Python Passwort Manager

PROJEKTARBEIT

des Studienganges Informatik

an der Dualen Hochschule Baden-Württemberg Ravensburg – Campus Friedrichshafen

von

Tobias Stoppelkamp, Christopher Groshert, David Prinz und Lion Wicki

Abgabedatum:16.08.2024

Kurs TIM/TIS

Inhaltsverzeichnis

Grundlegender Aufbau

Der Passwort Manager ist modular aufgebaut, wobei jede Datei eine spezifische Funktion erfüllt:

- **interface.py:** Benutzeroberfläche und Benutzerinteraktion.
- **passwordManager.py:** Backend-Logik für das Speichern und Verwalten von Passwörtern.
- **main.py:** Startpunkt des Programms, der die Module zusammenführt.

Die Sicherheit wird durch die Verwendung von Verschlüsselungstechniken gewährleistet, wobei das Master-Passwort zur Generierung eines Verschlüsselungsschlüssels verwendet wird. Die Daten werden verschlüsselt gespeichert und bei Bedarf entschlüsselt, um maximale Sicherheit zu gewährleisten.

Die Anwendung bietet umfassende Funktionen zur Passwortverwaltung, einschließlich der Überprüfung von Passwortstärke, Erkennung wiederverwendeter Passwörter und Überprüfung kompromittierter Passwörter. Die textbasierte Curses-Oberfläche ermöglicht eine einfache Navigation und Bedienung.

main.py

Das main.py-Modul dient als Einstiegspunkt für das Programm und startet die Passwort-Manager-Anwendung. Es initialisiert die Curses-Bibliothek und die Hauptschnittstelle.

Hauptfunktionen

- **getHiddenPassword(stdscr: Any, prompt: str) -> str:** Erfasst ein verstecktes Passwort vom Benutzer.
- **main(stdscr: Any) -> None:** Die Hauptfunktion, die den Passwort Manager startet. Sie fordert den Benutzer auf, das Master-Passwort einzugeben und validiert dieses, bevor die Benutzeroberfläche gestartet wird.

passwordManager.py

Das passwordManager.py-Modul implementiert die Logik für das sichere Speichern und Verwalten von Passwörtern.

Hauptklassen und Methoden:

PasswordManager
Diese Klasse verwaltet die Passwörter und stellt Methoden zur sicheren Speicherung und Abruf bereit.

Methoden:

- **__init__(self, masterPassword: str) -> None:** Initialisiert den Manager mit einem Master-Passwort und generiert einen Verschlüsselungsschlüssel.
- **generateKey(self, password: str) -> bytes:** Generiert einen Verschlüsselungsschlüssel auf Basis des Master-Passworts.
- **loadData(self) -> None**: Lädt und entschlüsselt gespeicherte Passwortdaten.
- **saveData(self) -> None**: Verschlüsselt und speichert die Passwortdaten.
- **addPassword(self, site: str, username: Optional[str] = None, password: Optional[str] = None, notes: Optional[str] = None, category: Optional[str] = None) -> None:** Fügt ein neues Passwort hinzu.
- **getPassword(self, site: str) -> Optional[Dict[str, Any]]:** Ruft ein Passwort anhand des Seitennamens ab.
- **deletePassword(self, site: str)** -> None: Löscht ein Passwort.
- **updatePassword(self, site: str, username: Optional[str] = None, password: Optional[str] = None, notes: Optional[str] = None, category: Optional[str] = None) -> None:** Aktualisiert ein bestehendes Passwort.
- **searchPassword(self, keyword: str) -> Dict[str, Dict[str, Any]]:** Sucht nach Passwörtern anhand eines Schlüsselworts.
- **checkPasswordStrength(self, password: str) -> Tuple[bool, List[str]]:** Überprüft die Stärke eines Passworts.
- **generateStrongPassword(self, length: int = 12) -> str:** Generiert ein sicheres, zufälliges Passwort.
- **checkReusedPassword(self, password: str) -> bool:** Überprüft, ob ein Passwort in der Datenbank bereits verwendet wurde.
- **checkPwnedPassword(self, password: str) -> bool:** Überprüft über eine API, ob ein Passwort kompromittiert wurde.

interface.py

Das interface.py-Modul enthält die Benutzeroberfläche des Passwort Managers, die mit der Curses-Bibliothek erstellt wurde. Diese Schnittstelle ermöglicht eine textbasierte Interaktion mit dem Benutzer.

