VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590018.

MINI PROJECT REPORT

ON

"Patient Information System"

Submitted in partial fulfillment for the requirement of VI semester for the

Degree of Bachelor of Engineering in

INFORMATION SCIENCE & ENGINEERING

For the academic year 2022-23

SUBMITTED BY:

SURYA.M[1DB20IS148]
SUPREET RAJ[1DB20IS146]

Under the guidance of:

Miss. Deepika A B

Assistant Professor,

Dept. of ISE

Department of Information Science and Engineering

DON BOSCO INSTITUTE OF TECHNOLOGY

Kumbalagodu, Bengaluru-560074

DON BOSCO INSTITUTE OF TECHNOLOGY

Kumbalagodu, Bengaluru-560074

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Mini Project Report entitled **"Patient Information System "** is a bonafide Mini Project work carried out by **Surya M(1DB20IS148)** & **Supreet Raj (1DB20IS146)** in partial fulfillment of VI semester for the Degree of **Bachelor of Engineering inInformation Science and Engineering** of Visvesvaraya Technological University, Belgaum, during the academic year 2022-23.It is certified that all corrections/suggestions indicated for Internal Assessments have been incorporated with the degree mentioned.

Project Guide	Head of Department
Mrs. Deepika A B	**Prof. Dr. B.K. Raghavendra**
Assistant Professor	Head of Department
Dept. of ISE	Dept. of ISE
DBIT, Bengaluru	DBIT, Bengaluru

External Viva

Name of the Examiners	Signature with Date
1. _____	_____
2. _____	_____

ACKNOWLEDGEMENT

At the various stages in making the mini project, a number of people have given me invaluable comment on the manuscript. We take this opportunity to express my deepest gratitude and appreciation to all those who helped me directly or indirectly towards the successful completion of this project.

We would like to thank our Principal **Prof. Nagabhushana B S**, Don Bosco Institute of Technology for his support though out this project.

We express my whole hearted gratitude to **Prof. Dr. B.K. Raghavendra,** who is our respectable Head of Dept. of Information Science. We wish to acknowledge for his valuable help and encouragement.

In this regard We owe a heartfelt gratitude to my guide **Mrs. Deepika A B** Assistant Professor of Department of Information Science and Engineering, for timely advice on the project and regular assistance throughout the project work. We would also like to thank the teaching and non-teachingstaff members of Department of Information Science and Engineering for their corporation.

SURYA M (1DB20IS148)
SUPREET RAJ(1DB20IS146)

ABSTRACT

A “Patient Information System” is a comprehensive digital platform designed to efficiently manage and organize patient-related data within a healthcare facility. It includes file operations such as storing,retrieving, searching, deleting and updating patient information, ensuring that healthcare providers have access to accurate and up-to-date data for effective decision-making and care coordination. A filestructure is a combination of representations for data in files. It is also a collection of operations for accessing the data. It enables applications to read, write, and modify data. File structures may also helpto find the data that matches certain criteria. Indexing involves creating a separate data structure that contains pointers to the location of the data in the file structure. This allows the system to quickly locateand retrieve data without having to search through the entire file structure. Hashing is a technique thatcan be used in patient management software to improve the efficiency of data storage and retrieval. Hashing involves taking a value, such as a patient unique id, and using it to generate a shorter fixed length value called a hash. The system typically includes functionalities such as patient full name, dateof birth, gender, address and phone number, patient history, blood group and clinical decision support.By streamlining administrative tasks and improving data accessibility, a patient information system enhances the overall quality of patient care, enhances communication among healthcare professionals, and increases operational efficiency within healthcare organizations. Overall, a patient information system aims to streamline and enhance the management of patient information, ultimately improving patient care, safety, and outcomes. By centralizing data, ensuring accessibility, and supporting clinicaldecision-making, the system plays a vital role in modern healthcare delivery.

CONTENTS

CHAPTER 1

INTRODUCTION

A Patient information system is an element of health informatics that focuses mainly on the administrational needsof hospitals. In many implementations, a PIS is a comprehensive, integrated information system designed to manage all the aspects of a hospital's operation, such as medical, administrative, financial , and legal issues and the corresponding processing of services. Hospital information system is also known as patient management software (PMS) or Patient management system.

Patient information systems provide a common source of information about a patient's health history, and doctorsschedule timing. The system has to keep data in a secure place and controls who can reach the datain certain circumstances. These systems enhance the ability of health care professionals to coordinate careby providing a patient's health information and visit history at the place and time that it is needed. Patient's laboratory test information also includes visual results such as X-ray, which may be reachable by professionals. PIS provide internaland external communication among health care providers.

The frontend is responsible for providing a user-friendly interface for users to interact with the system. The system's front-end comprises HTML, CSS, and JavaScript, which are responsible for rendering the graphical user interface(GUI) and handling user inputs. HTML is used to structure and organize the content of web pages, while CSS is responsible for styling the pages and creating visually appealing layouts. JavaScript, on the other hand, enables dynamic interactions between the user and the system, allowing for real-time updates and instant feedback.

The back-end of the Patient information System is built using Python, which provides robust functionalityand flexibility for processing and storing data. The system employs a file-based database approach, wherePatient related information is stored. The use of file-based databases allows for easy data manipulation and quick access to patient details. In summary, the patient information System is a software solution thatutilizes a combination offront-end and back-end technologies such as HTML, CSS, JavaScript, and Python, and employs indexing and hashing techniques for optimized data retrieval and storage.

1.1 Overview of the file structure

Hospitals currently use a manual system for the management and maintenance of critical information.The current system requires numerous paper forms, with data stores spread throughout the Patient management infrastructure. Often information (on forms) is incomplete, or does not follow management standards. Forms are often lost in transit between departments requiring a comprehensiveauditing process to ensure that no vital information is lost.

1.2 Problem statement

The Existing system includes problems like lack of time consuming, accuracy, high cost, securityproblems etc. The following problem statement highlights these issues:

- Inefficiency and Manual Processes.
- Lack of Transparency and Auditability.
- Security and Data Integrity.
- Limited Scalability and Flexibility.
- Complex Reconciliation and Reporting.
- Data Analysis and Business Insights.

.

1.3 Aim and Objectives

Aim:

The Aim of Patient Information System is to efficiently record, manage, and process financial transactions, ensuring accuracy, security, transparency, and compliance within organizations.

Objective:

- Automation and Efficiency.
- Accuracy and Auditability.
- Security and Data Protection.
- Scalability and Adaptability.
- Integration and Interoperability.
- Reporting and Analytics.

1.4 Existing system

Hospitals currently use a manual system for the management and maintenance of critical information. The current system requires numerous paper forms, with data stores spread throughout the Patient management infrastructure. Often information (on forms) is incomplete, or does not follow management standards. Formsare often lost in transit between departments requiring a comprehensive auditing process to ensure that no vital information is lost. Multiple copies of the same information exist in the hospital and may lead to inconsistencies in data in various data stores. A significant part of the operation of any hospital involves theacquisition, management and timely retrieval of great volumes of information. This information typically involves; patient personal information and medical history, doctor information, scheduling, and various facilities waiting lists. All of this information must be managed in an efficient and cost wise fashion so that an institution's resources may be effectively utilized PMS will automate the management of the hospital making it more efficient and error free. It aims at standardizing data, consolidating data ensuring data integrity and reducing inconsistencies.

1.4.1 Disadvantages

- Maintaining and managing data is very costly and time consuming, because there are many documents that have to be maintained by each branch and copies have to be transferred to relative branches. Transfer of information within the branches is costly and time consuming.
- One of the greatest disadvantages of the hospital management system is generally related to security. It is considered to be a matter of concern in case you go online without enough protection which can create some big problems related to security.
- Another greatest problem that is associated with the health care industry is the data breach. Not only that, but it is also known to be a sophisticated problem.
- Lack of employment is another such thing.
- The chances of employment usually become less with the automation of the system. Also, the gradual need for manual data drafting becomes an irrelevant aspect.

