**TABLE OF CONTENTS**

1. **INTRODUCTION**
2. **REQUIREMENT ANALYSIS**
3. **SYSTEM DESIGN**
4. **IMPLEMENTATION**
5. **RESULTS & CONCLUSION**
6. **REFERENCES**
7. **APPENDIX**

**CHAPTER:1**

**INTRODUCTION**

**1.1 Definition**

This project is a password management system that securely stores and retrieves passwords for various websites. Users authenticate using a master password to manage their credentials. The system allows users to manually add passwords, retrieve stored passwords, and generate random secure passwords with a minimum length. Passwords are stored persistently in a file, and user authentication ensures security. Key features include data storage using an unordered map for quick access, file handling for persistence, and enforcing minimum password length requirements for stronger protection.

**1.2 Problem Definition**

The challenge is to design and implement a password management system that securely stores passwords for various websites. Users need to authenticate with a master password and should be able to add, retrieve, and generate passwords in a safe manner. Passwords must be stored securely in a file and the system should enforce certain constraints, such as a minimum password length.

**1.3 Objectives**

* To develop a password manager that securely stores and retrieves passwords.
* To ensure user authentication with a master password.
* To provide the ability to generate random passwords with customizable length.
* To enforce minimum password length to enhance security.
* To allow password data to persist across sessions by saving to and loading from a file.
* To implement error handling to manage file access issues and invalid input.

**1.4 Scope & Limitation**

**Scope:**

* Users can add, retrieve, and generate passwords for different websites.
* The system ensures that passwords meet a minimum length requirement of 8 characters.
* Passwords are stored in a text file and can be loaded during program startup.
* User authentication with a master password is required before managing passwords.

**Limitation:**

* The password manager stores passwords in a text file, which can be less secure than encrypted databases or vaults.
* No encryption is used for storing passwords, leaving stored passwords vulnerable to being read in plain text.
* There is no support for password updates or deletions.
* The system does not handle multi-user accounts, only supporting one master password.

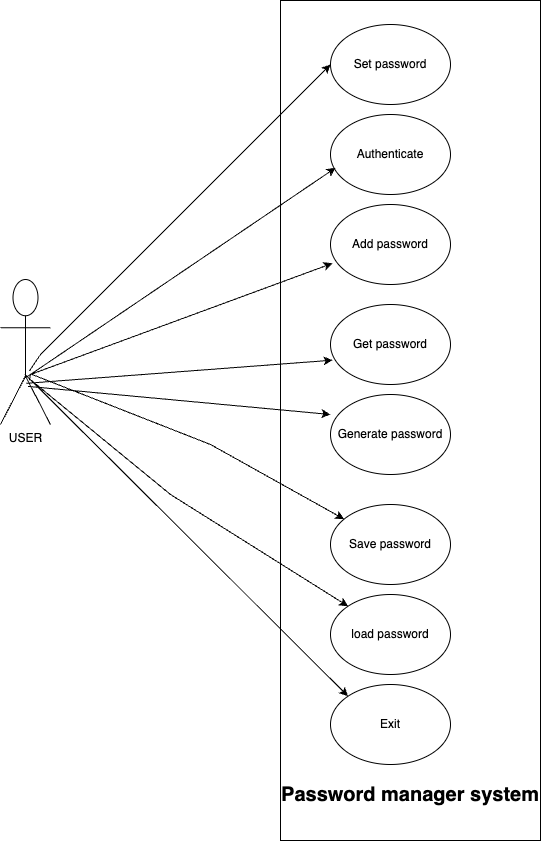
**CHAPTER 2**

**REQUIREMENT ANALYSIS**

**2.1 Functional Requirements**

* Users must be able to authenticate using a master password.
* The system must allow adding passwords to specific websites.
* The system should retrieve and display the stored password for a specified website.
* It should generate random passwords of a user-specified length (with a minimum of 8 characters).
* Use of efficient data structures like unordered\_map for quick access to stored passwords.

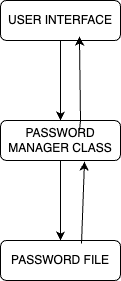
**2.2 Use Case diagram**



**CHAPTER 3**

**SYSTEM DESIGN**

**3.1 High level design in the system**



**3.2 Libraries / Headers used / IDE**

**Libraries/Headers used:**

**#include <iostream>:**

* **Purpose:** Provides functionality for input/output operations.
* **Use in the Program:** Used for displaying messages to the user and taking input via cin and cout.

**#include <string>**

* **Purpose:** Provides the string class for handling sequences of characters.
* **Use in the Program:** Used to store and manipulate strings such as website names, passwords, and the master password

**#include <unordered\_map>**

* **Purpose:** Provides the unordered\_map container, which stores key-value pairs and allows for fast lookup.
* **Use in the Program:** Used to store website-password pairs where the website is the key and the password is the value.

