A Detailed Project Report

On

Phonebook by TRIE Data structure

Submitted to

University of Petroleum and Energy Studies

In Accordance With The Award Of The Degree Of

BACHELORS IN TECHNOLOGY(NON-HONS.)

In

COMPUTER SCIENCE AND ENGINEERING

(With Major in cyber security and Forensics)

By

Name	Arpita Kumari	Arihant Vardhan	Tanushpreet Kaur	Vikash Kumar
Roll No	R2142201596	R2142201523	R2142201687	R2142201909

Under the guidance of

Mr. Keshav Kaushik

Assistant Professor – Senior Scale

SOCS

University of Petroleum and Energy Studies

Dehradun-India

December 2022

CANDIDATE'S DECLARATION

We hereby certify that the project entitled "**Implementation of Phone book Directory using TRIE data structure**" submitted to the Department of Systemic at the , is **School of Computer Science, University of Petroleum & Energy Studies, Dehradun** an authentic record of our work and satisfies the requirements for the award of the degree of BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE AND ENGINEERING with a focus on CYBER SECURITY AND FORENSICS. An authentic record of our work completed from August 2022 to December 2022 under the direction of **Mr. Keshav Kaushik**, Assistant Professor - Senior Scale, School of cyber security, has been submitted to the Department of Systemic.

We have not submitted the subject matter covered in this project for the granting of any other degree from this or any other University. This is to confirm that, to the best of my knowledge, the candidate's above statement is accurate.

Date: 07 December 2022

Mr. Keshav Kaushik

Project Guide

Dr. Neelu J. Ahuja

Head – Department of Systemics

School of Computer Science
University of Petroleum & Energy
Studies, Dehradun – 248 001
(Uttarakhand)

ACKNOWLEDGEMENT

We would like to extend our sincere thanks to our guide, **Mr. Keshav Kaushik**, for all the guidance, inspiration, and unwavering support he provided us with during the course of our project work. Without his encouragement and helpful recommendations, this effort would not have been feasible.

We would like to express our heartfelt gratitude to **Dr. Neelu J. Ahuja**, Head of the Department, for her exceptional assistance with our project, **Implementation of Phone Book Directory Using TRIE Data Structure at SOCS**.

We are also grateful to **Prof. S. Ravi Shankar**, the **School of Computer Science Dean at UPES**, who gave us the resources we need to successfully finish our project work.

We would like to thank all of our friends for their support and constructive feedback during the course of our project effort. Finally, we can only express our sincere gratitude to our parents for introducing us to the outside world and for all of the assistance they have provided.

Table of Contents

List Of Figures

Mentor Meet Report

S.NO	DATE	DISCUSSION
1	24 Aug 2022	Finalized topic for Minor project -1 with mentor.
2	7 Sep 2022	Researched about various data structure algorithm to choose best algorithm
3	23 Sep 2022	Synopsis presentation
4	2 Nov 2022	Documented SRS and report for mid-sem presentation
5	25 Nov 2022	Mid sem Presentation
6	28 Nov 2022	Analysing mistakes focused by mid-sem evaluation
7	5 Dec 2022	Documenting the Report file
8	6 Dec 2022	Adding and editing Final features
9	7 Dec 2022	Final Documentation

1. Abstract

A simple web application called Phone Book System was built. In the past, we kept all of our crucial contact information in books and papers. Here, we offered a novel method that allows us to keep all the information in one place utilizing an application. It is highly challenging to find the contact information using the manual way if we lose the information book. We may effortlessly view our contacts by utilizing this app. We may save our contacts in this project, look them up by name, and view all of their information at once.

The typical requirements of the user when using the phone directory book are considered in order to design this system. The user will have the access to add, search, and view existing records in the phone book directory in order to keep it up to date. The user will be able to search for a certain term and view all of the entries that match it.

The influence of storage and data structure on data retrieval is stronger. There are several implementations that can store and process data effectively. The type of application and the manner in which data are handled determine the proper structure to use. For data searches, the key is crucial. The majority of the time, implementation searches are done using the whole key value. A data structure called the TRIE Data Structure allows for storing and searching based on the individual characters or numbers that make up a key.

Because it is a quicker method, we sorted the names in the contact list using the TRIE Data Structure. The concept of a tree is the basis of the TRIE Data structure. A TRIE Data Structure is created by adding to the basic tree data structure. A value member and a pointer to the node's children are typically present at each node of the tree.

Keywords: Phonebook Directory, TRIE Data Structure and Java.

2. Introduction

It is often known as a phone book and the white-yellow pages, is a list of telephone subscribers in a particular region or subscribers to the services offered by the company that publishes the directory. Its goal is to make it possible to locate a subscriber's phone number after identifying them by name and address.[1]

It's console program. It resembles the contact manager on mobile devices. We have the ability to add, view, edit, search, and remove contacts in this project. We may list contacts using this program by name, phone number, address, and email. To implement, we used the TRIE Data Structure.

