‘Crime Control’

Project submitted to

SRM University - AP, Andhra Pradesh

for the partial fulfillment of requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Gnana Prudhvi Rasamsetti

(AP23110010542)

Under the Guidance of

Mrs. Kavitha Rani Karnena

SRM University - AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh - 522240

[August, 2027]

Certificate

Date: 20 November 2024

This is to certify that the work present in this Project entitled "**Crime Control**" has been carried out by Gnana Prudhvi Rasamsetti under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in **School of Engineering and Sciences**.

Supervisor

Dr. Kavitha Rani Karnena

Faculty - B.Tech CSE-H, Semester (AY 2024-25)

SRM University – *AP*

Acknowledgements

We would like to express our sincere gratitude to all those who supported and guided us throughout this project.

Dr. Kavitha Rani Karnena: For providing invaluable guidance, constructive feedback, and motivation at every stage of this project. Your insights and expertise were instrumental in shaping our understanding and approach.

SRM University-AP: For offering a conducive learning environment, access to resources, and encouragement to explore innovative ideas.

Our Team Members: Prudhvi, Kishore, Adithya, and Navanit—our collaboration, dedication, and mutual support were key to the successful completion of this project. Each member's unique contributions enriched the outcome, from technical implementation to documentation and presentation.

Friends and Family: For your encouragement and understanding, which inspired us to persevere even in challenging moments.

This project was not only a technical achievement but also a testament to the power of teamwork and mentorship. We are deeply grateful for all the opportunities to learn and grow through this experience.

Thank you!

Table of Contents

Abstract

This C++ program implements a basic prototype of a **'Crime Investigative Support & Reporting System'** that allows both public users and authority figures to interact with a report database. The system facilitates two primary functionalities:

1. Public User Operations

Public users can sign up by providing their name, phone number, and password. Their details are validated to ensure authenticity and stored in a persistent file-based database. Once registered, public users can sign in and report incidents by specifying the type and description of the case. These reports are saved in text files categorized by case type within a designated directory.

2. Authority Operations

Authority users can sign in using pre-defined credentials. Upon successful authentication, they can search for specific reports based on keywords. The system scans filenames and file contents within the directory, identifying matching reports. The results are displayed, and the user can choose to view the full contents of a report interactively.

Core Features:

- User Management: Includes sign-up and sign-in functionalities with robust validation.
- Report Filing: Public users can log incidents, which are stored in categorized text files.
- Search Functionality: Authority users can search reports by keywords in filenames or content.
- Dynamic Storage: Public user data is managed dynamically with resizing capabilities.
- Persistence: Public user information and reports are saved in files for reuse across sessions.

Use Case:

This system is a prototype for handling public complaints and administrative searches, suitable for organizations aiming to streamline incident reporting and management. The program leverages standard libraries for file handling and filesystem traversal while ensuring extensibility and ease of use.

Statement of Contributions

As a cohesive team of four — **Gnana Prudhvi Rasamsetti**, **Vuppu Kishore**, **Adithya Koduru**, and **Navanit Reddy Turpinti** — we collaborated on the development of this C++ program with clearly defined roles and responsibilities, leveraging our collective strengths to achieve the desired functionality and user experience.

Gnana Prudhvi Rasamsetti (AP23110010542)

- Led the implementation of the Authority user functionality, including the search feature for keyword-based report retrieval.
- Handled the implementation of file-based data storage and retrieval for public user information and incident reports.
- Integrated the `filesystem` library for efficient directory traversal and content analysis.
- Conducted extensive testing of all functionalities, ensuring robustness and resolving bugs.

Vuppu Kishore (AP23110010563)

- Managed the Public user operations, including the design of the sign-up and incident reporting features.
- Ensured robust error handling during file access and directory operations.
- Designed the presentation (PPT) for showcasing the program, ensuring clarity and visual appeal.
- Conducted extensive testing of all functionalities, ensuring robustness and resolving bugs.

Adithya Koduru (AP23110010571)

- Designed and implemented the Public User Database class, enabling dynamic storage and management of user data.
- Created functions for resizing the user database, loading users from a file, and saving data persistently.
- Streamlined the program's architecture for scalability and efficiency.

- Conducted extensive testing of all functionalities, ensuring robustness and resolving bugs.

Navanit Reddy Turpinti (AP23110010578)

- Coordinated the overall program structure and flow, ensuring seamless integration of all features.
- Focused on the user interface design, making the program intuitive and user-friendly.
- Developed the validation mechanisms for user credentials, ensuring accuracy and security.
- Designed the report, presenting the project's implementation, features, and outcomes in a structured and professional format.

Through consistent communication and teamwork, we successfully built a program that meets the objectives of 'Crime Investigative Support & Reporting System' while adhering to best practices in software development. Each member's contributions were integral to the project's success, highlighting the power of collaboration and shared expertise.

List of Figures

1 Introduction

1.1 Overview

This project aims to develop a comprehensive **'Crime Investigative Support & Reporting System'** that streamlines the interaction between public users and authority figures. Built using C++, the program incorporates robust functionalities to ensure secure user management, efficient incident reporting, and seamless report retrieval.

1.1.1 Key Objectives

1. **Simplified User Management**
- Allow public users to sign up and securely store their information.
- Facilitate user sign-in for both public and authority users with appropriate credential validation.

2. **Incident Reporting**
- Enable public users to report incidents with detailed descriptions.
- Save reports categorically in text files for easy retrieval and future reference.

3. **Keyword-Based Report Search**
- Provide authority users with the capability to search for reports based on keywords in filenames or content.
- Present search results interactively, allowing users to view the contents of relevant files.

1.1.2 Features

1. **User Classes**
- Public Users: Can register, log in, and report incidents.
- Authority Users: Have the additional ability to search and view reports.

2. **Data Management**
- Dynamic storage of public user data using a scalable database.
- Persistent file-based storage for user records and incident reports.

3. **Validation Mechanisms**
- Strict validation for user inputs (name, phone number, and password) to maintain data integrity.

4. **File Handling and Search**
- Efficient directory traversal using the `filesystem` library to locate and retrieve reports.

1.1.3 Applications

This system is ideal for organizations or institutions managing public grievances or incidents, such as:
- Law enforcement agencies for case reporting and tracking.
- Municipal bodies for grievance redressal systems.
- Corporate or educational institutions for streamlined complaint handling.

The project demonstrates the effective use of object-oriented programming principles, file handling, and dynamic memory management, making it a robust and scalable solution for real-world reporting systems.

