

Patient Information System

A Project Report

Submitted by:

Tanujit Roy<1641017280>

Abhinav Panigrahi<1641017230>

Suman Kumar Subudhi<1641017081>

in partial fulfillment for the award of the degree
of

**BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND INFORMATION
TECHNOLOGY**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Faculty of Engineering and Technology, Institute of Technical Education and
Research**

**SIKSHA 'O' ANUSANDHAN (DEEMED TO BE) UNIVERSITY
Bhubaneswar, Odisha, India
(May 2020)**

Table Of Contents

Declaration of the Student	i
Certificate of the Guide	ii
Report Approval	iii
Abstract	iv
Acknowledgment	v
List of Figures	vi
Timeline / Gantt Chart	vii
1. INTRODUCTION	1
1.1 Problem Definition	3
1.2 Project Overview/Specifications	4
1.3 Hardware Specification	5
1.4 Software Specification	5
2. LITERATURE SURVEY	6
2.1 Existing System	7
2.2 Proposed System	8
2.3 Feasibility Study	9
3. SYSTEM ANALYSIS & DESIGN	13
3.1 Requirement Specification	14
3.2 Flowcharts / UML Diagrams / ERDs/	21
3.3 Design and Test Steps / Criteria	25
3.3 Algorithms and Pseudo Code	
3.3.1 Algorithms	28
3.3.2 Pseudo Code	29
3.4 Testing Process	56
4. RESULTS / OUTPUTS	62
5. CONCLUSIONS / RECOMMENDATIONS	71
6. REFERENCES	73
7. SIMILARITY REPORT	75

DECLARATION

We declare that this written submission represents our ideas in our own words and where other's ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/fact/source in our submission. We understand that any violation of the above will cause for disciplinary action by the University and can also evoke penal action from the sources which have not been properly cited or from whom proper permission has not been taken when needed.

Tanujit Roy - 1641017280

Abhinav Panigrahi

Suman Kumar Subudhi - 1641017081

(1641017230)

Signature of Students with Registration Numbers

13/05/2020

Date: _____

CERTIFICATE

This is to certify that the project report titled “Patient Information System” being submitted by (Tanujit Roy, Suman Kumar Subudhi and Abhinav Panigrahi of CSIT-C) to the Institute of Technical Education and Research, Siksha ‘O’ Anusandhan (Deemed to be) University, Bhubaneswar for the partial fulfillment for the degree of Bachelor of Technology in Computer Science and Information Technology is a record of original confide work carried out by them under my/our supervision and guidance. The project work, in my/our opinion, has reached the requisite standard fulfilling the requirements for the degree of Bachelor of Technology.

The results contained in this thesis have not been submitted in part or full to any other Universityor Institute for the award of any degree or diploma.



Prabhat Ku Sahoo

Department of Computer Science and Engineering

Faculty of Engineering and Technology

Institute of Technical Education and Research

Siksha ‘O’ Anusandhan (Deemed to be) University

REPORT APPROVAL

This project report entitled "Patient Information System" by (Tanujit Roy and Abhinav Panigrahi) is approved for the degree of Bachelor of Technology in Computer Science and Information Technology.

Examiners

Patient Information System

ABSTRACT

Description:

The aim of Patient Information System is to help in feeding the informations through the system without manually doing it, so that their valuable data will be stored for a longer period and it's going to be simply accessed from anyplace. Admins can manipulate the datas, needed to be modified if required. The hospitals can maintain the computerized records without any unwanted or duplicate entries. It stores the patient's information, disease caused and therefore the allocated doctors that he/she has visited earlier.

Features:

Feed patient details.

To know the consulted doctors.

It allows adding / editing patient registration.

It allows to search for a patient.

It allows deleting the patients details.

ACKNOWLEDGMENT

Whenever a module of work is completed successfully, a source of inspiration and guidance is always there for the student. I, hereby take the opportunity to thank those entire people who helped me in many different ways. First and foremost, I am grateful to my thesis guide Professor/Prabhat Ku Sahoo, SOA Deemed to be University, for showing faith in my capability and providing able guidance and his generosity and advice extended to me throughout my project. Last, but not least I would like to thank all my faculty, my friends and my parents for helping me in all measure of life and for their kind cooperation and moral support.

Tanujit Roy - 1641017280

Abhinav Panigrahi

(1641017230)

Suman Kumar Subudhi - 1641017081

Place: Bhubaneswar

Signature of students

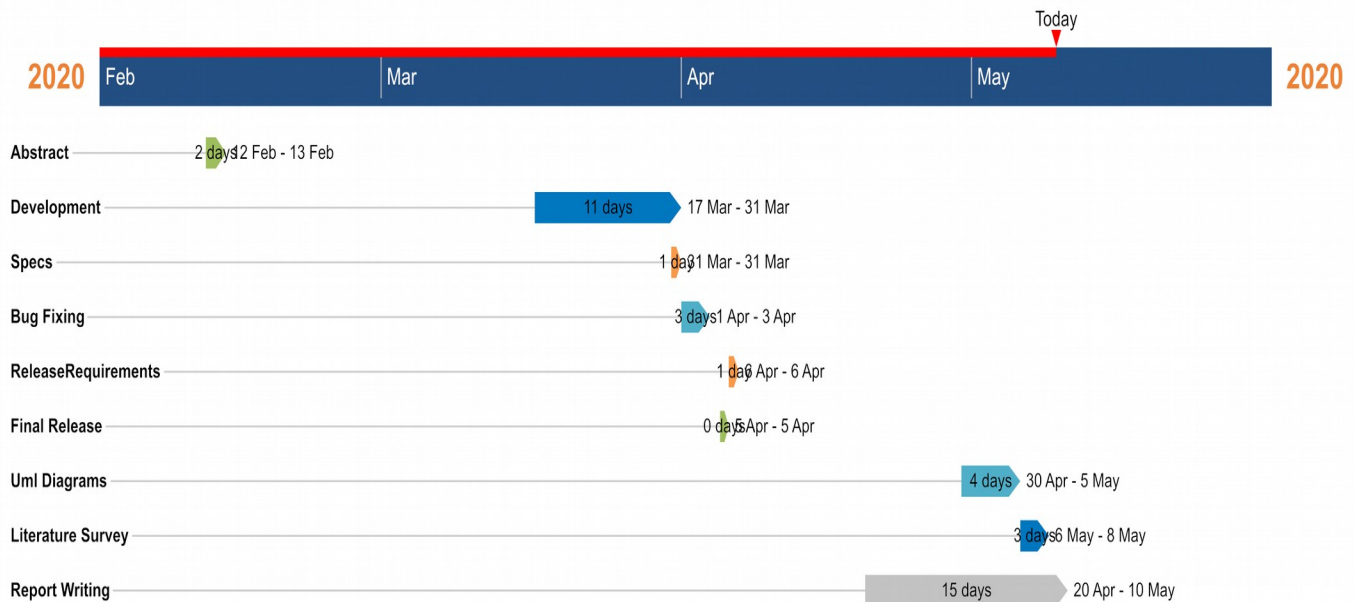
Date: 13/05/2020

LIST OF FIGURES

Fig -01 (Flowchart for insert)	Page - 13
Fig -02 (Flowchart for update)	Page - 14
Fig -03 (Flowchart for search)	Page - 15
Fig -04 (Flowchart for delete)	Page - 16
Fig -05 (Flowchart for display)	Page - 17
Fig -06 (Use case diagram)	Page - 18
Fig -07 (Class diagram)	Page - 19
Fig -08 (Sequence diagram)	Page - 20
Fig -09 (ER diagram)	Page - 21
Fig -10 (DB view-1)	Page - 53
Fig -11 (DB view2)	Page - 54
Fig -12 (Home)	Page - 55
Fig -13 (Insert)	Page - 56
Fig -14 (Update)	Page - 57
Fig -15 (Search)	Page - 58
Fig -16 (Delete)	Page - 59
Fig -17 (Display)	Page - 60

TIMELINE / GANTT CHART

Patient Information System



Gantt Chart

INTRODUCTION

1. INTRODUCTION

Patient Information System has been developed to keep track of patients visiting any hospitals or clinics for checkups, through this system one can access the previous visit histories of the patients by providing the id number if known. This system involves only three people's patient, the doctor and the receptionist. It helps to know the details of the previously visited doctors and the status of the patients who have undergone treatments under this doctors.

1.1 Problem Definition

- Problems with conventional system.
- The information is very difficult to retrieve and to find particular information like- E.g. - To find out about the patient's history, the user has to go through various registers.
- The information generated by various transactions takes time and efforts to be stored at right place.
- Various changes to information like patient details or immunization details of child are difficult to make as paper work is involved.
- Preparation of accurate and prompt reports: - This becomes a difficult task as information is difficult to collect from various registers.
- Manual calculations are error prone and take a lot of time this may result in incorrect information. For example calculation of patient's bill based on various treatments.

1.2 Project Overview

Patient Information System has been developed to keep track of patients visiting any hospitals or clinics for checkups, through this system one can access the previous visit histories of the patients by providing the id number if known. This system involves only three people's patient, the doctor and the receptionist. It helps to know the details of the previously visited doctors and the status of the patients who have undergone treatments under this doctors. Here the most important member is receptionist. He/She is the admin of the system and can add or delete the data's needed while registering a patient. Nowadays, every hospitals and clinics need this software to keep records. While entering data's it shows error message if it was an invalid entry. This system is very simple and easy to use for the users if he/she doesn't have any formal knowledge about the software. It contains the following tabs Insert, Update, Search, Delete, Display and Exit.

Features:

- This is totally offline based system. It maintains the records of patients from starting.
- To view the information of patient and details of doctor.
- Knowing the previous histories of patients

1.3 Hardware Specifications

Processor : Intel Core i3

Processor Speed : 2.00 GHz to 2.50 GHz

Ram : 512 MB

Hard Disk : 20 GB

1.4 Software Specifications

Operating System: Windows / Linux

Database : Sqlite 3

Language : Python 3

Framework : Tkinter

2. LITERATURE SURVEY

2.1 Existing System

The current manual system has a lot of paper work. To maintain the records of sale and service manually, is a Time-consuming task. With the increase in database, it will become a massive task to maintain the database. Requires large quantities of file cabinets, which are huge and require quite a bit of space in the office, which can be used for storing records of previous details. The retrieval of records of previously registered patients will be a tedious task. Lack of security for the records, anyone disarrange the records of your system. If someone want to check the details of the available doctors the previous system does not provide any necessary detail of this type.

2.2 Proposed System

Developing a system is feasible if and only if it is beneficial and removes all the drawbacks of the existing system and also enhances the way of operation to make the operation performing easy. The user should be satisfied with its functionality. If system does not fulfill this requirement then developing that system would be futile. My basic aim is to develop this system is to improve its functionality and removes all the drawback of existing system.

The proposed system can be summarized as:

- Input Model -
 1. Insert → Here we have to provide the information of the patients.
 2. Update → If any error occurs while entering the data this option allows you to update the wrong data.
 3. Search → This option allows the user to search for a patient by entering it's id.
 4. Delete → This option allows the user to delete a patient by entering its id.
 5. Display → In case the user don't know the patient's id then you can find the info in the database.
- Output Model -

The database contains each and every information about the patients, the user can retrieve every info from the database itself.

2.3 Feasibility Study

Feasibility is the determination of whether or not a project is worth doing the process followed making this determination is called feasibility study. This determines if a project can and should be taken. Once it has been determined that a project is feasible, the analyst can go ahead and prepare the project specification which finalizes project requirements. Generally, feasibility studies are undertaken within right time constraints and normally culminate in a written and oral feasibility report. The contents and recommendations of such a study will be used as a sound basis for deciding whether to proceed, postpone or cancel the project. Thus, since the feasibility study may lead to the commitment of large resources, it becomes necessary that it should be conducted competently and that no fundamental and that no fundamental errors of judgment are made.

