

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Santhibastawad Road, Machhe

Belagavi - 590018, Karnataka, India



**A REPORT
ON**

“Ledger Transaction Maintenance System”

**Submitted in the partial fulfilment of the requirements for the completion of the VI
Semester File Structures Laboratory with Mini Project [18ISL67] course of**

**BACHELOR OF ENGINEERING
IN
INFORMATION SCIENCE AND ENGINEERING**

For the Academic Year 2022-2023

Submitted by

ANUSHKA ROY

1JS20IS023

G SANJANA REDDY

1JS20IS036

Under the Guidance of

Mrs. Sahana V

Assistant Professor, Dept. of ISE, JSSATEB



2022-2023

**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING
JSS ACADEMY OF TECHNICAL EDUCATION**

**(Affiliated to VTU Belgavi and Approved by AICTE New Delhi)
JSSATE-B Campus, Dr. Vishnuvardhan Road, Bengaluru-560060**

JSS MAHAVIDYAPEETHA, MYSURU
JSS ACADEMY OF TECHNICAL EDUCATION

JSS Campus, Dr.Vishnuvardhan Road, Bengaluru-560060

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING



CERTIFICATE

This is to certify that the File Structures Mini Project entitled “EVENT MANAGEMENT” has been successfully completed by **ANUSHKA ROY [1JS20IS023]**, **G SANJANA REDDY [1JS20IS036]** of VI semester Information Science and Engineering from JSS Academy of Technical Education under Visvesvaraya Technological University during the academic year 2022- 2023.

Signature of the Guide

Mrs. Sahana V
Assistant Professor,
Dept. of ISE, JSSATEB

Signature of the HOD

Dr. Rekha P M,
Associate Professor & HOD,
Dept. of ISE, JSSATEB

Name of Examiners

Signature with date

- 1.
- 2.

ACKNOWLEDGEMENT

The success and final outcome of this project required a lot of guidance and assistance from many people and we are extremely privileged to have got this all along the completion of the project.

First and foremost, we would like to express my sincere gratitude to His Holiness Jagadguru Sri Sri Shivarathri Deshikendra Mahaswamiji for his divine blessings, without which the work wouldn't have been possible.

We would like to thank Dr Mrathyunjaya V Latte, Principal, ISSATE Bengaluru for his constant encouragement.

We would like to thank Dr. Rekha P M, Associate Professor and Head of the Department. Information Science and Engineering who has shared his opinions and thoughts, which helps in giving our presentation successfully.

We would like to express our gratitude to Mrs Sahana V, Asst. Professor, Dept. of ISE. For JSSATE, Bengaluru her guidance and assistance.

Finally, we take this opportunity to extend my earnest gratitude and respect to our parents, teaching & non-teaching staffs of the department and all my friends, for giving us valuable advices and support at all times in all possible ways.

ANUSHKA ROY [1JS20IS023]

G SANJANA REDDY [1JS20IS036]

ABSTRACT

This project report describes the design and implementation of a ledger transaction maintenance system using hashing indexing file structure. The system is designed to efficiently store and retrieve ledger transactions, while providing fast access to individual transactions. The system uses a combination of hashing and indexing to achieve this goal.

The hashing algorithm is used to map each transaction to a unique location in the file structure. This ensures that each transaction is stored in a fixed location, which makes it easy to find. The indexing mechanism is used to create a list of all transactions that match a given criteria. This makes it easy to retrieve a specific transaction or a set of transactions.

The system was implemented using the Python programming language. The system was tested using a set of synthetic transactions. The results of the tests showed that the system can store and retrieve transactions efficiently.

TABLE OF CONTENTS

| | |
|---|-----------|
| Chapter 1. Introduction | 6 |
| 1.1 Introduction to File Structure..... | 6 |
| 1.2 History..... | 6 |
| 1.3 About the File..... | 6 |
| 1.3 Application of File Structure..... | 7 |
| Chapter 2. System Analysis | 8 |
| 2.1 Analysis of Project..... | 8 |
| 2.2 Structure used to store the fields and records..... | 8 |
| 2.2.1 Field Structure..... | 8 |
| 2.2.2 Record Structures..... | 9 |
| 2.3 Operation Performed on a File..... | 10 |
| 2.3.1 Insertion..... | 10 |
| 2.3.2 Deletion..... | 10 |
| 2.3.3 Modify..... | 10 |
| 2.3.4 Searching..... | 11 |
| 2.4 File Structures used..... | 11 |
| Chapter 3. System Design | 12 |
| 3.1 Design of the Fields and Records..... | 12 |
| 3.2 User Interface..... | 13 |
| 3.2.1 Insertion of a record | 13 |
| 3.2.2 Display of a record..... | 13 |
| 3.2.3 Deletion of a record..... | 14 |
| 3.2.4 Assigning of a record..... | 14 |
| 3.2.5 Modification of a record..... | 14 |
| Chapter 4. Implementation | 15 |
| 4.1 About Python..... | 15 |
| 4.2 Source Code..... | 16 |
| Chapter 5. Testing | 26 |
| 5.1 Introduction..... | 26 |
| 5.2 Unit Testing..... | 26 |
| 5.3 Integration Testing..... | 27 |
| 5.4 System Testing..... | 28 |
| Chapter 6. Results | 30 |
| Chapter 7. Conclusion | 34 |
| Chapter 7. References | 35 |

CHAPTER 1

INTRODUCTION

1.1 Introduction to File Structures

File Structure is a combination of representations of data in files and operations for accessing the data. It allows applications to read, write and modify data. It supports finding the data that matches some search criteria or reading through the data in some particular order.

1.2 History

Early work with files presumed that files were on tape, since most files were. Access was sequential and the cost of access grew in direct proportion to the size of the file. As files grew intolerably large for unaided sequential access and as storage devices such as hard disks became available, indexes were added to files.

The indexes made it possible to keep a list of keys and pointers in a smaller file that could be searched more quickly. With key and pointer, the user had direct access to the large, primary file. But simple indexes had some of the same sequential flaws as the data file, and as the indexes grew, they too became difficult to manage, especially for dynamic files in which the set of keys changes.

1.3 About the File

There is one important distinction in file structures and that is the difference between the logical and physical organization of the data. On the whole a file structure will specify the logical structure of the data, that is the relationships that will exist between data items independently of the way in which these relationships may actually be realized within any computer. It is this logical aspect that we will concentrate on.

The physical organization is much more concerned with optimizing the use of the storage medium when a particular logical structure is stored on, or in it. Typically for every unit of a physical store there will be a number of units of the logical structure (probably records) to be stored in it.

A file system consists of two or three layers. Sometimes the layers are explicitly separated, and sometimes the functions are combined. The logical file system is responsible for interaction with the user application. It provides the application program interface (API) for file operations- open, close, read etc., and passes the requested operation to the layer below it for processing. The logical file system manages open file table entries and per-process file descriptors. This layer provides file access, directory operations, and security and protection.

The second optional layer is the virtual file system. This interface allows support for multiple concurrent instances of physical file systems, each of which is called a file system implementation.

The third layer is the physical file system. This layer is concerned with the physical operation of the storage device (eg. disk). It processes physical blocks being read or written. It handles buffering and memory management and is responsible for the physical placement of blocks in specific locations on the storage medium. The physical file system interacts with the device drivers or with the channel to drive the storage device.

1.4 Application of File Structure

Relative to other parts of a computer, disks are slow. One can pack thousands of megabytes on a disk that fits into a notebook computer. The time it takes to get information from even relatively slow electronic random-access memory (RAM) is about 120 nanoseconds. Getting the same information from a typical disk takes 30 milliseconds. Therefore, the disk access is a quarter of a million times longer than a memory access. Hence, disks are very slow compared to memory.

On the other hand, disks provide enormous capacity at much less cost than memory. They also keep the information stored on them when they are turned off. Tension between a disk's relatively slow access time and its enormous, nonvolatile capacity, is the driving force behind file structure design

CHAPTER 2

SYSTEM ANALYSIS

Systems analysis is the process of observing systems for troubleshooting or development purposes. It is applied to information technology, where computer-based systems require defined analysis according to their makeup and design.

2.1 Analysis of Project

The ledger transaction maintenance system project using hashing indexing and B-trees file structure is a well-designed system that can efficiently store and retrieve ledger transactions. The system uses a combination of hashing, indexing, and B-trees to achieve this goal. The hashing algorithm is used to map each transaction to a unique location in the file structure. This ensures that each transaction is stored in a fixed location, which makes it easy to find. The indexing mechanism is used to create a list of all transactions that match a given criteria. This makes it easy to retrieve a specific transaction or a set of transactions. The B-trees are used to store the index, which allows for efficient searching and retrieval of transactions.

