A Project Report on

Patient Information System in Python

Submitted in partial fulfillment of the

Requirements for the award of the degree

BACHELOR OF COMPUTER APPLICATIONS

Submitted by

A Muhil Raghavandhar (REG NO:212001840)

Neelesh SaiNath G S (REG NO:212001848)

Under the guidance of

Mrs. P.JAYABHARATHI MCA.,MPhil.,SET.,(P.hd.,)

UG DEPARTMENT OF COMPUTER APPLICATIONS

AGURCHAND MANMULL JAIN COLLEGE

(Affiliated to the University of Madras)

CHENNAI-600061

NOVEMBER 2022 – APRIL 2023

DECLARATION

AGURCHAND MANMULL JAIN COLLEGE

(Affiliated to the University of Madras)

CHENNAI-600061

This is to certify that the content of the project carried by **A Muhil Raghavandhar (REG NO:212001840), Neelesh SaiNath G S (REG NO:212001848),**in the partial fulfillment of the requirement of award of Bachelor of Computer Applications of the University of Madras during the year November 2022 – April 2023

Signature of the Facilitator

Dr. A. Udhayakumar

BONAFIDE

This is to certify that the project report in **Patient Information System in Python** is carried by A Muhil Raghavandhar (REG NO: 212001840), Neelesh SaiNath G S (REG NO:212001848), in partial fulfillment of the requirements for the award of Bachelor of Computer Applications of the University of Madras during November 2022 – April 2023.

Signature of the Guide
(Mrs. P.Jayabharathi)

Submitted for the viva-voce Examination held on 20/4/2023 at AgurchandManmull Jain College, Meenambakkam, Chennai- 600061.

Internal Examiner
Examiner

External

ACKNOWLEDGEMENT

we would like to thank all those who gave us their time, care and guidance which can never be repaid. The following dignitaries deserves a special mention.

we express our heartfelt gratitude to our honorable Secretary **Shri.UDHAN KUMAR CHORDIA** and Associate Secretary **Shri.HEMANTH P CHORDIA** for their continuous encouragement and providing all the resources needed behind the scenes.

we would like to express our sincere thanks to our beloved **Principal Dr. N. VENKATARAMAN**, the sustenance in the project work.

we express our profound thanks to our respected Dean **Dr. M.M. RAMYA** for her enthusiastic support, help and providing all the resources needed behind the scenes.

We wish to thank **Dr.A.Udhayakumar, Facilitator**, Department of Computer Applications for his guidance.

We would like to thank our Guide and Project trainer **Mrs. P.JAYABHARATHI MCA.,MPhil.,SET.,(P.hd.,)**Department of Computer Applications, AgurchandManmull Jain College , for her full involvement in every part of our project.

INDEX

1. INTRODUCTION

AnPatient Infromation System in Pythonisasoftwareapplicationthatallowsto store patient information for hospitalsorcliniestoperform medical checkups forvariouspatients they can interact with doctorsovertheinternet.Thesystemprovidesasecureandconvenientwayto manage the data of the patients withouttheyneed not tovisit Hospitals or Clinic.

The Patient Information System typically includes features such as Emergencies,General checkups, or Medical checkups,etc.,. It also offers all facilities over the internet and it offers health insurance.

To use an Patient Information System, patients need to register for an account with theirPatient I'd, Firstname,Lastname,Date of Birth,Contact Number,E-Mail I'd,Blood Group,Address,etc.,.

PROJECTOVERVIEW

The Patient Information System is a web-based application that enables users to managetheirdatabase.ThissystemisbuiltusingPython. The system includes features such as patient information creation, patient history,update patient information, search patient information, display patient information, and exit patient information to protect userdata.

Thepurposeof thisproject istodevelop auser-friendlyfor patients who cannot able to travel for hospitals and clinics can visit various doctors from online. With the rise of internet andmobilespatients can contact doctors through online.

Theuserinterfaceofthe system is designedusingPythonandis designed to beuser-friendlyfor patients.

The system includes various security measures to protect user data such as information about the patients.Thesystemalsoincludesbackupanddisasterrecoveryplanstoprotectthe data.

In this project documentation, we will outline the system requirements, architecture, insert patient information,update patient information,delete patient information,search patient information,display patient information,and supportforthe Patient Information System.

WHATISPYTHON?

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed .

WHATISPYTHONPROGRAMINGLANGUAGE?

Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific programs.

WHAT IS PYCHARM?

Pycharm is an integrated development environment used for programming in Python. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and supports web development with Django. Pycharm is developed by the Czech company JetBrains.

WHAT IS JETBRAINS?

JetBrains s.r.o. is a Czech software development private limited company which makes tools for software developers and project managers.The company has it's headquarters in Prague, and has offices in China, Europe and the United States.

WHAT IS SQLITE?

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a popular choice as an embedded database for local/client storage in application software such as web browsers. It is also used in many other applications that need a lightweight, embedded data

2. ABOUT THE PROJECT

The Patient Information System is a web-based application that enables users to insert the patient information throughonline.ThesystemisbuiltusingPython.JetBrains.

The primary objective of this project is to develop a secure and user-friendly patients detailsthat allows users to perform various details from the comfort of their offices.The system is designed to provide a convenient and reliable platform for users to access their patients information, manage their details.

The user interface of the system is designed using Python and is intended to be user-friendly andeasytonavigate.Thedatabaseforthesystemisbuiltusing SQLite,alightweightandreliabledatabasemanagementsystemthatiswell-suitedforsmalltomedium-sizedapplications.Thedatabaseincludestablesforstoringuser information,patient information, andtype of patient suffering information.

The system includes various security measures such as password encryption, anddata encryption to protect user data. The system also includes backup and disaster recovery plans toprotectagainst dataloss and systemfailures.

Thisprojectaimstoprovideafullyfunctionalandreliablepatient information systemthatmeetstheneedsof users and ensures the security and privacy of their data. The project is expected to be completedwithin the patient information and what the paient is suffering from, and doctor name for treatment

3.

SYSTEM CONFIGRATION

The purpose of system requirement specification is to produce the specification analysis of the taskandalsotoestablishcompleteinformationabouttherequirement,behaviorandotherconstraintssuchasf unctional performanceand so on.

The goal of system requirement specification is to completely specify the technical requirements fortheproduct in aconcise and unambiguous manner.

HARDWAREREQUIREMENTS

Processor	Intel(R)Core(TM)i5-7300U
Speed	2.70GHz
RAM	8GB
Hard Disk	20GB

SOFTWAREREQUIREMENTS

Operating System	Windows10 Pro
Ide	Pycharm JetBrains
Front -End	Python
Back–End	SQLite

4. SYSTEM DESIGN

USER INTERFACE DESIGN:

The user interface of the Patient Information System is designed using Python. Thesystem consists of several options, including Insert patient information, Update patient information, Search patient information, Display patient information, and Delete patient information. The screens are designed to be intuitive and easy touse,with clear navigation paths and user-friendlycontrols.

DATABASE DESIGN:

The Patient Information System uses SQLite as its database management system. Thedatabase schema consists of several tables, including insert, update, display, search, and delete patientinformation.Thetables arerelated usingprimarykeysandforeign keys.

SYSTEM ARCHITECTURE:

The Patient Information System is built using a client-server architecture. The client-sideof the system is developed using Python(Pycharm), while the server-side uses a combination of Python andSQLite.

5. MODULES IN SYSTEM

There are six modules in this system:

1. Insert
2. Update
3. Search
4. Delete
5. Display
6. Exit

- **Insert:** in this module if the patient can create a new data by entering the followinginformation Patient Id, Name, Address, Phone Number, Date Of Birth, Gender, E-mail address, Blood type andHistory of the patient&name of the Doctor and then click insertbutton thedata will be created.

- **Update:** in this module enter your patient Id (which you need to update) and click the search button it automaticallydisplays the particular data. In this you can edit the information you can and then select theupdatebutton to savethe data.

- **Search & Delete:** in this module enter your Id and click on search button it will show yourdata or if you need to delete the particular patient data enter the particular patient Idand click delete button andit will be deleted.

