



**VIT<sup>®</sup>**

**Vellore Institute of Technology**

(Deemed to be University under section 3 of UGC Act, 1956)

---

# **PASSWORD MANAGER APPLICATION**

*Submitted by*

**Aleena 20BCE0200**

*Submitted to*

**Professor Ruby D**

---

**School of Computer Science and Engineering**

**Summer 2023-24**

**CSE3502 – Information Security Management**

*J Component Report*

# Abstract

In today's digital age, managing passwords securely is paramount to safeguarding personal and sensitive information. This project aims to develop a secure and user-friendly password manager application that addresses the challenges of secure credential storage, retrieval, and management. The application incorporates advanced encryption techniques, multi-factor authentication (MFA), and robust recovery mechanisms to ensure the confidentiality and integrity of user credentials. By leveraging Python's versatile libraries and a modular architecture, the application provides a seamless user experience through a command-line interface (CLI) while promoting strong password practices and intuitive usability. This report details the methodology, implementation, and features of the password manager, highlighting its contributions to enhancing digital security practices and providing users with a reliable tool for managing their online identities securely.

## Introduction

In today's digital age, individuals and organizations rely heavily on numerous online services, each requiring unique login credentials. With the proliferation of online accounts for email, social media, banking, work, and entertainment, managing these credentials has become increasingly challenging. Password fatigue, the practice of reusing passwords across multiple sites, and the creation of weak passwords pose significant security risks. These issues are compounded by the increasing sophistication of cyber threats, making it imperative for users to adopt strong and unique passwords for each service. Password managers have emerged as an essential tool in this context. They are designed to help users generate, store, and manage their passwords securely. A password manager can create complex passwords that are difficult to crack, store them securely using encryption, and auto-fill them when needed, thus reducing the cognitive load on users and enhancing overall security.

The motivation behind developing a new password manager application stems from several factors. With the rise in cyber-attacks, including phishing, keylogging, and brute force attacks, there is a pressing need for robust password management solutions. By offering a secure way to store and manage passwords, a password manager can significantly reduce the risk of unauthorized access. Many users struggle with remembering multiple complex passwords. A password manager simplifies this by securely storing passwords and making them easily accessible, thereby improving user experience. Encouraging users to adopt strong, unique passwords for each of their accounts is a critical cybersecurity practice. A password manager promotes this by automatically generating strong passwords and eliminating the need for users to remember them. The tendency to reuse passwords or use simple, easy-to-remember passwords is a major security vulnerability. By automating password generation and storage, a password manager helps mitigate this issue. For both individuals and organizations, managing passwords efficiently can save significant time and effort. This is particularly important in professional settings where employees need to access multiple systems and applications securely and quickly. As technology evolves, so do the methods used by malicious actors to breach security systems. A modern password manager must adapt to these changes by incorporating advanced encryption techniques, multi-factor authentication, and secure storage solutions.

Given these motivations, the objective of this project is to develop a secure and user-friendly password manager that secures user credentials using encryption and multi-factor authentication. The application aims to provide users with the ability to safely store, retrieve, and manage their passwords and user IDs, while also offering robust recovery mechanisms in case of forgotten passwords or lost access. The system will include features for securely adding, updating, deleting, and searching for credentials, along with mechanisms to provide limited access through alternate login method that is used for emergencies. Along with secure storage, multi-factor authentication, the application will also offer the feature of strong password generation, and all these will be presented to the user with an intuitive user interface. The project will prioritize user experience and data security, ensuring that users have reliable access to their credentials while minimizing the risk of unauthorized access. By doing so, I hope to contribute to a safer digital environment and provide users with a reliable tool to manage their credentials efficiently.

## Related Work

The literature surrounding password managers is extensive, addressing various aspects such as usability, security, user adoption, and the challenges and opportunities in password management. Several key studies contribute to our understanding of how password managers are designed, used, and evaluated.

Simmons et al. (2021) conducted a comprehensive study that systematized seventeen use cases and seventy-seven design paradigms for password managers. By reviewing documentation and literature, the authors highlighted significant usability issues in existing desktop managers. They discovered that browser-based managers often suffer from limited functionality and security compared to extension-based managers. This study underscores the necessity for further research to improve password manager designs and usability, particularly by transitioning from browser-based to extension-based solutions. [1]

In a study by Oesch et al. (2022), the real-world usage of password managers was explored through observational interviews with 32 users. This research revealed that many users utilize both browser-based and third-party password managers concurrently, often without a deliberate strategy. Issues such as difficulties in using generated passwords across devices and the overwhelming nature of credential audit tools were identified. The study suggests that usability improvements in password generation, credential audits, and mobile password managers are needed to enhance user experience and adoption. [2]

Oesch and Ruoti (2020) evaluated the security of 13 popular browser-based password managers, examining password generation, storage, and autofill functionalities. They found improvements in encryption and safer autofill processes, but also identified persistent issues like unencrypted metadata and unsafe defaults. The study emphasizes the need for users to carefully select and configure password managers, recommending enhancements in filtering weak passwords and improving master password policies. [3]

Pearman et al. (2019) investigated the adoption and effective use of password managers through semi-structured interviews with 30 users. The study found that users of built-in managers prioritize convenience, often leading to weaker password practices, while users of separately installed managers prioritize security. Barriers such as security concerns, lack of

awareness, and usability issues were highlighted. The study calls for more tailored designs and improved usability testing to promote secure password management practices across diverse user groups. [4]

Ray et al. (2021) focused on the adoption of password managers among older adults. They found that older adults exhibit higher mistrust of cloud storage and synchronization but are more likely to use password managers when recommended by family. The study suggests advocacy and education as strategies for promoting password manager adoption among older adults, emphasizing the need for further research to address these demographic-specific barriers. [5]

Fernando et al. (2023) reviewed current solutions in password management, highlighting the enduring challenges despite various advancements. The review categorizes password managers into software and hardware-based types, discussing their respective advantages and limitations. The study identifies a research gap in the need for fully automated password management processes that address usability, security enhancements, backup mechanisms, and resistance to attacks. [6]

Petrov (2022) examined the data storage security of Android password managers Bitwarden and Keeper. Through advanced reverse engineering techniques, the study uncovered several vulnerabilities and demonstrated proof-of-concept attacks exploiting these weaknesses. The findings provide valuable insights for improving data storage security in password managers, suggesting future research on antipatterns and alternative security approaches. [7]

Pandare et al. (2023) introduced an enhanced password manager extension for web browsers, employing a hybrid approach of SHA-256 and AES encryption to ensure robust data security. The system was implemented using the Trongate development platform and stored data in a phpMyAdmin database. While the approach integrates strong cryptographic methods, potential drawbacks include scalability concerns and dependency on a single database system. [8]

Dhanalakshmi et al. (2023) presented a password manager with multi-factor authentication (MFA) to enhance security. The system uses AES-256 encryption and PBKDF2 hashing for the master password, offering both local and cloud-based storage options. The implementation includes biometric factors like fingerprints for MFA. The study highlights potential security bugs due to the complexity of MFA integration and the risk of compromising the master password hash in cloud-based scenarios. [9]

Jeong and Jung (2021) introduced MonoPass, a password manager designed to eliminate the use of a master password for authentication. The system generates unique passwords locally using PBKDF2 hashing and HMAC-SHA256/512 encryption, avoiding central password storage. While MonoPass addresses security concerns by decentralizing password storage, it relies on a central server for metadata synchronization, limiting functionality without an internet connection. [10]

Despite the advancements in password manager design and functionality, several research gaps remain.

1. Many studies highlight significant usability issues, particularly in cross-device usage, password generation, and credential audits [1, 2, 4]. The overwhelming nature of these tools and the lack of intuitive user interfaces deter effective adoption and use.
2. Security concerns persist, such as unencrypted metadata, unsafe default settings, and vulnerabilities in password storage and autofill functionalities [3, 7]. These undermine the reliability of many password managers. Users also face risks associated with cloud-based storage, including the potential compromise of master password hashes.
3. There is also a need for more tailored designs to address the specific needs of diverse user groups, including older adults and less tech-savvy individuals [5, 6]. There exist user adoption barriers due to mistrust of password managers, and concerns about cloud storage and synchronization. Additionally, a lack of awareness and education on the benefits and secure practices of password management hinders widespread adoption.
4. The integration of multi-factor authentication, advanced encryption techniques, robust recovery mechanism, and the development of fully automated password management processes require further exploration to enhance both security and usability [8, 9, 10]. These features are essential to adapt to evolving cyber threats and to provide comprehensive security for user credentials.

Future research should focus on these areas to improve the overall effectiveness and adoption of password managers.

Given these challenges, there is a pressing need for a new password manager application that addresses these shortcomings by offering enhanced security, usability, and user adoption features. This project delivers the need to develop a secure and user-friendly password manager that:

- Secures user credentials using state-of-the-art encryption and multi-factor authentication.
- Provides seamless cross-device functionality and an intuitive user interface to improve user experience.
- Includes features for securely adding, updating, deleting, and searching for credentials, along with robust recovery mechanisms for forgotten passwords or lost access.
- Promotes the adoption of strong, unique passwords by automating password generation and storage.
- Adapts to evolving security threats by incorporating advanced cryptographic methods and secure storage solutions.

By addressing these issues, the project aims to provide a reliable tool that enhances the overall security of users' online accounts, reduces the cognitive load of managing multiple passwords, and contributes to a safer digital environment.

# System Architecture

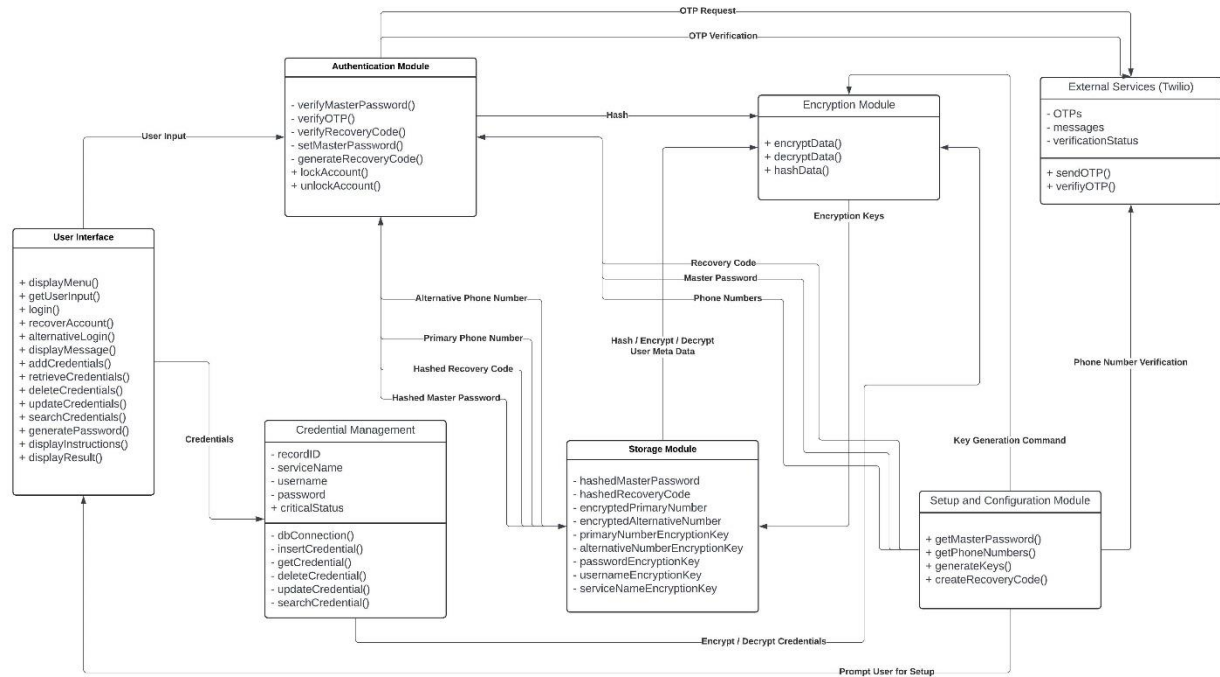


Figure 1 Class Diagram for Password Manager Application

## Methodology

The methodology of developing the password manager begins with a thorough planning phase where the project goals, requirements, and constraints were clearly defined. The primary objective was to create a secure and user-friendly application that can manage passwords and critical credentials, incorporating multiple layers of authentication and robust recovery mechanisms.