CHAPTER 2

PROPOSED SYSTEM

There are many activities that can be computerized in this widely spread organization. The proposed systemfor managing patient information is a digitalized system that leverages modern technology Python,and indexing hashing file structure. This system aims to automate many of the processes that are currentlydonemanually in the existing system, reducing the potential for errors and making the system more efficient. Inthe proposed system, healthcare faculty can access the patient information through a website.They can search for patient with unique id, view patient details and update, and make appointments and delete patientrecord online. The system stores all the necessary data in an indexed and hashed file structure,making it easyto retrieve and manage large amounts of data quickly.

2.1 ADVANTAGES

- Enhanced Efficiency: By digitizing and automating various tasks, patient information systems streamline administrative processes, reduce paperwork, and enable faster access to patient information, leading to improved efficiency and productivity for healthcare providers.

- Improved Patient Care: Having a comprehensive and up-to-date electronic health record allows healthcare professionals to make better-informed decisions, coordinate care effectively, and provide more personalized and timely treatments to patients.

- Increased Accuracy and Safety: Patient information systems minimize the risks of errors associated with manual data entry and paper-based records. They provide alerts for drug interactions, allergies, and other critical information, promoting patient safety and reducing medical errors.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1Software (Front end & Back end) requirements

- Processor: Intel dual Core, i3
- RAM: 1 GB
- Hard Disk: 80 GB

3.2HARDWARE REQUIREMENTS

- Programming Language: Python
- IDE Used: Visual Studio Codes

CHAPTER 4

IMPLEMENTATION

4.1 File creation

INDEXES:

All indexes are based on same basic concepts-keys and reference fields. The types of indexeswe see are called simple indexes because they are represented using simple arrays of structures that contain the keys and reference fields.

A Simple Index for Entry-Sequenced Files

Simple index: An index in which the entries are a key ordered linear list.

- Simple indexing can be useful when the entire index can be held in memory.
- Changes (additions and deletions) require both the index and the data file to bechanged.
- Updates affect the index if the key field is changed, or if the record is moved.
- An update which moves a record can be handled as a deletion followed by an addition.

Object Oriented Support for Indexed, Entry Sequenced Files

Entry-sequenced file: A file in which the record order is determined by the order in which theyare entered. The index should be read into memory when the data file is opened.

- Indexes That are too Large to Hold in Memory
- Searching of a simple index on disk takes too much time.
- Maintaining a simple index on disk in sorted order takes too much time.
- Tree structured indexes such as B-trees are a scalable alternative to simple indexes.
- Hashed organization is an alternative to indexing when only a primary index is needed.

4.1 Description of methods

File Handling:

Many real-life problems handle large volume of data and in such situations, we need to use some devices such as Floppy disk or hard disk to store the data. The data is stored in these devices usingthe concept of files. A file is a collection of related data stored in a particular area on the disk. Programs can be designed to perform the read and write operations on these files.

Class for file stream operation:

The I/O system of CPP contain a set of classes that define the file handling methods. This includes ifstream ofstream and fstream. These classes are derived from fstream base and the corresponding iostream class, the classes, designed manage the disk, are declared in fstream andtherefore we must include this file in any program that uses file.

Mode for opening the file:

1) ios::app ->To append at the end of file

2)ios::out->its open file in write only mode.

3) ios::in->its open file in read only mode.

Read file syntax:

Stream —object.read((char*)&class-object, sizeof(class-object));

Files syntax:

Stream -object.write((char*)&class-object,sizeof(class -object));

Of.close():- It closes the stream. All buffers associated with the stream are flushed before closing. System allocates buffers are freed upon closing. Buffers are assigned with set buf are not automatically freed.

4.2 IMPLEMENTATION & METHODLOGY

Implementation Methodology:

Model View Controller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

- **Model** - The lowest level of the pattern which is responsible for maintaining data.
- **View** - This is responsible for displaying all or a portion of the data to the user.
- **Controller** - Software Code that controls the interactions between the Model and View.

MVC is popular as it isolates the application logic from the user interface layer and supports separationof concerns. Here the Controller receives all requests for the application and then works with the Modelto prepare any data needed by the View. The View then uses the data prepared by the Controller to generatea final presentable response. The MVC abstraction can be graphically represented as follows.

CHAPTER 5

CODE

```
import tkinter
import tkinter.ttk
import tkinter.messagebox
import sqlite3


class Database:
    def__init_(self):
        self.dbConnection = sqlite3.connect("dbFile.db")
        self.dbCursor = self.dbConnection.cursor()
        self.dbCursor.execute("CREATE TABLE IF NOT EXISTS patient_info (id PRIMARYKEY text, fName text, lName text, dob text, mob text, yob text, gender text, addresstext, phone text, email text, bloodGroup text, history text, doctor text)")


    def__del_(self):
        self.dbCursor.close()
        self.dbConnection.close()


    def Insert(self, id, fName, lName, dob, mob, yob, gender, address, phone, email, bloodGroup, history, doctor):
        self.dbCursor.execute("INSERT INTO patient_info VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)", (id, fName, lName, dob, mob, yob, gender, address, phone, email, bloodGroup, history, doctor))
        self.dbConnection.commit()


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

```
            self.dbCursor.execute("UPDATE patient_info SET fName = ?, lName = ?, dob = ?, mob
        = ?, yob = ?, gender = ?, address = ?, phone = ?, email = ?, bloodGroup = ?, history = ?, doctor
 = ?  WHERE id = ?", (fName, lName, dob, mob, yob, gender, address, phone, email

bloodGroup, history, doctor, id))
self.dbConnection.commit()

def Search(self, id):
        self.dbCursor.execute("SELECT * FROM patient_info WHERE id = ?", (id, ))searchResults =
        self.dbCursor.fetchall()
        return searchResults

    def Delete(self, id):
        self.dbCursor.execute("DELETE FROM patient_info WHERE id = ?", (id, ))
        self.dbConnection.commit()

    def Display(self):
        self.dbCursor.execute("SELECT * FROM patient_info")
        records = self.dbCursor.fetchall()
        return records

 class Values:
    def Validate(self, id, fName, lName, phone, email, history, doctor):if
        not (id.isdigit() and (len(id) == 3)):
            return "id"
        elif not (fName.isalpha()):
            return "fName"
        elif not (lName.isalpha()):
            return "lName"
```

```
        elif not (phone.isdigit() and (len(phone) == 10)):
            return "phone"
        elif not (email.count("@") == 1 and email.count(".") > 0):
            return "email"
        elif not (history.isalpha()):
            return "history"
        elif not (doctor.isalpha()):
            return "doctor"

        else:
            return "SUCCESS"

class InsertWindow:
    def__init_(self):
        self.window = tkinter.Tk()
        self.window.wm_title("Insert data")

        # Initializing all the variables
        self.id = tkinter.StringVar()
        self.fName = tkinter.StringVar()
        self.lName = tkinter.StringVar()
        self.address = tkinter.StringVar()
        self.phone = tkinter.StringVar()
        self.email = tkinter.StringVar()
        self.history = tkinter.StringVar()
        self.doctor = tkinter.StringVar()
        self.genderList = ["Male", "Female", "Transgender", "Other"]
        self.dateList = list(range(1, 32))
```

```
        self.monthList = ["January", "February", "March", "April", "May", "June", "July","August",
"September", "October", "November", "December"]
        self.yearList = list(range(1900, 2020))
        self.bloodGroupList = ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]


