B Tech Computer Science and Engineering (Data Science) -311 Program

Semester – II

Academic Year 2023-24

Object Oriented Programming and Design

Mini Project

Notes and Password Manager

Developed by

L024 – Ishaan Sarode

L025 – Nidhi Shah

L030 – Avni Tapadiya

L034 – Devansh Kala

Faculty In charge

Dr. Dhirendra Mishra

Dr. Priteegandha Naik

SVKM's NMIMS

Mukesh Patel School of Technology Management and Engineering

Vile Parle West Mumbai-56

TABLE OF CONTENTS

1. ABSTRACT

Strong note and password management systems are now essential in the digital age because personal and professional information is now shared across multiple platforms. Password managers and notepads are essential tools for managing and preserving private information. They provide safe places to store a variety of data, including financial records, project summaries, login credentials, and meeting minutes. These managers protect data confidentiality by using encryption techniques, which reduces the possibility of identity theft and unauthorized access.

Additionally, note and password managers boost efficiency by enabling easy access to data stored on various devices. It is simple for users to update and retrieve their data, which promotes productivity in both personal and professional contexts. Data management procedures are further streamlined by the inclusion of features like version control and synchronization in many contemporary managers. These tools also encourage information sharing and teamwork between individuals and groups. Managers enable users to securely share sensitive data with trusted parties and work together on projects by enabling controlled access and permission settings. These tools are still necessary in an increasingly connected world to protect privacy and boost productivity as technology advances.

2. INTRODUCTION

Keeping track of a never-ending list of passwords and sensitive data in this modern digital world can be difficult, and potentially risky for a user's security.

To overcome this difficulty, a Password and Notes Manager has been created, by leveraging the power of object-oriented programming (OOP) concepts. This methodology cultivates a logical and modular design, guaranteeing that every constituent operates autonomously while exhibiting smooth interoperability. This results in a number of important advantages. The security of the users' data is the top priority in the Password and Notes Manager. It uses strong encryption techniques to prevent unauthorized access to the users' private data. Because users' passwords and notes are protected by top-notch security safeguards, one can keep them with total confidence.

But not only security, inherent flexibility are its main advantages. The Password and Notes Manager is easily customizable to meet changing requirements by design. The system is designed to easily integrate new features and functionalities without interfering with existing ones, as one's data management needs change. This guarantees that the data management system will continue to be relevant and useful for many years to come. Above all, great care has been taken to ensure that the user interface is clear and simple to use. It is quite easy to find the exact note or password users need, making it very easy to search through users' data.

3. SCOPE OF PROJECT - SOCIETAL NEED FOR THIS PROJECT

These days, with so many digital platforms available, a safe data management solution is essential. This project addresses the problem directly by offering a simple, safe, offline way to store login credentials and private notes. It gives trust in a time when the use of digital technologies is growing by directly addressing growing concerns about online security and privacy.

Societal Need:-

The following are the societal needs of the Notes and Password Manager created:
Notes Managers:

- Easily arrange and retrieve information.
- Assure information preservation and accessibility.

Password managers:

- Easily and safely keep track of passwords.
- Preserve the privacy of users.

4. ORGANIZATION OF THE PROJECT

After an incident leaves her trapped in a cave, Emma turns to contemporary technology for comfort in recalling crucial facts. To manage notes and passwords, her friend had her register for an account on the Notes and Password Manager application. Upon launching the application, she is prompted to authenticate herself. If she were a new user, she would have to create an account with a username and password. Since she already has an account, she is asked to log in with her pre-existing credentials. Upon successful authentication, she is logged into her account and gains access to the Notes and Password Manager.

After logging in, she finds herself at the main menu where she can access the two main features: the Notes Manager and the Password Manager. These managers offer user-friendly methods for organizing and securing information, making life easier for users like Emma. She is shown three different tools if she chooses the Notes Manager. She has the option to add or remove notes (which requires her to input the note's index in order to do so), see previously added notes that contain important information, or all three. If she is done working on her notes, she can then choose to close the Notes Manager.

She is presented with the same three alternatives, but for passwords, if she selects the Password Manager, she can view her previous passwords; add a password (where she enters the website; her username on the website, and the password); or remove a password (which also requires an index to be deleted). She can also close the Password Manager, just like she can with the Notes Manager.

She can close the Notes and Password Manager program once she has finished all of her intended tasks.

This report goes over the research question, problem statement. It also goes through UML Diagrams, the final code and its outputs. Following it, the list of errors, future scope of improvements and references used for the Notes and Password Manager are discussed.

5. **RESEARCH QUESTION**

How can one develop a combined note and password manager system, which uses file I/O for storage of data and incorporates user authentication and login, affecting user productivity and security?

6. **BLOCK DIAGRAM OF ALL FUNCTIONALITIES**

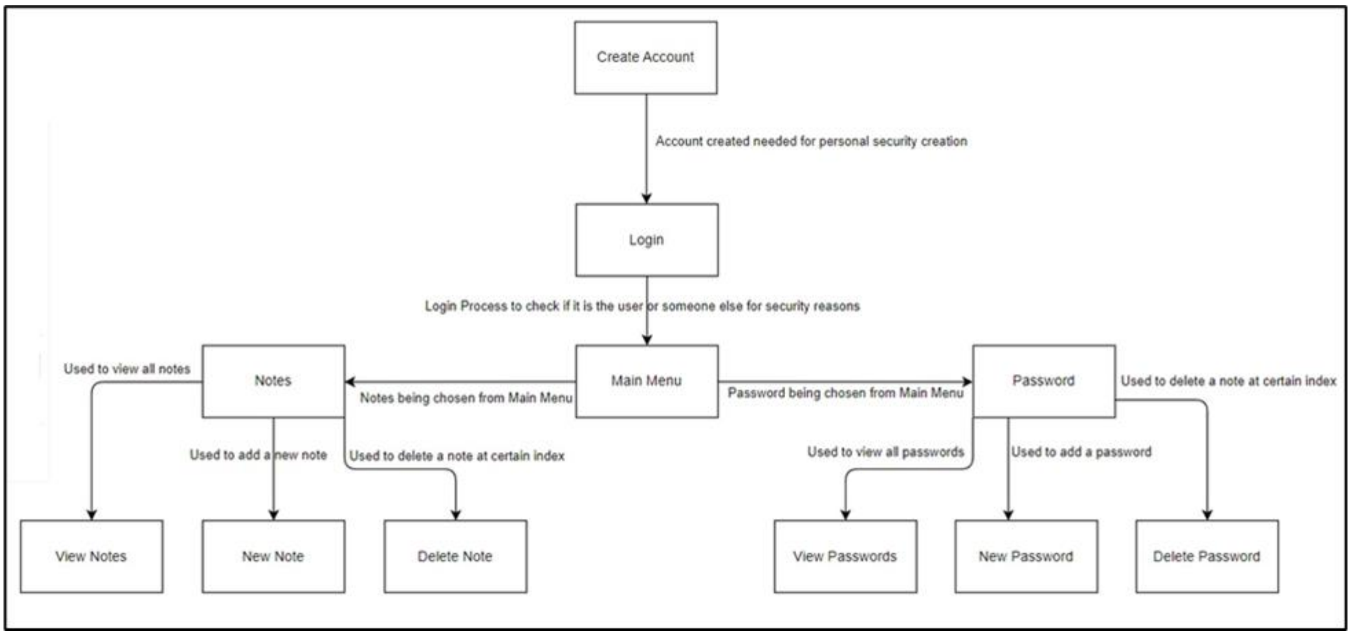

Create Account
Account created needed for personal security creation
Login
Login Process to check if it is the user or someone else for security reasons
Used to view all notes
Notes
Notes being chosen from Main Menu
Main Menu
Password being chosen from Main Menu
Password
Used to delete a note at certain index
Used to add a new note
Used to delete a note at certain index
Used to view all passwords
Used to add a password
View Notes
New Note
Delete Note
View Passwords
New Password
Delete Password

Figure 6.1 Block Diagram of Functionalities

Block diagrams are used to visualize the functional view of a system. The above figure shows the flow and the process the program will go through during execution; from Create Account to the options from each menu.

7. **PROBLEM STATEMENT**

As the number of websites continue to grow, managing a multitude of passwords and their associated logins has become a necessity. On the other hand, many users are cautious due to the growing frequency of data breaches involving online note-taking and password management services. Our program provides a safe, offline way for users to keep passwords and crucial notes on their cellphones in response to these worries. To allay concerns about privacy violations and data breaches, we offer a secure substitute for conventional online services.

8. LIST OF CLASSES AND ITS MEMBERS PLANNED

User:

Member Functions –

i. loginInput()
ii. loginProcess()
iii. loginOutput()

- Authenticator:

Member Functions –

i. createAccount()

- NotesManager

Data Members –

i. numNotes: string
ii. notes[]: string

Member Functions –

i. newNote()
ii. viewNote()
iii. deleteNote()
iv. saveNotesToFile()
v. loadNotesFromFile()

- Passwords

Data Members –

i. numPasswords: int
ii. website: string
iii. username: string
iv. password: string

Member Functions –

i. newPassword()
ii. viewPassword()
iii. deletePassword()
iv. savePasswordsToFile
v. loadPasswordsFromFile

9. USE CASE

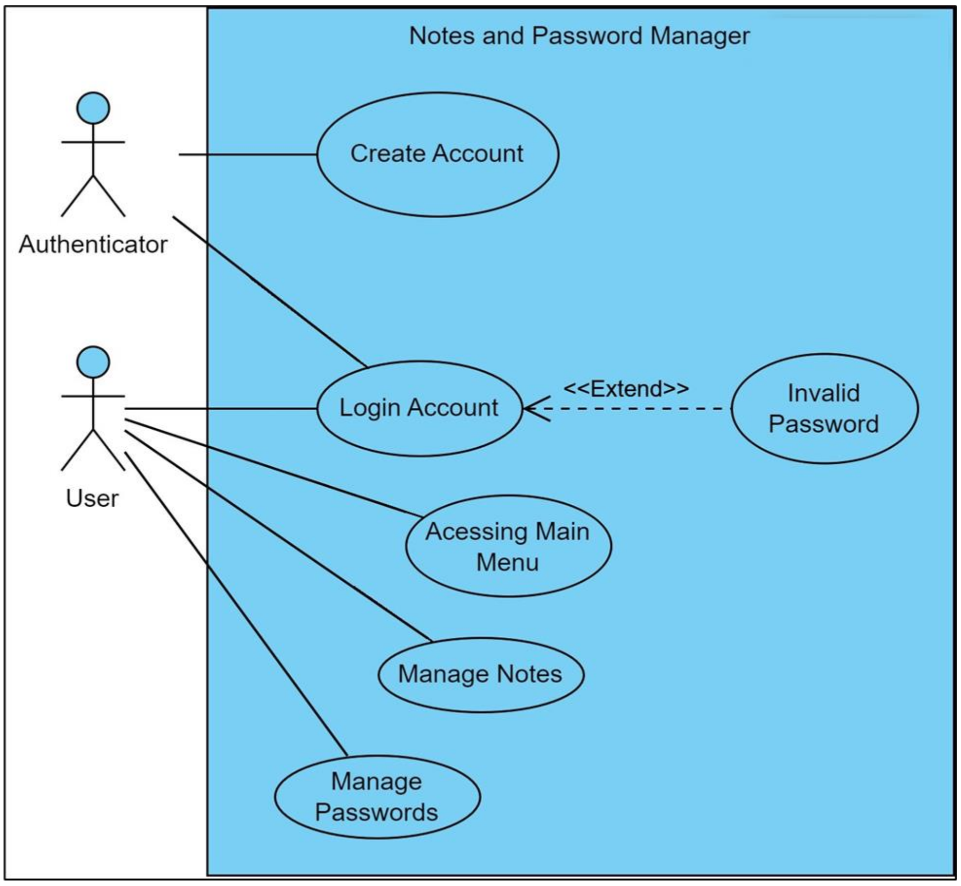


Figure 9.1 Use Case Diagram

Use Case diagrams are a type of a UML diagram used to describe the interactions between different external entities (actors) and the system.

In Figure 9.1, the Use Case diagram shows a user, and an authenticator, as actors. The Authenticator is an actor which is associated with 2 functions – Create Account and Login Account. The User is an actor associated with 4 functions – Login Account, Accessing Main Menu, Manage Notes and Manage Passwords.

The Authenticator uses the functionality "Create Account" to create the account for a new user, and then uses the "Login" functionality to log them in. The User uses the functionality "Login" to login to his account, and see the main menu, using the functionality "Accessing Main Menu". During the login, if the user enters the wrong username or password, he is sent back to the Login functionality, till the correct username and passwords are entered. The User can also manage his Notes and Passwords using the functionalities "Manage Notes" and "Manage Passwords".

10. **CLASS DIAGRAM**

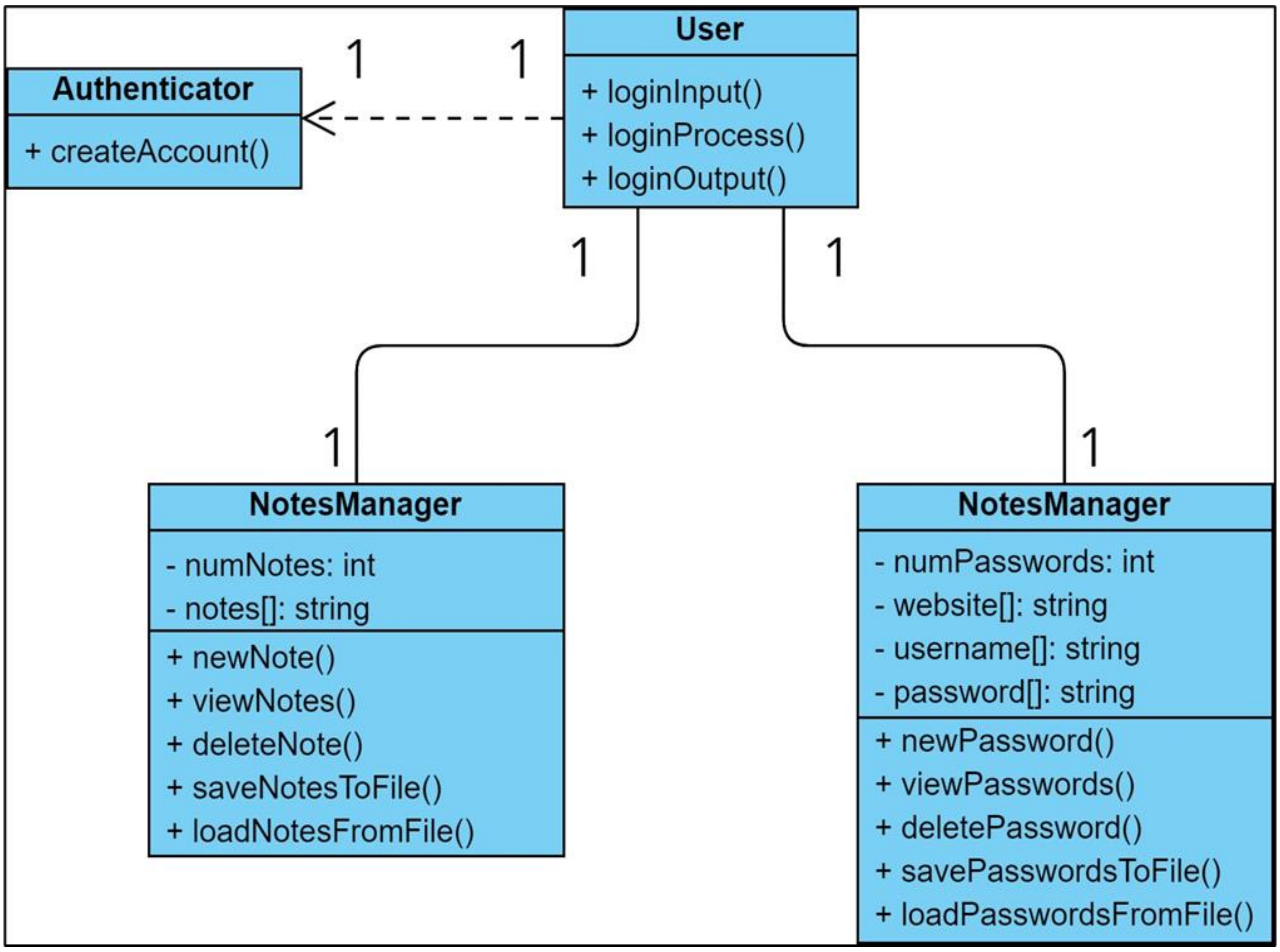


Figure 10.1 Class Diagram

One kind of UML diagram is class diagrams, which show the member functions and data members of a class as well as how they interact. The Notes and Password Manager's class diagram is shown in Figure 10.1. It displays four unique classes, each having its own member functions and data members.

