# Record Keeper Project

## 1. Introduction

This document provides an overview and a technical introduction to the Record Keeper project, that has been proposed as part of the assignment 1, part 1. It provides detailed features, diagram and software design of the project. It serves both as an introductory as well as a technical document for the Record Keepr Project.

The project merges a phone book, contact book and a record book for keeping interests and related resources. It does this by providing a generic record unit, which can store a phone entry, a contact entry or a genereal entry with the name and associated note. As the project is an integrated version of the phone book, contact book and a general record book, the name Record Keep (RecordKeeper) has been selelect to represent a generic name.

The scope of the project is limited to creation, deletion, searching of the record unit. The project is implemented as a command line interface (CLI) and there is no graphical user interface (GUI).

The goal and purpose of the project is to provide users with an easy to use record keeping application where they can record their friends and family phone numbers, contacts and also general notes about anything. The project provides provides quick searching facility based on binary search and any phone num, contact and note can easily be search using a word.

The project also provides persistence storage, the user records are saved when the program is exited and loaded when the program is started.

Following are the main objectives of the project.

### 1.1 Objectives of the Project:

* To provide an easy to use facility to store, search and retrive phone/contact/note records.
* To provide users with a command line interface to easily access the application features.
* To provide users with a document introducing the project features, design and other technical details so the project can be extended in the future.
* To learn to design and implement a basic working application of record keeping using the python language. Learning includes usage of dict, list and class data structures and algorithms for addition, deletion, and searching.
* To learn how to test the project features using unit testing which has been implemented in the project.
* To implement permanent storage of the data structures using python pick and to learn about serialization/deserialization

### 1.2 Project Execution

The project application has been named as “RecordKeeper” and can be easily executed in the command line of windows and Linux terminal.

**Pre-requisite:**

An operating system with Python Installed. The project requires Python 3 to work correctly.

Sufficient permissions to run the project and file read/write permissions.

**Running the application:**

1. Run the CLI
2. Unzip the application zip file “recordkeeper.zip”
3. cd “recordkeeper”
4. write “python3 main.py” to run the application

## 2. Data Structure and Algorithms Design

### 2.1 Data Structures

The following data structures have been used to implement the RecordKeeper project.

**HashMap**

A HashMap can stores objects in key, value pairs, it takes input as key and can return the object. We will be using Python Dict which is implemented as a HashMap. It is an efficient and quick way to store and retrieve user data type objects with big O complexity of 1.

In the application, dict has been used to store records. A record can be of a phone record, contact record or a note record.

The hashmap key is the name/tag of the record and value is the record object. In the case of phone and contact, the key is the name of the person and in the case of a note, the key is the tag/title of the note.

If we know the name/key of the person, we can retrive the record in O(1).

The dict structure has four attributes.

|  |  |
| --- | --- |
| **Dict Attribute** | **Description** |
| Name | The name of the person/contact/record |
| Address | The address of the person (optional) |
| Phone | The phone number of the contact (optional) |
| Misc | Any other details can be stored here (optional) |

**Reason:** The main reason hashmap/dict was used as the access to any attribute has a complexity of O(1).

**List Data structure**

All the phone book/contact records will be stored in the list. Each item in the list would be the Dict structure which will represent different attributes like name, address,phone, misc. When a new record is created, the list will be appended with the new record, and when a record is deleted, the record would be deleted from the list. Similarly for search an item in the list, the list would be traversed to find the required item.

**Reason:** Acts as a container for the hashmap/dict records. The whole list can be serialized to bytes and written to the disk using the pickle library.

**Serialization and Deserialization**

Serialization is the process of converting python data structures directly into byte streams so they can be written in a file and deserialization is reverse process of serialization which will convert the file back to the data structures.

For this project, serialization has been used to convert the list to bytes and written to a disk file using the library **“pickle”.** Pickly provides an easy to use functionality in Python to write data structures to disk and then load them back.