        # Labels
        tkinter.Label(self.window, text = "Patient ID", width = 25).grid(pady = 5, column = 1,row =
1)
        tkinter.Label(self.window, text = "First Name", width = 25).grid(pady = 5, column = 1,row =
2)


        tkinter.Label(self.window, text = "Last Name", width = 25).grid(pady = 5, column = 1,row =
3)
        tkinter.Label(self.window, text = "D.O.B",  width = 25).grid(pady = 5, column = 1, row =
4)
        tkinter.Label(self.window, text = "M.O.B", width = 25).grid(pady = 5, column = 1, row
=5)
        tkinter.Label(self.window, text = "Y.O.B", width = 25).grid(pady = 5, column = 1, row =
6)
        tkinter.Label(self.window, text = "Gender", width = 25).grid(pady = 5, column = 1, row
=7)
        tkinter.Label(self.window, text = "Home Address",  width = 25).grid(pady = 5, column =
```

```
1, row = 8)

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

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

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

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

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


    # Fields

    # Entry widgets

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

    self.fNameEntry = tkinter.Entry(self.window, width = 25, textvariable = self.fName)

    self.lNameEntry = tkinter.Entry(self.window, width = 25, textvariable = self.lName)

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

    self.phoneEntry = tkinter.Entry(self.window, width = 25, textvariable = self.phone)

    self.emailEntry = tkinter.Entry(self.window, width = 25, textvariable = self.email)

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

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

```
tkinter.Label(self.window, text = "Last Name", width = 25).grid(pady = 5, column = 1,row = 3)
        tkinter.Label(self.window, text = "D.O.B",  width = 25).grid(pady = 5, column = 1, row =
4)
        tkinter.Label(self.window, text = "M.O.B", width = 25).grid(pady = 5, column = 1, row
=5)
        tkinter.Label(self.window, text = "Y.O.B", width = 25).grid(pady = 5, column = 1, row =
6)
        tkinter.Label(self.window, text = "Gender", width = 25).grid(pady = 5, column = 1, row
=7)
        tkinter.Label(self.window, text = "Home Address",  width = 25).grid(pady = 5, column =
        1, row = 8)
            tkinter.Label(self.window, text = "Phone Number", width = 25).grid(pady = 5, column
        =1, row = 9)
            tkinter.Label(self.window, text = "Email ID",  width = 25).grid(pady = 5, column = 1, row
        = 10)
            tkinter.Label(self.window, text = "Blood Group",  width = 25).grid(pady = 5, column = 1,
        row = 11)
            tkinter.Label(self.window, text = "Patient History", width = 25).grid(pady = 5, column
        =1, row = 12)
            tkinter.Label(self.window, text = "Doctor", width = 25).grid(pady = 5, column = 1, row
        = 13)


            # Fields
            # Entry widgets
            self.idEntry = tkinter.Entry(self.window, width = 25, textvariable = self.id)
            self.fNameEntry = tkinter.Entry(self.window, width = 25, textvariable =
            self.fName) self.lNameEntry = tkinter.Entry(self.window, width = 25, textvariable =
            self.lName) self.addressEntry = tkinter.Entry(self.window, width = 25, textvariable
            = self.address)self.phoneEntry = tkinter.Entry(self.window, width = 25, textvariable
            = self.phone) self.emailEntry = tkinter.Entry(self.window, width = 25, textvariable
            = self.email) self.historyEntry = tkinter.Entry(self.window, width = 25, textvariable
            = self.history) self.doctorEntry = tkinter.Entry(self.window,  width = 25, textvariable
```

```
1, row = 8)

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

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

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

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

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


    # Fields

    # Entry widgets

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

    self.fNameEntry = tkinter.Entry(self.window, width = 25, textvariable = self.fName)

    self.lNameEntry = tkinter.Entry(self.window, width = 25, textvariable = self.lName)

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

    self.phoneEntry = tkinter.Entry(self.window, width = 25, textvariable = self.phone)

    self.emailEntry = tkinter.Entry(self.window, width = 25, textvariable = self.email)

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

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

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

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

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

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

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

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

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

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

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

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

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


        # Fields

        # Entry widgets

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

        self.fNameEntry = tkinter.Entry(self.window, width = 25, textvariable = self.fName)

        self.lNameEntry = tkinter.Entry(self.window, width = 25, textvariable = self.lName)

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

        self.phoneEntry = tkinter.Entry(self.window, width = 25, textvariable = self.phone)

        self.emailEntry = tkinter.Entry(self.window, width = 25, textvariable = self.email)

        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.fNameEntry.grid(pady = 5, column = 3, row = 2)
        self.lNameEntry.grid(pady = 5, column = 3, row = 3)
        self.addressEntry.grid(pady = 5, column = 3, row = 8)
        self.phoneEntry.grid(pady = 5, column = 3, row = 9)
        self.emailEntry.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.dobBox = tkinter.ttk.Combobox(self.window, values = self.dateList, width = 20)
        self.mobBox = tkinter.ttk.Combobox(self.window, values = self.monthList, width = 20)
        self.yobBox = tkinter.ttk.Combobox(self.window, values = self.yearList, width = 20)
        self.genderBox = tkinter.ttk.Combobox(self.window, values = self.genderList, width =
20)
        self.bloodGroupBox = tkinter.ttk.Combobox(self.window, values = self.bloodGroupList,
width = 20)

        self.dobBox.grid(pady = 5, column = 3, row = 4)
        self.mobBox.grid(pady = 5, column = 3, row = 5)
        self.yobBox.grid(pady = 5, column = 3, row = 6)
        self.genderBox.grid(pady = 5, column = 3, row = 7)
        self.bloodGroupBox.grid(pady = 5, column = 3, row = 11)

        # Button widgets
        tkinter.Button(self.window, width = 20, text = "Insert", command = self.Insert).grid(pady
= 15, padx = 5, column = 1, row = 14)
        tkinter.Button(self.window, width = 20, text = "Reset", command = self.Reset).grid(pady
= 15, padx = 5, column = 2, row = 14)
```

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


        self.window.mainloop()


    def Insert(self):
        self.values = Values()
        self.database = Database()
        self.test = self.values.Validate(self.idEntry.get(), self.fNameEntry.get(), self.lNameEntry.get(), self.phoneEntry.get(), self.emailEntry.get(), self.historyEntry.get(), self.doctorEntry.get())
        if (self.test == "SUCCESS"):
            self.database.Insert(self.idEntry.get(), self.fNameEntry.get(), self.lNameEntry.get(), self.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())
            tkinter.messagebox.showinfo("Inserted data", "Successfully inserted the above data inthe database")
        else:
            self.valueErrorMessage = "Invalid input in field " + self.test
            tkinter.messagebox.showerror("Value Error", self.valueErrorMessage)


    def Reset(self):
        self.idEntry.delete(0, tkinter.END)
        self.fNameEntry.delete(0, tkinter.END)
        self.lNameEntry.delete(0, tkinter.END)
        self.dobBox.set("")
        self.mobBox.set("")
        self.yobBox.set("")
        self.genderBox.set("")
        self.addressEntry.delete(0, tkinter.END)
        self.phoneEntry.delete(0, tkinter.END)
```

```
        self.emailEntry.delete(0, tkinter.END)
        self.bloodGroupBox.set("")
        self.historyEntry.delete(0, tkinter.END)
        self.doctorEntry.delete(0, tkinter.END)

class UpdateWindow:
    def__init_(self, id):
        self.window = tkinter.Tk()
        self.window.wm_title("Update data")