1.2 Structure of the Project

1.2.1 Flowcharts

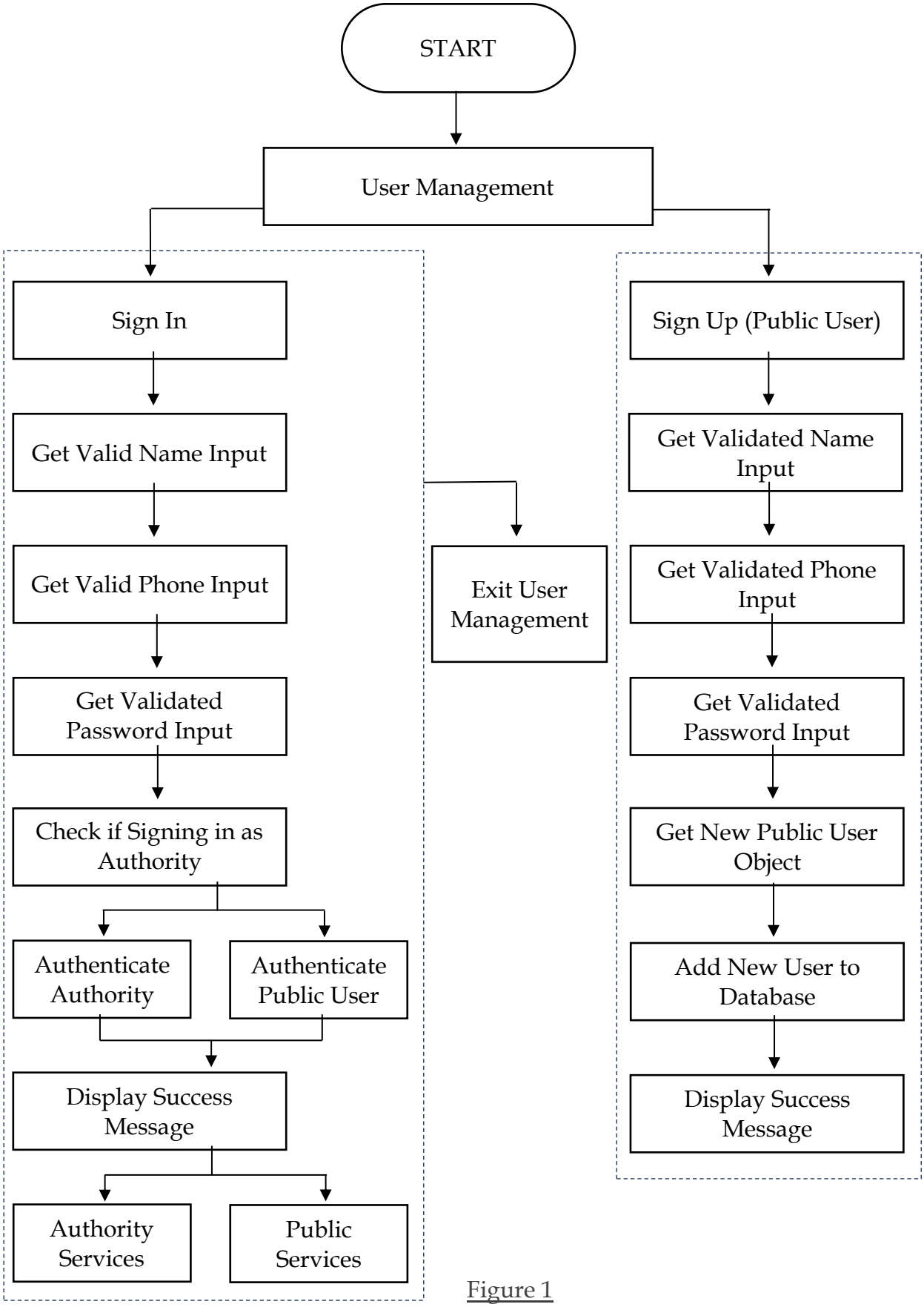


Figure 1

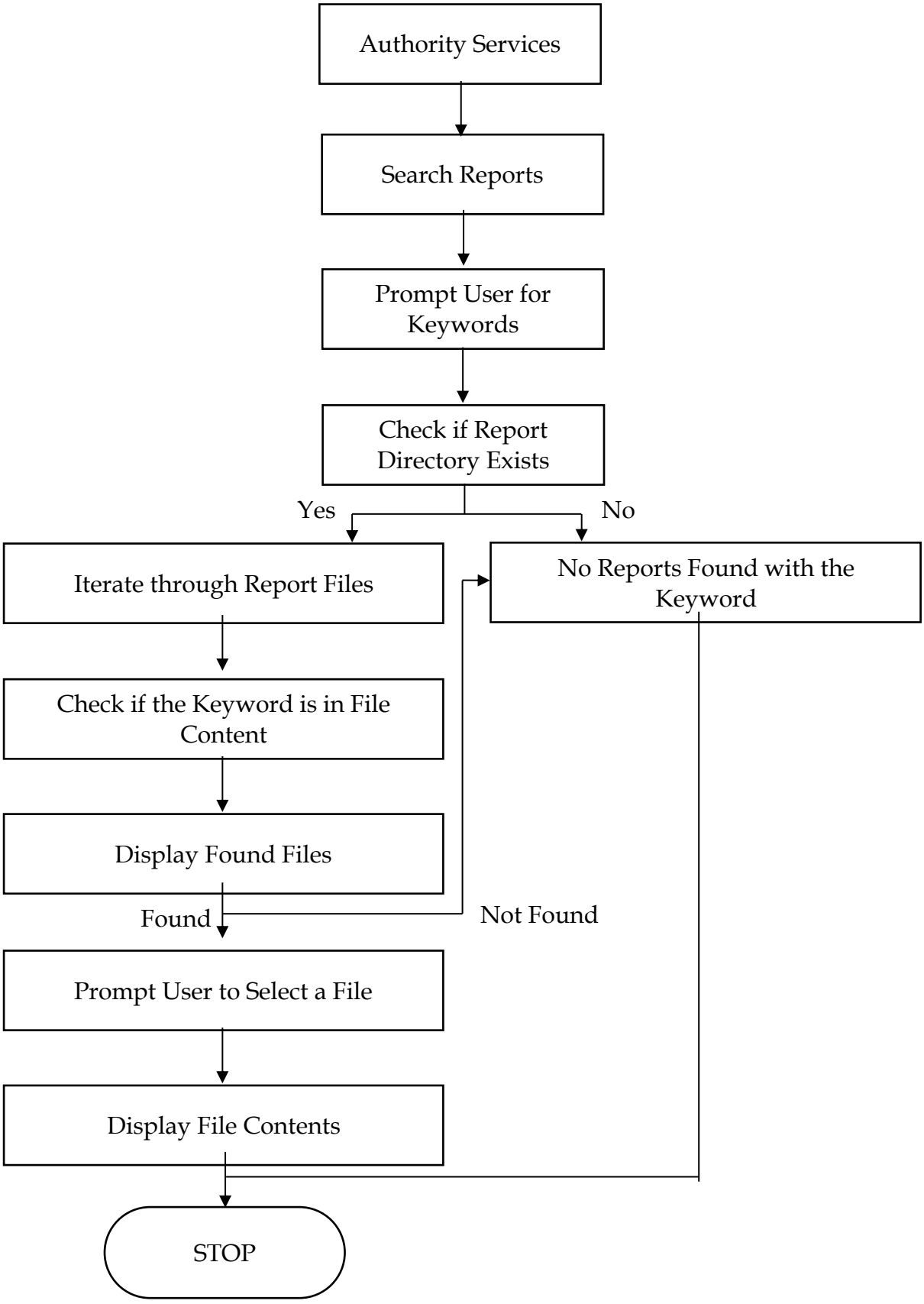

Authority Services
Search Reports
Prompt User for Keywords
Check if Report Directory Exists
Yes
No
Iterate through Report Files
No Reports Found with the Keyword
Check if the Keyword is in File Content
Display Found Files
Found
Not Found
Prompt User to Select a File
Display File Contents
STOP

Figure 2

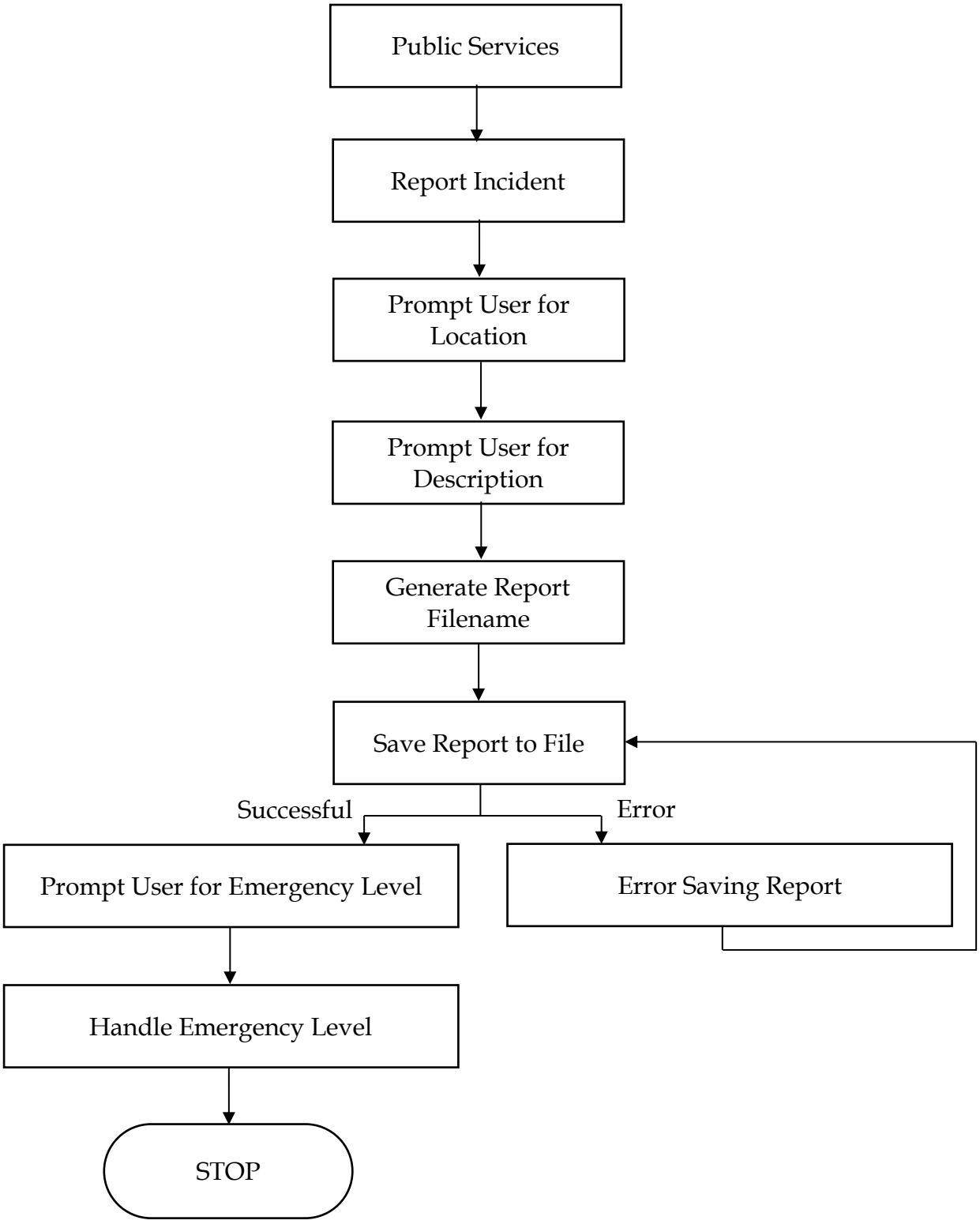

Public Services
Report Incident
Prompt User for Location
Prompt User for Description
Generate Report Filename
Save Report to File
Successful
Error
Prompt User for Emergency Level
Error Saving Report
Handle Emergency Level
STOP

Figure 3

1.2.2 Theoretical Analysis

This analysis provides a breakdown of the project's workflow, detailing the logical steps and considerations involved in its development and execution.

1. **Requirement Definition**

- Objective: Identify the problem and define the solution.
  - A system to manage user data, report incidents, and allow keyword-based report retrieval.
- Outcome: Clarity on system requirements:
  - Two user types: Public and Authority.
  - Key functionalities: User management, report filing, and searching.Persistent file-based storage for user records and incident reports.

