# Sprawozdanie z projektu Manager Haseł

Autorzy: Kamil Grzywaczewski (s25130) i Martyna Kwaśniak(s27640)

## Temat Projektu

Celem projektu było stworzenie menedżera haseł z graficznym interfejsem użytkownika, umożliwiającego rejestrację, logowanie oraz bezpieczne przechowywanie danych dostępowych do różnych serwisów użytkowników w lokalnej bazie danych. Aplikacja zawiera mechanizmy rejestracji użytkownika, logowania, a także funkcje dodawania, edytowania i usuwania wpisów haseł.

## Użyte technologie i biblioteki

Projekt wykorzystuje następujące technologie:

- Python 3.13

- SQLite (baza danych)

- Tkinter (interfejs graficzny)

- SQLAlchemy (ORM do bazy danych)

- Pillow (obsługa grafik – wymagane przez `ImageTk`)

- bcrypt (hashowanie haseł)

## Struktura projektu

**main.py** – wejście do aplikacji. Inicjalizuje interfejs użytkownika

**gui.py** – tworzy interfejs graficzny użytkownika po zalogowaniu. Zawiera elementy interakcji z użytkownikiem

**passwordTable.py** -zawiera klasę odpowiedzialną za tworzenie i obsługę tabeli z hasłami. Umożliwia edycję, dodawanie i usuwanie rekordów.

**databaseModels.py** - definiuje modele bazy danych (tabele, relacje) przy pomocy SQLAlchemy.

**database.py** - tworzy i konfiguruje połączenie z lokalną bazą danych SQLite, używając SQLAlchemy jako warstwy ORM.

**auth.py** - zawiera logikę rejestracji i logowania użytkownika. Odpowiada za hashowanie haseł (bcrypt) i ich weryfikację.

## Struktura bazy danych

Tabela Users:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Zawartość wygenerowana przez AI może być niepoprawna.

Tabela Passwords:

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Zawartość wygenerowana przez AI może być niepoprawna.

## Szczegółowy opis implementacji:

1. **Okno powitalne**

Obraz zawierający tekst, zrzut ekranu, Czcionka, zieleń

Zawartość wygenerowana przez AI może być niepoprawna.

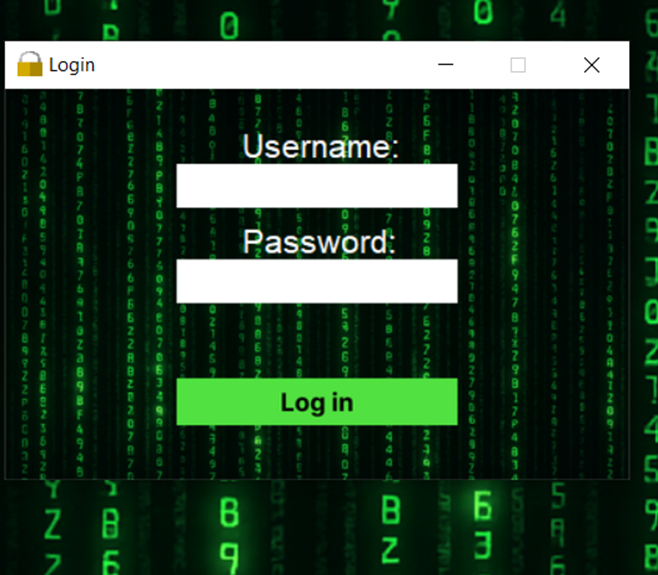
Główne okno programu, czyli ekran powitalny, jest tworzone w funkcji start\_gui(). Tworzy ono podstawowe okno aplikacji (tk.Tk()), ustawia jego tytuł oraz wymiary, a następnie wyświetla prosty tekst powitalny i dwa przyciski umożliwiające dalszą interakcję z programem. Fragment odpowiedzialny za tę logikę:

def start\_gui():  
 root = tk.Tk()  
 root.title("Password Manager - Start")  
 root.geometry("400x300")  
  
 tk.Label(root, text="Witaj w Password Manager!", font=("Helvetica", 16)).pack(pady=20)  
 tk.Button(root, text="Zaloguj się", command=lambda: open\_login\_window(root)).pack(pady=10)  
 tk.Button(root, text="Zarejestruj się", command=lambda: open\_register\_window(root)).pack(pady=10)  
  
 root.mainloop()

W tym kodzie tk.Tk() tworzy instancję głównego okna. Ustawiane są jego podstawowe właściwości, takie jak tytuł i rozmiar. tk.Label() wyświetla komunikat powitalny z nazwą aplikacji, natomiast tk.Button() tworzy przyciski, które umożliwiają przejście do logowania lub rejestracji użytkownika.