Hauptklassen und Methoden

Curses Interface:
Diese Klasse stellt die Hauptschnittstelle für die Passwortverwaltung bereit.

Methoden:

- **__init__(self, pm: Any) -> None:** Initialisiert die Schnittstelle und speichert eine Instanz des PasswordManager.
- **drawMenu(self, stdscr: Any) -> None:** Zeichnet das Hauptmenü, in dem der Benutzer navigieren kann.
- **getInput(self, stdscr: Any, prompt: str) -> str:** Erfasst Benutzereingaben als Text.
- **getPasswordInput(self, stdscr: Any, prompt: str, checkStrength: bool = False) -> Optional[str]:** Erfasst ein Passwort von Benutzern, wobei die Eingabe als * maskiert wird.
- **run(self, stdscr: Any) -> None:** Startet die Benutzeroberfläche und behandelt die Navigation im Menü.
- **addPassword(self, stdscr: Any) -> None**: Erfasst Daten und fügt ein neues Passwort hinzu.
- **getPassword(self, stdscr: Any) -> None:** Sucht und zeigt gespeicherte Passwörter an.
- **deletePassword(self, stdscr: Any) -> None:** Löscht ein gespeichertes Passwort.
- **updatePassword(self, stdscr: Any) -> None**: Aktualisiert ein vorhandenes Passwort.
- **searchPassword(self, stdscr: Any) -> None**: Sucht nach Passwörtern anhand eines Suchbegriffs.
- **checkPasswordStrength(self, stdscr: Any) -> None:** Überprüft die Stärke eines eingegebenen Passworts.
- **checkReusedPassword(self, stdscr: Any) -> None:** Überprüft, ob ein Passwort wiederverwendet wurde.
- **checkPwnedPassword(self, stdscr: Any) -> None:** Überprüft, ob ein Passwort in Datenlecks aufgetaucht ist.

Verwendete Bibliotheken und Versionen

Entwicklungs- und Testing-Bibliotheken:

- **pylint**: 3.2.3
  - Verwendung: Code-Analyse und Linting für Python-Code.
- **coverage**: 7.5.3
  - Verwendung: Code-Coverage-Analyse, um zu überprüfen, wie viel des Codes durch Tests abgedeckt wird.
- **mypy**: 1.10.0
  - Verwendung: Statische Typüberprüfung für Python.

Produktivitätsbibliotheken:

- **cryptography**: *keine spezifische Version angegeben*
  - Verwendung: Implementierung der Verschlüsselungstechniken für die sichere Speicherung von Passwörtern. Diese Bibliothek bietet starke Verschlüsselungs- und Entschlüsselungsfunktionen.
- **requests**: *keine spezifische Version angegeben*
  - Verwendung: HTTP-Anfragen, insbesondere zur Überprüfung von Passwörtern gegen die "Pwned Passwords"-API.
- **types-requests**: *keine spezifische Version angegeben*
  - Verwendung: Typinformationen für die requests-Bibliothek, um die Typüberprüfung zu unterstützen.

