



**WYŻSZA SZKOŁA  
INFORMATYKI i ZARZĄDZANIA**  
z siedzibą w Rzeszowie

# Projekt Okienkowy Menadżer Haseł

Jakub Serafin, Szymon Sutyła,  
Szymon Kojder, Adrian Lokaj

6IIZ/2021-GPL06

Inżynieria oprogramowania  
Prowadzący: Leszek Puzio

Rzeszów 2024

## Spis treści

Okienkowy menadżer haseł.....	3
Słownik pojęć.....	4
Specyfikacja wymagań.....	4
Diagram przypadków użycia .....	6
Opis przypadków użycia:.....	6
Diagramy.....	7
Diagram encji .....	7
Diagram klas .....	8
Implementacja – wzorce projektowe .....	9
1. Singleton.....	9
2. Obserwator (Observer).....	9
3. Model-Widok-Kontroler (MVC) .....	9
Dokumentacja kodu .....	9
Testy jednostkowe.....	11
Testy jednostkowe dla klasy root_window .....	11
Prezentacja aplikacji.....	12
.....	12
Podsumowanie .....	12
Wnioski.....	13

# Okienkowy menadżer haseł

Projekt "Menadżer haseł" jest aplikacją desktopową napisaną w języku Python, która umożliwia użytkownikom bezpieczne przechowywanie, zarządzanie oraz wyszukiwanie swoich danych logowania. Aplikacja została zbudowana z wykorzystaniem różnych bibliotek i technologii, w tym tkinter do budowy interfejsu graficznego oraz sqlite3 do zarządzania lokalną bazą danych. Poniżej szczegółowy opis projektu:

## *Funkcjonalności aplikacji:*

1. **Zarządzanie hasłami:**
  - Aplikacja umożliwia użytkownikowi dodawanie nowych rekordów zawierających tytuł, nazwę użytkownika oraz hasło.
  - Użytkownik może edytować istniejące wpisy, aktualizując ich tytuł, nazwę użytkownika lub hasło.
  - Istnieje również możliwość usuwania wybranych rekordów z bazy danych.
2. **Wyszukiwanie haseł:**
  - Aplikacja umożliwia szybkie wyszukiwanie haseł na podstawie ich tytułu. Użytkownik może wprowadzić fragment tytułu, a aplikacja zwróci odpowiednie rekordy.
3. **Kopiowanie haseł:**
  - Każde hasło w aplikacji można skopiować do schowka systemowego jednym kliknięciem, co ułatwia szybkie korzystanie z zapisanych danych logowania.
4. **Interfejs użytkownika:**
  - Interfejs graficzny aplikacji jest przejrzysty i łatwy w obsłudze. Zawiera listę rekordów haseł oraz formularze do dodawania, edytowania i usuwania haseł.
  - Rekordy haseł są wyświetlane w postaci tabeli, gdzie widoczne są tytuły, nazwy użytkowników (jeśli są zapisane) oraz zaszyfrowane hasła.

## *Technologie i biblioteki użyte w projekcie:*

1. **Tkinter:**
  - Tkinter jest standardową biblioteką do tworzenia aplikacji graficznych w Pythonie. W projekcie została wykorzystana do stworzenia interfejsu użytkownika, w tym okienek, przycisków, pól tekstowych oraz listy rekordów.
2. **SQLite3:**
  - SQLite3 jest lekką bazą danych SQL, która jest wykorzystywana w aplikacji do przechowywania danych logowania. Każdy rekord w bazie zawiera tytuł, nazwę użytkownika (opcjonalnie) oraz hasło.
3. **Pyperclip:**
  - Pyperclip jest biblioteką do pracy ze schowkiem systemowym w Pythonie. Umożliwia szybkie kopiowanie i wklejanie danych, co jest używane w aplikacji do kopiowania haseł do schowka.

## *Cel projektu:*

Celem projektu jest zapewnienie użytkownikom prostego, intuicyjnego narzędzia do bezpiecznego przechowywania i zarządzania danymi logowania. Aplikacja ma na celu zwiększenie bezpieczeństwa użytkowników poprzez eliminację potrzeby pamiętania wielu haseł oraz ułatwienie zarządzania nimi w sposób zorganizowany i efektywny.