The collection of strings is kept in a data structure called TRIE Data Structure, which is based on sorted trees. Each node has an equal amount of references as there are letters in the alphabet. It may use the prefix of a word to search the dictionary for that term.[2][3][4] Each TRIE node can contain a maximum of 26 points, for instance, if we suppose that all strings are constructed from the letters "a" through "z" in the English alphabet. A TRIE has several advantages over binary search trees and other data structures. A hash table may be replaced by this.

Each node's position in the tree defines the key that each node is related with, rather than storing it like a binary search tree does. All of a node's descendants have the same prefix of that string, which is connected with the empty string at the node's root. There may not always be a value associated with each node. Values are often only connected to a few inner nodes that are related to leaves and significant keys.[2][3][4]

A TRIE tree uses several keys to prevent key collisions as well. There is no need to provide a hash function or modify hash functions when new keys are added to a TRIE. An effective data structure for information retrieval is TRIE. Search complexity can be reduced to the ideal level using TRIE (key length). A well-balanced BST will need time proportional to M * log N if we store keys in a binary search tree, where M is the maximum string length and N is the number of keys in the tree.[2][3][4]

The most basic computing task in Phonebook is searching. The influence of storage and data structure on data retrieval is stronger. There are several implementations that can store and process data effectively. A data structure called a TRIE performs finding and storing depending on individual characters or numbers that make up a key. The storing of homophones is one of the numerous uses of the TRIE Data Structure where effective searching is the primary requirement.[5]

Whether two words are lexically equal or not, use the simple and fundamental word operation. There may be numerous ways to assess whether two words are equal. Identifying two words based on their pronunciation requires more work than just checking characters one by one. By using a set of rules on the provided word, the problem can be solved.[6][8]

2.1. Problem Statement

Today, everyone has thousands of contacts and maintaining all the contact details is tough. This has resulted in the need for digital Phonebooks. Phonebook enables users to store new contacts, update new contacts and delete contacts. It provides facilities to store the details of employees and other persons that are associated with an organization.

2.2. Objectives

- Data processing at a high speed.
- To keep track of the user's numerous actions in detail.
- It will make the job easier and require less paperwork.
- Because the project will be entirely created at the administrative end, access is ensured to just the administrator.
- Application of OOPS principles.

2.3. Methodology

A) GUI

Java AWT was utilized for the GUI. Platform dependence means that Java AWT components are shown in line with the operating system's view. Resources from the underlying operating system are used by AWT's components since it is a resource-intensive framework (OS).

The java.awt package offers classes for the AWT API, including TextField, Label, TextArea, RadioButton, CheckBox, Choice, List, etc.

Also we used Swing. A lightweight Java GUI called Swing is employed to construct a variety of apps. Platform-independent components are found in Swing. It allows for the creation of buttons and scroll bars. Packages for building Java desktop apps are part of Swing. Java is used to write Swing components. The Java Foundation Classes include it.

B)Backend

TRIE Node was used to create Directory.java and TRIEnode.java and to merge these service module was created named Application Service and Application Service Implementation. To run the project appdriver.java was used . A CSV file was generated from where data was retrieved using TRIE searching algorithm. A TRIE Data Structure is a discrete data structure that is significant but not particularly well-known or frequently covered in standard algorithm courses.

A TRIE, often referred to as a digital tree or prefix tree (since they may be searched by prefixes), is an ordered tree structure that makes use of the keys that it contains, which are typically strings. In contrast to binary search trees, where each node stores a key that only corresponds to that node, tries define the key with which each node is associated based on its position in the tree.

