Sprawozdanie z projektu Manager Haseł

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

1. 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.

2. 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)

3. 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.

4. Struktura bazy danych

Tabela Users:

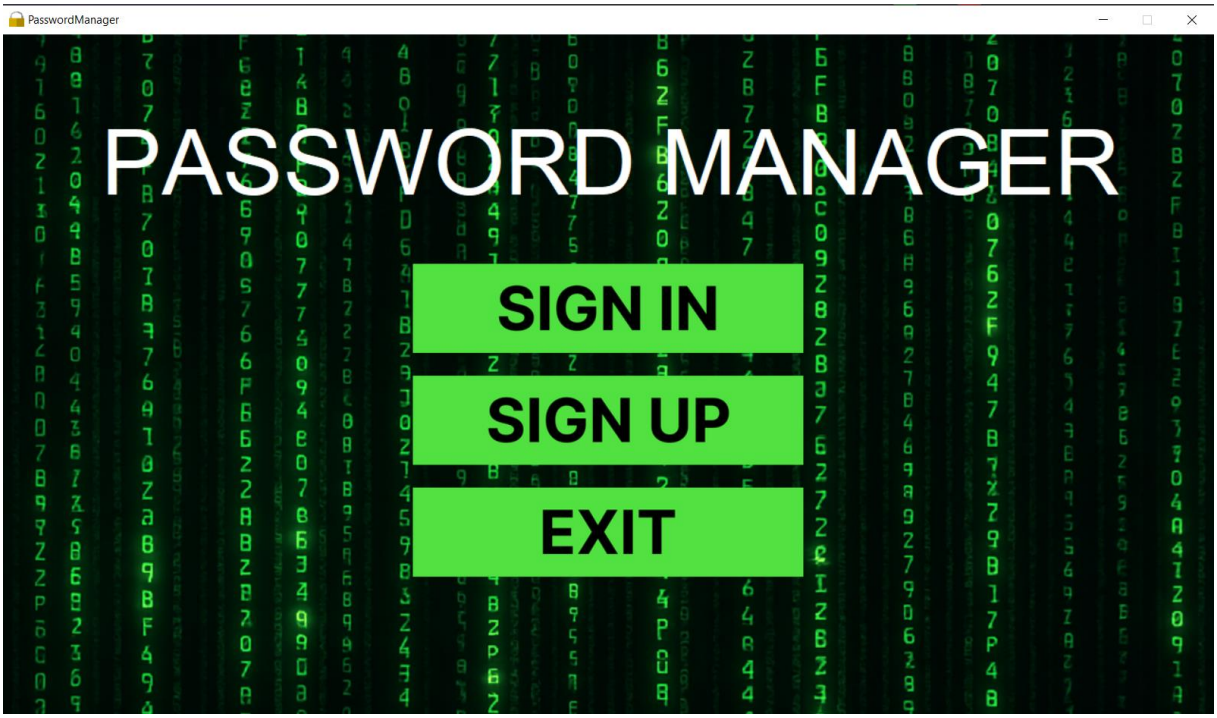
	id	username	master_password
	Filtr	Filtr	Filtr
1	1	admin	$2b$12$CxxHP2vSg6dha9XG.b7OsOijI0m7b…
2	2	marti	$2b$12$MY48tCDEHBguN3yhkhlOUurUM9IvY…
3	3	kamilos1000	$2b$12$UuI6XMZTgMdsWfLc8PakEelxkiRCd…
4	4	S25130	$2b$12$IaSu9j1/…

Tabela Passwords:

	id	service_name	service_username	encrypted_password	*user_id*
	Filtr	Filtr	Filtr	Filtr	Filtr
1	1	Gmail	admin.123	321admin123	1
2	2	Gakko	s25130	#Puszek121	3
3	3	Netflix	admin	Warszawa123!	1

5. Szczegółowy opis implementacji:

a. Okno powitalne



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:

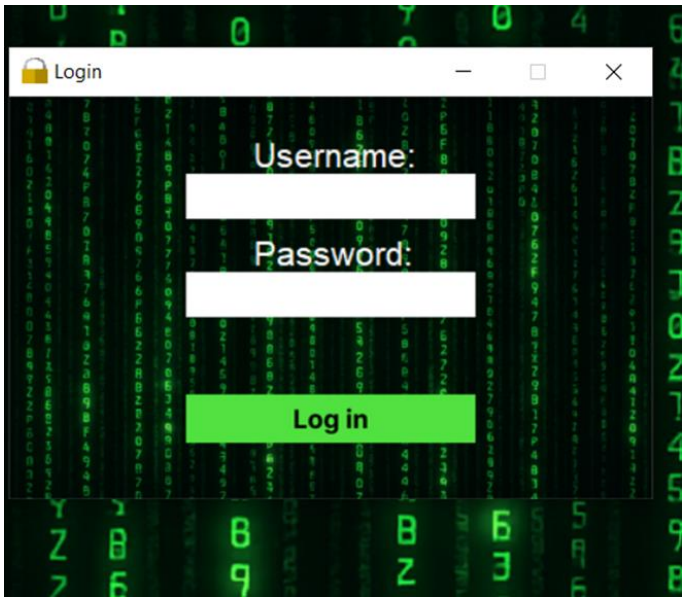
```
def start_gui():
    root = tk.Tk()
    root.title("Password Manager - Start")
    root.geometry("400x300")

    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.

b. 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).

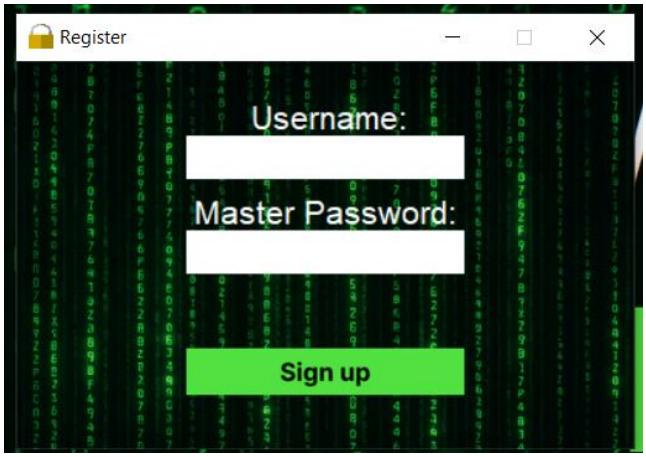
Po pomyślnym zalogowaniu:

- 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().

c. **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")
```

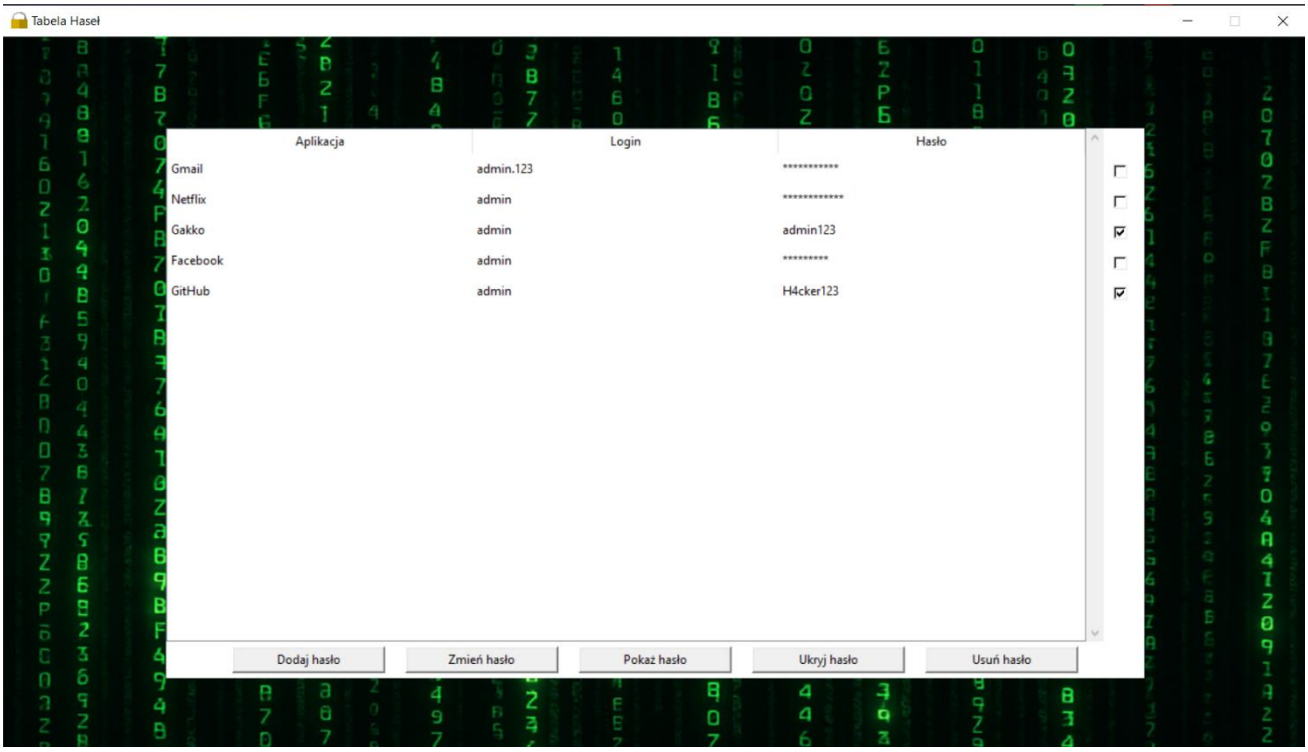
Wewnątrz handle_register() znajduje się funkcja submit_register(), która odpowiada za logikę po kliknięciu przycisku „SIGN UP”:

```
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.

d. 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.

e. **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.

f. **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.

g. **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"])
```

h. **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.