### *Potencjał rozwoju:*

Projekt "Menadżer haseł" ma potencjał do rozwoju poprzez dodanie dodatkowych funkcji, takich jak:

- **Zaawansowane opcje wyszukiwania** pozwalające na filtrowanie haseł według różnych kryteriów.
- **Zaawansowane opcje bezpieczeństwa**, np. szyfrowanie haseł w bazie danych.
- **Synchronizacja z chmurą**, aby użytkownicy mogli mieć dostęp do swoich haseł na różnych urządzeniach.
- **Zarządzanie grupami haseł**, umożliwiające organizację haseł według różnych kategorii.

Dzięki elastycznej strukturze projektowej i zastosowaniu dobrych praktyk programistycznych aplikacja jest gotowa do dalszego rozwoju i dostosowania do zmieniających się potrzeb użytkowników.

## Słownik pojęć

- **Menadżer haseł:** Aplikacja do bezpiecznego przechowywania i zarządzania hasłami użytkowników.
- **Baza danych SQLite:** Lekka, wbudowana baza danych SQL, używana w aplikacji do przechowywania danych o hasłach.
- **Tkinter:** Biblioteka Pythona do tworzenia aplikacji desktopowych opartych na GUI.
- **Unit testy:** Testy jednostkowe w Pythonie, które sprawdzają poprawność funkcjonalności poszczególnych części aplikacji.

## Specyfikacja wymagań

### *1. Wymagania funkcjonalne*

#### 1.1. Zarządzanie hasłami

- **Dodawanie rekordu:**
  - Użytkownik może dodać nowy rekord zawierający tytuł, nazwę użytkownika oraz hasło.
  - Pola "Tytuł" i "Hasło" są obowiązkowe, pole "Nazwa użytkownika" jest opcjonalne.
- **Edycja rekordu:**
  - Użytkownik może edytować istniejący rekord, aktualizując tytuł, nazwę użytkownika oraz hasło.
  - Aplikacja powinna umożliwić wybór rekordu do edycji poprzez zaznaczenie go na liście.
- **Usuwanie rekordu:**
  - Użytkownik może usunąć wybrany rekord z bazy danych.
  - Usunięcie rekordu wymaga potwierdzenia przez użytkownika.

#### 1.2. Wyszukiwanie haseł

- **Wyszukiwanie po tytule:**
  - Użytkownik może wyszukiwać rekordy na podstawie fragmentu tytułu.
  - Wyniki wyszukiwania są wyświetlane w tabeli.

### 1.3. Kopiowanie hasła do schowka

- **Kopiowanie hasła:**
  - Użytkownik może skopiować hasło do schowka systemowego jednym kliknięciem.
  - Hasło jest kopiowane w postaci jawnej, jednak w GUI jest pokazywane jako ciąg gwiazdek dla bezpieczeństwa.

## 2. Wymagania dotyczące interfejsu użytkownika

### 2.1. Okno główne

- **Układ okna:**
  - Okno główne aplikacji zawiera nagłówek z tytułem "Menadżer haseł".
  - Główne elementy interfejsu są umieszczone w logicznie uporządkowanych sekcjach: formularz CRUD, przyciski akcji, tabela rekordów.

### 2.2. Operacje CRUD

- **Przyciski CRUD:**
  - Przycisk "Zapisz" do dodawania nowych rekordów.
  - Przycisk "Aktualizuj" do edytowania zaznaczonych rekordów.
  - Przycisk "Usuń" do usuwania zaznaczonych rekordów.
  - Przycisk "Skopiuj hasło" do kopiowania hasła zaznaczonego rekordu.
  - Przycisk "Pokaż wszystkie dane" do wyświetlania wszystkich rekordów w bazie danych.

### 2.3. Widok tabeli

- **Tabela rekordów:**
  - Tabela wyświetla listę wszystkich haseł w bazie danych, pokazując ID, tytuł, nazwę użytkownika i zaszyfrowane hasło (gwiazdki).
  - Użytkownik może zaznaczyć wiersz w tabeli, aby edytować lub usunąć dany rekord.