C) Use Case

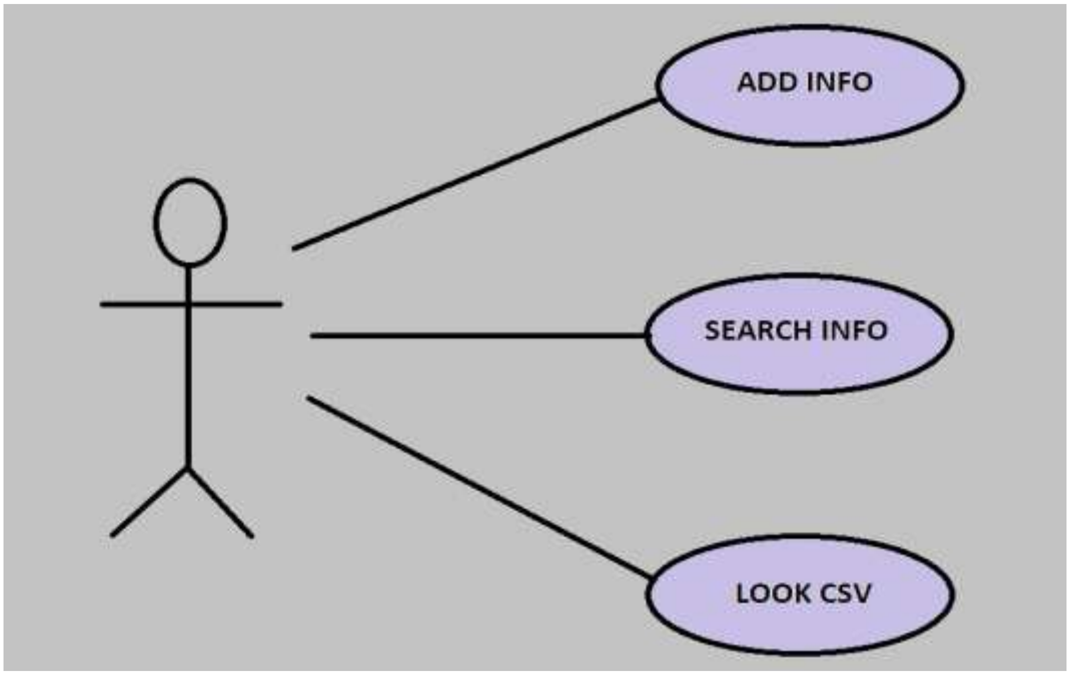


Fig.1: Use Case Diagram

3. Algorithm:

Implementation of TRIE:

1. The TRIENode() constructor is used to create the root node.
2. Array of strings to be inserted into the TRIE
3. The insertIntoTRIE() function inserts all strings into TRIE
4. Searching can be done with queryResults().

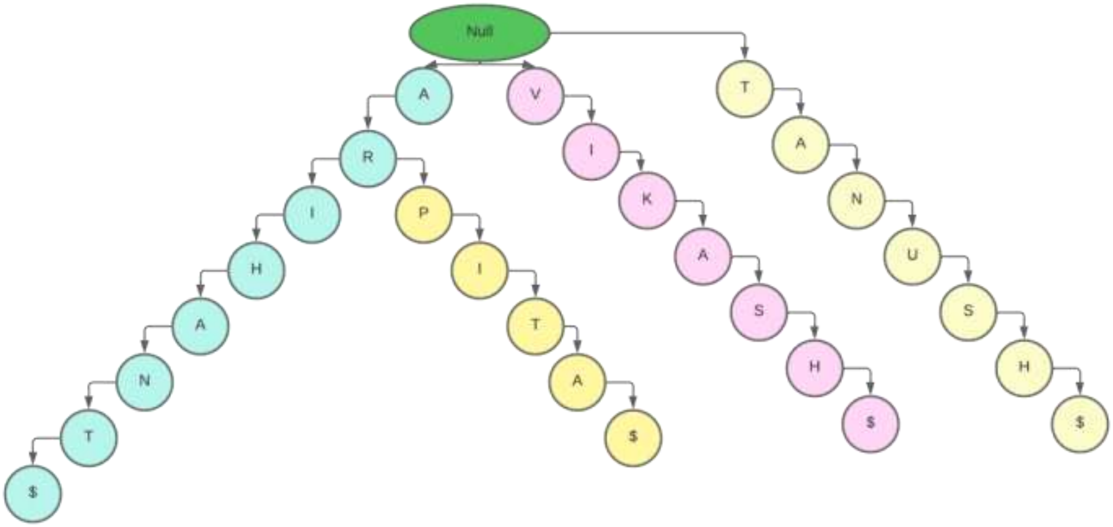


Fig.2: Sort Name Using TRIE

#TrieNode

STEP1: Declare TrieNode() class:
STEP2: Inside TrieNode() class class declare following public variables (public HashMap<Character, TrieNode> child;public Boolean isLast;public String fullName;public String companyName;public String email;public String phone;)
STEP3: Create a TrieNode Constructor:

```
  public TrieNode() {
     this.child = new HashMap<>();
     this.isLast = false;
  }
```

#insertIntoTrie

```
STEP1: Declare a method insertIntoTrie
STEP2: Inside the method declare a variable int length;
STEP3: Initiliaze length to size of directorylist.
STEP4: Now, for (int i = 0; i < length; i++) {
            insertContact(directoryList.get(i));
        }
```