The main advantage of serialization and deserialization is to provide a convenient byte stream structure which can then be used for permanent storage, so the contents of data structures would not be deleted after the program has been closed.

**2.2 Algorithms**

**Insertion Strategy**

For insertion into the list, we follow an algorithm which is given below.

The algorithm takes list and record as input parameters, then if the list is empty, inserts the record into the list straight away. If the list is not empty, then it searches for the correct position in the list for insertion using binary search, as we assume the records are sorted, and therefore insertions have to be sorted as well for efficient operation. Returns 1 on success, and 0 on error.

**Algo**

**Input:**

list: list of dict/hashmap

record: record(hashmap/dict) to insert

output: returns 1 after successful insertion

If (list.size==0){

list.insert(record)

}else {

#find the correct sorted position in the list

listItem=binarySearch(list,record)

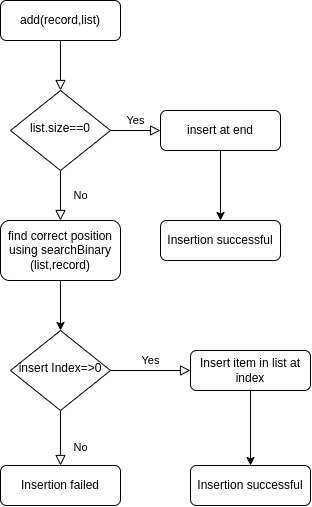
list.insertAfter(listItem,record)

#This will insert the item at the correct location in the list so the list remains sorted

return 1

}

Since insertion is the most complex algorith, a flow chart is also provided for convenience



**Deletion Strategy:**

For deletion from the list, we follow an algorithm which is given below. For deletion, basically we first use the binary search to search for the record quickly, and then use the list method remove to remove the record. Returns 1 on successful deletion and 0 if there is an error or the item couldn’t be found.

**Algo**

**Input:**

list: list of dict/hashmap

record: record(hashmap/dict) to be deleted from the list

output: returns 1 after successful insertion or 0 in case of error

If (list.size==0){

return 0

}else {

#find the correct sorted position in the list

listItem=binarySearch(list,record)

list.deleteItem(record)

#This will delete the item in a quick way, as we used binary search to find the item, making use of the fat that list is sorted

}

**Sorting Algorithm**

There is no need of a separate sorting algorithm for the list, as the items are sorted when they are inserted into the list. However, we could have used a quick sort if the items were not sorted.

**Searching Algorithm**

Binary search has been selected and implemented as the searching algorithm as it perform a quick search if the list is sorted. The worst case big o complexity of binary search is O(log N), which is better and more efficient than performing a linear search.

The pseudocode for binary search algorithm is given below and is used by insertion, deletion algorithms to quick search the item in the list. Firstly if the list is empty, then simple append the record to the list, however if the list is not empty, then insert the record into the list at its correct sorted position. The sorted position is obtained by performing a binary search on the list items. It is to be noted that the list is sorted when items are inserted into the list.

**Binary Search Algorithm**

**searchBinary(list, item)**

**input:**

list: list of items

item: item(dict) to look for

#find the mid point of list

mid=list.size/2

#find the mid Item

item=list.get(mid)

name=item.name

if (compareString(name,item)==0){

#if the result of comparision is 0, it means name and item are the same object, so we can simply return list item

return item

}

if (compareString(name,item)>0){

#if the name is greater than the item, it means the name is on the right side of the list

#binary search is a recursive algorithm, we need to give the binary search again the list from point mid to end and the item to search so

rightList=list.subset(mid, list.size)

return searchbinary(rightList,item)

}else{

#the name is on the left side of the item

leftList=list.subset(mid, list.size)

return searchbinary(leftList,item)

}

**Test Plan**

Testing is fundamental to developing a successful project. For testing, unit testing and black box testing were used. For each main feature of the project, a unit test was written and validated for input/output. The output of a unit test is either pass or fail, which indicates if the functionality has been validated successfully or not. The overal behavior of the project was tested using manual black box testing.