Bibliotheken aus dem Code:

- **curses**: *Teil der Python Standardbibliothek*
  - Verwendung: Erstellung einer textbasierten Benutzeroberfläche (TUI) für die Passwortverwaltung.
- **json**: *Teil der Python Standardbibliothek*
  - Verwendung: Serialisierung und Deserialisierung von Daten in JSON-Format zur Speicherung und zum Laden der Passwortinformationen.
- **random**: *Teil der Python Standardbibliothek*
  - Verwendung: Generierung von zufälligen Werten, insbesondere bei der Erstellung starker Passwörter.
- **hashlib**: *Teil der Python Standardbibliothek*
  - Verwendung: Erzeugung von Hashes für Passwörter, insbesondere für die Integration mit der "Pwned Passwords"-API.
- **base64**: *Teil der Python Standardbibliothek*
  - Verwendung: Kodierung von Bytes zur sicheren Speicherung und Übertragung von Daten, insbesondere bei der Verschlüsselung von Passwörtern.
- **string**: *Teil der Python Standardbibliothek*
  - Verwendung: Bereitstellung von Zeichenkettenoperationen, insbesondere bei der Passwortgenerierung.
- **datetime**: *Teil der Python Standardbibliothek*
  - Verwendung: Umgang mit Datums- und Zeitangaben, insbesondere zur Protokollierung von Zeitstempeln bei der Passworterstellung.

Architekturbeschreibung des Passwort Managers

Der Passwort Manager ist in Python implementiert und besteht aus einer modularen Architektur, die verschiedene Verantwortlichkeiten auf mehrere Module verteilt. Diese Architektur ermöglicht es, die einzelnen Komponenten des Systems unabhängig zu entwickeln, zu testen und zu warten. Im Folgenden wird die Architektur des Passwort Managers detailliert beschrieben.

1. Modularer Aufbau

- Der Passwort Manager ist in drei Hauptmodule unterteilt:
- **interface.py**: Verwaltung der Benutzerschnittstelle.
- **passwordManager.py**: Backend-Logik für die sichere Verwaltung und Speicherung von Passwörtern.
- **main.py**: Einstiegspunkt des Programms, der die Benutzeroberfläche und die Logik zusammenführt.

1.1 interface.py - Benutzerschnittstelle

- Dieses Modul implementiert die textbasierte Benutzeroberfläche (TUI) des Passwort Managers unter Verwendung der curses-Bibliothek. Es ist dafür verantwortlich, die Interaktionen des Benutzers zu erfassen, Menüs und Eingabeaufforderungen anzuzeigen sowie die entsprechenden Aktionen auszuführen.

Wichtige Komponenten:

- **Klasse CursesInterface**:
  - Diese Klasse kapselt alle Funktionen zur Benutzerinteraktion.
  - Sie enthält Methoden zum Zeichnen von Menüs, zum Erfassen von Eingaben und zur Handhabung der Navigation durch das System.
  - Die run()-Methode ist der zentrale Punkt, der den Fluss des Programms steuert, indem sie auf Benutzereingaben wartet und die entsprechenden Aktionen basierend auf der Auswahl des Benutzers ausführt.

1.2 passwordManager.py - Backend-Logik

- Dieses Modul implementiert die Kernlogik für das Speichern, Abrufen, Aktualisieren und Löschen von Passwörtern. Es verwendet Verschlüsselungstechniken, um sicherzustellen, dass die gespeicherten Passwörter sicher sind.

Wichtige Komponenten:

- **Klasse PasswordManager**:
  - Diese Klasse ist für die Verwaltung der Passwörter verantwortlich.
  - Sie kapselt alle Operationen, die zur sicheren Speicherung und Verarbeitung von Passwörtern erforderlich sind, wie z.B. das Hinzufügen neuer Passwörter, das Überprüfen der Passwortstärke und das Überprüfen von Passwörtern gegen die "Pwned Passwords"-API.
  - Die Passwortdaten werden verschlüsselt auf der Festplatte gespeichert, um ihre Sicherheit zu gewährleisten. Die Verschlüsselung erfolgt unter Verwendung eines Schlüssels, der aus dem Master-Passwort des Benutzers abgeleitet wird.

1.3 main.py - Programminitialisierung

- Dieses Modul ist der Einstiegspunkt des Programms. Es initialisiert die Benutzeroberfläche und das Backend und steuert den Ablauf des Programms.

