Dokumentation Passwortmanager

Philipp Loose, Geiger Leo

**Inhalt:**

1. Programmablauf
2. Funktionen
3. Nutzerinterface
4. Architektur
5. Speicherformat
6. Tests

**Programmablauf**

Bla Bla Bla bli bla blub mit Bildern den groben ablauf erklären

**Funktionen**

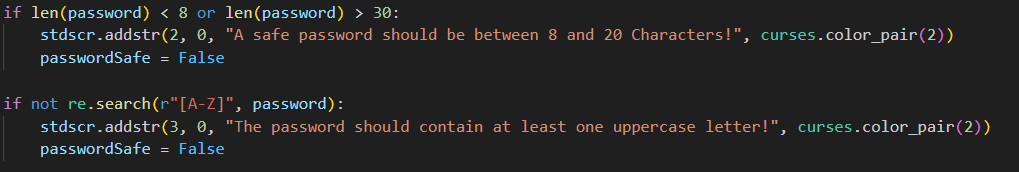
**Verschiedene Nutzer:**

Jeder Nutzer hat sein eigenes Masterpasswort und seine eigene JSON-Datei in der die Passwörter mit allen vorhandenen Infos verschlüsselt gespeichert werden.

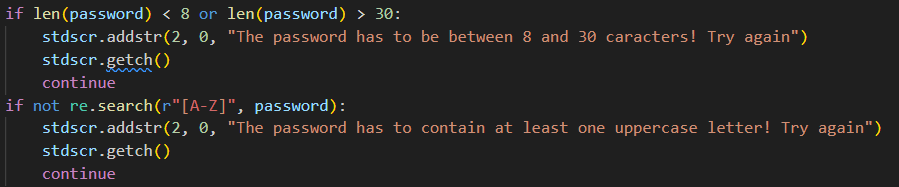
**Passwörter:**

Beim Setzen des Masterpasswort und im angemeldeten Zustand, beim Registrieren eines neuen Services, bei dem ein eigenes Passwort verwendet wird, wird bei einem schwachen Passwort auf alle möglichen Schwächen hingewiesen und es wird die Möglichkeit geboten anhand einer „Hilfestellung“ ein Passwort zu erstellen, das Groß- und Kleinschreibung, Zahlen und Sonderzeichen enthält, zusätzlich wird es noch auf Leaks geprüft. Wenn diese Anforderungen erfolgreich eingehalten werden, ist das Passwort sicher und es kann gespeichert werden.

Hinweis auf mögliche Schwachstellen:



Erstellung mit Guidelines:

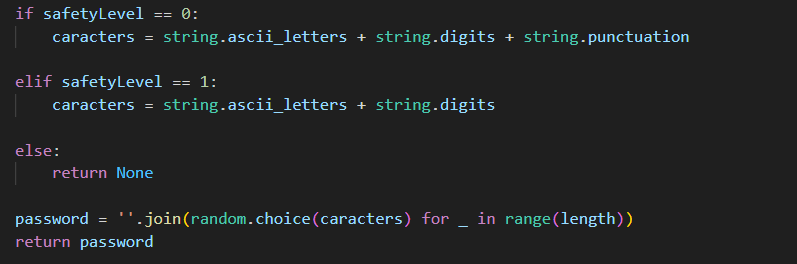


**\*BILDER IM PROGRAMM EINFÜGEN\***

**Automatische Passwörter:**

Wird bei der Passworterstellung die Generierung eines automatischen Passworts ausgewählt, muss eine gewünschte länge zwischen 8 und 30 Zeichen gegeben werden und zwischen der mittleren und starken Passwortstärke gewählt werden. Bei der mittleren Passwortstärke werden Groß- und Kleinbuchstaben und Zahlen in einer zufälligen Reihenfolge in der gegebenen Länge erstellt. Mit der starken Option werden noch Sonderzeichen miteingearbeitet aber sonst gleich.

Generierung der automischen Passwörter:



**Passwortverifizierung:**

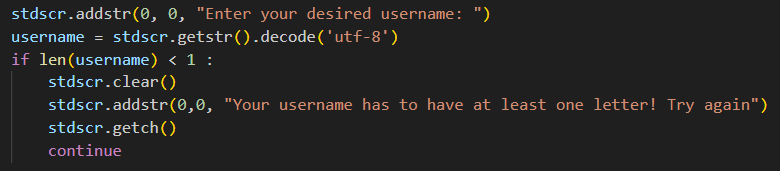
Die Passwortverifizierung läuft so, dass durch If-statements festgestellt wird, welche der Anforderungen an ein gutes Passwort erfüllt sind, und wenn sie nicht gegeben sind, wird in roter Schrift ausgegeben was noch fehlt und wie oben erwähnt die Option mithilfe der Guidelines das Passwort zu erstellen oder es so zu lassen gegeben. Sind alle Anforderungen erfüllt, so wird in Grüner Schrift ausgegeben, dass das Passwort sicher ist, und es geht mit den anderen Schritten weiter.