2.2 Structure used to store the fields and records

2.2.1 Field Structure

The fields in a ledger transaction maintenance system using hashing indexing and B-trees file structure are separated using a hash function. A hash function is a mathematical function that takes an input of any length and produces an output of a fixed length. The hash function is used to map each transaction to a unique location in the file structure.

The hash function is designed to be collision-resistant, which means that it is unlikely that two different transactions will hash to the same location. This ensures that each transaction is stored in a unique location, which makes it easy to find.

The hash function is also designed to be efficient, which means that it can quickly calculate the hash of a transaction. This ensures that the system can store and retrieve transactions quickly.

The fields are separated using the hash function as follows:

- The hash function is applied to the transaction.
- The output of the hash function is used to determine the location of the transaction in the file structure.
- The transaction is stored at the specified location.
- This process ensures that the fields are separated in a way that is efficient and collision-resistant.

The different fields used are:

- 1.The transaction-id field is a unique identifier for the transaction. This is important because it allows for the easy identification of transactions.
- 2.The timestamp field is the time at which the transaction was created. This is important because it allows for the ordering of transactions.
- 3.The account field is the account that the transaction was made to or from. This is important because it allows for the tracking of financial activity.
- 4.The amount field is the amount of money involved in the transaction. This is important because it allows for the calculation of balances.

The record structure is designed to be efficient for both storage and retrieval. The transaction-id field is a unique identifier, so it can be used to quickly find a specific record. The timestamp field can be used to order records, and the account and amount fields can be used to filter records.

The record structure is also flexible enough to accommodate additional fields. For example, if you need to store the type of transaction, you can add a transaction-type field.

2.2.2 Record Structures

The records in a ledger transaction maintenance system using hashing indexing and B-trees file structure are separated using a combination of hashing and indexing.

The hashing algorithm is used to map each record to a unique location in the file structure. This ensures that each record is stored in a fixed location, which makes it easy to find.

The indexing mechanism is used to create a list of all records that match a given criteria. This makes it easy to retrieve a specific record or a set of records.

The records are separated using the hash function and the indexing mechanism as follows:

- The hash function is applied to the record.
- The output of the hash function is used to determine the location of the record in the file structure.

- The record is stored at the specified location.
- The index is updated to reflect the location of the record.
- This process ensures that the records are separated in a way that is efficient and collision-resistant.

2.3 Operation Performed on a File

2.3.1 Insertion

To perform insertion in hashing indexing, you can follow these steps:

1. Calculate the hash value for the key: Use a hash function to calculate the hash value for the key of the item you want to insert. The hash function should map the key to an index in the hash table.
2. Check if the calculated index is occupied: Look at the hash table at the calculated index to see if it is occupied or not. If the index is unoccupied, you can directly insert the item at that index. If the index is occupied, you need to handle collisions.
3. Insert the item: Once you find the appropriate index (either directly or by handling collisions), insert the item into the hash table at that index. The item can be stored as a value at the index or within the data structure used for collision handling (e.g., linked list node)..

2.3.2 Deletion

1. Calculate the hash value for the key: Use the same hash function that was used during insertion to calculate the hash value for the key of the item you want to delete. This will determine the index in the hash table where the item is expected to be located.
2. Search for the item: Look at the calculated index in the hash table to find the item you want to delete. If the item is found at the index, proceed with the deletion. If the item is not found, it means the item does not exist in the hash table.
3. Delete the item: Once you have located the item, remove it from the hash table. The specific steps to delete the item will depend on how the data is stored within the hash table.
 - If the item is stored as a value at the index, simply remove the item from the hash table by setting the value at that index to None or an appropriate placeholder value.
 - If the item is stored within a data structure like a linked list or another container, remove the item from the appropriate location within that data structure.

2.3.3 Modify

1. Calculate the hash value for the key: Use the same hash function that was used during insertion to calculate the hash value for the key of the item you want to modify. This will determine the index in the hash table where the item is expected to be located.
2. Search for the item: Look at the calculated index in the hash table to find the item you want to modify. If the item is found at the index, proceed with the modification. If the item is not found, it means the item does not exist in the hash table.

3. Update the item: Once you have located the item, update its attributes or values as desired. Modify the item directly if it is stored as a value at the index, or update the relevant data within the data structure used for collision handling.

2.3.4 Searching

To perform searching in hashing indexing, you can follow these steps:

1. Calculate the hash value for the key: Use the same hash function that was used during insertion to calculate the hash value for the search key. This will determine the index in the hash table where the item is expected to be located.
2. Search for the item: Look at the calculated index in the hash table to find the item you are searching for. If the item is found at the index, you have successfully found the item. If the item is not found, it means the item does not exist in the hash table.
3. Retrieve the item: Once you have located the item, retrieve and return it. The specific steps to retrieve the item will depend on how the data is stored within the hash table.
 - If the item is stored as a value at the index, simply retrieve the value at that index.
 - If the item is stored within a data structure like a linked list or another container, retrieve the item from the appropriate location within that data structure.

2.4 File Structures used:

Hashing and Indexing for Ledger Transaction Maintenance:

Hashing and indexing are two techniques that can be used to store and retrieve data efficiently. Hashing is a technique for mapping data to a unique value. This value can then be used to store or retrieve the data. Indexing is a technique for creating a list of all data items that match a given criteria. This list can then be used to quickly find the data items that you are looking for.

In a ledger transaction maintenance system using hashing indexing file structure, hashing is used to map each transaction to a unique location in the file structure. This ensures that each transaction is stored in a fixed location, which makes it easy to find. Indexing is used to create a list of all transactions that match a given criteria. This makes it easy to retrieve a specific transaction or a set of transactions.

The combination of hashing and indexing provides a number of advantages for storing and retrieving ledger transactions. First, it is efficient. The hashing algorithm can quickly map a transaction to a unique location in the file structure. The index can quickly find all transactions that match a given criteria. Second, it is scalable. The system can easily be expanded to accommodate more transactions. Third, it is fault-tolerant. If a transaction is lost, it can be easily retrieved from the index.

CHAPTER 3

SYSTEM DESIGN

The System Design Document describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces. Systems design is the process of defining the architecture, modules, interfaces, and data for a system to satisfy specified requirements. Systems design could be seen as the application of systems theory to product development.

3.1 Design of the Fields and Records

The design of the fields and records for ledger transaction maintenance system using hashing indexing file structure:

Fields:

The following fields are used to store ledger transactions:

- Transaction ID: A unique identifier for the transaction.
- Timestamp: The time at which the transaction was created.
- Account: The account that the transaction was made to or from.
- Amount: The amount of money involved in the transaction.
- Description: A description of the transaction.

Records:

- The fields are stored in records. Each record contains a single transaction.
- The records are stored in a hash table, which is a data structure that maps keys to values.
- The keys are the transaction IDs, and the values are the records.

Index:

The index is used to store a list of all transactions that match a given criteria.

The index is stored in a B-tree, which is a data structure that is optimized for searching and retrieval.

The index contains the transaction IDs and the corresponding locations in the hash table.

File Structure:

The hash table and the index are stored in a file structure. The file structure is designed to be efficient for both reading and writing.

Design Considerations:

The following design considerations were considered when designing the fields and records for the ledger transaction maintenance system:

3.2 User Interface

The user interface (UI), in the industrial design field of human–computer interaction, is the space where interactions between humans and machines occur. The goal of this interaction is to allow effective operation and control of the machine from the human end, whilst the machine simultaneously feeds back information that aids the operators' decision-making process.

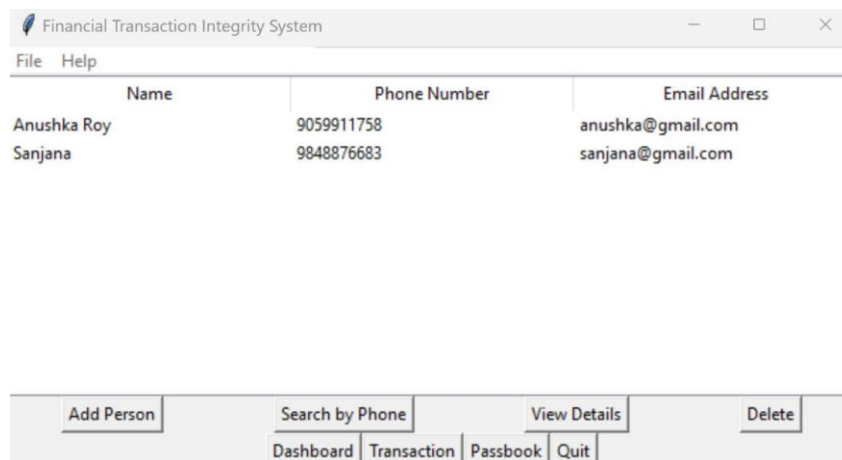


Figure 3.1 User Interface

3.2.1 Insertion of a record

The steps on how to insert a record in ledger transaction maintenance system using hashing indexing file structure:

1. Hash the transaction ID. The transaction ID is hashed to a unique value. This value is used to store the transaction in the hash table and the index.
2. Store the record in the hash table. The record is stored in the hash table using the hash value of the transaction ID as the key.
3. Update the index. The index is updated to reflect the location of the record in the hash table.