Przyciski te są powiązane z funkcjami open\_login\_window() oraz open\_register\_window(), które tworzą nowe okna (tk.Toplevel) do obsługi logowania i rejestracji. Mechanizm ten działa w oparciu o przypisanie funkcji do argumentu command w konstruktorze przycisków, co jest standardowym podejściem w programowaniu z użyciem tkinter.

Całość interfejsu graficznego działa w głównej pętli zdarzeń (root.mainloop()), która monitoruje akcje użytkownika i aktualizuje interfejs w czasie rzeczywistym.

1. **Po kliknięciu w SIGN IN**

Po kliknięciu przycisku „SIGN IN” na ekranie powitalnym, wywoływana jest funkcja handle\_login(root). Funkcja ta tworzy nowe okno typu tk.Toplevel, czyli osobne, podrzędne okno względem głównego GUI.

login\_window = tk.Toplevel(root)  
login\_window.title("Login")  
login\_window.geometry("400x250")

Wewnątrz funkcji handle\_login() zdefiniowana jest funkcja submit\_login(), która odpowiada za obsługę kliknięcia w przycisk logowania:

def submit\_login():  
 username = username\_entry.get()  
 password = password\_entry.get()  
 user = login\_user(username, password)  
 if user:  
 dane\_uzytkownika = get\_user\_passwords(user)  
 login\_window.destroy()  
 PasswordTableApp(root, dane\_uzytkownika)  
 else:  
 messagebox.showerror("Błąd logowania", "Nieprawidłowy login lub hasło")

Funkcja login\_user(username, password) (z pliku auth.py) sprawdza, czy użytkownik istnieje i czy hasło jest poprawne (porównując je z hashem w bazie danych, używając bcrypt).

Po pomyślnym zalogowaniu:

* Dane użytkownika zostają pobrane funkcją get\_user\_passwords(user), która zwraca listę zapisanych haseł z bazy.
* Okno logowania zostaje zamknięte (login\_window.destroy()).
* Tworzony jest nowy widok aplikacji: PasswordTableApp, czyli interfejs z tabelą haseł przypisaną do danego użytkownika.

Po błędnym zalogowaniu:

* Zostaje wyświetlony komunikat błędu za pomocą tk.messagebox.showerror().

1. **Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

   Zawartość wygenerowana przez AI może być niepoprawna.Po kliknięciu** „**SIGN UP**”

Po kliknięciu przycisku „SIGN UP” w ekranie startowym aplikacji, uruchamiana jest funkcja handle\_register(), która otwiera osobne okno rejestracyjne przy użyciu tk.Toplevel() z biblioteki tkinter.

register\_window = tk.Toplevel()  
register\_window.title("Register")  
register\_window.geometry("400x250")

Wewnątrz handle\_register() znajduje się funkcja submit\_register(), która odpowiada za logikę po kliknięciu przycisku „SIGN UP”:

def submit\_register():  
 username = username\_entry.get()  
 password = password\_entry.get()  
  
 if not username or not password:  
 messagebox.showwarning("Błąd", "Uzupełnij wszystkie pola.")  
 return  
  
 if not is\_strong\_password(password):  
 messagebox.showwarning("Błąd", "Hasło powinno zawierać min. 8 znaków, duże litery, cyfry i symbole.")  
 return  
  
 if register\_user(username, password):  
 messagebox.showinfo("Sukces", "Rejestracja zakończona pomyślnie.")  
 register\_window.destroy()  
 else:  
 messagebox.showerror("Błąd", "Użytkownik już istnieje.")

Funkcja register\_user() (z pliku auth.py) sprawdza, czy użytkownik już istnieje w bazie.

Jeżeli nie istnieje, hasło jest hashowane algorytmem bcrypt i zapisywane wraz z nazwą użytkownika do tabeli User.