- **Display:** in this module it displays all the data which u have inserted.

- **Exit:** in this module it exits from the program output page.

6. SOURCE CODE

MODULES USED:

1.Importing a Database:

```
class Database:

def __init__(self):

self.dbConnection = sqlite3.connect("patientdb.db")

self.dbCursor = self.dbConnection.cursor()

self.dbCursor.execute(

"CREATE TABLE IF NOT EXISTS patient_table (id PRIMARYKEY text, firstname text, lastname text, dateOfBirth text, monthOfBirth text, yearOfBirth text, gender text, address text, contactNumber text, emailAddress text, bloodType text, history text, doctor text)")

def __del__(self):

self.dbCursor.close()

self.dbConnection.close()

def Insert(self, id, firstname, lastname, dateOfBirth, monthOfBirth, yearOfBirth, gender, address, contactNumber, emailAddress, bloodType, history, doctor):

self.dbCursor.execute("INSERT INTO patient_table VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", (

id, firstname, lastname, dateOfBirth, monthOfBirth, yearOfBirth, gender, address, contactNumber, emailAddress, bloodType, history, doctor))

self.dbConnection.commit()

def Update(self, firstname, lastname, dateOfBirth, monthOfBirth, yearOfBirth, gender, address, contactNumber, emailAddress, bloodType, history, doctor, id):

self.dbCursor.execute(
```

"UPDATE patient_table SET firstname = ?, lastname = ?, dateOfBirth = ?, monthOfBirth = ?, yearOfBirth = ?, gender = ?, address = ?, contactNumber = ?, emailAddress = ?, bloodType = ?, history = ?, doctor = ? WHERE id = ?",

(firstname, lastname, dateOfBirth, monthOfBirth, yearOfBirth, gender, address, contactNumber, emailAddress, bloodType, history, doctor, id))

self.dbConnection.commit()

def Search(self, id):

self.dbCursor.execute("SELECT * FROM patient_table WHERE id = ?", (id,))

searchResults = self.dbCursor.fetchall()

return searchResults

def Delete(self, id):

self.dbCursor.execute("DELETE FROM patient_table WHERE id = ?", (id,))

self.dbConnection.commit()

def Display(self):

self.dbCursor.execute("SELECT * FROM patient_table")

records = self.dbCursor.fetchall()

return record

2.Importing Validation

class Values:

def Validate(self, id, firstname, lastname, contactNumber, emailAdress, history, doctor):

if not (id.isdigit() and (len(id) == 3)):

return "id"

elif not (firstname.isalpha()):

```
return "firstname"
elif not (lastname.isalpha()):
return "lastname"
elif not (contactNumber.isdigit() and (len(contactNumber) == 11)):
return "contactNumber"
elif not (emailAdress.count("@") == 1 and emailAdress.count(".") > 0):
return "emailAddress"
elif not (history.isalpha()):
return "history"
elif not (doctor.isalpha()):
return "doctor"
else:
return "SUCCESS"
```

3.Creating a Home Page

```
class HomePage:
def __init__(self):
self.homePageWindow = tkinter.Tk()
self.homePageWindow.wm_title("Patient Information Details")
bg_color = "blue"
fg_color = "white"
lbl_color = 'GREEN'
```

```
tkinter.Label(self.homePageWindow, relief=tkinter.GROOVE, fg=fg_color,
bg=bg_color, text="Home Page", font=("times new roman",20,"bold"),
width=30).grid(pady=20, column=1, row=1)


tkinter.Button(self.homePageWindow, width=20, relief=tkinter.GROOVE,
fg=fg_color, bg=bg_color, text="Insert", font=("times new roman",15,"bold"),
command=self.Insert).grid(pady=15, column=1, row=2)

tkinter.Button(self.homePageWindow, width=20, relief=tkinter.GROOVE,
fg=fg_color, bg=bg_color, text="Update", font=("times new roman",15,"bold"),
command=self.Update).grid(pady=15, column=1, row=3)

tkinter.Button(self.homePageWindow, width=20, relief=tkinter.GROOVE,
fg=fg_color, bg=bg_color, text="Search", font=("times new roman",15,"bold"),
command=self.Search).grid(pady=15, column=1, row=4)

tkinter.Button(self.homePageWindow, width=20, relief=tkinter.GROOVE,
fg=fg_color, bg=bg_color, text="Delete", font=("times new roman",15,"bold"),
command=self.Delete).grid(pady=15, column=1, row=5)

tkinter.Button(self.homePageWindow, width=20, relief=tkinter.GROOVE,
fg=fg_color, bg=bg_color, text="Display", font=("times new roman",15,"bold"),
command=self.Display).grid(pady=15, column=1,

row=6)

tkinter.Button(self.homePageWindow, width=20, relief=tkinter.GROOVE,
fg=fg_color, bg=bg_color, text="Exit", font=("times new roman",15,"bold"),
command=self.homePageWindow.destroy).grid(pady=15,

column=1,

row=7)


self.homePageWindow.mainloop()
```

```
def Insert(self):
self.insertWindow = InsertWindow()

def Update(self):
self.updateIDWindow = tkinter.Tk()
self.updateIDWindow.wm_title("Update data")

# Initializing all the variables
self.id = tkinter.StringVar()

# Label
tkinter.Label(self.updateIDWindow, text="Enter the ID to update",
width=50).grid(pady=20, row=1)

# Entry widgets
self.idEntry = tkinter.Entry(self.updateIDWindow, width=5, textvariable=self.id)

self.idEntry.grid(pady=10, row=2)

# Button widgets
tkinter.Button(self.updateIDWindow, width=20, text="Update",
command=self.updateID).grid(pady=10, row=3)
```

```
self.updateIDWindow.mainloop()

def updateID(self):
self.updateWindow = UpdateWindow(self.idEntry.get())
self.updateIDWindow.destroy()

def Search(self):
self.searchWindow = SearchDeleteWindow("Search")

def Delete(self):
self.deleteWindow = SearchDeleteWindow("Delete")

def Display(self):
self.database = Database()
self.data = self.database.Display()
self.displayWindow = DatabaseView(self.data)


homePage = HomePage()
```