User, Authenticator, Notes, and Password are the four classes.

Authenticator: This class is in charge of establishing new user accounts. Its createAccount function asks the user to provide a username and password before saving the data to a file.

User: This class handles the user login process. It has methods for prompting the user for their login credentials (loginInput) and validating those credentials against the stored credentials (loginProcess). It also has a method for displaying login success or failure messages (loginOutput).

NotesManager: This class oversees the application's notes feature. It has methods for viewing notes (viewNotes), adding notes (addNote), deleting notes (deleteNote), and saving notes to a file (saveNotesToFile) and loading notes from a file (loadNotesFromFile).

PasswordManager: This class is in charge of managing passwords rather than notes. Passwords can be viewed using the viewPasswords function, added using addPassword, deleted using deletePassword, and stored to a file using savePasswordsToFile and loaded from a file using loadPasswordsFromFile.

11. ACTIVITY DIAGRAM

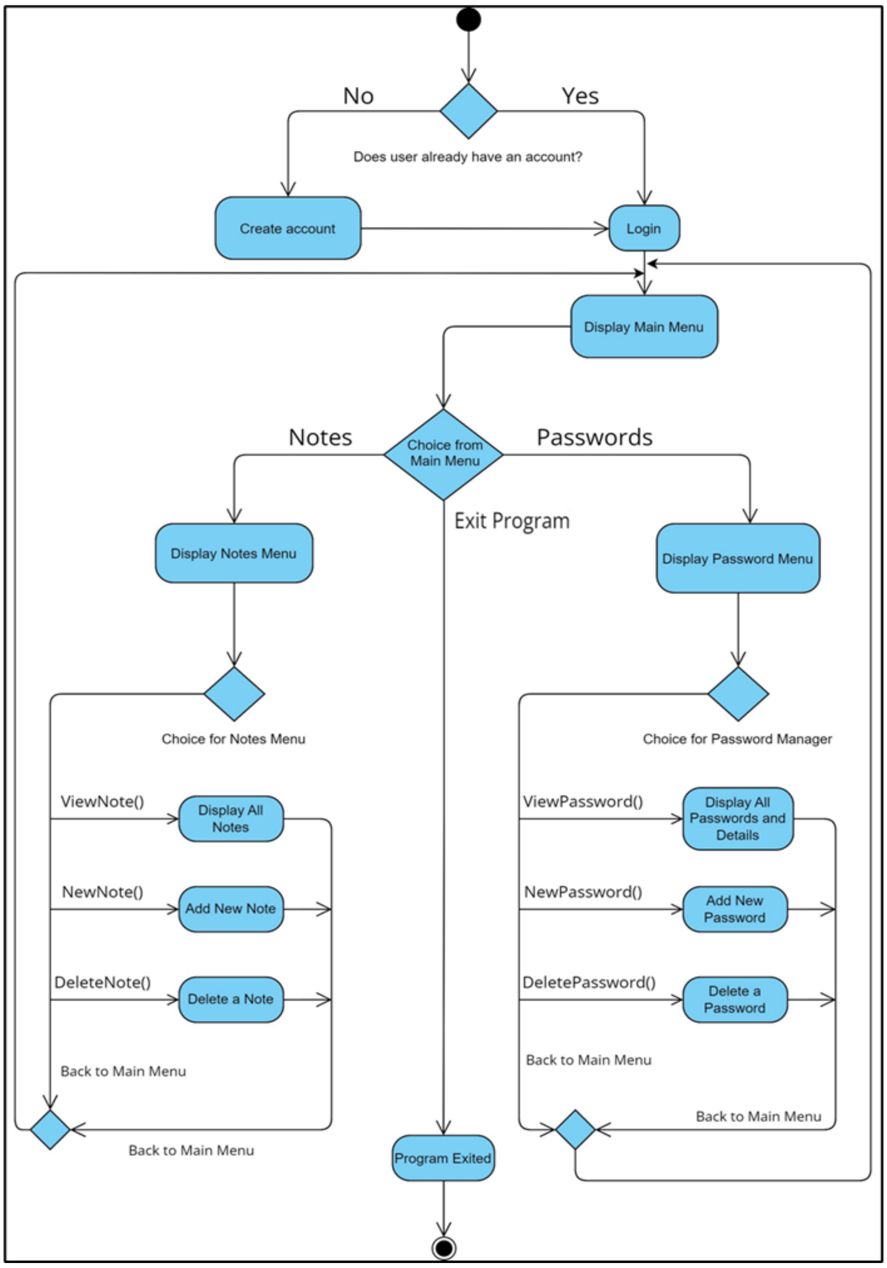


Figure 11.1 Activity Diagram of Overall Code

An activity diagram shows the decisions, actions, and sequential and parallel activities that make up a program. The Notes and Password Manager's general operation and workflow are shown in Figure 11.1.

The user is first prompted by the application to select whether to create a new account or log in to an already-existing one. The main menu appears with three options after a successful login: Notes, Passwords, and Exit Program.

The Notes option opens a submenu with three options: view notes, which shows all notes; add note, which lets you add new notes; and delete note, which lets you remove particular notes. Passwords have a sub-menu structure that is the same. The user is then brought back to the main menu, regardless of the function they selected.

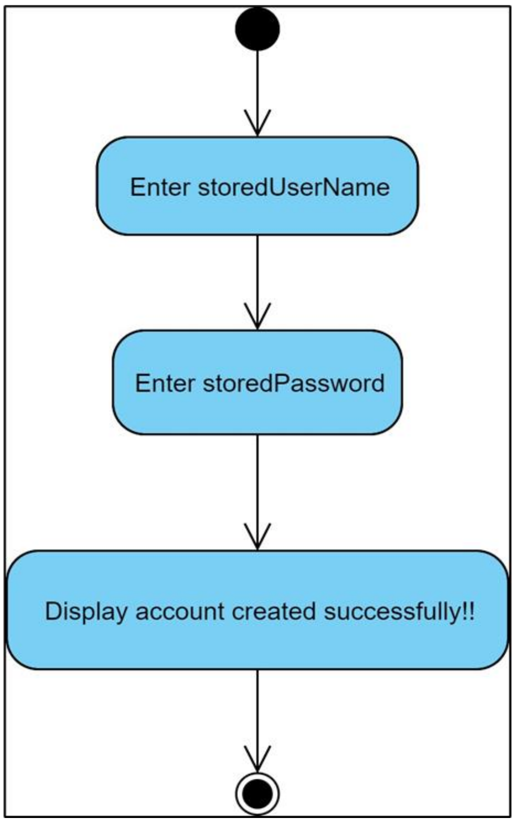


Figure 11.2 Activity Diagram of Account Creation

This Activity Diagram shows the process of account creation for the user. The user has to enter a storedUsername and storedPassword, after which his/her account is created successfully. The system then displays "Account Created Successfully!".

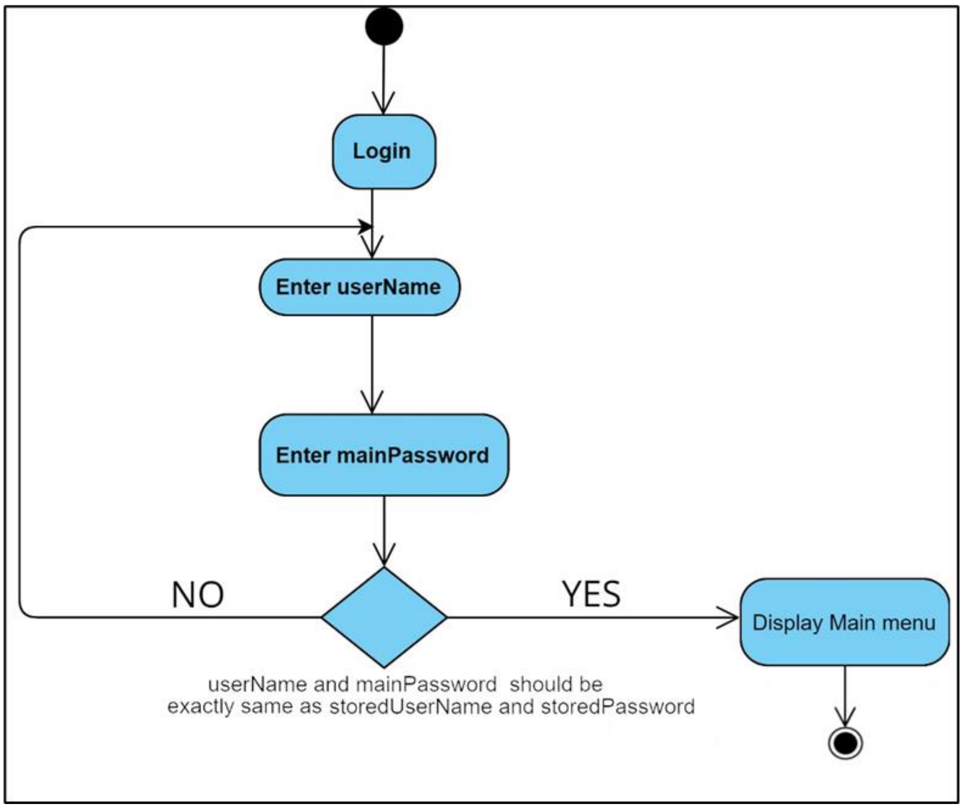

Login
Enter userName
Enter mainPassword
NO
YES
Display Main menu
userName and mainPassword should be exactly same as storedUserName and storedPassword

Figure 11.3 User login activity diagram

Figure 11.3 illustrates the user login process. The user must input the username and password for their current account, which the application will validate. If the user's information matches what was entered when the account was created, he will be able to securely log in. Otherwise, he/she will be sent back to the main menu to re-enter the credentials. This loop will continue till he/she gets them right.

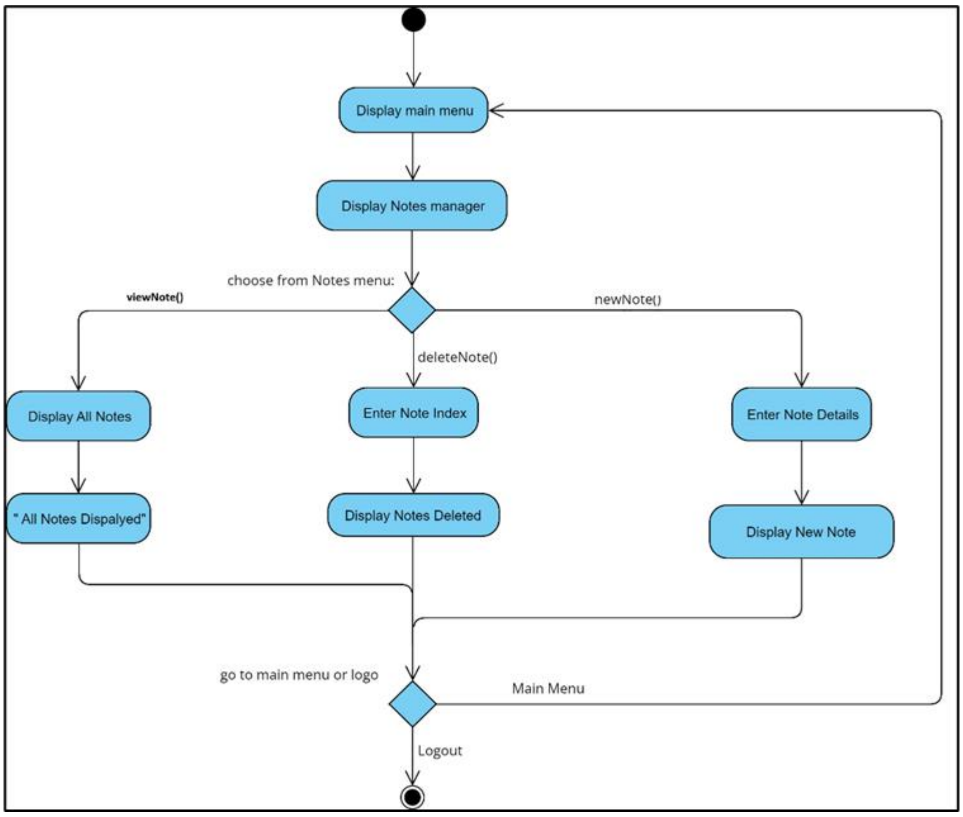


Figure 11.4 Activity Diagram of Notes

Figure 11.4 illustrates the activity diagram for the Notes, where upon choosing Notes from the Main Menu, the User has 3 options: to view the notes, delete a note or add a new note. If he/she chooses the view option, he/she will be shown all his notes. If he chooses to delete a note, he/she enters the index number of the note, and the note will be deleted from the system. For the addition of a new note, once he/she chooses the option, he will enter the information, and the new note will be added. After performing the desired operation, he/she has an option of either going to the main menu, or to logout.

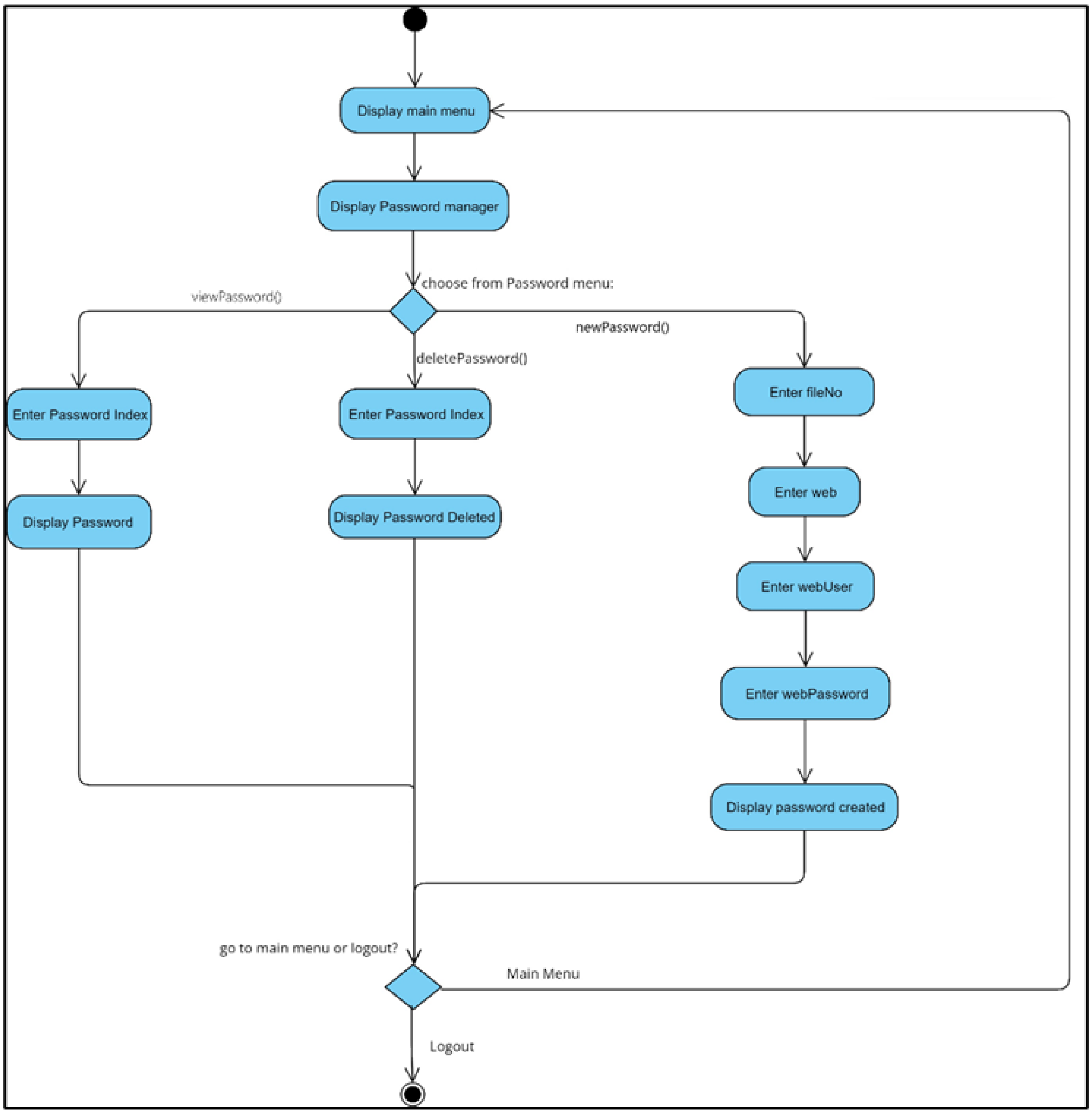


Figure 11.5 Activity Diagram for Passwords

Figure 11.5 illustrates the activity diagram for the Passwords, where upon choosing passwords from the Main Menu, the User has 3 options: to view the passwords, delete a note or add a new note. If he/she chooses the view option, he/she will be shown all his passwords. If he chooses to delete a password, he/she enters the index number of the password, and the password will be deleted from the system. For the addition of a new password, once he/she chooses the option, he will enter the information, and the new password will be added. After performing the desired operation, he/she has an option of either going to the main menu, or logout.

