Presented by : Himanshu Osti, Apurva Timilsina ,Ayush karki

Project Report Structure (Phone Diary)

1. Introduction

A phone diary is a digital system to store, manage, and retrieve contact details. Traditionally, people kept diaries or address books to note down phone numbers, but with computers, this can be managed more efficiently.
This project develops an Object-Oriented Phone Diary System that lets users add, view, search, edit, and delete contacts.

---

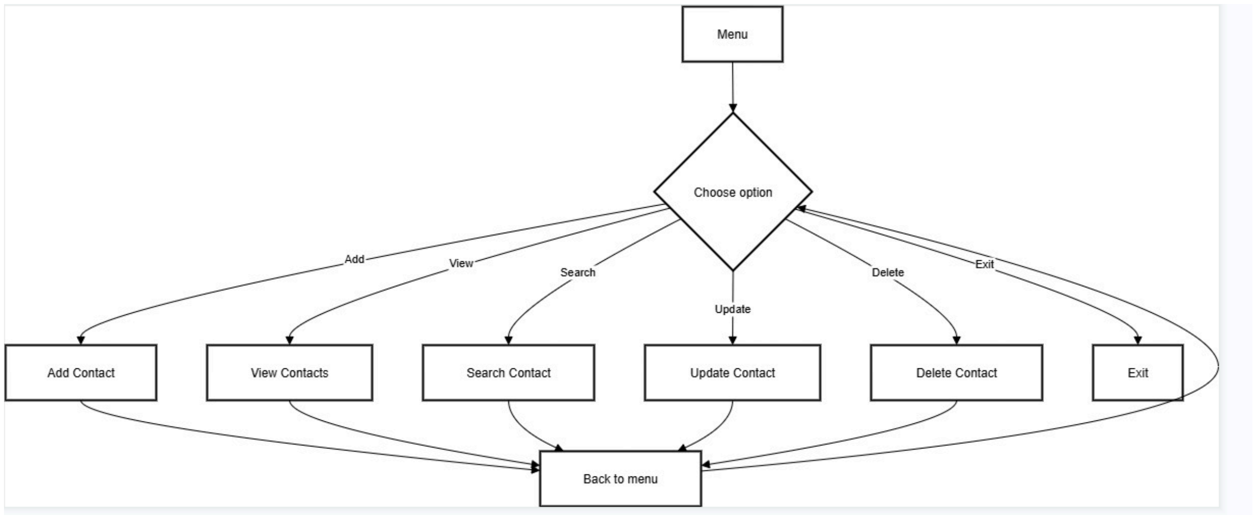
2. Problem Statement

People often lose or misplace physical diaries, and searching in them takes time. A digital phone diary provides an easy way to manage contacts in an organized manner using Object-Oriented Programming principles.