There are following types of inter-related feasibility:

- Technical feasibility
- Operational feasibility
- Economic feasibility

Technical Feasibility → This is concerned with specifying equipment and software and hardware that will successfully satisfy the user requirement. The technical needs of the system may vary considerably, but might include:

- The facility to produce output in a given time.
- Response time under certain conditions.
- Ability to process a certain volume of transaction at a particular speed.
- Facility to communicate data to distant location.

According to the definition of technical feasibility the compatibility between front-end and back-end is very important. In our project the compatibility of both is very good. The compatibility of Python(Tkinter) and Sqlite3 is very good. The speed of output is very good, when we enter the data and click button then the response time is very fast and give result very quickly.

Operational Feasibility → It is mainly related to human organizational and political aspects. The points to be considered are:

1. What changes will be brought with the system?
2. What organization structures are disturbed?
3. What new skills will be required? Do the existing staff members have these skills? If not, can they be trained in due course of time?

At present stage all the work is done by manual. So, throughput and response time is too much. Finding out the detail regarding any information was very difficult, because data store in different books and each books at different places. In case of any problem, no one can solve the problem until the master of this field is not present. I will not change the structure of organization. I will deliver a system that will look like a current structure of organization. But the system, which is, delivered by me, removes all the overheads. All the computational work will be done automatically in our system. Response time is very quick.

Economic Feasibility → Economic analysis is the most frequently used technique for evaluating the effectiveness of a proposed system. More commonly known as cost/benefit analysis: the procedure is to determine the benefits and saving that are expected from a proposed system and compare them with cost. If benefits outweigh cost, a decision is taken to design and implement the system. Otherwise, further justification or alternative in the proposed system will have to be made if it is to have a chance of being approved. This is an ongoing effort that improves in accuracy at each phase of the system life cycle.

Hence the user will not find any difficulty at the installation time and after installation user also never find difficulty i.e. hang, slow speed or slow response time.

Patient Information System

SYSTEM ANALYSIS & DESIGN

3. SYSTEM ANALYSIS & DESIGN:

System analysis will be performed to determine if it is feasible to design information based on policies and plans of the organization and on user requirements and to eliminate the weaknesses of the present system.

- The new system should be cost effective.
- To augment management, improve productivity and services.
- To enhance user / system interface.
- To improve information quality and usability.
- To upgrade systems reliability, availability, flexibility and growth potential.

3.1 Requirement Specification

There are mainly two types of software requirement specifications:

- Functional Requirements
- Non-functional Requirements

Functional Requirements:

- A user shall be able to search the patient information list.
- The system shall generate a list of patients who have consulted with the specific doctors.
- Depending on the type of software, expected users and the type of system where the software is used.

Non-functional Requirements:

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.
- The PIS shall be available to all clinics and hospitals during normal working hours.

- The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized.

Patient Information System

3.2 Flowcharts/UML Diagrams/ERDs

Flowchart:

A flowchart is a graphical representation of the source code. It is the diagram that represents the workflow of the program.

