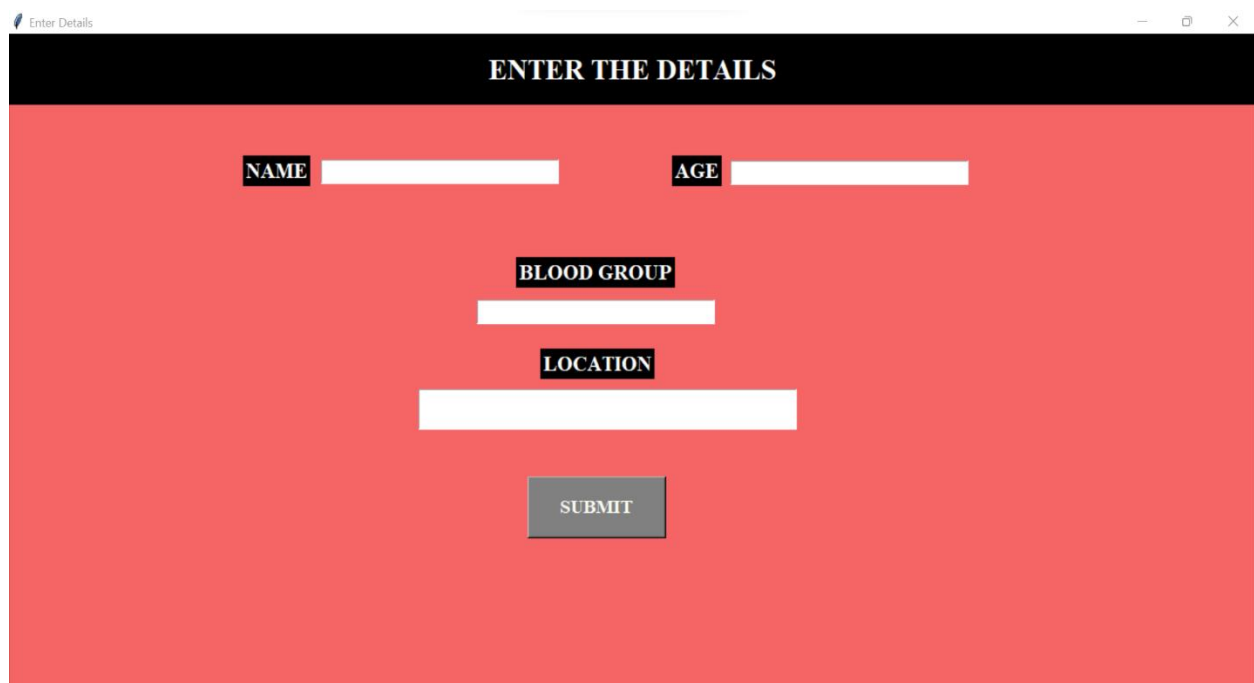
The screenshot shows a web application window titled "Enter Details". The window has a black header bar with the title in white. The main content area has a red background. It contains several input fields with black labels: "NAME", "AGE", "BLOOD GROUP", "LOCATION", and "INFECTED". Each label is followed by a white input field. At the bottom center, there is a grey button labeled "SUBMIT".

Error prompted when complete details are not provided while donating or requesting blood :