### I. Initial Setup

The first-time setup process ensures that the application is correctly configured and ready for use. This process is crucial for establishing a secure environment from the outset.

#### 1. Master Password Setup

The master password setup process is a critical component of the password manager. It involves ensuring that users set a strong, secure master password, which is then hashed and stored securely. This password acts as the primary key to access all other stored credentials.

**A. Prompting the User for a Master Password:** During the initial setup, users are prompted to enter a master password. This is done using a secure input method to ensure the password is not displayed on the screen. The user is also asked to confirm the password to avoid any typographical errors. The entered password and the

confirmation password are compared. If they match, the process proceeds to the next step. If not, the user is prompted to re-enter the passwords until they match.

- B. Password Strength Check:** The password is checked against predefined strength criteria to ensure it is secure. These criteria include a minimum length of 12 characters, and inclusion of at least one lowercase letter, at least one uppercase letter, at least one digit, at least one special character. If the password meets all the requirements, it is considered strong. Otherwise, the user is prompted to enter a new password that meets the strength criteria.
- C. Hashing and Secure Storage:** Once a strong password is confirmed, it is hashed using the bcrypt hashing algorithm. The hashed password is stored securely in a file, ensuring that the plaintext password is never stored, thus reducing the risk of compromise and unauthorized access.

#### *bcrypt Hashing Algorithm*

bcrypt is a password hashing function designed to be computationally intensive and slow. This deliberate slowness helps protect against brute-force attacks, by making it computationally expensive and impractical to hash each password, even with the prevalence of powerful computing resources. It is specifically designed and is widely considered a reliable choice for securely hashing passwords. When a password is hashed using bcrypt, a random salt is generated and combined with the password before hashing. Salting prevents the same password from being hashed to the same value across different users or even multiple uses by the same user, thus preventing attackers from using precomputed hash tables (rainbow tables) effectively. bcrypt uses the Blowfish cipher internally. It applies a key derivation function (KDF) that includes multiple iterations (called rounds) of hashing to slow down the hashing process. The number of rounds is a parameter that can be adjusted to increase the computational workload required to hash passwords. The result of bcrypt hashing is a hashed password string that includes both the salt used and the final hashed password. This string can be safely stored in a database or file. Even if the password file is compromised, the actual passwords remain secure due to the computational effort required to reverse-engineer the hashes.

## **2. Recovery Code Generation**

The recovery code generation process is a crucial step to ensure that users can regain access to their accounts if they forget their master password. This process involves generating a secure, random recovery code, hashing it, and storing it securely.

- A. Random Recovery Code Generation:** A random recovery code is generated using a combination of uppercase letters, lowercase letters, and digits. This ensures a strong and unpredictable code that is difficult to guess. After generation, the user will be provided with the original recovery code and instructed to save it securely. This notification ensures that users are aware of the importance of the recovery code and are encouraged to store it in a safe place.
- B. Hashing and Secure Storage:** Once the recovery code is generated, it is hashed using secure bcrypt hashing algorithm to ensure its security. Hashing the recovery code

ensures that even if someone gains access to the stored hash, they cannot easily reverse-engineer the original code. The hashed recovery code is then stored in a secure file.

### 3. Key Generation

Cryptographic keys are generated to encrypt and decrypt sensitive data within the application. These keys are stored securely and are essential for maintaining the integrity and confidentiality of the sensitive information stored in the database.

**A. Generate Symmetric Keys:** Symmetric keys are generated for encrypting different types of data within the application. Symmetric encryption involves using the same key for both encryption and decryption, making the secure handling of these keys vital. Separate keys are generated for different types of data. This compartmentalization enhances overall security by ensuring that a compromise of one key does not affect other data types. Keys are generated for the user's primary and alternative phone numbers, and for the usernames, passwords and service names that will be stored in the application database.

The Fernet symmetric encryption is used to generate most of the encryption keys. Fernet uses AES (Advanced Encryption Standard) in CBC (Cipher Block Chaining) mode with a 128-bit key. AES is a widely accepted and secure encryption standard, ensuring strong data protection. Fernet also uses HMAC (Hash-based Message Authentication Code) to ensure data integrity. This means that any tampering with the encrypted data can be detected. Keys are Base64 encoded, making them safe for storage and transport as ASCII strings.

FPE (Format-Preserving Encryption) is used for service names to maintain their format while still encrypting the data. This allows encrypted service names to be used in database queries and lookups without revealing their actual content, thus ensuring both security and functionality.

**B. Secure Storage:** Each generated key is securely stored in a separate file ensuring that the keys are kept secure and are only accessible by authorized parts of the application.

### 4. Phone Number Registration

The phone number registration process is an essential part of ensuring secure access to the password manager. The procedure involves collecting, verifying, and securely storing both a primary and an alternative phone number for the user. These phone numbers are critical for OTP (One-Time Password) verification during login and account recovery processes, providing an additional layer of security.

**A. Phone Number Collection:** Users are prompted to enter their primary phone number first. After the primary number is successfully registered and verified, users are asked to enter an alternative phone number. The registration of both a primary and an alternative phone number ensures that there is always a backup option. If the primary phone number is unavailable for any reason, the alternative number can be used to receive OTPs, ensuring continuous access to the account and reducing the risk of being locked out.



- B. OTP Verification:** Upon entering a phone number, an OTP is sent to the provided number using an SMS service (such as Twilio). The user must enter the OTP received on their phone within a specified timeframe to verify their phone number. If the OTP is entered correctly within the allowed time, the phone number is considered verified. This ensures that the phone number provided by the user is valid and accessible, adding an extra layer of security.
- C. Secure Storage:** Once verified, the phone number is encrypted using Fernet symmetric encryption. Encrypting phone numbers with Fernet ensures that any tampering with the stored data can be detected, maintaining data integrity. Separate encryption keys are used for the primary and alternative phone numbers, ensuring that each number is encrypted independently for added security. The encrypted phone numbers are then stored securely in files.

## II. Authentication Mechanisms

### 1. Master Password Verification

The master password verification ensures that only authorized users can access their stored credentials.

- A. Password Input and Verification:** The user is prompted to input their master password. The provided password is hashed and compared against the stored hash. If the hashed password matches the stored hash, the user is authenticated and allowed to proceed.
- B. Failure Handling and Lockout:** The system keeps track of the number of failed password attempts. If the number of failed attempts exceeds a predefined limit, the account is locked for a specified duration. This mechanism helps deter brute force attacks.

### 2. OTP Verification

OTP (One-Time Password) verification adds an extra layer of security.

- A. OTP Generation and Sending:** An OTP is generated and sent to the user's registered phone number. The user must enter the OTP correctly within a limited number of attempts and within a specified timeframe.
- B. Failure Handling and Lockout:** Similar to password verification, failed OTP attempts are tracked. Upon exceeding the allowed attempts, the account is locked for a set period of time to prevent continuous guessing.

### 3. Recovery Code Verification

In case users forget their master password, they can recover their account using the recovery code. Users input their recovery code, which is hashed and compared with the stored hash. If it matches, an OTP is sent to the primary phone number for further verification. The system limits the number of recovery attempts to prevent unauthorized access, and exceeding the allowed recovery attempts triggers a lockout similar to the normal login procedure. The number of attempts and lockout duration may differ from normal login.

#### **4. Alternative Login**

An alternative login method is provided for situations where the primary phone number is inaccessible. The user inputs their master password, which is verified as described above. After verification, an OTP is sent to the alternative phone number instead of the primary number. The user must enter the correct OTP to gain access. Failed attempts for alternative login are tracked separately from ordinary login. Exceeding the allowed number of failed attempts triggers a lockout for a specified period. The number of attempts and lockout duration may differ from normal login. Since the secondary phone number is being used for user verification, some additional layers of security are included in case an unauthorized entity has managed to gain access to the master password and alternative phone.

- After verification of master password when OTP is sent to the alternative phone, another OTP is simultaneously sent to the primary phone number. This will notify the authorized user in case they are not the ones attempting to access their Password Manager app, and enable them to inform the admin/customer service to lock the app.
- Most of the features of the app will be disabled in the alternative login mode. Only a few credentials, which were marked as most critical, will be viewable, while others will be inaccessible. Similarly, edit access is also denied.

### **III. Credential Management**

The core functionality of the application is credential management, and it is designed to ensure secure handling of user credentials through various operations such as adding, retrieving, updating, deleting, and searching for credentials. Each operation involves a sequence of steps that ensure security and data integrity.

#### **1. Adding Credentials**

Users are prompted to enter details such as service name, username, password, and criticality of the service. Validation checks ensure the service name adheres to specified constraints (maximum length and valid characters). Users can mark a service as critical, influencing access restrictions during alternative login scenarios. Users are asked to confirm the details before they are saved. Once confirmed, the credentials are encrypted using distinct cryptographic keys for different fields. The encrypted credentials are then stored in the database.

#### **2. Retrieving Credentials**

The user can request the credentials for a specific service. The stored encrypted credentials are decrypted before being displayed to the user in a secure manner. Additionally, if the user has logged in via alternative method, only the critical flagged credentials can be retrieved, while access to all other credentials will be denied.

#### **3. Updating Credentials**

Users can choose to update specific fields (e.g., service name, username, password, critical status) or all fields of a service credential. Validation and confirmation steps

follow to ensure accuracy and intentional updates. The relevant credentials are encrypted again before updating the database.

#### **4. Deleting Credentials**

Users can delete credentials for a specific service. Upon confirmation, the credentials are securely removed from the database.

#### **5. Searching for Credentials**

Users can search for services using a search term. The application fetches all services and decrypts them to find matches. Matching services and corresponding usernames are displayed to the user.

### **IV. Password Generator**

The password generator in this application is designed to create strong and secure passwords based on user-specified criteria.

#### **1. User Customization Input**

The user specifies the desired length of the password and whether special characters should be included.

#### **2. Character Selection**

The application constructs a pool of characters from which the password can be generated. This pool always includes uppercase and lowercase letters and digits, and if the user opts to include special characters, then these are added to the pool (e.g., [@\$%Z^\_\*-]).

#### **3. Random Selection**

The password is created by randomly selecting characters from the specified pool until the password reaches the desired length. This ensures that the password is both random and strong, reducing the likelihood of it being easily guessed or cracked.

### **V. User Interface and User Experience**

The application employs a Command-Line Interface (CLI) designed for simplicity and usability. The logic of the CLI revolves around the following key principles:

#### **1. Clear Instructions**

- **Step-by-Step Guidance:** The CLI provides clear and concise instructions at each step, ensuring that the user understands what input is required. For example, during login, the user is prompted to enter the master password and then an OTP, with explicit prompts and feedback at each stage.
- **Error Handling:** When the user makes an error, such as entering an incorrect password or OTP, the system provides informative messages and the number of remaining attempts.