        # Initializing all the variables
        self.id = id

        self.fName = tkinter.StringVar()
        self.lName = tkinter.StringVar()
        self.address = tkinter.StringVar()
        self.phone = tkinter.StringVar()
        self.email = tkinter.StringVar()
        self.history = tkinter.StringVar()
        self.doctor = tkinter.StringVar()

        self.genderList = ["Male", "Female", "Transgender", "Other"]
        self.dateList = list(range(1, 32))
        self.monthList = ["January", "February", "March", "April", "May", "June", "July","August",
"September", "October", "November", "December"]
        self.yearList = list(range(1900, 2020))
        self.bloodGroupList = ["A+", "A-", "B+", "B-", "O+", "O-", "AB+", "AB-"]

        # Labels
```

```
        tkinter.Label(self.window, text = "Patient ID", width = 25).grid(pady = 5, column = 1,row =
1)

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

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

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

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

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

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

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

        tkinter.Label(self.window, text = "Home Address", width = 25).grid(pady = 5, column =

1, row = 8)

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

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

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

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

        tkinter.Label(self.window, text = "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)


        # Fields

        # Entry widgets

        self.fNameEntry = tkinter.Entry(self.window, width = 25, textvariable = self.fName)

        self.lNameEntry = tkinter.Entry(self.window, width = 25, textvariable = self.lName)

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

        self.phoneEntry = tkinter.Entry(self.window, width = 25, textvariable = self.phone)

        self.emailEntry = tkinter.Entry(self.window, width = 25, textvariable = self.email)

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

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

```
        self.fNameEntry.grid(pady = 5, column = 3, row = 2)
        self.lNameEntry.grid(pady = 5, column = 3, row = 3)
        self.addressEntry.grid(pady = 5, column = 3, row = 8)
        self.phoneEntry.grid(pady = 5, column = 3, row = 9)
        self.emailEntry.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.dobBox = tkinter.ttk.Combobox(self.window, values = self.dateList, width = 20)
        self.mobBox = tkinter.ttk.Combobox(self.window, values = self.monthList, width = 20)
        self.yobBox = tkinter.ttk.Combobox(self.window, values = self.yearList, width = 20)
        self.genderBox = tkinter.ttk.Combobox(self.window, values = self.genderList, width =
20)
        self.bloodGroupBox = tkinter.ttk.Combobox(self.window, values = self.bloodGroupList,
width = 20)

        self.dobBox.grid(pady = 5, column = 3, row = 4)
        self.mobBox.grid(pady = 5, column = 3, row = 5)
        self.yobBox.grid(pady = 5, column = 3, row = 6)
        self.genderBox.grid(pady = 5, column = 3, row = 7)
        self.bloodGroupBox.grid(pady = 5, column = 3, row = 11)

        # Button widgets
        tkinter.Button(self.window, width = 20, text = "Update", command =
self.Update).grid(pady = 15, padx = 5, column = 1, row = 14)
        tkinter.Button(self.window, width = 20, text = "Reset", command = self.Reset).grid(pady
= 15, padx = 5, column = 2, row = 14)
        tkinter.Button(self.window, width = 20, text = "Close", command =
self.window.destroy).grid(pady = 15, padx = 5, column = 3, row = 14)
```

```
        self.window.mainloop()


    def                 Update(self):

        self.database = Database()

        self.database.Update(self.fNameEntry.get(), self.lNameEntry.get(), self.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(), self.id)

        tkinter.messagebox.showinfo("Updated data", "Successfully updated the above data in the
database")


    def Reset(self):

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

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

        self.dobBox.set("")

        self.mobBox.set("")

        self.yobBox.set("")

        self.genderBox.set("")

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

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

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

        self.bloodGroupBox.set("")

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

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


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", "fName", "lName", "dob", "mob", "yob", "gender", "address", "phone", "email", "bloodGroup", "history", "doctor")

# Treeview column headings

self.databaseView.heading("id", text = "ID")

self.databaseView.heading("fName", text = "First Name")

self.databaseView.heading("lName", text = "Last Name")

self.databaseView.heading("dob", text = "D.O.B")

self.databaseView.heading("mob", text = "M.O.B")

self.databaseView.heading("yob", text = "Y.O.B")

self.databaseView.heading("gender", text = "Gender")

self.databaseView.heading("address", text = "Home Address")

self.databaseView.heading("phone", text = "Phone Number")

self.databaseView.heading("email", text = "Email ID")

self.databaseView.heading("bloodGroup", text = "Blood Group")

self.databaseView.heading("history", text = "History")

self.databaseView.heading("doctor", text = "Doctor")

# Treeview columns self.databaseView.column("id",

width = 40) self.databaseView.column("fName",

width = 100)self.databaseView.column("lName",

width = 100)self.databaseView.column("dob", width

= 60) self.databaseView.column("mob", width = 60)
```

```
        self.databaseView.column("yob", width = 60)
        self.databaseView.column("gender", width = 60)
        self.databaseView.column("address", width = 200)
        self.databaseView.column("phone", width = 100)
        self.databaseView.column("email", width = 200)
        self.databaseView.column("bloodGroup", 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()

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

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


class HomePage:
    def__init_(self):
        self.homePageWindow = tkinter.Tk()
        self.homePageWindow.wm_title("Patient Information System")
tkinter.Label(self.homePageWindow, text = "Home Page", width = 100).grid(pady = 20,column = 1,
row = 1)
  tkinter.Button(self.homePageWindow, width = 20, text = "Insert", command =
self.Insert).grid(pady = 15, column = 1, row = 2)
```