**#include <fstream>**

* **Purpose:** Provides file handling classes (ifstream, ofstream, fstream) for reading from and writing to files.
* **Use in the Program:** Used to read and write the password data to a text file (passwords.txt) for persistence.

**#include <ctime>**

* **Purpose:** Provides functions for handling date and time. Can be used to seed randomness or to log timestamps.
* **Use in the Program:** Although not directly used in this example, it is generally included when working with random or time-related functionality. In this case, random\_device is used for seeding the random number generator.

**#include <random>**

* **Purpose:** Provides random number generation utilities, including random\_device and the mt19937 engine.
* **Use in the Program:** Used to generate random, secure passwords.

**#include <stdexcept>**

* **Purpose:** Provides the runtime\_error class, which is used for handling runtime errors.
* **Use in the Program:** Used to throw exceptions when invalid operations occur, such as trying to add a password shorter than 8 characters or failing to access a file.

Integrated Development Environment (IDE): VS Code

**3.3 OOP Concepts used:**

**1) Classes and Objects**

**ClassDefinition:**  
The PasswordManager class defines a blueprint for a password management system, encapsulating all related functionality such as password generation, storage, and retrieval.

**ObjectCreation:**  
In the main() function, an instance of the PasswordManager class is created.

Example:

**PasswordManager manager(masterPwd);**

Here, manager is an **object** of the PasswordManager class, which gives access to all class members (methods and attributes).

**2) Encapsulation**

Encapsulation is the concept of bundling data (attributes) and methods that operate on the data within a class, and restricting direct access to some of the object’s components.

**PrivateMembers:**  
The program uses private members to ensure data hiding. For example, the passwords are stored in the private member passwords so that external code cannot access or modify them directly.

**PublicInterface:**  
The class provides public methods such as addPassword(), getPassword(), generateAndAddPassword(), and authenticate() to allow controlled access and modification of the data.

**3) Abstraction**

Abstraction means exposing only the necessary details to the outside world while hiding the implementation details.

In this program, the class methods such as addPassword(), getPassword(), and generatePassword() abstract the underlying logic of managing passwords, such as how passwords are stored, retrieved, or generated. The user does not need to know the internal implementation details; they just interact with the interface provided by the class.

**4) Constructor and Destructor**

**Constructor:**

The program uses a constructor to initialize the object of the PasswordManager class. The constructor takes the master password as input and loads any previously stored passwords from a file.

**Example:**

PasswordManager(const string &masterPwd) : masterPassword(masterPwd) {

loadFromFile(); // Load passwords from file at startup

}

This constructor initializes the masterPassword member and automatically calls loadFromFile() to load the password database into memory.

**5) Exception Handling**

Exception Handling is often considered part of robust OOP design. The program uses try-catch blocks and throws runtime\_error exceptions when there is an issue (like a file not being found or invalid input).

Example:

try {

manager.addPassword(website, password);

} catch (const runtime\_error &e) {

cerr << "Error: " << e.what() << endl;

}

The PasswordManager class handles specific error cases by throwing exceptions, while the main() function catches and handles those exceptions to display error messages to the user.

**CHAPTER 4**

**IMPLEMENTATION**

The functions are:

1. generatePassword(int length)

This function generates a random password of the specified length using a random character set containing numbers, uppercase and lowercase letters, and special symbols.

2. saveToFile()

This function saves the current passwords from the passwords map to a text file (passwords.txt) for persistence. It handles file errors using exception handling.

3. loadFromFile()

This function loads previously saved passwords from the passwords.txt file into the program's memory at the start. It handles missing or inaccessible files by providing an error message and initializing a fresh session.

4. addPassword(const string &website, const string &password)

This function allows the user to add a new password for a website to the passwords map. It also ensures that passwords have a minimum length of 8 characters before saving.

5. getPassword(const string &website)

This function retrieves a stored password for a given website. If the website does not exist in the map, it throws an exception.

6. generateAndAddPassword(const string &website, int length)

This function generates a random password for a website, then adds it to the password map using the addPassword function.

7. authenticate(const string &password)

This function authenticates the user by comparing the input password with the master password. It returns true if the passwords match.

8. Main function

The main function handles user interaction, authentication, and menu selection. It lets the user set a master password, then authenticate with it before performing any password management tasks.