#### **2. Enhanced Usability**

- **Structured Menus:** The CLI is organized into menus and sub-menus that guide users through various functionalities. For example, after logging in, the user is presented with options to add, retrieve, update, or delete credentials, among other actions.
- **Feedback and Confirmation:** The CLI frequently asks for user confirmation before performing critical actions, such as changing a password or deleting credentials. This helps prevent accidental changes.
- **Lockout Mechanism:** To enhance security, the CLI includes lockout mechanisms that temporarily block access after too many failed attempts. The user is informed of the lockout and the duration they need to wait before retrying.
- **Instructions Command:** The CLI includes an option to view detailed instructions on how to use the application, ensuring that users can always refer to guidance if needed.

## Implementation

Python was chosen as the primary programming language for developing the password manager application due to its versatility, readability, and extensive support for libraries and frameworks essential for secure application development. The several Python libraries used in the implementation are –

- **bcrypt:** Used for hashing passwords securely before storing them in the database. Bcrypt is known for its resistance to brute-force attacks due to its adaptive hashing algorithm.
- **twilio.rest:** Utilized for integrating SMS-based multi-factor authentication (MFA) into the application, enhancing security by verifying user identity through a second factor.
- **cryptography.fernet:** Employed for symmetric encryption and decryption of sensitive data such as user credentials stored locally. Fernet provides a straightforward API for encryption using symmetric keys.
- **pyffx:** Implemented for format-preserving encryption of certain data fields, ensuring that encrypted data maintains its original format (e.g., preserving the structure of phone numbers).
- **getpass:** Integrated to securely prompt users for sensitive information such as master passwords during authentication, ensuring confidentiality by concealing user input.
- **sqlite3:** Used as the relational database management system (RDBMS) for storing encrypted user credentials and application data. SQLite is lightweight, serverless, and well-suited for embedded applications like your password manager.

The application's architecture emphasizes modularity and integration of Python libraries to achieve robust functionality. By leveraging these libraries, the password manager ensures secure storage, retrieval, and management of user credentials while adhering to best practices

in cybersecurity and user authentication. The project's codebase is structured into modular components imported from separate Python files (`first_time_setup.py`, `auth.py`, `db.py`, `encryption.py`) to maintain code clarity and separation of concerns. This modular approach facilitates easier maintenance, debugging, and future enhancements to the application.

## Sample Code

[illegible]

```

        continue
        pattern = re.compile(r'^[a-zA-Z0-9]+$')
        if not bool(pattern.match(service_name)):
            print("Invalid service name!")
            continue
        else:
            break
    username = input("\nEnter service username/email address: ")
    password = input("\nEnter service password: ")
    print("Will you be needing the credentials of this service even during Alternative Login?")
    is_critical = input("Is the service highly critical? (y/n): ")

    if is_critical.lower()=='y':
        is_critical = 1
    else:
        is_critical = 0

    print(f"\nConfirm the details entered - \nService Name: {service_name}\nUsername: {username}\nPassword: {password}\nIs Critical: {'YES' if is_critical else 'NO'}\n")

    confirm = input("Continue to Save?(y/n): ")
    if confirm.lower()=='y':
        if
add_credentials(conn,service_name,username,password,is_critical):
            print("New Credentials successfully added!")

    elif action=='2':
        print("\nRetrieve Credentials\n")
        service_name = input("Enter service name: ")
        username,password = get_credentials(conn,service_name)
        if username and password:
            print(f"\nUsername: {username}\nPassword: {password}")
        else:
            print("No credentials for this service!")

    elif action=='3':
        while True:

```

```

        print("\nUpdate Credentials\n")
        print("1. Change Service Name")
        print("2. Change Service Username")
        print("3. Change Service Password")
        print("4. Change Service Critical Status (is/is not
critical)")

        print("5. Change all")
        print("6. Go back")
        update_action = input("Enter your choice: ")
        if update_action=='1':
            old_service_name = input("Enter current service
name: ")

            new_service_name = input("Enter new service
name:")

            print(f"Changing service name from
{old_service_name} to {new_service_name}")
            confirm = input("Save changes?(y/n): ")
            if confirm.lower()=='y':
                if
update_service(conn,old_service_name,new_service_name):
                    print("Service name successfully
modified!")

            elif update_action=='2':
                service_name = input("Enter service name: ")
                new_username = input("Enter new username: ")
                old_username,_ =
get_credentials(conn,service_name)
                print(f"Changing username from {old_username} to
{new_username}")

                confirm = input("Save changes?(y/n): ")
                if confirm.lower()=='y':
                    if
update_username(conn,service_name,new_username):
                        print("Service Username successfully
modified!")

            elif update_action=='3':

```

```

        service_name = input("Enter service name: ")
        new_password = input("Enter new password: ")
        __,old_password =
get_credentials(conn,service_name)
        print(f"Changing password from {old_password} to
{new_password}")

        confirm = input("Save changes?(y/n): ")
        if confirm.lower()=='y':
            if
update_password(conn,service_name,new_password):
                print("Service Password successfully
modified!")

            elif update_action=='4':
                service_name = input("Enter service name: ")
                new_critical_status = input("Is the service highly
critical? (y/n): ")

                if new_critical_status.lower()=='y':
                    new_critical_status = 1
                    print("Setting service as Critical")
                else:
                    new_critical_status = 0
                    print("Setting service as Not Critical")
                confirm = input("Save changes?(y/n): ")
                if confirm.lower()=='y':
                    if
update_critical_status(conn,service_name,new_critical_status):
                        print("Service Critical Status
successfully modified!")

            elif update_action=='5':
                old_service_name = input("Enter current service
name: ")

                new_service_name = input("Enter new service
name:")

                new_username = input("Enter new username: ")
                new_password = input("Enter new password: ")

```



```

        new_critical_status = input("Is the service highly
critical? (y/n): ")

        old_username,old_password =
get_credentials(conn,old_service_name)

        print(f"Changing service name from
{old_service_name} to {new_service_name}")

        print(f"Changing username from {old_username} to
{new_username}")

        print(f"Changing password from {old_password} to
{new_password}")

        if new_critical_status.lower()=='y':
            new_critical_status = 1
            print("Setting service as Critical")
        else:
            new_critical_status = 0
            print("Setting service as Not Critical")
        confirm = input("Save changes?(y/n): ")
        if confirm.lower()=='y':
            if
update_all(conn,old_service_name,new_service_name,new_username,new_password,new_cr
itical_status):

                print("Service Credentials successfully
modified!")

        elif update_action=='6':
            break

        else:
            print("Invalid choice!")

    elif action=='4':
        print("\nDelete Credentials\n")
        service_name = input("Enter service name: ")
        print(f"Deleting credentials of {service_name} service")
        confirm = input("Save changes?(y/n): ")
        if confirm.lower()=='y':
            if delete_service(conn,service_name):
                print("Service Credentials successfully deleted!")

```

```

elif action=='5':
    print("\nSearch for Services\n")
    search_term = input("Enter search term: ")
    results = search_services(conn,search_term)
    if results:
        print("\nSearch results-\n")
        for i in range(len(results)):
            print(f"Service {i+1}: {results[i][0]}\nUsername:
{results[i][1]}\n")
    else:
        print("No matching services!")

elif action=='6':
    print("\nChange Master Password\n")
    stored_master_password = load_master_password()
    provided_master_password = getpass("Enter current master
password: ")

    if verify_password(stored_master_password,
provided_master_password):
        phone_number, _ = load_phone_numbers()
        status, verified = verify_phone_number(phone_number)
        if status and verified:
            if set_master_password():
                print("Master Password successfully changed!")
            else:
                print("Master Password could not be changed!")
        else:
            print("Incorrect OTP!")
    else:
        print("Incorrect Password!")

elif action=='7':
    while True:
        print("\nChange Linked Phone Numbers\n")
        print("1. Change Primary Phone Number")
        print("2. Change alternative Phone Number")
        print("3. Go back")

```

```

        update_action = input("Enter your choice: ")
        if update_action=='3':
            break
        old_ph_num, old_alt_ph_num = load_phone_numbers()
        if update_action=='1':
            print("\nChanging Primary Phone Number\n")
            stored_master_password = load_master_password()
            provided_master_password = getpass("Enter master
password: ")

            if verify_password(stored_master_password,
provided_master_password):

                if primary_phone_setup():
                    new_ph_num,_ = load_phone_numbers()
                    print(f"Primary Phone Number successfully
changed from {old_ph_num} to {new_ph_num}!")
                else:
                    print("Incorrect Password!")

            elif update_action=='2':
                print("\nChanging Alternative Phone Number\n")
                stored_master_password = load_master_password()
                provided_master_password = getpass("Enter master
password: ")

                if verify_password(stored_master_password,
provided_master_password):

                    if alternative_phone_setup():
                        _,new_alt_ph_num = load_phone_numbers()
                        print(f"Alternative Phone Number
successfully changed from {old_alt_ph_num} to {new_alt_ph_num}!")
                    else:
                        print("Incorrect Password!")
                else:
                    print("Invalid choice!")

    elif action=='8':
        print("Logging out...")
        break
    else:

```

```

        print("Invalid choice!")

    elif choice == '2':
        if recover_account():
            print("\n---PASSWORD MANAGER (RECOVERY MODE)---\n")
            if reset_master_password():
                print("Master Password successfully changed!")
            else:
                print("Master Password could not be changed!")

    elif choice == '3':
        if alternative_login():
            while True:
                print("\n---PASSWORD MANAGER (ALTERNATIVE MODE)---\n")
                print("1. Retrieve Critical Credentials")
                print("2. Change Primary Phone Number")
                print("3. Logout")
                action = input("Enter your choice: ")
                if action=='1':
                    print("\nRetrieve Critical Credentials\n")
                    service_name = input("Enter service name: ")
                    username,password =
get_credentials(conn,service_name,critical_only=True)
                    if username and password:
                        print(f"\nUsername: {username}\nPassword: {password}")
                    else:
                        print("No credentials for this service or service
inaccessible!")

                elif action=='2':
                    print("\nChanging Primary Phone Number\n")
                    stored_master_password = load_master_password()
                    provided_master_password = getpass("Enter master password:
")
                    if verify_password(stored_master_password,
provided_master_password):
                        old_ph_num, alternative_phone_number
= load_phone_numbers()

```

```

        status, verified =
verify_phone_number(alternative_phone_number)
        if status and verified:
            if primary_phone_setup():
                new_ph_num,_ = load_phone_numbers()
                print(f"Primary Phone Number successfully
changed from {old_ph_num} to {new_ph_num}!")
            else:
                print("Incorrect Password!")
                break

        elif action=='3':
            print("Logging out...")
            break

        else:
            print("Invalid choice!")
elif choice == '4':
    print("\n---STRONG PASSWORD GENERATOR---\n")
    length = int(input("Enter the length needed for password: "))
    use_special_chars = input("Should the password include special
characters?(y/n): ")
    if use_special_chars.lower()=='y':
        use_special_chars = True
    else:
        use_special_chars = False
    while True:
        generated_password = generate_password(length,use_special_chars)
        if generated_password:
            print("Strong Password successfully generated!")
            print(f"Password: {generated_password}")
            next_action = "3"
            while next_action!="1" and next_action!="2":
                print("1. Regenerate Password")
                print("2. Exit")
                next_action = input("Enter your choice: ")
                if next_action == '1':
                    print()

```

```

        elif next_action == '2':
            print("Exiting Password Generator...")
        else:
            print("Invalid choice!")
    if next_action=="2":
        break
    else:
        print("Password generation failed! Try again.")
        break

elif choice == '5':
    print("\n---INSTRUCTIONS FOR USING PASSWORD MANAGER APP---\n")
    instructions()

elif choice == '6':
    print('Exiting Password Manager...')
    break

if __name__ == '__main__':
    main()

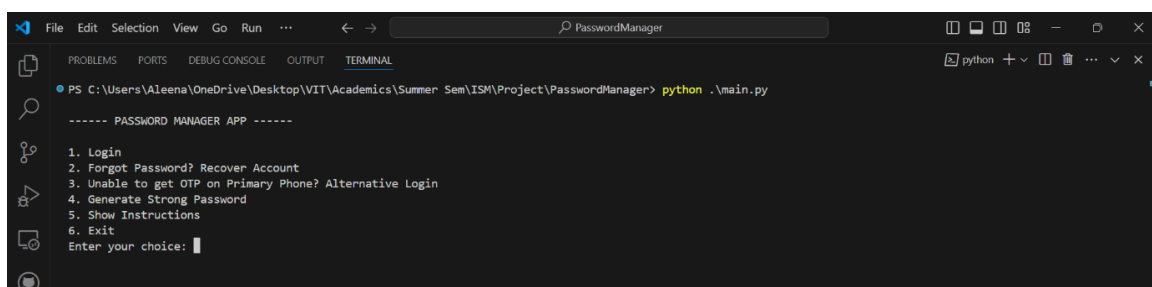
```

To view the complete implementation of the Password Manager Application, visit the following GitHub link:  
<https://github.com/a-leena/PasswordManager>

## Results and Discussion

### Screenshots of Execution

On entering the Password Manager application, the user is presented with several options for further usage of the app. They can choose any mode of login.