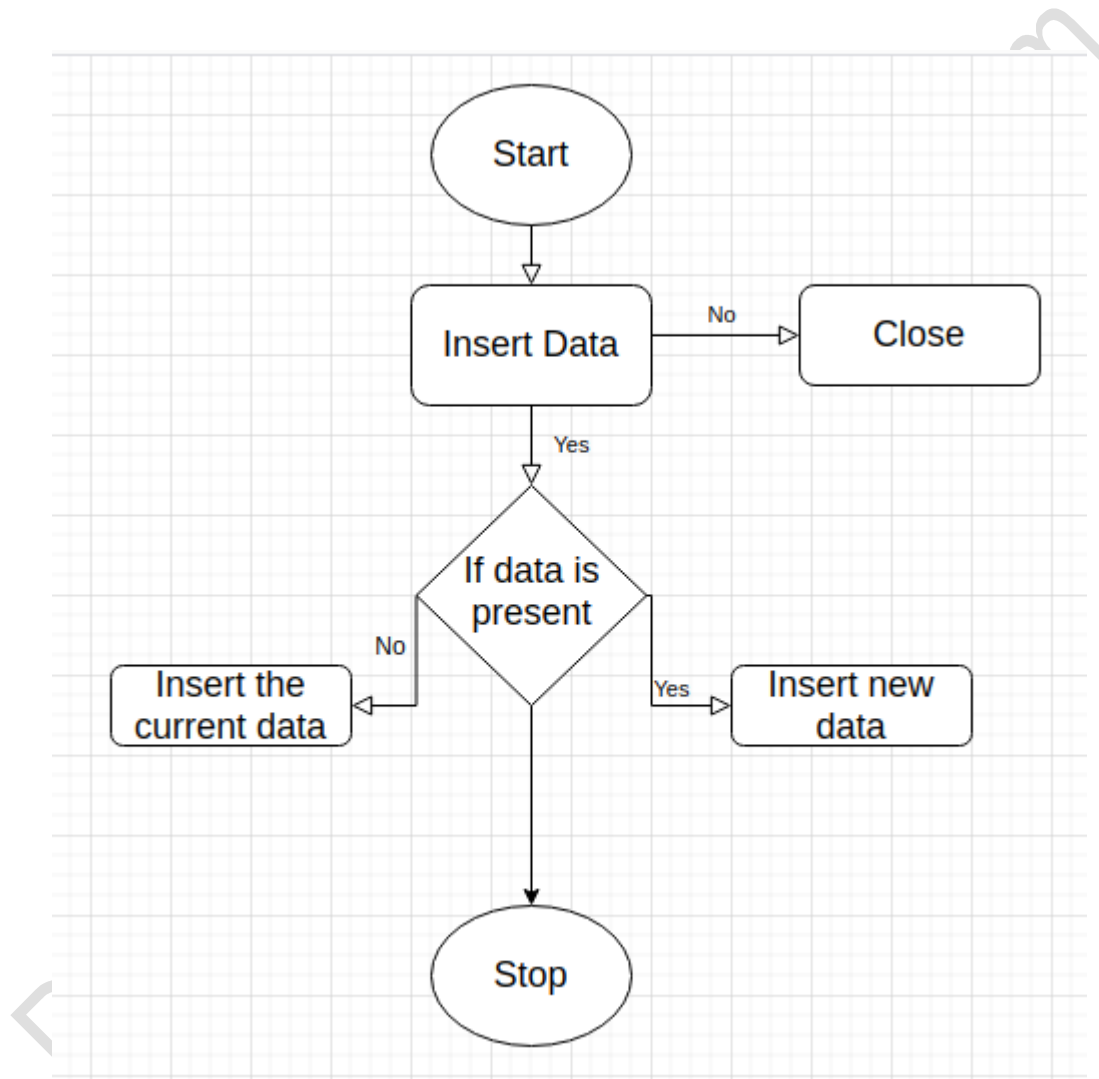


Fig-01

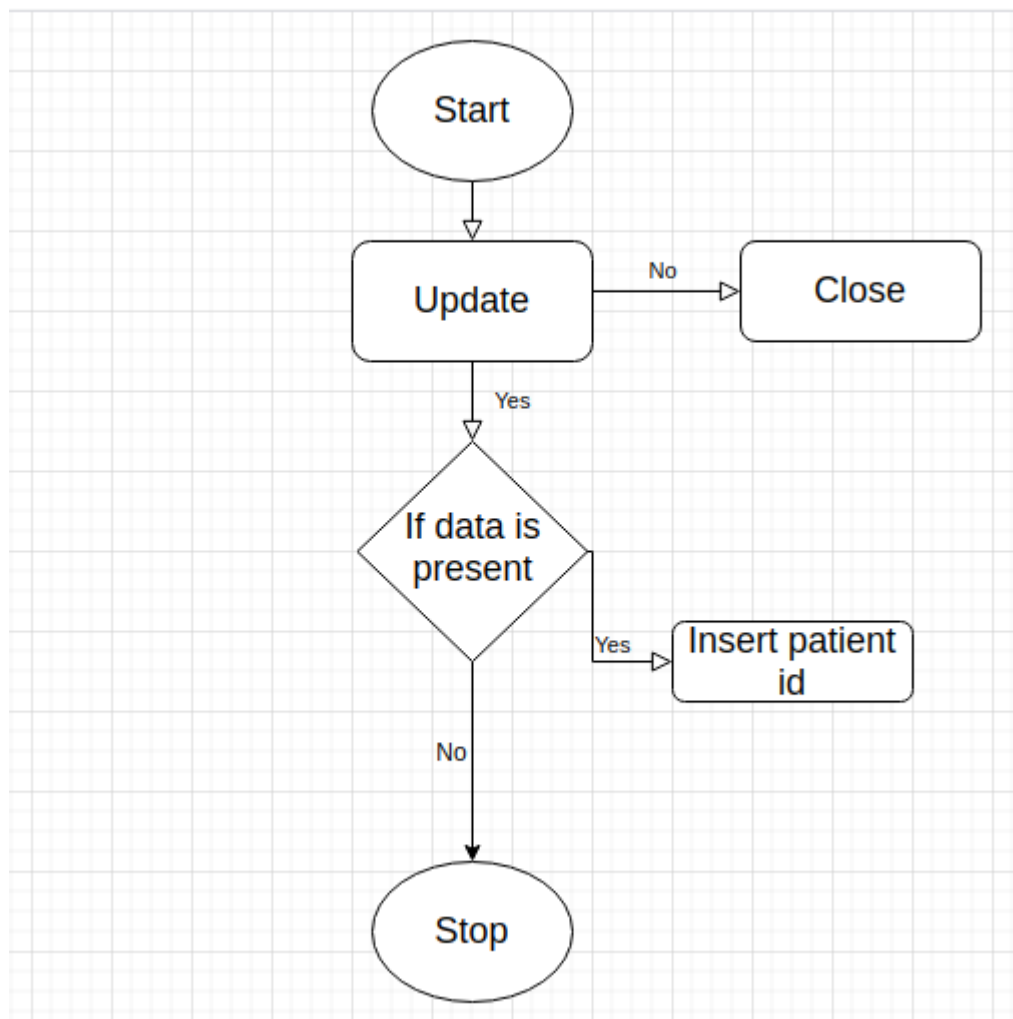


Fig-02

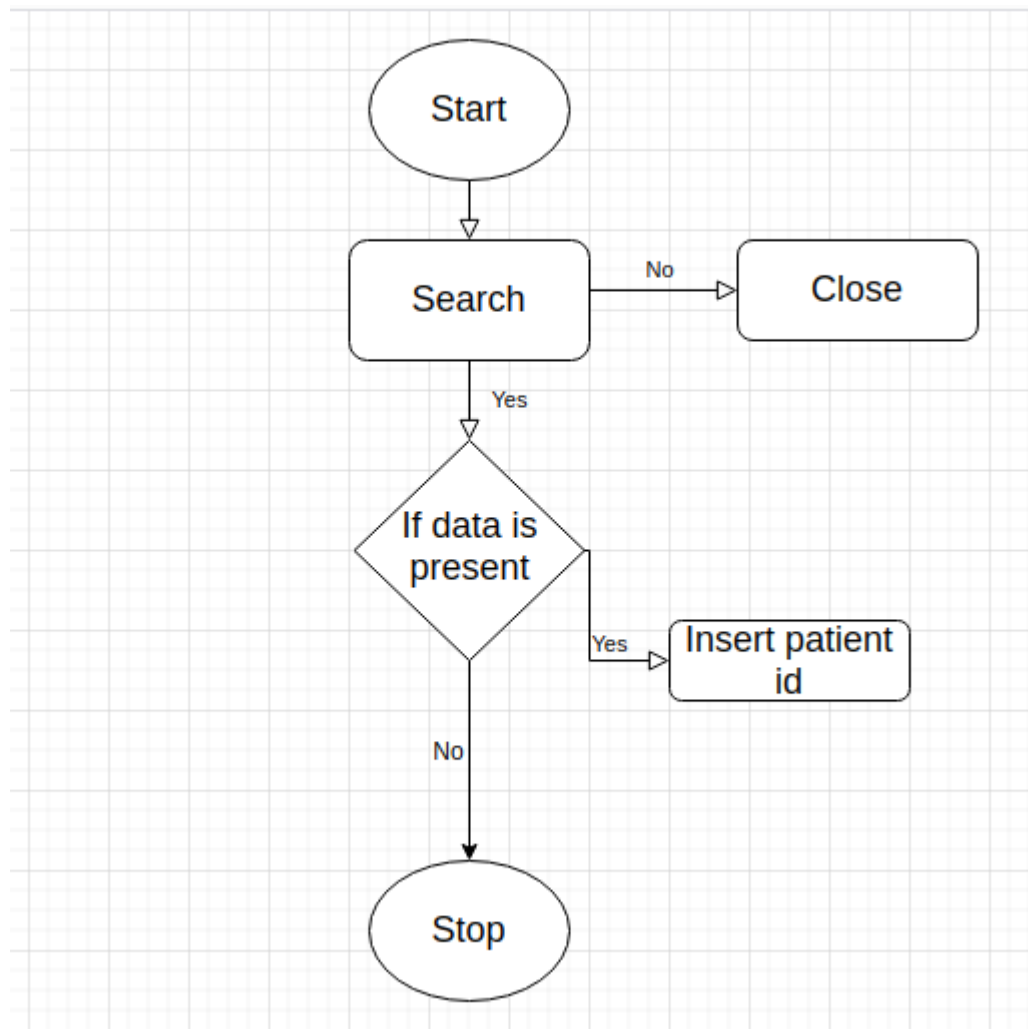


Fig-03

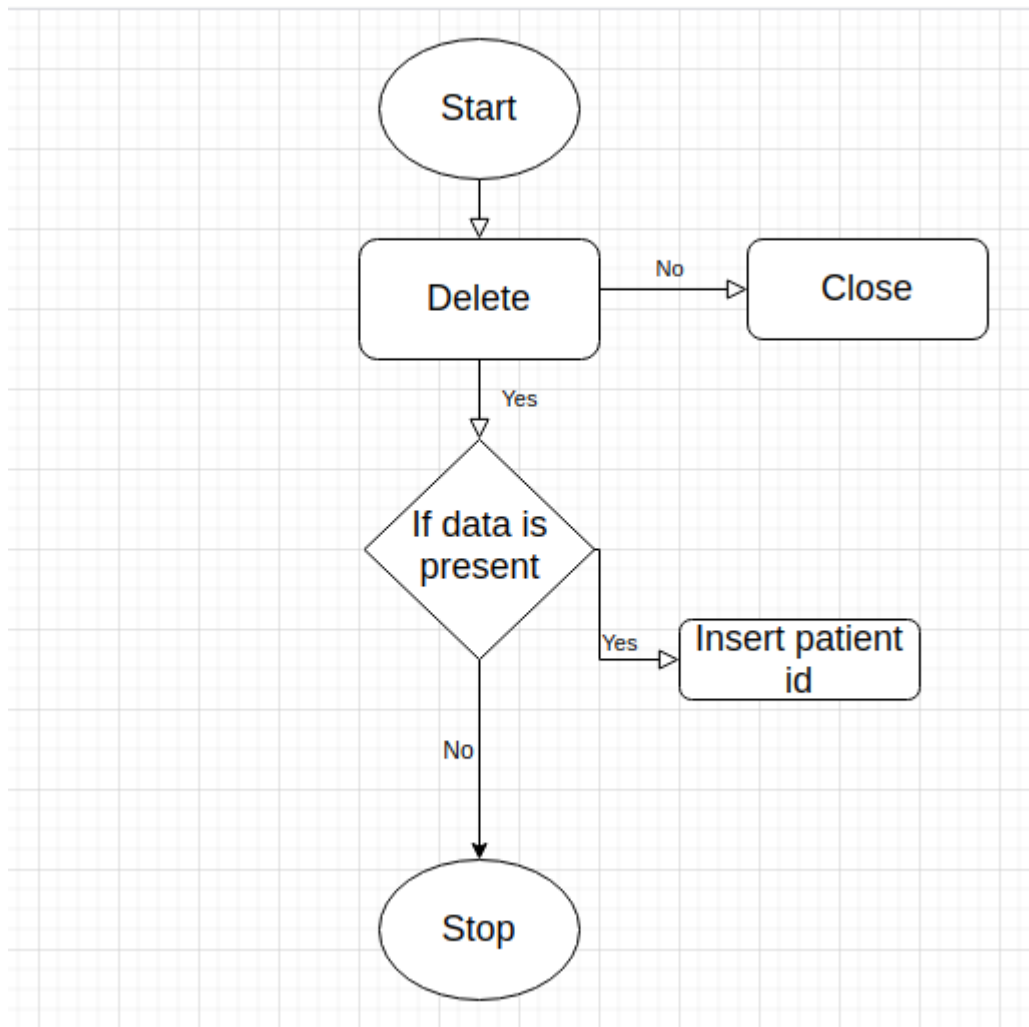


Fig-04

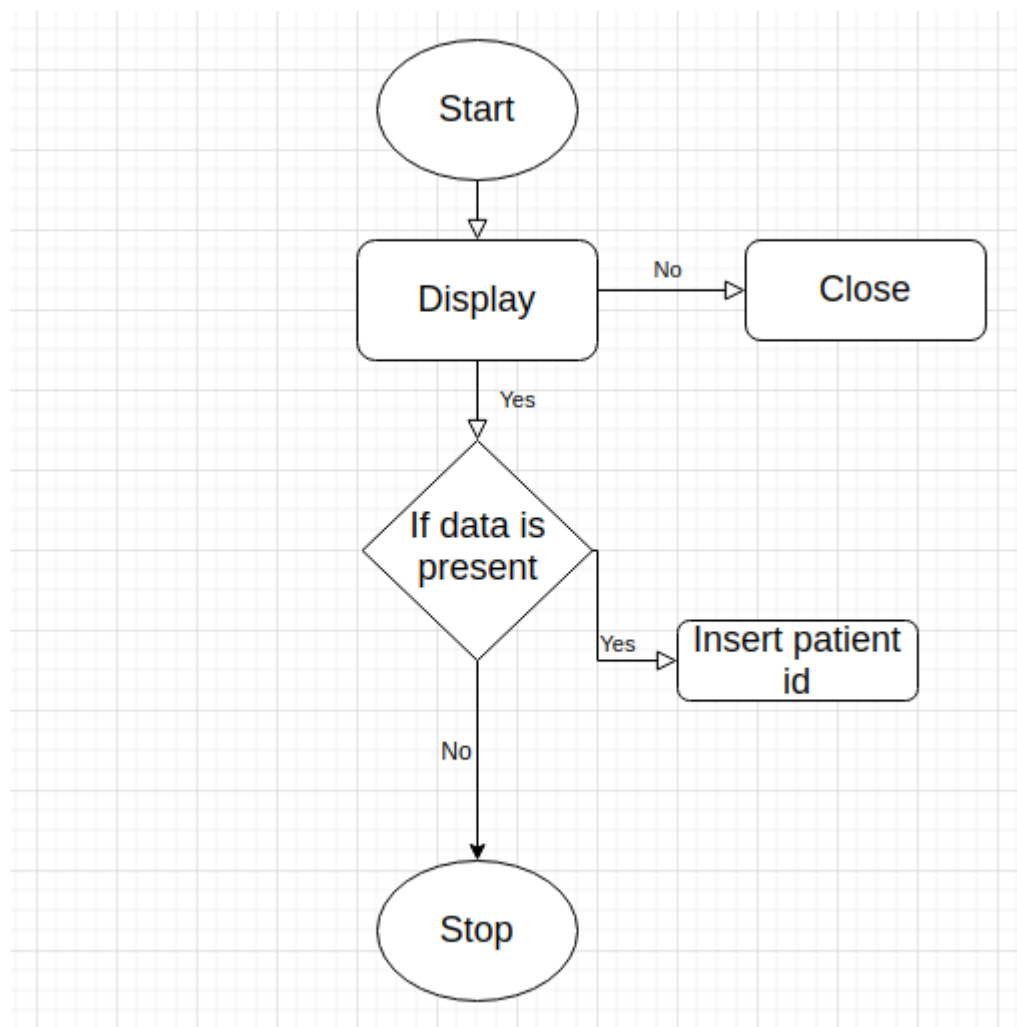


Fig-05

Uml Diagrams:

The Unified Modeling Language allows the programmer to precise an analysis model using the modeling notation that's governed by a group of syntactic semantic and pragmatic rules.