4.Inserting a Data Frame:

```
class InsertWindow:

def __init__(self):

self.window = tkinter.Tk()

self.window.wm_title("Insert Patient Data ")

bg_color = "Blue"

fg_color = "white"


self.id = tkinter.StringVar()

self.firstname = tkinter.StringVar()

self.lastname = tkinter.StringVar()

self.address = tkinter.StringVar()

self.contactNumber = tkinter.StringVar()

self.emailAddress = tkinter.StringVar()

self.history = tkinter.StringVar()

self.doctor = tkinter.StringVar()


self.genderType = ["Male", "Female", "Transgender", "Other"]

self.dateType = list(range(1, 32))

self.monthType = ["January", "February", "March", "April", "May", "June",
"July", "August", "September",

"October", "November", "December"]
```

```
self.yearType = list(range(1900, 2020))

self.bloodListType = ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]


# Labels

tkinter.Label(self.window, fg=fg_color, bg=bg_color, text="Patient Id",
font=("times new roman",10,"bold"), width=25).grid(pady=5, column=1, row=1)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, text="Patient First Name",
font=("times new roman",10,"bold"), width=25).grid(pady=5, column=1, row=2)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"), text="Patient Last Name", width=25).grid(pady=5, column=1,
row=3)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"), text="Date of Birth", width=25).grid(pady=5, column=1,
row=4)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"),text="Month of Birth", width=25).grid(pady=5, column=1,
row=5)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"),text="Year of Birth", width=25).grid(pady=5, column=1,
row=6)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"),text="Patient Gender", width=25).grid(pady=5, column=1,
row=7)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"),text="Patient Address", width=25).grid(pady=5, column=1,
row=8)
```

```
tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"),text="Patient Contact Number", width=25).grid(pady=5,
column=1, row=9)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"),text="Patient Email Address", width=25).grid(pady=5,
column=1, row=10)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"),text="Patient Blood Type", width=25).grid(pady=5, column=1,
row=11)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"),text="History of Patient", width=25).grid(pady=5, column=1,
row=12)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new
roman",10,"bold"),text="Name of Doctor", width=25).grid(pady=5, column=1,
row=13)


self.idEntry = tkinter.Entry(self.window, width=25, textvariable=self.id)

self.firstnameEntry = tkinter.Entry(self.window, width=25,
textvariable=self.firstname)

self.lastnameEntry = tkinter.Entry(self.window, width=25,
textvariable=self.lastname)

self.addressEntry = tkinter.Entry(self.window, width=25,
textvariable=self.address)

self.contactNumberEntry = tkinter.Entry(self.window, width=25,
textvariable=self.contactNumber)

self.emailAddressEntry = tkinter.Entry(self.window, width=25,
textvariable=self.emailAddress)
```

```
self.historyEntry = tkinter.Entry(self.window, width=25, textvariable=self.history)
self.doctorEntry = tkinter.Entry(self.window, width=25, textvariable=self.doctor)

self.idEntry.grid(pady=5, column=3, row=1)
self.firstnameEntry.grid(pady=5, column=3, row=2)
self.lastnameEntry.grid(pady=5, column=3, row=3)
self.addressEntry.grid(pady=5, column=3, row=8)
self.contactNumberEntry.grid(pady=5, column=3, row=9)
self.emailAddressEntry.grid(pady=5, column=3, row=10)
self.historyEntry.grid(pady=5, column=3, row=12)
self.doctorEntry.grid(pady=5, column=3, row=13)

# Combobox widgets
self.dateOfBirthBox = tkinter.ttk.Combobox(self.window, values=self.dateType,
width=25)
self.monthOfBirthBox = tkinter.ttk.Combobox(self.window,
values=self.monthType, width=25)
self.yearOfBirthBox = tkinter.ttk.Combobox(self.window, values=self.yearType,
width=25)
self.genderBox = tkinter.ttk.Combobox(self.window, values=self.genderType,
width=25)
self.bloodListBox = tkinter.ttk.Combobox(self.window,
values=self.bloodListType, width=25)
```

```
self.dateOfBirthBox.grid(pady=5, column=3, row=4)

self.monthOfBirthBox.grid(pady=5, column=3, row=5)

self.yearOfBirthBox.grid(pady=5, column=3, row=6)

self.genderBox.grid(pady=5, column=3, row=7)

self.bloodListBox.grid(pady=5, column=3, row=11)


# Button widgets

tkinter.Button(self.window, width=10, fg=fg_color, bg=bg_color, font=("times
new roman",10,"bold"), text="Insert", command=self.Insert).grid(pady=15,
padx=5, column=1,

row=14)

tkinter.Button(self.window, width=10, fg=fg_color, bg=bg_color, font=("times
new roman",10,"bold"), text="Reset", command=self.Reset).grid(pady=15,
padx=5, column=2, row=14)

tkinter.Button(self.window, width=10, fg=fg_color, bg=bg_color, font=("times
new roman",10,"bold"), text="Close",
command=self.window.destroy).grid(pady=15, padx=5, column=3,

row=14)


self.window.mainloop()
```

5.Updating Patient Information:

```
class UpdateWindow:

def __init__(self, id):

self.window = tkinter.Tk()

self.window.wm_title("Update data")
```

```
bg_color = "Blue"
fg_color = "white"

# Initializing all the variables
self.id = id

self.firstname = tkinter.StringVar()
self.lastname = tkinter.StringVar()
self.address = tkinter.StringVar()
self.contactNumber = tkinter.StringVar()
self.emailAddress = tkinter.StringVar()
self.history = tkinter.StringVar()
self.doctor = tkinter.StringVar()

self.genderType = ["Male", "Female", "Transgender", "Other"]
self.dateType = list(range(1, 32))
self.monthType = ["January", "February", "March", "April", "May", "June",
"July", "August", "September",
"October", "November", "December"]
self.yearType = list(range(1900, 2020))
self.bloodListType = ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]

# Labels
```

```
tkinter.Label(self.window, fg=fg_color, bg=bg_color, text="Patient Id",
font=("times new roman", 10, "bold"),

width=25).grid(pady=5, column=1, row=1)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, text="Patient First Name",

font=("times new roman", 10, "bold"), width=25).grid(pady=5, column=1, row=2)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"),

text="Patient Last Name", width=25).grid(pady=5, column=1, row=3)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"), text="Date of Birth",

width=25).grid(pady=5, column=1, row=4)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"),

text="Month of Birth", width=25).grid(pady=5, column=1, row=5)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"), text="Year of Birth",

width=25).grid(pady=5, column=1, row=6)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"),

text="Patient Gender", width=25).grid(pady=5, column=1, row=7)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"),

text="Patient Address", width=25).grid(pady=5, column=1, row=8)

tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"),
```

```
text="Patient Contact Number", width=25).grid(pady=5, column=1, row=9)
tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"),
text="Patient Email Address", width=25).grid(pady=5, column=1, row=10)
tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"),
text="Patient Blood Type", width=25).grid(pady=5, column=1, row=11)
tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"),
text="History of Patient", width=25).grid(pady=5, column=1, row=12)
tkinter.Label(self.window, fg=fg_color, bg=bg_color, font=("times new roman",
10, "bold"),
text="Name of Doctor", width=25).grid(pady=5, column=1, row=13)


# Set previous values
self.database = Database()
self.searchResults = self.database.Search(id)


tkinter.Label(self.window, text=self.searchResults[0][1], width=25).grid(pady=5,
column=2, row=2)
tkinter.Label(self.window, text=self.searchResults[0][2], width=25).grid(pady=5,
column=2, row=3)
tkinter.Label(self.window, text=self.searchResults[0][3], width=25).grid(pady=5,
column=2, row=4)
```

```
tkinter.Label(self.window, text=self.searchResults[0][4], width=25).grid(pady=5,
column=2, row=5)

tkinter.Label(self.window, text=self.searchResults[0][5], width=25).grid(pady=5,
column=2, row=6)

tkinter.Label(self.window, text=self.searchResults[0][6], width=25).grid(pady=5,
column=2, row=7)

tkinter.Label(self.window, text=self.searchResults[0][7], width=25).grid(pady=5,
column=2, row=8)

tkinter.Label(self.window, text=self.searchResults[0][8], width=25).grid(pady=5,
column=2, row=9)

tkinter.Label(self.window, text=self.searchResults[0][9], width=25).grid(pady=5,
column=2, row=10)

tkinter.Label(self.window, text=self.searchResults[0][10], width=25).grid(pady=5,
column=2, row=11)

tkinter.Label(self.window, text=self.searchResults[0][11], width=25).grid(pady=5,
column=2, row=12)

tkinter.Label(self.window, text=self.searchResults[0][12], width=25).grid(pady=5,
column=2, row=13)


self.idEntry = tkinter.Entry(self.window, width=25, textvariable=self.id)

self.firstnameEntry = tkinter.Entry(self.window, width=25,
textvariable=self.firstname)

self.lastnameEntry = tkinter.Entry(self.window, width=25,
textvariable=self.lastname)

self.addressEntry = tkinter.Entry(self.window, width=25,
textvariable=self.address)
```

```
self.contactNumberEntry = tkinter.Entry(self.window, width=25,
textvariable=self.contactNumber)

self.emailAddressEntry = tkinter.Entry(self.window, width=25,
textvariable=self.emailAddress)

self.historyEntry = tkinter.Entry(self.window, width=25, textvariable=self.history)

self.doctorEntry = tkinter.Entry(self.window, width=25, textvariable=self.doctor)


self.idEntry.grid(pady=5, column=3, row=1)

self.firstnameEntry.grid(pady=5, column=3, row=2)

self.lastnameEntry.grid(pady=5, column=3, row=3)

self.addressEntry.grid(pady=5, column=3, row=8)

self.contactNumberEntry.grid(pady=5, column=3, row=9)

self.emailAddressEntry.grid(pady=5, column=3, row=10)

self.historyEntry.grid(pady=5, column=3, row=12)

self.doctorEntry.grid(pady=5, column=3, row=13)


# Combobox

self.dateOfBirthBox = tkinter.ttk.Combobox(self.window, values=self.dateType,
width=20)

self.monthOfBirthBox = tkinter.ttk.Combobox(self.window,
values=self.monthType, width=20)

self.yearOfBirthBox = tkinter.ttk.Combobox(self.window, values=self.yearType,
width=20)

self.genderBox = tkinter.ttk.Combobox(self.window, values=self.genderType,
width=20)
```

```
self.bloodListBox = tkinter.ttk.Combobox(self.window,
values=self.bloodListType, width=20)

self.dateOfBirthBox.grid(pady=5, column=3, row=4)
self.monthOfBirthBox.grid(pady=5, column=3, row=5)
self.yearOfBirthBox.grid(pady=5, column=3, row=6)
self.genderBox.grid(pady=5, column=3, row=7)
self.bloodListBox.grid(pady=5, column=3, row=11)


# Button
tkinter.Button(self.window, width=10, fg=fg_color, bg=bg_color, font=("times
new roman", 10, "bold"),
text="Insert", command=self.Insert).grid(pady=15, padx=5, column=1,
row=14)
tkinter.Button(self.window, width=10, fg=fg_color, bg=bg_color, font=("times
new roman", 10, "bold"),
text="Reset", command=self.Reset).grid(pady=15, padx=5, column=2, row=14)
tkinter.Button(self.window, width=10, fg=fg_color, bg=bg_color, font=("times
new roman", 10, "bold"),
text="Close", command=self.window.destroy).grid(pady=15, padx=5, column=3,
row=14)


self.window.mainloop()
```