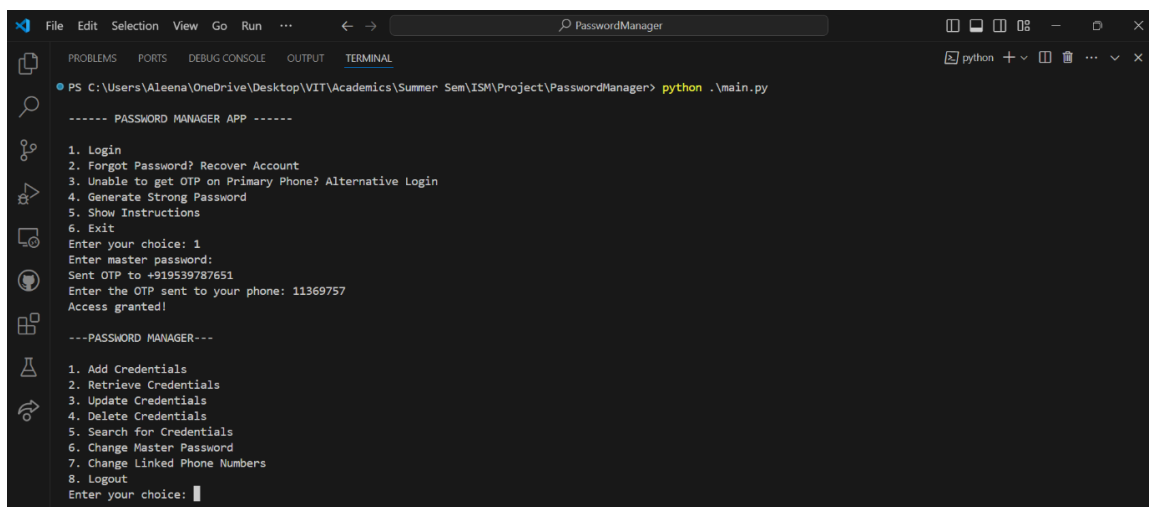


```

File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project\PasswordManager> python .\main.py
----- PASSWORD MANAGER APP -----
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: █

```

On successful normal login, the user is given the option to choose several operations, such as adding, retrieving, updating, deleting, and searching credentials, and additionally also to change their master password or linked phone numbers.



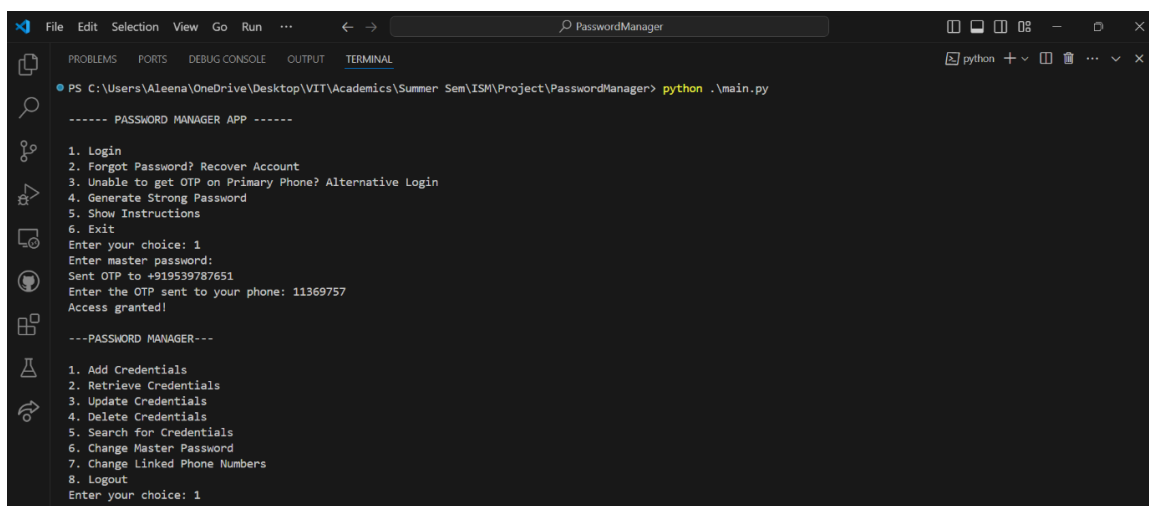
```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project\PasswordManager> python .\main.py

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 1
Enter master password:
Sent OTP to +919539787651
Enter the OTP sent to your phone: 11369757
Access granted!

---PASSWORD MANAGER---

1. Add Credentials
2. Retrieve Credentials
3. Update Credentials
4. Delete Credentials
5. Search for Credentials
6. Change Master Password
7. Change Linked Phone Numbers
8. Logout
Enter your choice: 
```



```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project\PasswordManager> python .\main.py

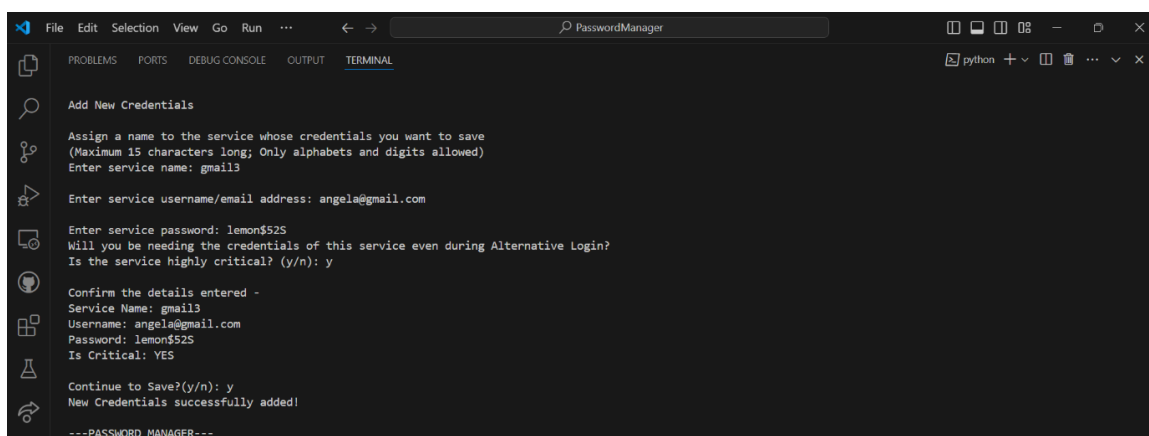
----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 1
Enter master password:
Sent OTP to +919539787651
Enter the OTP sent to your phone: 11369757
Access granted!

---PASSWORD MANAGER---

1. Add Credentials
2. Retrieve Credentials
3. Update Credentials
4. Delete Credentials
5. Search for Credentials
6. Change Master Password
7. Change Linked Phone Numbers
8. Logout
Enter your choice: 1
```

When adding credentials, after the user has provided all required details, they are displayed before saving them into the database.



```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
Add New Credentials

Assign a name to the service whose credentials you want to save
(Maximum 15 characters long; Only alphabets and digits allowed)
Enter service name: gmail3

Enter service username/email address: angela@gmail.com

Enter service password: lemon$525
Will you be needing the credentials of this service even during Alternative Login?
Is the service highly critical? (y/n): y

Confirm the details entered -
Service Name: gmail3
Username: angela@gmail.com
Password: lemon$525
Is Critical: YES

Continue to Save?(y/n): y
New Credentials successfully added!

---PASSWORD MANAGER---
```

```
---PASSWORD MANAGER---
1. Add Credentials
2. Retrieve Credentials
3. Update Credentials
4. Delete Credentials
5. Search for Credentials
6. Change Master Password
7. Change Linked Phone Numbers
8. Logout
Enter your choice: 2

Retrieve Credentials

Enter service name: gmail3

Username: angela@gmail.com
Password: lemon$525

---PASSWORD MANAGER---
1. Add Credentials
2. Retrieve Credentials
3. Update Credentials
4. Delete Credentials
5. Search for Credentials
6. Change Master Password
7. Change Linked Phone Numbers
8. Logout
```

When updating the various fields of the database (service names, usernames, passwords, critical status, or all), after the user has entered the changes they want to make, the old and new data are displayed back to them for confirmation before making any changed in the database.

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL

Enter your choice: 3

Update Credentials

1. Change Service Name
2. Change Service Username
3. Change Service Password
4. Change Service Critical Status (is/is not critical)
5. Change all
6. Go back
Enter your choice: 2
Enter service name: gmail3
Enter new username: angela26@gmail.com
Changing username from angela@gmail.com to angela26@gmail.com
Save changes?(y/n): y
Service Username successfully modified!

Update Credentials

1. Change Service Name
2. Change Service Username
3. Change Service Password
4. Change Service Critical Status (is/is not critical)
5. Change all
6. Go back
Enter your choice: 6
```

The user can choose the search option and enter a query term, which must be a service name or a possible substring of the service name. They will be displayed all the matching service names and their corresponding usernames. This feature is useful if the user cannot completely remember what the exact service name of a particular credential was.