### 2.4. Wyszukiwanie

- **Pole wyszukiwania:**
  - Pole tekstowe umożliwiające wpisanie fragmentu tytułu do wyszukania.
  - Przycisk "Wyszukaj" uruchamiający operację wyszukiwania i wyświetlający wyniki w tabeli.

## 3. Wymagania bezpieczeństwa

### 3.1. Przechowywanie danych

- **Baza danych:**
  - Hasła są przechowywane w lokalnej bazie danych SQLite.
  - Każdy rekord w bazie danych zawiera ID, datę utworzenia, datę aktualizacji, tytuł, nazwę użytkownika (opcjonalnie) i hasło.

### 3.2. Wyświetlanie haseł

- **Maskowanie haseł:**
  - Hasła są wyświetlane w GUI jako ciąg gwiazdek, aby zapobiec przypadkowemu podglądowi.

- Pełne hasło jest ujawniane jedynie podczas kopiowania do schowka systemowego.

### 3.3. Operacje na hasłach

- **Kopiowanie hasła:**
  - Użytkownik może skopiować hasło do schowka systemowego poprzez kliknięcie odpowiedniego przycisku.
  - Kopiowanie hasła wymaga zaznaczenia rekordu w tabeli.

### 3.4. Bezpieczeństwo interfejsu

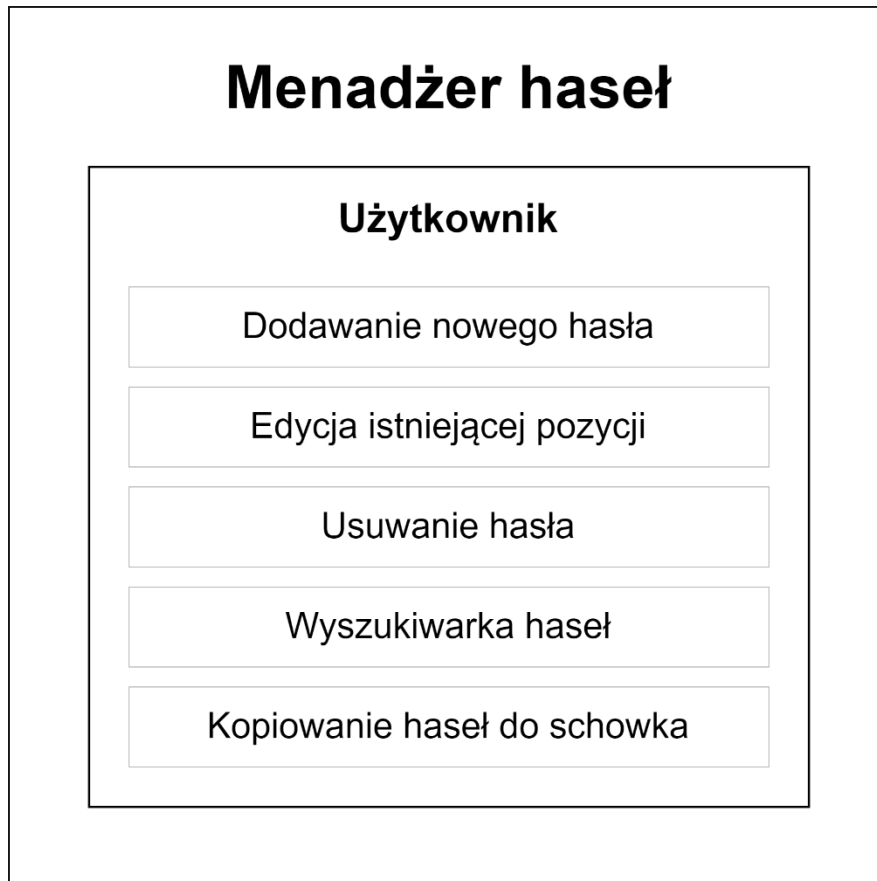
- **Zabezpieczenia GUI:**
  - Przycisk "Usuń" wymaga potwierdzenia operacji przez użytkownika, aby zapobiec przypadkowym usunięciom.
  - Formularze wejściowe posiadają walidację, aby upewnić się, że wymagane pola są wypełnione poprawnie.