7.Importing Search and Delete Window:

```
class SearchDeleteWindow:
  def __init__(self, task):
    window = tkinter.Tk()
    window.wm_title(task + " data")

    # Initializing all the variables
    self.id = tkinter.StringVar()
    self.firstname = tkinter.StringVar()
    self.lastname = tkinter.StringVar()
    self.heading = "Please enter Patient ID to " + task

    # Labels
    tkinter.Label(window, text=self.heading, width=50).grid(pady=20, row=1)
    tkinter.Label(window, text="Patient ID", width=10).grid(pady=5, row=2)

    # Entry widgets
    self.idEntry = tkinter.Entry(window, width=5, textvariable=self.id)

    self.idEntry.grid(pady=5, row=3)

    # Button widgets
    if (task == "Search"):
      tkinter.Button(window, width=20, text=task,
command=self.Search).grid(pady=15, padx=5, column=1, row=14)
    elif (task == "Delete"):
      tkinter.Button(window, width=20, text=task,
command=self.Delete).grid(pady=15, padx=5, column=1, row=14)

  def Search(self):
    self.database = Database()
    self.data = self.database.Search(self.idEntry.get())
    self.databaseView = DatabaseView(self.data)
```

```
def Delete(self):
    self.database = Database()
    self.database.Delete(self.idEntry.get())
```

8.Displaying a Database Table:

```
class DatabaseView:
def __init__(self, data):
self.databaseViewWindow = tkinter.Tk()
self.databaseViewWindow.wm_title("Database View")

# Label widgets
tkinter.Label(self.databaseViewWindow, text="Database View Window",
width=25).grid(pady=5, column=1, row=1)

self.databaseView = tkinter.ttk.Treeview(self.databaseViewWindow)
self.databaseView.grid(pady=5, column=1, row=2)
self.databaseView["show"] = "headings"
self.databaseView["columns"] = (
"id", "firstname", "lastname", "dateOfBirth", "monthOfBirth", "yearOfBirth",
"gender", "address", "contactNumber", "emailAddress", "bloodType", "history",
"doctor")

# Treeview column headings
self.databaseView.heading("id", text="Patient ID")
self.databaseView.heading("firstname", text="First Name")
```

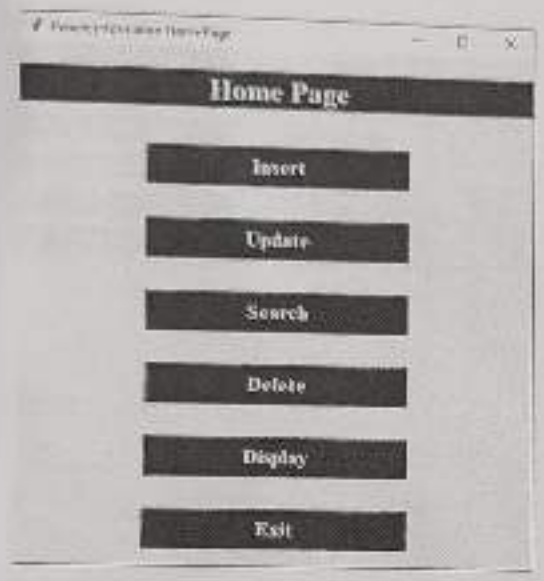
```
self.databaseView.heading("lastname", text="Last Name")
self.databaseView.heading("dateOfBirth", text="Date of Birth")
self.databaseView.heading("monthOfBirth", text="Month of Birth")
self.databaseView.heading("yearOfBirth", text="Year of Birth")
self.databaseView.heading("gender", text="Gender")
self.databaseView.heading("address", text="Home Address")
self.databaseView.heading("contactNumber", text="Contact Number")
self.databaseView.heading("emailAddress", text="Email Address")
self.databaseView.heading("bloodType", text="Blood Type")
self.databaseView.heading("history", text="History")
self.databaseView.heading("doctor", text="Doctor")

# Treeview columns
self.databaseView.column("id", width=100)
self.databaseView.column("firstname", width=100)
self.databaseView.column("lastname", width=100)
self.databaseView.column("dateOfBirth", width=100)
self.databaseView.column("monthOfBirth", width=100)
self.databaseView.column("yearOfBirth", width=100)
self.databaseView.column("gender", width=100)
self.databaseView.column("address", width=200)
self.databaseView.column("contactNumber", width=100)
self.databaseView.column("emailAddress", width=200)
```

```
self.databaseView.column("bloodType", width=100)
self.databaseView.column("history", width=100)
self.databaseView.column("doctor", width=100)

for record in data:
self.databaseView.insert('', 'end', values=(record))

self.databaseViewWindow.mainloop()
```

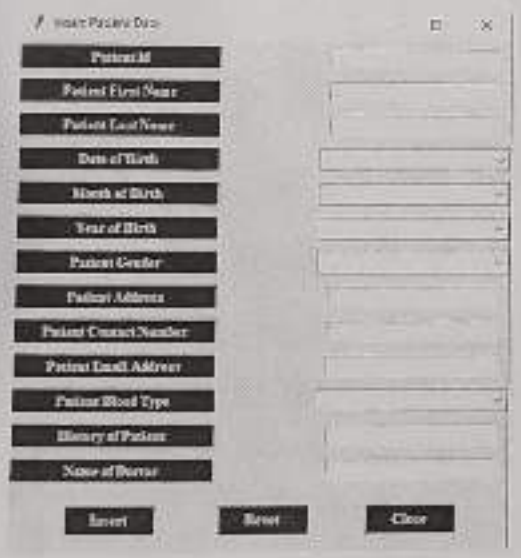
7. OUTPUT

General Output:-

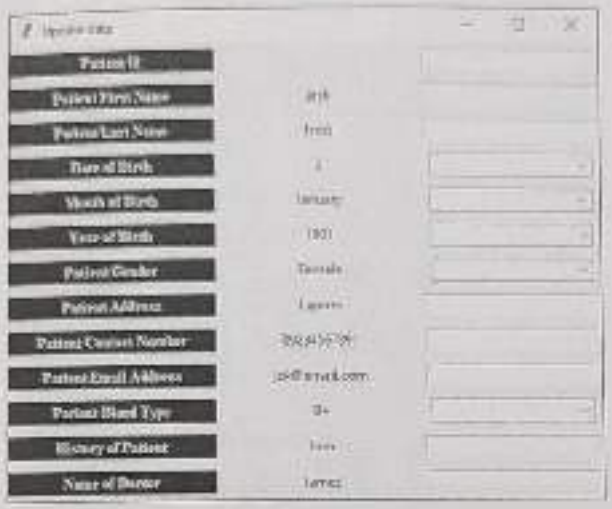
Home Page:



Insert:

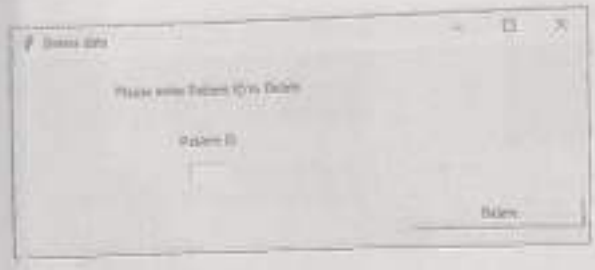


Update:

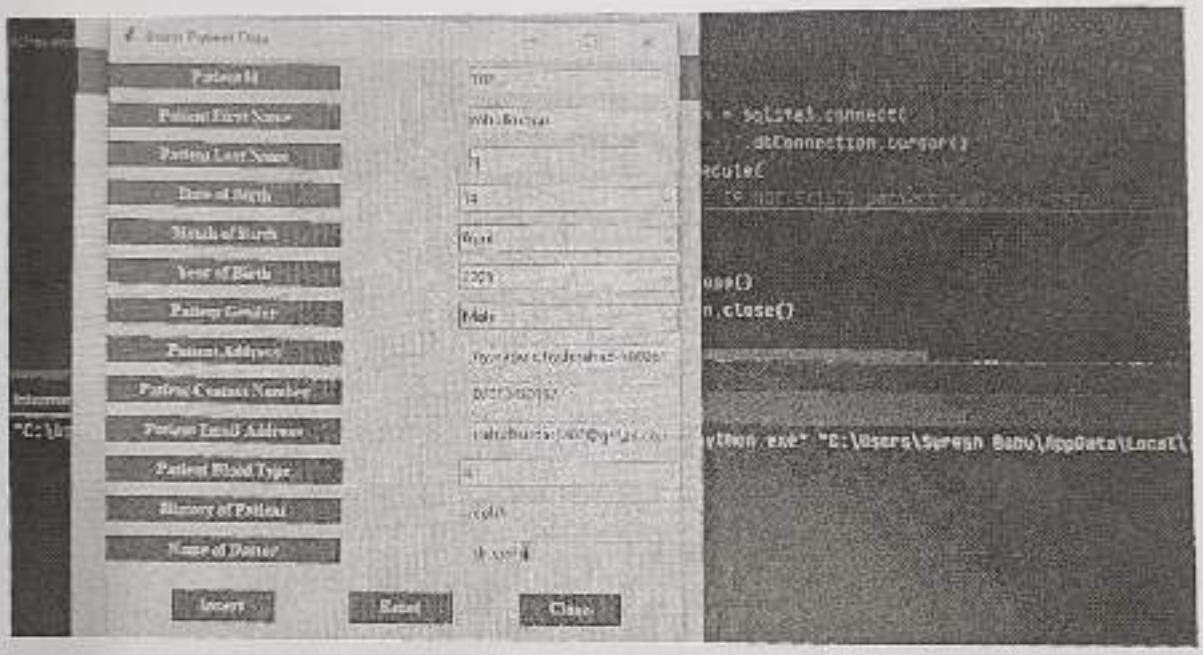


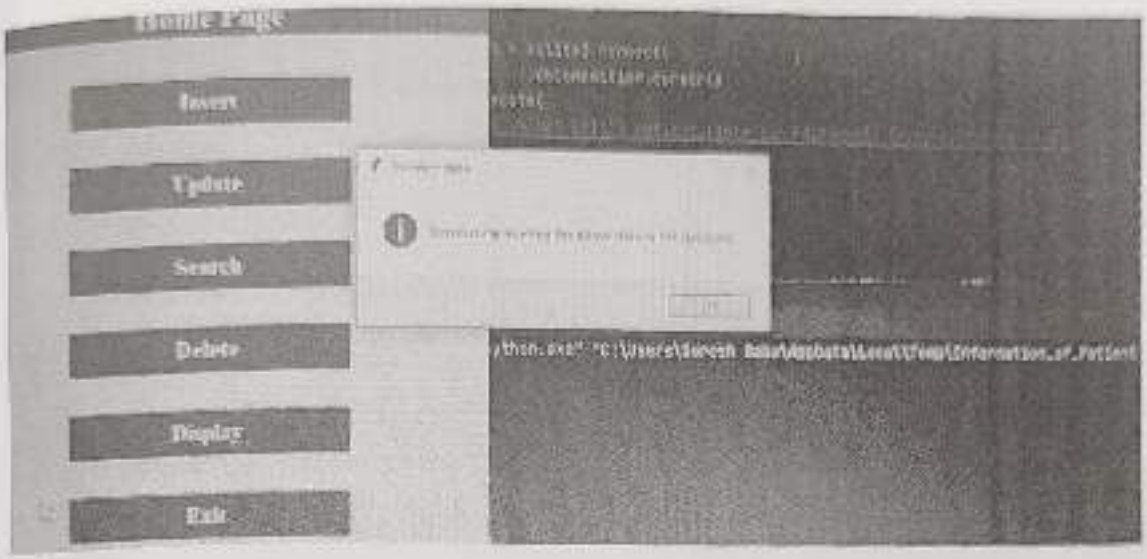
Display:

Delete:

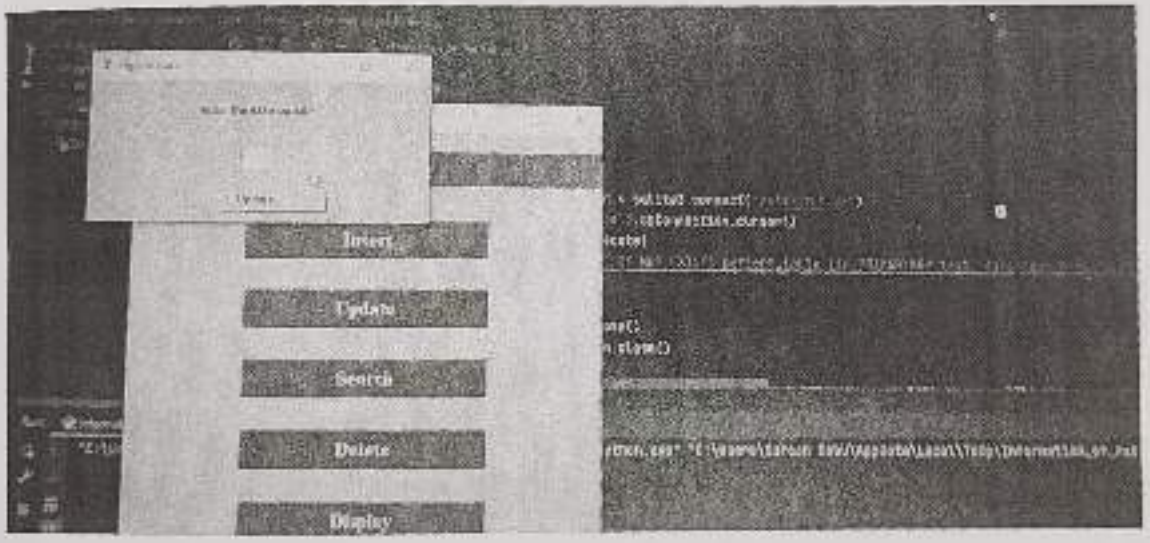


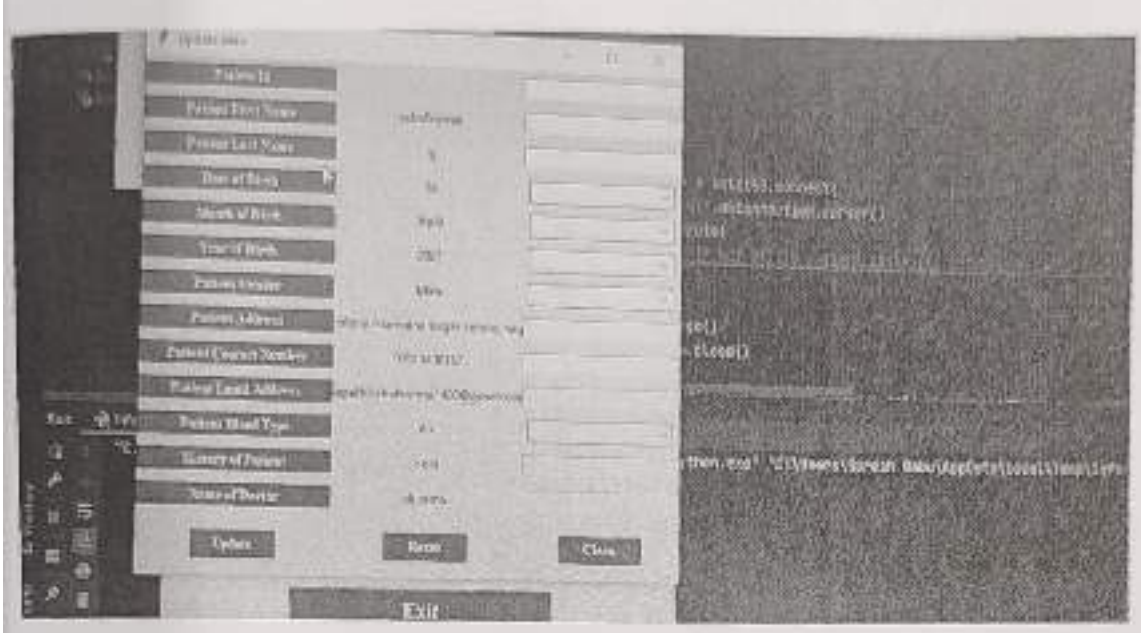
Inserting patient information:-

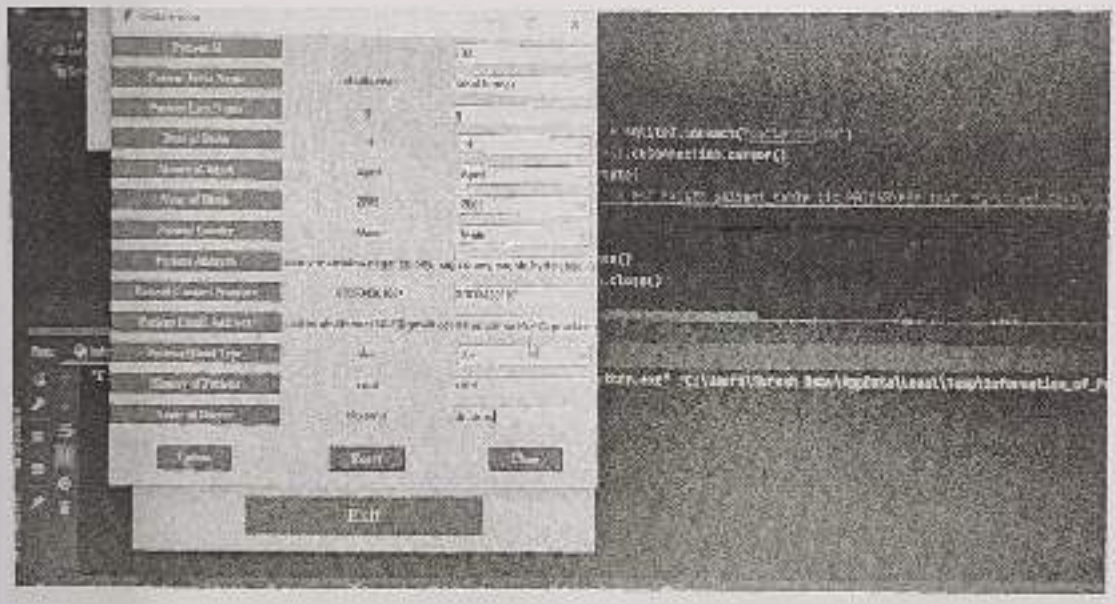


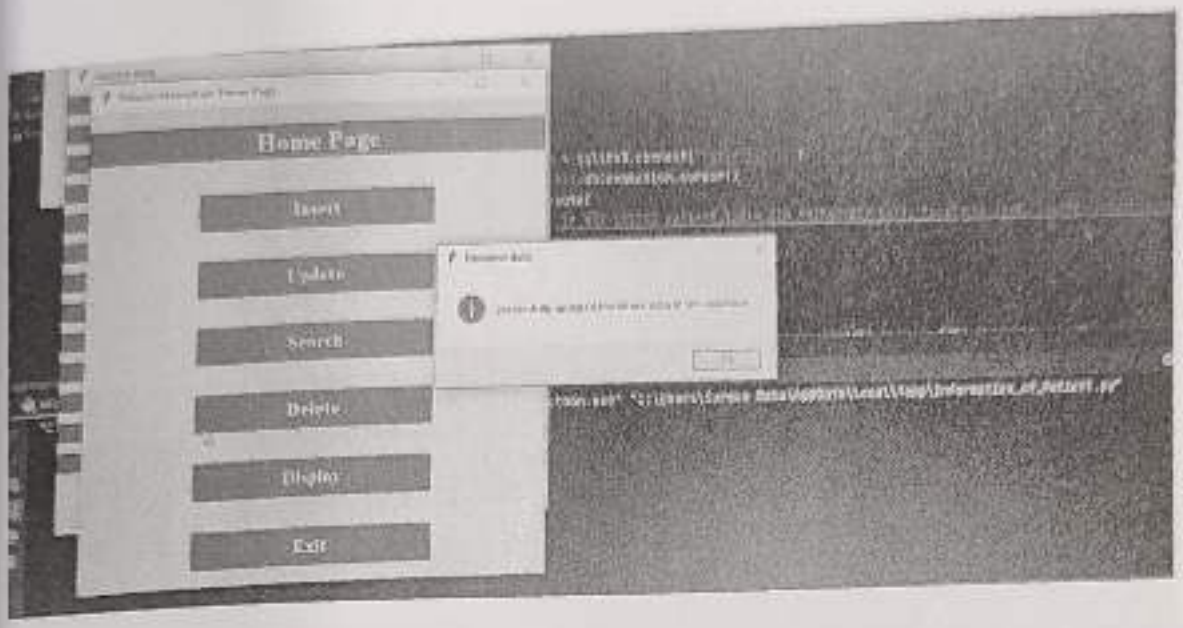


Updating patient information:-