Following features were testing using unit testing as shown in the table below

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Unit Test Name | Test DataLamp  plugged in? | Expected output | Description |
| 1 | TestInsertRecord() | 1. Name: John  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data  2. Name: Oliver  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data  2. Name: Zee  Address: 15 London Street, LE7 88D  Phone: TestPhone  Misc: This is a test data | 1. Output: 1  2. Output: 1  3. Output: 0 | A method to test the functionality of the input data, two data points are provided, both should return 1 which means the insertion has been successful  . For 3) The input details can’t be validated as the phone number format is invalid, so the insertion fails |
| 2 | TestDeleteRecord() | Name: Oliver  Name: Zee | 1. Output: 1  2. Output: 0(record doesn’t exist) | To test the functionality of deleting a record with the two test data points |
| 3 | TestSearchRecord() | 1. Name: Oliver  2. Name: Zee | 1: Output: Record of Oliver  2. 0 | 1. Item exists and should return the complete dictionary object.  2. Item doesn’t exist, so will return 0 |
| 4 | TestPhoneFormat | 1. Phone: 183730  2. Phone: XDF888 | 1. Output: 1  2. Output: 0 | The phone number must consist of digitals only, test 2) should fail as the phone format is invalid and test 1 should pass. |

**Project Structure**

The project is implemented in python and functionality has been encapsulated in classes. A brief explanation of the project structure and classes is given below

main.py: Main File name, consist of all the project classes including recordkeeper

RecordKeepr: This class has all the logic, algorithms and data structures to provide functionality of insertion/deletion/searching of the records.

TestRecordKeepr: This class is used for testing the individual functionality of the RecordKeeper. It has functions such as testDeleteRecord, testInsertRecord etc. It is based on python UnitTest module.

Main: This class serves as an entry point to the program, it create instances of recordkeeper and also provides menu functionality to the user. This is the first class which is executed when the program runs.

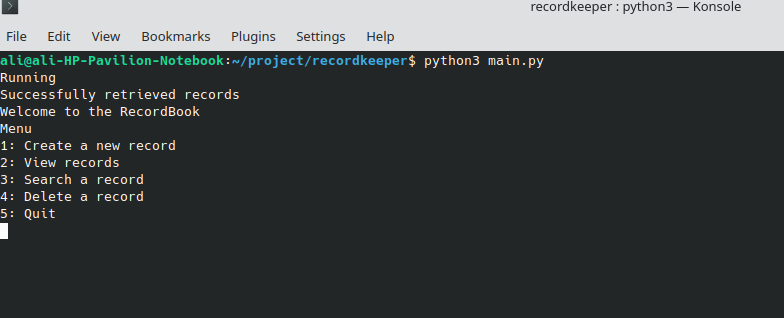
**Limitation:**

The List structure each item is a dict object which is stored in a file. When the program starts, the file is loaded by the pickle library and stored back to the disk overwriting the whole content of the file, although it might not be a problem for a phone record/contact record application, however it is an inefficient way to manage the persistence storage. A better way would be to utilize the database such as MySQL or even SQLLite to provide persistence storage.