Wichtige Komponenten:

- **Funktion main()**:
  - Diese Funktion wird aufgerufen, wenn das Programm startet. Sie fordert den Benutzer auf, ein Master-Passwort einzugeben, und verwendet dieses, um die Passwortdaten zu entschlüsseln und zu laden.
  - Nach erfolgreicher Authentifizierung wird die Benutzeroberfläche gestartet, und der Benutzer kann den Passwort Manager bedienen.

2. Datenfluss und Interaktion

- Der Datenfluss zwischen den Modulen erfolgt in einer sequentiellen Weise:

1. **Benutzereingabe**: Der Benutzer interagiert mit dem Programm über die Curses-Oberfläche, die im interface.py-Modul implementiert ist. Der Benutzer wählt eine Aktion aus, z.B. das Hinzufügen oder Abrufen eines Passworts.
2. **Datenverarbeitung**: Die CursesInterface-Klasse leitet die Benutzeranfragen an die entsprechenden Methoden der PasswordManager-Klasse weiter. Diese Klasse ist für die eigentliche Verarbeitung und Manipulation der Daten verantwortlich.
3. **Speicherung und Abruf**: Die PasswordManager-Klasse speichert und ruft Daten aus der verschlüsselten Datei ab. Wenn der Benutzer eine neue Passwort-Eingabe vornimmt, wird diese verschlüsselt und in der Datei passwords.json gespeichert. Beim Abruf eines Passworts wird es entschlüsselt und zur Anzeige an die Benutzeroberfläche übergeben.
4. **Feedback an den Benutzer**: Die Ergebnisse der Operationen werden über die Benutzeroberfläche (interface.py) zurück an den Benutzer kommuniziert. Erfolgreiche Aktionen oder Fehlermeldungen werden entsprechend dargestellt.

3. Sicherheitsarchitektur

- Ein wesentlicher Aspekt dieser Architektur ist die Sicherheit, insbesondere bei der Speicherung und Verarbeitung von Passwörtern. Die Sicherheitsarchitektur besteht aus folgenden Hauptkomponenten:

- **Master-Passwort**: Das Master-Passwort wird verwendet, um einen Verschlüsselungsschlüssel zu generieren. Dieser Schlüssel wird dann zur Verschlüsselung und Entschlüsselung der Passwortdaten verwendet.
- **Verschlüsselung**: Alle Passwörter und zugehörigen Informationen werden verschlüsselt gespeichert. Dies schützt die Daten sowohl im Ruhezustand als auch bei der Übertragung vor unbefugtem Zugriff.
- **Starke Passwörter**: Der Passwort Manager unterstützt die Generierung starker Passwörter und bietet Funktionen zur Überprüfung der Passwortstärke, um sicherzustellen, dass gespeicherte Passwörter den gängigen Sicherheitsstandards entsprechen.

- **Externe Sicherheitsprüfung**: Der Passwort Manager integriert die "Pwned Passwords"-API, um zu überprüfen, ob ein Passwort kompromittiert wurde. Diese API prüft, ob ein Passwort in bekannten Datenlecks aufgetaucht ist.

Beschreibung des Speicherformats

Der Passwort Manager speichert die Passwörter und zugehörigen Informationen in einer verschlüsselten Datei, die im JSON-Format strukturiert ist. Diese Datei trägt in der Regel den Namen passwords.json. Um die gespeicherten Daten zu schützen, wird der Inhalt der Datei vor der Speicherung vollständig verschlüsselt.

Verschlüsselungsprozess

Bevor die JSON-Daten in der Datei gespeichert werden, durchlaufen sie einen Verschlüsselungsprozess, um die Vertraulichkeit der Informationen sicherzustellen.

