DLITHE PROJECT REPORT

PROJECT ID : CP037

PROJECT TITLE : Password Management System

TEAM MEMBERS : Varsha R Girish (4MT21CS178)

Srujan S (4MT21CS164)

Sathvik Kulal (4MT21CS136)

Skanda B N (4MT21CS158)

Shraddha Singh (4MT21CS149)

REPORT

Abstract :

A Password Manager System is a software application or service designed to securely store, manage, and organize your passwords and other sensitive information, such as login credentials, credit card details, and personal identification numbers (PINs). Password managers are essential tools for individuals and organizations to enhance security and simplify the process of managing multiple passwords. Below, I'll explain the four main functions of a password manager system:

Introduction :

Background :

Password manager system is a comprehensive tool that helps users and administrators securely manage passwords and maintain a high level of data security. It combines encryption, authentication, and good security practices to protect sensitive information.

Objectives :

Enhanced Security: Password managers generate strong, unique passwords and store them securely, reducing the risk of using weak or easily guessable passwords.
Convenience: Password managers can automatically fill in login forms on websites and apps, saving you time and effort.
Organization: Password managers allow you to organize your passwords into categories or folders, making it easier to manage a large number of credentials.
Encryption: Password managers use strong encryption to protect your stored data, ensuring that even if the password manager's database is compromised, your passwords remain secure.
Cross-Platform Accessibility: Many password managers offer apps and browser extensions for various devices and platforms, allowing you to access your passwords on smartphones, tablets, and computers.
Overall, a password manager system simplifies the process of managing and securing your passwords, making it an essential tool for modern digital life. It helps you maintain strong and unique passwords for each of your accounts, reducing the risk of security breaches and unauthorized access.

Technologies Used :

Database: SQL or NoSQL databases for storing encrypted passwords and user data.

Web Development: Technologies like HTML, CSS, JavaScript, and frameworks such as Angular, React, or Vue for the user interface.

Encryption: AES (Advanced Encryption Standard) for securely storing and transmitting passwords.

Authentication: OAuth, OpenID, or custom authentication mechanisms for user login.

Security: SSL/TLS for secure communication, and security libraries for code protection.

System Architecture :

Front-End :

The front-end of the system is implemented using the console or command-line interface, where users interact with the system by entering commands and providing input.

Back-End :

The back-end of the system consists of C programming logic that performs various operations such as adding, displaying, modifying, and deleting records. It also handles file handling to store and retrieve data.

Database :

The system uses files to act as databases for storing information about username, password. Each entity has a separate file for data storage.

File Handling in this Program:

File Format: Choose a secure and standardized file format, such as JSON or XML, to store password data. This makes it easier to read and write structured data.

File Location: Store the file in a secure location that's not easily accessible by unauthorized users. Avoid storing it in a publicly accessible directory.

Access Control: Implement access control mechanisms to restrict who can read and write to the password data file. Only authorized users (e.g., the admin or the program itself) should have access.

Error Handling: Implement robust error handling for file operations. Handle scenarios where the file may be missing, corrupted, or inaccessible gracefully.

Project Modules :

The project consists of the following modules:

Module 1 Insert:

Inserting or adding a password to a password manager involves entering the login information for a specific account or website. This typically includes a username or email address and the

associated password. Password managers often allow you to categorize or label passwords for easier organization. You may also have the option to add additional information, such as the website URL, account number, or notes related to the login.

Module 2 Modify:

Modifying or updating a password within a password manager is crucial for maintaining security. Passwords should be changed regularly to reduce the risk of unauthorized access to your accounts. To modify a password in a password manager, you can select the specific entry for the account you want to update and then edit the stored information. This usually involves changing the existing password to a new, stronger one.

Module 3 Search:

Searching for a stored password is an essential feature of any password manager, as it allows you to quickly locate the login information you need. You can typically search for passwords by entering keywords, website names, or other relevant information in the search bar of the password manager. The tool will then display a list of matching entries.