3.2.2 Display of a record

The steps on how to display a record in ledger transaction maintenance system using hashing indexing file structure:

1. Retrieve the record from the hash table. The record is retrieved from the hash table using the transaction ID as the key.
2. Display the record. The record is displayed to the user.

Here are the detailed steps:

The transaction ID is used as the key to retrieve the record from the hash table. The hash

table is a data structure that maps keys to values. The keys are the hash values of the transaction IDs, and the values are the records.

The record is displayed to the user. The record contains the following information:

- Transaction ID
- Timestamp
- Account
- Amount
- Description

3.2.3 Deletion of a record

The steps on how to delete a record in ledger transaction maintenance system using hashing indexing file structure:

1. Retrieve the record from the hash table. The record is retrieved from the hash table using the transaction ID as the key.
2. Remove the record from the hash table. The record is removed from the hash table.
3. Update the index. The index is updated to reflect the removal of the record from the hash table.

3.2.4 Assigning of a record

The steps on how to assign a record in ledger transaction maintenance system using hashing indexing file structure:

1. Hash the transaction ID. The transaction ID is hashed to a unique value. This value is used to store the transaction in the hash table and the index.
2. Store the record in the hash table. The record is stored in the hash table using the hash value of the transaction ID as the key.
3. Update the index. The index is updated to reflect the location of the record in the hash table.
4. Assign the record to a user. The record is assigned to a user by updating the record with the user's ID.

3.2.5 Modification of a record

The steps on how to modify a record in ledger transaction maintenance system using hashing indexing file structure:

1. Retrieve the record from the hash table. The record is retrieved from the hash table using the transaction ID as the key.
2. Update the record with the new values. The record is updated with the new values.
3. Store the updated record in the hash table. The updated record is stored in the hash table.
4. Update the index. The index is updated to reflect the changes to the record.

CHAPTER 4

IMPLEMENTATION

Implementation is the process of defining how the system should be built, ensuring that it is operational and meets quality standards. It is a systematic and structured approach for effectively integrating a software-based service or component into the requirements of end users.

4.1 About PYTHON

Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation via the off-side rule.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s as a successor to the ABC programming language and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000. Python 3.0, released in 2008, was a major revision not completely backward-compatible with earlier versions. Python 2.7.18, released in 2020, was the last release of Python 2. Python consistently ranks as one of the most popular programming languages.

4.1.1 About Tkinter

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task.

To create a tkinter app:

- Importing the module – tkinter

- Create the main window (container)

- Add any number of widgets to the main window

- Apply the event Trigger on the widgets.

- Importing tkinter is same as importing any other module in the Python code.

Note that the name of the module in Python 2.x is 'Tkinter' and in Python 3.x it is 'tkinter'.

4.2 Source Code

MAIN.py

```

from tkinter import *
import tkinter.ttk as table
import webbrowser as web
import people
import dashboard
import transaction
import passbook

# TODO: Testing the switching between frames (change_frame meethod)

def change_frame(frame, name, prev="None"):
    global window, people_button, transactions_button, passbook_button
    frame.destroy()
    if name == "Dashboard":
        frame = dashboard.get_frame(window)
        frame.pack(side=TOP)
        if prev == "People" or prev == "Transaction" or prev == "Passbook":
            people_button.config(text="People", command=lambda: change_frame(frame, "People", "Dashboard"))
            transactions_button.config(text="Transaction",
                                      command=lambda: change_frame(frame, "Transaction", "Dashboard"))
            passbook_button.config(text="Passbook", command=lambda: change_frame(frame, "Passbook", "Dashboard"))
            people_button.pack(side=LEFT)
            transactions_button.pack(side=LEFT)
            passbook_button.pack(side=LEFT)

    elif name == "People":
        if prev == "Transaction":
            people_button.config(text="Dashboard", command=lambda: change_frame(frame, "Dashboard", "People"))
            transactions_button.config(text="Transaction", command=lambda: change_frame(frame, "Transaction", "People"))
            passbook_button.config(text="Passbook", command=lambda: change_frame(frame, "Passbook", "People"))
        if prev == "Passbook":
            people_button.config(text="Dashboard", command=lambda: change_frame(frame, "Dashboard", "People"))
            transactions_button.config(text="Transaction", command=lambda: change_frame(frame, "Transaction", "People"))
            passbook_button.config(text="Passbook", command=lambda: change_frame(frame, "Passbook", "People"))
        if prev == "Dashboard":
            people_button.config(text="Dashboard", command=lambda: change_frame(frame, "Dashboard", "People"))
            transactions_button.config(command=lambda: change_frame(frame, "Transaction", "People"))
            passbook_button.config(command=lambda: change_frame(frame, "Passbook", "People"))

    frame = people.get_frame(window)
    frame.pack(side=TOP)

    elif name == "Transaction":
        if prev == "People":
            transactions_button.config(text="Dashboard",
                                      command=lambda: change_frame(frame, "Dashboard", "Transaction"))
            people_button.config(text="People", command=lambda: change_frame(frame, "People", "Transaction"))
            passbook_button.config(text="Passbook", command=lambda: change_frame(frame, "Passbook", "Transaction"))
        if prev == "Passbook":
            transactions_button.config(text="Dashboard",
                                      command=lambda: change_frame(frame, "Dashboard", "Transaction"))
            people_button.config(text="People", command=lambda: change_frame(frame, "People", "Transaction"))
            passbook_button.config(text="Passbook", command=lambda: change_frame(frame, "Passbook", "Transaction"))
        if prev == "Dashboard":
            transactions_button.config(text="Dashboard",
                                      command=lambda: change_frame(frame, "Dashboard", "Transaction"))
            people_button.config(command=lambda: change_frame(frame, "People", "Transaction"))
            passbook_button.config(command=lambda: change_frame(frame, "Passbook", "Transaction"))
        frame = transaction.get_frame(window)
        frame.pack(side=TOP)

    elif name == "Passbook":
        if prev == "People":
            passbook_button.config(text="Dashboard",
                                   command=lambda: change_frame(frame, "Dashboard", "Passbook"))
            transactions_button.config(text="Transaction",
                                      command=lambda: change_frame(frame, "Transaction", "Passbook"))
            people_button.config(text="People", command=lambda: change_frame(frame, "People", "Passbook"))
        if prev == "Transaction":
            passbook_button.config(text="Dashboard",
                                   command=lambda: change_frame(frame, "Dashboard", "Passbook"))
            people_button.config(text="People", command=lambda: change_frame(frame, "People", "Passbook"))
            transactions_button.config(text="Transaction",

```



```

        command=lambda: change_frame(frame, "Transaction", "Passbook"))
    if prev == "Dashboard":
        passbook_button.config(text="Dashboard", command=lambda: change_frame(frame, "Dashboard", "Passbook"))
        transactions_button.config(command=lambda: change_frame(frame, "Transaction", "Passbook"))
        people_button.config(command=lambda: change_frame(frame, "People", "Passbook"))
    frame = passbook.get_frame(window)
    frame.pack(side=TOP)

# This method adds the menu to the program.
def add_menu(window):
    menu = Menu(window)
    window.config(menu=menu)

    files_menu = Menu(menu)
    help_menu = Menu(menu)

    files_menu.add_command(label="Exit", command=window.quit)

    about_menu = Menu(help_menu)
    about_menu.add_command(label="Sandeep N S",
                           command=lambda: web.open("https://www.linkedin.com/in/sandeep-n-s-6b3888165/"))
    about_menu.add_command(label="Suprad S Parashar",
                           command=lambda: web.open("https://www.linkedin.com/in/supradparashar/"))

    help_menu.add_cascade(label="About", menu=about_menu)

    menu.add_cascade(label="File", menu=files_menu)
    menu.add_cascade(label="Help", menu=help_menu)