# insertContact

```
#insertContact
STEP1: Declare a method insertContact
STEP2: Inside the method Initiliaze TrieNode temp = root;
STEP3: Declare a variable char[] ch and Initiliaze it with fullName as Character array;
STEP4: Now, Run a loop for each Character of fullName:
    for (int i = 0; i < ch.length; i++) {//loop start
            TrieNode dir = temp.child.get(ch[i]);//sets dir to next node
            if (dir == null) {//if it next node is null then Initiliaze TrieNode()
                dir = new TrieNode();
                temp.child.put(ch[i], dir);// add character from ch variable
            }
            temp = dir;// changes temp value to current node at each iteration
            if (i == ch.length - 1) { //TODO
                temp.isLast = true;  // sets the value as true setting that last node for TRIE has reached
                temp.companyName = directory.getCompanyName();//getsCompanyname
                temp.phone = directory.getPhone();//gets phone number
                temp.fullName = directory.getFullName();// gets Full name
                temp.email = directory.getEmail();// gets Email address
            }
        }// loop end
```

#queryResults

```
STEP1: Declare a method queryResults
STEP2: Inside the method Initiliaze TrieNode temp = root;
STEP3: Declare a variable char[] ch and Initiliaze it with fullName as Character array;
STEP4: LOOP START:
    4.1:IF CURRENTNODE = NULL : RETURN NULL;
    4.2:NOW, Initiliaze temp=CURRENTNODE;
      END LOOP
STEP5: NOW, instantiate List<Directory> result = new ArrayList<>();
```

```
STEP6: Call  printQueryResults method with input as CURRENTNODE,query/prefix, result
STEP7: return the result;
```

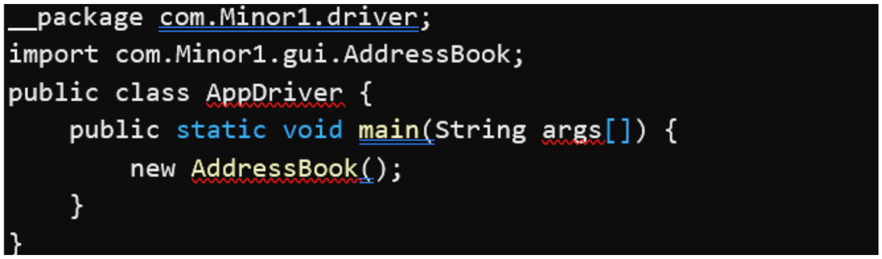
#printQueryResults

```
STEP1: Declare a method printQueryResults;
STEP2:IF CURRENTNODE=lastnode i.e. isLast=TRUE : then, add all corresponding values to CURRENTNODE like fullName, companyName, phone, email to result;
   result.add(new Directory(currNode.fullName, currNode.companyName, currNode.phone, currNode.email));
STEP3: LOOP START:
   3.1:Recursively call printQueryResults method till CURRENTNODE is not equal to null;
     LOOP END
```

3.1. CODE

3.1.a. Main

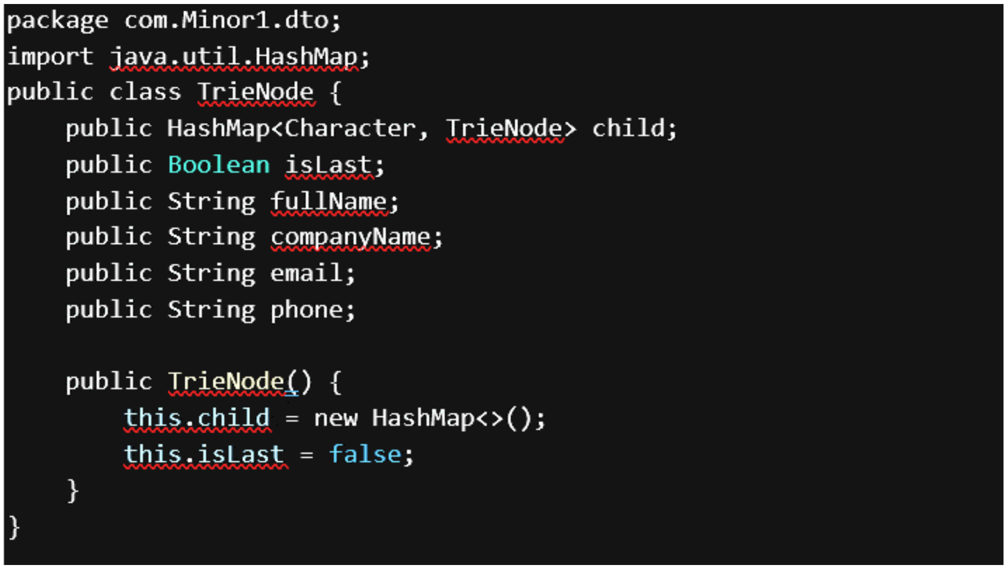
Appdriver.java



```
__package com.Minor1.driver;
import com.Minor1.gui.AddressBook;
public class AppDriver {
    public static void main(String args[]) {
        new AddressBook();
    }
}
```