**\*BILDER IM PROGRAMM EINFÜGEN\***

**Hinzufügen von Username, Service und Notizen:**

Beim Registrieren eines neuen Services und Usernames wird geprüft, ob die gegebenen Namen eine Mindestlänge von 1 haben, dass keine leeren Services gespeichert werden. Dann wird das Passwort wie oben genannt überprüft und hinzugefügt und zu guter Letzt gibt es die Möglichkeit zu dem Registrierten Service eine Notiz hinzuzufügen. Wird hier nein gewählt, so wird nichts gespeichert, und einfach eine Leere Notiz gezeigt, bei ja kann eine beliebig lange Nachricht angegeben werden.

Prüfung ob der Username eine Mindestlänge von 1 einhält:



**Speichern und aufrufen der Informationen:**

Die Speicherung der Daten verläuft in einer JSON-Datei. Beim Programmstart schaut das Programm, ob es schon eine User- und diverse Passwortdateien gibt und öffnet diese, sonst werden sie erst beim Neuerstellen eines Users oder beim Speichern eines Passworts erstellt. Die ganzen User werden in einer „Users.json“ Datei gespeichert, in der sich der Name und das verschlüsselte Masterpasswort befindet. In der „passwords\_user.json“ wird der Service, der Username, das Passwort und die Zeit der Erstellung oder der letzten Änderung verschlüsselt gespeichert.

Um den Nutzer zu verifizieren, wird der Name genutzt und durch die „users.json“ gefiltert. Wenn es einen Treffer gibt, wird das verschlüsselte Passwort mit dem dementsprechenden Algorithmus entschlüsselt und mit dem eingegebenen Wert verglichen.

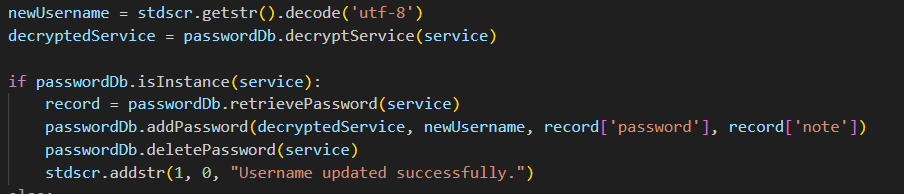
Wenn man sich ein gespeichertes Passwort ansehen will, muss sich mit dem Nutzer angemeldet werden und die Option „retrieve password“ ausgewählt werden. Dann wird eine Liste der gespeicherten Services angezeigt, aus denen dann der gesuchte ausgewählt werden kann. Nach diesen Services wird auch in der „passwords\_user.json“ gefiltert. Da der Service auch verschlüsselt gespeichert wird muss für die Filterung erst jeder Service dekodiert werden, dass man mit dem gegebenen Wert vergleichen kann. Wenn dann einer der Services ausgewählt wurde, werden der Username, das Passwort, die Notiz und das gespeicherte Datum mit der Zeit entschlüsselt angezeigt.

**\*USERS.JSON UND PASSWORD.JSON BILD EINFÜGEN\***

**Ändern aller Informationen:**

Es gibt die Möglichkeit in die in der „password\_user.json“ Datei gespeicherten Daten jeweils einzeln zu bearbeiten. Hier wird wie bei der „retrieve password“ Funktion zuerst nach dem zu bearbeitenden Service gefragt und dann nach dem zu bearbeitenden Teil des Services. Wenn man zwischen Passwort, Username oder Notiz gewählt hat wird wieder anhand des Services die richtige Information herausgesucht und kann dann bearbeitet werden. Nachdem zum Beispiel ein neuer Username eingegeben wurde, werden die anderen Informationen, die mit dem Username gespeichert sind als neues JSON-Objekt gespeichert und das alte JSON-Objekt wird gelöscht, so sind alle neuen Daten vorhanden und das Datum wird an die letzte Änderung angepasst.

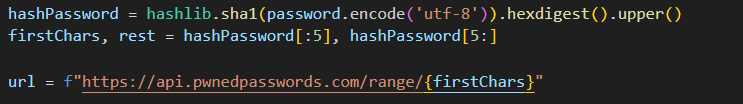
Bearbeiten des Usernames:



**Überprüfen ob Passwort veröffentlicht:**

Um zu überprüfen ob das von einem gewünschte Passwort je aus irgend welchen Datenbänken veröffentlicht wurde, wird die „HaveIBeenPawned“ Api genutzt, die anhand der ersten 5 Zeichen des gehashten Passworts überprüft ob das eingegebene Passwort noch sicher ist. Diese Überprüfung wurde in die Verifizierung des Passworts eingebaut und wird jedes mal durchgeführt, wenn ein neues Passwort registriert wird oder das Passwort geändert wird.