```

DASHBOARD.py

```

from tkinter import *
import transaction
import people
import tkinter.ttk as table
from datetime import datetime

def get_frame(window):
    frame = Frame(window, name = "dashboard")

    now = datetime.now()
    date_string = now.strftime("%d %B %Y")

    date = Label(frame, text = date_string, borderwidth = 2, padx = 5, pady = 5)
    date.grid(row = 0, column = 0, columnspan = 6, rowspan = 1)

    pay_money_card = Frame(frame, borderwidth = 2, relief = "raised", padx = 5, pady = 5)
    recieve_money_card = Frame(frame, borderwidth = 2, relief = "raised", padx = 5, pady = 5)
    transactions_card = Frame(frame, borderwidth = 2, relief = "raised", padx = 5, pady = 5)

    give, get = people.get_total_balance_dashboard()
    give_text = Label(pay_money_card, text = "Pay ₹" + str(give))
    give_text.pack()
    get_text = Label(recieve_money_card, text = "Receive ₹" + str(get))
    get_text.pack()

    trans_table = table.Treeview(transactions_card)
    trans_table.grid(row = 0, column = 0, columnspan = 6)
    trans_table["columns"] = ["name", "amount", "type", "des", "dot"]
    trans_table["show"] = "headings"
    trans_table.heading("name", text = "Name")
    trans_table.heading("amount", text = "Amount")
    trans_table.heading("type", text = "Type")
    trans_table.heading("dot", text = "Date Of Transaction")
    trans_table.heading("des", text = "Description")
    index = 0
    for trans in transaction.TRANSACTIONS[:5]:
        trans_table.insert("", index, values=(trans.person_name, trans.amount, trans.type, trans.description, trans.date))
        index += 1
    pay_money_card.grid(row = 1, column = 0, columnspan = 3, rowspan = 2)
    recieve_money_card.grid(row = 1, column = 3, columnspan = 3, rowspan = 2)
    transactions_card.grid(row = 3, column = 1, columnspan = 4, rowspan = 2)

    return frame

```

HELPER.py

```
import re as regex
import pickle
import os
import people
```

A lambda function to check if the email address passed is valid or not.

```
isEmailValid = lambda email: True if email == "" else regex.search("^\w+([\.|-]?\w+)*@\w+([\.|-]?\w+)*\.(\w{2,3})+$", email)
```

A method to check if a given date is valid.

```
def isDateValid(date):
    if date == "":
        return True
    isLeapYear = lambda year: (year % 4 == 0 and year % 100 != 0) or year % 400 == 0
    try:
        dates = list(map(int, date.split("/")))
        if dates[1] == 2:
            if isLeapYear(dates[2]) and dates[0] in range(1, 30):
                return True
            elif (not isLeapYear) and dates[0] in range(1, 29):
                return True
        elif dates[1] in [1, 3, 5, 7, 8, 10, 12] and dates[0] in range(1, 32):
            return True
        elif dates[1] not in [1, 3, 5, 7, 8, 10, 12] and dates[0] in range(1, 31):
            return True
        else:
            return False
    except:
        return False
```

def is_amount_valid(amount):

```
    if amount == "":
        return False
    elif not amount.isdigit():
        return False
    elif int(amount) == 0:
        return False
    else:
        return True
```

A method to check if the passed phone number is valid or not.

```
def isPhoneValid(phone):
    if phone == "":
        return False
    if phone[0] not in ['6', '7', '8', '9']:
        return False
    elif len(phone) != 10:
        return False
    elif not phone.isdigit():
        return False
    else:
        return True
```

Returns the list of people.

```
def read_people(people_file_name, people_index_file_name):
    people_names = []
    try:
        with open(people_file_name, "rb") as people_file, open(people_index_file_name, "r") as index_file:
            while True:
                line = index_file.readline()
                if line == "":
                    break
                data = eval(line)
                people_file.seek(data[1])
                person = pickle.load(people_file)
                people_names.append(person)
    except EOFError:
        pass
    return people_names
```

Returns the list of people.

```
def write_people(people_list, people_file_name, people_index_file_name):
    indices = {}
    with open(people_file_name, "wb") as people_file, open(people_index_file_name, "w") as index_file:
        for person in people_list:
            index = people_file.tell()
            data = tuple([person.id, index])
```

```

        indices[data[0]] = data[1]
        pickle.dump(person, people_file)
        index_file.write(str(data) + "\n")
    return indices

# Refreshes the table to reflect changes.
def refresh_table(table, data):
    for entry in table.get_children():
        table.delete(entry)
    index = 0
    for entry in data:
        table.insert("", index, values=entry.get_table_data(), tags=entry.id)
        index += 1

def load_indices(people_file_name, people_index_file_name):
    indices = {}
    try:
        with open(people_index_file_name, "r") as file:
            while True:
                line = file.readline()
                if line == "":
                    break
                data = eval(line)
                indices[data[0]] = data[1]
    except EOFError:
        pass
    except FileNotFoundError:
        if not os.path.exists("files"):
            os.mkdir("files")
        with open(people_file_name, "wb") as _, open(people_index_file_name, "w") as _:
            pass
    return indices

def read_transactions(transaction_file_name):
    transactions = []
    try:
        with open(transaction_file_name, "rb") as file:
            while True:
                transactions.append(pickle.load(file))
    except EOFError:
        pass
    except FileNotFoundError:
        if not os.path.exists("files"):
            os.mkdir("files")
        with open(transaction_file_name, "wb") as _:#, open(transaction.INDEX_FILE_NAME, "w") as _:
            pass
    transactions.sort(key = lambda a_trans: a_trans.date, reverse = True)
    return transactions

def write_transactions(transactions, transaction_file_name):
    with open(transaction_file_name, "wb") as file:
        for trans in transactions:
            pickle.dump(trans, file)

```

PASSBOOK.py

```

from tkinter import *
import tkinter.ttk as table
from datetime import datetime
import tkinter.messagebox as dialog
import people
import transaction
import helper

pre_frame = 0

def get_frame(window):
    frame = Frame(window, name="passbook")

    top_frame = Frame(frame)
    top_frame.pack(side=TOP)

    bottom_frame = Frame(frame)
    bottom_frame.pack(side=BOTTOM)

    n_label = Label(bottom_frame, text="Last Transaction")
    n_entry = Entry(bottom_frame)

```

```

people_choices = table.Treeview(bottom_frame)
people_choices["columns"] = ["name", "phone"]
people_choices["show"] = "headings"
people_choices.heading("name", text="Name")
people_choices.heading("phone", text="Phone")
index = 0
for person in people.PEOPLE:
    people_choices.insert("", index, values=(person.name, person.phone), tags=person.id)
    index += 1

from_label = Label(bottom_frame, text="FROM : (dd/mm/yyyy)")
from_entry = Entry(bottom_frame)
to_label = Label(bottom_frame, text="TO :")
to_entry = Entry(bottom_frame)
ok_button = Button(bottom_frame, text="Generate")

def bottom(radio_button_var):
    global pre_frame
    clear_frame()
    if radio_button_var == 1:
        n_label.grid(row=1, column=1)
        n_entry.grid(row=1, column=2)
        pre_frame = 1
        ok_button.config(command=lambda: display(n=n_entry.get()))
        # ok_button.grid(row=2, columnspan=1, padx=5, pady=5, sticky="nsew")
        ok_button.grid(row=2, column=2, columnspan=1, padx=2, sticky='W')

    elif radio_button_var == 2:
        people_choices.grid(row=2, column=0, columnspan=2)
        ok_button.config(
            command=lambda: display(name=people_choices.item(people_choices.selection()[0])['values'][0])) # ,
        ok_button.grid(row=3, column=0, columnspan=1, sticky="E")
        pre_frame = 2

    elif radio_button_var == 3:
        from_label.grid(row=1, column=1)
        from_entry.grid(row=1, column=2)
        to_label.grid(row=2, column=1)
        to_entry.grid(row=2, column=2)
        ok_button.config(
            command=lambda: display(from_date=from_entry.get(), to_date=to_entry.get())) # , width=10, height=1)
        ok_button.grid(row=3, column=2, columnspan=1, sticky='W')
        pre_frame = 3

def clear_frame():
    if pre_frame == 0:
        pass
    elif pre_frame == 1:
        n_label.grid_forget()
        n_entry.grid_forget()
        ok_button.grid_forget()
    elif pre_frame == 2:
        people_choices.grid_forget()
        ok_button.grid_forget()
    elif pre_frame == 3:
        from_label.grid_forget()
        to_entry.grid_forget()
        from_entry.grid_forget()
        to_label.grid_forget()
        ok_button.grid_forget()

last_n = Radiobutton(top_frame, text="Last N Transactions", value=0, command=lambda: bottom(1))
by_person = Radiobutton(top_frame, text="By Person", value=1, command=lambda: bottom(2))
by_date = Radiobutton(top_frame, text="By Date", value=2, command=lambda: bottom(3))
last_n.select()
bottom(1)

last_n.pack(side=LEFT)
by_person.pack(side=LEFT)
by_date.pack(side=LEFT)

return frame

def display(n=None, name=None, from_date=None, to_date=None):
    passbook_sub_window = Tk()
    passbook_sub_window.title("Passbook")

    passbook_table = table.Treeview(passbook_sub_window)