Over all Use Case

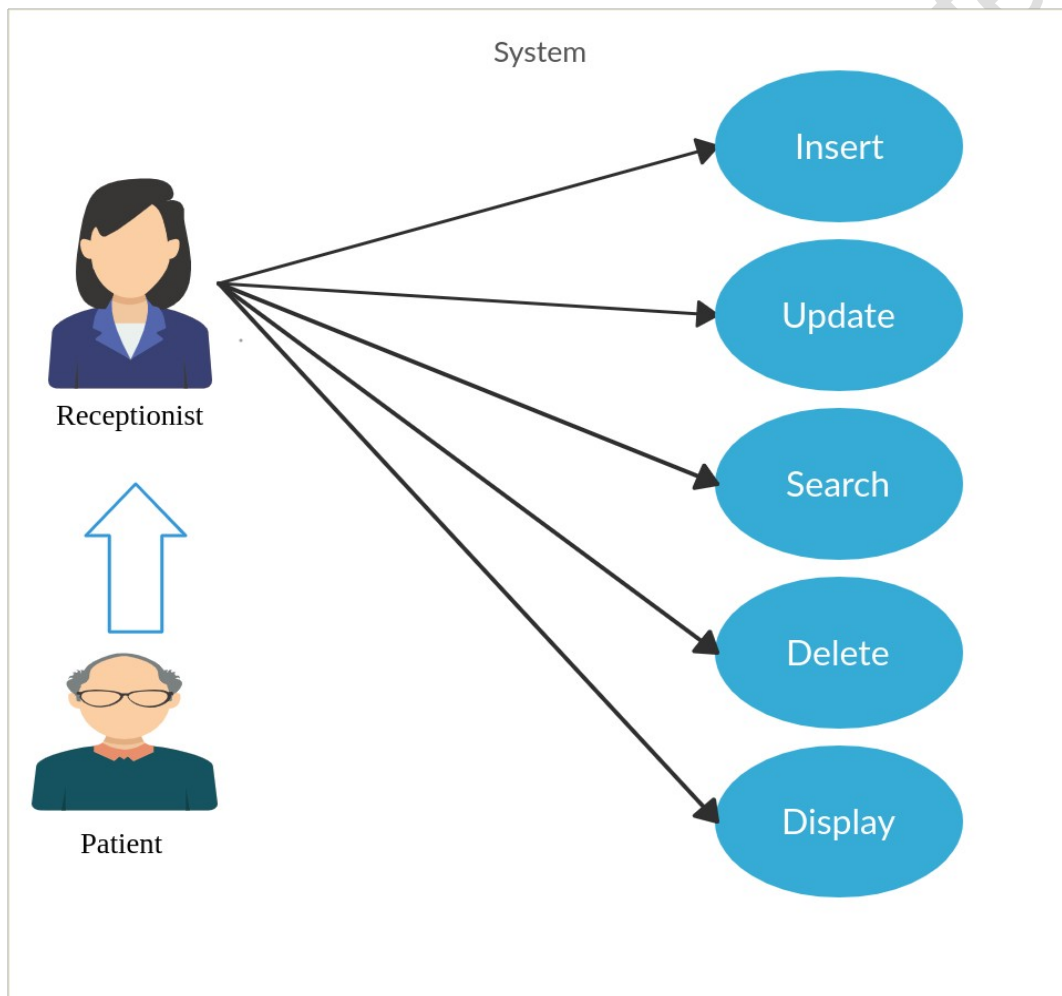


Fig -06

Class Diagram

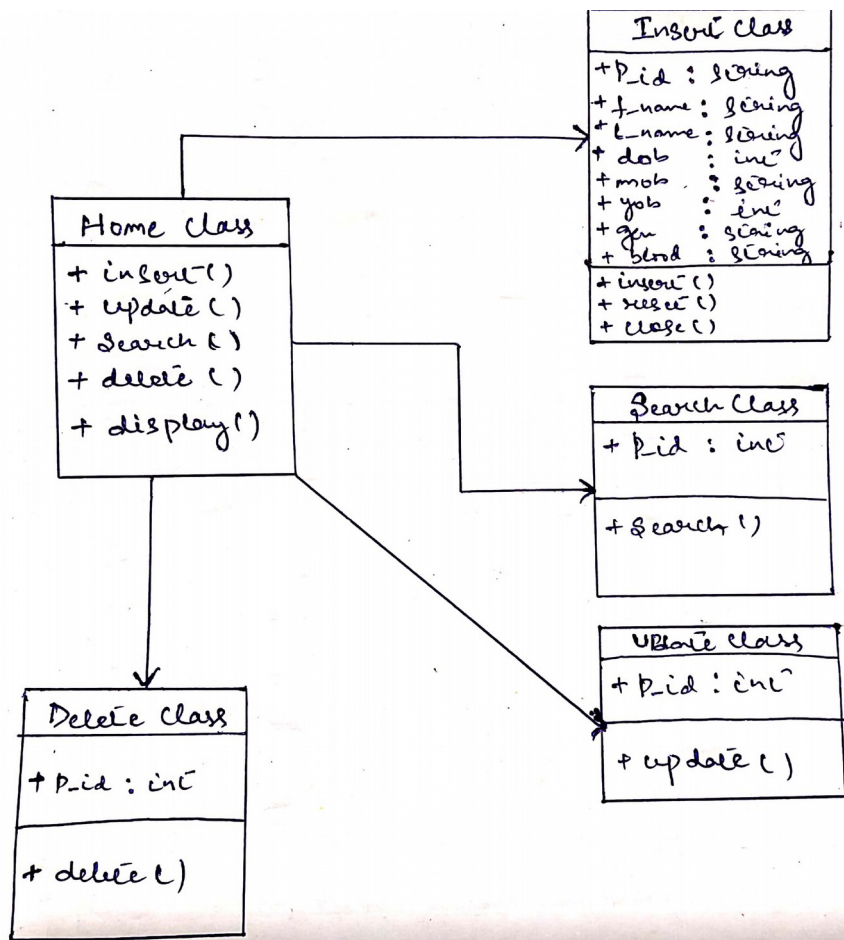


Fig -07

Sequence Diagram

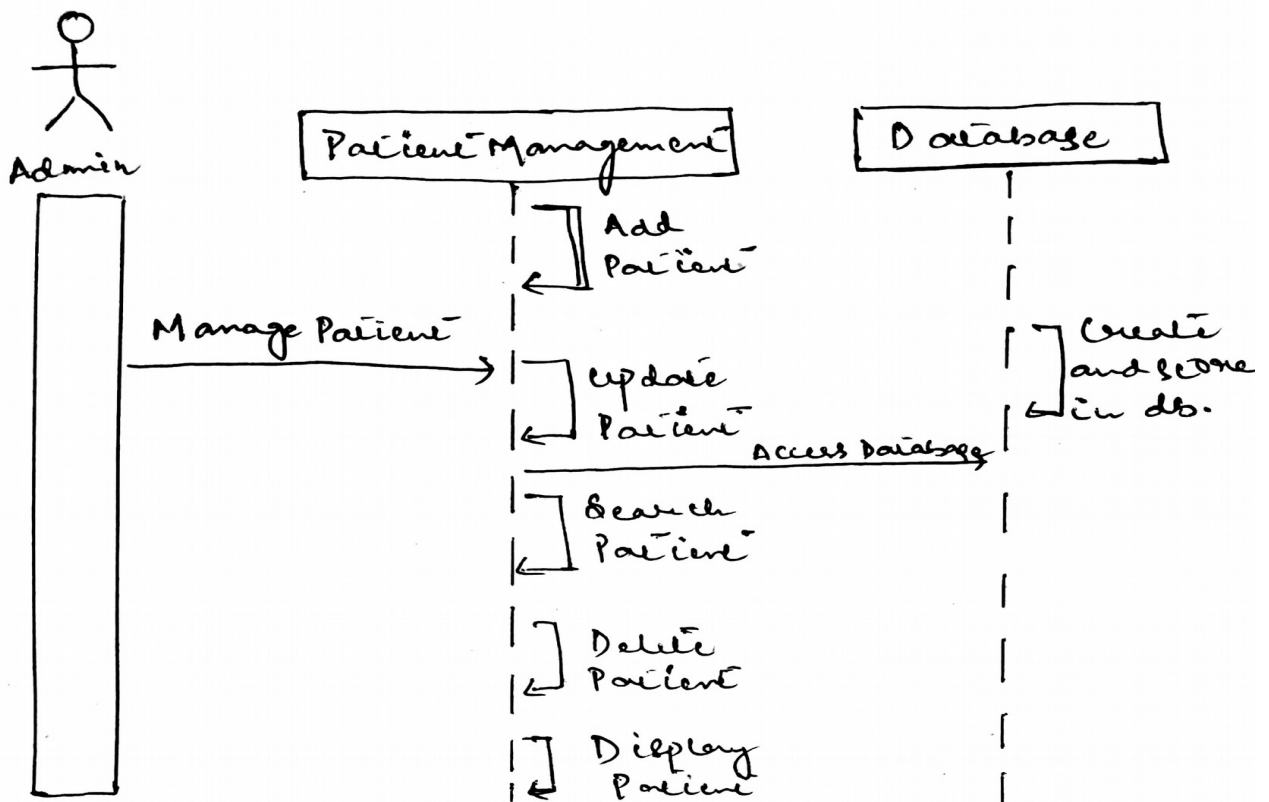


Fig-08

Patient Information System

ER Diagram

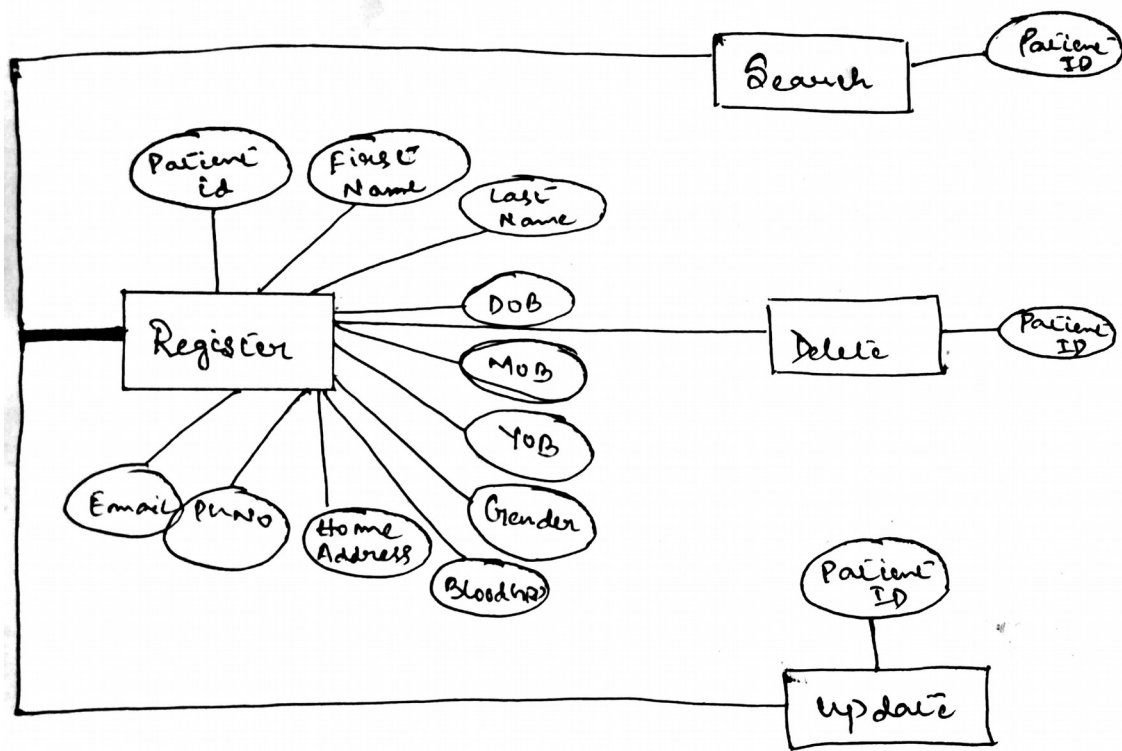


Fig - 09

3.3 Design and Test Steps

Software Implementation

Python

Python is an interpreted, high-level, programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of serious white space. The Python is designed to fulfill the following objectives:

- A simple and intuitive language even as powerful as major competitors.
- Open source so any one contribute to its development.
- Code that is understandable as plain english.[1]

Tkinter Framework

Python features a lot of GUI frameworks, but Tkinter is that 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 OS elements, so applications built with Tkinter appear as if 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.[2]

SQLite 3

SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine. It is the most used database engine in the world.

It is an in-process library and its code is publicly available. It is free for use

for any purpose, commercial or private. It is basically an embedded SQL database engine. Ordinary disk files can be easily read and write by SQLite because it does not have any separate server like SQL. The SQLite database file format is cross-platform so that anyone can easily copy a database between 32-bit and 64-bit systems. Due to all these features, it is a popular choice as an Application File Format.[3]

Testing

Testing is the process of detecting errors. Testing performs a really critical role for quality assurance and for ensuring the reliability of software. The results of testing are used afterward during maintenance also.[4]

Psychology of Testing

The aim of testing is usually to demonstrate that a program works by showing that it's no errors. The basic purpose of testing phase is to detect the errors which will be present within the program. Hence one shouldn't start testing with the intent of showing that a program works, but the intent should be to point out that a program doesn't work.

Testing is that the process of executing a program with the intent of finding errors.[4]

Testing Objectives:

The main objective of testing is to uncover a number of errors, systematically and with minimum effort and time. Stating formally, we can say,

- Testing may be a process of executing a program with the intent of finding a mistake .
- A successful test is one that uncovers an so far undiscovered error.
- An honest test suit is one that features a high probability of finding error, if it exists.
- The tests are inadequate to detect possibly present errors.

- The software more or less confirms to the standard and reliable standards.

3.4 Algorithm and Pseudo Code

3.4.1 Algorithm →

The following steps to be followed:

For Insert:

Step 1 → Click on the insert button to register a patient.

Step 2 → Fill the registration form.

Step 3 → Enter the patient id.

Step 4 → Enter the first name.

Step 5 → Enter the last name.

Step 6 → Enter the date of birth.

Step 7 → Enter the month of birth.

Step 8 → Enter the year of birth.

Step 9 → Enter the gender.

Step 10 → Enter the home address.

Step 11 → Enter the phone number.

Step 12 → Enter the email id.

Step 13 → Enter the blood group.

Step 14 → Enter the patient's history.

Step 15 → Enter the allocated doctor.

Step 16 → Press insert to register a patient in the database.

Step 17 → Press reset to enter another patient's info, else press close to exit.

For Update:

Step 1 → Click on the update button to update a patient's information.

Step 2 → Now enter the patient's id to update it.

Step 3 → Change the required part and press update.

For Search:

Step 1 → Click on the search button to search for a patient's information.

Step 2 → Now enter the patient's id to search it.

For Delete:

Step 1 → Click on the delete button to delete a patients information.

Step 2 → Now enter the patient's id to delete it.

For Display:

Step 1 → Click on the display button to display all patients information which are stored in the database.