## Diagram przypadków użycia

### Opis przypadków użycia:

1. **Dodaj nowe hasło:**
  - Użytkownik dodaje nowe hasło w aplikacji, wpisując tytuł, nazwę użytkownika i hasło.
  - Dane są zapisywane w bazie danych.
2. **Edytuj istniejące hasło:**
  - Użytkownik wybiera istniejące hasło z listy.
  - Edytuje tytuł, nazwę użytkownika lub hasło.
  - Zmiany są zapisywane w bazie danych.
3. **Usuń hasło:**
  - Użytkownik wybiera istniejące hasło z listy.
  - Potwierdza usunięcie hasła.
  - Hasło zostaje usunięte z bazy danych.
4. **Wyszukaj hasło:**
  - Użytkownik wprowadza frazę do wyszukania.
  - Aplikacja wyszukuje hasła zawierające podaną frazę.
  - Wyniki są wyświetlane w tabeli.
5. **Kopiuj hasło do schowka:**
  - Użytkownik wybiera hasło z listy.
  - Klikając na przycisk "Kopiuj hasło", hasło jest kopiowane do schowka systemowego.
6. **Pokaż wszystkie hasła:**
  - Aplikacja wyświetla wszystkie hasła przechowywane w bazie danych.
  - Użytkownik może przewijać listę hasel i przeglądać ich szczegóły.

Diagram przedstawia główne funkcje, które są dostępne dla użytkownika w aplikacji Menadżera hasel. każdy przypadek użycia reprezentuje konkretną funkcjonalność, która może być wykonana przez użytkownika.