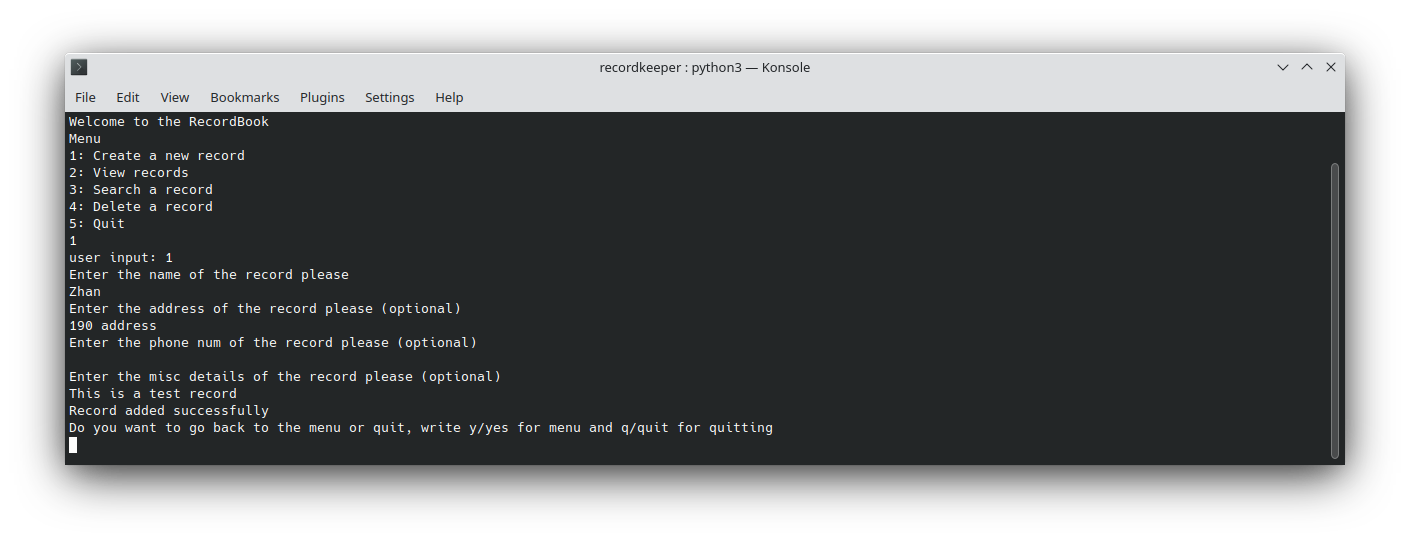
Screenshots:

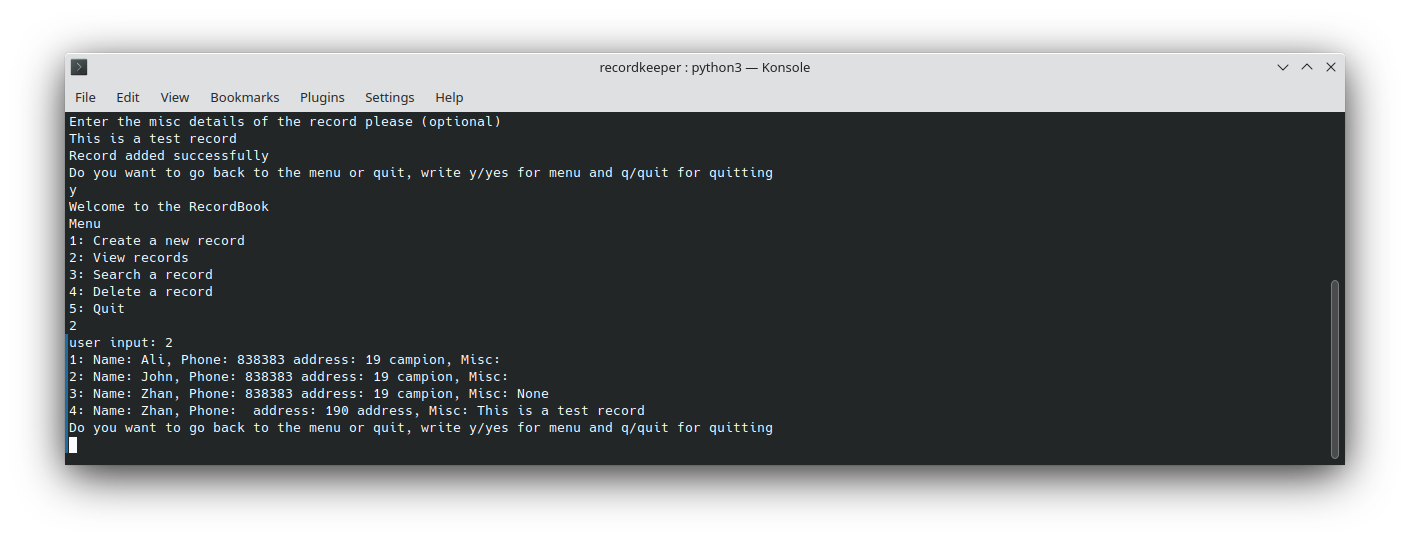
The main screenshot of the project

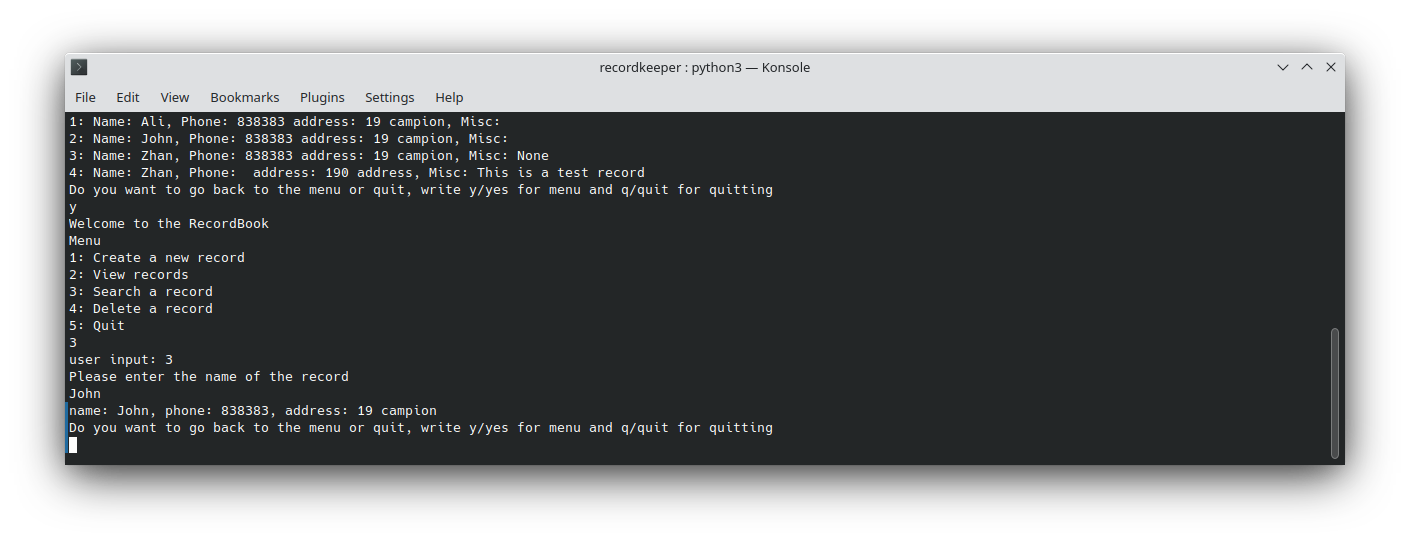
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Unit Test Name | Test DataLamp  plugged in? | Expected output | Description |
| 1 | TestInsertRecord() | 1. Name: John  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data  2. Name: Oliver  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data  3. Name: Zee  Address: 15 London Street, LE7 88D  Phone: TestPhone  Misc: This is a test data | 1. Output: 1  2. Output: 1  3. Output: 0 | A method to test the functionality of the input data, two data points are provided, both should return 1 which means the insertion has been successful  . For 3) The input details can’t be validated as the phone number format is invalid, so the insertion fails |
| 2 | TestDeleteRecord() | Name: Oliver  Name: Zee | 1. Output: 1  2. Output: 0(record doesn’t exist) | To test the functionality of deleting a record with the two test data points |
| 3 | TestSearchRecord() | 1. Name: Oliver  2. Name: Zee | 1: Output: Record of Oliver  2. 0 | 1. Item exists and should return the complete dictionary object.  2. Item doesn’t exist, so will return 0 |
| 4 | TestPhoneFormat | 1. Phone: 183730  2. Phone: XDF888 | 1. Output: 1  2. Output: 0 | The phone number must consist of digitals only, test 2) should fail as the phone format is invalid and test 1 should pass. |