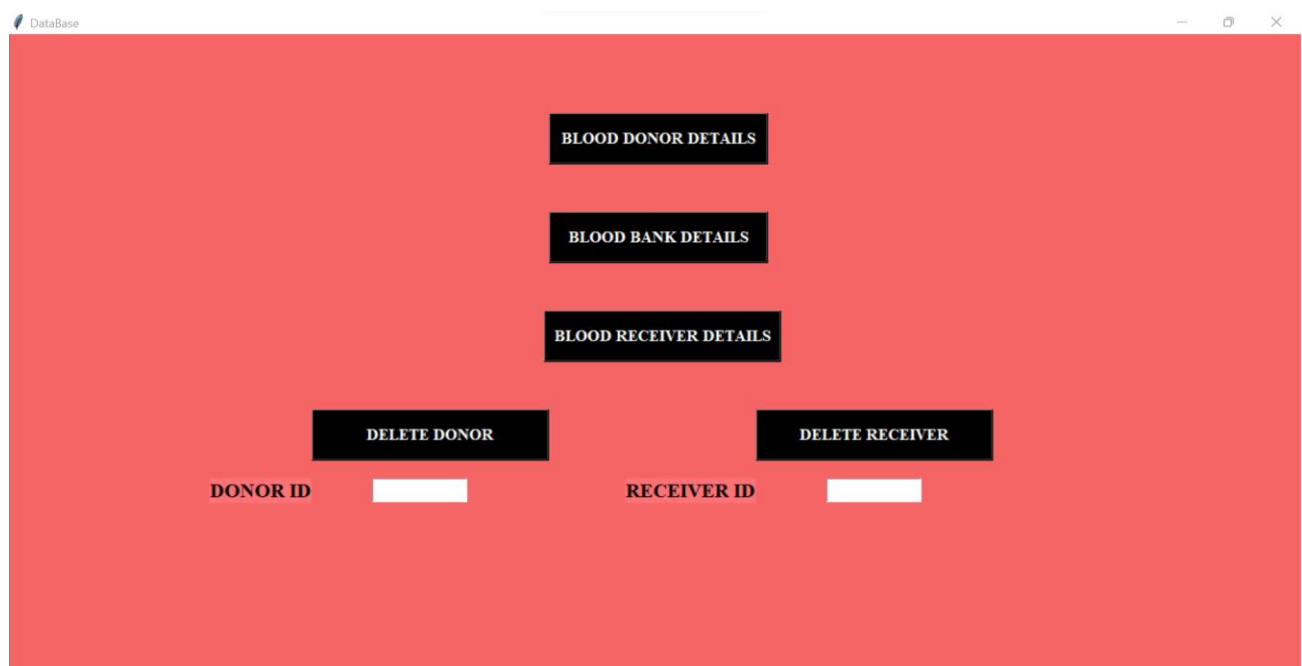
The screenshot shows a web application window titled "BLOOD BANK". The window has a black header bar with the title in white. The main content area has a red background with a large, stylized image of red blood cells. There are two input fields with black labels: "USERNAME" and "PASSWORD". Below these fields are two grey buttons: "BLOOD DONATE" and "BLOOD REQUEST". An error dialog box is open in the center of the window, displaying the message "Please fill all the details" with an "OK" button.

Window for users for requesting blood :



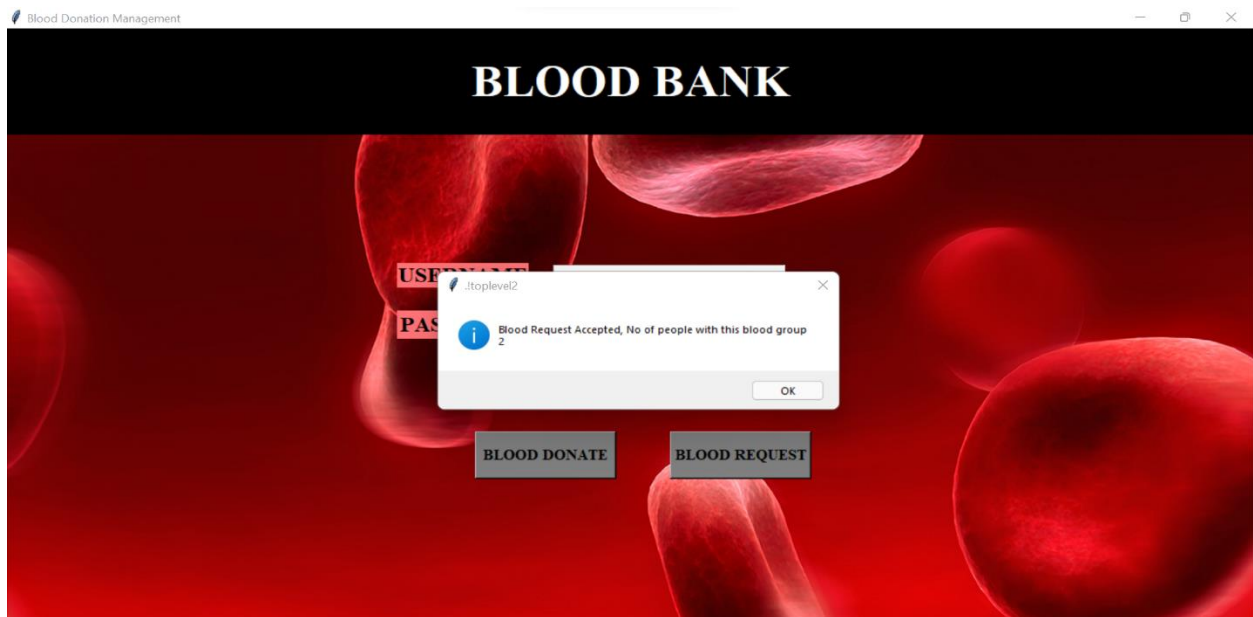
The screenshot shows a web application window titled "Enter Details". The window has a black header bar with the title "ENTER THE DETAILS" in white capital letters. The main content area has a light red background. It contains four input fields: "NAME" and "AGE" are side-by-side at the top, followed by "BLOOD GROUP" and "LOCATION" stacked vertically in the center. At the bottom center is a grey "SUBMIT" button.