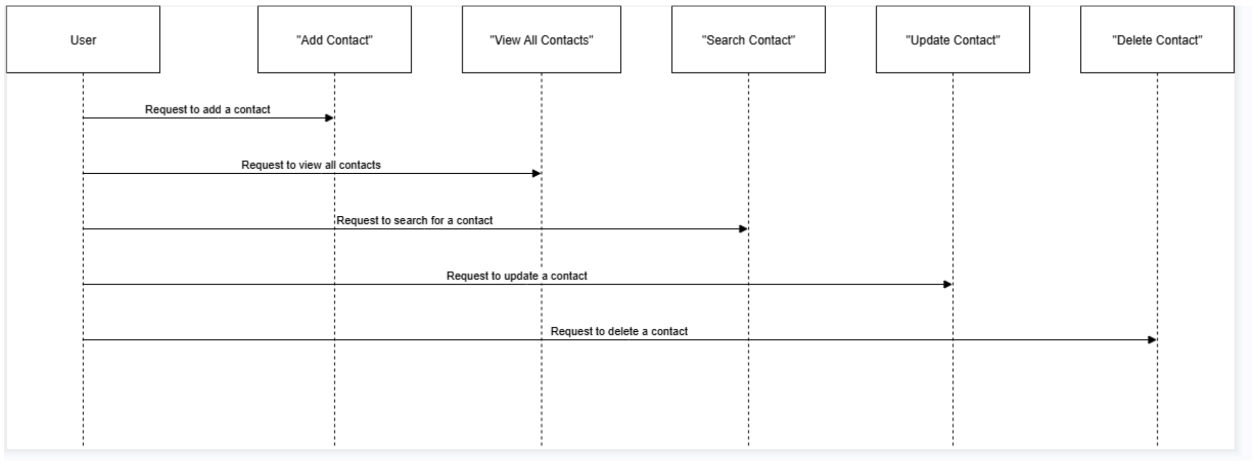
3. Objectives

- To design a Phone Diary System using OOP.
- To allow users to add, view, search, update, and delete contacts.
- To apply UML modeling (Use Case, Class, Sequence, Activity diagrams).
- To ensure the class diagram matches the implementation code.
- To use GitHub for version control and submission.