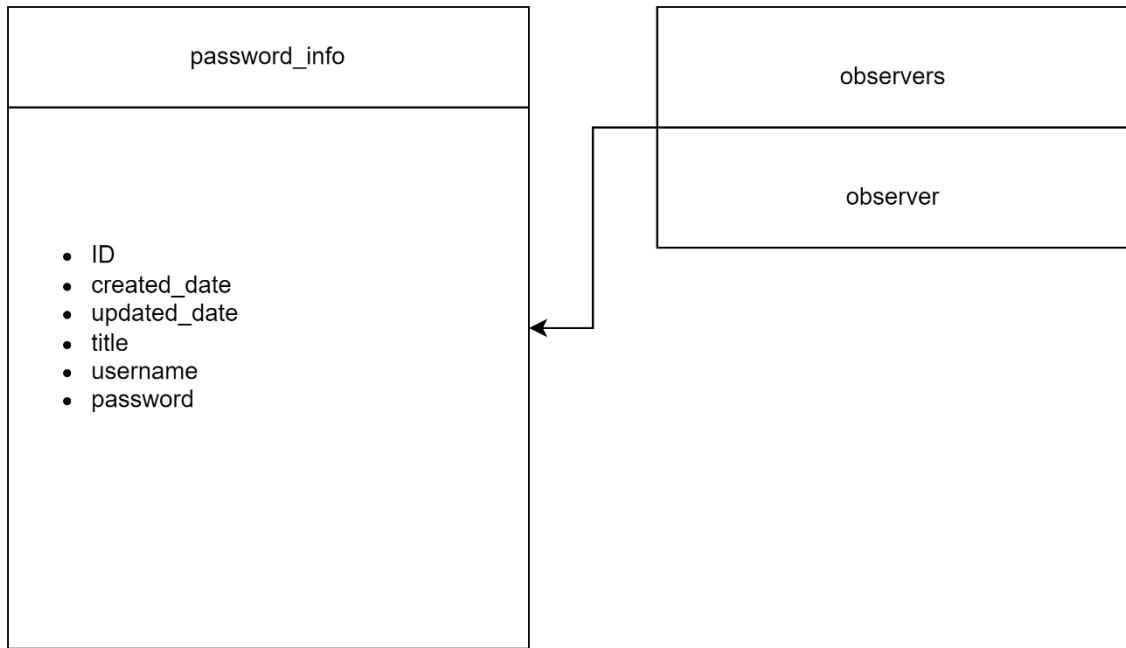


## Diagramy

### Diagram encji

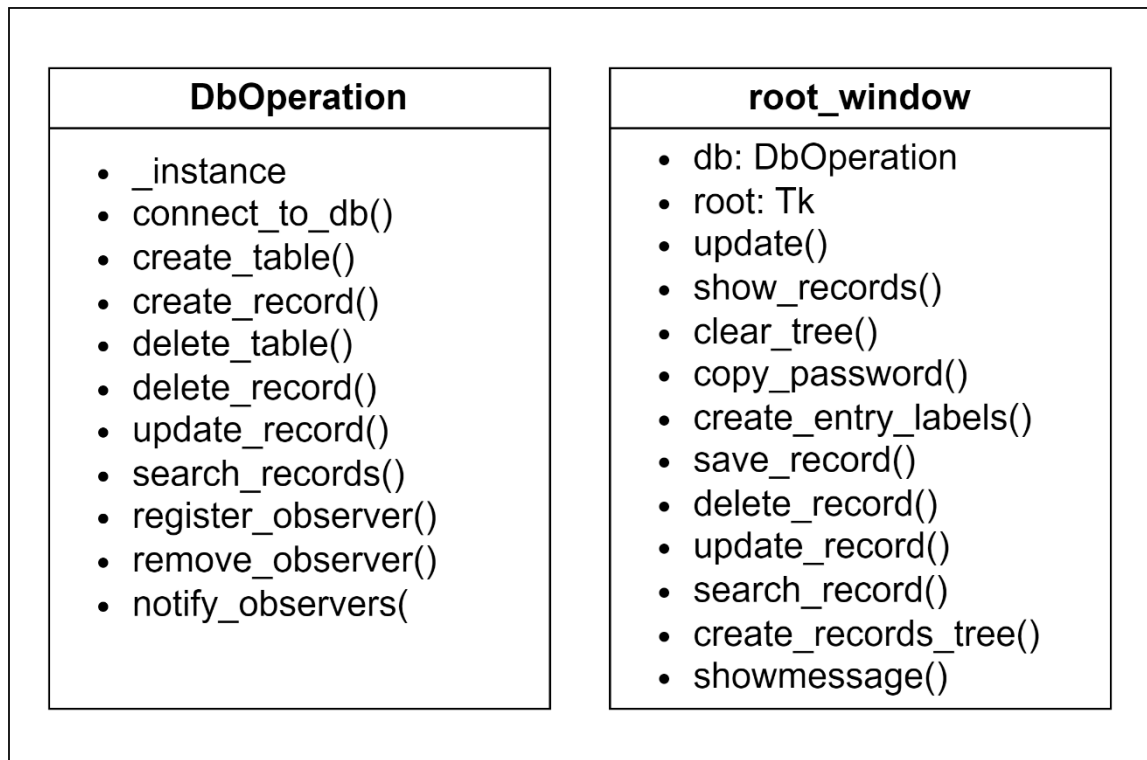
Diagram encji przedstawia strukturę danych w aplikacji Menadżera haseł, czyli jak informacje są przechowywane w bazie danych SQLite.

- **password\_info**: Tabela przechowująca informacje o hasłach.
  - ID: Klucz główny
  - created\_date, updated\_date: Daty utworzenia i aktualizacji rekordu
  - title, username, password: Informacje o hasle
- **observers**: Klasa lub interfejs dla wzorca Obserwator, umożliwiająca powiadamianie GUI o zmianach w danych.



## Diagram klas

Diagram klas/interfejsów przedstawia strukturę klas i interfejsów w aplikacji Menadżera haseł, które są odpowiedzialne za obsługę operacji na danych oraz interakcję z użytkownikiem.



- **DbOperation:** Klasa zarządzająca operacjami na bazie danych SQLite.
- **root\_window:** Klasa obsługująca GUI aplikacji, interakcje użytkownika oraz powiadamianie o zmianach w danych.