```

```

passbook_table.grid(row=0, column=0, columnspan=6)
passbook_table["columns"] = ["date", "trans_id", "person_name", "withdrawal", "deposits"]
passbook_table["show"] = "headings"
passbook_table.heading("date", text="Date")
passbook_table.heading("trans_id", text="Transaction Id")
passbook_table.heading("person_name", text="Name")
passbook_table.heading("withdrawal", text="Debit")
passbook_table.heading("deposits", text="Credit")
if n != None:
    n = int(n)
    balance = 0
    index = 0
    for trans in transaction.TRANSACTIONS[:n]:
        if trans.type == "Debit":
            balance += trans.amount
            passbook_table.insert("", index,
                                values=(trans.date, trans.id, trans.person_name, trans.amount, " "))
        else:
            balance -= trans.amount
            passbook_table.insert("", index,
                                values=(trans.date, trans.id, trans.person_name, " ", trans.amount))
        index += 1
    passbook_sub_window.mainloop()

elif name != None:
    index = 0
    for trans in transaction.TRANSACTIONS:
        if trans.person_name == name:
            if trans.type == "Debit":
                passbook_table.insert("", index,
                                    values=(trans.date, trans.id, trans.person_name, trans.amount, " "))
                index += 1
            else:
                passbook_table.insert("", index,
                                    values=(trans.date, trans.id, trans.person_name, " ", trans.amount))
                index += 1
    passbook_sub_window.mainloop()

elif from_date != None and to_date != None:
    if helper.isDateValid(from_date) and helper.isDateValid(to_date):
        from_date1 = datetime.strptime(from_date, "%d/%m/%Y").date()
        to_date1 = datetime.strptime(to_date, "%d/%m/%Y").date()
        if from_date1 <= to_date1 and to_date1 <= datetime.now().date():
            index = 0
            for trans in transaction.TRANSACTIONS:
                d = datetime.strptime(trans.date, "%d/%m/%Y %H:%M:%S").date()
                if from_date1 <= d <= to_date1:
                    if trans.type == "Debit":
                        passbook_table.insert("", index, values=(
                            trans.date, trans.id, trans.person_name, trans.amount, " "))
                    else:
                        passbook_table.insert("", index, values=(
                            trans.date, trans.id, trans.person_name, " ", trans.amount))
                    index += 1
            else:
                dialog.showinfo("Invalid Data", "Invalid Date Entered")
                passbook_sub_window.destroy()
        else:
            dialog.showinfo("Invalid Data", "Invalid Date Enter ! Please Enter in (dd/mm/yyyy) format")
            passbook_sub_window.destroy()

```

PEOPLE.py

```

from tkinter import *
import tkinter.messagebox as dialog
from tkinter.simpledialog import askstring
import tkinter.ttk as table
import hashlib as hash
import pickle
from datetime import datetime
import helper
import transaction

FILE_NAME = "files/people.lms"
INDEX_FILE_NAME = "files/people_index.txt"

```

A Person class to store the details of a person.

```

class Person:
    # The constructor of the class.
    def __init__(self, person_id, name, email, phone, address, gender, dob, balance=0):
        self.id = person_id
        self.name = name
        self.email = email
        self.phone = phone
        self.address = address
        self.gender = gender
        self.dob = dob
        self.balance = balance

    # Returns the Data of the person.
    def get_data(self):
        return [
            ("Name", self.name),
            ("Phone Number", self.phone),
            ("Email Address", self.email),
            ("Balance", "You have to {} ₹{}".format("give" if self.balance < 0 else "receive", abs(self.balance))),
            ("Address", self.address),
            ("Gender", "Male" if self.gender == 0 else "Female" if self.gender == 1 else "Other"),
            ("Date of Birth", self.dob)
        ]

    # Returns a tuple having the name, phone and email of the person.
    def get_table_data(self):
        return self.name, self.phone, self.email

    # Returns a frame containing the key-value pair of information of the person.
    def get_data_frame(self, window):
        frame = Frame(window, borderwidth=2, relief="raised")
        details_table = table.Treeview(frame)
        details_table["columns"] = ["parameter", "value"]
        details_table["show"] = "headings"
        index = 0
        for data in self.get_data():
            details_table.insert("", index, values=data)
            index += 1
        details_table.grid(row=0, column=0, columnspan=3)
        return frame

def get_total_balance_dashboard():
    give, get = 0, 0
    for person in PEOPLE:
        if person.balance < 0:
            give += abs(person.balance)
        else:
            get += person.balance
    return give, get

def search_person(people_table):
    person_phone = askstring("Search", "Enter Phone Number")
    status = False
    if person_phone != None:
        for people in PEOPLE:
            if people.phone == person_phone:
                view_person(people.id, people_table, True)
                status = True
    if not status:
        dialog.showerror("Not Found", "There exists no person with the phone number {}".format(person_phone))

# This method takes in the main window of the program as a parameter and generates and returns the frame of the Person Module.
def get_frame(window):
    frame = Frame(window, name="people")

    # Table to display the people.
    people_table = table.Treeview(frame)
    people_table.grid(row=0, column=0, columnspan=4)
    people_table["columns"] = ["name", "phone", "email"]
    people_table["show"] = "headings"
    people_table.heading("name", text="Name")
    people_table.heading("email", text="Email Address")
    people_table.heading("phone", text="Phone Number")

    helper.refresh_table(people_table, PEOPLE)

```

```

# Buttons to add, modify and delete a person.
add_button = Button(frame, text="Add Person", command=lambda: add_person(people_table))
search_button = Button(frame, text="Search by Phone", command=lambda: search_person(people_table))
view_button = Button(frame, text="View Details",
                      command=lambda: view_person(people_table.item(people_table.selection()[0]), people_table))
delete_button = Button(frame, text="Delete",
                      command=lambda: delete_person(people_table.item(people_table.selection()[0]), people_table))

add_button.grid(row=1, column=0)
search_button.grid(row=1, column=1)
view_button.grid(row=1, column=2)
delete_button.grid(row=1, column=3)

return frame

# Opens a window displaying the information and the recent transactions of the person.
def view_person(item, people_table, direct=False):
    person_id = str(item["tags"][0]) if not direct else item
    person = get_person(INDICES[person_id])

    person_details_window = Tk()
    transactions_frame = Frame(person_details_window)

    trans_table = table.Treeview(transactions_frame)
    trans_table.grid(row=0, column=0, columnspan=4)
    trans_table["columns"] = ["date", "amount", "type", "des"]
    trans_table["show"] = "headings"
    trans_table.heading("amount", text="Amount")
    trans_table.heading("type", text="Type")
    trans_table.heading("date", text="Date Of Transaction")
    trans_table.heading("des", text="Description")
    index = 0
    for trans in transaction.get_person_transactions(person):
        trans_table.insert("", index, values=(trans.date, trans.amount, trans.type, trans.description))
        index += 1
    transactions_frame.grid(row=0, column=0, columnspan=2)
    person_frame = person.get_data_frame(person_details_window)
    person_frame.grid(row=0, column=2, columnspan=2)
    edit_button = Button(person_frame, text="Edit", command=lambda: add_person(people_table, person))
    edit_button.grid(row=1, column=0)
    clear_balance_button = Button(person_frame, text="Clear Balance",
                                command=lambda: clear_balance(person, trans_table, person_details_window))
    clear_balance_button.grid(row=1, column=1)
    close_button = Button(person_frame, text="Close", command=person_details_window.destroy)
    close_button.grid(row=1, column=2)
    person_details_window.mainloop()

def get_person(index):
    with open(FILE_NAME, "rb") as file:
        file.seek(index)
        return pickle.load(file)

def clear_balance(person, trans_table, person_details_window):
    if abs(person.balance) == 0:
        dialog.showerror("Error", "Balance is Already Zero")
    else:
        result = dialog.askquestion("Clear Balance",
                                   "Do you want to clear the balance of ₹{} of {}?" format(abs(person.balance),
                                           person.name), icon='warning')
        if result == 'yes':
            now = datetime.now()
            dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
            trans_id = hash.md5((person.id + str(-person.balance) + dt_string).encode()).hexdigest()
            trans = transaction.Transaction(trans_id, person.name, person.id, "Clear Balance", abs(person.balance),
                                           dt_string, 1 if person.balance > 0 else 0)
            transaction.add_transaction(trans_table, trans)
            person_details_window.destroy()

# This method deletes the selected item from the people table.
def delete_person(item, people_table):
    global PEOPLE, INDICES
    delete_id = str(item["tags"][0])
    print(delete_id)
    print("-----")
    delete_index = -1
    for i in range(len(PEOPLE)):