**CHAPTER 5**

**RESULTS & CONCLUSION**

**Results:** The results of the password manager program demonstrate the successful implementation of a secure and functional password management system. Users can set a master password, which authenticates their access to the manager. Once authenticated, users can add, retrieve, and generate random secure passwords for different websites. The program ensures that passwords meet a minimum length requirement of eight characters, enforcing security best practices. Generated passwords include a mix of alphanumeric characters and special symbols, providing strong protection against password attacks. Additionally, passwords are saved to a file and loaded on subsequent use, ensuring persistence across sessions. Error handling mechanisms are in place, providing informative messages when issues arise, such as file access errors or invalid inputs. Overall, the program achieves its goal of securely managing passwords while maintaining ease of use through an intuitive menu-driven interface.

**Conclusion:** The program achieves its goal of providing a simple, secure password management system with functionality for adding, retrieving, and generating passwords. However, improvements such as encrypting stored passwords and enhancing the security of the master password would be necessary for use in real-world scenarios where higher levels of security are required.

**REFERENCES**

(No date a) Chatgpt. Available at: https://chatgpt.com/ (Accessed: 18 September 2024).

(No date b) R/CPP on reddit: Quality of life small improvements for every day C++ coding. Available at: https://www.reddit.com/r/cpp/comments/18ucvsc/quality\_of\_life\_small\_improvements\_for\_every\_day/ (Accessed: 18 September 2024).

Answered: Problem statement write a C++ class that...: 24ha (no date) 24HourAnswers. Available at: https://www.24houranswers.com/college-homework-library/Computer-Science/C-Family-Programming/17823 (Accessed: 18 September 2024).

GeeksforGeeks (2024) What is high level design – learn system design, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/what-is-high-level-design-learn-system-design/ (Accessed: 18 September 2024).

tTargiel (no date) TTARGIEL/opensesame-password-manager: OpenSESAME is a cross-platform password manager developed in C++ with GUI created in QML., GitHub. Available at: https://github.com/tTargiel/openSESAME-Password-Manager (Accessed: 18 September 2024).

**APPENDIX**

#include <iostream>

#include <string>

#include <map>

#include <fstream>

#include <ctime>

#include <cstdlib>

using namespace std;

class PasswordManager {

private:

map<string, string> passwords;

string masterPassword;

const string filename = "passwords.txt";

// Function to generate a random password with at least 8 characters

string generatePassword(int length) {

if (length < 8) {

length = 8; // Ensure password length is at least 8

}

const char charset[] =

"0123456789"

"ABCDEFGHIJKLMNOPQRSTUVWXYZ"

"abcdefghijklmnopqrstuvwxyz"

"!@#$%^&\*()";

string password;

bool hasUpper = false, hasLower = false, hasDigit = false, hasSpecial = false;

while (!(hasUpper && hasLower && hasDigit && hasSpecial)) {

password.clear();

hasUpper = hasLower = hasDigit = hasSpecial = false;

for (int i = 0; i < length; i++) {

char ch = charset[rand() % (sizeof(charset) - 1)];

password += ch;

if (isupper(ch)) hasUpper = true;

else if (islower(ch)) hasLower = true;

else if (isdigit(ch)) hasDigit = true;

else hasSpecial = true;

}

}

return password;

}

// Save passwords to file

void saveToFile() {

ofstream file(filename);

if (file.is\_open()) {

for (const auto& pair : passwords) {

file << pair.first << ":" << pair.second << endl;

}

file.close();

} else {

cerr << "Unable to open file for writing." << endl;

}

}

// Load passwords from file

void loadFromFile() {

ifstream file(filename);

if (file.is\_open()) {

string line;

while (getline(file, line)) {

size\_t pos = line.find(':');

if (pos != string::npos) {

string website = line.substr(0, pos);

string password = line.substr(pos + 1);

passwords[website] = password;

}

}

file.close();

} else {

cerr << "Unable to open file for reading." << endl;

}

}

public:

PasswordManager(const string &masterPwd) : masterPassword(masterPwd) {

srand(time(0)); // Seed random number generator

loadFromFile(); // Load saved passwords at start

}

// Add a new password

void addPassword(const string &website, const string &password) {

passwords[website] = password;

saveToFile(); // Save changes

}

// Retrieve a password

string getPassword(const string &website) {

return passwords[website];

}

// Generate and add a new password

void generateAndAddPassword(const string &website, int length) {

string newPassword = generatePassword(length);

addPassword(website, newPassword);

cout << "Generated password for " << website << ": " << newPassword << endl;

}

// Authenticate user

bool authenticate(const string &password) {

return password == masterPassword;

}

};

int main() {

string masterPwd;

cout << "Set your master password: ";

cin >> masterPwd;