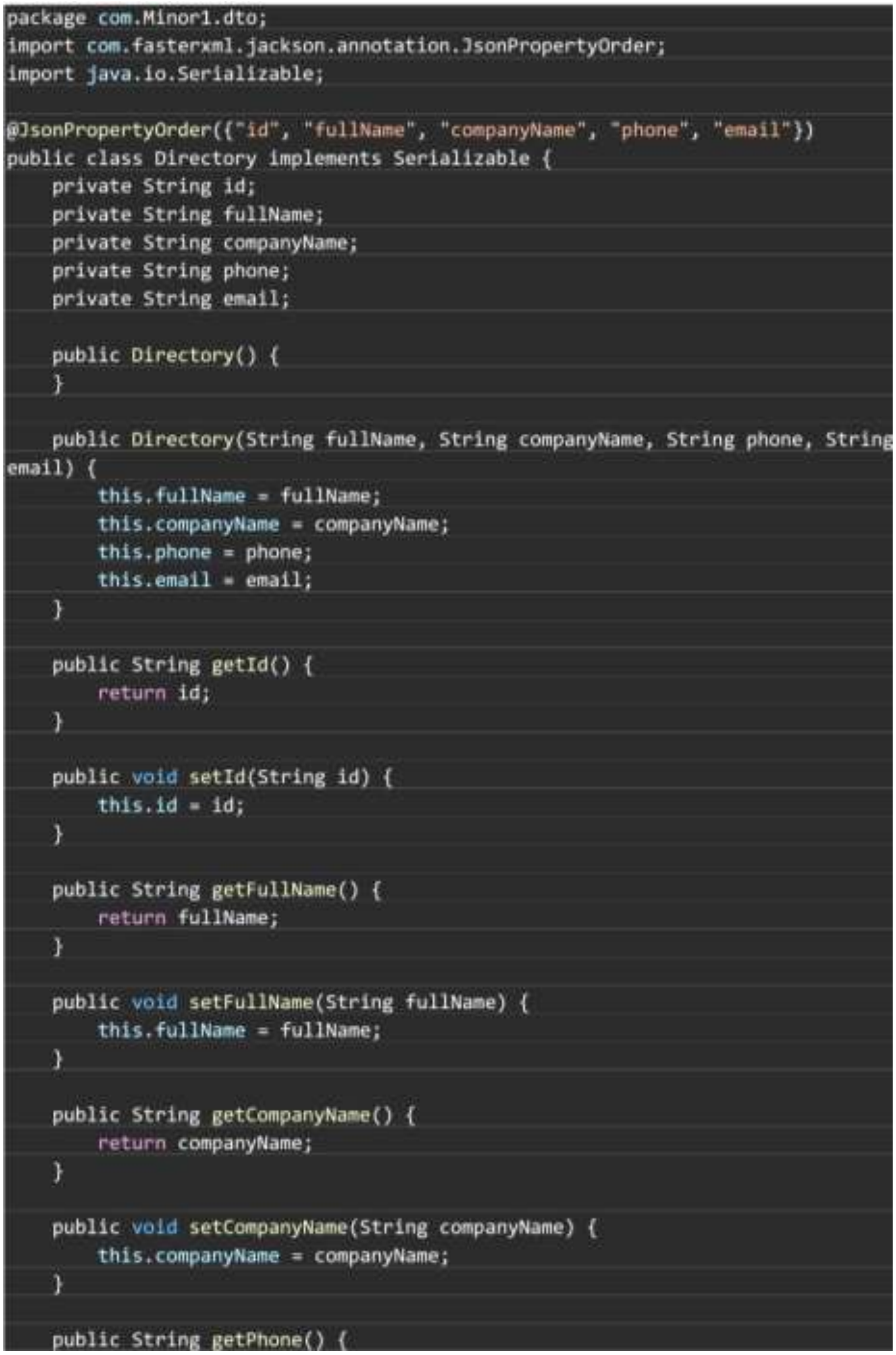
3.1.b

trieNode.java

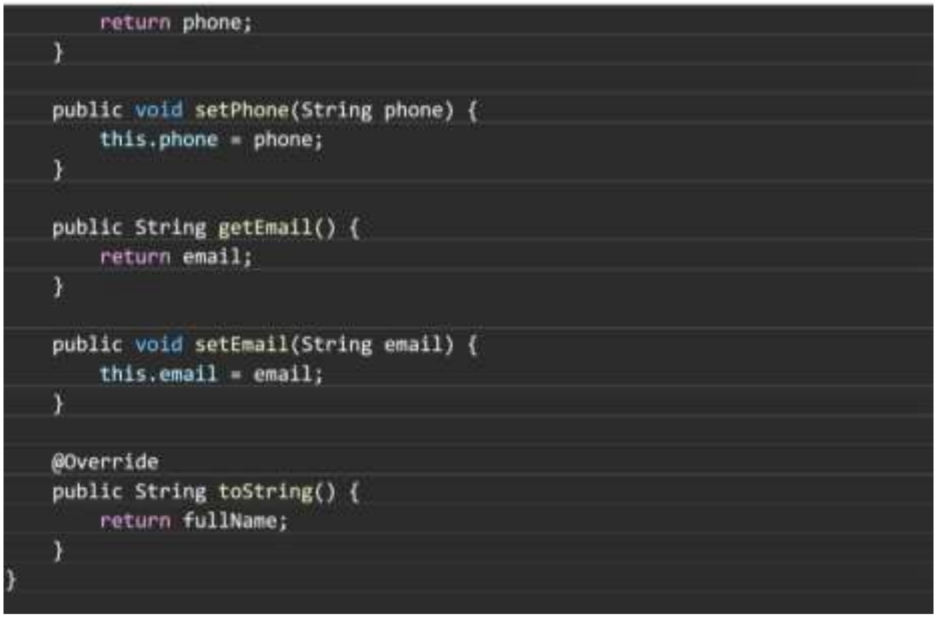


```
package com.Minor1.dto;
import java.util.HashMap;
public class TrieNode {
    public HashMap<Character, TrieNode> child;
    public Boolean isLast;
    public String fullName;
    public String companyName;
    public String email;
    public String phone;

    public TrieNode() {
        this.child = new HashMap<>();
        this.isLast = false;
    }
}
```

Directory.java



```
package com.Minor1.dto;
import com.fasterxml.jackson.annotation.JsonPropertyOrder;
import java.io.Serializable;

@JsonPropertyOrder({"id", "fullName", "companyName", "phone", "email"})
public class Directory implements Serializable {
    private String id;
    private String fullName;
    private String companyName;
    private String phone;
    private String email;

    public Directory() {
    }