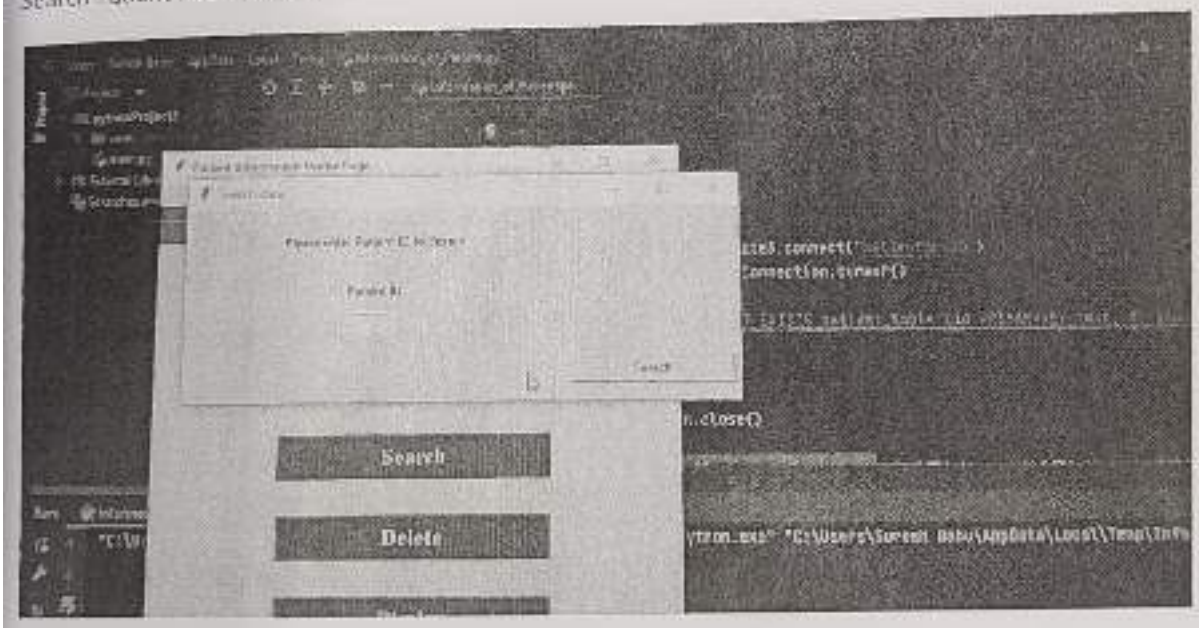


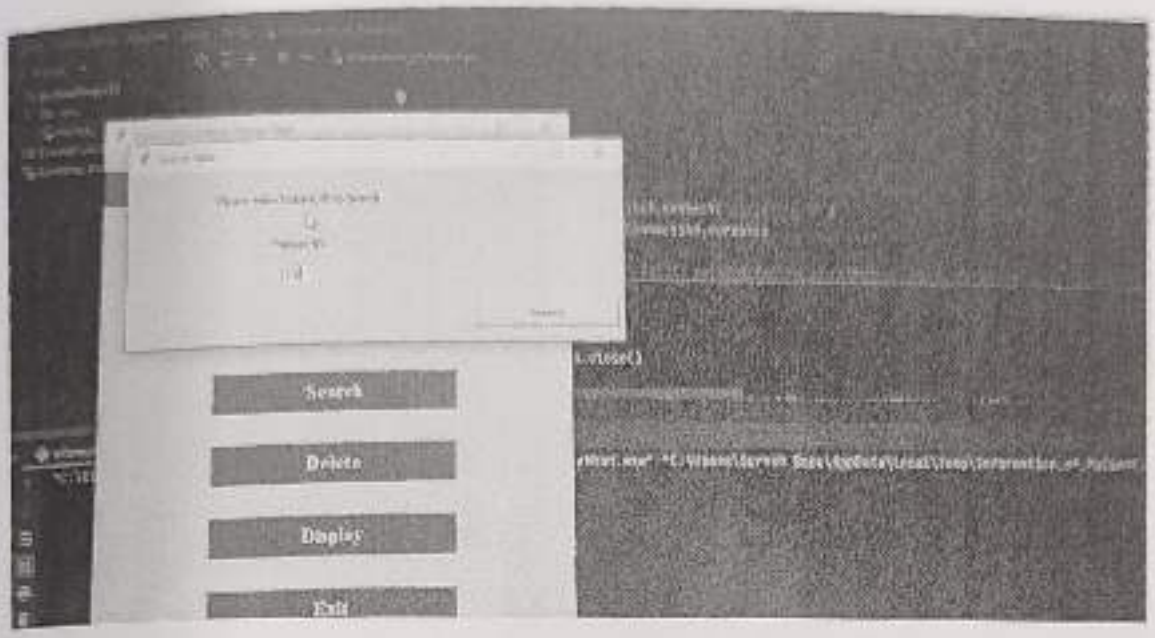

Exit

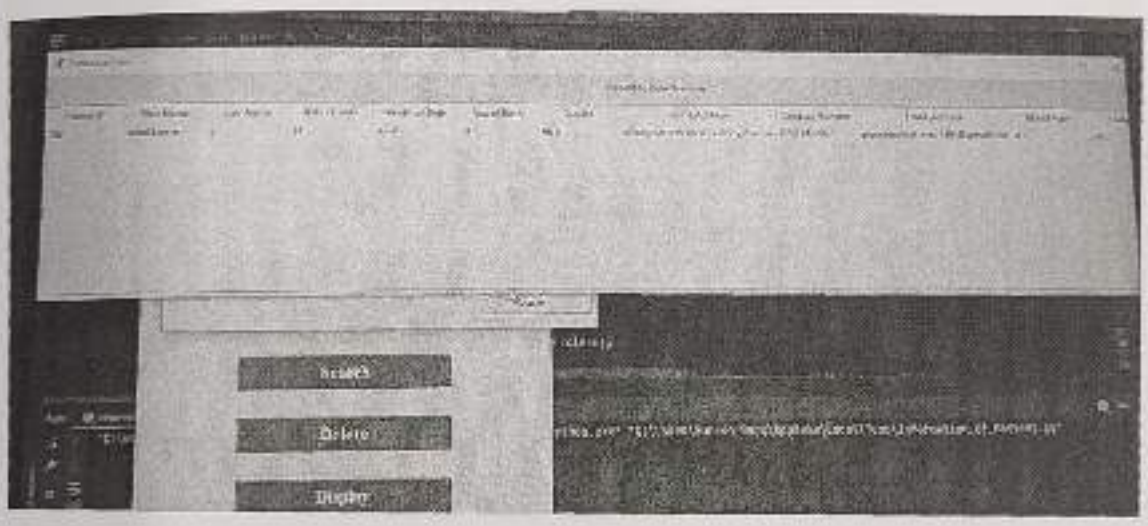

Exit



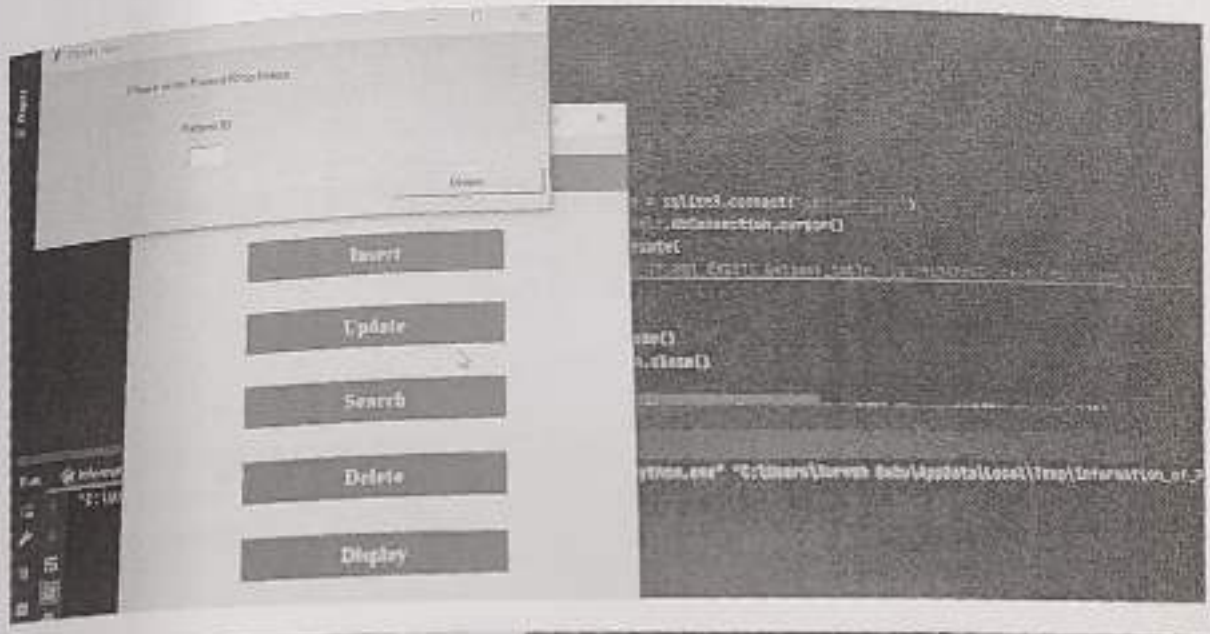
Search Patient Information:

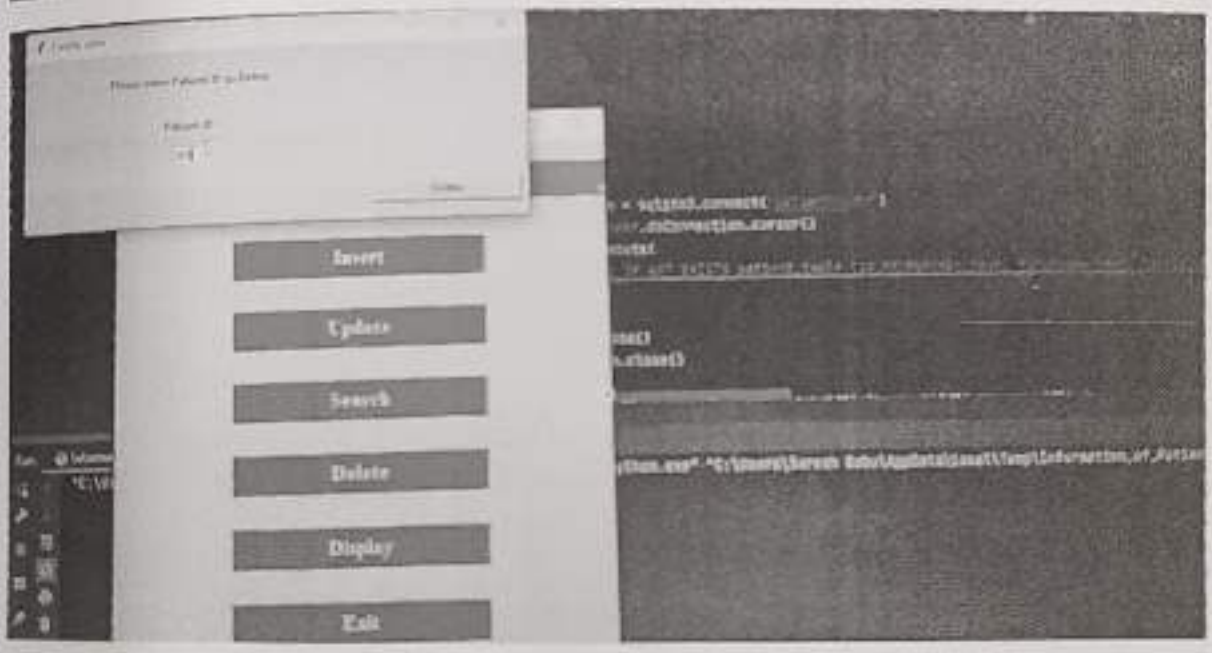


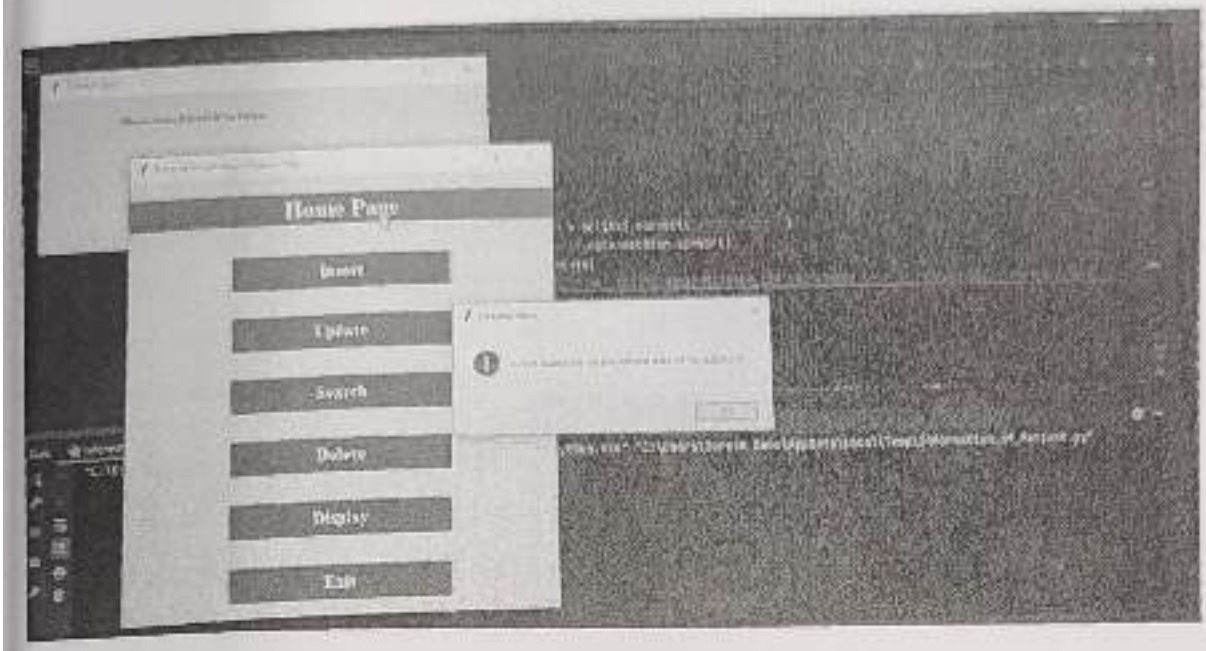




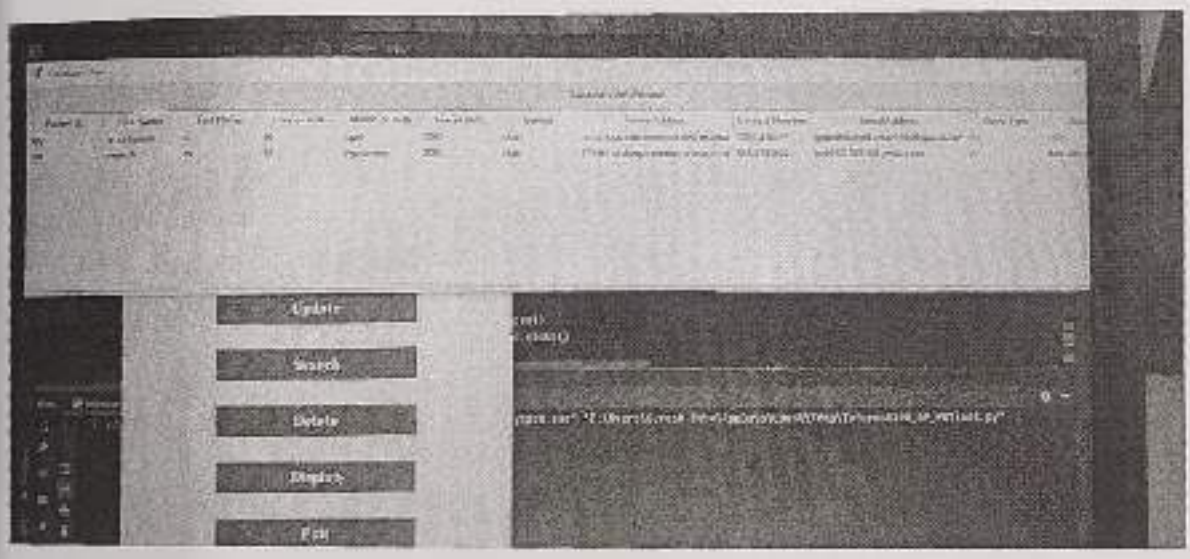
Deleting Patient Information:-


Insert
Update
Search
Delete
Display


Insert
Update
Search
Delete
Display
Exit



Displaying Patient Information:-



CONCLUSION

In conclusion, the use of Python, NetBeans, and SQLite provides an efficient andeffective way to develop an online banking management system. Python provides awide range of components for building a user-friendly graphical user interface, NetBeansis a powerful integrated development environment that allows for easy development anddeploymentofpythonapplications,andSQLiteisalightweightdatabasemanagementsystemthatis suitableforpatient details applications.

The patient information developed using these technologies can includevarious functionalities such as patient registration and user login,address of patient management,doctor name management, body check-up enquiry. The use of SSL encryption, multi-factor authentication, and accesscontrolensures the securityof thesystem.

Overall, the combination of Python, and SQLite provides a powerful andefficientwaytodeveloparelfable and securepatient informationmanagementsystem.