Window for accessing all the tables and deleting records which the admin can only access :



The screenshot shows a web application window titled "DataBase". The window has a light red background. It contains five buttons: "BLOOD DONOR DETAILS", "BLOOD BANK DETAILS", and "BLOOD RECEIVER DETAILS" are stacked vertically in the center. At the bottom, there are two buttons: "DELETE DONOR" on the left and "DELETE RECEIVER" on the right. Below each of these bottom buttons is a text label ("DONOR ID" and "RECEIVER ID" respectively) followed by a small white input field.

Message Box that shows number of people with the blood group requested by the user :



Records of Donor (viewed only by the admin):

DonorDatabase					
D_ID	Name	Age	BloodGroup	Location	Infected
1	Mithran	18	O+	Chennai	No
2	Parthiban	19	B+	Agra	No
3	Mithul	19	AB+	Chennai	No
4	Jack	32	A+	Kochin	No
5	Henry	44	AB-	Mumbai	No
6	Kevin	26	AB-	Mumbai	No

Records of Blood Bank (viewed only by the admin):

Blood Bank Database					
D_ID	Blood Group	Location			
1	O+	Chennai			
2	B+	Agra			
3	AB+	Chennai			
4	A+	Kochin			
5	AB-	Mumbai			
6	AB-	Mumbai			

Records of Blood Receiver (viewed only by the admin):

ReceiverDatabase									
R_ID	Name	Age	BloodGroup	Location					
0	Dennis	36	O+	Chennai					
1	Dennis	36	O+	Chennai					
3	None	None	None	None					
4	None	None	None	None					
5	None	None	None	None					

APPENDIX: Report on Python GUI Tkinter.

Tkinter is an inbuilt **Python** module used to create simple **GUI** apps. It is the most commonly used module for **GUI** apps in the **Python**.

The [tkinter](#) package (“Tk interface”) is the standard Python interface to the Tcl/Tk GUI toolkit. Both Tk and [tkinter](#) are available on most Unix platforms, including macOS, as well as on Windows systems.

Running `python -m tkinter` from the command line should open a window demonstrating a simple Tk interface, letting you know that [tkinter](#) is properly installed on your system, and also showing what version of Tcl/Tk is installed, so you can read the Tcl/Tk documentation specific to that version.

Tkinter supports a range of Tcl/Tk versions, built either with or without thread support. The official Python binary release bundles Tcl/Tk 8.6 threaded. See the source code for the `_tkinter` module for more information about supported versions.

Tkinter is not a thin wrapper, but adds a fair amount of its own logic to make the experience more pythonic. This documentation will concentrate on these additions and changes, and refer to the official Tcl/Tk documentation for details that are unchanged.

Python has a lot of [GUI frameworks](#), but [Tkinter](#) is the only framework that’s built into the Python standard library. Tkinter has several strengths. It’s **cross-platform**, so the same code works on Windows, macOS, and Linux. Visual elements are rendered using native operating system elements, so applications built with Tkinter look like they belong on the platform where they’re run.

Although Tkinter is considered the de-facto Python GUI framework, it’s not without criticism. One notable criticism is that GUIs built with Tkinter look outdated. If you want a shiny, modern interface, then Tkinter may not be what you’re looking for.

However, Tkinter is lightweight and relatively painless to use compared to other frameworks. This makes it a compelling choice for building GUI applications in Python, especially for applications where a modern sheen is unnecessary, and the top priority is to build something that’s functional and cross-platform quickly.

Anything that happens in a user interface is an event. We say that an event is fired whenever the user does something – for example, clicks on a button or types a keyboard shortcut. Some events could also be triggered by occurrences which are not controlled by the user – for example, a background task might complete, or a network connection might be established or lost.

Our application needs to monitor, or listen for, all the events that we find interesting, and respond to them in some way if they occur. To do this, we usually associate certain functions with particular events. We call a function which performs an action in response to an event an event handler – we bind handlers to events.

Architecture

Tcl/Tk is not a single library but rather consists of a few distinct modules, each with separate functionality and its own official documentation. Python's binary releases also ship an add-on module together with it.