12. **SEQUENCE DIAGRAM**

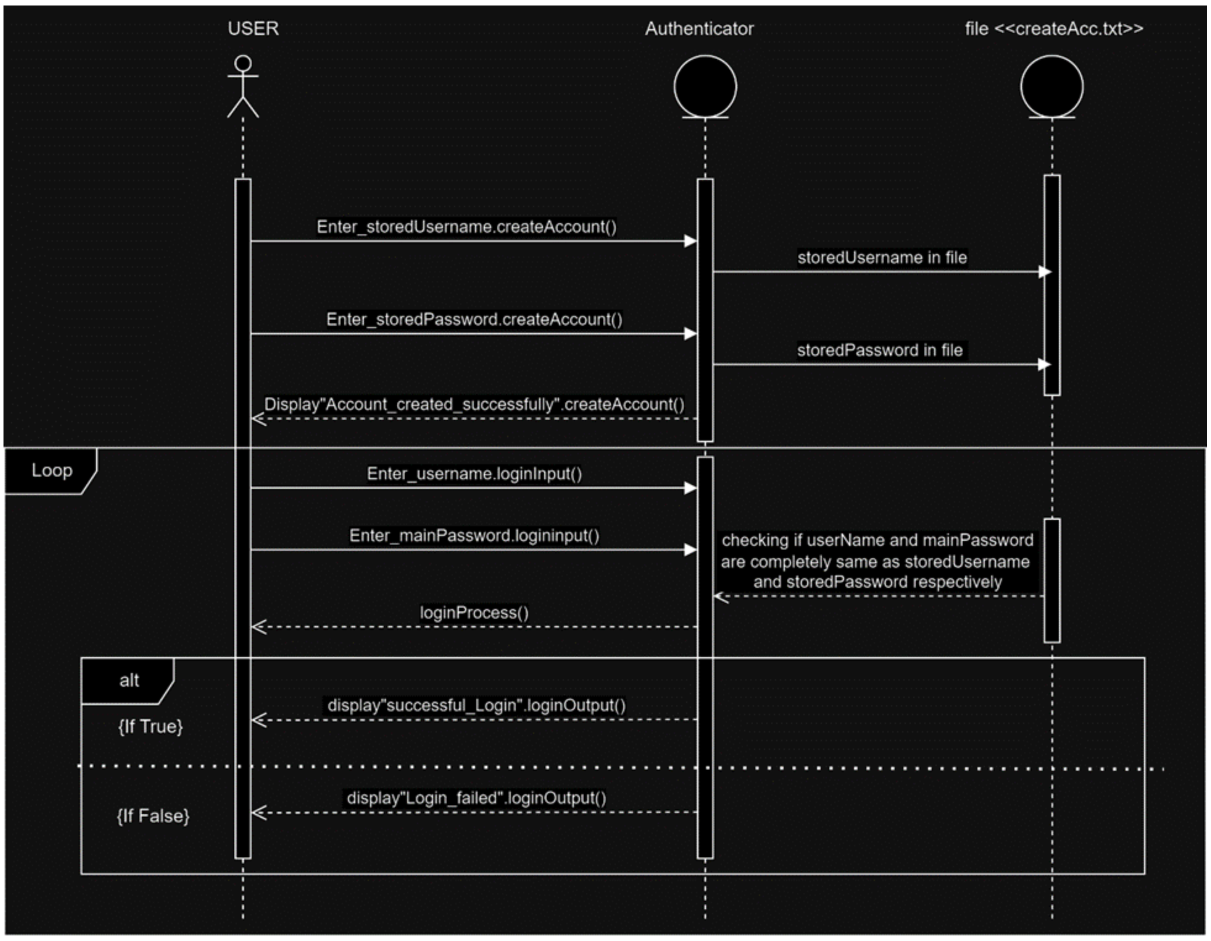

USER
Authenticator
file <<createAcc.txt>>
Enter_storedUsername.createAccount()
storedUsername in file
Enter_storedPassword.createAccount()
storedPassword in file
Display"Account_created_successfully".createAccount()
Loop
Enter_username.loginInput()
Enter_mainPassword.logininput()
checking if userName and mainPassword are completely same as storedUsername and storedPassword respectively
loginProcess()
alt
display"successful_Login".loginOutput()
{If True}
display"Login_failed".loginOutput()
{If False}

Fig. 12.1, Sequence Diagram from user's login and account creation

Sequence diagrams are a type of UML diagram which show the messages sent between the components of the system over time.

Here, fig 12.1 has three classes User, Authenticator and createAcc.txt which is a storing file. This fig shows how first the account will be created by the user if he doesn't have any account ,where the user has to add his details like "storedUsername" and "storedPassword" after which the account will be created successfully. If the user already has an account then he has to enter the details of the account in order to login.

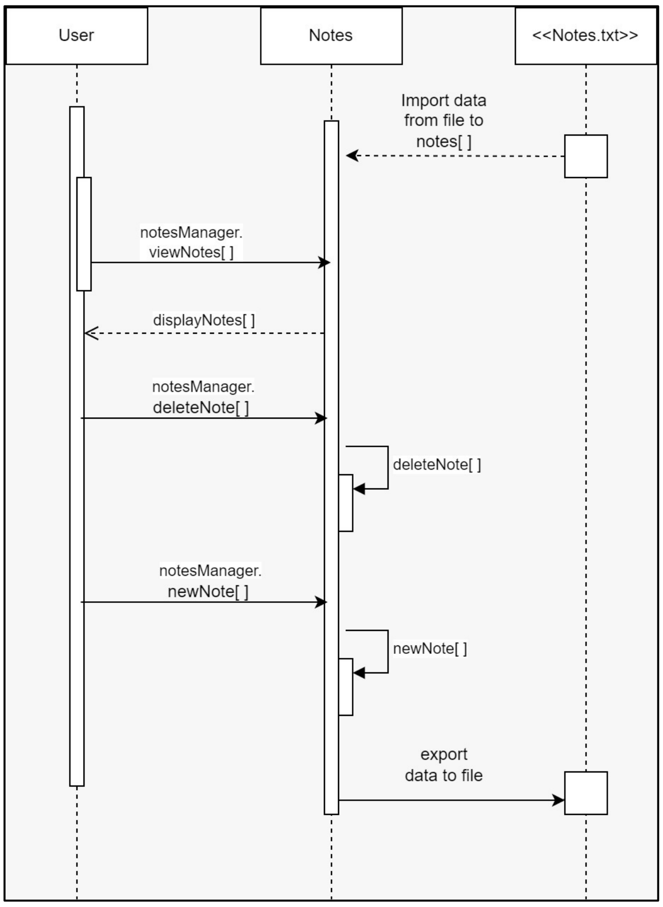


Figure 12.2 Sequence diagram for Notes

Diagram 12.2 shows how the user interacts with the NotesManager class to view existing notes. The user triggers the process, through the main menu. On choosing viewNotes, the NotesManager then checks if there are any stored notes. If there are none, the user is informed and the process ends. If notes exist, the NotesManager goes through each note in its storage and displays the content along with a number to identify it. This sequence diagram illustrates a simple flow where the user requests to see their notes, and the NotesManager retrieves and presents them if available. They can delete the note too, by choosing the deleteNote option, and then entering an index for the note. A new note can be added too, by choosing the option of newNote. The new note will then be added to the Notes.txt file.

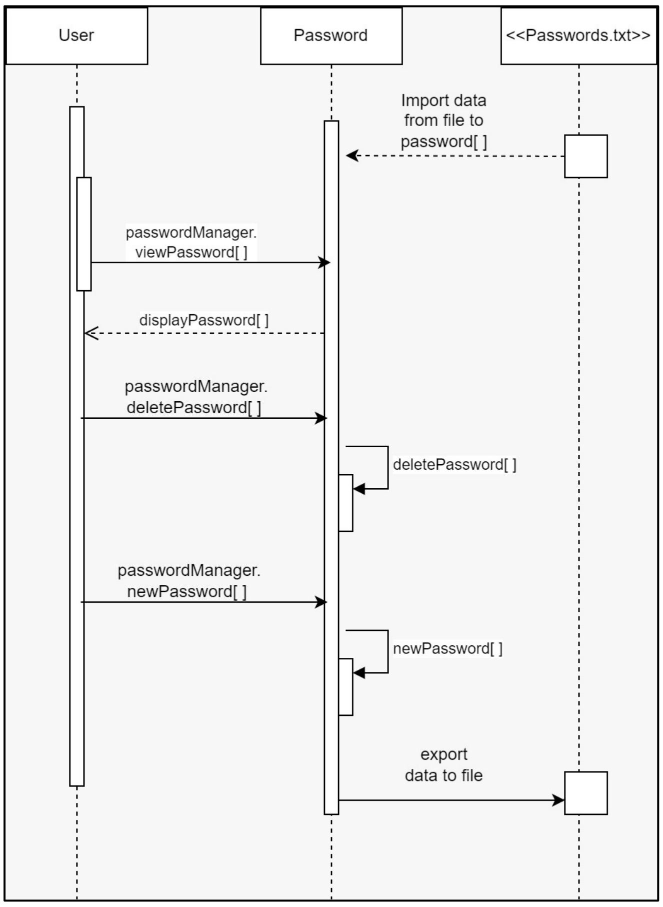

User
Password
<<Passwords.txt>>
Import data from file to password[ ]
passwordManager. viewPassword[ ]
displayPassword[ ]
passwordManager. deletePassword[ ]
deletePassword[ ]
passwordManager. newPassword[ ]
newPassword[ ]
export data to file

Figure 12.3 Sequence diagram for Passwords

Diagram 12.3 shows how the user interacts with the PasswordsManager class to view existing passwords. The user triggers the process, through the main menu. On choosing viewPasswords, the PasswordsManager then checks if there are any stored passwords. If there are none, the user is informed and the process ends. If passwords exist, the PasswordsManager goes through each password in its storage and displays the content along with a number to identify it. This sequence diagram illustrates a simple flow where the user requests to see their passwords, and the PasswordsManager retrieves and presents them if available. They can delete the password too, by choosing the deletePassword option, and then entering an index for the password. A new password can be added too, by choosing the option of newPassword. The new password then will be added to the Passwords.txt file.

13. OBJECT ORIENTED PROGRAMMING IMPLEMENTED IN CODE WITH JUSTIFICATION

i. Namespace: It is used in the code "using namespace std". It allows us to use the names defined in the std namespace without having to use std::. The std namespace is the standard library namespace and it contains most of the basic functions. [1]

ii. Class: It is the backbone of Object-Oriented Programming. It is user-defined data along with data members and member functions. It is used to hold data and functions to be called by the object. [1]

iii. Object: It is the instance of a class. It is used to call the member functions of a class. Without this most of the program will not run. [1]

iv. Abstraction: Abstraction is achieved by defining classes and only showing what is required to the user. In the case of the project all the code and the major chunk of processing is hidden and the user only sees what is required for the program to run. [1]

v. File Handling: File Handling is the major part of the project as it is used to store the data for login, notes, and passwords permanently on the system which can be accessible whenever the user logs in. [1]

vi. Encapsulation: It is demonstrated by using the classes such as 'NotesManager' and 'PasswordManager' where the notes[], structure of website, username, and password are the private members of the class. This provides controlled access to the details of the classes. [1]

vii. Constructors: Default constructors have been used to initialize the 'numNotes' and 'numPasswords' to 0 so that we know the list size in each case. [1]

viii. Destructors: A destructor is a special member function of a class used to deallocate memory and do other cleanup for a class object and its class members when the object is destroyed. [1]

ix. Templates: Templates are tools that enable generic coding. They help create data members of various data types, without having to create separate variables for each data type. [1]

14. ALGORITHM

1. Start.

2. Check if <<user.txt>> is empty; If YES go to Step 3, else go to Step 7.

3. Enter storedUsername.

4. Enter storedPassword.

5. Store storedUsername and storedPassword in <<user.txt>>.

6. Display "Account Created Successfully".

7. Enter username.

8. Enter mainPassword.

9. Check if storedUsername and storedPassword are the same as username and mainPassword; If yes go to Step 11, else go to Step 10.