    public Directory(String fullName, String companyName, String phone, String
email) {
        this.fullName = fullName;
        this.companyName = companyName;
        this.phone = phone;
        this.email = email;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getFullName() {
        return fullName;
    }

    public void setFullName(String fullName) {
        this.fullName = fullName;
    }

    public String getCompanyName() {
        return companyName;
    }

    public void setCompanyName(String companyName) {
        this.companyName = companyName;
    }

    public String getPhone() {
```



```
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return fullName;
    }
}
```

```
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return fullName;
    }
}
```

4. RESULT SECTION:

Figure 3, shows the GUI interface for user to input contact details.

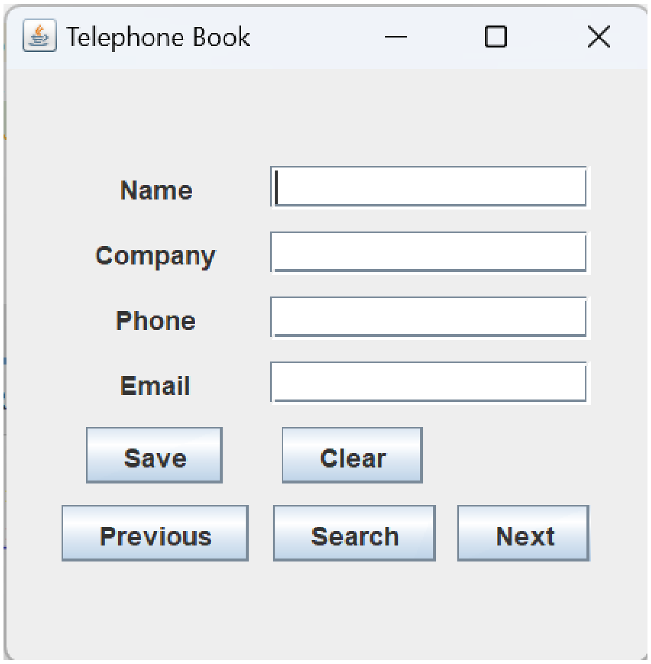
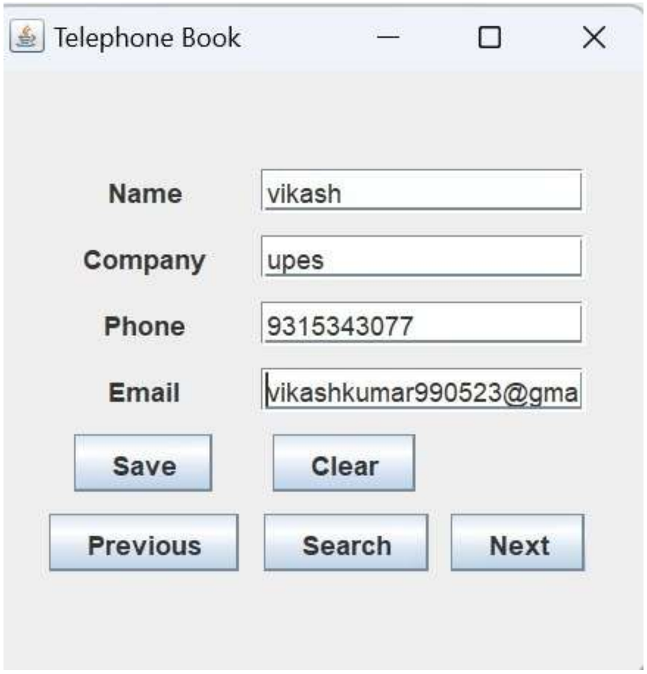