3.4.2 Pseudo Code →

```
import tkinter
```

```
import tkinter.ttk
```

```
import tkinter.messagebox
```

```
import sqlite3
```

CLASS Database:

```
FUNCTION __init__(self):
```

```
    dbConnection <- sqlite3.connect("dbFile.db")
```

```
    dbCursor <- dbConnection.cursor()
```

```
ENDFUNCTION
```

```
FUNCTION __del__(self):
```

```
    dbCursor.close()
```

```
    dbConnection.close()
```

```
ENDFUNCTION
```

```
FUNCTION Insert(self, id, fName, lName, dob, mob, yob, gender, address, phone,  
email, bloodGroup, history, doctor):
```

```
    dbConnection.commit()
```

```
ENDFUNCTION
```

```
FUNCTION Update(self, fName, lName, dob, mob, yob, gender, address, phone,  
email, bloodGroup, history, doctor, id):
```

```
dbConnection.commit()
```

```
ENDFUNCTION
```

```
FUNCTION Search(self, id):
```

```
    searchResults <- dbCursor.fetchall()
```

```
    RETURN searchResults
```

```
ENDFUNCTION
```

```
FUNCTION Delete(self, id):
```

```
    dbConnection.commit()
```

```
ENDFUNCTION
```

```
FUNCTION Display(self):
```

```
    records <- dbCursor.fetchall()
```

```
    RETURN records
```

```
ENDFUNCTION
```

```
ENDCLASS
```

```
CLASS Values:
```

```
FUNCTION Validate(self, id, fName, lName, phone, email, history, doctor):
```

```
    IF not (id.isdigit() AND (len(id) = 3)):
```

```
        RETURN "id"
```

```

ELSEIF not (fName.isalpha()):
    RETURN "fName"

ELSEIF not (lName.isalpha()):
    RETURN "lName"

ELSEIF not (phone.isdigit() AND (len(phone) = 10)):
    RETURN "phone"

ELSEIF not (email.count("@") = 1 AND email.count(".") > 0):
    RETURN "email"

ELSEIF not (history.isalpha()):
    RETURN "history"

ELSEIF not (doctor.isalpha()):
    RETURN "doctor"

ELSE:
    RETURN "SUCCESS"

ENDIF

ENDFUNCTION

ENDCLASS

CLASS InsertWindow:

    FUNCTION __init__(self):

        window <- tkinter.Tk()

        window.wm_title("Insert data")

```

```

# Initializing all the variables

id <- tkinter.StringVar()

fName <- tkinter.StringVar()

lName <- tkinter.StringVar()

address <- tkinter.StringVar()

phone <- tkinter.StringVar()

email <- tkinter.StringVar()

history <- tkinter.StringVar()

doctor <- tkinter.StringVar()

genderList <- ["Male", "Female", "Transgender", "Other"]

dateList <- list(range(1, 32))

monthList <- ["January", "February", "March", "April", "May", "June", "July",
"August", "September", "October", "November", "December"]

yearList <- list(range(1900, 2020))

bloodGroupList <- ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]

# Labels

tkinter.Label( window, text <- "Patient ID", width <- 25).grid(pady <- 5, column <-
1, row <- 1)

tkinter.Label( window, text <- "First Name", width <- 25).grid(pady <- 5, column
<- 1, row <- 2)

tkinter.Label( window, text <- "Last Name", width <- 25).grid(pady <- 5, column <-
1, row <- 3)

```

```

tkinter.Label( window, text <- "D.O.B", width <- 25).grid(pady <- 5, column <- 1,
row <- 4)

tkinter.Label( window, text <- "M.O.B", width <- 25).grid(pady <- 5, column <- 1,
row <- 5)

tkinter.Label( window, text <- "Y.O.B", width <- 25).grid(pady <- 5, column <- 1,
row <- 6)

tkinter.Label( window, text <- "Gender", width <- 25).grid(pady <- 5, column <- 1,
row <- 7)

tkinter.Label( window, text <- "Home Address", width <- 25).grid(pady <- 5,
column <- 1, row <- 8)

tkinter.Label( window, text <- "Phone Number", width <- 25).grid(pady <- 5,
column <- 1, row <- 9)

tkinter.Label( window, text <- "Email ID", width <- 25).grid(pady <- 5, column <-
1, row <- 10)

tkinter.Label( window, text <- "Blood Group", width <- 25).grid(pady <- 5, column
<- 1, row <- 11)

tkinter.Label( window, text <- "Patient History", width <- 25).grid(pady <- 5,
column <- 1, row <- 12)

tkinter.Label( window, text <- "Doctor", width <- 25).grid(pady <- 5, column <- 1,
row <- 13)

# Fields

# Entry widgets

idEntry <- tkinter.Entry( window, width <- 25, textvariable <- id)

```



```

fNameEntry <- tkinter.Entry( window, width <- 25, textvariable <- fName)
lNameEntry <- tkinter.Entry( window, width <- 25, textvariable <- lName)
addressEntry <- tkinter.Entry( window, width <- 25, textvariable <- address)
phoneEntry <- tkinter.Entry( window, width <- 25, textvariable <- phone)
emailEntry <- tkinter.Entry( window, width <- 25, textvariable <- email)
historyEntry <- tkinter.Entry( window, width <- 25, textvariable <- history)
doctorEntry <- tkinter.Entry( window, width <- 25, textvariable <- doctor)
idEntry.grid(pady <- 5, column <- 3, row <- 1)
fNameEntry.grid(pady <- 5, column <- 3, row <- 2)
lNameEntry.grid(pady <- 5, column <- 3, row <- 3)
addressEntry.grid(pady <- 5, column <- 3, row <- 8)
phoneEntry.grid(pady <- 5, column <- 3, row <- 9)
emailEntry.grid(pady <- 5, column <- 3, row <- 10)
historyEntry.grid(pady <- 5, column <- 3, row <- 12)
doctorEntry.grid(pady <- 5, column <- 3, row <- 13)
# Combobox widgets
dobBox <- tkinter.ttk.Combobox( window, values <- dateList, width <- 20)
mobBox <- tkinter.ttk.Combobox( window, values <- monthList, width <- 20)
yobBox <- tkinter.ttk.Combobox( window, values <- yearList, width <- 20)
genderBox <- tkinter.ttk.Combobox( window, values <- genderList, width <- 20)
bloodGroupBox <- tkinter.ttk.Combobox( window, values <- bloodGroupList,
width <- 20)
dobBox.grid(pady <- 5, column <- 3, row <- 4)

```

```

mobBox.grid(pady <- 5, column <- 3, row <- 5)

yobBox.grid(pady <- 5, column <- 3, row <- 6)

genderBox.grid(pady <- 5, column <- 3, row <- 7)

bloodGroupBox.grid(pady <- 5, column <- 3, row <- 11)

# Button widgets

tkinter.Button( window, width <- 20, text <- "Insert", command <- Insert).grid(pady
<- 15, padx <- 5, column <- 1, row <- 14)

tkinter.Button( window, width <- 20, text <- "Reset", command <- Reset).grid(pady
<- 15, padx <- 5, column <- 2, row <- 14)

tkinter.Button( window, width <- 20, text <- "Close", command <-
window.destroy).grid(pady <- 15, padx <- 5, column <- 3, row <- 14)

window.mainloop()

ENDFUNCTION

FUNCTION Insert(self):
    values <- Values()
    database <- Database()
    test <- values.Validate( idEntry.get(), fNameEntry.get(), lNameEntry.get(),
self.phoneEntry.get(), self.emailEntry.get(), self.historyEntry.get(),
self.doctorEntry.get())

    IF ( test = "SUCCESS"):
        database.Insert( idEntry.get(), fNameEntry.get(), lNameEntry.get(),
dobBox.get(), self.mobBox.get(), self.yobBox.get(), self.genderBox.get(),

```

```

self.addressEntry.get(), self.phoneEntry.get(), self.emailEntry.get(),
self.bloodGroupBox.get(), self.historyEntry.get(), self.doctorEntry.get())

ELSE:

    valueErrorMessage <- "Invalid input in field " + test

    tkinter.messagebox.showerror("Value Error", valueErrorMessage)

ENDIF

ENDFUNCTION

FUNCTION Reset(self):

    idEntry.delete(0, tkinter.END)

    fNameEntry.delete(0, tkinter.END)

    lNameEntry.delete(0, tkinter.END)

    dobBox.set("")

    mobBox.set("")

    yobBox.set("")

    genderBox.set("")

    addressEntry.delete(0, tkinter.END)

    phoneEntry.delete(0, tkinter.END)

    emailEntry.delete(0, tkinter.END)

    bloodGroupBox.set("")

    historyEntry.delete(0, tkinter.END)

    doctorEntry.delete(0, tkinter.END)

ENDFUNCTION

```

```
ENDCLASS
```

```
CLASS UpdateWindow:
```

```
    FUNCTION __init__(self, id):
```

```
        window <- tkinter.Tk()
```

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

```
        # Initializing all the variables
```

```
        id <- id
```

```
        fName <- tkinter.StringVar()
```

```
        lName <- tkinter.StringVar()
```

```
        address <- tkinter.StringVar()
```

```
        phone <- tkinter.StringVar()
```

```
        email <- tkinter.StringVar()
```

```
        history <- tkinter.StringVar()
```

```
        doctor <- tkinter.StringVar()
```

```
        genderList <- ["Male", "Female", "Transgender", "Other"]
```

```
        dateList <- list(range(1, 32))
```

```
        monthList <- ["January", "February", "March", "April", "May", "June", "July",  
"August", "September", "October", "November", "December"]
```

```
        yearList <- list(range(1900, 2020))
```

```
        bloodGroupList <- ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]
```

```
        # Labels
```

```

tkinter.Label( window, text <- "Patient ID", width <- 25).grid(pady <- 5, column <-
1, row <- 1)

tkinter.Label( window, text <- id, width <- 25).grid(pady <- 5, column <- 3, row <-
1)

tkinter.Label( window, text <- "First Name", width <- 25).grid(pady <- 5, column
<- 1, row <- 2)

tkinter.Label( window, text <- "Last Name", width <- 25).grid(pady <- 5, column <-
1, row <- 3)

tkinter.Label( window, text <- "D.O.B", width <- 25).grid(pady <- 5, column <- 1,
row <- 4)

tkinter.Label( window, text <- "M.O.B", width <- 25).grid(pady <- 5, column <- 1,
row <- 5)

tkinter.Label( window, text <- "Y.O.B", width <- 25).grid(pady <- 5, column <- 1,
row <- 6)

tkinter.Label( window, text <- "Gender", width <- 25).grid(pady <- 5, column <- 1,
row <- 7)

tkinter.Label( window, text <- "Home Address", width <- 25).grid(pady <- 5,
column <- 1, row <- 8)

tkinter.Label( window, text <- "Phone Number", width <- 25).grid(pady <- 5,
column <- 1, row <- 9)

tkinter.Label( window, text <- "Email ID", width <- 25).grid(pady <- 5, column <-
1, row <- 10)

```

```

tkinter.Label( window, text <- "Blood Group", width <- 25).grid(pady <- 5, column
<- 1, row <- 11)

tkinter.Label( window, text <- "Patient History", width <- 25).grid(pady <- 5,
column <- 1, row <- 12)

tkinter.Label( window, text <- "Doctor", width <- 25).grid(pady <- 5, column <- 1,
row <- 13)

# Set previous values

database <- Database()

searchResults <- database.Search(id)

tkinter.Label( window, text <- searchResults[0][1], width <- 25).grid(pady <- 5,
column <- 2, row <- 2)

tkinter.Label( window, text <- searchResults[0][2], width <- 25).grid(pady <- 5,
column <- 2, row <- 3)

tkinter.Label( window, text <- searchResults[0][3], width <- 25).grid(pady <- 5,
column <- 2, row <- 4)

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

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

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

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

```

```

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

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

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

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

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

# Fields

# Entry widgets

fNameEntry <- tkinter.Entry( window, width <- 25, textvariable <- fName)
lNameEntry <- tkinter.Entry( window, width <- 25, textvariable <- lName)
addressEntry <- tkinter.Entry( window, width <- 25, textvariable <- address)
phoneEntry <- tkinter.Entry( window, width <- 25, textvariable <- phone)
emailEntry <- tkinter.Entry( window, width <- 25, textvariable <- email)
historyEntry <- tkinter.Entry( window, width <- 25, textvariable <- history)
doctorEntry <- tkinter.Entry( window, width <- 25, textvariable <- doctor)

fNameEntry.grid(pady <- 5, column <- 3, row <- 2)
lNameEntry.grid(pady <- 5, column <- 3, row <- 3)
addressEntry.grid(pady <- 5, column <- 3, row <- 8)
phoneEntry.grid(pady <- 5, column <- 3, row <- 9)