```
---PASSWORD MANAGER---
1. Add Credentials
2. Retrieve Credentials
3. Update Credentials
4. Delete Credentials
5. Search for Credentials
6. Change Master Password
7. Change Linked Phone Numbers
8. Logout
Enter your choice: 5

Search for Services

Enter search term: mail

Search results-

Service 1: gmail1
Username: aleena22@gmail.com

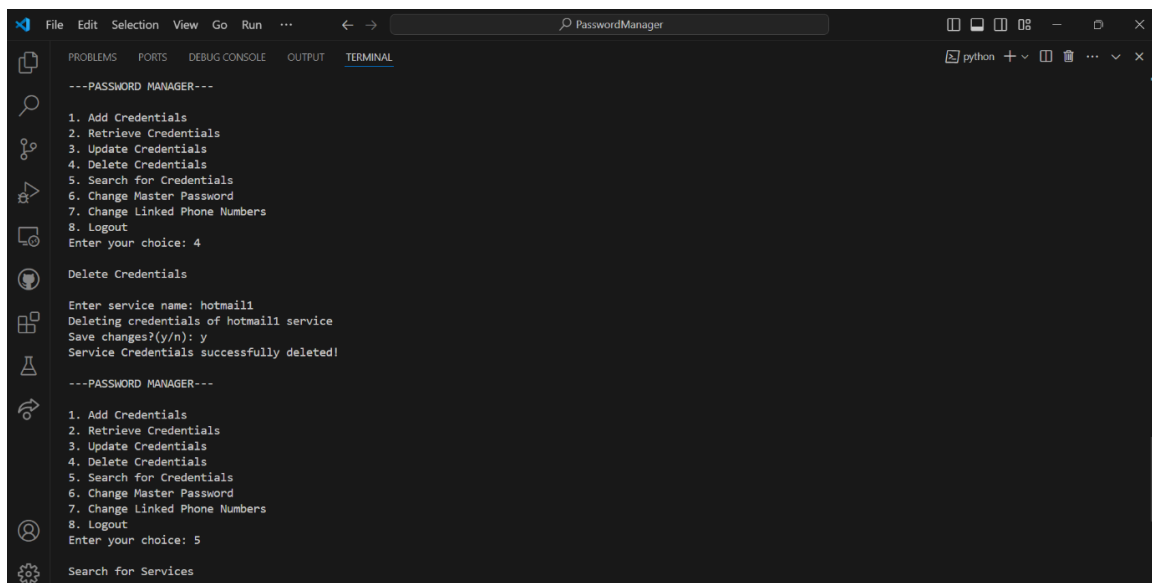
Service 2: gmail2
Username: juniper@gmail.com

Service 3: hotmail1
Username: rupert4Fun@hotmail.com

Service 4: gmail3
Username: angela26@gmail.com
```



Additional confirmation is asked to the user when deleting any record from the database.



```
---PASSWORD MANAGER---
1. Add Credentials
2. Retrieve Credentials
3. Update Credentials
4. Delete Credentials
5. Search for Credentials
6. Change Master Password
7. Change Linked Phone Numbers
8. Logout
Enter your choice: 4

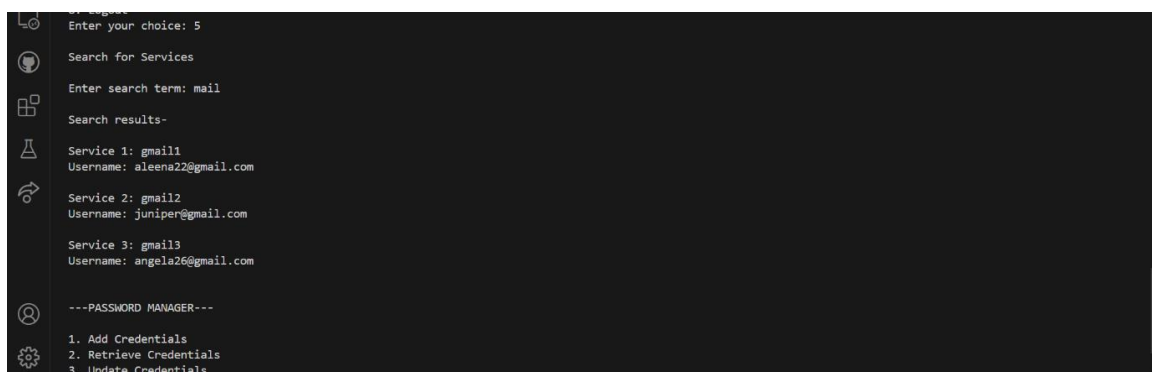
Delete Credentials

Enter service name: hotmail1
Deleting credentials of hotmail1 service
Save changes?(y/n): y
Service Credentials successfully deleted!

---PASSWORD MANAGER---
1. Add Credentials
2. Retrieve Credentials
3. Update Credentials
4. Delete Credentials
5. Search for Credentials
6. Change Master Password
7. Change Linked Phone Numbers
8. Logout
Enter your choice: 5

Search For Services
```

After deletion of a credential, the changes can be immediately viewed when attempting to search for or retrieve that credential.



```
Enter your choice: 5

Search for Services

Enter search term: mail

Search results-

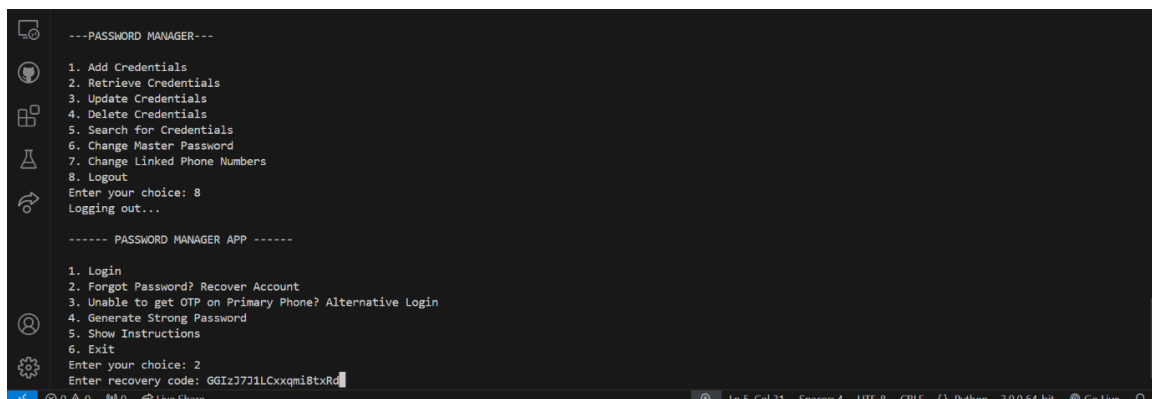
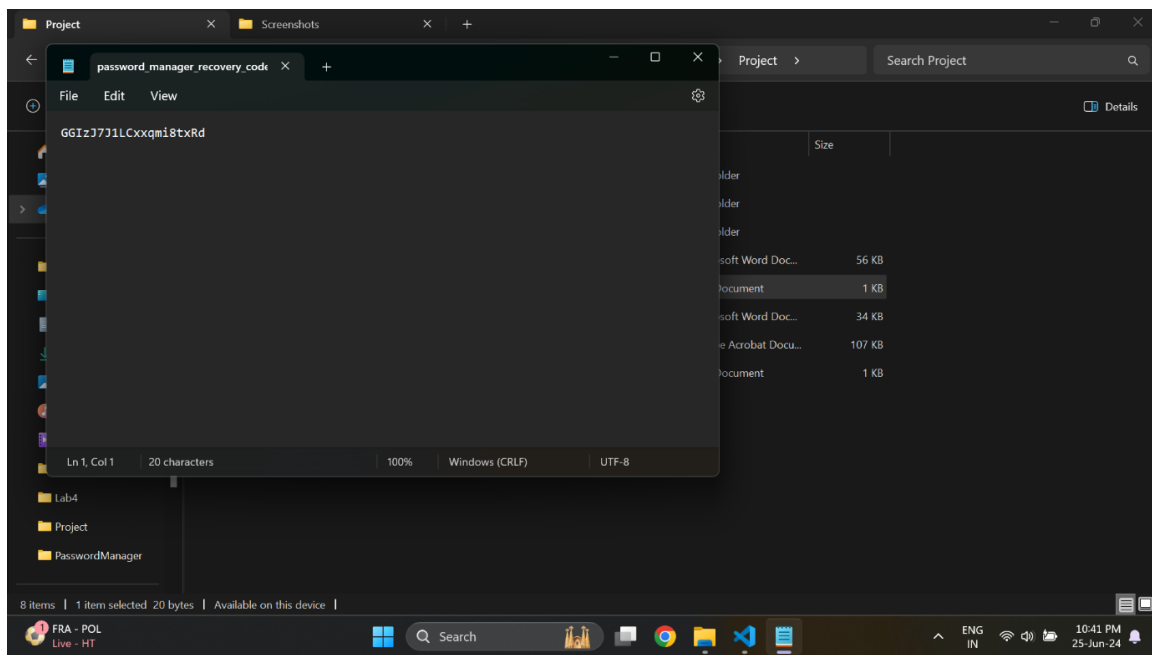
Service 1: gmail1
Username: aleena22@gmail.com

Service 2: gmail2
Username: juniper@gmail.com

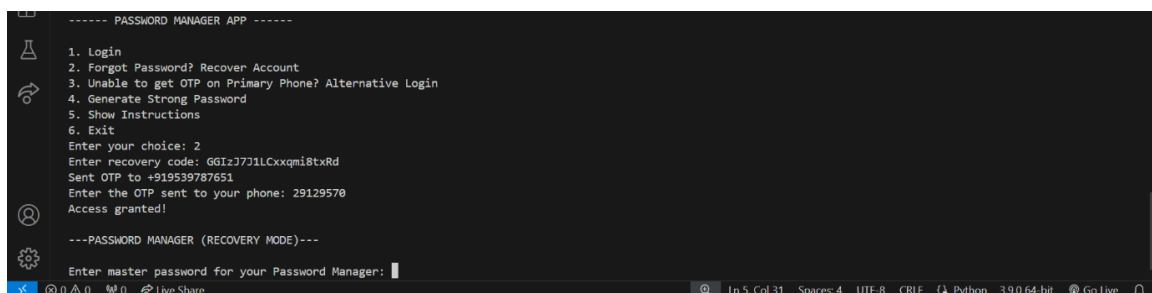
Service 3: gmail3
Username: angela26@gmail.com

---PASSWORD MANAGER---
1. Add Credentials
2. Retrieve Credentials
3. Update Credentials
```

In case the user has forgotten their master password, they can select the option to recover their account. Here they are required to copy/paste or type the recovery code that they were asked to save securely during the application setup.



After having the recovery code verified, and the OTP verified, the user will only have one possible course of action, that is to change their master password.



After entering and re-entering their new master password, the new password will be overwritten in the file where the previous password was stored.



If the user doesn't have access to their primary phone number, either temporarily or permanently, they can choose the Alternative login option. They will be required to enter the master password, as well get OTP verification done through their alternative phone number.

```
----- PASSWORD MANAGER APP -----
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 3
Enter master password:
Sent OTP to +916238265531
Enter the OTP sent to your phone: 51810075
Access granted!

---PASSWORD MANAGER (ALTERNATIVE MODE)---
1. Retrieve Critical Credentials
2. Change Primary Phone Number
3. Logout
Enter your choice: 
```

On successful login via alternative mode, the user has access to only a few functionalities, which will be retrieving critical credentials for emergencies, or changing their primary phone number in case the number has been permanently lost or disabled.

```
1. Retrieve Critical Credentials
2. Change Primary Phone Number
3. Logout
Enter your choice: 1

Retrieve Critical Credentials

Enter service name: gmail

Username: aleena22@gmail.com
Password: abcd@1234

---PASSWORD MANAGER (ALTERNATIVE MODE)---
1. Retrieve Critical Credentials
2. Change Primary Phone Number
3. Logout
Enter your choice: 1

Retrieve Critical Credentials

Enter service name: gmail2
No credentials for this service or service inaccessible!

---PASSWORD MANAGER (ALTERNATIVE MODE)---
1. Retrieve Critical Credentials
2. Change Primary Phone Number
3. Logout
Enter your choice: 
```

The application can generate strong passwords for the user, without having them to login. The user will be asked to give the desired length and also whether the password must contain any special characters.

```
3. Logout
Enter your choice: 3
Logging out...