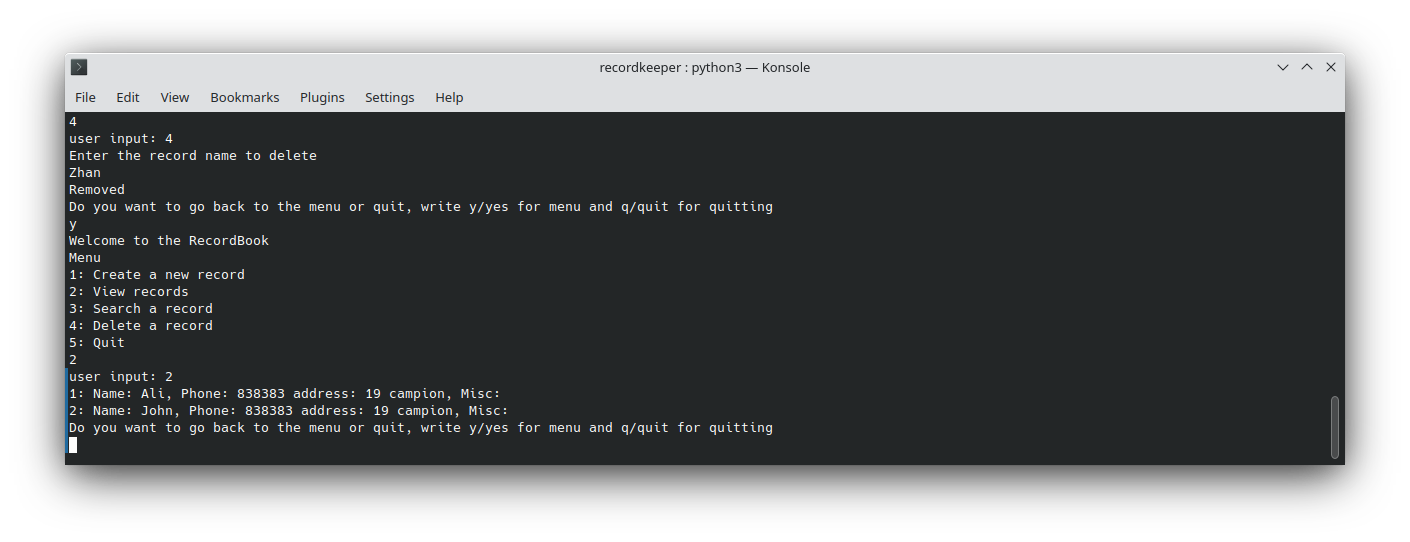
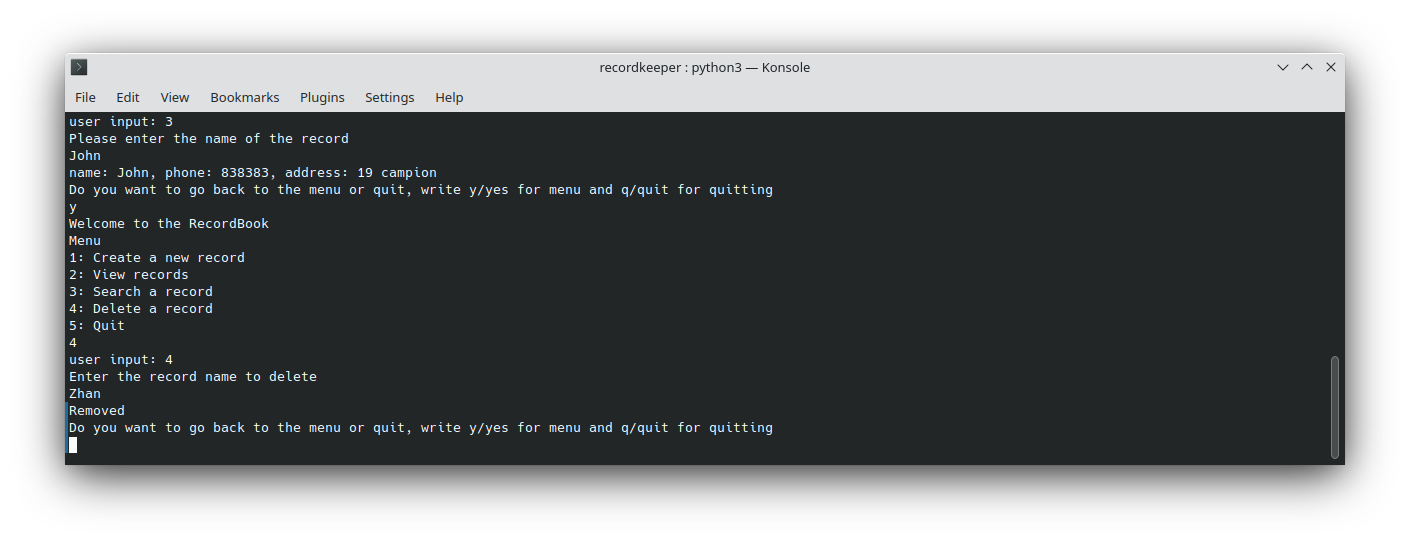
To create a new record