Tcl

Tcl is a dynamic interpreted programming language, just like Python. Though it can be used on its own as a general-purpose programming language, it is most commonly embedded into C applications as a scripting engine or an interface to the Tk toolkit. The Tcl library has a C interface to create and manage one or more instances of a Tcl interpreter, run Tcl commands and scripts in those instances, and add custom commands implemented in either Tcl or C. Each interpreter has an event queue, and there are facilities to send events to it and process them. Unlike Python, Tcl's execution model is designed around cooperative multitasking, and Tkinter bridges this difference (see [Threading model](#) for details).

Tk

Tk is a [Tcl package](#) implemented in C that adds custom commands to create and manipulate GUI widgets. Each [Tk](#) object embeds its own Tcl interpreter instance with Tk loaded into it. Tk's widgets are very customizable, though at the cost of a dated appearance. Tk uses Tcl's event queue to generate and process GUI events.

Ttk

Themed Tk (Ttk) is a newer family of Tk widgets that provide a much better appearance on different platforms than many of the classic Tk widgets. Ttk is distributed as part of Tk, starting with Tk version 8.5. Python bindings are provided in a separate module, [tkinter.ttk](#).

Internally, Tk and Ttk use facilities of the underlying operating system, i.e., Xlib on Unix/X11, Cocoa on macOS, GDI on Windows.

When your Python application uses a class in Tkinter, e.g., to create a widget, the [tkinter](#) module first assembles a Tcl/Tk command string. It passes that Tcl command string to an internal `_tkinter` binary module, which then calls the Tcl interpreter to evaluate it. The Tcl interpreter will then call into the Tk and/or Ttk packages, which will in turn make calls to Xlib, Cocoa, or GDI.

Tkinter Modules

Support for Tkinter is spread across several modules. Most applications will need the main [tkinter](#) module, as well as the [tkinter.ttk](#) module, which provides the modern themed widget set and API:

```
from tkinter import *
from tkinter import ttk
```

```
class tkinter.Tk(screenName=None, baseName=None, className='Tk', useTk=1)
```

The [Tk](#) class is instantiated without arguments. This creates a toplevel widget of Tk which usually is the main window of an application. Each instance has its own associated Tcl interpreter.

```
tkinter.Tcl(screenName=None, baseName=None, className='Tk', useTk=0)
```

The [Tcl\(\)](#) function is a factory function which creates an object much like that created by the [Tk](#) class, except that it does not initialize the Tk subsystem. This is most often useful when driving the Tcl interpreter in an environment where one doesn't want to create extraneous toplevel windows, or where one cannot (such as Unix/Linux systems without an X server). An object created by the [Tcl\(\)](#) object can have a Toplevel window created (and the Tk subsystem initialized) by calling its `loadtk()` method.

The modules that provide Tk support include:

[tkinter](#)

Main Tkinter module.

[tkinter.colorchooser](#)

Dialog to let the user choose a color.

[tkinter.commondialog](#)

Base class for the dialogs defined in the other modules listed here.

[tkinter.filedialog](#)

Common dialogs to allow the user to specify a file to open or save.

[tkinter.font](#)

Utilities to help work with fonts.

[tkinter.messagebox](#)

Access to standard Tk dialog boxes.

[tkinter.scrolledtext](#)

Text widget with a vertical scroll bar built in.

[tkinter.simpledialog](#)

Basic dialogs and convenience functions.

[tkinter.ttk](#)

Themed widget set introduced in Tk 8.5, providing modern alternatives for many of the classic widgets in the main [tkinter](#) module.

Additional modules:

[_tkinter](#)

A binary module that contains the low-level interface to Tcl/Tk. It is automatically imported by the main [tkinter](#) module, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

[idlelib](#)

Python's Integrated Development and Learning Environment (IDLE). Based on [tkinter](#).

[tkinter.constants](#)

Symbolic constants that can be used in place of strings when passing various parameters to Tkinter calls. Automatically imported by the main [tkinter](#) module.

[tkinter.dnd](#)

(experimental) Drag-and-drop support for [tkinter](#). This will become deprecated when it is replaced with the Tk DND.

[tkinter.tix](#)

(deprecated) An older third-party Tcl/Tk package that adds several new widgets. Better alternatives for most can be found in [tkinter.ttk](#).

[turtle](#)

Turtle graphics in a Tk window.

References:

1. Abraham Silberschatz, Henry F. Korth, S. Sudharshan, Database System Concepts, Sixth Edition, Tata McGraw Hill, 2011.
2. Ramez Elmasri and Shamkant B. Navathe, Fundamentals of Database Systems, Seventh Edition, Pearson Education, 2016.
3. C.J.Date, A.Kannan, S.Swamynathan, An Introduction to Database Systems, Eighth Edition, Pearson Education, 2003.