2. **System Design**

- User Management Structure: Design a base class User with shared attributes (name, phone, and password) and create derived classes:
  - Public: For reporting incidents.
  - Authority: For searching reports.
- Data Storage Plan:
  - Use text files for persistent storage of public user data and reports.
  - Employ dynamic memory allocation for managing in-memory user data.
- Search Mechanism:
  - Utilize the std::filesystem library for directory traversal and keyword matching.
- Flowchart Development:
  - Visualize interaction paths for sign-up, sign-in, report filing, and report retrieval.

3. **Implementation**

- Coding Framework:
  - Define classes and methods.
  - Write reusable utility functions for validation (isValidName, isValidPhone, isValidPassword).
- Public User Features:
  - Sign-up: Collect and validate user inputs, store in a database.
  - Report filing: Save categorized reports in the file system.
- Authority Features:
  - Sign-in: Validate credentials against pre-defined Authority data.
  - Report search: Traverse directories and files, displaying results interactively.

- Dynamic User Database:
  - Implement a resizable PublicUserDatabase for managing in-memory user data.

4. Validation and Security

- Input Validation:
  - Ensure correct formats for name, phone, and password.
- Data Security:
  - Secure sensitive information by limiting access to passwords and user data in code.
- Error Handling:
  - Prevent crashes due to invalid inputs, missing files, or inaccessible directories.

5. Testing

- Unit Testing:
  - Test each function and method independently.
  - Verify input validation, file handling, and database operations.
- Integration Testing:
  - Test interactions between modules, ensuring smooth transitions.
  - Validate user actions (e.g., sign-up followed by incident reporting).
- Edge Cases:
  - Invalid user inputs (e.g., special characters in name or incorrect phone number).
  - Large datasets in directory traversal.
  - Non-existent directories or empty files.

6. Refinement

- Performance Optimization:
  - Optimize directory traversal to minimize search time.
  - Reduce redundant file access during keyword-based searches.
- User Experience (UX):
  - Ensure clear and concise prompts for user actions.
  - Provide meaningful error messages and confirmations.

7. Deployment

- System Setup:
  - Create a structured directory (C++ Project) for storing reports.
  - Preload authority user data into the system.
- Execution:
  - Run and test the application in different environments to confirm portability.

8. Documentation

- Technical Report:
  - Document system design, methodology, implementation, and testing results.
- Presentation:
  - Prepare a professional PowerPoint summarizing objectives, features, and outcomes.

Step-by-Step Execution Workflow

1. User Sign-Up:
   - Public user provides name, phone, and password.
   - System validates inputs and stores user data in the database and file.

2. User Sign-In:
   - User chooses between Public or Authority login.
   - Credentials are validated.

3. Incident Reporting (Public):
   - User describes the case and saves the report under a categorized filename.

4. Report Search (Authority):
   - Authority user inputs a keyword.
   - System searches for matches in filenames and content, presenting results interactively.

5. Exit:
   - Save all changes (e.g., new users) to persistent storage.

2 Methodology

2.1 Algorithm

This algorithm provides a clear, structured flow to manage user actions and system functionalities effectively.

Step 1: Start.

Step 2: Initialize system variables and classes.

i. Declare PublicUserDatabase instance to manage public users.
ii. Create an array authorityUsers[] with pre-defined Authority credentials.

Step 3: Display the main menu with options:

i. Sign Up (Public).
ii. Sign In (Public/Authority).
iii. Exit.

Step 4: Read the user's choice.

i. If the choice is 1:
   Proceed with the Sign-Up process.
ii. If the choice is 2:
   Proceed with the Sign-In process.
iii. If the choice is 3:
   Exit the program.

Step 5: Sign-Up Process (Public):

i. Read the user's name, phone number, and password.
ii. Validate the inputs:
   Name should contain only alphabets.
   Phone number should have exactly 10 digits.
   Password should be at least 8 characters long.
iii. If valid:

Create a new Public user object.
Add the user to the database.
Save the updated database to a file.
Display a success message.

iv. Else:
Display an error message and prompt the user to try again.

Step 6: Sign-In Process:

i. Read the user's name, phone number, and password.
ii. Ask if the user is an Authority or a Public user.
iii. If Authority:
Check the entered credentials against authorityUsers[].
If valid:
Allow the user to search reports by keyword.
Traverse the report directory for matching files or content.
Display the matched files and allow the user to view their content.
Else:
Display an error message for invalid credentials.
iv. If Public:
Search the PublicUserDatabase for matching credentials.
If valid:
Allow the user to report an incident.
Read the case type and description.
Save the report in a categorized text file.
Else:
Display an error message for invalid credentials.

Step 7: Handle incorrect inputs or invalid choices in both Sign-Up and Sign-In processes.

Step 8: If the user chooses to exit:

i. Save all changes (e.g., new users) to persistent storage.
ii. Display a farewell message.

Step 9: Stop.