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

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

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

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

        tkinter.Button(self.homePageWindow, width = 20, text = "Exit", 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()
```

CHAPTER 6

SNAPSHOTS

6.1 Login Page

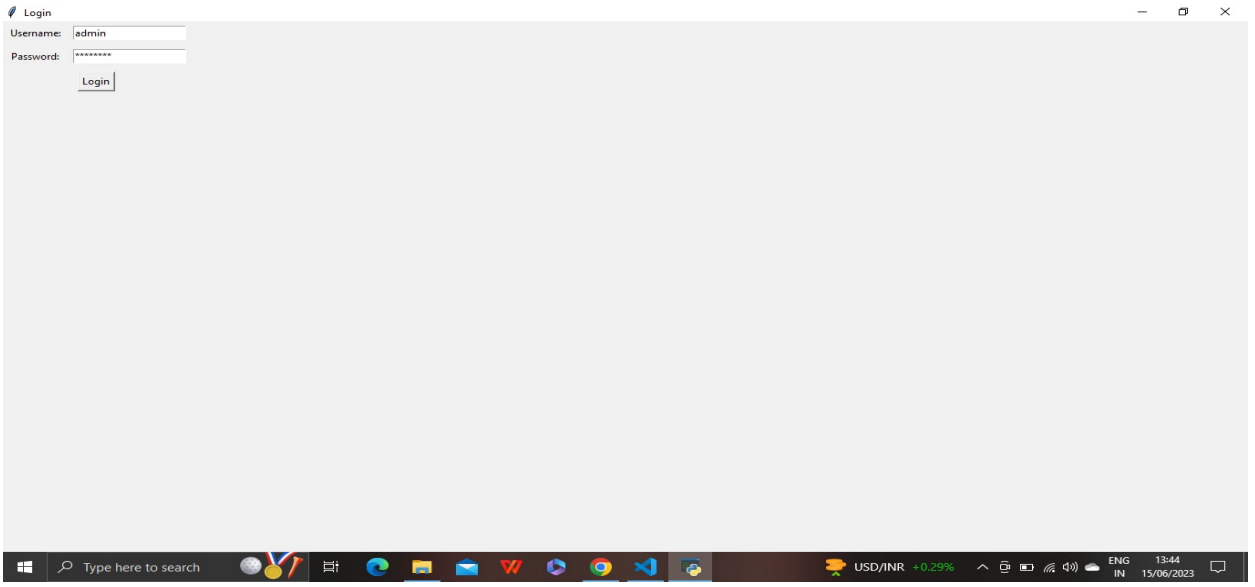

Figure 6.1: Login Page

6.2 DASHBOARD PAGE

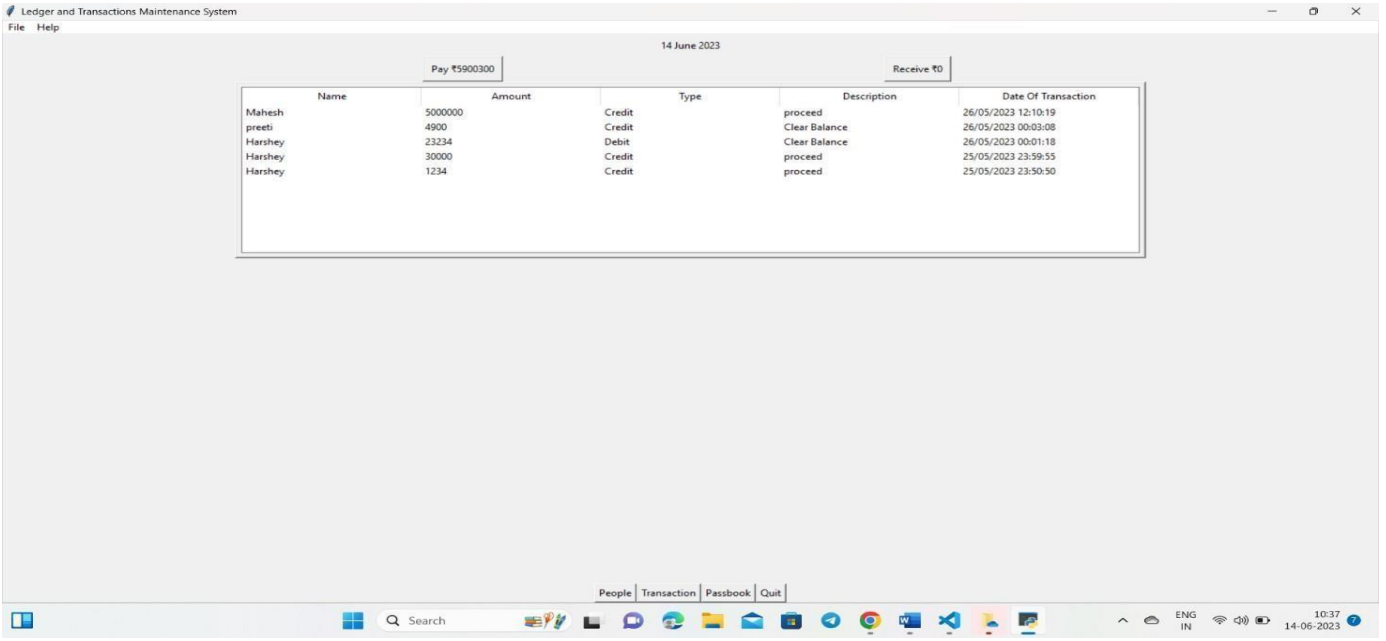

Figure 6.2: Dashboard Page

.

6.3 PEOPLE PAGE

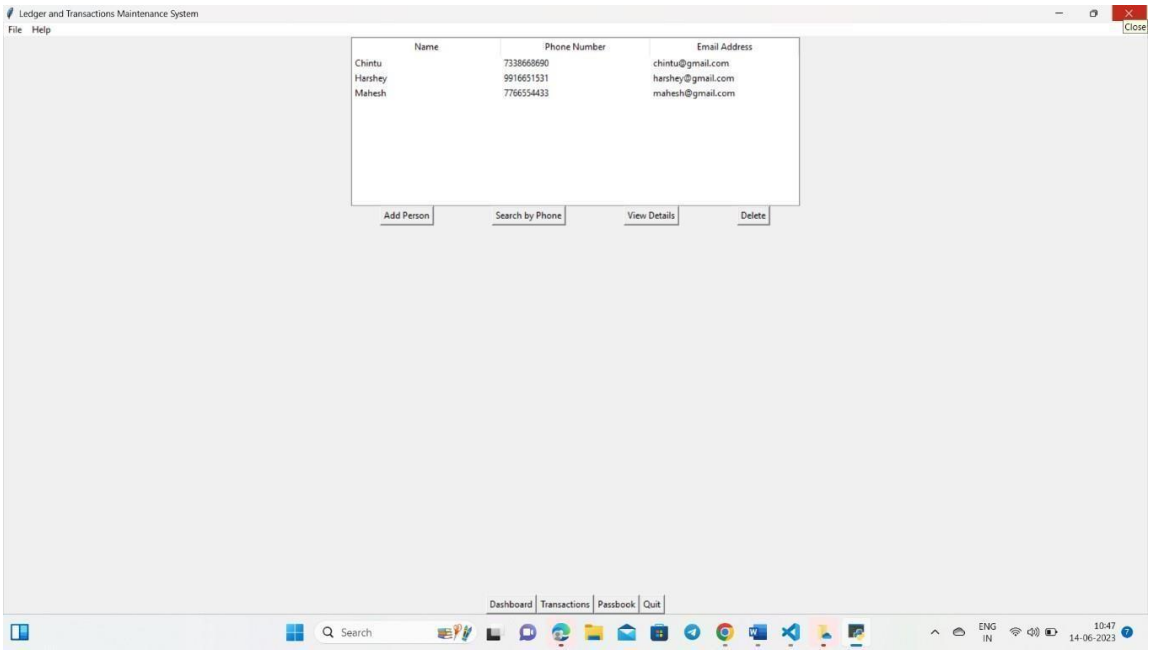

Figure 6.3: People Page