```

```

emailEntry.grid(pady <- 5, column <- 3, row <- 10)

historyEntry.grid(pady <- 5, column <- 3, row <- 12)

doctorEntry.grid(pady <- 5, column <- 3, row <- 13)

# Combobox widgets

dobBox <- tkinter.ttk.Combobox( window, values <- dateList, width <- 20)

mobBox <- tkinter.ttk.Combobox( window, values <- monthList, width <- 20)

yobBox <- tkinter.ttk.Combobox( window, values <- yearList, width <- 20)

genderBox <- tkinter.ttk.Combobox( window, values <- genderList, width <- 20)

bloodGroupBox <- tkinter.ttk.Combobox( window, values <- bloodGroupList,
width <- 20)

dobBox.grid(pady <- 5, column <- 3, row <- 4)

mobBox.grid(pady <- 5, column <- 3, row <- 5)

yobBox.grid(pady <- 5, column <- 3, row <- 6)

genderBox.grid(pady <- 5, column <- 3, row <- 7)

bloodGroupBox.grid(pady <- 5, column <- 3, row <- 11)

# Button widgets

tkinter.Button( window, width <- 20, text <- "Update", command <-
Update).grid(pady <- 15, padx <- 5, column <- 1, row <- 14)

tkinter.Button( window, width <- 20, text <- "Reset", command <- Reset).grid(pady
<- 15, padx <- 5, column <- 2, row <- 14)

tkinter.Button( window, width <- 20, text <- "Close", command <-
window.destroy).grid(pady <- 15, padx <- 5, column <- 3, row <- 14)

window.mainloop()

```



```
ENDFUNCTION
```

```
FUNCTION Update(self):
```

```
    database <- Database()
```

```
    database.Update( fNameEntry.get(), lNameEntry.get(), dobBox.get(),  
mobBox.get(), self.yobBox.get(), self.genderBox.get(), self.addressEntry.get(),  
self.phoneEntry.get(), self.emailEntry.get(), self.bloodGroupBox.get(),  
self.historyEntry.get(), self.doctorEntry.get(), self.id)
```

```
ENDFUNCTION
```

```
FUNCTION Reset(self):
```

```
    fNameEntry.delete(0, tkinter.END)
```

```
    lNameEntry.delete(0, tkinter.END)
```

```
    dobBox.set("")
```

```
    mobBox.set("")
```

```
    yobBox.set("")
```

```
    genderBox.set("")
```

```
    addressEntry.delete(0, tkinter.END)
```

```
    phoneEntry.delete(0, tkinter.END)
```

```
    emailEntry.delete(0, tkinter.END)
```

```
    bloodGroupBox.set("")
```

```
    historyEntry.delete(0, tkinter.END)
```

```
    doctorEntry.delete(0, tkinter.END)
```

ENDFUNCTION

ENDCLASS

CLASS DatabaseView:

```
FUNCTION __init__(self, data):  
    databaseViewWindow <- tkinter.Tk()  
    databaseViewWindow.wm_title("Database View")  
    # Label widgets  
    tkinter.Label( databaseViewWindow, text <- "Database View Window", width <-  
25).grid(pady <- 5, column <- 1, row <- 1)  
    databaseView <- tkinter.ttk.Treeview( databaseViewWindow)  
    databaseView.grid(pady <- 5, column <- 1, row <- 2)  
    databaseView["show"] <- "headings"  
    databaseView["columns"] <- ("id", "fName", "lName", "dob", "mob", "yob",  
"gender", "address", "phone", "email", "bloodGroup", "history", "doctor")  
    # Treeview column headings  
    databaseView.heading("id", text <- "ID")  
    databaseView.heading("fName", text <- "First Name")  
    databaseView.heading("lName", text <- "Last Name")  
    databaseView.heading("dob", text <- "D.O.B")  
    databaseView.heading("mob", text <- "M.O.B")  
    databaseView.heading("yob", text <- "Y.O.B")
```

```

databaseView.heading("gender", text <- "Gender")
databaseView.heading("address", text <- "Home Address")
databaseView.heading("phone", text <- "Phone Number")
databaseView.heading("email", text <- "Email ID")
databaseView.heading("bloodGroup", text <- "Blood Group")
databaseView.heading("history", text <- "History")
databaseView.heading("doctor", text <- "Doctor")

# Treeview columns

databaseView.column("id", width <- 40)
databaseView.column("fName", width <- 100)
databaseView.column("lName", width <- 100)
databaseView.column("dob", width <- 60)
databaseView.column("mob", width <- 60)
databaseView.column("yob", width <- 60)
databaseView.column("gender", width <- 60)
databaseView.column("address", width <- 200)
databaseView.column("phone", width <- 100)
databaseView.column("email", width <- 200)
databaseView.column("bloodGroup", width <- 100)
databaseView.column("history", width <- 100)
databaseView.column("doctor", width <- 100)

for record in data:

    databaseView.insert("", 'end', values=(record))

```

```

        ENDFOR

        databaseViewWindow.mainloop()

    ENDFUNCTION

ENDCLASS

CLASS SearchDeleteWindow:

    FUNCTION __init__(self, task):

        window <- tkinter.Tk()

        window.wm_title(task + " data")

        # Initializing all the variables

        id <- tkinter.StringVar()

        fName <- tkinter.StringVar()

        lName <- tkinter.StringVar()

        heading <- "Please enter Patient ID to " + task

        # Labels

        tkinter.Label(window, text <- heading, width <- 50).grid(pady <- 20, row <- 1)

        tkinter.Label(window, text <- "Patient ID", width <- 10).grid(pady <- 5, row <- 2)

        # Entry widgets

        idEntry <- tkinter.Entry(window, width <- 5, textvariable <- id)

        idEntry.grid(pady <- 5, row <- 3)

        # Button widgets

        IF (task = "Search"):

```

```
tkinter.Button(window, width <- 20, text <- task, command <- Search).grid(pady  
<- 15, padx <- 5, column <- 1, row <- 14)
```

```
ELSEIF (task = "Delete"):
```

```
tkinter.Button(window, width <- 20, text <- task, command <- Delete).grid(pady  
<- 15, padx <- 5, column <- 1, row <- 14)
```

```
ENDIF
```

```
ENDFUNCTION
```

```
FUNCTION Search(self):
```

```
database <- Database()
```

```
data <- database.Search( idEntry.get())
```

```
databaseView <- DatabaseView( data)
```

```
ENDFUNCTION
```

```
FUNCTION Delete(self):
```

```
database <- Database()
```

```
database.Delete( idEntry.get())
```

```
ENDFUNCTION
```

```
ENDCLASS
```

```
CLASS HomePage:
```

```
FUNCTION __init__(self):
```

```

homePageWindow <- tkinter.Tk()

homePageWindow.wm_title("Patient Information System")

ENDFOR

tkinter.Label( homePageWindow, text <- "Home Page", width <- 100).grid(pady <-
20, column <- 1, row <- 1)

tkinter.Button( homePageWindow, width <- 20, text <- "Insert", command <-
Insert).grid(pady <- 15, column <- 1, row <- 2)

tkinter.Button( homePageWindow, width <- 20, text <- "Update", command <-
Update).grid(pady <- 15, column <- 1, row <- 3)

tkinter.Button( homePageWindow, width <- 20, text <- "Search", command <-
Search).grid(pady <- 15, column <- 1, row <- 4)

tkinter.Button( homePageWindow, width <- 20, text <- "Delete", command <-
Delete).grid(pady <- 15, column <- 1, row <- 5)

tkinter.Button( homePageWindow, width <- 20, text <- "Display", command <-
Display).grid(pady <- 15, column <- 1, row <- 6)

tkinter.Button( homePageWindow, width <- 20, text <- "Exit", command <-
homePageWindow.destroy).grid(pady <- 15, column <- 1, row <- 7)

application <- HomePage

homePageWindow.mainloop()

ENDFUNCTION

FUNCTION Insert(self):

insertWindow <- InsertWindow()

```

ENDFUNCTION

FUNCTION Update(self):

 updateIDWindow <- tkinter.Tk()

 updateIDWindow.wm_title("Update data")

 # Initializing all the variables

 id <- tkinter.StringVar()

 # Label

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

 # Entry widgets

 idEntry <- tkinter.Entry(updateIDWindow, width <- 5, textvariable <- id)

 idEntry.grid(pady <- 10, row <- 2)

 # Button widgets

 tkinter.Button(updateIDWindow, width <- 20, text <- "Update", command <-
updateID).grid(pady <- 10, row <- 3)

 updateIDWindow.mainloop()

ENDFUNCTION

FUNCTION updateID(self):

 updateWindow <- UpdateWindow(idEntry.get())

 updateIDWindow.destroy()

ENDFUNCTION

```
FUNCTION Search(self):
```

```
    searchWindow <- SearchDeleteWindow("Search")
```

```
ENDFUNCTION
```

```
FUNCTION Delete(self):
```

```
    deleteWindow <- SearchDeleteWindow("Delete")
```

```
ENDFUNCTION
```

```
FUNCTION Display(self):
```

```
    database <- Database()
```

```
    data <- database.Display()
```

```
    displayWindow <- DatabaseView( data)
```

```
ENDFUNCTION
```

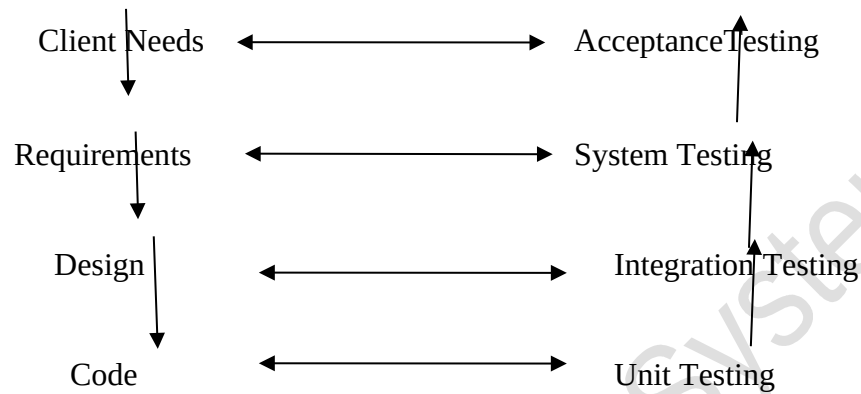
```
ENDCLASS
```

```
homePage <- HomePage()
```

3.5 TEST PROCESS

Levels Of Testing

In order to uncover the errors present in several phases we've the concept of levels of testing. The basic levels of testing are



A series of testing is completed for the proposed system before the system is prepared for the user acceptance testing.

The steps involved in Testing are:

Unit Testing:

Unit testing focuses verification efforts on the littlest unit of the software design, the module. This is also known as “Module Testing”. The modules are tested separately. This testing carried out during programming stage itself. In this testing each module is found to be working satisfactorily as regards to the expected output from the module.

Integration Testing:

Data are often grossed across an interface; one module can have adverse efforts on another. Integration testing is systematic testing for construction the program structure while at an equivalent time conducting tests to uncover errors related to within the interface. The objective is to require unit tested modules and build a program structure. All the modules are combined and tested as an entire . Here correction is difficult because the isolation of cause is complicate by the vast expense of the whole program. Thus within the

integration testing stop, all the errors uncovered are corrected for the text testing steps.

System testing:

System testing is that the stage of implementation that's aimed toward ensuring that the system works accurately and efficiently for live operation commences. Testing is significant to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct, then goal are going to be successfully achieved.

The four major system testing are

- Recovery Testing
- Security Testing
- Stress Testing
- Performance Testing

Recovery Testing

Recovery testing may be a system test that forces the software to fail during a sort of ways and verifies that recovery is correctly performed. If recovery is automatic, re-initialization, checkpoint mechanisms, data recovery, and restart are each evaluated for correctness. If recovery requires human intervention, the mean solar time to repair is evaluated to work out whether it's within acceptable limits.