2.2 Project Code

```
#include <iostream>
#include <fstream>
#include <string>
#include <cctype>
#include <algorithm>
#include <filesystem>
#include <vector>

using namespace std;
namespace fs = std::filesystem;

// Base User Class
class User {
protected:
    string name;
    string phone;
    string password;

public:
    User(string n, string p, string pass) : name(move(n)), phone(move(p)),
password(move(pass)) {}

    string getName() const { return name; }
    string getPhone() const { return phone; }
    string getPassword() const { return password; }

    // Validation functions
    static bool isValidName(const string& name) {
        return all_of(name.begin(), name.end(), ::isalpha);
    }

    static bool isValidPhone(const string& phone) {
        return phone.length() == 10 && all_of(phone.begin(), phone.end(), ::isdigit);
    }

    static bool isValidPassword(const string& password) {
        return password.length() >= 8;
    }
};

// Authority Class (inherits from User)
```

```
class Authority : public User {
public:
    Authority(string n, string p, string pass) : User(move(n), move(p), move(pass)) {}

    // Search reports
    void searchReports(const string& initialKeyword) const {
        string currentKeyword = initialKeyword; // Start with the initial keyword

        while (true) {
            cout << "Searching reports for keyword: " << currentKeyword << "\n";

            string reportDir = "C++ Project"; // Directory to search

            if (!fs::exists(reportDir)) {
                cout << "Directory " << reportDir << " does not exist.\n";
                return;
            }

            vector<fs::path> foundFiles;

            // Search reports in the directory
            for (const auto& file : fs::directory_iterator(reportDir)) {
                if (file.path().filename().string().find(currentKeyword) != string::npos) {
                    foundFiles.push_back(file.path());
                    cout << "Found in filename: " << file.path().filename() << "\n";
                }

                ifstream infile(file.path());
                if (infile) {
                    string line;
                    while (getline(infile, line)) {
                        if (line.find(currentKeyword) != string::npos) {
                            foundFiles.push_back(file.path());
                            cout << "Found in content of: " << file.path().filename() << "\n";
                            break;
                        }
                    }
                }
            }

            // Handle cases when no reports are found
            if (foundFiles.empty()) {
                cout << "No reports found with the keyword \"" << currentKeyword << "\".\n";
                cout << "Do you want to search for another keyword? (y/n): ";
```

```
            char retry;
            cin >> retry;
            cin.ignore(); // Clear input buffer

            if (tolower(retry) == 'y') {
                cout << "Enter a new keyword to search: ";
                getline(cin, currentKeyword); // Update the current keyword
                continue; // Restart the search
            } else {
                cout << "Exiting the search.\n";
                return;
            }
        }

        // Handle cases when reports are found
        int choice;
        while (true) {
            cout << "\nFiles found:\n";
            for (size_t i = 0; i < foundFiles.size(); ++i) {
                cout << i + 1 << ". " << foundFiles[i].filename() << "\n";
            }
            cout << "Enter the number of the file you want to read (or 0 to search for another
keyword, -1 to exit): ";
            cin >> choice;

            if (choice == -1) {
                cout << "Exiting the search.\n";
                return;
            }

            if (choice == 0) {
                cout << "Enter a new keyword to search: ";
                cin.ignore(); // Clear input buffer
                getline(cin, currentKeyword); // Update the current keyword
                break; // Restart the search loop
            }

            if (choice > 0 && choice <= static_cast<int>(foundFiles.size())) {
                ifstream infile(foundFiles[choice - 1]);
                if (infile) {
                    cout << "\n--- Contents of " << foundFiles[choice - 1].filename() << " ---\n";
                    string line;
                    while (getline(infile, line)) {
                        cout << line << "\n";
```

```
                }
                cout << "----------------------------------------\n";
            } else {
                cout << "Error opening file.\n";
            }
        } else {
            cout << "Invalid choice. Try again.\n";
        }
    }
  }
};

// Public Class (inherits from User)
class Public : public User {
public:
  Public(string n, string p, string pass) : User(move(n), move(p), move(pass)) {}

  void reportIncident() const {
    string caseType, location, description;

    cout << "Enter the type of case you are reporting (e.g., Theft, Murder): ";
    cin.ignore(); // Clear the input buffer
    getline(cin, caseType);

    cout << "Enter the location of the incident: ";
    getline(cin, location);

    cout << "Enter a detailed description of the case:\n";
    getline(cin, description);

    // Sanitize input for filename by replacing invalid characters with underscores
    auto sanitize = [](string& str) {
      for (char& c : str) {
        if (!isalnum(c) && c != '_' && c != '-') {
          c = '_'; // Replace invalid characters with '_'
        }
      }
    };

    sanitize(caseType);
    sanitize(location);

    // Construct the filename
```

```
        string filename = "C++ Project/" + caseType + "-" + location + "-" + name + ".txt";

        ofstream file(filename);
        if (file) {
            file << "Reported by: " << name << "\n";
            file << "Phone: " << phone << "\n";
            file << "Location: " << location << "\n";
            file << "Description: " << description << "\n";
            file << "-------------------------------------\n";

            cout << "\nIncident report saved successfully!!\n";
            cout << "Please select the level of emergency:\n";
            cout << "1. Highly Emergency (Sending location to control room and nearest
hospital)\n";
            cout << "2. Normal Crime (To be solved after investigation)\n";
            cout << "Enter your choice: ";

            int emergencyLevel;
            cin >> emergencyLevel;

            if (emergencyLevel == 1) {
                cout << "An emergency alert has been sent with your location to the control room
and the nearest hospital.\n";
            } else if (emergencyLevel == 2) {
                cout << "Your case has been recorded and will be solved after investigation.\n";
            } else {
                cout << "Invalid choice! Defaulting to normal processing.\n";
            }
        } else {
            cout << "Error saving the report.\n";
        }
    }
};

// Public User Database
class PublicUserDatabase {
    Public** publicUsers;
    int userCount;
    int capacity;

    void resize() {
        capacity *= 2;
        Public** newUsers = new Public * [capacity];
        for (int i = 0; i < userCount; ++i) {
```

```
            newUsers[i] = publicUsers[i];
        }
        delete[] publicUsers;
        publicUsers = newUsers;
    }

public:
    PublicUserDatabase(int initialCapacity = 5) : userCount(0), capacity(initialCapacity) {
        publicUsers = new Public * [capacity];
        loadUsersFromFile();
    }

    ~PublicUserDatabase() {
        saveUsersToFile();
        for (int i = 0; i < userCount; ++i) {
            delete publicUsers[i];
        }
        delete[] publicUsers;
    }

    void addPublicUser(Public* newUser) {
        if (userCount == capacity) {
            resize();
        }
        publicUsers[userCount++] = newUser;
    }

    Public* findPublicUser(const string& name, const string& phone, const string& password)
const {
        for (int i = 0; i < userCount; ++i) {
            if (publicUsers[i]->getName() == name &&
                publicUsers[i]->getPhone() == phone &&
                publicUsers[i]->getPassword() == password) {
                return publicUsers[i];
            }
        }
        return nullptr;
    }

    void loadUsersFromFile() {
        ifstream file("public_users.txt");
        string name, phone, password;

        while (file >> name >> phone >> password) {
```