```

```

    print(type(PEOPLE[i].id), type(delete_id))
    if PEOPLE[i].id == delete_id:
        print()
        delete_index = i
        break
    deleted_person = PEOPLE[delete_index]
    result = dialog.askquestion("Delete Person", "Do you want to delete {} from contacts?".format(deleted_person.name),
                               icon='warning')
    if result == 'yes':
        PEOPLE.pop(delete_index)
        INDICES = helper.write_people(PEOPLE, FILE_NAME, INDEX_FILE_NAME)
        dialog.showinfo("Deletion Successful",
                        "The person named {} has been deleted from the record.".format(deleted_person.name))
        helper.refresh_table(people_table, PEOPLE)
        r = dialog.askquestion("Delete Transactions",
                               "Do you want to delete the transactions related to {}".format(deleted_person.name),
                               icon='warning')
        if r == "yes":
            transaction.remove_person_transactions(deleted_person)

# Changes the balance of the person.
def change_balance(person_id, amount):
    global PEOPLE, INDICES
    for person in PEOPLE:
        if person.id == person_id:
            person.balance += amount
            print(person.balance)
            break
    INDICES = helper.write_people(PEOPLE, FILE_NAME, INDEX_FILE_NAME)

# This method is used to add a person.
def add_person(people_table, edit_person=None):
    # This method saves the person to the file.
    def save_person(edit):
        global INDICES, PEOPLE

        person_name = name_input.get()
        person_email = email_input.get()
        person_phone = phone_input.get()
        person_address = address_input.get("1.0", "end-1c")
        person_gender = gender_int.get()
        person_dob = dob_input.get()
        hash_string = (person_name + person_phone)
        person_id = hash.md5(hash_string.encode()).hexdigest()

        person = Person(person_id, person_name, person_email, person_phone, person_address, person_gender, person_dob,
                        edit_person.balance if edit_person is not None else 0)

        if person_name == "":
            dialog.showerror("Invalid Input", "Name cannot be empty.")
        elif not helper.isPhoneValid(person_phone):
            dialog.showerror("Invalid Input", "Invalid Phone Number.")
        elif not helper.isEmailValid(person_email):
            dialog.showerror("Invalid Input", "Invalid Email.")
        elif not helper.isDateValid(person_dob):
            dialog.showerror("Invalid Input", "Invalid Date of Birth.")
        elif edit:
            if edit_person.id != person_id and person_id in INDICES:
                dialog.showerror("Duplicate Entry", "Person already exists.")
                return
            result = dialog.askquestion("Confirm Changes", "Do you want to save changes?", icon='warning')
            if result == 'yes':
                for i in range(len(PEOPLE)):
                    if PEOPLE[i].id == edit_person.id:
                        PEOPLE[i] = person
                        break
                INDICES = helper.write_people(PEOPLE, FILE_NAME, INDEX_FILE_NAME)
                transaction.update_transactions(edit_person.id, person.id)
                PEOPLE.sort(key=lambda person: person.name)
                person_sub_window.destroy()
                helper.refresh_table(people_table, PEOPLE)
            else:
                for person_id in INDICES.keys():
                    if person_id == person.id:
                        dialog.showerror("Duplicate Entry", "Person already exists.")
                        break
                else:
                    with open(FILE_NAME, "ab") as file:

```



```

        pickle.dump(person, file)
        PEOPLE.append(person)
    INDICES = helper.write_people(PEOPLE, FILE_NAME, INDEX_FILE_NAME)
    PEOPLE.sort(key=lambda person: person.name)
    person_sub_window.destroy()
    helper.refresh_table(people_table, PEOPLE)

person_sub_window = Tk()
person_sub_window.title("Add Person")

bottom_frame = Frame(person_sub_window)
bottom_frame.pack(side=BOTTOM)

save_button = Button(bottom_frame, text="Save", command=lambda: save_person(edit_person is not None))
cancel_button = Button(bottom_frame, text="Cancel", command=person_sub_window.destroy)
cancel_button.pack(side=RIGHT)
save_button.pack(side=RIGHT)

top_frame = Frame(person_sub_window)
top_frame.pack(side=TOP)

name = Label(top_frame, text="Name")
name.grid(row=0, column=0)
name_input = Entry(top_frame)
name_input.insert(END, "" if edit_person is None else edit_person.name)
name_input.grid(row=0, column=1)

email = Label(top_frame, text="Email")
email.grid(row=1, column=0)
email_input = Entry(top_frame)
email_input.insert(END, "" if edit_person is None else edit_person.email)
email_input.grid(row=1, column=1)

phone = Label(top_frame, text="Phone")
phone.grid(row=2, column=0)
phone_input = Entry(top_frame)
phone_input.insert(END, "" if edit_person is None else edit_person.phone)
phone_input.grid(row=2, column=1)

address = Label(top_frame, text="Address")
address.grid(row=3, column=0)
address_input = Text(top_frame, width=50, height=4)
address_input.insert(END, "" if edit_person is None else edit_person.address)
address_input.grid(row=3, column=1)

gender = Label(top_frame, text="Gender")
gender.grid(row=4, column=0)
gender_int = IntVar(top_frame)

gender_frame = Frame(top_frame)
male_radio = Radiobutton(gender_frame, text="Male", value=0, variable=gender_int)
female_radio = Radiobutton(gender_frame, text="Female", value=1, variable=gender_int)
other_radio = Radiobutton(gender_frame, text="Other", value=2, variable=gender_int)

gender_value = 0 if edit_person is None else edit_person.gender
if gender_value == 0:
    male_radio.select()
elif gender_value == 1:
    female_radio.select()
else:
    other_radio.select()

male_radio.pack(side=LEFT)
female_radio.pack(side=LEFT)
other_radio.pack(side=LEFT)
gender_frame.grid(row=4, column=1)

dob = Label(top_frame, text="Date of Birth (DD/MM/YYYY)")
dob.grid(row=5, column=0)
dob_input = Entry(top_frame)
dob_input.insert(END, "" if edit_person is None else edit_person.dob)
dob_input.grid(row=5, column=1)

person_sub_window.mainloop()

```

CHAPTER 5

TESTING

5.1 Introduction

The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals, that is conducted tests to uncover errors and ensure that defined input will produce actual results that agree with required results. Broadly speaking, there are at least three levels of testing: unit testing, integration testing, and system testing.

5.2 Unit Testing

Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs as designed. A unit is the smallest testable part of any software. It usually has one or a few inputs and usually a single output.

Unit testing is commonly automated, but may still be performed manually. The objective in unit testing is to isolate a unit and validate its correctness. A manual approach to unit testing may employ a step-by-step instructional document. Unit testing is the process of testing the part of the program to verify whether the program is working correctly or not. In this part the main intention is to check each and every input which we are inserting to our file. Here the validation concepts are used to check whether the program is taking the inputs in the correct format or not.

Unit testing may reduce uncertainty in the units themselves and can be used in a bottom-up testing style approach. By testing the parts of a program first and then testing the sum of its parts, integration testing becomes much easier. Unit test cases embody characteristics that are critical to the success of the unit.

Table 4.1 Unit Testing for Event

| Test Case ID | Description | Input data | Expected Output | Actual Output | Status |
|--------------|------------------------------|--------------------------|------------------------------------|---|--------|
| 1 | Insertion of new transaction | Insert a new transaction | The transaction should be inserted | The transaction is inserted successfully. | Pass |

| | | | | | |
|---|-----------------------------------|------------|--|---|------|
| | | | successfully. | | |
| 2 | Retrieve a transaction by date | 12/09/2017 | The transaction should be retrieved successfully.. | The transaction is retrieved successfully. | Pass |
| 3 | Delete a transaction | 12 | The transaction should be deleted successfully.. | The transaction is deleted successfully. | Pass |
| 4 | Search for transactions by person | Anushka | The transactions should be searched successfully. | The transactions are searched successfully. | Pass |

This is just a sample tabular form for unit testing in ledger transaction maintenance system. The specific tests that you need to include will depend on the specific features of your system.

Unit tests for ledger transaction maintenance system:

Use a unit testing framework. There are many unit testing frameworks available, such as unit test and pytest. Using a unit testing framework, we could write and run the test cases easily Automated the tests. Once we have written the tests, we have automated them so that we could run them easily and quickly. Unit testing is an important part of software development. By unit testing your ledger transaction maintenance system, you can help to ensure that your system is working correctly and that it is free of bugs.

5.3 Integration Testing

Integration testing is also taken as integration and testing this is the major testing process where the units are combined and tested. Its main objective is to verify whether the major parts of the program are working fine or not. This testing can be done by choosing the options in the program and by giving suitable inputs it is tested.