----- PASSWORD MANAGER APP -----
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 4

---STRONG PASSWORD GENERATOR---

Enter the length needed for password: 12
Should the password include special characters?(y/n): y
Strong Password successfully generated!
Password: o9LRAUtiWkQ
1. Regenerate Password
2. Exit
Enter your choice: 
```

The user is given the option to regenerate the password if they are not satisfied by it.

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
python + - - - - -

---STRONG PASSWORD GENERATOR---

Enter the length needed for password: 12
Should the password include special characters?(y/n): y
Strong Password successfully generated!
Password: o9LRauTiwHwQ
1. Regenerate Password
2. Exit
Enter your choice: 1

Strong Password successfully generated!
Password: 2\9bZk1TrdIm
1. Regenerate Password
2. Exit
Enter your choice: 1

Strong Password successfully generated!
Password: 6Jsdieky9@wB
1. Regenerate Password
2. Exit
Enter your choice: 2
Exiting Password Generator...

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: █
```

Lastly, the user can choose to view the instructions for usage of the password manager application in case they encounter any issues or doubts regarding the same.

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
python + - - - - -

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 5

---INSTRUCTIONS FOR USING PASSWORD MANAGER APP---

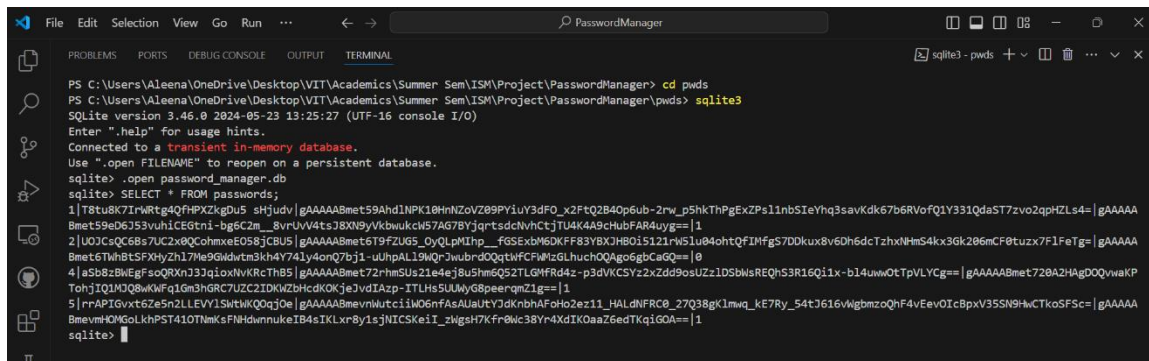
1. For initial setup of your Password Manager-
  a. Create a strong Master Password.
     - Password be at least 12 characters long.
     - Password must contain upper & lower-case alphabets, numerical digits and special characters.
  b. Have one primary and one alternative phone number.
     - Make sure that your primary phone number is always accessible to you for OTP verifications during Login.
  c. A recovery code will be generated.
     - You must ensure that this code is stored somewhere securely in case of forgetting your master password.
2. Login to the Password Manager.
     - Enter your master password and OTP sent to your primary phone number.
     - For each credential (pair of username/email and password), that you want to store in your Password Manager, assign a name to its service.
     - When requesting to view a credential the service name must be provided.
     - If there are many services with similar names, then the Search feature can be used. Enter the part of the name you recall and all services with names matching your query will be displayed to you along with their respective usernames. Once you have found the service-username pair you actually wanted, use that service name to retrieve the password.
3. In case you have forgotten your master password, you can Recover your Password Manager using the recovery code and an OTP will be sent to your primary phone number for verification.
4. If your primary phone number is temporarily inaccessible you may use the alternative Login method to access your Password Manager using your master password and an OTP sent to your alternative phone number.
     - You will only have limited access to your Password Manager.
     - You can only view the credentials that are highly critical for you to access.
     - Search will be disabled, you are required to remember the names you had given to the few services whose credentials are highly critical to you.
     - For the security of your credentials it is advisable to not set all of them as highly critical.
```

```
3. In case you have forgotten your master password, you can Recover your Password Manager using the recovery code and an OTP will be sent to your primary phone number for verification.
4. If your primary phone number is temporarily inaccessible you may use the alternative Login method to access your Password Manager using your master password and an OTP sent to your alternative phone number.
   - You will only have limited access to your Password Manager.
   - You can only view the credentials that are highly critical for you to access.
   - Search will be disabled, you are required to remember the names you had given to the few services whose credentials are highly critical to you.
   - For the security of your credentials it is advisable to not set all of them as highly critical.