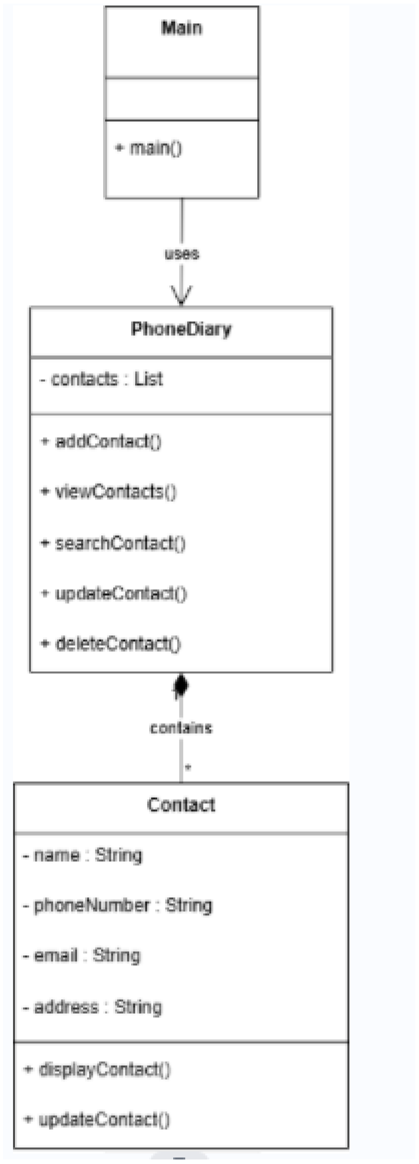
Activity diagram



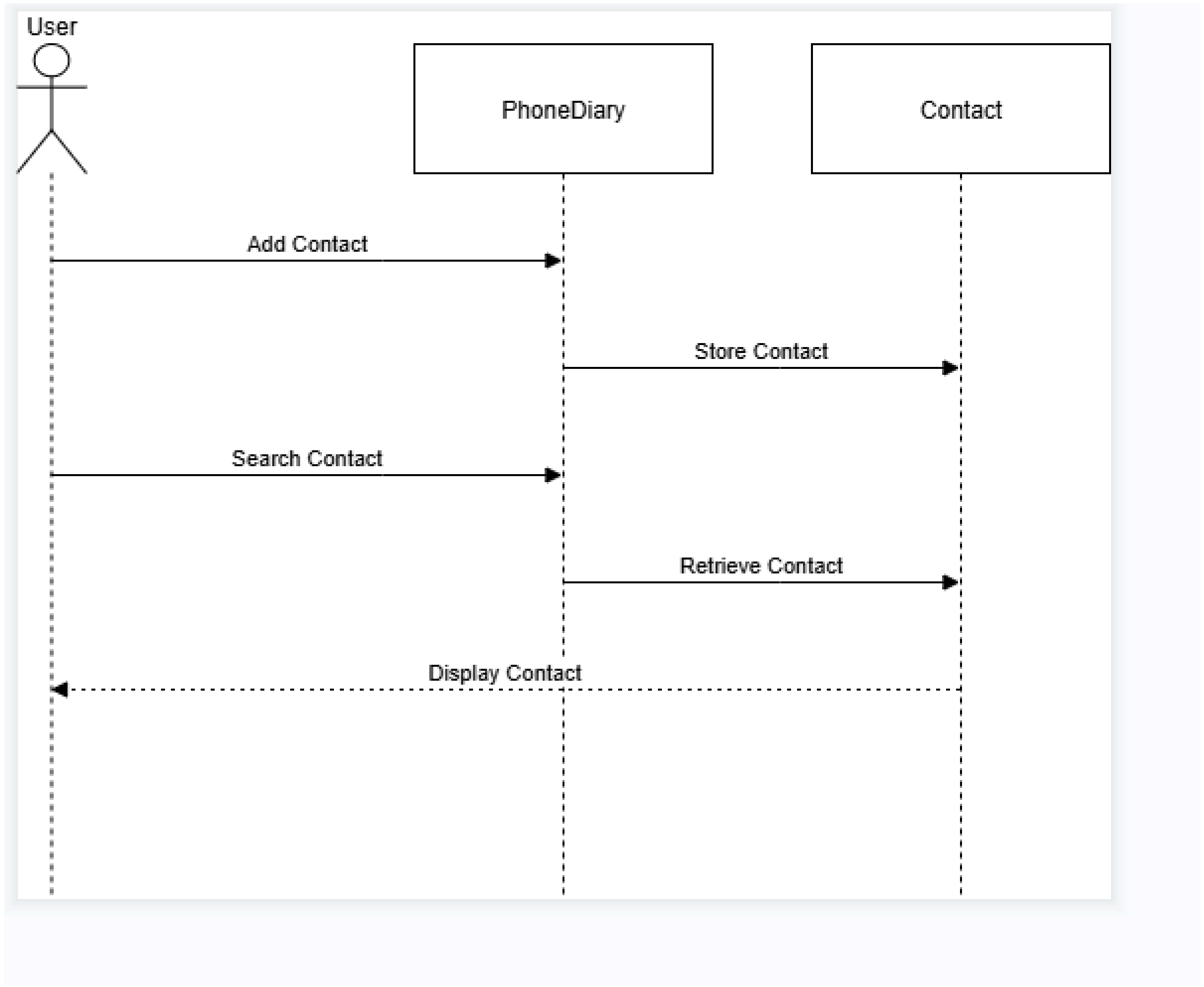
Use case diagram


User
"Add Contact"
"View All Contacts"
"Search Contact"
"Update Contact"
"Delete Contact"
Request to add a contact
Request to view all contacts
Request to search for a contact
Request to update a contact
Request to delete a contact

Class digram



Sequences Diagram



Guidance for Creating UML Diagrams

You can use Draw.io (https://app.diagrams.net/
) to create these diagrams:

Use Case Diagram:

Select UML shapes from the sidebar.

Use an actor for the User.

Add ovals to represent the actions: Add Contact, View All Contacts, Search Contact, Update Contact, Delete Contact.

Connect the User actor with each use case using association lines.

Class Diagram:

Use class boxes to represent the classes: Contact, PhoneDiary, and Main.

Add attributes and methods inside each class box:

Contact: name, phoneNumber, email, address; methods → displayContact(), updateContact().

PhoneDiary: contacts[] (list of Contact objects); methods → addContact(), viewContacts(), searchContact(), updateContact(), deleteContact().

Main: method → main().

Connect PhoneDiary with Contact (composition) and connect Main with PhoneDiary (usage).

Sequence Diagram:

Create lifelines for User, PhoneDiary, and Contact.

Show the interactions:

User → PhoneDiary: Add Contact → PhoneDiary → Contact: Store Contact.

User → PhoneDiary: Search Contact → PhoneDiary → Contact: Retrieve Contact → Contact → User: Display Contact.

Activity Diagram:

Use rounded rectangles to represent actions like Menu, Add Contact, View Contacts, Search Contact, Update Contact, Delete Contact.