Table 4.2 Integration Testing

| Test Case ID | Description | Input data | Expected Output | Actual Output | Status |
|--------------|--|-----------------|---|---|--------|
| 1 | Hashing function produces unique hash values | New Transaction | The hash values for all records should be unique. | The hash values for all records are unique. | Pass |
| 2 | B-trees | Search for | The B-trees | The B-trees | Pass |

| | | | | | |
|---|---|--|---|--|------|
| | are able to store and retrieve data efficiently | transaction with date 27/07/2017 | should be able to store and retrieve data in a timely manner. | are able to store and retrieve data in a timely manner. | |
| 3 | Transactions are able to access the data correctly | Search for transaction by date- 27/07/2017 | The transactions should be able to access the data correctly. | The transactions are able to access the data correctly. | Pass |
| 4 | System is able to handle a large volume of transactions | Adding 20 transactions | The system should be able to handle a large volume of transactions without performance degradation. | The system is able to handle a large volume of transactions without performance degradation. | Pass |
| 5 | System is able to handle invalid data | Phone number entered= 12345 | The system should be able to handle invalid data gracefully and not crash. | The system is able to handle invalid data gracefully and not crash. | Pass |

5.4 System Testing

System testing is defined as testing of a complete and fully integrated software product. This testing falls in black-box testing wherein knowledge of the inner design of the code is not a prerequisite and is done by the testing team. System testing is done after integration testing is complete. System testing should test functional and non-functional requirements of the software. And is implemented in below table 4.3

Table 4.3 System Testing

| Test Case ID | Description | Input data | Expected Output | Actual Output | Test Case ID |
|--------------|--|---------------------|--|---|--------------|
| 1 | The system is able to start up and run | Running the program | The system should start up without errors and be able to run without crashing. | The system is starting up without errors and be able to run | 1 |

| | correct ly. | | | without crashing. | |
|---|---|---------------------------------|---|---|---|
| 2 | The syste m is able to handle invalid data gracef ully. | Email entered= sanjana123 | The system should be able to handle invalid data gracefully and not crash. | The system is able to handle invalid data gracefully and not crash. | 2 |

CHAPTER 6

RESULTS

6.1 Discussions of results

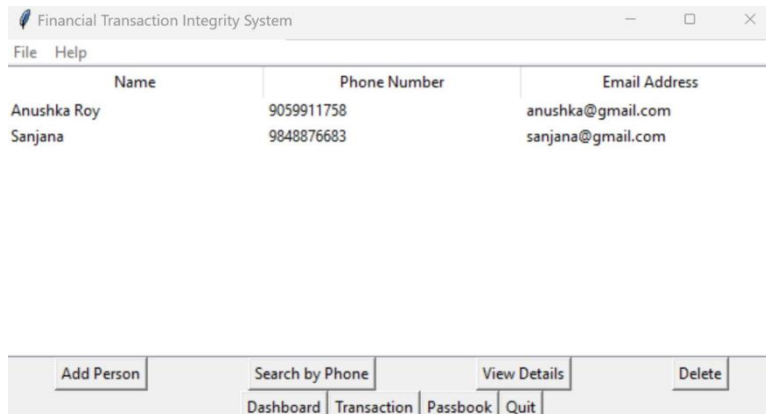


Figure 6.1 Main menu

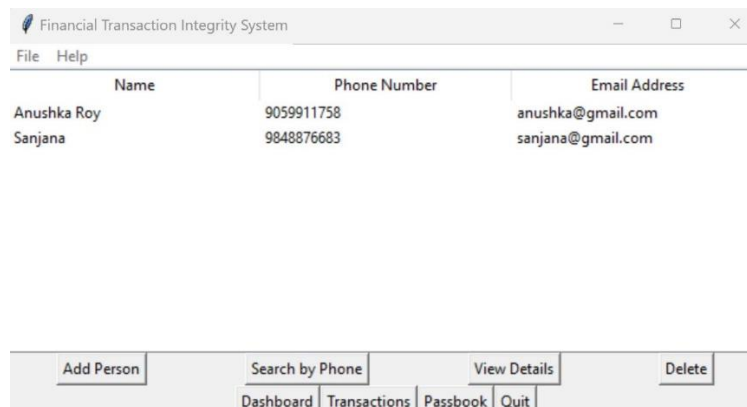


Figure 6.2 Different Profiles added on the system

Different profiles can be added to a ledger transaction maintenance system to allow different users to access and modify different sets of data.

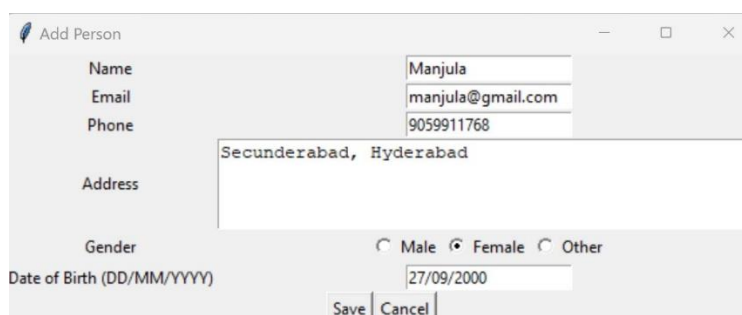


Figure 6.3 Adding a profile

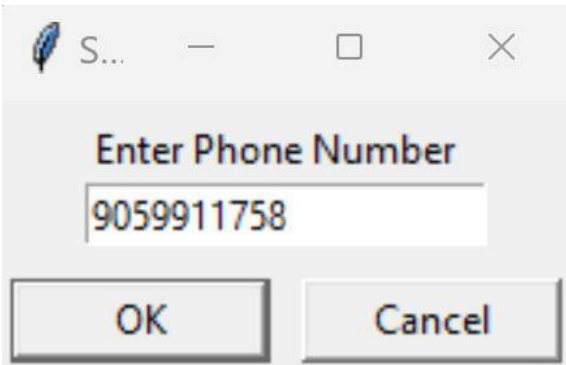


Figure 6.4 Searching a person and their transaction by their phone number

Searching a person and their transactions by their phone number on the ledger transaction maintenance system can be a useful tool for businesses and organizations. It can help you to track customer activity, investigate fraud, resolve customer disputes, and comply with regulations.