10. Display "Login Failed!!! Please check Username or Password". Go to Step 7.

11. Display the Main Menu.

12. Enter choose from the Main Menu for switch case.

13. Choices to choose from the Main Menu for switch case; If '1' go to Step 14, '2' go to step 24, '3' go to step 34.

14. Display the Notes Menu.

15. Enter choice for notes menu.

16. Choices to choose from the Notes Menu; If '1' go to Step 17, '2' go to Step 18, '3' go to Step 21.

17. Display all the notes from the <<notes.txt>>. Go to step 11.

18. Enter a note to be added.

19. Store the next note in the file on the next index.

20. Display "Note has been added". Go to step 11.

21. Enter index of note to be deleted.

22. Note from that index is deleted from the <<notes.txt>>.

23. Display "Note at index has been deleted". Go to Step 11.

24. Display the Password Menu.

25. Enter choice for password menu.

26. Choices to choose from the Password Menu; If '1' go to Step 27, '2' go to Step 28, '3' go to Step 31.

27. Display all the website, websiteUsername, and password from their respective files. Go to step 11.

28. Enter website, websiteUsername, and password.

29. Store website, webUsername, and password into their respective files.

30. Display "Password has been added with all the details". Go to Step 11.

31. Enter the index of password to be deleted.

32. Note from that index is deleted from the <<password.txt>>.

33. Display "Password at index has been deleted".

34. Display "You have exited the program".

35. Stop.

15. **FLOWCHART**

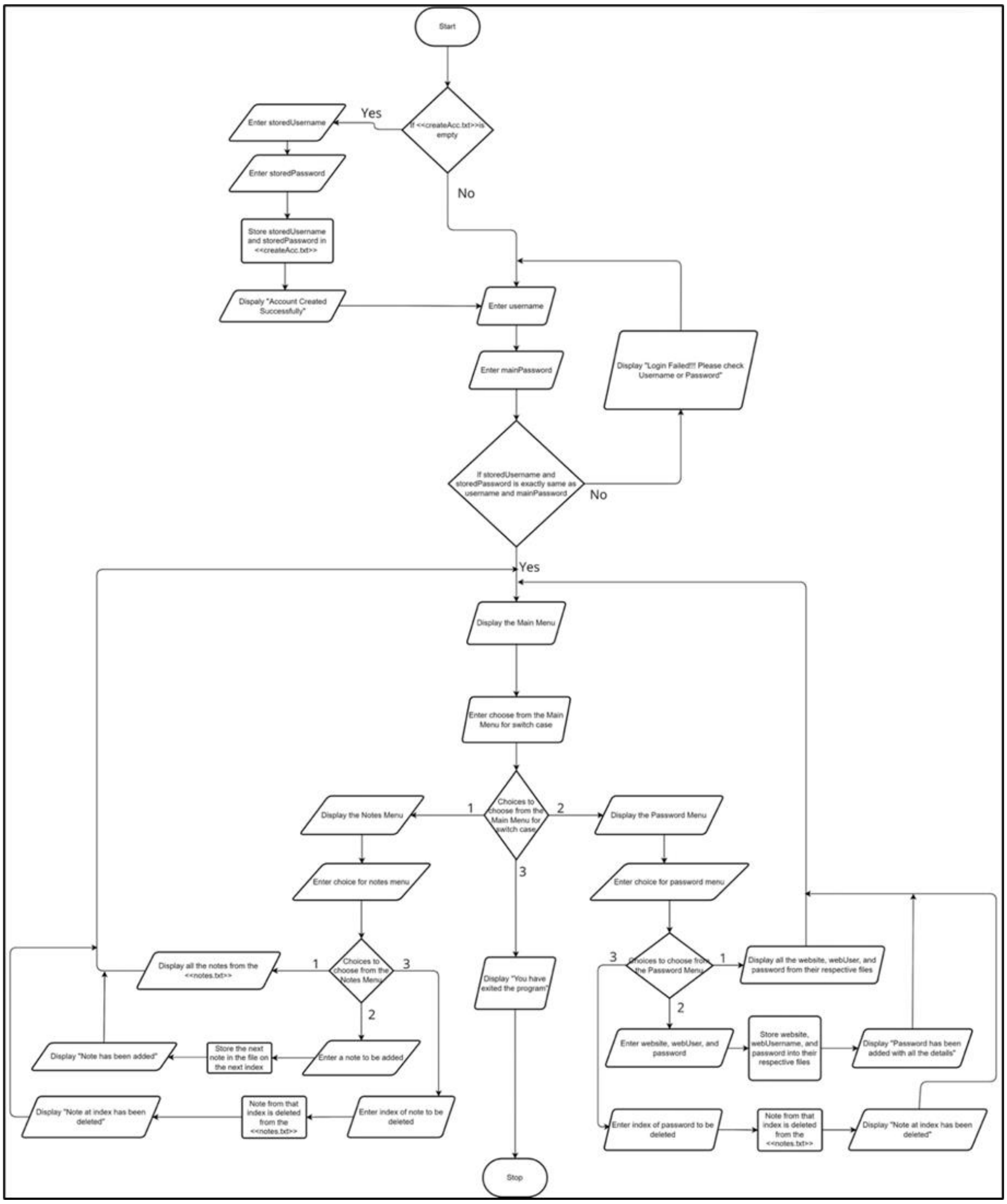

Start
If <<createAcc.txt>>is empty
Yes
No
Enter storedUsername
Enter storedPassword
Store storedUsername and storedPassword in <<createAcc.txt>>
Dispaly "Account Created Successfully"
Enter username
Enter mainPassword
Display "Login Failed!!! Please check Username or Password"
If storedUsername and storedPassword is exactly same as username and mainPassword
No
Yes
Display the Main Menu
Enter choose from the Main Menu for switch case
Choices to choose from the Main Menu for switch case
1
2
3
Display the Notes Menu
Enter choice for notes menu
Choices to choose from the Notes Menu
1
2
3
Display all the notes from the <<notes.txt>>
Enter a note to be added
Store the next note in the file on the next index
Display "Note has been added"
Enter index of note to be deleted
Note from that index is deleted from the <<notes.txt>>
Display "Note at index has been deleted"
Display "You have exited the program"
Stop
Display the Password Menu
Enter choice for password menu
Choices to choose from the Password Menu
1
2
3
Display all the website, webUser, and password from their respective files
Enter website, webUser, and password
Store website, webUsername, and password into their respective files
Display "Password has been added with all the details"
Enter index of password to be deleted
Note from that index is deleted from the <<notes.txt>>
Display "Note at index has been deleted"

Figure 15.1 Flowchart

This flowchart shows the basic control and procedure that goes into working of the program.

16. **CODE WRITTEN IN C++**

```
#include <iostream>
#include <fstream> //for i/o files
#include <string> //for string
#include <conio.h> // for system("cls")
#include <windows.h> // for delay/sleep in windows
#include <limits> //used for cin.ignore()

using namespace std;

//MAXIMUM SIZE IS SET TO 100
const int MAX_NOTES = 100;
const int MAX_PASSWORDS = 100;

template <typename T>
class NotesManager
{
   private:
      T notes[MAX_NOTES];
      int numNotes; //To know the number of notes we have

   public:
      NotesManager() //default constructor
      {
         numNotes = 0;
      }

      void viewNotes()
      {
         if (numNotes == 0)
         {
            cout << "Oops! It seems the notes manager went on vacation...without leaving any
notes behind!\n" << endl;
            return; //exits the function and control returns to the main without returning any
value
         }
         cout << "Notes:" << endl;
         for (int i = 0; i < numNotes; ++i)
         {
            cout << i + 1 << ". " << notes[i] << endl;
         }
         cout << endl;
      }
```

```
    void addNote(string &note) //&note is used to avoid unnecessary copies
    {
        if (numNotes < MAX_NOTES) //if enough space is there
        {
            notes[numNotes++] = note;
            cout << "\nNoted and acknowledged!!\n" << endl;
        }
        else //if it is full
        {
            cout << "\"Sorry, the notes manager is full! No more room for your brilliant
ideas.\"" << endl;
        }
    }

    void deleteNote(int index)
    {
        if (numNotes == 0)
        {
            cout << "\nAll clear! No notes to delete.\n" << endl;
            return; //exits the function and skips rest of the function
        }

        if (index < 1 || index > numNotes) {
            cout << "\nWhoops! Looks like you're trying to reach notes from a parallel
universe... If the list is empty enter any number... \nInvalid index number...\n" << endl;
            return; //exits the function and control returns to the main without returning any
value
        }

        for (int i = index - 1; i < numNotes - 1; ++i) {
            notes[i] = notes[i + 1]; //similar to deleting in stacks
        }
        numNotes--;
        cout << "\nYour note has been deleted!\n" << endl;
    }

    void saveNotesToFile(const string& filename)  //const added  since file name is not
changing + error being shown without it
    {
        ofstream outfile(filename);
        if (outfile.is_open())
        {
            for (int i = 0; i < numNotes; ++i)
            {
                outfile << notes[i] << endl;
            }
            //cout << "Notes saved to file successfully." << endl; //tesingPhase
        }
        else
        {
            cout << "Oops, we are having trouble to save your notes..." << endl;
        }
    }

    void loadNotesFromFile(const string& filename) //const added  since file name is not
changing + error being shown without it
    {
        ifstream infile(filename);
        if (infile.is_open())
        {
            string note;
            while (getline(infile, note)) //if the line has data it will run
            {
                if (numNotes < MAX_NOTES)
                {
                    notes[numNotes++] = note;
                }
                else
                {
                    cout << "Sorry, we've reached the maximum limit..." << endl;
                    break;
                }
            }
            // cout << "Notes loaded from file successfully." << endl; //tesingPhase
        }
        else
        {
            //cout << "Uh-oh! The file is MIA. Starting with a clean slate for your notes..." <<
endl; //debugging
        }
    }
    ~NotesManager(){}
  };

template <typename T>
```

```
class PasswordManager
{
   private:
      struct PasswordEntry
      {
         T website;
         T username;
         T password;
      };

      PasswordEntry entry[MAX_PASSWORDS];
      int numPasswords;

   public:
      PasswordManager()
      {
         numPasswords = 0;
      }

      void viewPasswords()
      {
         if (numPasswords == 0)
         {
            cout << "Looks like the password fairy took a day off!! No passwords lists are
available!\n" << endl;
            return;
         }
         cout << "Passwords:" << endl;
         for (int i = 0; i < numPasswords; ++i)
         {
            cout << i + 1 << ". Website: " << entry[i].website << ", Username: " <<
entry[i].username << ", Password: " << entry[i].password << endl;
         }
      }

      void addPassword(string &website, string &username, string &password)
      {
         if (numPasswords < MAX_PASSWORDS)
         {
            entry[numPasswords].website = website;
            entry[numPasswords].username = username;
            entry[numPasswords].password = password;
            numPasswords++;
            cout << "\nSuccess! Your new password has been saved!\n" << endl;
        }
        else
        {
            cout << "Oops, your password manager is feeling a bit cramped. Time to de-
clutter!!" << endl;
        }
    }

    void deletePassword(int index)
    {
        if (numPasswords == 0)
        {
            cout << "No passwords found, nothing to delete here!!" << endl;
            return;
        }

        if (index < 1 || index > numPasswords)
        {
            cout << "Invalid input. Please choose a valid option from the provided list!\nIf the
list is empty then input number!" << endl;
            return;
        }

        for (int i = index - 1; i < numPasswords - 1; ++i)
        {
            entry[i] = entry[i + 1];
        }
        numPasswords--;
        cout << "Done! Password deleted!" << endl;
    }

    void savePasswordsToFile(const string &filename)
    {
        ofstream outfile(filename);
        if (outfile.is_open())
        {
            for (int i = 0; i < numPasswords; ++i)
            {
                outfile << entry[i].website << "," << entry[i].username << "," <<
entry[i].password << endl;
            }
            //cout << "Passwords saved to file successfully." << endl;
        }
        else
        {
            cout << "Uh-oh! File access denied for saving passwords. Try again later!" << endl;
        }
    }

    void loadPasswordsFromFile(const string& filename)
    {
        ifstream infile(filename);
        if (infile.is_open())
        {
            string website, username, password;
            while (getline(infile, website, ',') && getline(infile, username, ',') && getline(infile,
password))
            //istream& getline(istream& is, string& str, char delim);
            //three pieces of information separated by commas and stored in each variable to
array (structure)
            {
                if (numPasswords < MAX_PASSWORDS)
                {
                    entry[numPasswords].website = website;
                    entry[numPasswords].username = username;
                    entry[numPasswords].password = password;
                    numPasswords++;
                } else
                {
                    cout << "Uh-oh! Maximum limit reached. Please delete some passwords to
enter more!" << endl;
                    break;
                }
            }
            // cout << "Passwords loaded from file successfully." << endl;
        }
        else
        {
            //cout << "File not found. Starting with empty passwords." << endl; //debugging
        }
    }
    ~PasswordManager()
    {}
};
```

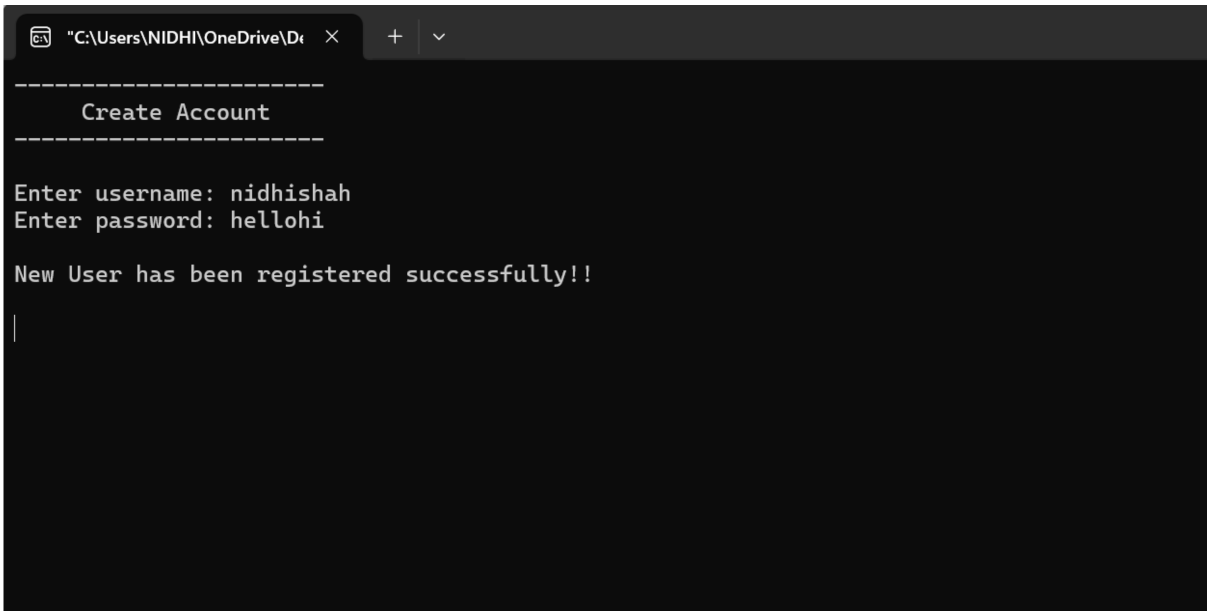
```
class Authenticator
{
   public:
   void createAccount()
   {
      ofstream file("user.txt");
      if (file.is_open())
      {
         string username, password;
         cout << "-----------------------\n";
         cout << "     Create Account    \n";
         cout << "-----------------------\n";
         cout << "\nEnter username: ";
         cin >> username;
         cout << "Enter password: ";
         cin >> password;
         file << username << " " << password;
         cout << "\nNew User has been registered successfully!!\n\n";
         Sleep(1000);
         system("cls");
      }
      else
      {
         cerr << "Error: Unable to open file for writing!\n";
      }
   }
};

class User
{
   public:
   void loginInput(string& username, string& password) {
      cout << "Welcome! To proceed, please log in with your account credentials...\n" <<
endl;
      cout << "-----------------------\n";
      cout << "       Login Page      \n";
      cout << "-----------------------\n";
      cout << "\nEnter username: ";
      cin >> username;
      cout << "Enter password: ";

      char ch;
      do {
            ch = getch(); // returns the ASCII value of the character
            if (isdigit(ch) || isalpha(ch) || ispunct(ch)) {
                password += ch;
                cout << "*";
            }
        } while (isdigit(ch) || isalpha(ch) || ispunct(ch));
    }

    bool loginProcess(const string& storedUsername, const string& storedPassword, string
username, string password)
    {
        return (username == storedUsername && password == storedPassword);
    }

    void loginOutput(bool success)
    {
        cout << endl;
        if (success)
        {
            cout << "\n-----------------------\n";
            cout << "    Login successful!  \n";
            cout << "-----------------------\n\n";
        }
        else
        {
            cout << "\n-----------------------\n";
            cout << " Invalid username or password! \n";
            cout << "-----------------------\n\n";
        }
    }
};

int main()
{
    const string notesFilename = "notes.txt";
    const string passwordsFilename = "passwords.txt";

    Authenticator authenticator;
    User user;

    //template used
    NotesManager <string> notesManager;
    PasswordManager <string> passwordManager;
```

```
bool success = false; // Initialize success flag to false
int loginCount = 0;

do
{
    ifstream file("user.txt");

    // If the file stream is in a failed state or if it has reached the end of the file
    if (!file || file.peek() == ifstream::traits_type::eof())
    {
        authenticator.createAccount();
    }
    else
    {
        string storedUsername, storedPassword;
        file >> storedUsername >> storedPassword;
        file.close();

        string username, password;
        user.loginInput(username, password);
        success = user.loginProcess(storedUsername, storedPassword, username, password);
        user.loginOutput(success);
        if((success == false) && (loginCount < 3))
        {
            cout << "You have " << (3 - loginCount) << " attempts left!\n\n";
            loginCount++;
            Sleep(1500);
            system("cls");
        }
        else if (((success == false)) && (loginCount >= 3))
        {
            system("cls");
            cout << "Due to multiple attempts... You have been logged out!!\n\n";
            cout << "-------------------------\n";
            cout << "Exiting the program.\n";
            cout << "-------------------------\n";
            exit(1);

        }
    }
} while (!success);
Sleep(1500);
```

```
    system("cls");

    notesManager.loadNotesFromFile(notesFilename);
    passwordManager.loadPasswordsFromFile(passwordsFilename); // Load passwords from
file

    int mainMenuChoice;
    do {
        cout << "-------------------------\n";
        cout << "       Main Menu:\n";
        cout << "-------------------------\n\n";
        cout << "1. Enter Notes Manager\n";
        cout << "2. Enter Password Manager\n";
        cout << "3. Exit Program\n";
        cout << "Enter your choice: ";
        cin >> mainMenuChoice;
        if (cin.fail())
        {
            cout << "Invalid input. Please choose a valid option from the provided list...\n" <<
endl;
            cin.clear();
            // to ingore all the remaining characters in the buffer
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            Sleep(1500);
            system("cls");
            continue;
        }

        system("cls");

        switch (mainMenuChoice) {
            case 1: {
                int notesChoice;
                string note;
                do {
                    cout << "-------------------------\n";
                    cout << "    Notes Manager Menu:" << endl;
                    cout << "-------------------------\n\n";
                    cout << "1. View Note" << endl;
                    cout << "2. Add Note" << endl;
                    cout << "3. Delete Note" << endl;
                    cout << "4. Exit Notes Manager" << endl;
                    cout << "Enter your choice: ";
                    cin >> notesChoice;
                    if (cin.fail())
                    {
                        cout << "Invalid input. Please choose a valid number option from the provided
list!" << endl;
                        cin.clear();
                        // to ingore all the remaining characters in the buffer
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');
                        Sleep(1500);
                        system("cls");
                        continue;
                    }
                    cin.ignore();

                    system("cls");
                    switch (notesChoice)
                    {
                        case 1:
                            notesManager.viewNotes();
                            break;
                        case 2:
                            cout << "Enter your note: ";
                            getline(cin, note);
                            notesManager.addNote(note);
                            break;
                        case 3:
                            notesManager.viewNotes();
                            int index;
                            cout << "Enter the index of the note to delete: ";
                            cin >> index;
                            if (cin.fail())
                            {
                                cout << "Invalid input. Please choose a valid number option from the
provided list..." << endl;
                                cin.clear();
                                // to ingore all the remaining characters in the buffer
                                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                                Sleep(1500);
                                system("cls");
                                continue;
                            }
                            notesManager.deleteNote(index);
                            break;
                        case 4:
                            cout << "------------------------\n";
                            cout << "Exiting Notes Manager..." << endl;
                            cout << "------------------------\n";
                            Sleep(1000);
                            system("cls");
                            break;
                        default:
                            cout << "\nInvalid input. Please choose a valid option from the provided
list!" << endl;
                    }
                } while (notesChoice != 4);
                notesManager.saveNotesToFile(notesFilename);
                break;
            }
            case 2: {
                int passwordChoice;
                do {
                    cout << "------------------------\n";
                    cout << "  Password Manager Menu:\n";
                    cout << "------------------------\n\n";
                    cout << "1. View Passwords\n";
                    cout << "2. Add Password\n";
                    cout << "3. Delete Password\n";
                    cout << "4. Exit Password Manager\n";
                    cout << "Enter your choice: ";
                    cin >> passwordChoice;
                    if (cin.fail())
                    {
                        cout << "Invalid input. Please choose a valid number option from the provided
list!" << endl;
                        cin.clear();
                        // to ingore all the remaining characters in the buffer
                        cin.ignore(numeric_limits<streamsize>::max(), '\n');
                        Sleep(1500);
                        system("cls");
                        continue;
                    }

                    system("cls");
                    switch (passwordChoice) {
                        case 1:
                            passwordManager.viewPasswords();
                            break;
                        case 2: {
                            string website, username, password;
                            cout << "Enter website: ";
                            cin.ignore();
                            getline(cin, website);
                            cout << "Enter username: ";
                            getline(cin, username);
                            cout << "Enter password: ";
                            getline(cin, password);
                            passwordManager.addPassword(website, username, password);
                            break;
                        }
                        case 3: {
                            passwordManager.viewPasswords();
                            int index;
                            cout << "\nEnter the index of the password to delete: ";
                            cin >> index;
                            if (cin.fail())
                            {
                              cout << "Invalid input. Please choose a valid number option from the
provided list!" << endl;
                              cin.clear();
                              // to ingore all the remaining characters in the buffer
                              cin.ignore(numeric_limits<streamsize>::max(), '\n');
                              Sleep(1500);
                              system("cls");
                              continue;
                            }
                            passwordManager.deletePassword(index);
                            cout << "\n";
                            break;
                        }
                        case 4:
                            cout << "-------------------------\n";
                            cout << "Exiting Password Manager...\n";
                            cout << "-------------------------\n";
                            Sleep(1000);
                            system("cls");
                            break;
                        default:
                            cout << "Invalid input. Please choose a valid option from the provided
list!\n";
```