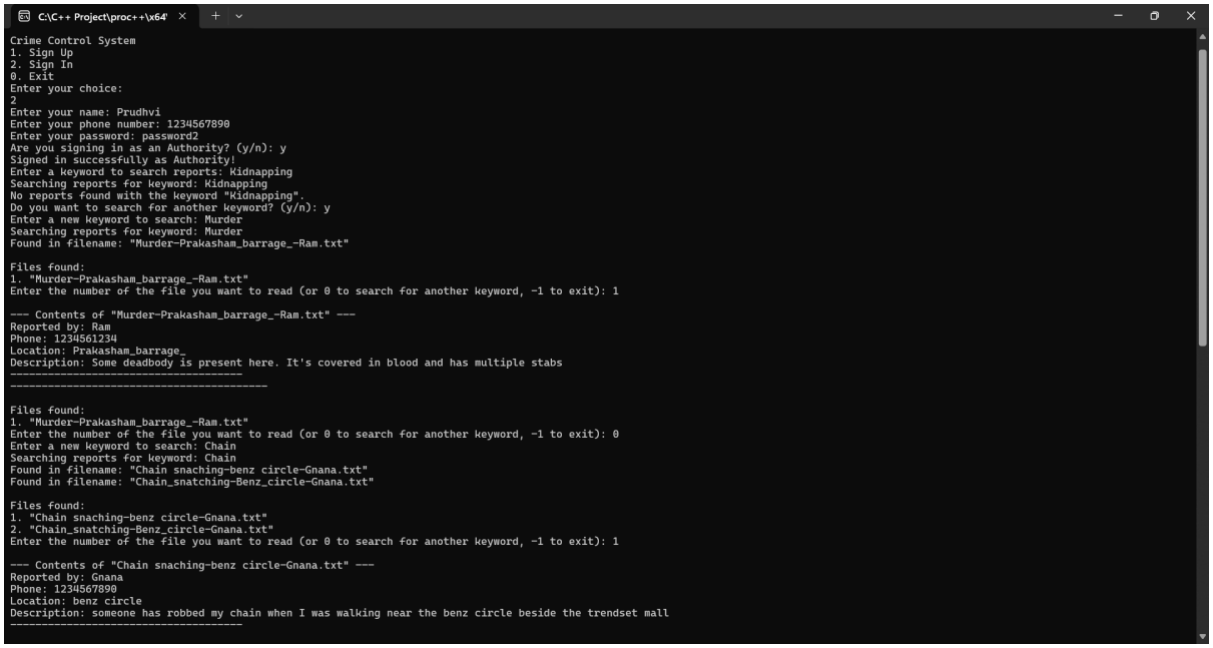
```
            addPublicUser(new Public(name, phone, password));
        }
    }

    void saveUsersToFile() const {
        ofstream file("public_users.txt");

        for (int i = 0; i < userCount; ++i) {
            file << publicUsers[i]->getName() << " "
                 << publicUsers[i]->getPhone() << " "
                 << publicUsers[i]->getPassword() << "\n";
        }
    }
};

// Helper function for validated input
static string getValidatedInput(const string& prompt, bool (*validate)(const string&)) {
    string input;

    while (true) {
        cout << prompt;
        cin >> input;

        if (validate(input)) {
            return input;
        } else {
            cout << "Invalid input! Please try again.\n";
        }
    }
}

// Sign-Up function
static void signUp(PublicUserDatabase& db) {
    string name = getValidatedInput("Enter your name: ", User::isValidName);
    string phone = getValidatedInput("Enter your phone number: ", User::isValidPhone);
    string password = getValidatedInput("Enter your password: ", User::isValidPassword);

    // Check if the user already exists
    Public* existingUser = db.findPublicUser(name, phone, password);

    if (existingUser) {
        cout << "User already exists! Please sign in instead.\n";
        return;
    }
```

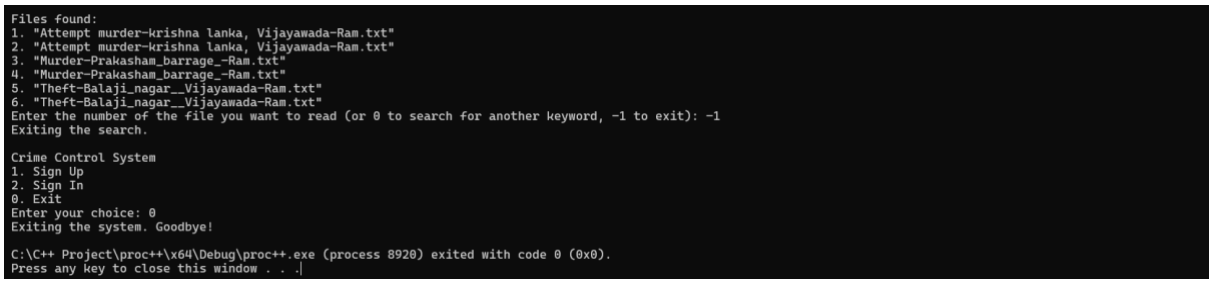
```
    // Add new user and save to file
    db.addPublicUser(new Public(move(name), move(phone), move(password)));
    cout << "Public user signed up successfully!\n";
}

// Sign-In function
static void signIn(PublicUserDatabase& db, Authority* authorityUsers, int authorityCount) {
    string name = getValidatedInput("Enter your name: ", User::isValidName);
    string phone = getValidatedInput("Enter your phone number: ", User::isValidPhone);
    string password = getValidatedInput("Enter your password: ", User::isValidPassword);

    char choice;
    cout << "Are you signing in as an Authority? (y/n): ";
    cin >> choice;
    cin.ignore();

    if (tolower(choice) == 'y') {
        for (int i = 0; i < authorityCount; ++i) {
            if (authorityUsers[i].getName() == name &&
                authorityUsers[i].getPhone() == phone &&
                authorityUsers[i].getPassword() == password) {
                cout << "Signed in successfully as Authority!\n";

                string keyword;
                cout << "Enter a keyword to search reports: ";
                getline(cin, keyword);

                authorityUsers[i].searchReports(keyword);
                return;
            }
        }
        cout << "Invalid credentials for Authority.\n";
    } else {
        Public* user = db.findPublicUser(name, phone, password);

        if (user) {
            cout << "Signed in successfully as Public user!\n";
            user->reportIncident();
        } else {
            cout << "Invalid credentials for Public user.\n";
        }
    }
}
```