To view the records

To search for a record named “John”



To delete a record name Zhan

To verify the record has been deleted

**Assignment 1 Part 2/3**

**Testing Strategy**

In the first part, test cases were used to test the core functionality of the record keeper project. However, the part two includes GUI over the existing core functionality. So in this part, testing strategy focuses on testing the GUI functionality testing and execution. The GUI testing is also known as user interface (UI) testing. Two types of testing were performed which are given below

1) Verification and Validation testing

2) White box testing

**Verification and Validation Testing**

In this testing, all the required features of the project was tested and implemented. In below table, each feature is listed with its implementation status

|  |  |  |  |
| --- | --- | --- | --- |
| Num | Feature | Sub-Feature Tested | Implemented/Working? |
| 1 | Add Record | Enter a record with name, phone, address, misc | Yes |
|  |  | Enter 5 records | Yes, also tested using unit test case testAddFiveRecords |
| 2 | Delete Record | Allows deletion of record | Yes |
| 3 | Viewing/Sorting | Allows viewing of all records in a table | Yes |
|  |  | Allows sorting using name, phone,address and misc | Yes |
| 4 | Option Selection GUI |  | Yes |
| 5 | Prompt on screen before adding/deleting records |  | Yes |
| 6 | Unit Test data verification |  | Yes |
| 7 | Binary Search Algo Testing |  | Yes |

**White box testing**

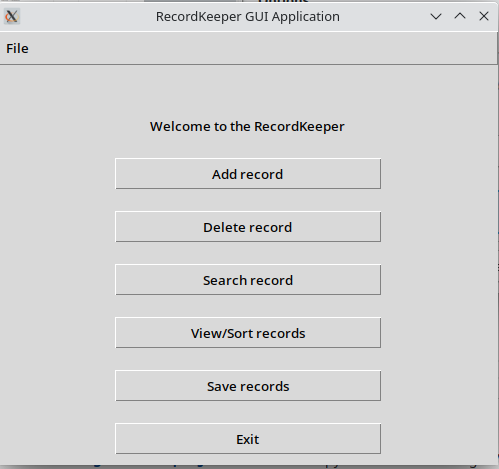
In this testing, individual features were tested using unit tests, most of the unit tests have been written as part of the part 1 Test plan in a table. These tests are re-executed to make sure no accidental changes made due to part 2 have changed the internal working of the features. All the testing has been passed.