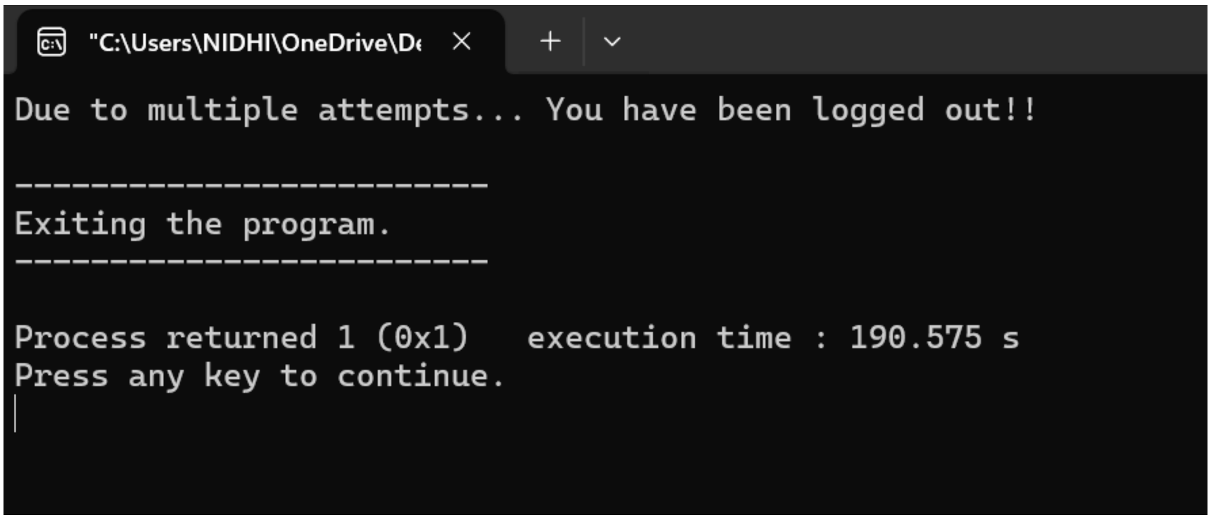
```
                    break;
                }
            } while (passwordChoice != 4);

            passwordManager.savePasswordsToFile(passwordsFilename); // Save passwords to
file before exiting
            break;
        }
        case 3:
            system("cls");
            cout << "----------------------------------\n";
            cout << "Exiting program. Have a great day!\n";
            cout << "----------------------------------\n";
            break;
        default:
            cout << "Invalid input. Please choose a valid option from the provided list!\n";
            break;
      }
  } while (mainMenuChoice != 3);
  return 0;
}
```

17. OUTPUT SCREENSHOT OF ALL FUNCTIONS



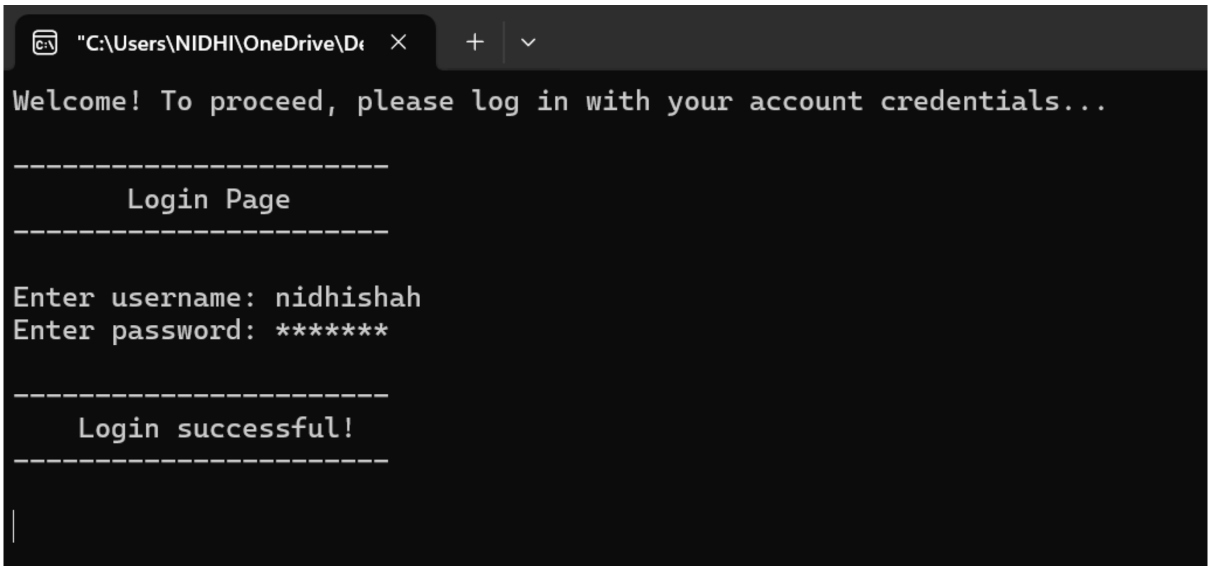
```
-----------------------
     Create Account
-----------------------

Enter username: nidhishah
Enter password: hellohi

New User has been registered successfully!!
```

Figure 17.1 Account Creation



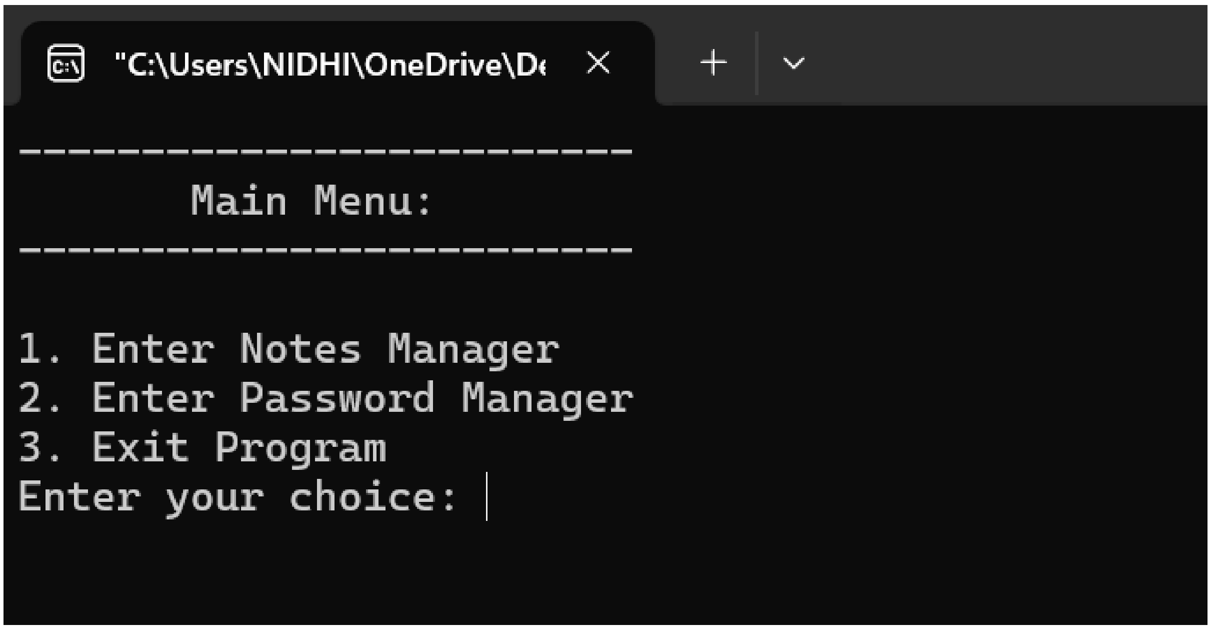
```
Due to multiple attempts... You have been logged out!!

-------------------------
Exiting the program.
-------------------------

Process returned 1 (0x1)   execution time : 190.575 s
Press any key to continue.
```

Figure 17.2 Logout due to multiple failed attempts



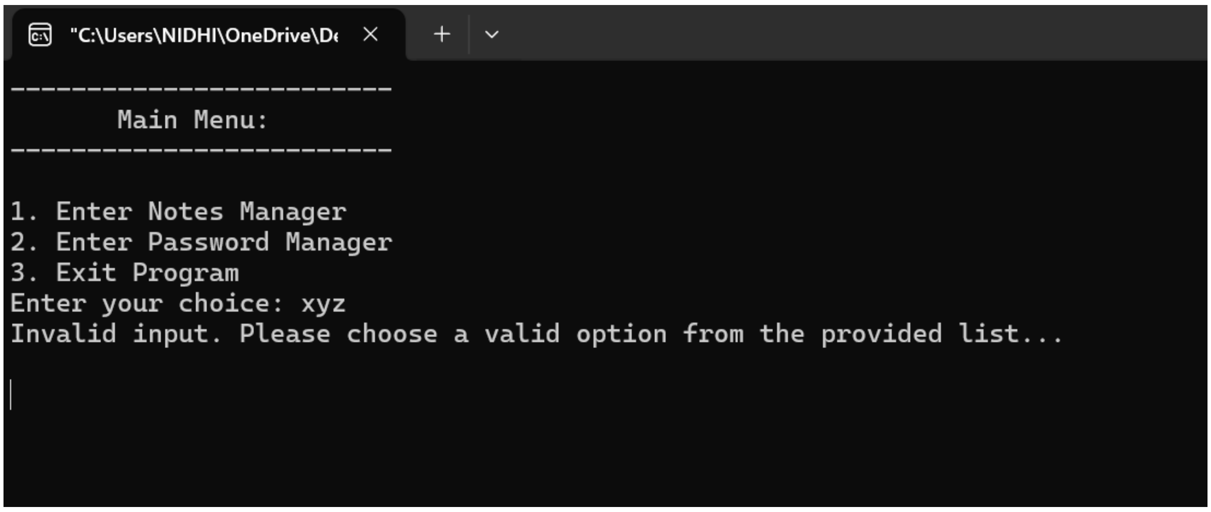
```
Welcome! To proceed, please log in with your account credentials...

-----------------------
       Login Page
-----------------------

Enter username: nidhishah
Enter password: *******

-----------------------
    Login successful!
-----------------------
```

Figure 17.3 Successful Login



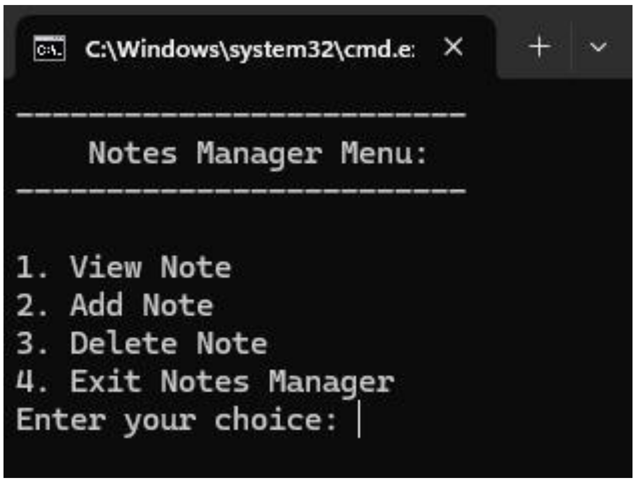
```
-------------------------
       Main Menu:
-------------------------

1. Enter Notes Manager
2. Enter Password Manager
3. Exit Program
Enter your choice:
```

Figure 17.4 Main Menu



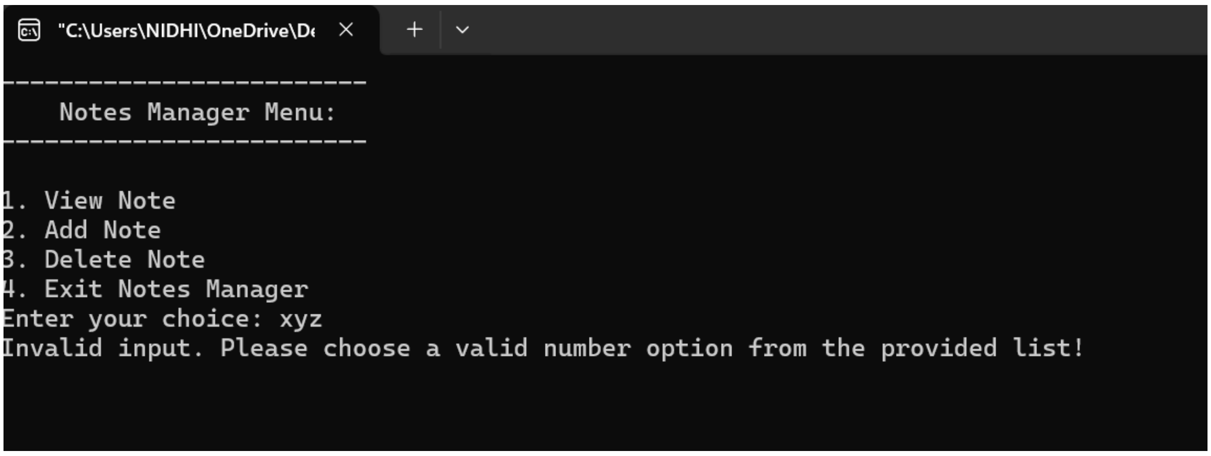
```
-------------------------
       Main Menu:
-------------------------

1. Enter Notes Manager
2. Enter Password Manager
3. Exit Program
Enter your choice: xyz
Invalid input. Please choose a valid option from the provided list...
```