# Implementacja – wzorce projektowe

W projekcie aplikacji "Menadżer haseł" zastosowano kilka wzorców projektowych, które pomagają w organizacji kodu, separacji odpowiedzialności oraz umożliwiają łatwiejsze rozszerzanie funkcjonalności. Oto opis zastosowanych wzorców projektowych:

## 1. Singleton

Wzorec Singleton został zastosowany do klasy DbOperation, która zarządza operacjami na bazie danych SQLite. Gwarantuje on, że klasa ta będzie miała tylko jedną instancję w czasie działania programu. Wzorec ten jest użyteczny w kontekście zarządzania połączeniem do bazy danych i zapewnienia, że istnieje tylko jedno połączenie do bazy danych w całej aplikacji.

## 2. Obserwator (Observer)

Wzorec Obserwatora został wykorzystany do powiadamiania interfejsu użytkownika (klasa root\_window) o zmianach w danych przechowywanych w bazie. Klasa DbOperation pozwala rejestrować obserwatorów i powiadamiać ich za pomocą metody notify\_observers(), gdy nastąpią zmiany w danych.

## 3. Model-Widok-Kontroler (MVC)

MVC jest architekturą wzorców projektowych, która została częściowo zastosowana w aplikacji poprzez podział na:

- **Model:** Reprezentowany przez klasę DbOperation, zarządzającą danymi w bazie.
- **Widok:** Reprezentowany przez klasę root\_window, która odpowiada za interfejs użytkownika i wyświetlanie danych.
- **Kontroler:** Kontrolery są zaimplementowane bezpośrednio w root\_window, zarządzają interakcjami użytkownika i wywołują odpowiednie metody w DbOperation.

Zastosowane wzorce projektowe pomagają w organizacji i strukturyzacji kodu aplikacji "Menadżer haseł", co ułatwia zarządzanie danymi, interfejsem użytkownika oraz zapewnia rozszerzalność i elastyczność w przypadku ewentualnych zmian i dodatkowych funkcji. Ich wybór jest uzasadniony potrzebą efektywnej pracy z bazą danych, obsługi interfejsu użytkownika oraz powiadamiania o zmianach w danych.

# Dokumentacja kodu

W projekcie kod został zdokumentowany za pomocą docstringów, które są specjalnymi komentarzami wewnątrz kodu służącymi do generowania dokumentacji. Docstringi są umieszczane w definicjach klas, metod i funkcji, aby opisać ich działanie, parametry oraz zwracane wartości. Narzędzie pydoc w Pythonie może wykorzystać te docstringi do automatycznego generowania dokumentacji.

**Pydoc** – Za pomocą pydoc wygenerowano dokument w formacie HTML dla klas DbOperation oraz root\_window


Module Index

Search...

## API Documentation

class root\_window

root\_window()
db
root
crud\_frame
search\_entry
password\_dict
update()
show\_records()
clear\_tree()
copy\_password()
create\_entry\_labels()
create\_crud\_buttons()
create\_entry\_boxes()
save\_record()
delete\_record()
update\_record()
search\_record()
create\_records\_tree()
showmessage()

built with


## password\_manager

<b>class</b> root_window:	<a href="#">View Source</a>
root_window(root, db)	<a href="#">View Source</a>
db	
root	
crud_frame	
search_entry	
password_dict	
def update(self):	<a href="#">View Source</a>
def show_records(self):	<a href="#">View Source</a>
def clear_tree(self):	<a href="#">View Source</a>
def copy_password(self):	<a href="#">View Source</a>
def create_entry_labels(self):	<a href="#">View Source</a>
def create_crud_buttons(self):	<a href="#">View Source</a>
def create_entry_boxes(self):	<a href="#">View Source</a>
def save_record(self):	<a href="#">View Source</a>
def delete_record(self):	<a href="#">View Source</a>
def update_record(self):	<a href="#">View Source</a>
def search_record(self):	<a href="#">View Source</a>
def create_records_tree(self):	<a href="#">View Source</a>


Module Index

Search...