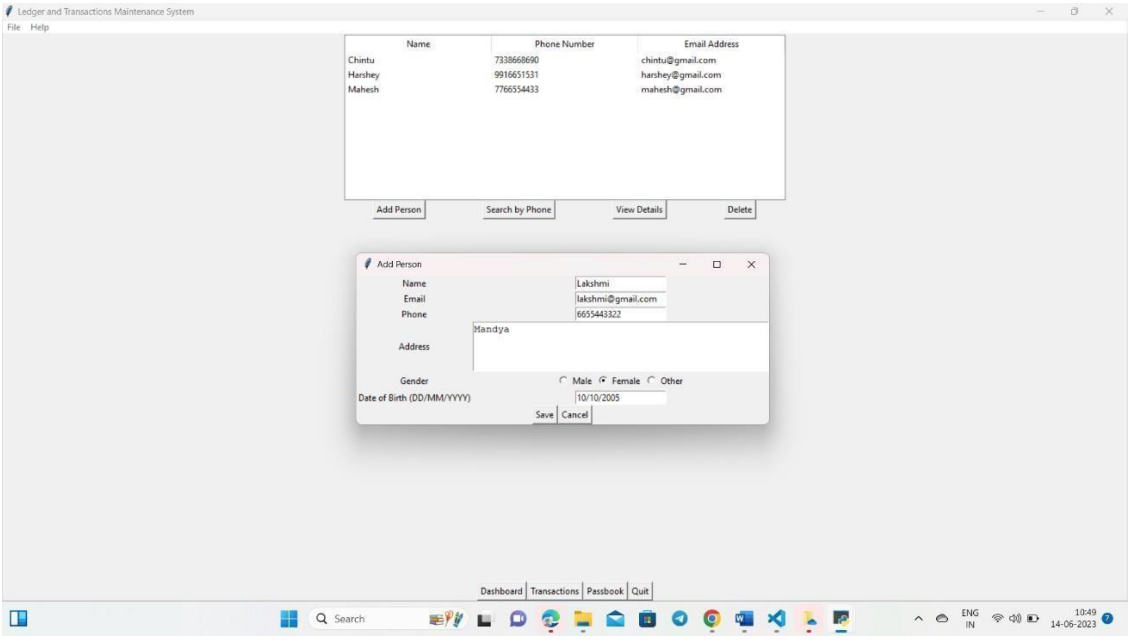

Figure 6.3.1 Add People Page

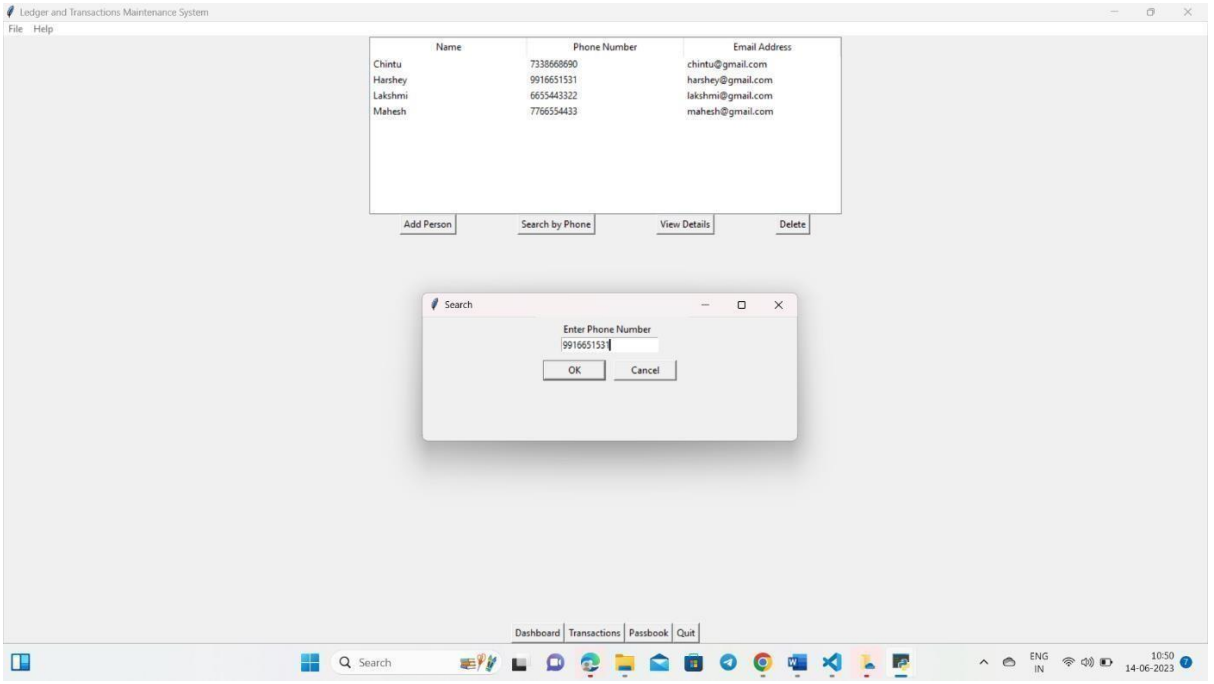

Figure 6.3.2 Search By Phone Page

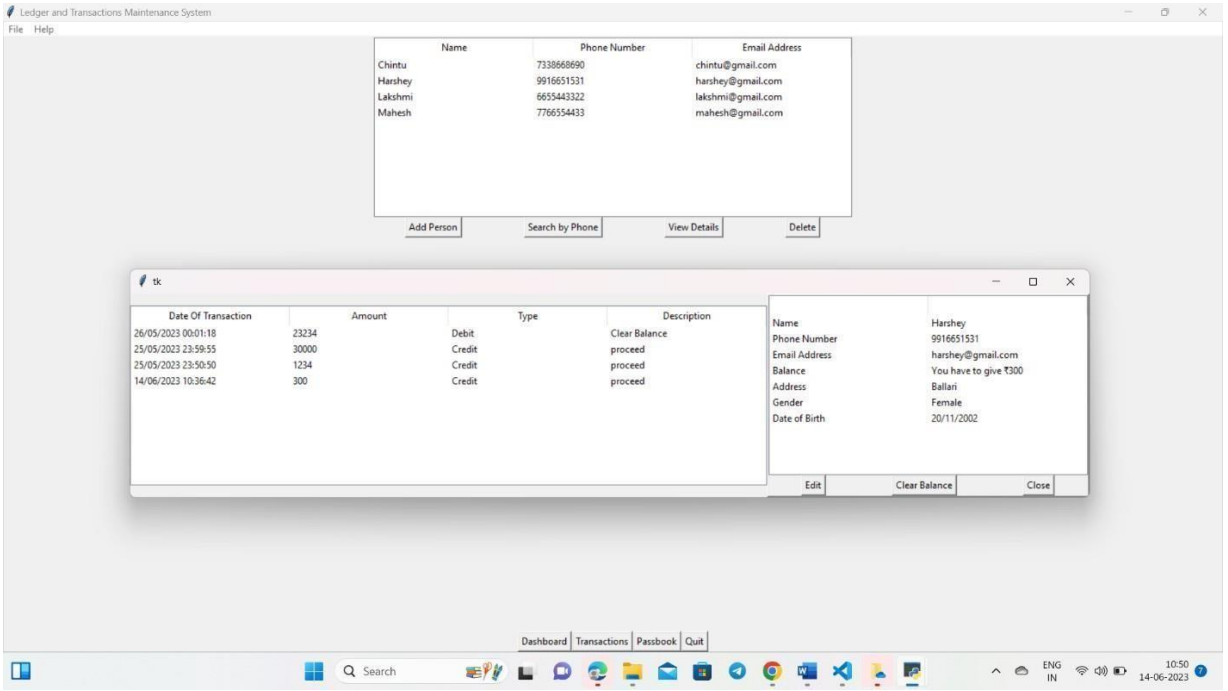

Figure 6.3.4 View Details Page

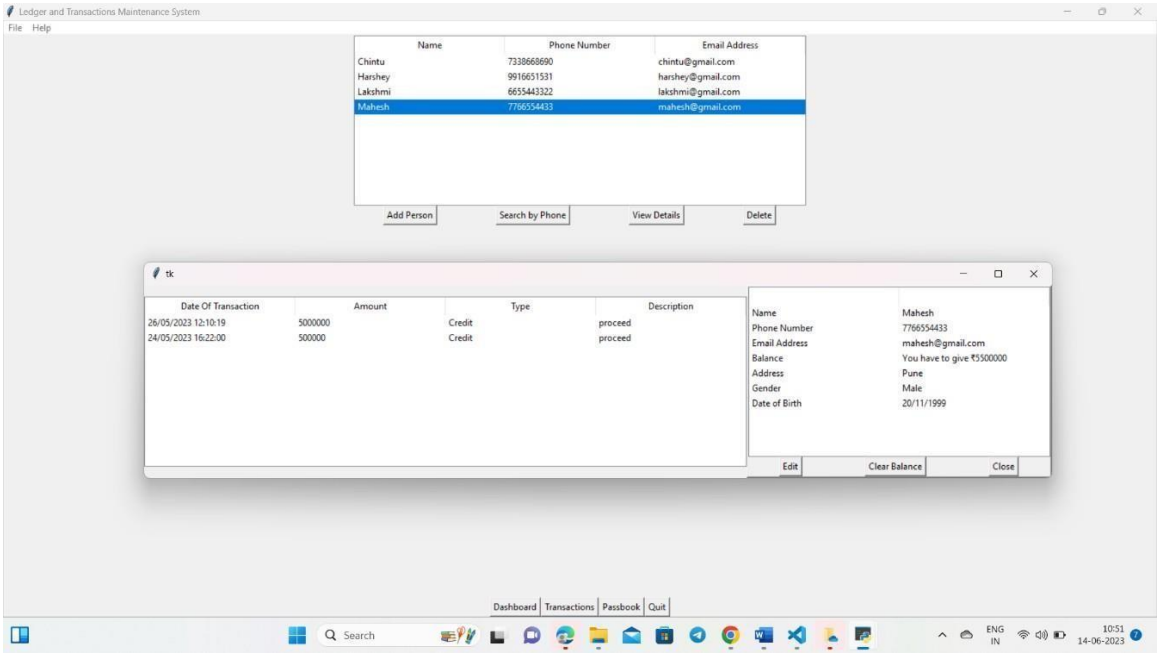

Figure 6.3.5 Delete Page

6.4TRANSACTION PAGE

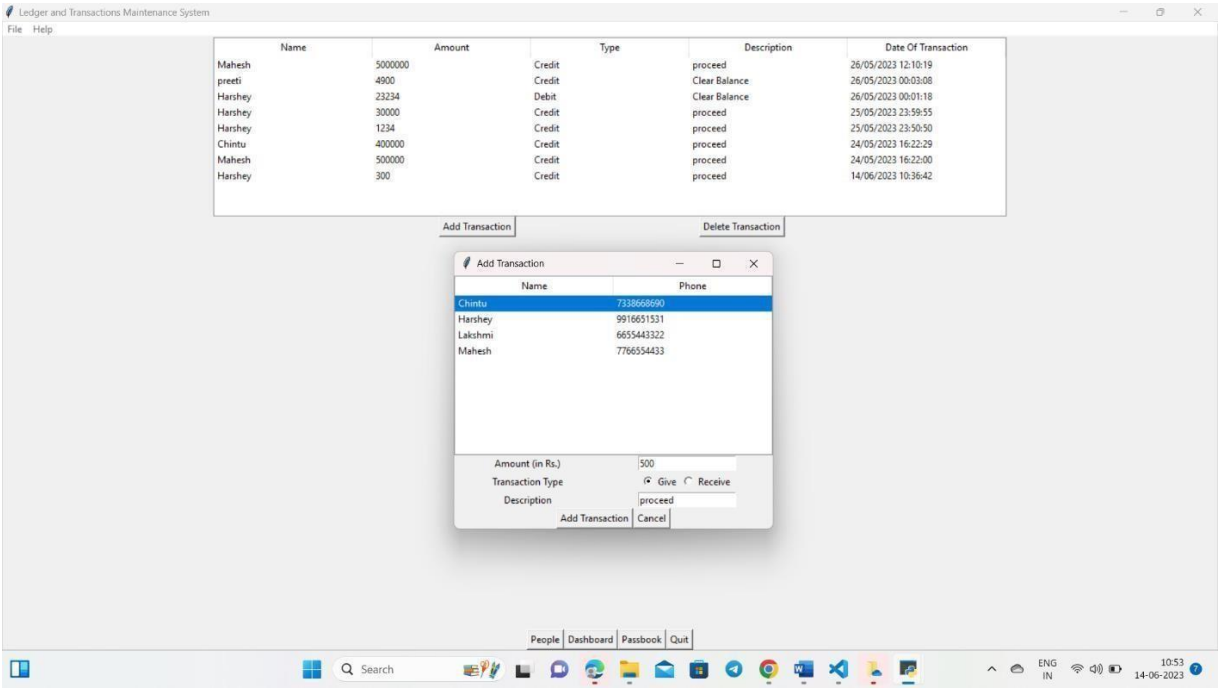

Figure 6.4: Transaction Page

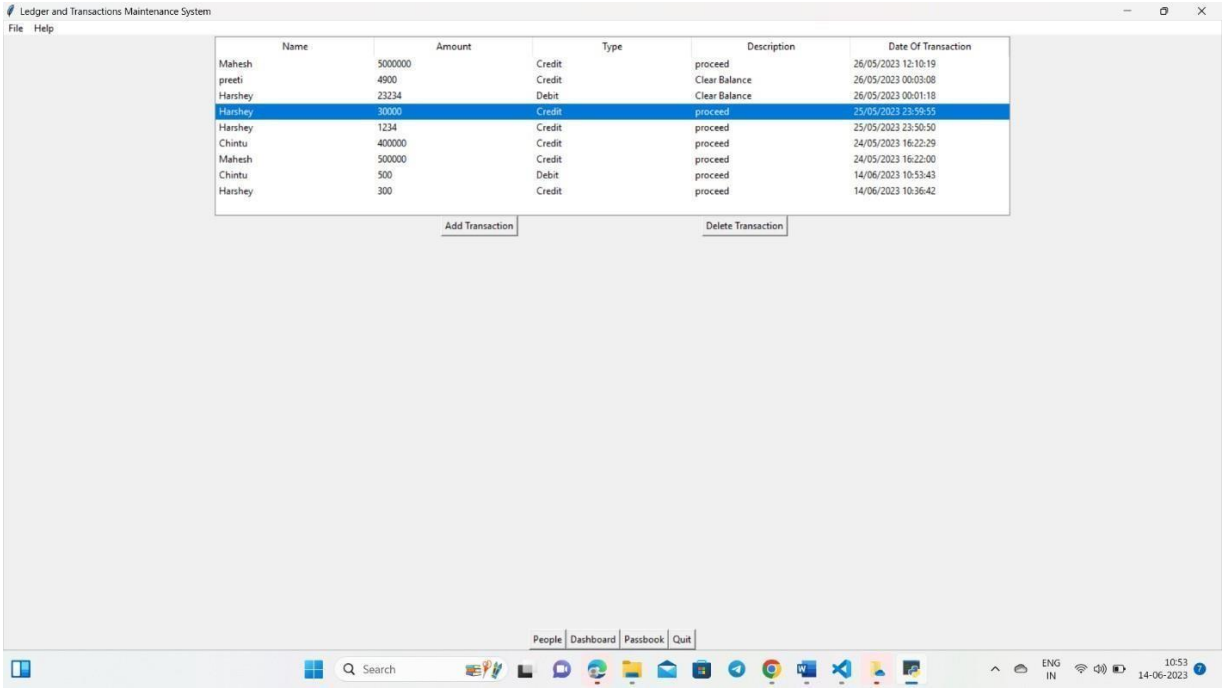

Figure 6.4.1 Delete Transaction Page

6.1 PASSBOOK PAGE

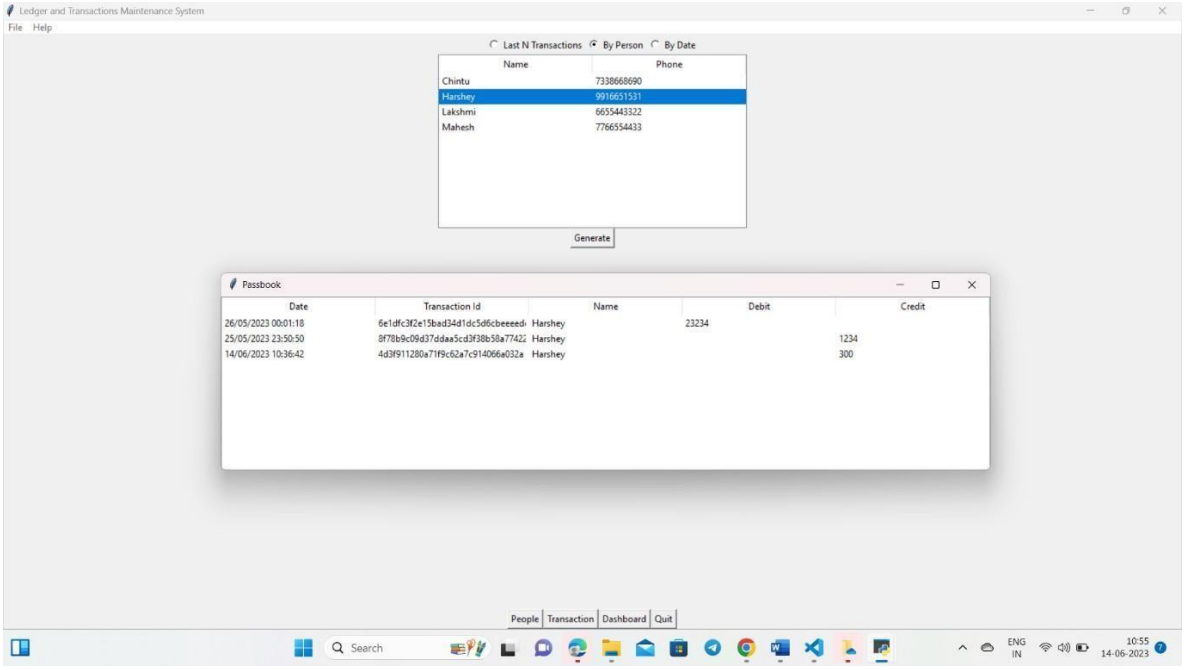

Figure 6.4: Passbook Page

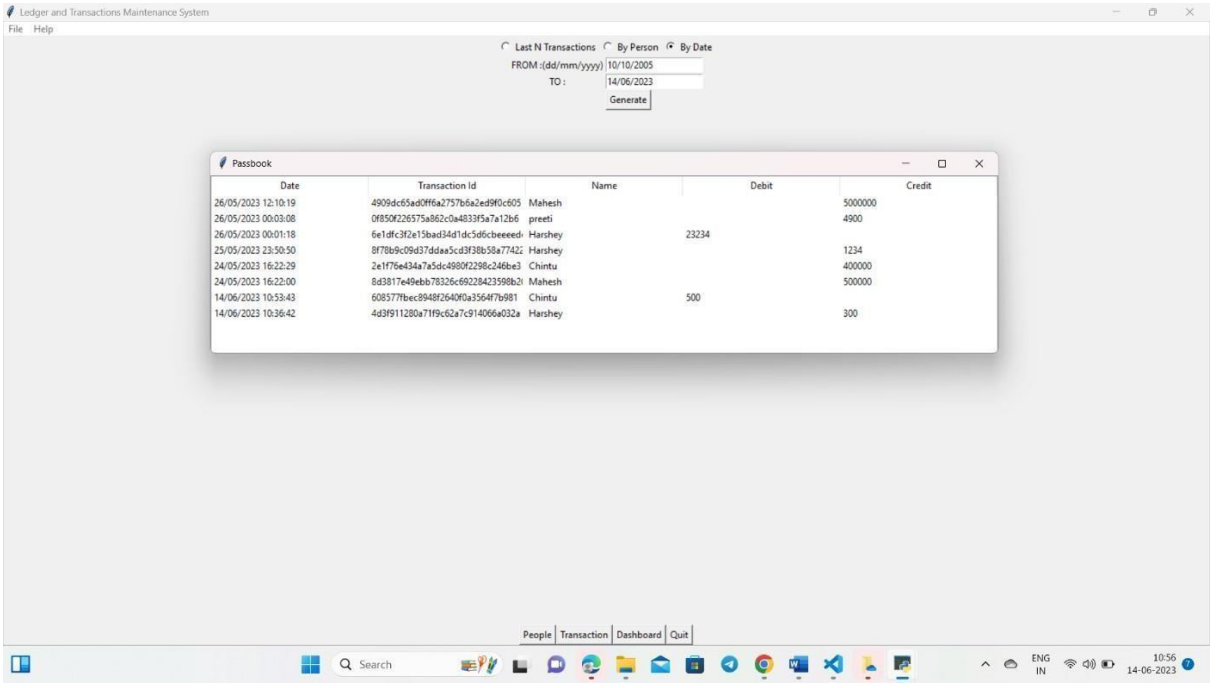

Figure 6.4.1 Search By Date Page

6.2 CLEAR BALANCE PAGE