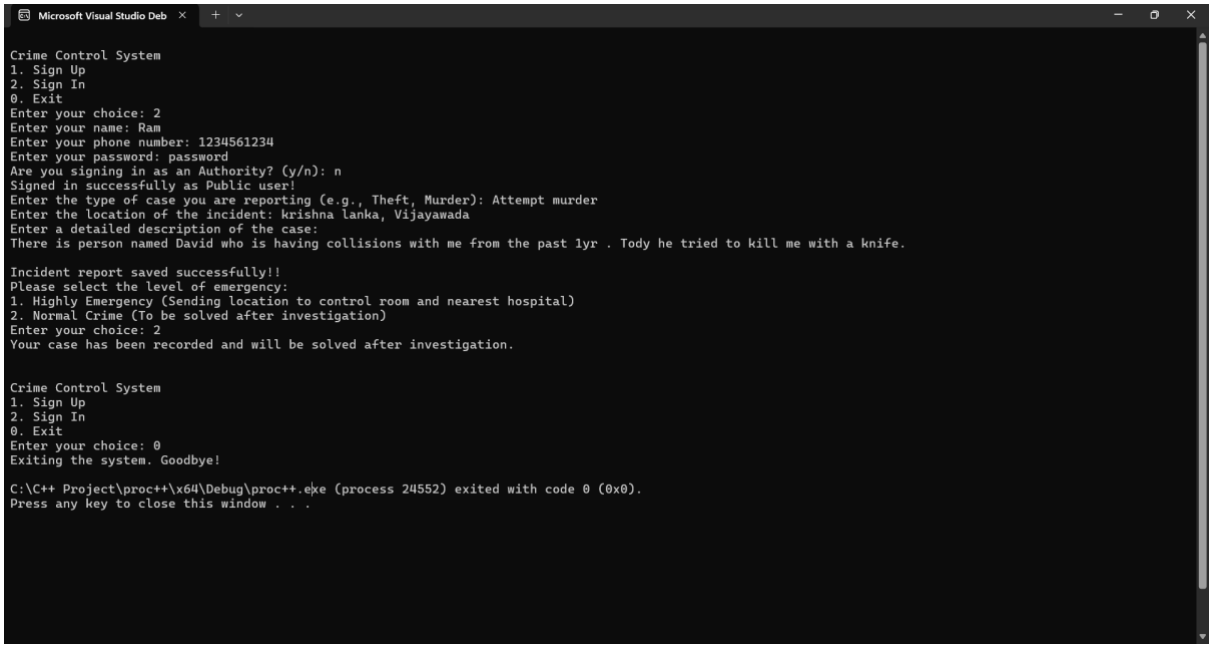
```
// Main Function
int main() {
    PublicUserDatabase publicDb;

    Authority authorityUsers[] = {
        Authority("Dr.Kavatha Rani", "1234512345", "password1"),
        Authority("Prudhvi", "1234567890", "password2"),
        Authority("Navnith", "6789067890", "password3"),
        Authority("Kishore", "0987654321", "password4"),
        Authority("Adithya", "2345678910", "password5")
    };

    int authorityCount = sizeof(authorityUsers) / sizeof(authorityUsers[0]);
    int choice;

    do {
        cout << "\nCrime Control System\n";
        cout << "1. Sign Up\n";
        cout << "2. Sign In\n";
        cout << "0. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            signUp(publicDb);
            break;
        case 2:
            signIn(publicDb, authorityUsers, authorityCount);
            break;
        case 0:
            cout << "Exiting the system. Goodbye!\n";
            break;
        default:
            cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 0);

    return 0;
}
```

2.2.1 Test Cases

Major Test Cases







```
Enter a new keyword to search: Ram
Searching reports for keyword: Ram
Found in filename: "Attempt murder-krishna lanka, Vijayawada-Ram.txt"
Found in content of: "Attempt murder-krishna lanka, Vijayawada-Ram.txt"
Found in filename: "Murder-Prakasham_barrage_-Ram.txt"
Found in content of: "Murder-Prakasham_barrage_-Ram.txt"
Found in filename: "Theft-Balaji_nagar__Vijayawada-Ram.txt"
Found in content of: "Theft-Balaji_nagar__Vijayawada-Ram.txt"

Files found:
1. "Attempt murder-krishna lanka, Vijayawada-Ram.txt"
2. "Attempt murder-krishna lanka, Vijayawada-Ram.txt"
3. "Murder-Prakasham_barrage_-Ram.txt"
4. "Murder-Prakasham_barrage_-Ram.txt"
5. "Theft-Balaji_nagar__Vijayawada-Ram.txt"
6. "Theft-Balaji_nagar__Vijayawada-Ram.txt"
Enter the number of the file you want to read (or 0 to search for another keyword, -1 to exit):
```

```
Microsoft Visual Studio Deb

Crime Control System
1. Sign Up
2. Sign In
0. Exit
Enter your choice: 1
Enter your name: Ram
Enter your phone number: 1234561234
Enter your password: password
Public user signed up successfully!

Crime Control System
1. Sign Up
2. Sign In
0. Exit
Enter your choice: 0
Exiting the system. Goodbye!

C:\C++ Project\proc++\x64\Debug\proc++.exe (process 21380) exited with code 0 (0x0).
Press any key to close this window . . .
```

Error-based Test Cases

(a) Invalid input in the menu section

```
Crime Control System
1. Sign Up
2. Sign In
0. Exit
Enter your choice: 4
Invalid choice. Try again.
```

(b) Invalid ‘Name’ entry

```
C:\Crime Control\proc++\x6

Crime Control System
1. Sign Up
2. Sign In
0. Exit
Enter your choice: 1
Enter your name: Ram2
Invalid input! Please try again.
Enter your name:
```

(c) Invalid ‘Phone Number’ entry

```
Enter your phone number: 1234567891
Invalid input! Please try again.
Enter your phone number:
```

(d) Invalid ‘Password’ entry (Less than 8 characters)

```
Enter your password: 1234567
Invalid input! Please try again.
Enter your password:
```

(e) User does not exist

2.3 Concepts Incorporated

2.3.1 Object-Oriented Programming (OOP)

1. Classes and Objects

- User, Authority, and Public classes represent entities in the system.
- Objects of these classes store and manage data for users.

2. Inheritance

- The Authority and Public classes inherit from the User base class.
- This demonstrates code reuse and extensibility.

3. Encapsulation

- Private and protected members (name, phone, password) restrict direct access, while public getter methods expose necessary data.

4. Polymorphism

- Demonstrated via the overridden behavior of functions like reportIncident() (specific to the Public class).

5. Dynamic Memory Allocation

- The PublicUserDatabase class uses dynamic memory for storing an array of pointers to Public objects.

6. Constructors and Destructors

- Constructors are used for initializing objects (e.g., User, Authority, and Public classes).
- The destructor of PublicUserDatabase handles cleanup and file saving.

2.3.2 Input Validation

1. Static Functions for Validation

- ValidName, isValidPhone, and isValidPassword ensure data consistency.
- These are static because they operate independently of object instances.

2. Validation Loops

- Repeatedly prompt the user until valid input is provided.

2.3.3 File Handling

1. **Reading from Files**
- The Authority::searchReports function reads file content to locate keywords.

2. **Writing to Files**
- The Public::reportIncident function saves incident reports to uniquely named files.

3. **File Existence Checks**
- The std::filesystem library ensures directories and files are checked before processing.

2.3.4 User Input and Output

1. **Console Interaction**
- Prompts and reads user input using std::cin and std::cout.

2. **Menu Navigation**
- Loops provide a consistent interface to handle user choices.