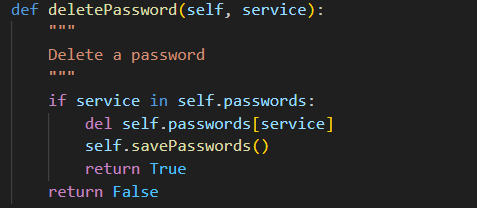
Api Aufruf mit gehashtem Passwort:



**Passwörter löschen:**

Das Löschen des Passwortes funktioniert ähnlich wie das aufrufen und das ändern, es wird wieder der Service abgefragt, woraufhin die JSON-Datei danach durchsucht wird und wenn der richtige Eintrag gefunden wurde wird dieser mit der delete Funktion gelöscht.

Delete Funktion in der PasswortDatabase Klasse:



**Nutzerinterface**

Das Nutzerinterface wurde mit Curses erstellt, was es ermöglicht zum Beispiel nur mit Pfeiltasten in dem Programm zu navigieren. In dem Passwortmanager wurde in den Menüs die Option gewählt, mit den Pfeiltasten eine der Optionen auszuwählen und dann mit Enter zu bestätigen. Für Eingaben wurde die „getstr()“ Funktion des Curses Pakets genutzt um Strings festzuhalten.

Für manche Ausgaben wurde die standart Textausgabe „addstr()“ genutzt, die einfach einen Text in dem Terminal ausgibt. Dies wurde zum Beispiel für das Anzeigen der Warnungen bei einem schwachen Passwort genutzt, wobei hier auch noch Farbe hinzugefügt wurde.

Beispiel für „addstr()“ mit Farbe:



\*BILDER VON UI IN HAUPTMENÜ UND VERSCHIEDENEN UNTERMENÜS\*

**Architektur**

Die ganze Logik des Programms wurde in die Main.py Datei implementiert, wobei hier drei Klassen eingeführt wurden, welche die Ver- und Entschlüsselung, die Benutzer und die Passwortdatenbank verwalten. Dazu gibt es 23 Funktionen, die in Zusammenarbeit mit den Grundklassen alle Logik und das Graphische Interface des Programms beinhalten. Es gibt auch mehrere Funktionen, die nicht viel Logik enthalten, sondern nur dafür verantwortlich sind entweder die Menüs anzuzeigen oder je nach Auswahl an die richtige Funktion weiterzuleiten.

**\*IRGENDWELCHE BILDER EINFÜGEN\***

**Speicherformat**

Die meisten der Gespeicherten Daten wurden mit dem AES-256 Algorithmus verschlüsselt bevor diese in einer JSON-Datei gespeichert wurden. Der Aufbau der dieser Datei ist in diesem Falle so:

1. Users.json

Nutzername: Name {

Passwort: verschlüsseltes Passwort

}

1. Password\_User.json

Service: verschlüsselter Service {

Nutzername: verschlüsselter Nutzername

Passwort: verschlüsseltes Passwort

Notiz: unverschlüsselte Notiz

Datum: verschlüsseltes Datum

}

**\*BILD JSON FILE VON USER UND PASSWORT IN SCHÖN FORMATIERT\***

JSON wurde als Speicherformat für den Passwortmanager gewählt, weil es eine einfache, leicht lesbare und gut strukturierte Möglichkeit bietet, Daten in einem text-Format zu speichern. JSON ermöglicht eine flexible Organisation der Daten, was für die Speicherung von vielen Informationen wie Services, Benutzernamen und Passwörtern ideal ist. Zudem ist JSON weit verbreitet und wird von vielen Programmiersprachen und Tools unterstützt, was die Integration und Weiterverarbeitung der gespeicherten Daten erleichtert.

**Tests**

**Unittest**

**\*BILD VOM UNITTEST EINFÜGEN\***

Wie viele Tests durchgeführt wurden und wie die Coverage am Ende ist.

**Pylint**

**\*BILD VON DEM PYLINT ERGEBIS EINFÜGEN\***

Muss auf 0 Fehler sein sonst schlechte Note

Es wurden für jede Funktion Docstrings (“““so“““) eingefügt, die grob beschreiben was passiert

**\*BILD VON GUTEM DOCSTRING BEISPIEL EINFÜGEN\***

**Type Checker – MyPy**

**\*KLEINES BILD VON MYPY CHECKER EINFÜGEN\***

Definitionen und Aufrufe von Funktionen müssen vorhanden sein, was unmöglich ist.