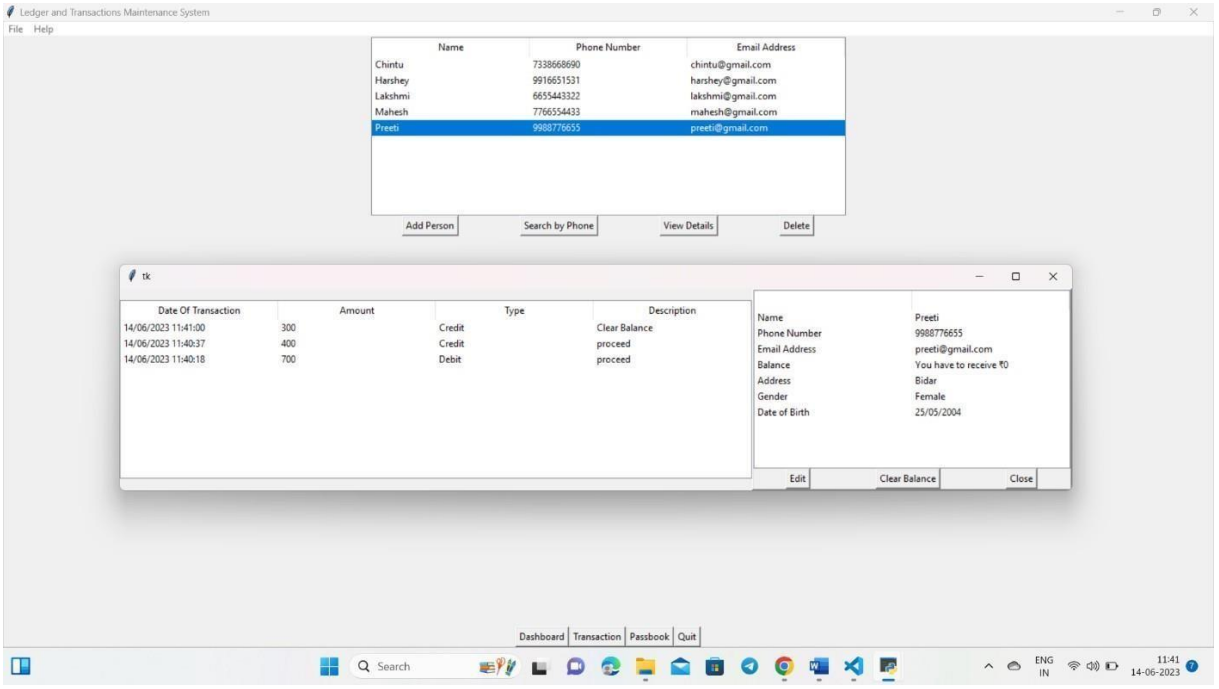

Figure 6.5: Clear Balance Page

CHAPTER 7

CONCLUSION

In conclusion, Ledger and Transaction Maintenance systems play a vital role in accurately recording, managing, and securing financial transactions. These systems provide organizations withthe ability to maintain an auditable and transparent record of their financial activities.

Through careful implementation and adherence to best practices, a well-designed ledger and transaction maintenance system ensures data integrity, privacy, and security. By following a systematic methodology that includes system design, transaction processing, ledger maintenance, reporting and analysis, system integration, testing and deployment, and ongoing maintenance, organizations can create a robust and reliable system.

Such systems can be further enhanced in the future advancements in security measures, the adoptionof blockchain technology, real-time transaction processing, integration with AI and machinelearning, regulatory compliance features, scalability and performance improvements, mobile and web interfaces, integration with the financial ecosystem, audit trail and compliance reporting, anda commitment to continuous improvements.

REFERENCES

[1] https://github.com/sandy13521/PatientInformation System