Schlüsselgenerierung: Ein Verschlüsselungsschlüssel wird aus dem Master-Passwort des Benutzers generiert. Dieser Schlüssel wird durch die SHA-256-Hashfunktion erzeugt und dann mit base64 codiert, um ihn für die Verschlüsselung und Entschlüsselung nutzbar zu machen.

Verschlüsselung: Die JSON-Daten werden in eine Zeichenkette (UTF-8 codiert) konvertiert und dann mit der Fernet-Verschlüsselung aus der cryptography-Bibliothek verschlüsselt. Das Ergebnis ist eine verschlüsselte Byte-Sequenz, die in der Datei passwords.json gespeichert wird.

Beispiel einer verschlüsselten Datei: Die verschlüsselte Datei sieht aus wie ein zufälliger Zeichensalat und kann ohne den richtigen Schlüssel nicht gelesen werden. Ein Beispiel: gAAAAABe7H8tBphzKRaLQBCznf5p_zf8ERZ10SJPboNqA...

- Dieser Datenstrom stellt die verschlüsselte Version der ursprünglichen JSON-Daten dar.

Speicherort und Dateistruktur

Die verschlüsselte Datei wird typischerweise im gleichen Verzeichnis gespeichert, in dem das Programm ausgeführt wird. Der Dateiname ist standardmäßig passwords.json, kann jedoch je nach Implementierung variieren.

Datenintegrität und Sicherheit

Datenintegrität: Beim Laden der Daten aus der Datei überprüft der Passwort Manager, ob das Master-Passwort korrekt ist. Eine falsche Eingabe führt zu einer InvalidToken-Ausnahme, was anzeigt, dass die Daten nicht korrekt entschlüsselt werden konnten.

Sicherheit: Da die gesamte Datei verschlüsselt ist, sind die gespeicherten Daten selbst bei unbefugtem Zugriff auf die Datei sicher. Ohne das richtige Master-Passwort können die Daten nicht entschlüsselt oder manipuliert werden.

Beschreibung des Nutzerinterfaces

Das Nutzerinterface des Passwort Managers ist eine textbasierte Benutzeroberfläche (TUI), die mithilfe der Python-Bibliothek curses erstellt wurde. Diese Oberfläche ermöglicht es Benutzern, auf einfache und intuitive Weise mit dem Passwort Manager zu interagieren. Das Interface ist in einer Menüstruktur organisiert, die es dem Benutzer ermöglicht, verschiedene Aktionen durchzuführen, wie das Hinzufügen, Abrufen, Aktualisieren und Löschen von Passwörtern.

1. Startbildschirm und Navigation

- **Startbildschirm**: Beim Start des Programms wird der Benutzer zunächst aufgefordert, das Master-Passwort einzugeben. Dieser Bildschirm zeigt eine einfache Eingabeaufforderung, bei der das Passwort verdeckt (mit *-Zeichen) eingegeben wird.
- **Hauptmenü**: Nach der erfolgreichen Eingabe des Master-Passworts wird der Benutzer zum Hauptmenü weitergeleitet. Das Hauptmenü listet die verfügbaren Optionen auf, die der Benutzer auswählen kann. Die Navigation erfolgt über die Pfeiltasten (nach oben/nach unten), und die Auswahl wird mit der Eingabetaste bestätigt.

Hauptmenü-Optionen:

1. **Add Password**: Fügt ein neues Passwort hinzu.
2. **Get Password**: Ruft ein gespeichertes Passwort ab.
3. **Delete Password**: Löscht ein gespeichertes Passwort.
4. **Update Password**: Aktualisiert ein gespeichertes Passwort.
5. **Search Password**: Sucht nach Passwörtern anhand eines Suchbegriffs.
6. **Check Password Strength**: Überprüft die Stärke eines Passworts.
7. **Check Reused Password**: Überprüft, ob ein Passwort in der Datenbank wiederverwendet wurde.
8. **Check Pwned Password**: Überprüft, ob ein Passwort in bekannten Datenlecks kompromittiert wurde.
9. **Exit**: Beendet die Anwendung.