Figure 17.5 Wrong Option



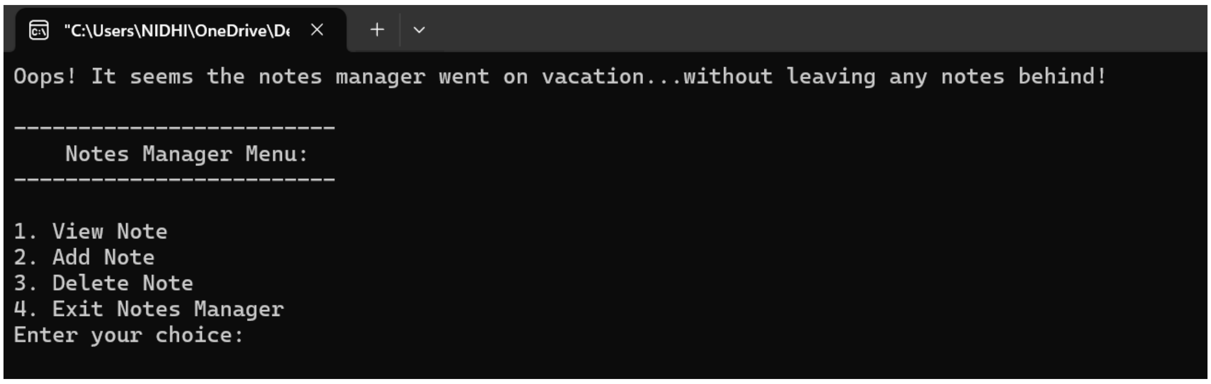
```
-------------------------
    Notes Manager Menu:
-------------------------

1. View Note
2. Add Note
3. Delete Note
4. Exit Notes Manager
Enter your choice: |
```

Figure 17.6 Notes Menu



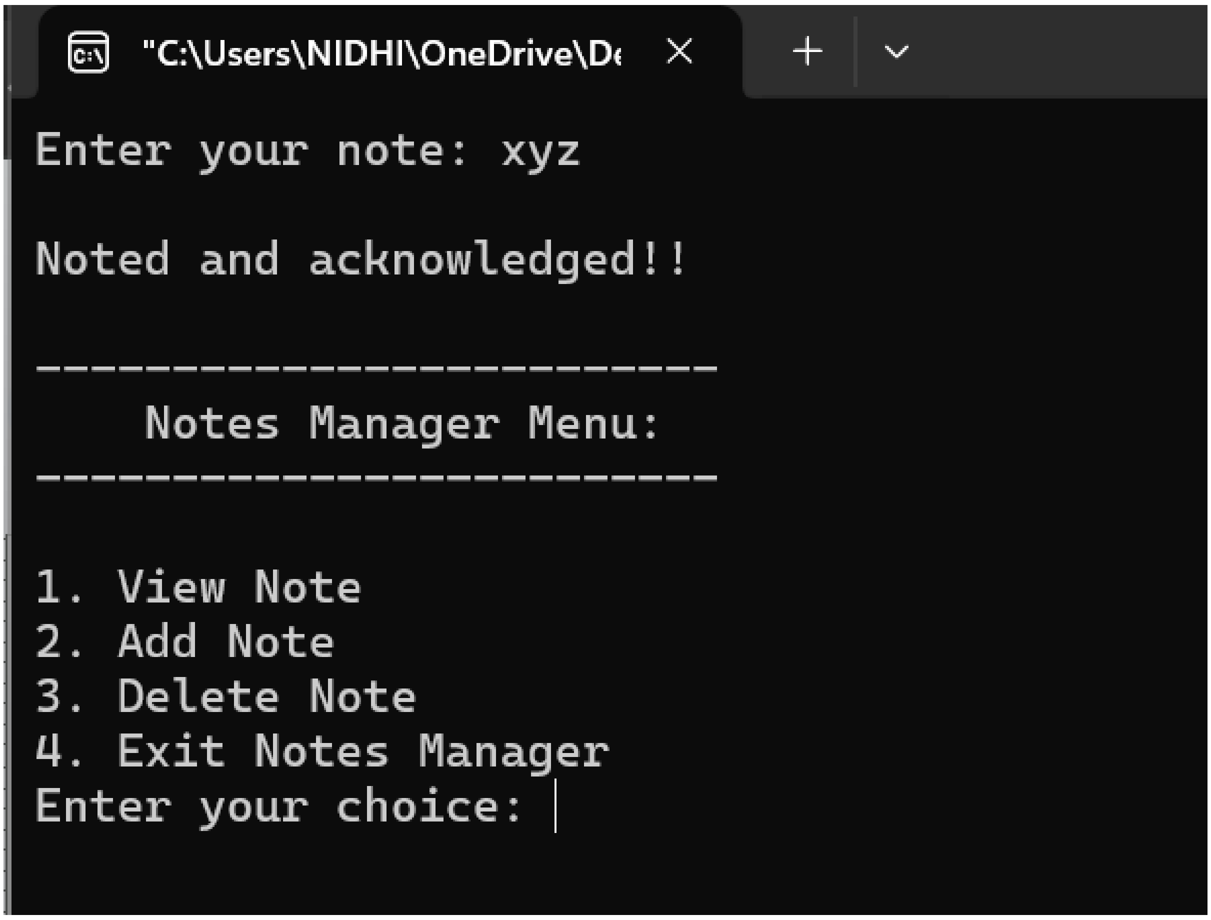
```
-------------------------
    Notes Manager Menu:
-------------------------

1. View Note
2. Add Note
3. Delete Note
4. Exit Notes Manager
Enter your choice: xyz
Invalid input. Please choose a valid number option from the provided list!
```

Figure 17.7 Wrong Option for Notes



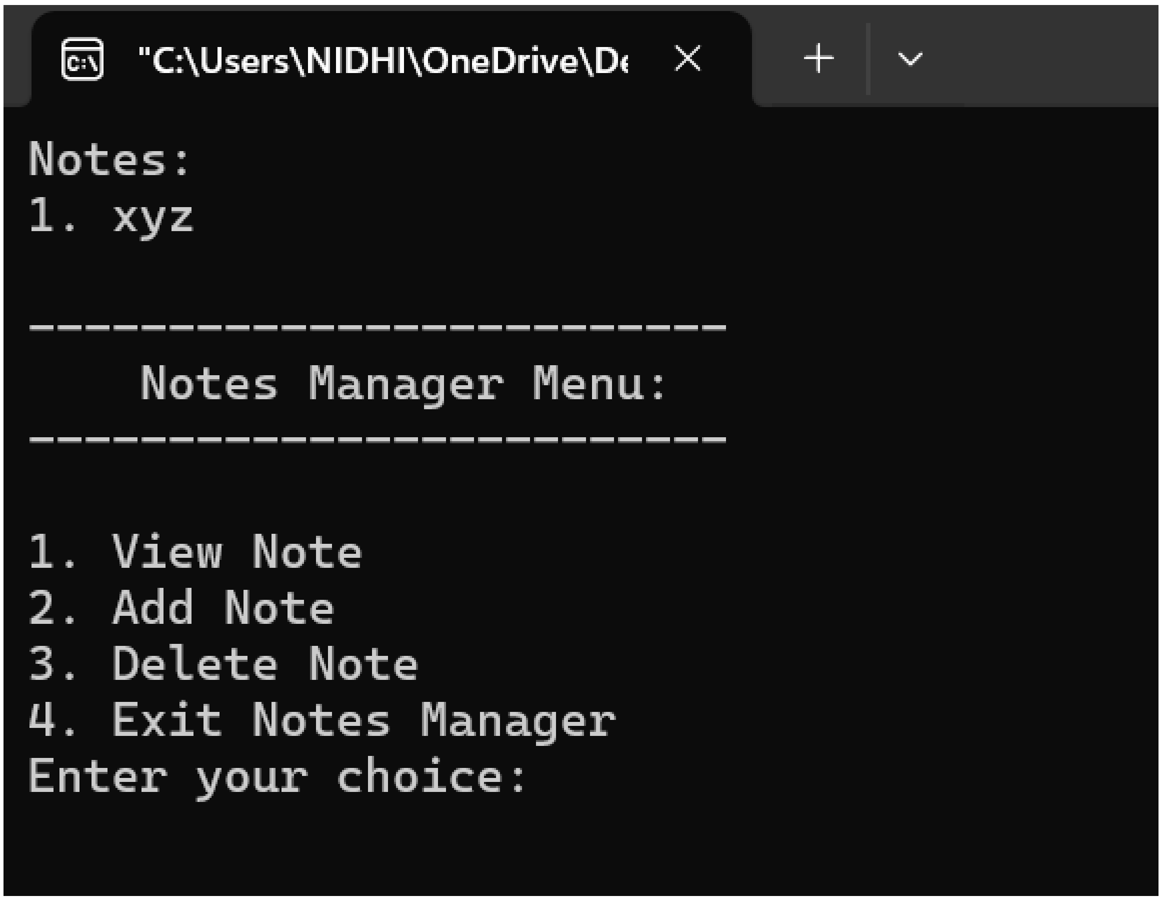
```
Oops! It seems the notes manager went on vacation...without leaving any notes behind!

-------------------------
    Notes Manager Menu:
-------------------------

1. View Note
2. Add Note
3. Delete Note
4. Exit Notes Manager
Enter your choice:
```

Figure 17.9 Empty Notes



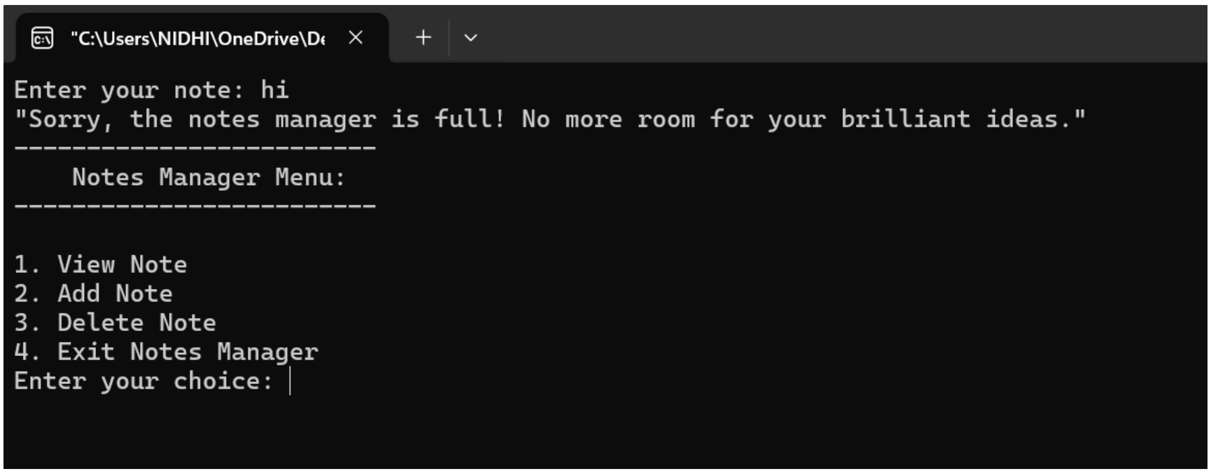
```
Enter your note: xyz

Noted and acknowledged!!

-------------------------
    Notes Manager Menu:
-------------------------

1. View Note
2. Add Note
3. Delete Note
4. Exit Notes Manager
Enter your choice: 
```

Figure 17.10 Add Notes



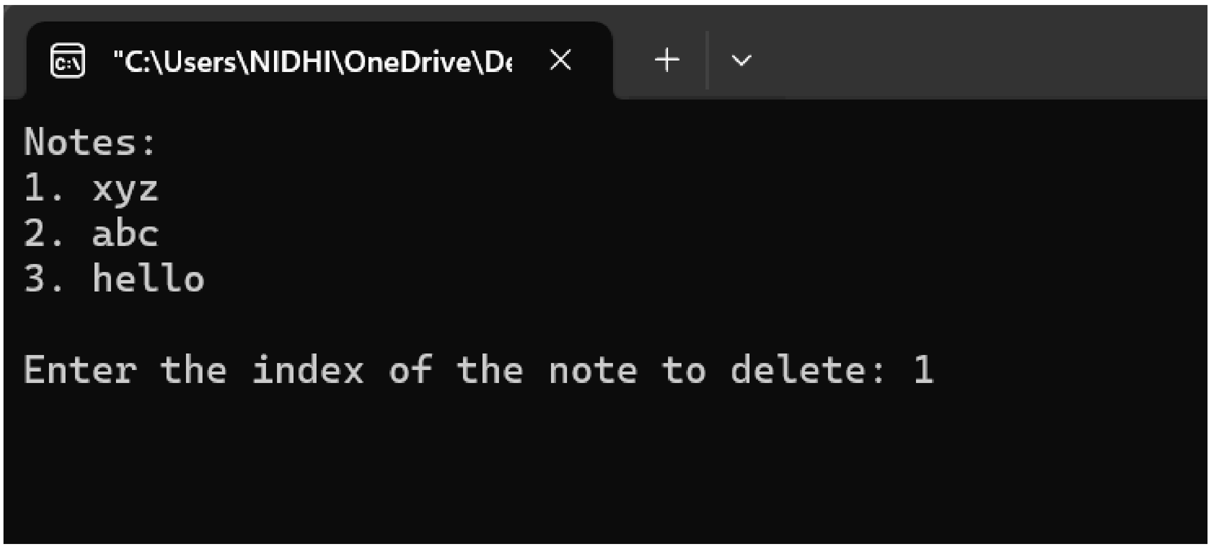
```
Notes:
1. xyz

-------------------------
    Notes Manager Menu:
-------------------------


1. View Note
2. Add Note
3. Delete Note
4. Exit Notes Manager
Enter your choice:
```

Figure 17.11 View Notes



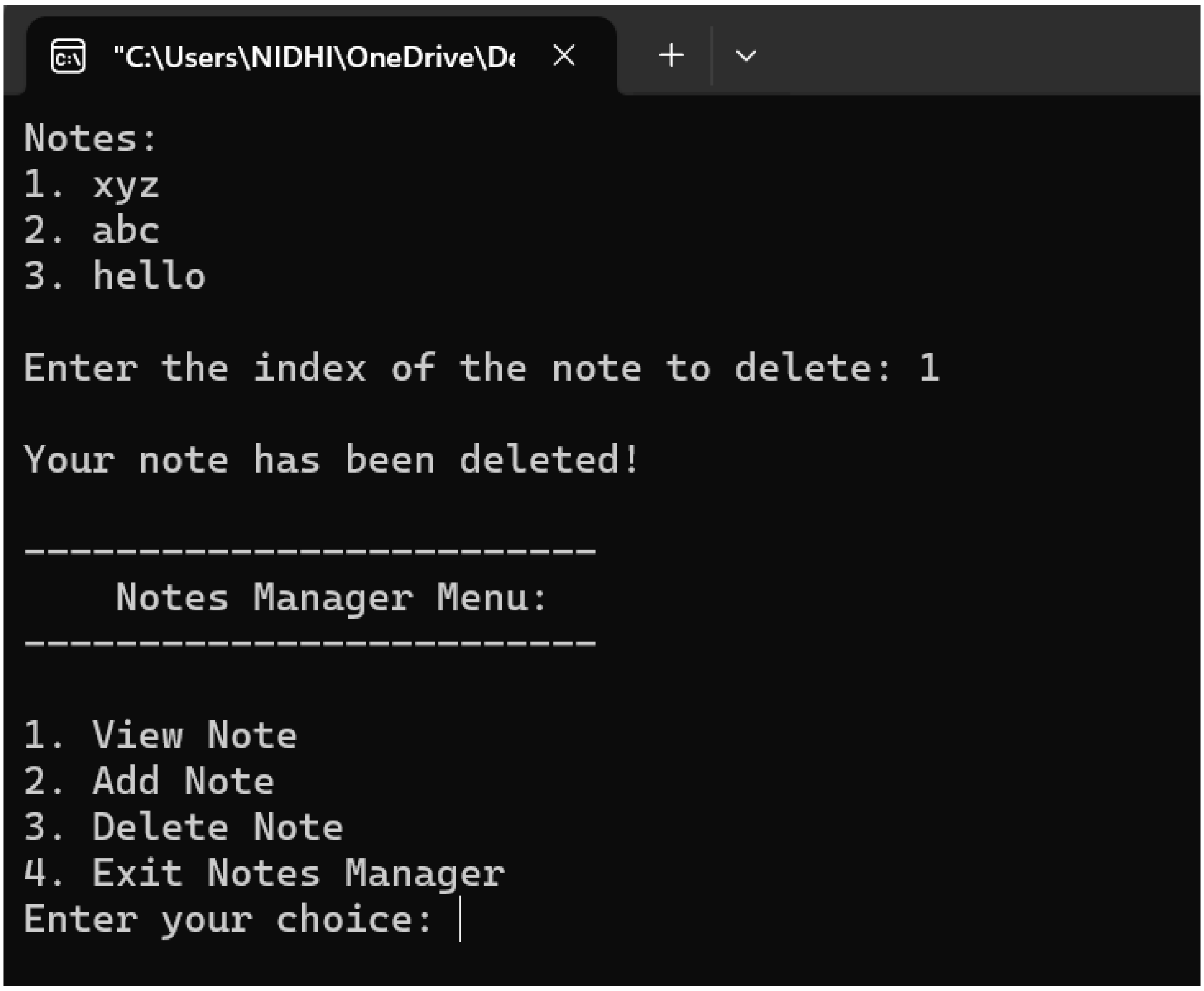
```
Enter your note: hi
"Sorry, the notes manager is full! No more room for your brilliant ideas."
-------------------------
    Notes Manager Menu:
-------------------------

1. View Note
2. Add Note
3. Delete Note
4. Exit Notes Manager
Enter your choice:
```