3. **Formatted Output**
- The code displays results and report content in a user-friendly way.

2.3.5 Dynamic Memory Management

1. **Pointers and Arrays**
- The PublicUserDatabase uses a dynamically allocated array of pointers to store user data.

2. **Resizing**
- The resize method ensures the array expands as needed.

3. **Manual Cleanup**
- The destructor ensures no memory leaks occur by deleting all allocated objects.

2.3.6 Namespaces

- The std namespace is used throughout for standard library functions and objects.
- The std::filesystem namespace (fs) is aliased for handling file and directory operations.

2.3.7 STL and Algorithms

1. **String Operations**
- std::all_of and std::find are used for validation and keyword searches.

2. **Vector**
- A std::vector stores search results dynamically in searchReports.

3. **Iterators**
- Used with STL algorithms to iterate over collections like files.

2.3.8 Error Handling

1. **File Open Checks**
- The code ensures file streams are valid before processing them.

2. **Directory Checks**
- std::filesystem checks prevent errors from non-existent directories.

3. **Invalid Input Handling**
- Input validation ensures the program doesn't crash due to bad data.

2.3.9 Loops and Conditional Statements

1. **While Loops**
- Used in the main menu, validation loops, and file selection.

2. **If-Else and Switch**
- Used for decision-making, such as determining the level of emergency or processing user roles.

2.3.10 Standard Library Functions

1. **std::move**
- Used in constructors to avoid unnecessary copies when initializing strings.

2. **std::string**
- Strings are used extensively for input handling, validation, and report generation.

3. **std::ifstream and std::ofstream**
- File input/output operations.

4. **std::cin.ignore()**
- Clears the input buffer before accepting multi-line input.

2.3.11 Advanced Features

1. **Namespaces**
- fs alias for std::filesystem.

2. **File Iteration**
- std::filesystem::directory_iterator iterates over files in a directory.

3. **Dynamic File Naming**
- Unique filenames are created based on user input, improving system organization.

2.3.12 Modular Design

1. **Functions**
- Helper functions like getValidatedInput encapsulate repeated logic.

2. **Separation of Concerns**
- Different classes handle distinct responsibilities:
  - Authority for search operations.
  - Public for reporting incidents.
  - PublicUserDatabase for managing user data.

2.3.13 Code Scalability

1. **Reusable Components**
- Validation, file handling, and report generation are modular and extensible.

2. **Authority List**
- Stored as an array, enabling easy addition of more authorities.

3 Concluding Remarks

3.1 Summary of Key Takeaways

3.1.1 Empowering Users

1. **Classes and Objects**
- The dual-role structure (Public and Authority) ensures inclusivity, enabling citizens to report incidents while granting officials the tools to act swiftly.

2. **Efficiency Through Automation**
- Key features like emergency alerts, detailed report generation, and keyword-based search simplify traditionally cumbersome processes.

3. **Security and Accessibility**
- The system underscores the importance of secure user validation and accessible interfaces to encourage widespread adoption.

3.1.2 Importance of Collaboration

This project highlights that public safety is not the sole responsibility of authorities but a collective effort. Systems like these demonstrate how collaboration between technology and community can achieve significant results.

3.1.3 Call to Action

The Incident Reporting System is more than a project—it's a call to action for a safer tomorrow. Let's work together to implement solutions that make reporting accessible and ensure that no voice goes unheard in times of need.

4 Future Work

4.1 Enhancing User Experience

4.1.1 Multilingual Support

- Integrate support for multiple languages to cater to diverse user bases and break communication barriers.

4.1.2 Voice and Chatbox Assistance

- Implement AI-driven voice recognition and chatbot functionality for hands-free reporting and immediate assistance.

4.2 Expanding Functional Capabilities

4.2.1 Real-Time Location Tracking

- Add GPS integration to pinpoint incident locations accurately and provide live updates to authorities.

4.2.2 Integration with National Emergency Systems

- Collaborate with government databases and control rooms to streamline emergency responses.

4.2.3 Advanced Search Tools

- Include filters for date, location, and case type to improve report retrieval for authorities.

4.3 Leveraging Technology

4.3.1 AI-Powered Insights

- Deploy machine learning to analyze incident patterns, predict high-risk areas, and prevent crimes proactively.

4.3.2 Blockchain for Data Integrity

- Ensure tamper-proof storage of sensitive reports to maintain trust and transparency.

4.3.3 Cloud-Based Systems

- Transition to a cloud infrastructure for real-time synchronization and scalability across multiple regions.

4.4 Strengthening Security

4.4.1 Enhanced Authentication Mechanisms

- Adopt biometric verification (fingerprint or facial recognition) to bolster account security.

4.4.2 End-to-End Encryption

- Secure communication between users and the system to protect sensitive information.

4.5 Collaboration and Scalability

4.5.1 Collaboration with Healthcare and NGOs

- Partner with hospitals and non-governmental organizations for comprehensive emergency management.

4.5.2 Scalability Across Regions

- Adapt the system for use in rural areas, including offline modes for regions with limited connectivity.

4.6 Public Awareness Campaigns

4.6.1 Training and Outreach Programs

- Educate the public on how to use the system effectively through workshops, tutorials, and community outreach.

4.6.2 Gamification for Engagement

- Introduce gamified elements to encourage regular usage and familiarize users with the system's features.

Our ultimate goal is to evolve this system into a robust, universally adaptable platform that not only addresses incidents but actively prevents them. By leveraging advanced technologies, fostering collaborations, and prioritizing user-centric design, we aim to create a safer and more responsive world.

References

1. **Web Resources**

- GeeksforGeeks. "Object-Oriented Programming in C++." Available at: https://www.geeksforgeeks.org.
- TutorialsPoint. "C++ File Handling Tutorial." Available at: https://www.tutorialspoint.com.
- cppreference.com. "Standard C++ Library Reference." Available at: https://en.cppreference.com.

2. **Technology and Stack**

- Visual Studio Code. "Integrated Development Environment for C++ Programming."
- GCC Compiler Documentation. "GNU Compiler Collection Reference."

3. **Practical Applications and Case Studies**

- National Incident Reporting Systems. Case Studies and Implementation Models. Available at: https://www.nirs.gov.
- "Case Study: Incident Reporting at Scale in Urban Areas." Published by UrbanTech Solutions, 2021.

4. **Official Documentation**

- Microsoft Documentation. Available at: https://learn.microsoft.com

5. **Code Enhancements**

- ChatGPT – https://chat.openai.com