Security Testing

Security testing attempts to verify that protection mechanisms built into a system will, in fact, protect it from improper penetration. During security testing, the tester plays the role of the individual who desire to penetrate the system. Given enough time and resources, good security testing will ultimately penetrate

a system. The role of the system designer is to form penetration cost quite the worth of the knowledge which will be obtained.

Stress Testing

During earlier software testing steps, white box and recorder techniques resulted during a radical evaluation of normal program functions and performance. Stress tests are designed to confront programs with abnormal situations.

Performance Testing

For real-time and embedded systems, software that gives required function but doesn't inform performance requirements is unacceptable. Performance testing is meant to check the run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps within the testing process.

Performance tests are sometimes including stress testing and sometimes require both hardware and software instrumentation. That is, it's often necessary to live resource utilization. By instrumenting a system, the tester can uncover situations that cause degradation and possible system failure.

Validation Testing:

At the conclusion of integration testing software is completely assembled as a package, interfacing errors are uncovered and corrected and a final series of software tests begins, validation test begins. Validation test can be defined in many ways. But the straightforward definition is that validation succeeds when the software function during a manner which will reasonably expected by the customer. After validation test has been conducted one among two possible conditions exists.

One is that the function or performance characteristics inform specifications and are accepted and thus the opposite is deviation from specification is uncovered and a deficiency list is formed . Proposed system into

account has been tested by using validation testing and located to be working satisfactorily.

Output Testing:

After performing validation testing, subsequent step is output testing of the proposed system since no system might be useful if it doesn't produce the specified output within the

specified format. Asking the users about the format required by them tests the outputs generated by the system into account. Here the output format is taken into account in two ways, one is on the screen and other is that the printed format. The output format on the screen is found to be correct because the format was designed within the system designed phase consistent with the user needs. For the text also the output comes because the specified requirements by the users. Hence output testing doesn't result any corrections within the system.

User Acceptance Testing:

User acceptance of a system is that the key factor of the success of any system. The system under study is tested for the user acceptance by constantly keeping in-tuned with the potential system users at the time of developing and making changes wherever required.

Test Data:

Taking various sorts of test data does the above testing. Preparation of test data plays an important role within the system testing after preparing the test data the system under study is tested using the test data. While testing the system by using the test data errors are again uncovered and corrected by using above testing steps and corrections also are noted from the longer term use.

Patient Information System

RESULTS / OUTPUTS

4. RESULTS / OUTPUTS

Database View

DB View-1

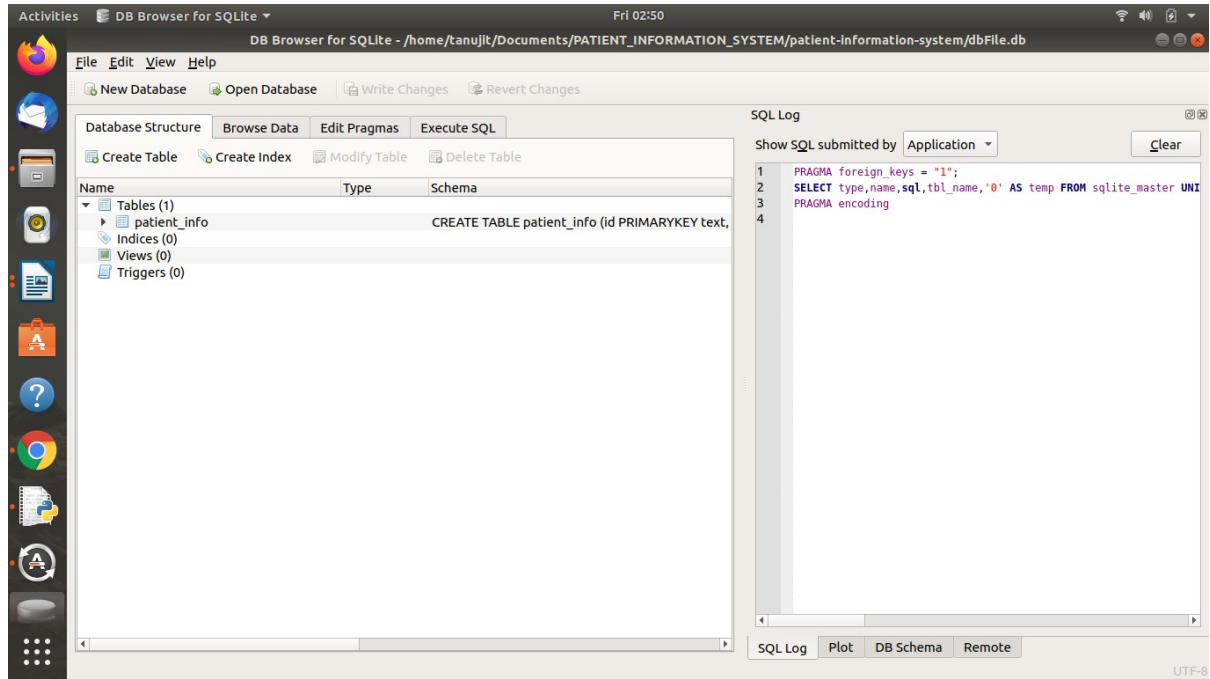


Fig -10

DB View-2

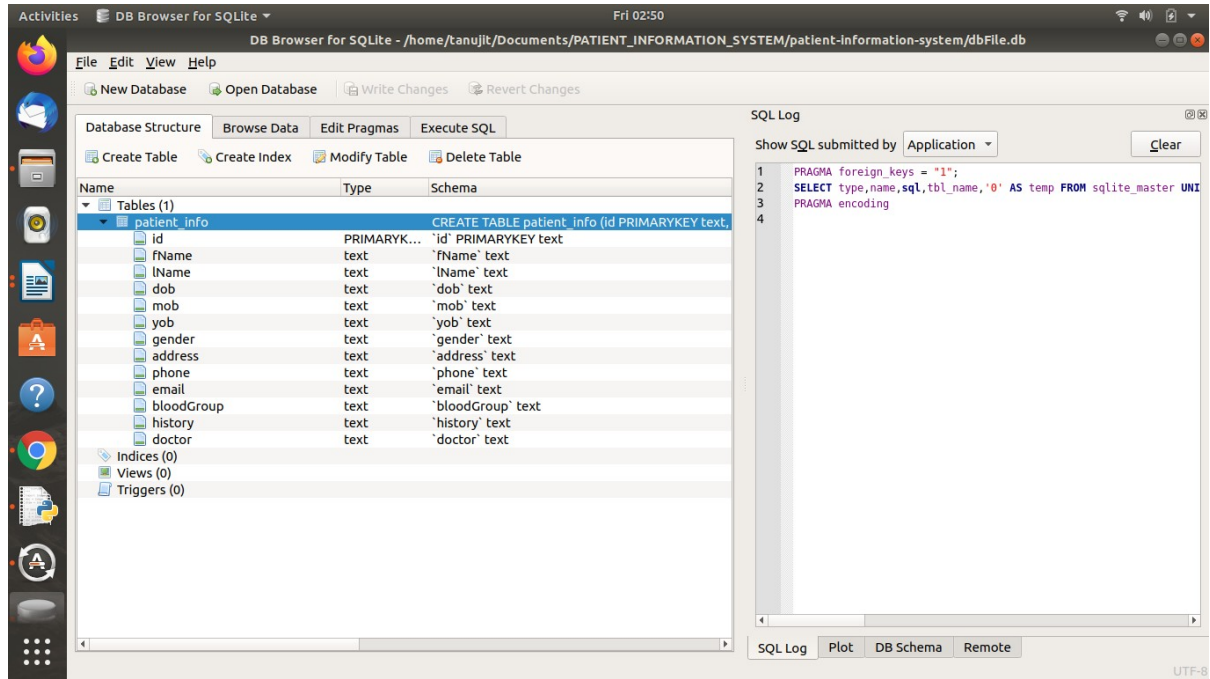


Fig -11

PIS View

Home:

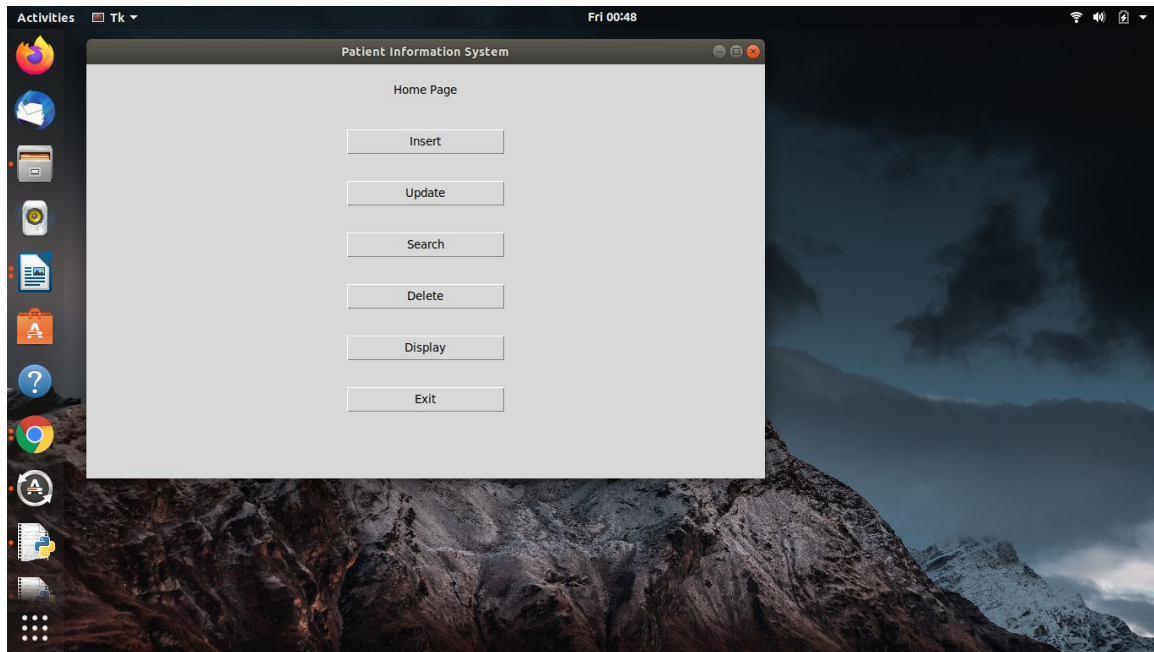
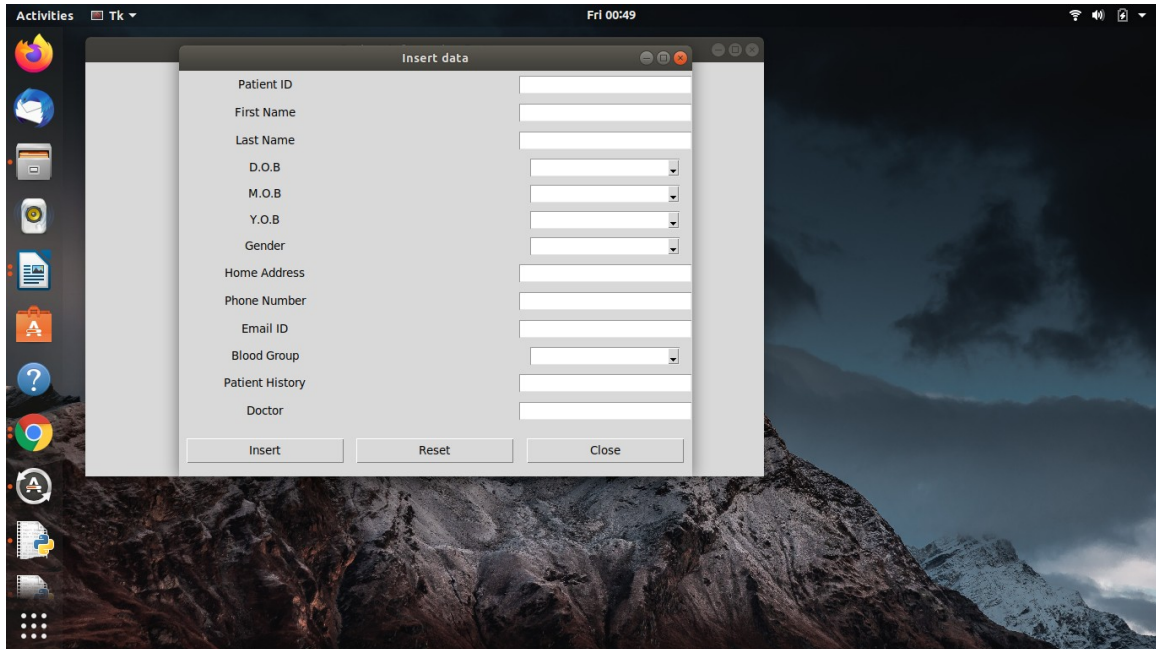


Fig -12

Insert:



The screenshot shows a Linux desktop environment with a dark theme. A window titled "Insert data" is open, displaying a form for entering patient information. The form includes fields for Patient ID, First Name, Last Name, D.O.B, M.O.B, Y.O.B, Gender, Home Address, Phone Number, Email ID, Blood Group, Patient History, and Doctor. At the bottom of the form are three buttons: "Insert", "Reset", and "Close". The desktop background is a dark, rocky landscape. A large, diagonal watermark reading "Patient Information" is visible across the lower half of the image.