Fig.3: GUI of Phone Book

Figure 4, shows the value that has been inputted to be added to the directory.



Name vikash

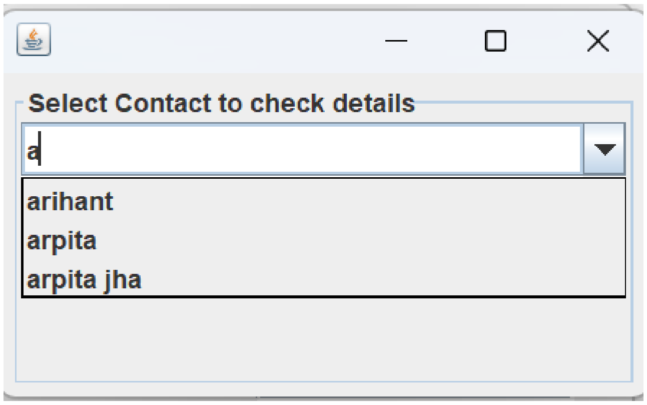
Company upes

Phone 9315343077

Email vikashkumar990523@gma

Fig.4: Insert Person Details

Figure 5, shows the searching GUI as well as suggestion based on the input query “a”



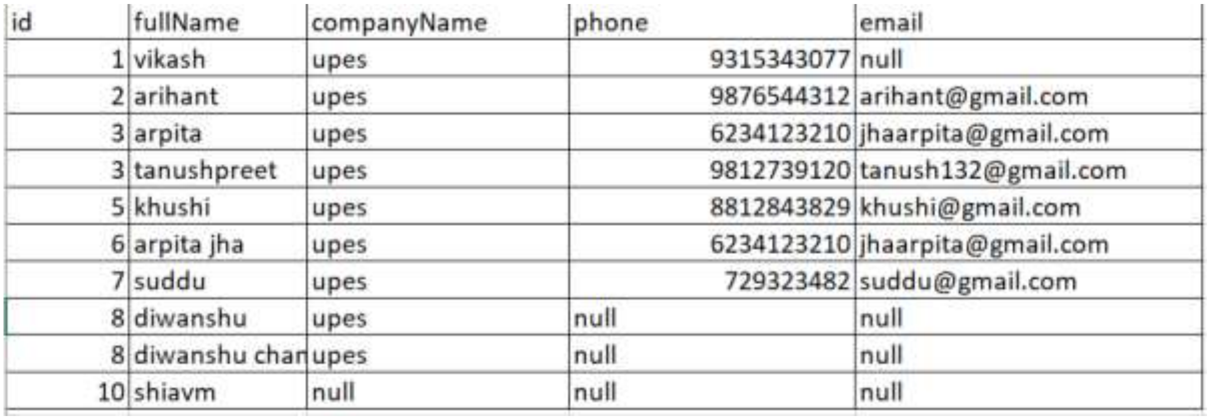
Select Contact to check details

a

arihant
arpita
arpita jha

Fig.5: Search by name

Figure 6, shows all the data that is stored in CSV file.



id	fullName	companyName	phone	email
1	vikash	upes	9315343077	null
2	arihant	upes	9876544312	arihant@gmail.com
3	arpita	upes	6234123210	jhaarpita@gmail.com
3	tanushpreet	upes	9812739120	tanush132@gmail.com
5	khushi	upes	8812843829	khushi@gmail.com
6	arpita jha	upes	6234123210	jhaarpita@gmail.com
7	suddu	upes	729323482	suddu@gmail.com
8	diwanshu	upes	null	null
8	diwanshu char	upes	null	null
10	shiavm	null	null	null

Fig.6: CSV

5. SWOT Analysis

Strength:

- Simple and user-friendly tool.
- Consists of basic features like add, View , Search

Weakness:

- Data is not being checked before being stored. (No, Format Checking).
- Searching done only on the basis of name.
- Large Storage requirement.