In addition to the four unit tests given in part 1, here we only have one new test case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Unit Test Name | Test Data | Expected output | Description |
| 1 | TestAddFiveRecords | 1. Name: John  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data  2. Name: Zee  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data  3. Name: Ali  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data  4. Name: Jackie  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data  5. Name: Josh  Address: 15 London Street, LE7 88D  Phone: 188  Misc: This is a test data | 1. Output: 1  2. Output: 1  3. Output: 1  4. Output: 1  5. Output: 1 | To test the insertion of five records |

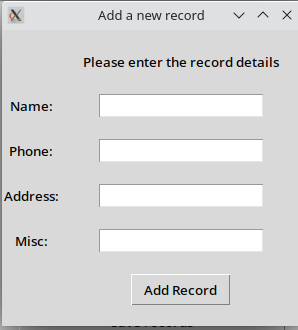
**Testing Screenshots:**

**Main GUI Application**

****

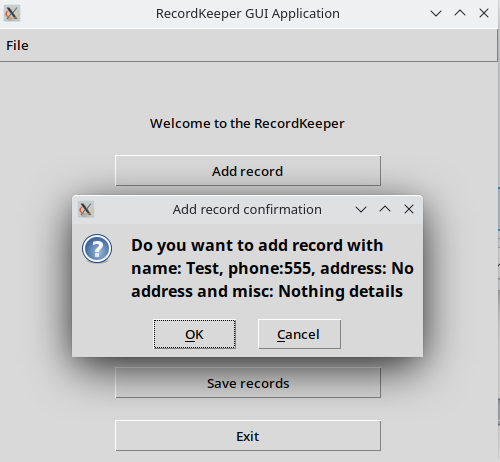
**Add a new record Frame**

Allows to add a new record graphically by providing the input data.

****

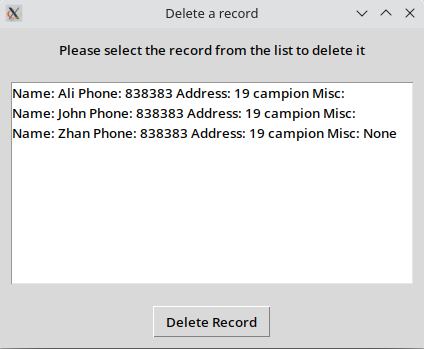
**Record Adding Action Validation**

Validates the user action, before adding the record.

****

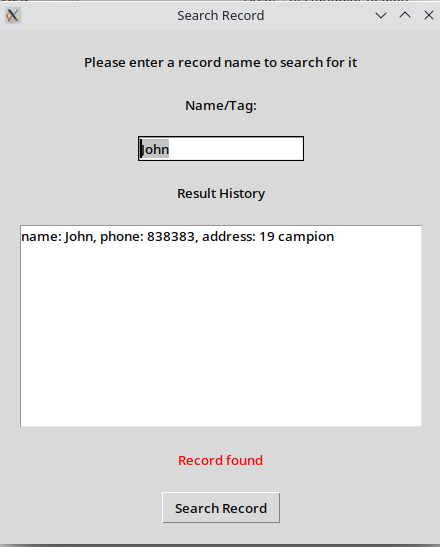
**Delete a record**

Select a record to delete it

****

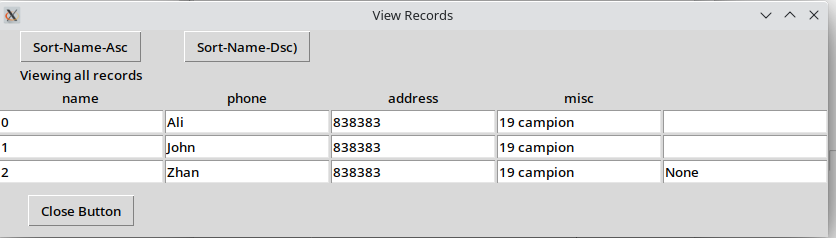
**Search for a record**

To search for a record, enter the record name/tag in the input field

****

**View/Sort Records**

Shows all the records, can be sorted in ascending and descending order.

****