Field	Input Type
Patient ID	Text
First Name	Text
Last Name	Text
D.O.B	Text
M.O.B	Text
Y.O.B	Text
Gender	Text
Home Address	Text
Phone Number	Text
Email ID	Text
Blood Group	Text
Patient History	Text
Doctor	Text

Buttons: Insert, Reset, Close

Fig -13

Update:

The image shows a desktop environment with a dark theme and a mountain landscape wallpaper. Two windows titled 'Update data' are open. The top window displays a form for updating patient data for Patient ID 001. The form includes fields for First Name (Tanujit), Last Name (Roy), D.O.B (11), M.O.B (March), Y.O.B (1998), Gender (Male), Home Address (st colony), Phone Number (8018406502), Email ID (tanujit@gmail.com), Blood Group (A+), Patient History (cold), and Doctor (Mishra). At the bottom of this window are 'Update', 'Reset', and 'Close' buttons. The bottom window is a smaller dialog box titled 'Update data' that prompts the user to 'Enter the ID to update' with the value 001 and an 'Update' button.

Update data	
Patient ID	001
First Name	Tanujit
Last Name	Roy
D.O.B	11
M.O.B	March
Y.O.B	1998
Gender	Male
Home Address	st colony
Phone Number	8018406502
Email ID	tanujit@gmail.com
Blood Group	A+
Patient History	cold
Doctor	Mishra
<input type="button" value="Update"/> <input type="button" value="Reset"/> <input type="button" value="Close"/>	

Update data

Enter the ID to update

001

Fig -14

Search:

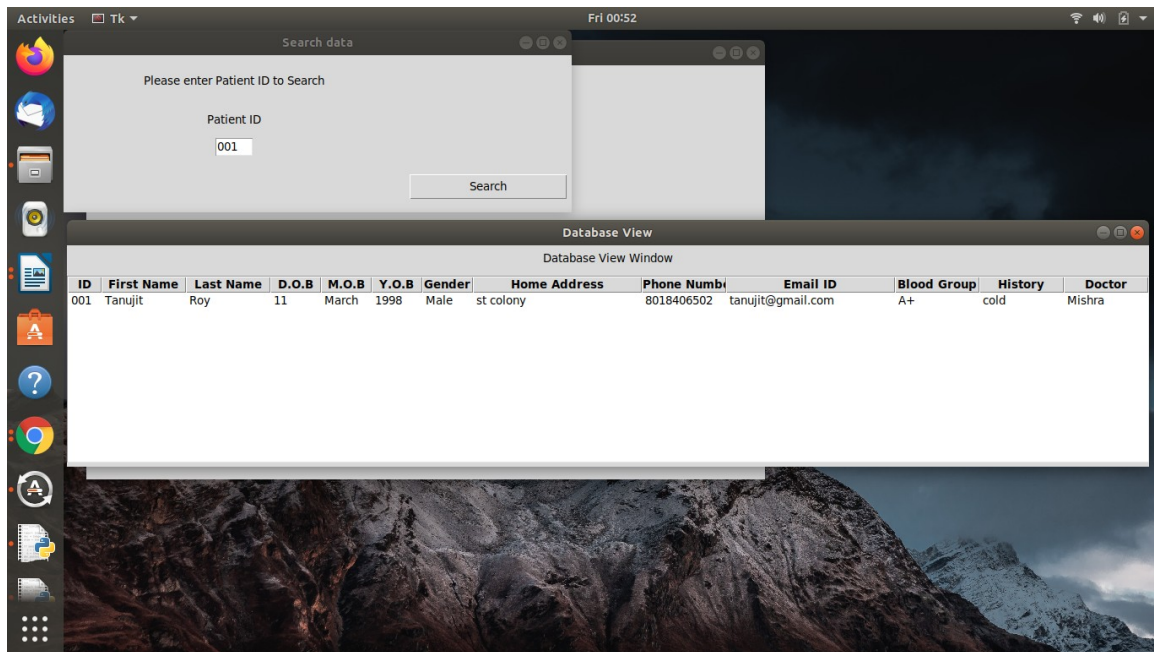


Fig -15

Delete:

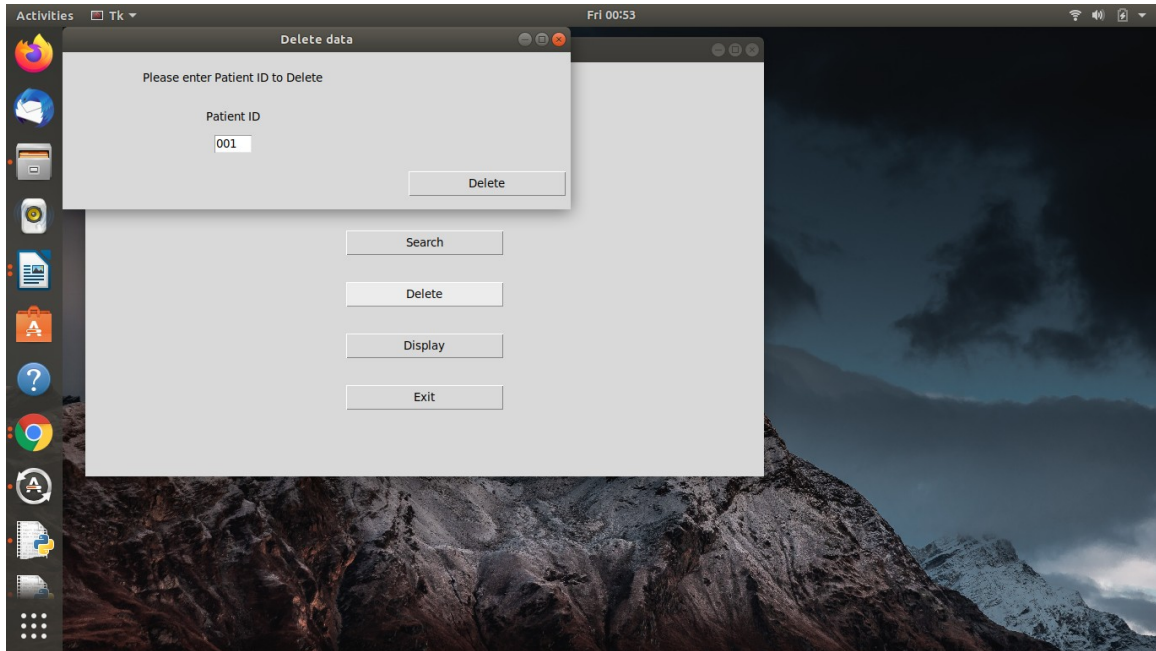


Fig -16

Display:

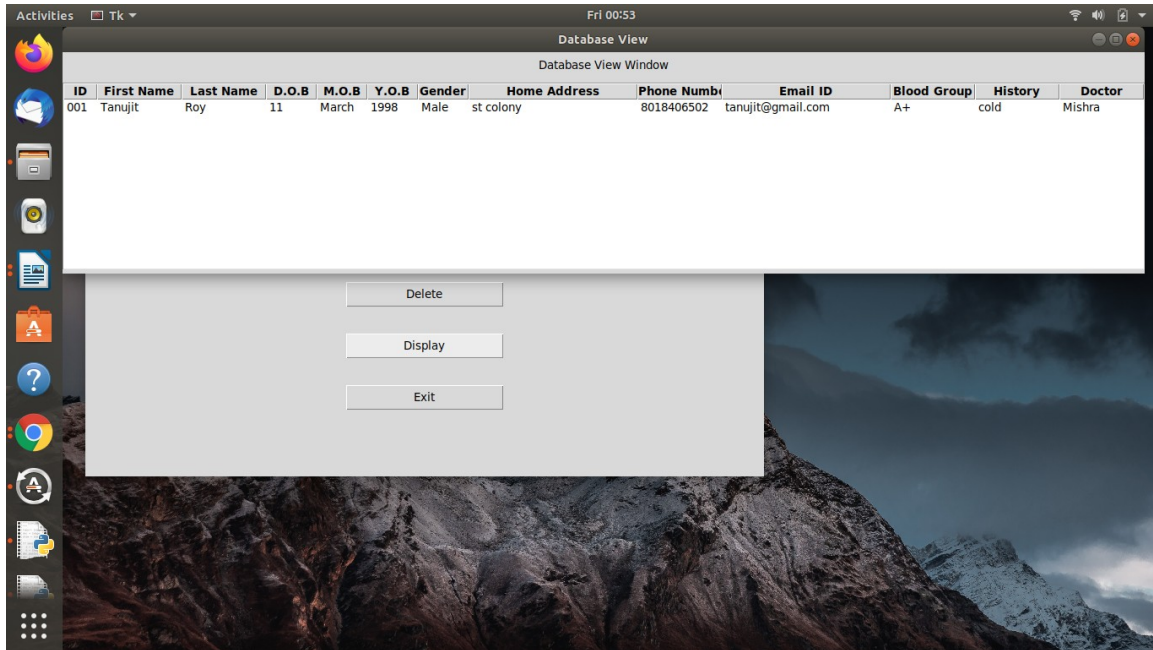


Fig -17

CONCLUSION

5. Conclusion:

It often seen that deploying IT can help the medical community in improving its quality of service and thus automatically increasing the preparedness and defensiveness. It is important that the software must have the proper sort of modularity and openness in order that it's manageable, maintainable and upgradable. The hardware should be reliable, available and have the required performance capacity.

Patient Information System

REFERENCES

6. REFERENCES

- [1] <https://pythonspot.com/introduction/>
- [2] <http://effbot.org/tkinterbook/>
- [3] <https://docs.python.org/3/library/sqlite3.html>
- [4] <https://www.geeksforgeeks.org/types-software-testing/>

Patient Information System

SIMILARITY REPORT

Tanujit Roy - project paper

ORIGINALITY REPORT

9%

SIMILARITY INDEX

1%

INTERNET SOURCES

0%

PUBLICATIONS

9%

STUDENT PAPERS

PRIMARY SOURCES

1

Submitted to The British College

Student Paper

2%

2

Submitted to Kaplan International Colleges

Student Paper

2%

3

Submitted to International School of
Management and Technology (ISMT), Nepal

Student Paper

2%

4

Submitted to Sim University

Student Paper

1%

5

Submitted to Austin High School

Student Paper

1%

6

Submitted to Gujarat Technological University

Student Paper

1%

7

Submitted to University of West London

Student Paper

<1%

8

Submitted to Ibrahim Babangida University

Student Paper

<1%

9

archive.org

Internet Source

<1%

10

Submitted to University of Wales Institute,
Cardiff

Student Paper

<1%

Exclude quotes

On

Exclude matches

< 14 words

Exclude bibliography

On

Patient In