Module 4 Delete:

Deleting a password entry from your password manager may be necessary if you no longer use the associated account or if you want to remove outdated or obsolete information. To delete a password entry, you typically select the entry and choose the "delete" or "remove" option. This action removes the stored information from the password manager's database.

Design and Implementation :

Front-End Design :

- The front-end design is based on a command-line interface, where users can select options by entering numbers corresponding to their desired actions.

Back-End Design :

- The back-end logic is implemented in the C programming language.
- Each module has its functions for adding, displaying, modifying, and deleting records.
- File handling functions are used to perform operations on data files.

Database Design :

- Data for password and username are stored in separate files.
- Each file contains records in a structured format.

Testing :

Unit Testing :

- Each module is tested individually to ensure that it performs its functions correctly.
- Test cases are created to cover various scenarios.

Integration Testing :

- Modules are integrated to test the overall functionality of the system.
- Data flow between modules is tested.

User Acceptance Testing :

- The system is tested by end-users to ensure it meets their requirements and expectations.
  - Any feedback or issues raised by users are addressed and resolved.

Challenges Faced :

- Implementing file handling for data storage and retrieval.
- Ensuring data consistency and accuracy.
- Handling errors and exceptions gracefully.
- Future Enhancements
- Implementing a graphical user interface (GUI) for a more user-friendly experience.
- Adding features for generating reports and statistics.
- Implementing security measures to protect data.

Conclusion :

In conclusion, a password management system is a vital tool for securely storing, organizing, and accessing passwords. It enhances security by enforcing policies, encrypting sensitive data, and providing an intuitive user interface. Additionally, features like role-based access control, audit logging, and automated backups contribute to robust password management and data protection.

References :

Academic Journals: Search academic databases like IEEE Xplore, ACM Digital Library, and Google Scholar for research papers and articles on password management and security.

Websites and Blogs: Explore reputable websites and blogs related to cybersecurity and IT security. Websites like OWASP (Open Web Application Security Project) often provide valuable information.

Appendices:

In a comprehensive project or documentation for a password management system, you may include appendices to provide supplementary information, resources, or additional details. Here are some potential appendices you can consider:

1. Appendix A: Glossary
   - Definitions of technical terms and acronyms used throughout the document.

2. Appendix B: User Manual
   - A detailed guide for users on how to use the password management system.

3. Appendix C: Admin Manual
   - Instructions for administrators on how to configure, maintain, and manage the system.

4. Appendix D: Data Flow Diagrams
   - Diagrams illustrating the flow of data within the system, including user interactions and data processing.

5. Appendix E: Sample Reports
   - Examples of reports or analytics generated by the system.

6. Appendix F: Error Code Reference
   - A list of error codes and their meanings for troubleshooting purposes.

7. Appendix G: Regulatory Compliance
   - Details on how the system complies with relevant data protection regulations (e.g., GDPR, HIPAA).

8. Appendix H: Security Measures
   - Information on the security measures implemented in the system, including encryption, access controls, and threat mitigation strategies.

9. Appendix I: Third-Party Tools and Libraries
   - A list of any third-party tools, libraries, or software components used in the development of the system, along with their licenses and references.

10. Appendix J: Backup and Recovery Procedures
    - Detailed steps for performing data backups and recovery in case of data loss or system failures.

11. Appendix K: User Survey Results
    - If applicable, the results of user surveys or feedback gathered during system testing or usage.

12. Appendix L: References
- A list of all external sources, documents, or research materials referenced in the main document.

13. Appendix M: Change Log
- A record of changes, updates, and version history for the document and the password management system.