1. **Obraz zawierający zrzut ekranu, wyświetlacz, oprogramowanie, tekst

   Zawartość wygenerowana przez AI może być niepoprawna.Tabela haseł zalogowanego użytkownika**

Po pomyślnym zalogowaniu, aplikacja przekazuje użytkownika do klasy PasswordTableApp, która odpowiada za graficzne przedstawienie i obsługę jego haseł. Jest to centralne miejsce, w którym użytkownik może przeglądać, dodawać, edytować oraz usuwać swoje dane logowania.

class PasswordTableApp:  
 def \_\_init\_\_(self, root, dane, user\_id):  
 self.root = root  
 self.dane = dane  
 self.user\_id = user\_id

Widok użytkownika dzieli się na kilka warstw i ramek:

* **Center Frame** – centralna biała strona z tabelą i przyciskami.
* **TreeView** (ttk.Treeview) – tabela, która wyświetla dane użytkownika w trzech kolumnach: Aplikacja, Login, Hasło.
* self.tree = ttk.Treeview(  
   self.left\_frame,  
   columns=("aplikacja", "login", "haslo"),  
   show="headings",  
   height=15  
  )

Wszystkie dane logowania są powiązane z user\_id, co gwarantuje, że użytkownik po zalogowaniu widzi tylko swoje dane. Dane są pobierane z tabeli PasswordEntry (model SQLAlchemy) filtrowanej po identyfikatorze zalogowanego użytkownika.

1. **Dodaj hasło**

Funkcja open\_add\_password\_window() tworzy nowe okno (tk.Toplevel), w którym użytkownik może wprowadzić dane nowego wpisu: nazwę aplikacji, login oraz hasło. Interfejs składa się z trzech pól tekstowych i przycisku zapisu (w dalszym kodzie).

popup = tk.Toplevel(self.root)  
popup.title("Dodaj nowe hasło")  
popup.geometry("300x200")  
  
tk.Label(popup, text="Aplikacja:").pack()  
entry\_app = tk.Entry(popup)  
entry\_app.pack()  
  
tk.Label(popup, text="Login:").pack()  
entry\_login = tk.Entry(popup)  
entry\_login.pack()  
  
tk.Label(popup, text="Hasło:").pack()  
entry\_pass = tk.Entry(popup, show="\*")  
entry\_pass.pack()

Po kliknięciu „Zapisz” dane są zapisywane w bazie SQLite przy pomocy SQLAlchemy jako nowy obiekt PasswordEntry, przypisany do aktualnego user\_id.

1. **Zmień hasło**

Funkcja change\_password() umożliwia edycję wybranego wpisu. Funkcja wymusza, by zaznaczone było dokładnie jedno hasło do edycji. W nowym oknie użytkownik podaje stare hasło i nowe hasło (dwukrotnie).

selected\_indices = [idx for idx, var in enumerate(self.checkbox\_vars) if var.get() == 1]  
if len(selected\_indices) != 1:  
 messagebox.showwarning("Błąd", "Zaznacz dokładnie jedno hasło do zmiany.")  
 return  
  
popup = tk.Toplevel(self.root)  
popup.title("Zmień hasło")  
  
tk.Label(popup, text="Stare hasło:")  
entry\_old = tk.Entry(popup, show="\*")  
entry\_old.pack()  
  
tk.Label(popup, text="Nowe hasło:")  
entry\_new = tk.Entry(popup, show="\*")  
entry\_new.pack()  
  
tk.Label(popup, text="Powtórz nowe hasło:")  
entry\_repeat = tk.Entry(popup, show="\*")  
entry\_repeat.pack()

Dane są następnie walidowane i aktualizowane w bazie danych. Nowe hasło nadpisuje stare dla danego wpisu.

1. **Pokaż hasło**

Funkcja show\_Passwords() odpowiada za ujawnienie hasła w tabeli. W bazowej wersji hasła są ukryte (np. jako \*\*\*\*\*\*), a ta metoda aktualizuje kolumnę haslo w Treeview, pokazując faktyczną wartość z bazy danych:

for idx, var in enumerate(self.checkbox\_vars):  
 if var.get() == 1:  
 self.tree.set(idx, "haslo", self.dane[idx]["haslo"])

1. **Usuń hasło**

Funkcja przeszukuje listę haseł i identyfikuje te, które zostały zaznaczone (przy pomocy checkboksów). Następnie usuwa odpowiadające im rekordy z bazy danych:

for idx, var in enumerate(self.checkbox\_vars):  
 if var.get() == 1:  
 rekord = self.dane[idx]  
 to\_delete\_ids.append(rekord["id"])

Usuwanie wykonywane jest przez SQLAlchemy

entry = session.query(PasswordEntry).filter\_by(id=pid, user\_id=self.user\_id).first()  
if entry:  
 session.delete(entry)

Po zakończeniu operacji tabela jest odświeżana (self.refresh\_table()), aby użytkownik zobaczył zaktualizowaną listę.