- **Hervorhebung der Auswahl**: Die aktuelle Auswahl im Menü wird durch eine Hervorhebung (inverser Text) angezeigt, die es dem Benutzer erleichtert, zu sehen, welche Option ausgewählt ist.

2. Eingabemasken

- **Text- und Passwort-Eingaben**: Bei der Auswahl einer Option, die Benutzereingaben erfordert (z.B. bei der Eingabe eines neuen Passworts oder beim Abrufen eines vorhandenen Passworts), wird eine Eingabemaske angezeigt. Der Benutzer gibt die erforderlichen Informationen direkt in das Interface ein.
  - **Normale Eingaben** (z.B. Website, Benutzername, Notizen): Diese Eingaben werden als Klartext angezeigt.
  - **Passworteingaben**: Passwörter werden verdeckt eingegeben, wobei anstelle des eingegebenen Textes *-Zeichen angezeigt werden.
- **Erweiterte Eingabefelder**: Einige Eingabemasken bieten zusätzliche Optionen, wie z.B. die Möglichkeit, die Stärke eines eingegebenen Passworts zu überprüfen oder ein starkes Passwort automatisch zu generieren.

3. Rückmeldungen und Benachrichtigungen

- **Erfolgsnachrichten**: Nach einer erfolgreichen Aktion, wie dem Hinzufügen oder Löschen eines Passworts, zeigt das Interface eine Bestätigungsmeldung an, z.B. "Password added successfully!" oder "Password deleted successfully!".
- **Fehlermeldungen**: Wenn eine Operation fehlschlägt (z.B. bei einer falschen Eingabe oder wenn das Master-Passwort nicht stimmt), wird eine entsprechende Fehlermeldung angezeigt. Bei der Überprüfung des Master-Passworts gibt das Interface z.B. "Incorrect master password. Please try again." aus.
- **Wartezustände**: Bei Aktionen, die eine Bestätigung erfordern (z.B. das Generieren eines neuen Passworts oder die Überprüfung der Passwortstärke), wartet das Interface, bis der Benutzer eine Taste drückt, bevor es zum Hauptmenü zurückkehrt.

. Beispielhafter Ablauf

- **Passwort hinzufügen**:
  1. Der Benutzer wählt "Add Password" aus dem Hauptmenü.
  2. Eine Eingabemaske erscheint, in der der Benutzer die Website, den Benutzernamen und das Passwort eingibt. Das Passwort kann auf Wunsch auf Stärke geprüft werden.
  3. Optional kann der Benutzer Notizen und eine Kategorie hinzufügen.
  4. Nach der Eingabe wird das Passwort gespeichert und eine Bestätigungsmeldung angezeigt.
- **Passwort abrufen**:
  1. Der Benutzer wählt "Get Password" aus dem Hauptmenü.
  2. Der Benutzer wird aufgefordert, den Namen der Website einzugeben.
  3. Wenn das Passwort gefunden wird, werden die Details (Benutzername, Passwort, Notizen, Kategorie) angezeigt.

5. Beenden der Anwendung

- Der Benutzer kann das Programm jederzeit beenden, indem er die Option "Exit" aus dem Hauptmenü auswählt oder die ESC-Taste drückt. Eine Bestätigung ist hierfür nicht erforderlich, und das Programm wird direkt geschlossen.

Programmablauf

1. Programmstart

- **Einstiegspunkt (main.py)**:
  - Das Programm startet in der Datei main.py, wo die curses.wrapper(main)-Funktion aufgerufen wird. Diese Funktion initialisiert die Curses-Umgebung und ruft die main()-Funktion auf.