Opportunities:

- In the future we can store and connect it to cloud storage. So, that data can be accessed at multiple locations.

Threats:

- Keylogging: It can be used to collect data about the user organization or person.

6. PERT Chart

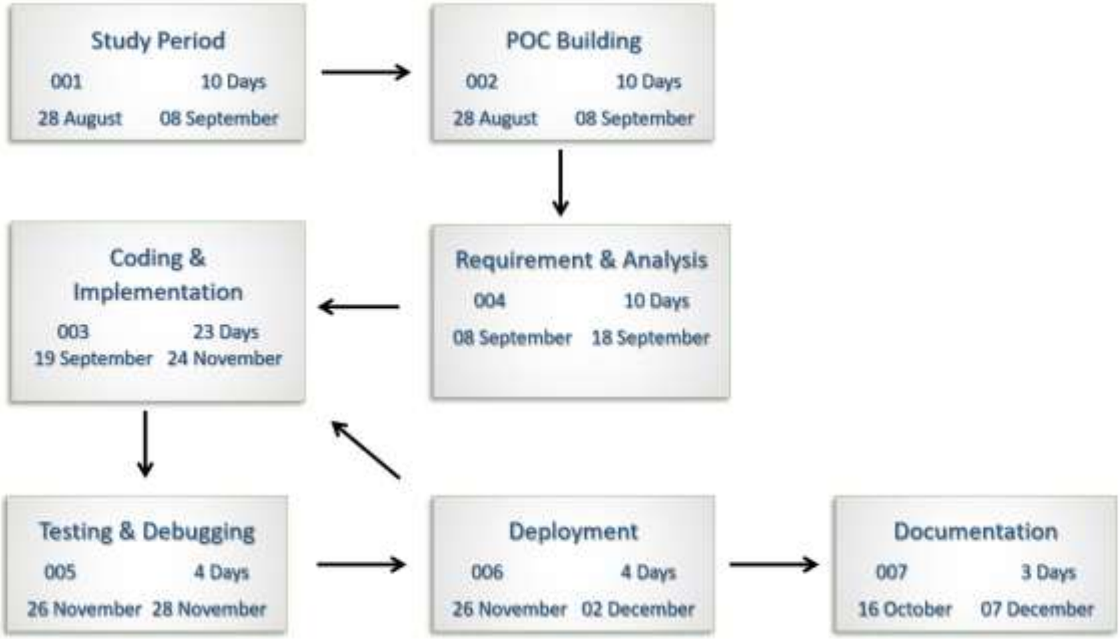


Fig.7: Timeline

7. Future Scope:

- Update ()- These features help users to update or modify the current database.
- Delete ()- This feature helps deleting a specific record.
- Event Reminder (Like: Birthday and so on.)

8. Reference:

[1] Kazuya Tsuruta, Dominik Köppl, Shunsuke Kanda, Yuto Nakashima, Shunsuke Inenaga,Hideo Bannai, Masayuki Takeda, c-TRIE++: A dynamic TRIE tailored for fast prefix searches, Information and Computation, Volume 285, Part B,2022 ,104794, ISSN 0890-5401,https://doi.org/10.1016/j.ic.2021.104794.

[2] An Introduction to data structures with applications - by Jean-Paul Tremblay and Paul Sorenson.

[3] Programming in ANSI C – E Balagurusamy.

[4] Let Us C – Yashawant kanitkar.

[5] Vimal P Parmar and C K Kumbharana Paper: Implementation of Trie Structure for Storing and Searching of English Spelled Homophone Words. International Journal of Scientific and Research Publications, Volume 7, Issue 1, January 2017 1 ISSN 2250-3153.

[6] Vimal P Parmar and C K Kumbharana Article: Study Existing Various Phonetic Algorithms and Designing and Development of a working model for the New Developed Algorithm and Comparison by implementing it with Existing Algorithm(s). International Journal of Computer Applications 98(19):45-49, July 2014

[7] Name and address matching strategy – White Paper December 2010.

[8] Vimal P Parmar, C K Kumbharana and Apurva K Pandya - Determining the Character Replacement Rules and Implementing Them for Phonetic Identification of Given Words to Identify Similar Pronunciation Words. Futuristic Trends on Computation Analysis and Knowledge Management (ABLAZE) 2015 International Conference at Greater Noida, India Pages: 272-277 Print ISBN: 978-1-4799-8432-9 DOI: 10.1109/ABLAZE.2015.7155010

[9] Analysis and Comparative Study on Phonetic Matching Techniques, Rima Shah, Dheeraj Kumar Singh, IJCA Volume 87 – No.9, February 2014**.**