Screenshots :

main.c

```c
#include <stdio.h>
#include <string.h>

struct User {
    char username[50];
    char password[50];
};

// Function to check if a user exists in the database
int userExists(const char *username, FILE *file) {
    struct User user;
    rewind(file); // Reset file pointer to the beginning
    while (fscanf(file, "%s %s", user.username, user.password) == 2) {
        if (strcmp(username, user.username) == 0) {
            return 1; // User found
        }
    }
    return 0; // User not found
}

// Function to add a new user to the database
void addUser(FILE *file) {
    struct User newUser;
    printf("Enter username: ");
    scanf("%s", newUser.username);

    // Check if the user already exists
```

```c
    // Check if the user already exists
    if (userExists(newUser.username, file)) {
        printf("User already exists\n");
        return;
    }

    printf("Enter password: ");
    scanf("%s", newUser.password);

    fprintf(file, "%s %s\n", newUser.username, newUser.password);
    printf("User added successfully!\n");
}

// Function to authenticate a user
int authenticateUser(FILE *file, char *username, char *password) {
    struct User user;
    rewind(file); // Reset file pointer to the beginning
    while (fscanf(file, "%s %s", user.username, user.password) == 2) {
        if (strcmp(username, user.username) == 0 && strcmp(password, user.password) == 0) {
            return 1; // Authentication successful
        }
    }
    return 0; // Authentication failed
}
```

main.c

```c
int main() {
    FILE *file = fopen("passwords.txt", "a+"); // Change the file extension to ".txt"

    if (!file) {
        printf("Error opening file!\n");
        return 1;
    }

    int choice;
    char username[50], password[50];

    while (1) {
        printf("1. Log in\n");
        printf("2. Create a new user\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter username: ");
                scanf("%s", username);
                printf("Enter password: ");
                scanf("%s", password);
                if (authenticateUser(file, username, password)) {
                    printf("Login successful!\n");
```

Code Snippets :

```
#include <stdio.h>
#include <string.h>

struct User {
    char username[50];
    char password[50];
};

// Function to check if a user exists in the database
int userExists(const char *username, FILE *file) {
    struct User user;
    rewind(file); // Reset file pointer to the beginning
    while (fscanf(file, "%s %s", user.username, user.password) == 2) {
        if (strcmp(username, user.username) == 0) {
            return 1; // User found
        }
    }
    return 0; // User not found
}

// Function to add a new user to the database
void addUser(FILE *file) {
    struct User newUser;
    printf("Enter username: ");
```

```
    scanf("%s", newUser.username);

    // Check if the user already exists
    if (userExists(newUser.username, file)) {
        printf("User already exists\n");
        return;
    }

    printf("Enter password: ");
    scanf("%s", newUser.password);

    fprintf(file, "%s %s\n", newUser.username, newUser.password);
    printf("User added successfully!\n");
}

// Function to authenticate a user
int authenticateUser(FILE *file, char *username, char *password) {
    struct User user;
    rewind(file); // Reset file pointer to the beginning
    while (fscanf(file, "%s %s", user.username, user.password) == 2) {
        if (strcmp(username, user.username) == 0 && strcmp(password, user.password) == 0) {
            return 1; // Authentication successful
        }
    }
    return 0; // Authentication failed
}

int main() {
    FILE *file = fopen("passwords.txt", "a+"); // Change the file extension to ".txt"

    if (!file) {
        printf("Error opening file!\n");
        return 1;
    }

    int choice;
    char username[50], password[50];

    while (1) {
        printf("1. Log in\n");
        printf("2. Create a new user\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter username: ");
                scanf("%s", username);
                printf("Enter password: ");
                scanf("%s", password);
                if (authenticateUser(file, username, password)) {
                    printf("Login successful!\n");
                } else {
                    printf("Login failed!\n");
                }
                break;

            case 2:
                addUser(file);
                break;

            case 3:
                fclose(file);
                return 0;

            default:
                printf("Invalid choice. Try again.\n");
                break;
        }
    }

    fclose(file);
    return 0;
}
```

OUTPUT :


1. Log in
2. Create a new user
3. Exit
Enter your choice: 1
Enter username: varsha
Enter password: 123
Login failed!
1. Log in
2. Create a new user
3. Exit
Enter your choice: 2
Enter username: varsha
Enter password: 123
User added successfully!
1. Log in
2. Create a new user
3. Exit
Enter your choice: 3
...Program finished with exit code 0
Press ENTER to exit console.