PasswordManager manager(masterPwd);

string inputPwd;

cout << "Enter master password to access the manager: ";

cin >> inputPwd;

if (manager.authenticate(inputPwd)) {

cout << "Authentication successful.\n";

while (true) {

cout << "\nMenu:\n";

cout << "1. Add Password\n";

cout << "2. Get Password\n";

cout << "3. Generate and Add Password\n";

cout << "4. Exit\n";

cout << "Enter choice: ";

int choice;

cin >> choice;

if (choice == 4) break;

string website;

string password;

switch (choice) {

case 1:

cout << "Enter website: ";

cin >> website;

cout << "Enter password: ";

cin >> password;

manager.addPassword(website, password);

cout << "Password added.\n";

break;

case 2:

cout << "Enter website: ";

cin >> website;

cout << "Password for " << website << ": " << manager.getPassword(website) << endl;

break;

case 3:

int length;

cout << "Enter website: ";

cin >> website;

cout << "Enter password length (8 or greater): ";

cin >> length;

manager.generateAndAddPassword(website, length);

break;

default:

cout << "Invalid choice.\n";

break;

}

}

} else {

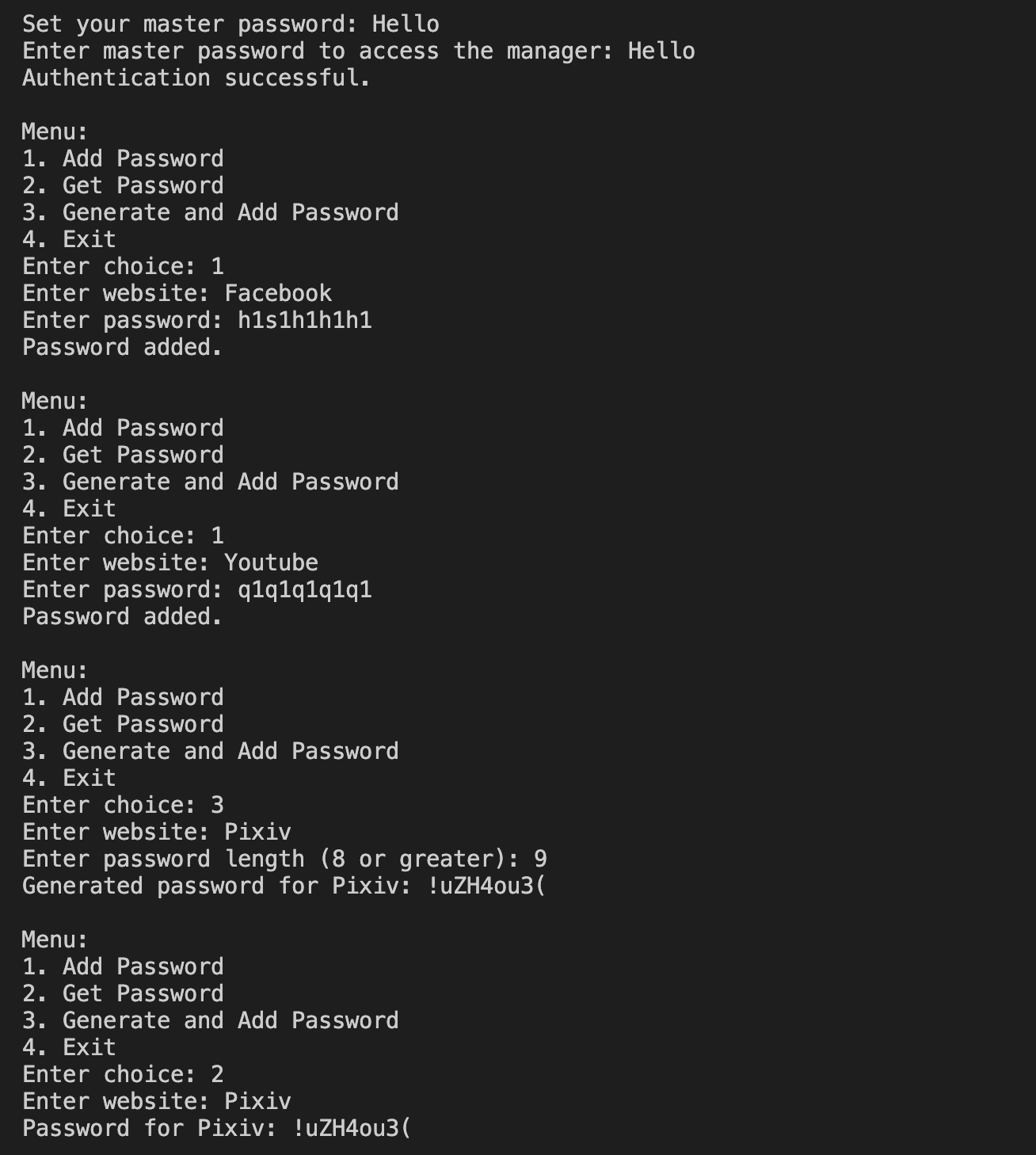
cout << "Authentication failed.\n";

}

return 0;

}

**Output:**

****

.