## API Documentation

class DbOperation

connect\_to\_db()
create\_table()
create\_record()
show\_records()
delete\_table()
update\_record()
delete\_record()
search\_records()
register\_observer()
remove\_observer()
notify\_observers()

built with


## db\_operations

<b>class</b> DbOperation:	<a href="#">View Source</a>
def connect_to_db(self):	<a href="#">View Source</a>
def create_table(self, table_name='password_info'):	<a href="#">View Source</a>
def create_record(self, data, table_name='password_info'):	<a href="#">View Source</a>
def show_records(self, table_name='password_info'):	<a href="#">View Source</a>
def delete_table(self, table_name='password_info'):	<a href="#">View Source</a>
def update_record(self, data, table_name='password_info'):	<a href="#">View Source</a>
def delete_record(self, ID, table_name='password_info'):	<a href="#">View Source</a>
def search_records(self, title, table_name='password_info'):	<a href="#">View Source</a>
def register_observer(self, observer):	<a href="#">View Source</a>
def remove_observer(self, observer):	<a href="#">View Source</a>
def notify_observers(self):	<a href="#">View Source</a>

# Testy jednostkowe

## **test\_create\_record**

Test sprawdza poprawność funkcji `create_record()` klasy `DbOperation`, która dodaje nowy rekord do bazy danych. Sprawdzane są:

- Czy rekord został poprawnie dodany do bazy danych.
- Czy wszystkie pola zostały ustawione zgodnie z oczekiwaniami.

## **test\_update\_record**

Test sprawdza poprawność funkcji `update_record()` klasy `DbOperation`, która aktualizuje istniejący rekord w bazie danych. Sprawdzane są:

- Czy rekord został poprawnie zaktualizowany.
- Czy nowe wartości zostały zapisane w bazie danych.

## **test\_delete\_record**

Test sprawdza poprawność funkcji `delete_record()` klasy `DbOperation`, która usuwa rekord z bazy danych. Sprawdzane są:

- Czy rekord został poprawnie usunięty z bazy danych.
- Czy po usunięciu liczba rekordów w bazie jest zgodna z oczekiwaniami.

## **test\_search\_records**

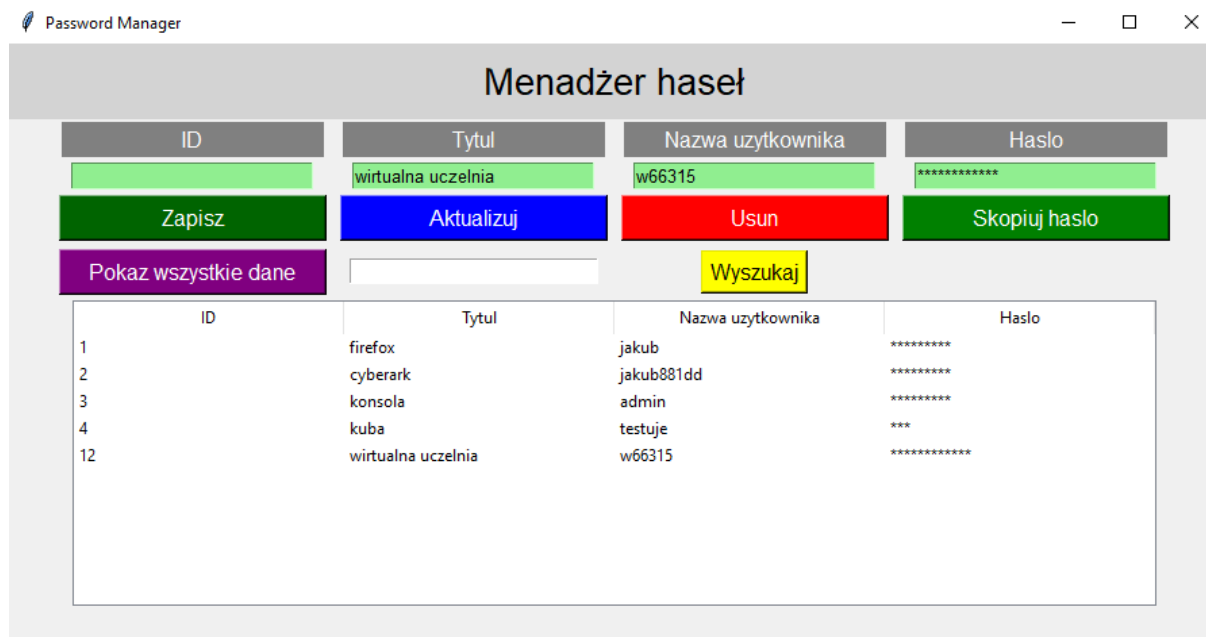
Test sprawdza poprawność funkcji `search_records()` klasy `DbOperation`, która wyszukuje rekordy w bazie danych na podstawie tytułu. Sprawdzane są:

- Czy funkcja zwraca oczekiwane wyniki wyszukiwania.