Figure 17.11 Full Notes File



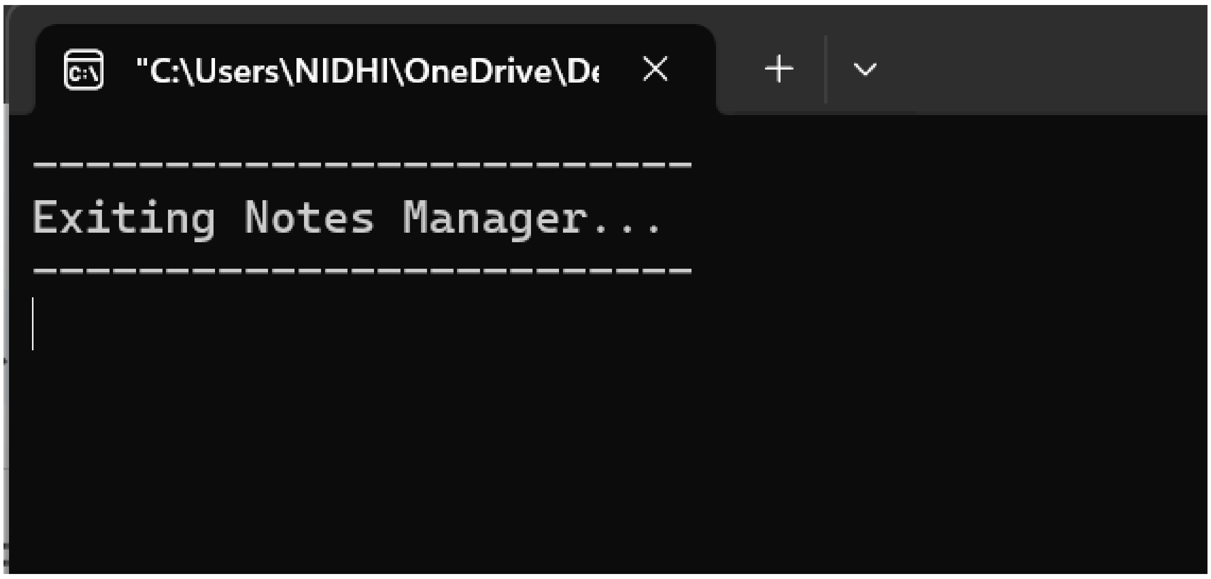
```
"C:\Users\NIDHI\OneDrive\De

Notes:
1. xyz
2. abc
3. hello

Enter the index of the note to delete: 1
```

Figure 17.12 Delete Notes



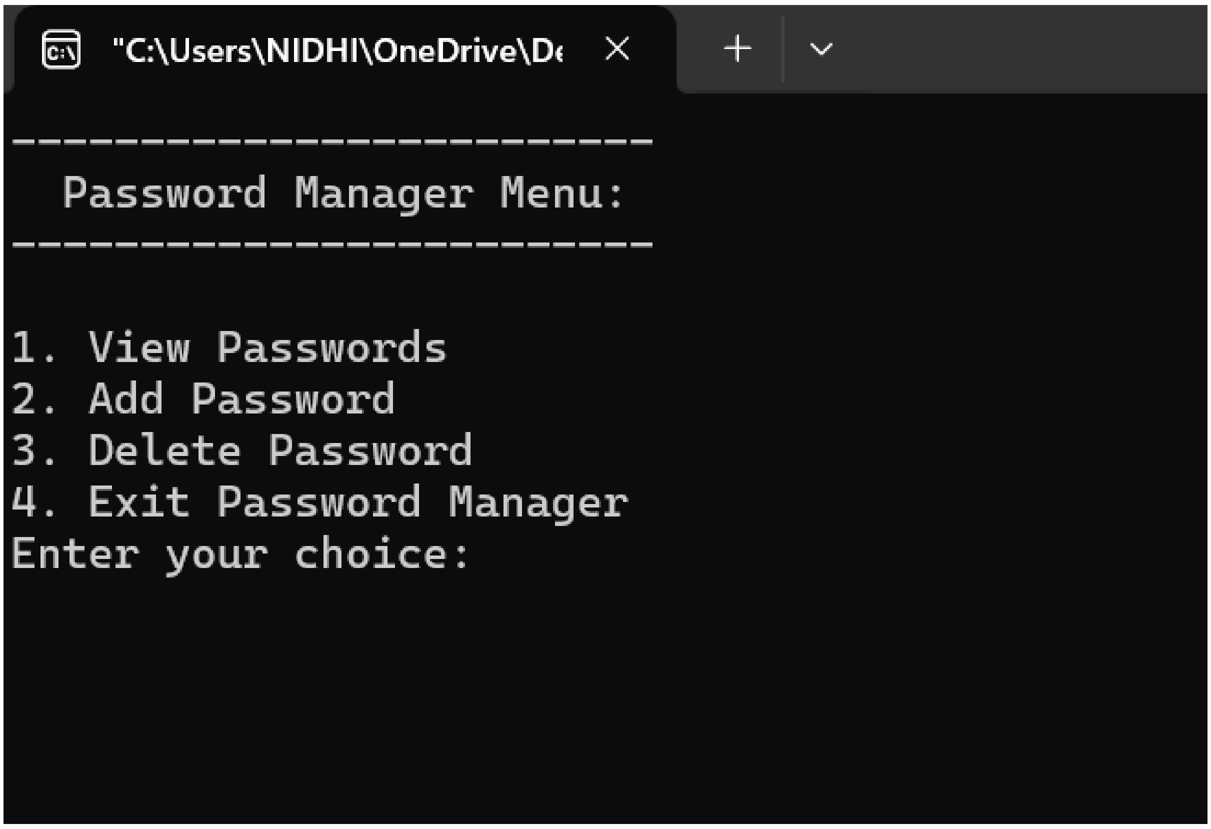
```
"C:\Users\NIDHI\OneDrive\De

Notes:
1. xyz
2. abc
3. hello

Enter the index of the note to delete: 1

Your note has been deleted!

-------------------------
    Notes Manager Menu:
-------------------------

1. View Note
2. Add Note
3. Delete Note
4. Exit Notes Manager
Enter your choice:
```

Figure 17.13 Note Deleted from List



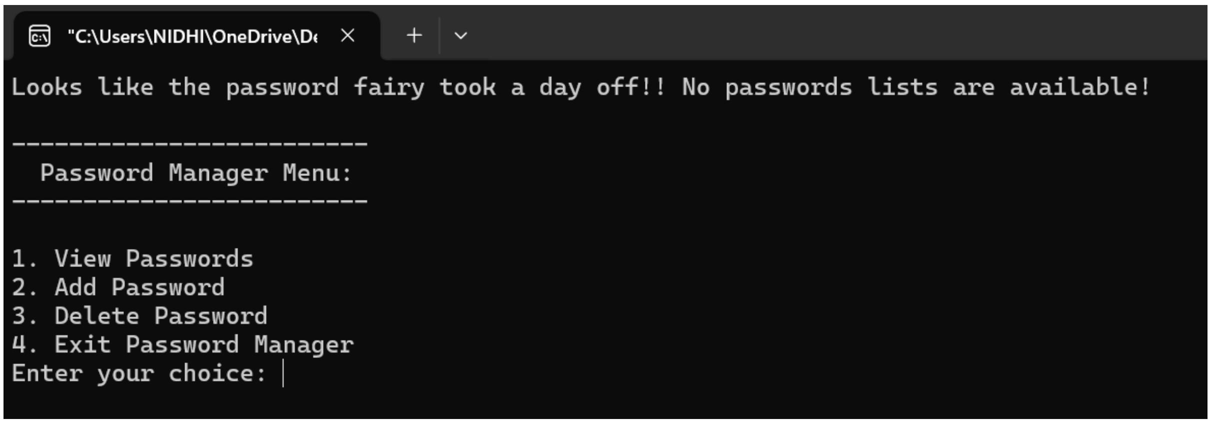
```
"C:\Users\NIDHI\OneDrive\De
-------------------------
Exiting Notes Manager...
-------------------------
```

Figure 17.14 Exiting from Notes Manager



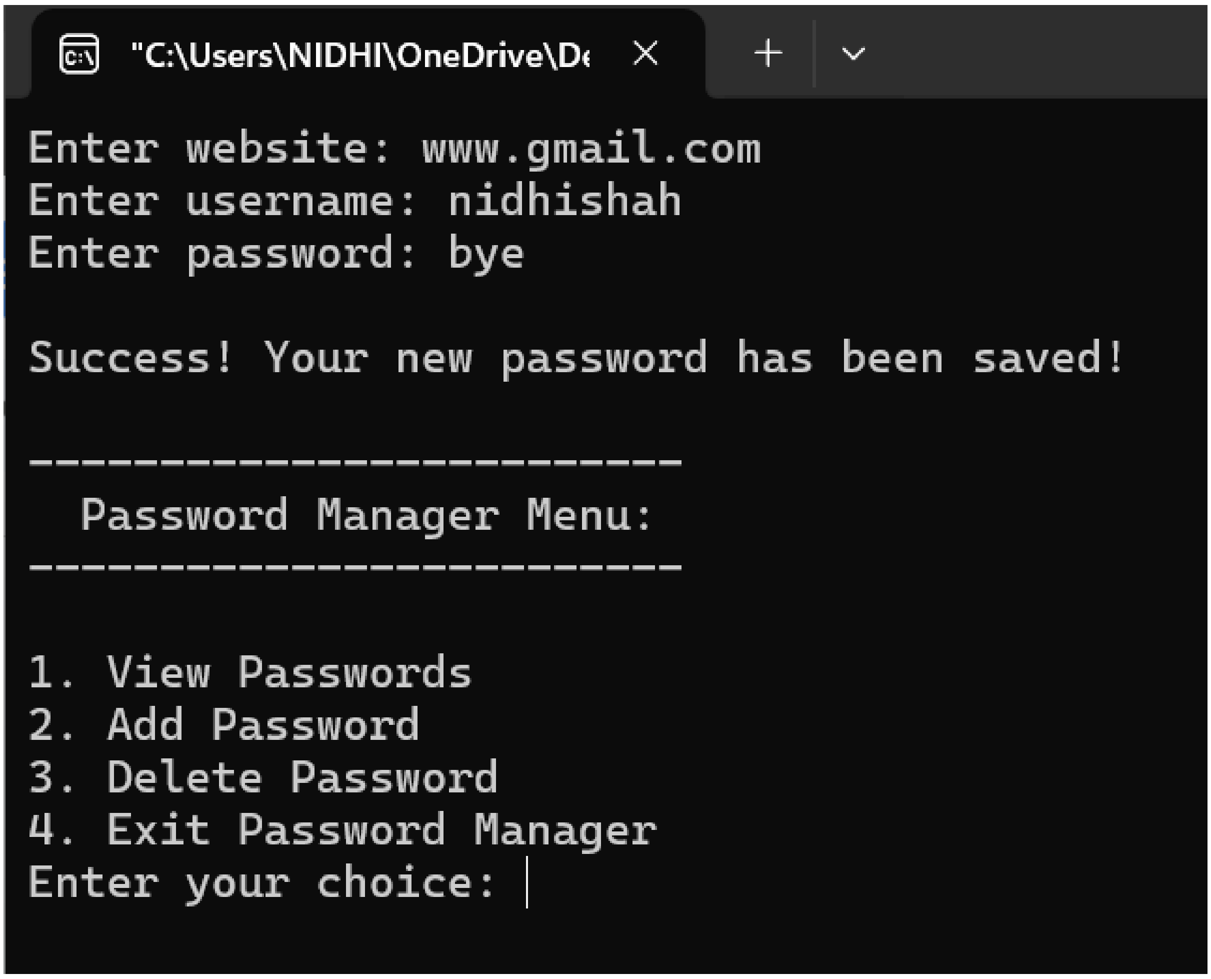
```
"C:\Users\NIDHI\OneDrive\De
-------------------------
  Password Manager Menu:
-------------------------

1. View Passwords
2. Add Password
3. Delete Password
4. Exit Password Manager
Enter your choice:
```

Figure 17.15 Passwords Menu



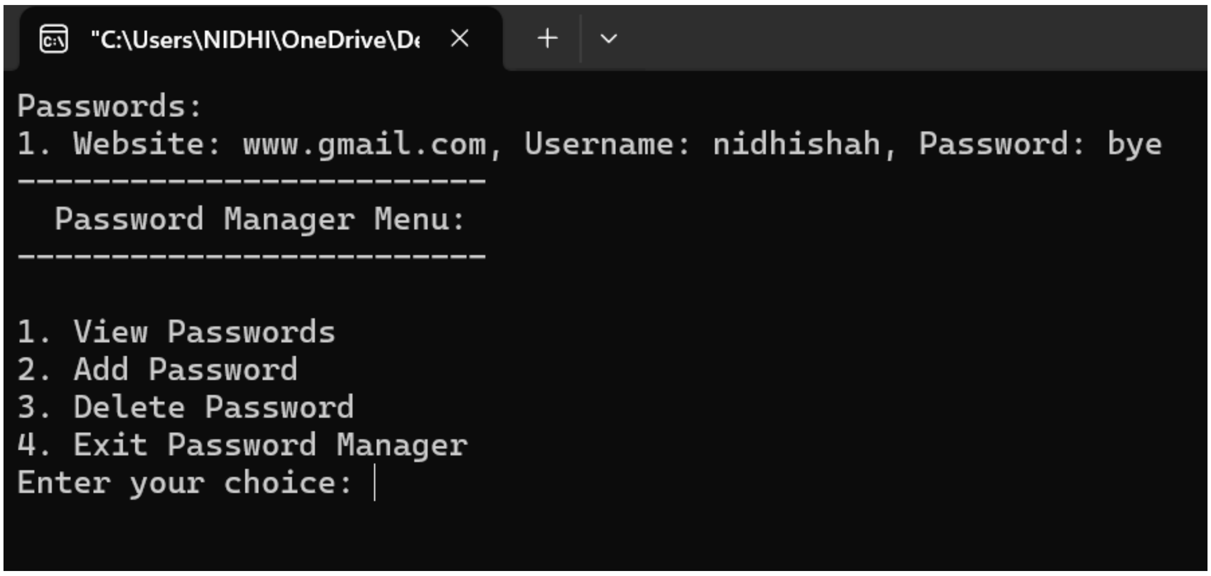
```
Looks like the password fairy took a day off!! No passwords lists are available!

-------------------------
  Password Manager Menu:
-------------------------

1. View Passwords
2. Add Password
3. Delete Password
4. Exit Password Manager
Enter your choice:
```