Use a diamond shape to represent decision points (choosing an option).

Show the flow: Start → Menu → Choose option (Add/View/Search/Update/Delete) → Perform operation → Back to Menu → Exit.

After completing the diagrams, export each one as an image (PNG or JPEG)

Code of the project OOP

```
#include <iostream>
#include <vector>
```

```
#include <string>
#include <fstream>
using namespace std;

class Contact {
private:
    string name, phone, email, address;

public:
    Contact(string n="", string p="", string e="", string a="") {
        name = n; phone = p; email = e; address = a;
    }

    string getName() { return name; }
    string getPhone() { return phone; }
    string getEmail() { return email; }
    string getAddress() { return address; }

    void display() {
        cout << "Name: " << name
             << "\nPhone: " << phone
             << "\nEmail: " << email
             << "\nAddress: " << address << endl;
    }

    // Save contact in file
    void saveToFile(ofstream &out) {
        out << name << "|" << phone << "|" << email << "|" << address << "\n";
    }

    // Load contact from file line
    static Contact fromString(string line) {
        string n, p, e, a;
        size_t pos = 0;

        pos = line.find("|"); n = line.substr(0, pos); line.erase(0, pos+1);
        pos = line.find("|"); p = line.substr(0, pos); line.erase(0, pos+1);
        pos = line.find("|"); e = line.substr(0, pos); line.erase(0, pos+1);
        a = line;
        return Contact(n, p, e, a);
    }
};

class PhoneDiary {
private:
    vector<Contact> contacts;
    string filename = "contacts.txt";

public:
    PhoneDiary() { loadFromFile(); }
    ~PhoneDiary() { saveToFile(); }

    void addContact(Contact c) {
        contacts.push_back(c);
        saveToFile();
    }

    void viewContacts() {
        if (contacts.empty()) {
            cout << "No contacts saved yet!\n";
            return;
        }
        for (size_t i = 0; i < contacts.size(); i++) {
            cout << "\nContact " << i+1 << ":\n";
            contacts[i].display();
        }
    }

    void searchContact(string name) {
        bool found = false;
        for (auto &c : contacts) {
            if (c.getName() == name) {
                c.display();
                found = true;
                break;
            }
        }
        if (!found) cout << "Contact not found!" << endl;
    }
    void deleteContact(string name) {
        for (size_t i = 0; i < contacts.size(); i++) {
            if (contacts[i].getName() == name) {
                contacts.erase(contacts.begin() + i);
                cout << "Contact deleted.\n";
                saveToFile();
                return;
            }
        }
        cout << "Contact not found!\n";
    }

    void saveToFile() {
        ofstream out(filename);
        for (auto &c : contacts) {
            c.saveToFile(out);
        }
        out.close();
    }

    void loadFromFile() {
        ifstream in(filename);
        string line;
        while (getline(in, line)) {
            if (!line.empty()) {
                contacts.push_back(Contact::fromString(line));
            }
        }
        in.close();
    }
};

int main() {
    PhoneDiary diary;
    int choice;
    string name, phone, email, address;

    do {
        cout << "\n--- PHONE DIARY ---";
        cout << "\n1. Add Contact\n2. View All\n3. Search\n4. Delete\n5. Exit\nChoice: ";
        cin >> choice;
        cin.ignore();

        if (choice == 1) {
            cout << "Enter Name: "; getline(cin, name);
            cout << "Enter Phone: "; getline(cin, phone);
            cout << "Enter Email: "; getline(cin, email);
            cout << "Enter Address: "; getline(cin, address);
            Contact c(name, phone, email, address);
            diary.addContact(c);
        } else if (choice == 2) {
            diary.viewContacts();
        } else if (choice == 3) {
            cout << "Enter Name to Search: "; getline(cin, name);
            diary.searchContact(name);
        } else if (choice == 4) {
            cout << "Enter Name to Delete: "; getline(cin, name);
            diary.deleteContact(name);
        }

    } while (choice != 5);

    return 0;
}
```