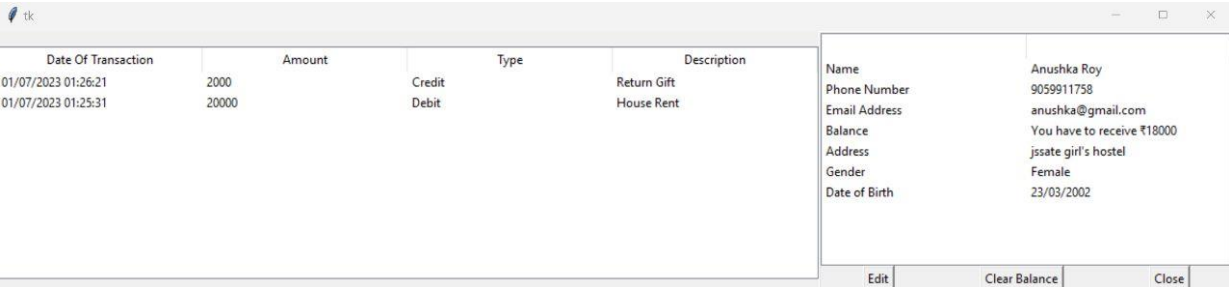


Figure 6.5 Result of searching a person by their phone number

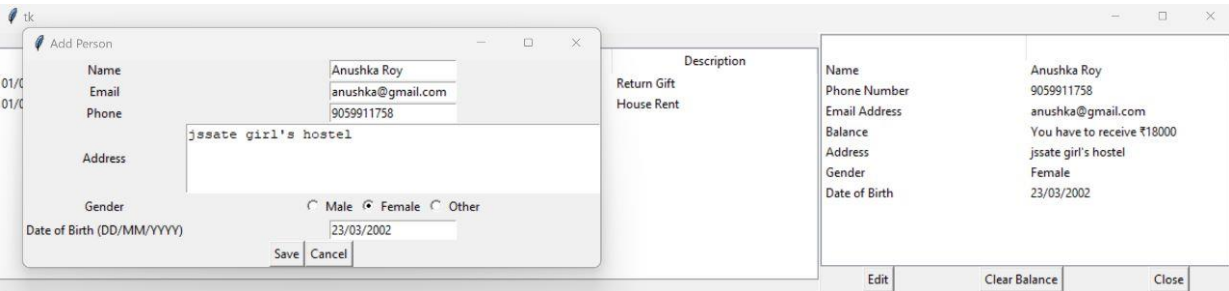


Figure 6.6 editing a profile

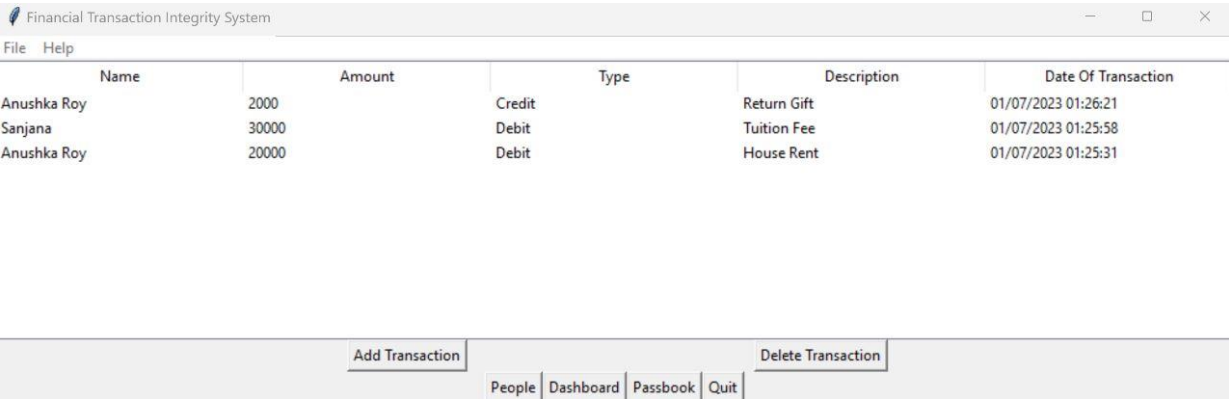


Figure 6.7 transactions

Add Transaction

| Name | Phone |
|-------------|------------|
| Anushka Roy | 9059911758 |
| Manjula | 9059911768 |
| Sanjana | 9848876683 |

Amount (in Rs.)

20000

Transaction Type

☐ Give

☒ Receive

Description

lent money to friend

Add Transaction

Cancel

Figure 6.8 Adding transactions of a person

Financial Transactio...

File

Help

☒ Last N Transactions

☐ By Person

☐ By Date

Last Transaction

Generate

People

Transaction

Dashboard

Quit

Figure 6.9 Passbook dashboard

Financial Transaction Integrity System

File

Help

☐ Last N Transactions

☒ By Person

☐ By Date

| Name | Phone |
|-------------|------------|
| Anushka Roy | 9059911758 |
| Manjula | 9059911768 |
| Sanjana | 9848876683 |

Generate

People

Transaction

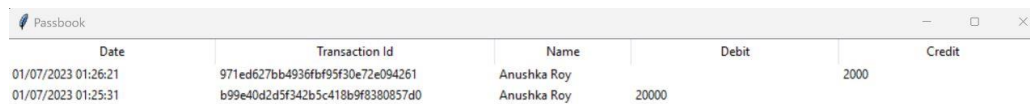
Dashboard

Quit

Figure 6.10 Generating all the transactions of a person

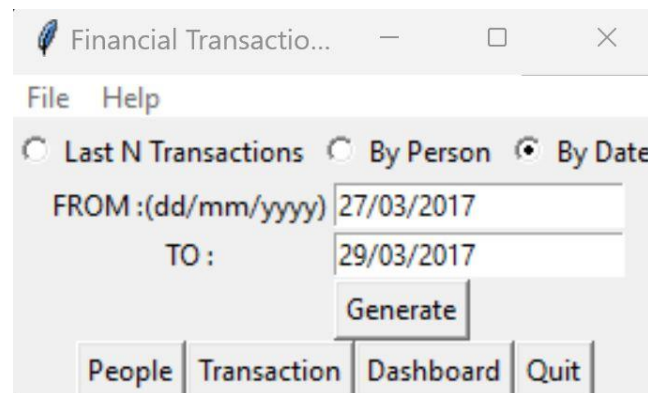
Examples of how generating transactions by date on the ledger transaction maintenance system can be useful:

- A business can use this feature to track its sales and expenses daily. This information can be used to identify trends in sales and expenses, and to make informed business decisions.
- A bank can use this feature to track its customer activity daily. This information can be used to identify fraudulent activity and to prevent money laundering.
- A government agency can use this feature to track its tax payments daily. This information can be used to enforce tax laws and to investigate financial crimes.
- Overall, generating transactions by date on the ledger transaction maintenance system can be a valuable tool for businesses, organizations, and government agencies.



| Date | Transaction Id | Name | Debit | Credit |
|---------------------|----------------------------------|-------------|-------|--------|
| 01/07/2023 01:26:21 | 971ed627bb4936fbf95f30e72e094261 | Anushka Roy | | 2000 |
| 01/07/2023 01:25:31 | b99e40d2d5f342b5c418b9f8380857d0 | Anushka Roy | 20000 | |

Figure 6.11 result of generating transactions of a person



Financial Transaction...

File Help

☐ Last N Transactions
 ☐ By Person
 ☒ By Date

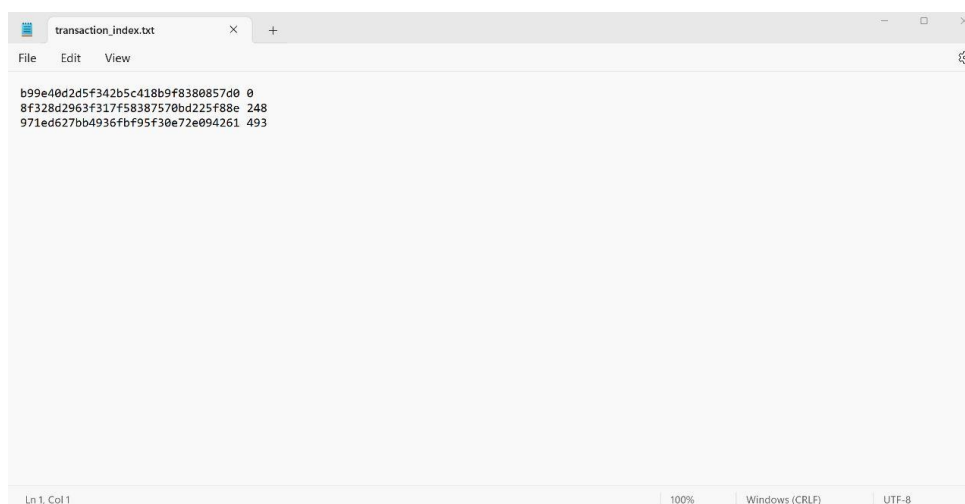
FROM : (dd/mm/yyyy) 27/03/2017

TO : 29/03/2017

Generate

People Transaction Dashboard Quit

Figure 6.12 generating transactions by date



```

b99e40d2d5f342b5c418b9f8380857d0 0
8f328d2963f317f58387570bd225f88e 248
971ed627bb4936fbf95f30e72e094261 493
  
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

Figure 6.13 transaction index text file

CHAPTER 7

CONCLUSION

The Ledger Transaction Maintenance system that incorporates hashing and B-tree indexing file structure offers numerous advantages for managing and organizing transaction data. The combination of these techniques provides fast and efficient retrieval, ensures data integrity, optimizes storage space, supports scalability, and enhances system security.

By using hashing, each transaction is assigned a unique identifier (hash), allowing for quick and constant-time access to transactions. This ensures fast retrieval even when dealing with a large volume of data. Additionally, the use of hashing provides a means of verifying data integrity, as any modifications to a transaction would result in a different hash value, making tampering easily detectable.

The integration of B-tree indexing further enhances the system's performance and efficiency. B-trees allow for efficient searching, insertion, and deletion operations, making it easy to locate transactions based on various criteria such as transaction ID or date. The balanced structure of B-trees optimizes storage space and reduces access time, resulting in improved overall system performance.

The ledger transaction maintenance system utilizing hashing and B-tree indexing file structure provides a robust and efficient solution for managing and maintaining ledger transactions. It offers fast retrieval, ensures data integrity, optimizes storage space, supports scalability, and enhances system security, making it an ideal choice for a wide range of applications.

CHAPTER 8

REFERENCES

- [1] "Database Management Systems" by Raghu Ramakrishnan and Johannes Gehrke
- [2] "File Structures: An Analytic Approach" by Michael J. Folk, Bill Zoellick, and Greg Riccard
- [3] "File Organization and Processing" by Alan L. Tharp
- [4] "Advanced File Structures" by S. B. Kulkarni and M. S. Gadage
- [5] "File Organization and Processing" by Alan L. Tharp
- [6] "Python Crash Course: A Hands-On, Project-Based Introduction to Programming"
- [7] www.geeksforgeeks.com
- [8] www.baeldung.com
- [9] www.medium.com
- [10] www.gov.mb.ca