Figure 17.16 Empty Password File



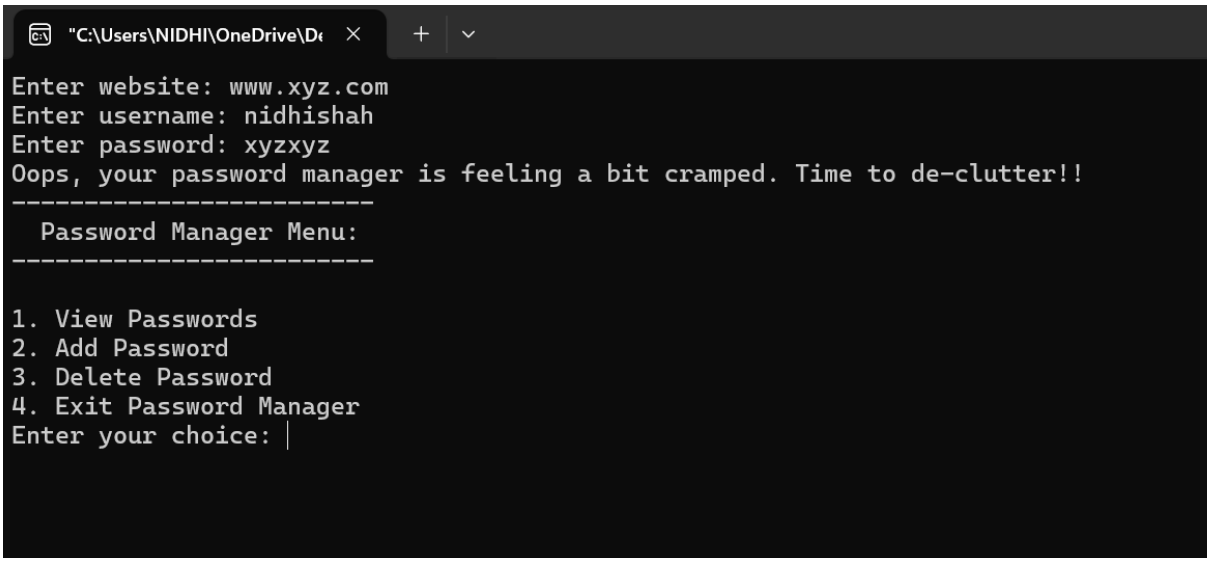
```
Enter website: www.gmail.com
Enter username: nidhishah
Enter password: bye

Success! Your new password has been saved!

-------------------------
  Password Manager Menu:
-------------------------

1. View Passwords
2. Add Password
3. Delete Password
4. Exit Password Manager
Enter your choice:
```

Figure 17.17 Add a new Password



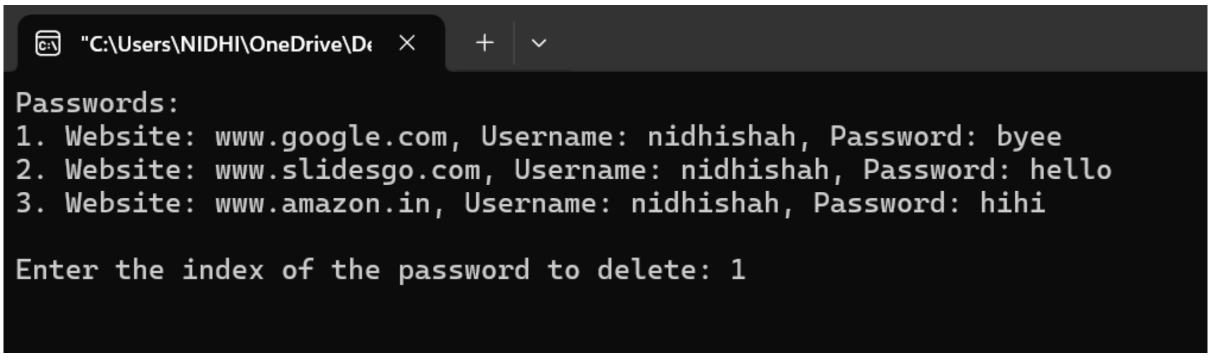
```
Passwords:
1. Website: www.gmail.com, Username: nidhishah, Password: bye
-------------------------
  Password Manager Menu:
-------------------------

1. View Passwords
2. Add Password
3. Delete Password
4. Exit Password Manager
Enter your choice:
```

Figure 17.18 View Passwords



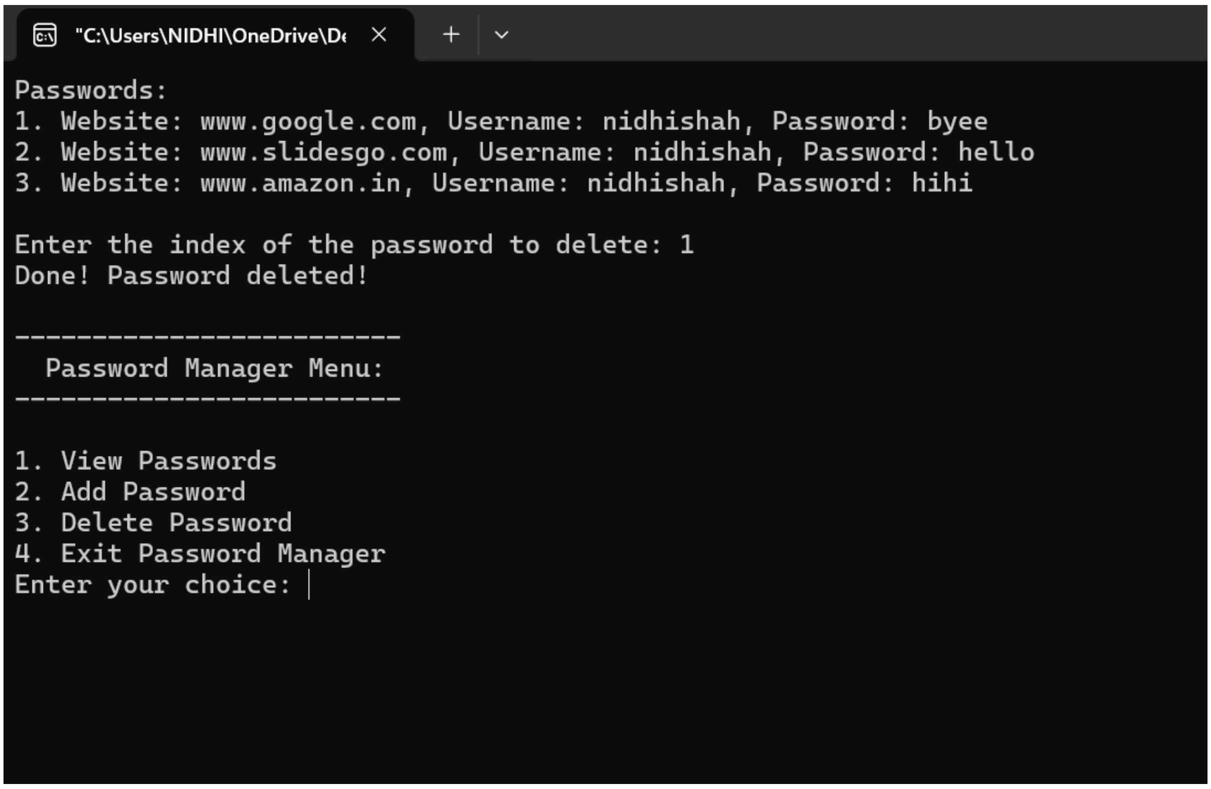
```
Enter website: www.xyz.com
Enter username: nidhishah
Enter password: xyzxyz
Oops, your password manager is feeling a bit cramped. Time to de-clutter!!
-------------------------
  Password Manager Menu:
-------------------------

1. View Passwords
2. Add Password
3. Delete Password
4. Exit Password Manager
Enter your choice:
```

Figure 17.19 Full Passwords File



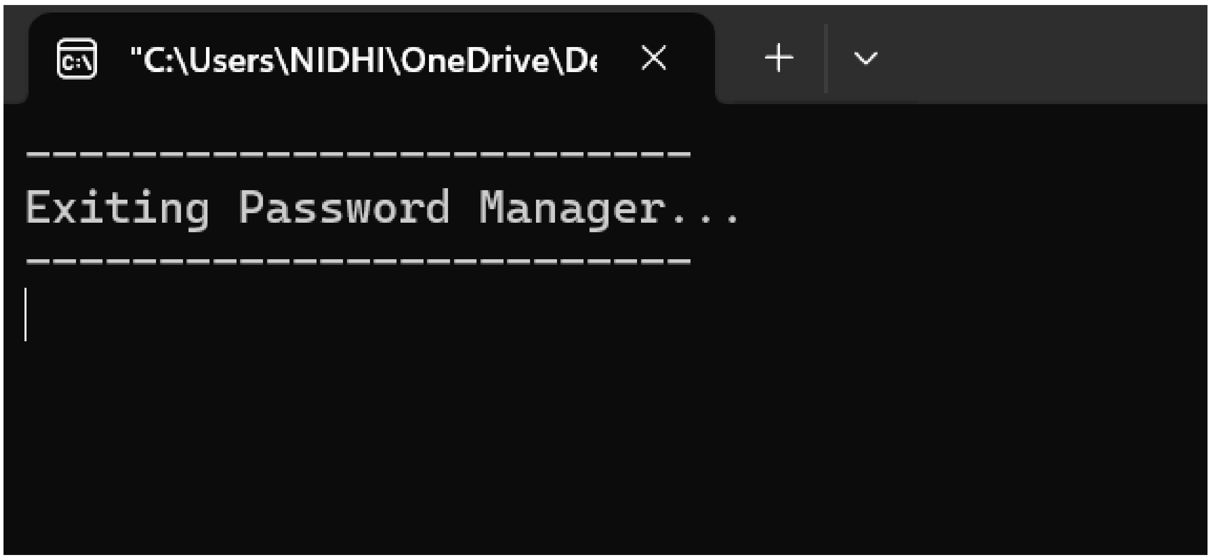
```
Passwords:
1. Website: www.google.com, Username: nidhishah, Password: byee
2. Website: www.slidesgo.com, Username: nidhishah, Password: hello
3. Website: www.amazon.in, Username: nidhishah, Password: hihi

Enter the index of the password to delete: 1
```

Figure 17.20 Delete Password



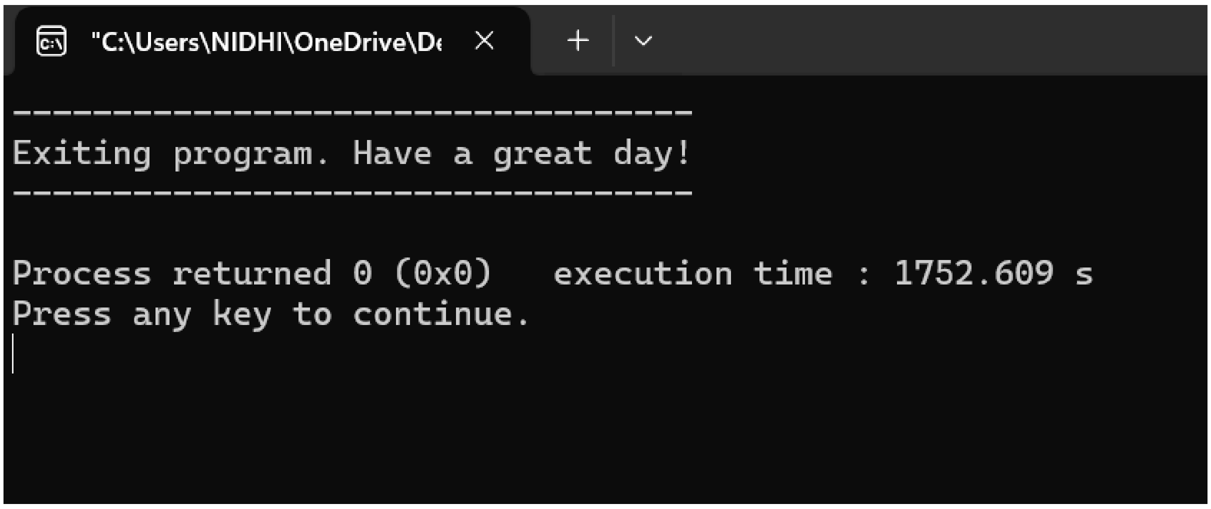
```
Passwords:
1. Website: www.google.com, Username: nidhishah, Password: byee
2. Website: www.slidesgo.com, Username: nidhishah, Password: hello
3. Website: www.amazon.in, Username: nidhishah, Password: hihi

Enter the index of the password to delete: 1
Done! Password deleted!

-------------------------
  Password Manager Menu:
-------------------------

1. View Passwords
2. Add Password
3. Delete Password
4. Exit Password Manager
Enter your choice:
```

Figure 17.21 Deleted Password



```
-------------------------
Exiting Password Manager...
-------------------------
```

Figure 17.22 Exiting Password Manager



```
----------------------------------
Exiting program. Have a great day!
----------------------------------

Process returned 0 (0x0)   execution time : 1752.609 s
Press any key to continue.
```

Figure 17.23 Exiting Program

18. LIST OF ERRORS AND ITS RESOLUTIONS

These are the list of errors and the solutions used while creating the application:

1. File Handling: Every file interaction in the original implementation was done through separate ifstream and ofstream objects. This resulted in confusion and errors related to file pointer management. A method based on arrays was used to solve this. Now, information is taken from the file into an array, worked with inside the application, and then saved again into a fresh, empty file. This minimizes the possibility of pointer-related errors and streamlines file handling.
2. Empty File/Array Handling: Trying to remove a note or password from an empty file or array caused the original code to get into an infinite loop. The loop would never end, even in the case of if-else structures containing break statements. The workaround was to include return statements in void functions so that they would terminate early in the event that they encountered empty data. By doing this, the main function's appropriate program flow is ensured to continue.
3. Array Bounds Issue: Five notes or passwords were the maximum allowed during testing. During addition, the code would, nevertheless, record an additional note (index + 1) in the file. This resulted in out-of-bounds access and was superfluous. This problem was addressed by the return statements added to void functions (discussed in point 2), which prevented passwords and notes from being added in excess of the allowed amount.
4. Login File Handling: The program first determined that there was no user account by determining whether the login file's end-of-file indicator matched the login process's opening line. This strategy did, however, have an unexpected result. It recognized an empty file correctly, but it would store any more user data on the line after that, leaving the first line blank. The file pointer then moved to the following line, causing this blank line to be mistaken for the username. In order to address this, the file pointer movement was controlled by the peek function, which makes sure it stays on the first line for accurate user account detection.

There are currently no errors in the execution of the program.

19. SCOPE OF IMPROVEMENTS

This notes and password manager offers a solid foundation for managing sensitive data. However, future iterations could expand its functionalities. Some improvements could be:

1. Error Handling: Enhance the way errors are handled throughout the code. At present, the program includes error checks for invalid inputs, but to enhance error handling to encompass additional edge cases and deliver more informative error messages to the user.

2. Data Encryption: Before storing confidential information, such as passwords, in a file, it is advised to encrypt it. An additional layer of protection is added to protect user data when sensitive data is encrypted before being saved to a file, particularly when the file is kept on a public system.

3. Dynamic Memory Management: For memory management, it is recommended to employ dynamic data structures like linked lists instead of fixed-size arrays for storing notes and passwords. This functionality provides increased flexibility in dealing with different quantities of notes and passwords.

20. LIST OF REFERENCES/LIST OF RESEARCH PAPERS, BOOKS, WHITE PAPERS REFERRED

[1] E. Balaguruswamy, Object Oriencted Programming with C++, New Delhi: Tata McGraw-Hill Publishing Company Limited, 2008.

Other References:

- C. Sugatan, "The Design and Development of an Interactive Story," University of Michigan, Michigan, 2020.
- GeeksforGeeks, "GeeksforGeeks," GeeksforGeeks, 08 January 2024. [Online]. Available: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/?ref=lbp.
- A. Adithya, "Scribd," 08 January 2020. [Online]. Available: https://www.scribd.com/document/442157428/Password-Manager-File. [Accessed 04 February 2024].
- A. Cainikovs, "stackoverflow," 10 July 2010. [Online]. Available: https://stackoverflow.com/questions/3219393/stdlib-and-colored-output-in-c. [Accessed 04 February 2024].