----- PASSWORD MANAGER APP -----

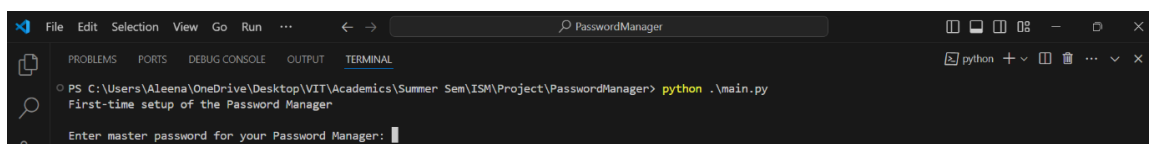
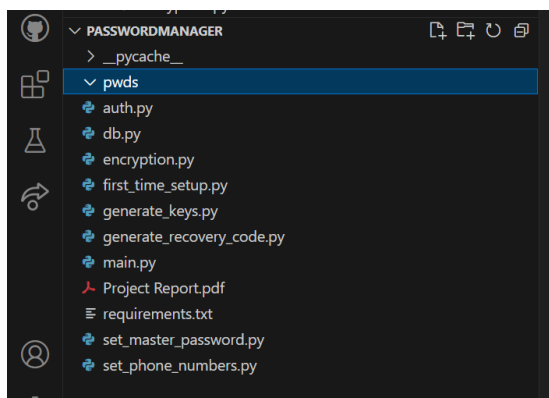
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 6
Exiting Password Manager...
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project\PasswordManager>
```

Directly accessing the database, would not reveal any of the credentials or even the service names for the credentials, since all data are encrypted and obscured. This makes sure that no unauthorized entity can view the stored credentials, as they can only be viewed by opening the application after having successfully logged in.



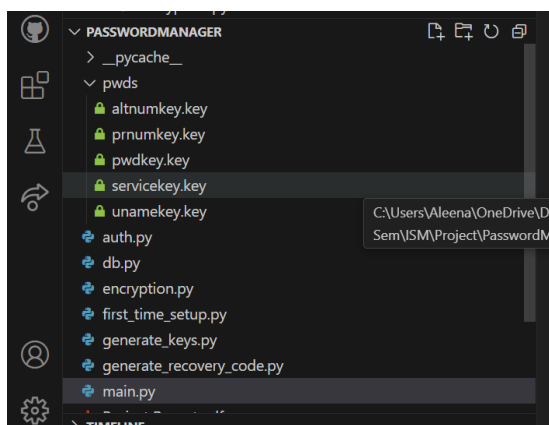
```
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project\PasswordManager> cd pwds
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project\PasswordManager\pwds> sqlite3
SQLite version 3.46.0 2024-05-23 13:25:27 (UTF-16 console I/O)
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open password_manager.db
sqlite> SELECT * FROM passwords;
1|78tU8U71wRtg4QfH9PZk9DUS_ahJudy|gAAAAABmet59Ahd1NPK10HhNZoVZ89PviUy3dFO_x2FtQ2B40pGub-2my_p5hkTHPgExZPs11nb5IeYhq3savKdk67b6RVoFQ1Y331QdaST7zvo2qpHLs4=|gAAAAABmet59Ahd1NPK10HhNZoVZ89PviUy3dFO_x2FtQ2B40pGub-2my_p5hkTHPgExZPs11nb5IeYhq3savKdk67b6RVoFQ1Y331QdaST7zvo2qpHLs4=|1
2|U03CqC68s7UC2x0QcohmmeE0S8jCBUS|gAAAAABmet59FZUG5_OyQlpMhp__fGSExbW6DKFF83Y8JH80iS121rW51u0qhtQfIHfS7DDkux8v6Dh6dTzhxMm54kx3Gk286nCF0tuzx7F1FeTg=|gAAAAABmet59FZUG5_OyQlpMhp__fGSExbW6DKFF83Y8JH80iS121rW51u0qhtQfIHfS7DDkux8v6Dh6dTzhxMm54kx3Gk286nCF0tuzx7F1FeTg=|0
4|aSB8zBwEgfc0QRXn3J3jioxNvKrcTh85|gAAAAABmet72rhmsUu21a4ej8u5hm6Q52TLGwFRd4z-p3dVKCSY22xZdd9osUZz1DSbwsREQhS3R16QiiX-b14UumOtTpVLYCg==|gAAAAABmet72rhmsUu21a4ej8u5hm6Q52TLGwFRd4z-p3dVKCSY22xZdd9osUZz1DSbwsREQhS3R16QiiX-b14UumOtTpVLYCg==|1
5|rrrAP7GvxtGZesn2LLEvY1ShtHk0QqjOe|gAAAAABmetvnuMtciiMO6nfAsAluAtYJdknbhAFoHo2er11_HALdNFRCE_27Q38gKlmmq_kE7Ry_54tJ616vWgBmzoQhF4vEevOicBpxV35SN9HwCTkoSFSc=|gAAAAABmetvnuMtciiMO6nfAsAluAtYJdknbhAFoHo2er11_HALdNFRCE_27Q38gKlmmq_kE7Ry_54tJ616vWgBmzoQhF4vEevOicBpxV35SN9HwCTkoSFSc=|1
sqlite>
```

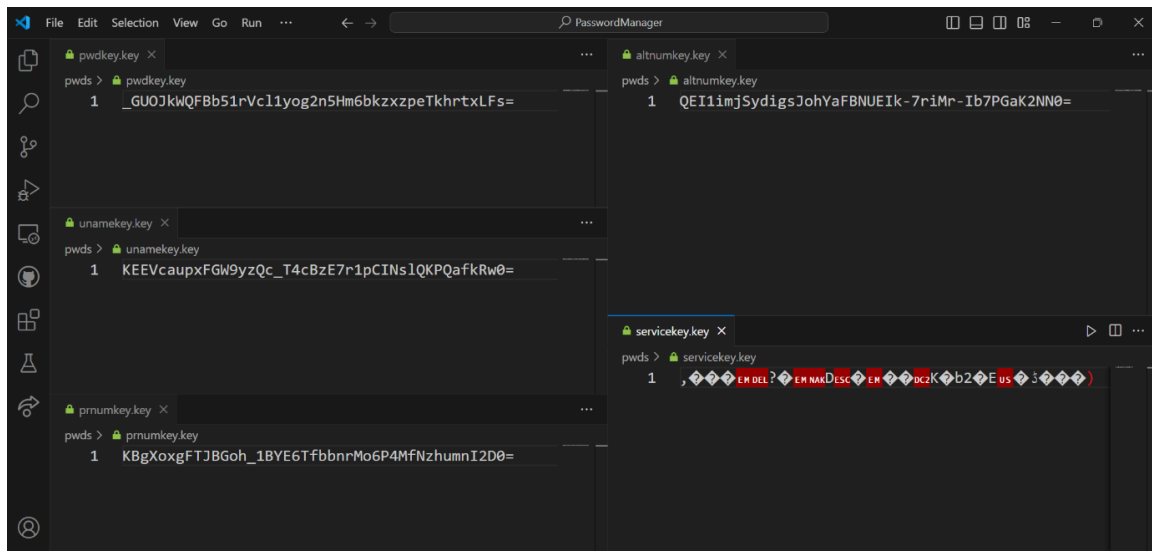
If none of the user-setup files such as the keys, master password, recovery code, phone numbers, etc. are found, then on execution the application will automatically perform the first-time setup.



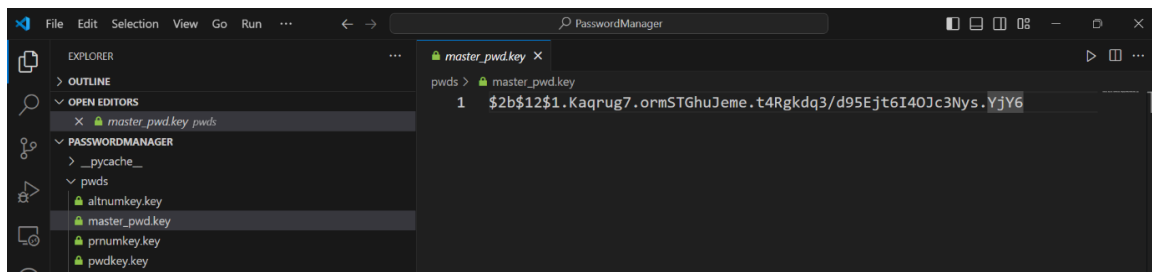
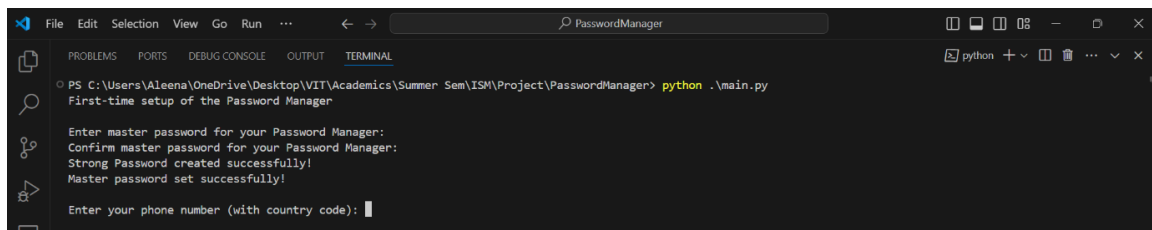
```
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project\PasswordManager> python .\main.py
First-time setup of the Password Manager
Enter master password for your Password Manager: 
```

First the encryption keys are immediately generated and stored.

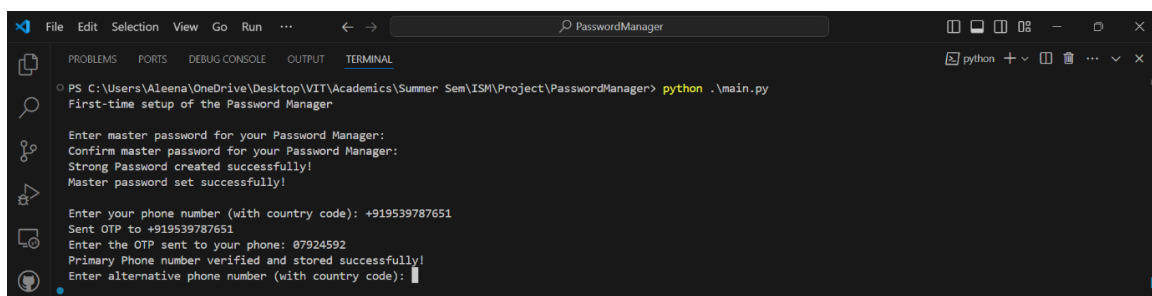


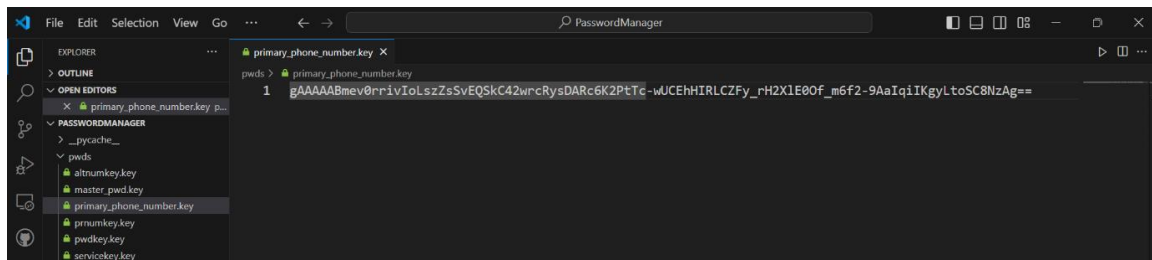


Next the user is prompted to enter a master password (they will have to create this on their own, and also remember it). After entering it, the user will be prompted again to confirm the password by re-entering it. After checking whether the two passwords match, and if they did successfully match, the master password is hashed and stored.

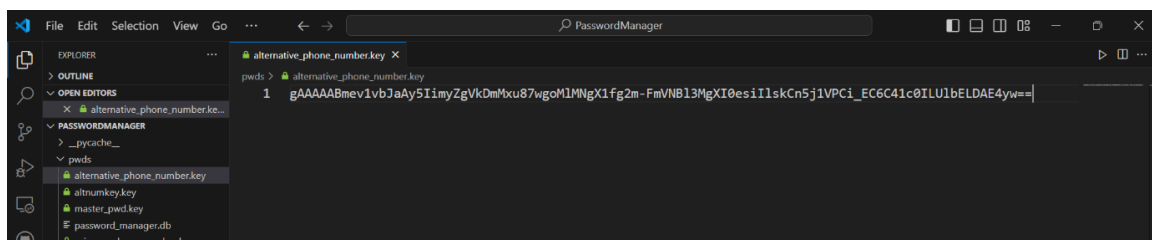
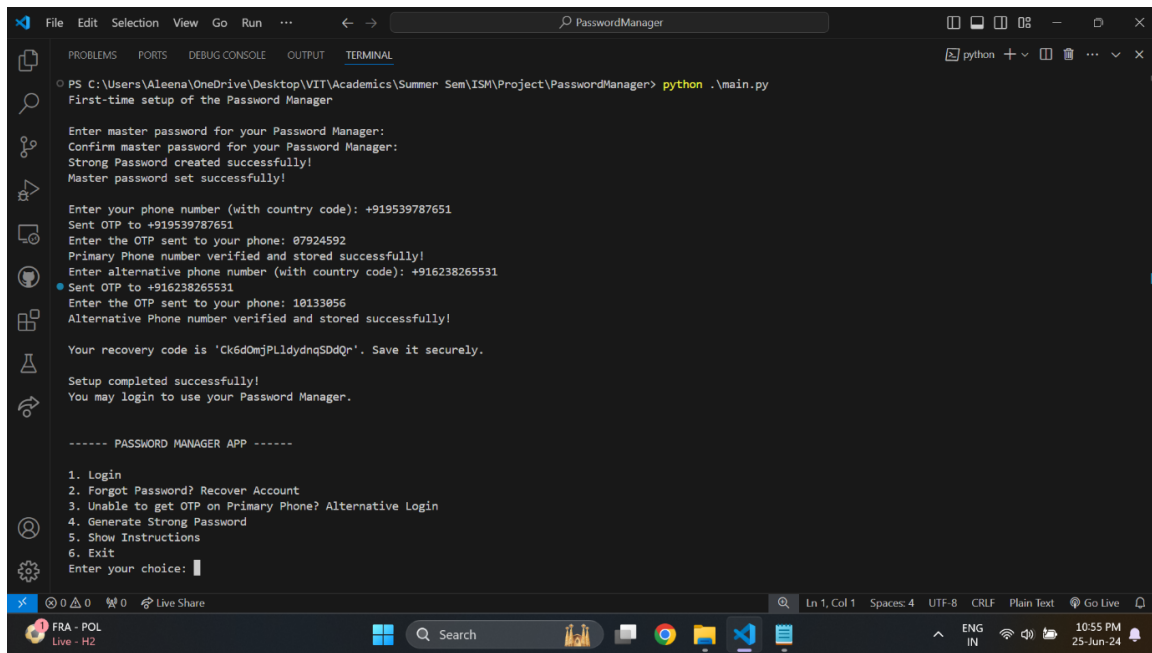


Next the user is asked to enter their phone number, which will be used as their primary phone number for verification purposes. Once the user enters their number, an OTP will be sent to it, which needs to be typed within 1 minute. After the number is verified, it is encrypted and stored.

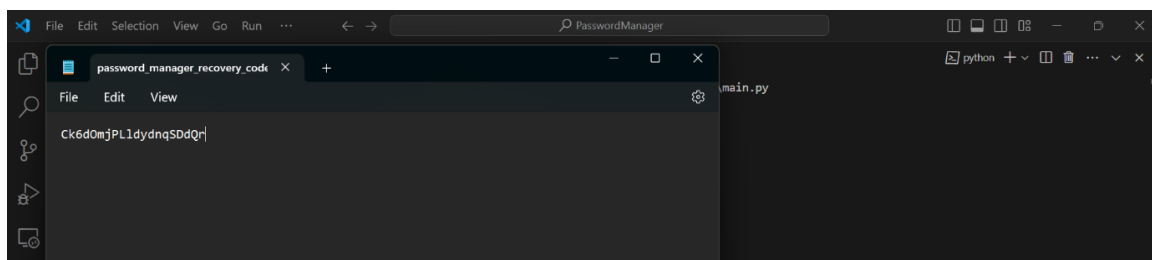




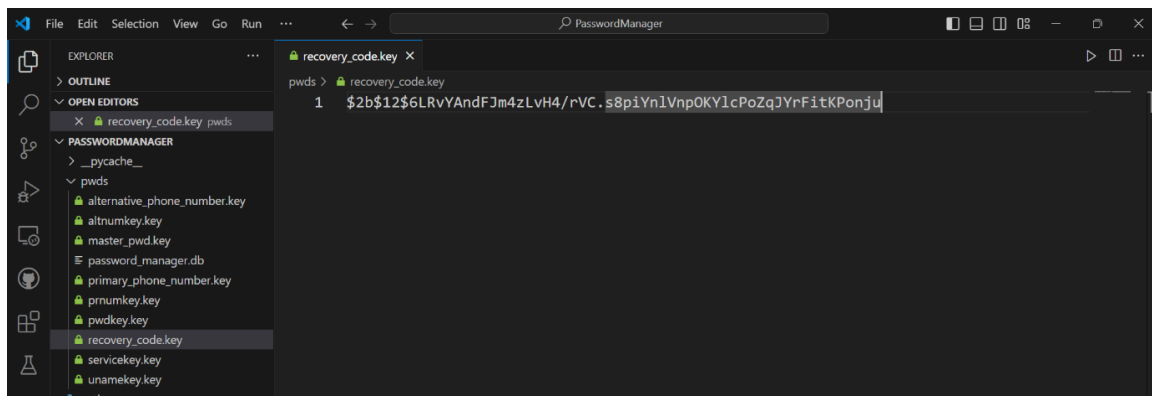
Similarly, the secondary/alternative phone number is also entered, verified, encrypted and stored.



At the end a recovery code is also generated, which the user needs to store somewhere safely for later use, if the need for recovering the account arises.

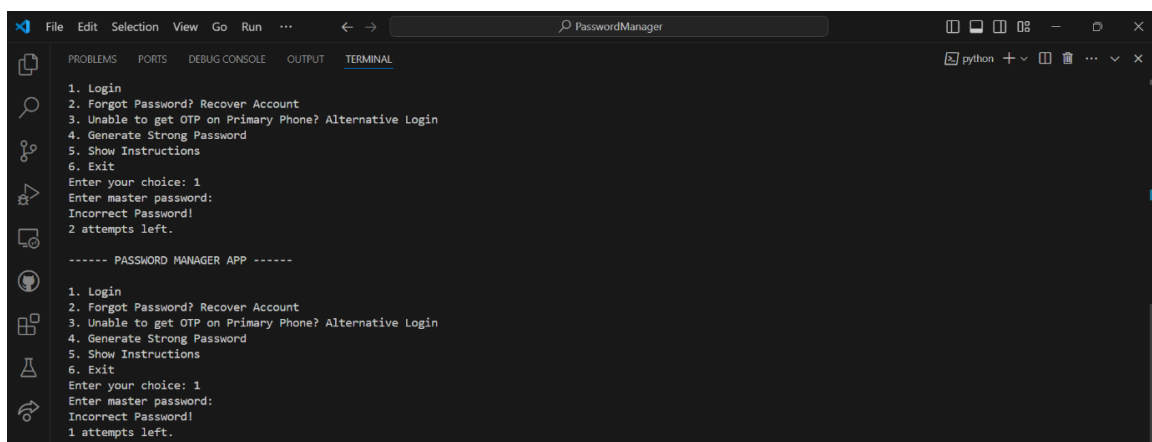


From the application side, the recovery code is also hashed and stored, similar to the master password. Thus, all user-setup files have been created and the user can now login to the Password Manager application.

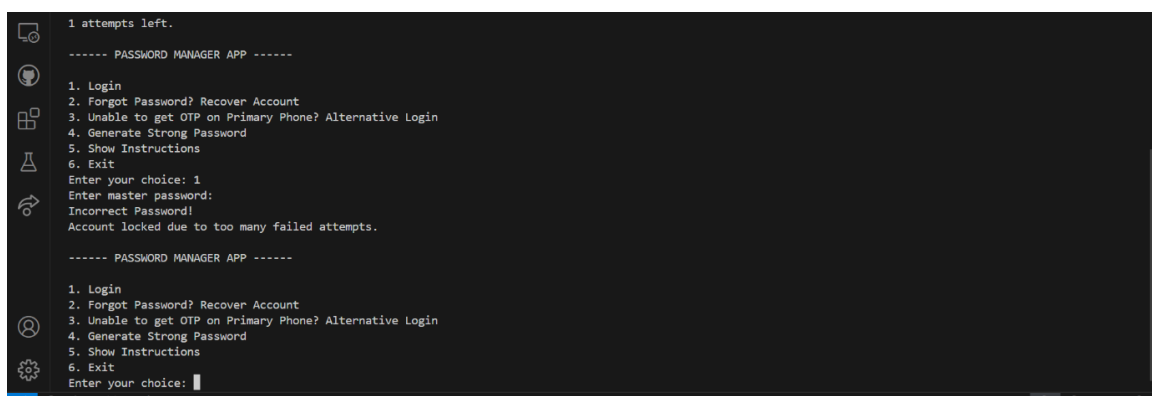


The screenshot shows a code editor with a file explorer on the left. The file explorer lists several files under the 'PASSWORDMANAGER' directory, including 'recovery\_code.key.pwds'. The main editor window displays the content of 'recovery\_code.key', which is a single line of text: '1 \$2b\$12\$6LRvYAndFJm4zLvH4/rVC.s8piYnlVnpOKYlcPoZqJYrFitKJonju'.

On entering incorrect password, recovery code or OTP, the application will keep a track of the number of failed attempts. If the maximum allowed number of attempts is crossed, then the application will be locked for a predefined duration.



The screenshot shows a terminal window with the Password Manager application interface. The terminal displays a menu with options: 1. Login, 2. Forgot Password? Recover Account, 3. Unable to get OTP on Primary Phone? Alternative Login, 4. Generate Strong Password, 5. Show Instructions, 6. Exit. The user enters '1' for Login, then 'Enter master password:'. The terminal shows 'Incorrect Password!' and '2 attempts left.'.



The screenshot shows a terminal window with the Password Manager application interface. The terminal displays a menu with options: 1. Login, 2. Forgot Password? Recover Account, 3. Unable to get OTP on Primary Phone? Alternative Login, 4. Generate Strong Password, 5. Show Instructions, 6. Exit. The user enters '1' for Login, then 'Enter master password:'. The terminal shows 'Incorrect Password!' and 'Account locked due to too many failed attempts.'.

Once the account is locked, login is not possible through any of the methods.