## Testy jednostkowe dla klasy `root_window`

Testy jednostkowe dla interfejsu użytkownika `root_window` są zdefiniowane w osobnym teście jednostkowym i sprawdzają funkcjonalność poszczególnych metod interfejsu użytkownika, takich jak dodawanie, aktualizowanie, usuwanie, wyszukiwanie haseł oraz operacje związane z interakcją użytkownika.

# Prezentacja aplikacji



## Podsumowanie

Aplikacja "Password Manager" to desktopowy menadżer haseł zaprojektowany i zaimplementowany w języku Python. Wykorzystuje bibliotekę tkinter do stworzenia interfejsu graficznego oraz sqlite3 do zarządzania bazą danych. Projekt ten stanowi kompletny system do przechowywania, zarządzania i wyszukiwania haseł użytkowników w bezpieczny sposób.

## Kluczowe Funkcjonalności

### 1. Zarządzanie Hasłami:

- Dodawanie nowych rekordów haseł.
- Edytowanie istniejących rekordów.
- Usuwanie wybranych haseł.
- Wyszukiwanie haseł na podstawie tytułu.

### 2. Interfejs Graficzny:

- Przyjazny dla użytkownika GUI zbudowany przy użyciu tkinter.
- Widok tabeli wyświetlający listę wszystkich przechowywanych haseł.
- Pola tekstowe i przyciski umożliwiające operacje CRUD (Create, Read, Update, Delete).
- Pole wyszukiwania do filtrowania rekordów.

### 3. Bezpieczeństwo:

- Hasła są przechowywane w bazie danych SQLite.
- W interfejsie użytkownika hasła są wyświetlane jako ukryte (przykryte gwiazdkami) dla zwiększenia bezpieczeństwa.
- Funkcja kopiowania hasła do schowka pozwala na bezpieczne przeniesienie hasła do innej aplikacji.

## Architektura i Wzorce Projektowe

Aplikacja wykorzystuje wzorzec Singleton do zarządzania instancją klasy DbOperation, co gwarantuje, że tylko jedna instancja tej klasy istnieje przez cały czas działania programu. Ponadto, zastosowano wzorzec Observer do powiadamiania interfejsu graficznego o zmianach w bazie danych, co zapewnia aktualizację wyświetlanych danych w czasie rzeczywistym.

## Testy Jednostkowe

Testy jednostkowe zostały przeprowadzone przy użyciu modułu unittest. Zaimplementowane testy pokrywają wszystkie kluczowe funkcje operacji na bazie danych, w tym tworzenie, aktualizowanie, usuwanie i wyszukiwanie rekordów. Dodatkowo, przetestowano interakcje użytkownika z GUI za pomocą mockowania metod bazy danych.

## Dokumentacja

Kod został bogato udokumentowany przy użyciu docstringów, co umożliwia generowanie dokumentacji za pomocą narzędzia pydoc. Każda klasa i metoda zawiera opis ich funkcji, parametrów i zwracanych wartości, co znacznie ułatwia zrozumienie i rozwijanie kodu.

## Wnioski

Projekt "Password Manager" stanowi kompleksowe rozwiązanie do zarządzania hasłami w lokalnym środowisku desktopowym. Przyjazny dla użytkownika interfejs graficzny, wspierany przez solidną bazę danych i przemyślaną architekturę kodu, zapewnia efektywne i bezpieczne zarządzanie poufnymi danymi. Implementacja wzorców projektowych oraz pełne pokrycie testami jednostkowymi świadczą o wysokiej jakości kodu. Aplikacja jest nie tylko funkcjonalna, ale także łatwa w utrzymaniu i rozwoju.