2. Master-Passwort-Abfrage

- **Eingabe des Master-Passworts**:
  - In der main()-Funktion wird der Benutzer aufgefordert, sein Master-Passwort einzugeben. Diese Eingabe erfolgt verdeckt, d.h., das Passwort wird als * dargestellt.
- **Prüfung des Master-Passworts**:
  - Nach der Eingabe des Master-Passworts versucht das Programm, die gespeicherten Daten mit dem abgeleiteten Schlüssel zu entschlüsseln (mittels der Methode loadData() in der Klasse PasswordManager).
  - **Erfolgreiche Entschlüsselung**: Wenn das Passwort korrekt ist, wird die Benutzeroberfläche gestartet.
  - **Fehlgeschlagene Entschlüsselung**: Bei falscher Eingabe wird eine Fehlermeldung angezeigt, und der Benutzer kann es erneut versuchen (bis zu drei Versuche). Nach drei Fehlversuchen beendet sich das Programm.

3. Hauptmenü und Navigation

- **Anzeige des Hauptmenüs**:
  - Nach erfolgreicher Authentifizierung wird das Hauptmenü angezeigt. Der Benutzer navigiert durch die Menüoptionen mit den Pfeiltasten.
- **Menüoptionen**:
  - Der Benutzer kann eine der folgenden Aktionen auswählen:
    1. **Add Password**: Hinzufügen eines neuen Passworts.
    2. **Get Password**: Abrufen eines gespeicherten Passworts.
    3. **Delete Password**: Löschen eines gespeicherten Passworts.
    4. **Update Password**: Aktualisieren eines gespeicherten Passworts.
    5. **Search Password**: Suchen nach Passwörtern anhand eines Suchbegriffs.
    6. **Check Password Strength**: Überprüfen der Stärke eines Passworts.
    7. **Check Reused Password**: Überprüfen, ob ein Passwort wiederverwendet wurde.
    8. **Check Pwned Password**: Überprüfen, ob ein Passwort kompromittiert wurde.
    9. **Exit**: Beenden des Programms.

4. Verarbeitung der Benutzeraktionen

5. Rückmeldungen an den Benutzer

- **Bestätigung und Fehlermeldungen**:
  - Nach jeder Aktion erhält der Benutzer eine visuelle Rückmeldung, ob die Aktion erfolgreich war oder ob ein Fehler aufgetreten ist (z.B. "Password added successfully!" oder "Password not found!").

6. Programmende

- **Beenden des Programms**:
  - Der Benutzer kann das Programm jederzeit durch Auswahl der Option "Exit" oder durch Drücken der ESC-Taste beenden. Vor dem Beenden werden alle ungespeicherten Änderungen gesichert.

Zusammenfassung des Ablaufdiagramms:

- **Start** -> **Eingabe des Master-Passworts** -> **Überprüfung** -> **Hauptmenü anzeigen** -> **Benutzer wählt Aktion** -> **Aktion ausführen** -> **Rückmeldung** -> **Zurück zum Hauptmenü** -> **Exit oder Weitere Aktion** -> **Ende**

Ergebnisse aller statischen und dynamischen Tools

Mypy

```
tobias@TobiasPc:~/projects/manager$ mypy main.py
Success: no issues found in 1 source file
tobias@TobiasPc:~/projects/manager$ mypy source
Success: no issues found in 2 source files
```

Pylint code rated 9.85/10

Unittests:

Run unittests

```
▶ Run python -m unittest discover -s tests -p "*.py" -v
testAddPasswordGenerateStrong (testInterface.TestCursesInterface.testAddPasswordGenerateStrong) ... ok
testAddPasswordStrong (testInterface.TestCursesInterface.testAddPasswordStrong) ... ok
testCheckPwnedPasswordNoPwnedPasswords (testInterface.TestCursesInterface.testCheckPwnedPasswordNoPwnedPasswords) ... ok
testCheckPwnedPasswordWithPwnedPasswords (testInterface.TestCursesInterface.testCheckPwnedPasswordWithPwnedPasswords) ... ok
testCheckReusedPasswordNoReusedPasswords (testInterface.TestCursesInterface.testCheckReusedPasswordNoReusedPasswords) ... ok
testCheckReusedPasswordWithReusedPasswords (testInterface.TestCursesInterface.testCheckReusedPasswordWithReusedPasswords) ... ok
testDeletePassword (testInterface.TestCursesInterface.testDeletePassword) ... ok
testDrawMenu (testInterface.TestCursesInterface.testDrawMenu) ... ok
testDrawMenuDifferentSize (testInterface.TestCursesInterface.testDrawMenuDifferentSize) ... ok
testGetInput (testInterface.TestCursesInterface.testGetInput) ... ok
testGetInputEmpty (testInterface.TestCursesInterface.testGetInputEmpty) ... ok
testGetPassword (testInterface.TestCursesInterface.testGetPassword) ... ok
testGetPasswordInput (testInterface.TestCursesInterface.testGetPasswordInput) ... ok
testGetPasswordInputEmpty (testInterface.TestCursesInterface.testGetPasswordInputEmpty) ... ok
testInitialization (testInterface.TestCursesInterface.testInitialization) ... ok
testRun (testInterface.TestCursesInterface.testRun) ... ok
testSearchPassword (testInterface.TestCursesInterface.testSearchPassword) ... ok
testUpdatePassword (testInterface.TestCursesInterface.testUpdatePassword) ... ok
testGetHiddenPassword (testMain.TestPasswordManagerInterface.testGetHiddenPassword) ... ok
testMainCorrectPassword (testMain.TestPasswordManagerInterface.testMainCorrectPassword) ... ok
testMainIncorrectPassword (testMain.TestPasswordManagerInterface.testMainIncorrectPassword) ... ok
testAddPassword (testPasswordManager.TestPasswordManager.testAddPassword) ... ok
testCheckPasswordStrength (testPasswordManager.TestPasswordManager.testCheckPasswordStrength) ... ok
testCheckPwnedPassword (testPasswordManager.TestPasswordManager.testCheckPwnedPassword) ... ok
testCheckReusedPassword (testPasswordManager.TestPasswordManager.testCheckReusedPassword) ... ok
testDeletePassword (testPasswordManager.TestPasswordManager.testDeletePassword) ... ok
testGenerateKey (testPasswordManager.TestPasswordManager.testGenerateKey) ... ok
testGenerateStrongPassword (testPasswordManager.TestPasswordManager.testGenerateStrongPassword) ... ok
testGetPassword (testPasswordManager.TestPasswordManager.testGetPassword) ... ok
testLoadDataEmptyFile (testPasswordManager.TestPasswordManager.testLoadDataEmptyFile) ... ok
testLoadDataInvalidToken (testPasswordManager.TestPasswordManager.testLoadDataInvalidToken) ... ok
testSaveData (testPasswordManager.TestPasswordManager.testSaveData) ... ok
testSearchPassword (testPasswordManager.TestPasswordManager.testSearchPassword) ... ok
testUpdatePassword (testPasswordManager.TestPasswordManager.testUpdatePassword) ... ok

----------------------------------------------------------------------
Ran 34 tests in 0.052s

OK
```

Coverage:

Run coverage

```
▶ Run coverage run -m unittest discover -s tests -p "*.py"
..................................
----------------------------------------------------------------------
Ran 34 tests in 0.094s

OK
Generated password: <Oz1>2fP{=r3
Name                          Stmts   Miss  Cover   Missing
-----------------------------------------------------------
main.py                          52      7    87%   26-31, 72
source/interface.py             206     46    78%   63, 65-70, 79-91, 107, 112, 115-128, 152, 154, 182, 229, 237-244, 288-289
source/passwordManager.py        96      9    91%   44, 94, 96, 121, 132, 136, 159-161
tests/testInterface.py          153      1    99%   232
tests/testMain.py                53      1    98%   65
tests/testPasswordManager.py     83      2    98%   87, 113
-----------------------------------------------------------
TOTAL                           643     66    90%
Wrote JSON report to coverage.json
```