```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
python + - - - - - x

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 1

Account Locked! Try again after 4.0 minutes.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 2

Account Locked! Try again after 4.0 minutes.

----- PASSWORD MANAGER APP -----
```

The only features that will not be locked are strong password generation and instructions display.

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
python + - - - - - x

Account Locked! Try again after 4.0 minutes.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 3

Account Locked! Try again after 4.0 minutes.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 4

---STRONG PASSWORD GENERATOR---

Enter the length needed for password: 8
Should the password include special characters?(y/n): n
Strong Password successfully generated!
Password: WkInuRZn
1. Regenerate Password
2. Exit
Enter your choice: 
```

```
Password: WkInuRZn
1. Regenerate Password
2. Exit
Enter your choice: 2
Exiting Password Generator...

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 
```

```
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 5

---INSTRUCTIONS FOR USING PASSWORD MANAGER APP---

1. For initial setup of your Password Manager-
  a. Create a strong Master Password.
     - Password be at least 12 characters long.
     - Password must contain upper & lower-case alphabets, numerical digits and special characters.
  b. Have one primary and one alternative phone number.
     - Make sure that your primary phone number is always accessible to you for OTP verifications during Login.
  c. A recovery code will be generated.
     - You must ensure that this code is stored somewhere securely in case of forgetting your master password.
2. Login to the Password Manager.
   - Enter your master password and OTP sent to your primary phone number.
   - For each credential (pair of username/email and password), that you want to store in your Password Manager, assign a name to its service.
   - The generation of a credential (the credential name must be provided
```

```
2. Login to the Password Manager.
- Enter your master password and OTP sent to your primary phone number.
- For each credential (pair of username/email and password), that you want to store in your Password Manager, assign a name to its service.
- When requesting to view a credential the service name must be provided.
- If there are many services with similar names, then the Search feature can be used. Enter the part of the name you recall and all services with names matching your query will be displayed to you along with their respective usernames. Once you have found the service-username pair you actually wanted, use that service name to retrieve the password.
3. In case you have forgotten your master password, you can Recover your Password Manager using the recovery code and an OTP will be sent to your primary phone number for verification.
4. If your primary phone number is temporarily inaccessible you may use the alternative Login method to access your Password Manager using your master password and an OTP sent to your alternative phone number.
- You will only have limited access to your Password Manager.
- You can only view the credentials that are highly critical for you to access.
- Search will be disabled, you are required to remember the names you had given to the few services whose credentials are highly critical to you.
- For the security of your credentials it is advisable to not set all of them as highly critical.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 6
Exiting Password Manager...
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project\PasswordManager>
```

On entering the recovery code incorrectly, the application is locked after a different number of failed attempts and for a different duration of time.

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project\PasswordManager> python .\main.py

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 2
Enter recovery code: dmdmdndskkkd
Incorrect Recovery Code!
1 attempts left.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: █
```

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 2
Enter recovery code: sksksksksk
Incorrect Recovery Code!
Account locked due to too many failed attempts.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 1

Account Locked! Try again after 119.0 minutes.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: █
```

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 2

Account Locked! Try again after 119.0 minutes.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 3

Account Locked! Try again after 119.0 minutes.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 6
Exiting Password Manager...
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project>PasswordManager>
```

Similarly, for alternative login method the number of failed attempts before the account gets locked is different, and so is the duration of locking.

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
PS C:\Users\Aleena\OneDrive\Desktop\VIT\Academics\Summer Sem\ISM\Project>PasswordManager> python .\main.py

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 3
Enter master password:
Incorrect Password!
2 attempts left.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: █
```

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 3
Enter master password:
Incorrect Password!
1 attempts left.

----- PASSWORD MANAGER APP -----

1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 3
Enter master password:
Incorrect Password!
Account locked due to too many failed attempts.

----- PASSWORD MANAGER APP -----
```

```
----- PASSWORD MANAGER APP -----
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 3

Account Locked! Try again after 9.0 minutes.

----- PASSWORD MANAGER APP -----
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 
```

```
File Edit Selection View Go Run ... PasswordManager
PROBLEMS PORTS DEBUG CONSOLE OUTPUT TERMINAL
----- PASSWORD MANAGER APP -----
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 1

Account Locked! Try again after 9.0 minutes.

----- PASSWORD MANAGER APP -----
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 2

Account Locked! Try again after 9.0 minutes.

----- PASSWORD MANAGER APP -----
1. Login
2. Forgot Password? Recover Account
3. Unable to get OTP on Primary Phone? Alternative Login
4. Generate Strong Password
5. Show Instructions
6. Exit
Enter your choice: 
```

## Conclusion

This project aimed to develop a secure and user-friendly password manager application that addresses the critical challenges of managing digital credentials securely. By leveraging Python's robust ecosystem and integrating advanced security measures, the application offers a reliable solution for users to store, retrieve, and manage their passwords and user IDs with confidence. The core functionalities of the application include secure password storage using bcrypt hashing, multi-factor authentication (MFA) via SMS-based OTP verification, and robust recovery mechanisms using hashed recovery codes. These features ensure that users can access their credentials securely across devices while mitigating the risk of unauthorized access. The implementation of symmetric encryption with Fernet and format-preserving encryption (FPE) for sensitive data such as phone numbers, service names, usernames and passwords, further enhance data confidentiality and integrity. The application's modular architecture and clear command-line interface (CLI) facilitate ease of use and maintenance, ensuring intuitive navigation and operational clarity for users. Moreover, the project emphasizes user experience by incorporating features like strong password generation and detailed instructions for usage. This approach aims to simplify password management while promoting the adoption of secure practices among users.

Looking forward, future enhancements could explore additional security measures, such as biometric authentication or integration with cloud-based backup solutions, to further enhance usability and resilience against evolving cyber threats.

In conclusion, this password manager application represents a significant step towards enhancing digital security practices, providing users with a robust tool to safeguard their online credentials effectively.

## References

- [1] Simmons, J., Diallo, O., Oesch, S., & Ruoti, S. (2021, December). Systematization of password manager use cases and design paradigms. In *Proceedings of the 37th Annual Computer Security Applications Conference* (pp. 528-540).
- [2] Oesch, S., Ruoti, S., Simmons, J., & Gautam, A. (2022, April). "It Basically Started Using Me:" An Observational Study of Password Manager Usage. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (pp. 1-23).
- [3] Oesch, S., & Ruoti, S. (2020, August). That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers. In *Proceedings of the 29th USENIX Conference on Security Symposium* (pp. 2165-2182).
- [4] Pearman, S., Zhang, S. A., Bauer, L., Christin, N., & Cranor, L. F. (2019). Why people (don't) use password managers effectively. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)* (pp. 319-338).
- [5] Ray, H., Wolf, F., Kuber, R., & Aviv, A. J. (2021). Why older adults (Don't) use password managers. In *30th USENIX Security Symposium (USENIX Security 21)* (pp. 73-90).
- [6] Fernando, W. P. K., Dissanayake, D. A. N. P., Dushmantha, S. G. V. D., Liyanage, D. L. C. P., & Karunatilake, C. (2023). Challenges and Opportunities in Password Management: A Review of Current Solutions.
- [7] Petrov, M. (2022). Android password managers and vault applications: data storage security issues identification. *Journal of Information Security and Applications*, 67, 103152.
- [8] Pandare, P., Uniyal, S., Vani, R., Mali, S., & Rumao, P. (2023, April). Enhanced Password Manager using Hybrid Approach. In *2023 International Conference on Inventive Computation Technologies (ICICT)* (pp. 1793-1798). IEEE.
- [9] Dhanalakshmi, R., Vijayaraghavan, N., Narasimhan, S., & Basha, S. (2023, April). Password Manager with Multi-Factor Authentication. In *2023 International Conference on Networking and Communications (ICNWC)* (pp. 1-5). IEEE.
- [10] Jeong, H., & Jung, H. (2021, April). MonoPass: a password manager without master password authentication. In *26th international conference on intelligent user interfaces-companion* (pp. 52-54).