Password Manager

Team Members	Registration number
Moksh Shah	221080064
Jash Shah	221080062

Problem Statement :

Create a user-friendly PasswordManager with SQlite database connection which allows users to securely store and manage passwords. Challenges faced are development of responsive design using Tkinter GUI, stroring users credentials, including username and passwords in an encrypted database, to generate and suggest strong random passwords.

Motivation:

The motivation driving the development of this Python code lies in creation of a secure and efficient password manager using Tkinter library for graphical user interface(GUI) and SQLite for database management. Code aims to leverage the simplicity and reliability SQLite for secure password storage with an intuitive and user-friendly interface ,providing users with a robust platform to manage their passwords seamlessly. It offers a convenient and secure way to manage passwords, ultimately enhancing overall online security for users.

Methodology:

The methodology for the code involves using the SQLite database and Tkinter library to create a graphical user interface (GUI) for a Password Manager application. Here's a breakdown of the key steps in the code:

1. Importing Libraries:
   - The code starts by importing necessary libraries, including Tkinter, sqlite3, os,prettytable.

2. Creating a database:
   - Creating a database for storing the passwords. Attempt to establish a connection to the”user.db”database.
-Create a table with various columns like id , website,url,passwords etc in database

3. Defining Functions:
   - Making function that checks whether database is present , if not creates a new one in the give path along with the table .
- Creating a function that will help us to connect to database .

4. Storing and deleting Passwords:
   - A function that takes input from the user and stores the information along with the password in the database .
- Similarly a function that deletes the record of data stored in database when its respectice Id is given.

5. Display :
   - Showing all the details in the table stored in database except the password.
   - We use a module named prettytable for this part .

6. Random Password Generator:
   - Create a function that generates random password for the user .

7. Creating Tkinter Window and the GUI elements:

- EntryBox prompt user to add required information
- MessageBox used to ask a yes/no type of Question
- Menu to allow user to choose the action he wants to do
- Dialog for confirmation to confirm certain actions
- Main window is the main window for GUI
- Labels are used to display text that provide information to the user

8. Execution:
   - The code runs, and the GUI window is displayed.

Pseudocode:

_user_db.py

```
import sqlite3
import os

def create_dbtable():
    """
    Function to create a table named 'Userdatabase' in the SQLite
database if it does not exist.

    The table has the following columns:
    - Id: INTEGER PRIMARY KEY
    - Website: CHAR(50)
    - URL: TEXT(500)
    - Username: CHAR(50)
    - Email: TEXT(100)
    - Password: TEXT(100)
    - Description: TEXT(5000)
    """
    conn =
sqlite3.connect(r"C:\\Users\\jashs\\OneDrive\\Desktop\\PasswordMa
nager\\iTHack-
PassMag\\Password_Manager\\user\\leveldb\\user.db")
    cur = conn.cursor()
    cur.execute('''CREATE TABLE IF NOT EXISTS Userdatabase
                   (Id INTEGER PRIMARY KEY,
                   Website CHAR(50),
                   URL TEXT(500),
                   Username CHAR(50),
                   Email TEXT(100),
                   Password TEXT(100),
                   Description TEXT(5000));''')
    conn.commit()
    conn.close()

def create_database():
    """
    Function to check if the directory and database file exist,
and create them if not.

    The database file 'user.db' is created using the
create_dbtable function.
    """
    if
os.path.exists(r'C:\\Users\\jashs\\OneDrive\\Desktop\\PasswordMan
ager\\iTHack- PassMag\\Password_Manager\\user\\leveldb\\'):
        if not
os.path.exists(r"C:\\Users\\jashs\\OneDrive\\Desktop\\PasswordMan
ager\\iTHack-
PassMag\\Password_Manager\\user\\leveldb\\user.db"):
            create_dbtable()
    else:
        os.mkdir(r'C:\\Users\\jashs\\OneDrive\\Desktop\\PasswordM
anager\\iTHack- PassMag\\Password_Manager\\user\\leveldb')
        create_dbtable()

def connect_database():
    """
    Function to connect to the SQLite database and return the
connection object.
```

```
    Returns:
    - conn: SQLite connection object
    """
    conn = 
sqlite3.connect(r"C:\\Users\\jashs\\OneDrive\\Desktop\\PasswordMa
nager\\iTHack-
PassMag\\Password_Manager\\user\\leveldb\\user.db")
    return conn

if __name__ == "__main__":
    create_database()
```

_db_manager.py

```
from Password_Manager.user._user_db import connect_database
import prettytable as PrettyTable

def store_password(website, url, username, email, password, 
description):
    """
    Function to store a new password entry in the database.

    Parameters:
    - website: Website name (str)
    - url: Website URL (str)
    - username: Username for the website (str)
    - email: Email associated with the account (str)
    - password: Password for the account (str)
    - description: Additional description or notes (str)
    """
    conn = connect_database()
    myCur = conn.cursor()

    sqlQuery = "INSERT INTO UserDataBase (Website, URL, Username, 
Email, Password, Description) VALUES(?, ?, ?, ?, ?, ?)"
    val = (website, url, username, email, password, description)
    myCur.execute(sqlQuery, val)

    conn.commit()
    conn.close()

def delete_password(acc_id):
    """
    Function to delete a password entry from the database based
on the given account ID.

    Parameters:
    - acc_id: Account ID to be deleted (int)
    """
    conn = connect_database()
    myCur = conn.cursor()
    sqlQuery = "DELETE FROM UserDataBase WHERE Id=?"
    accToDelete = (acc_id,)
    myCur.execute(sqlQuery, accToDelete)
    conn.commit()
    conn.close()
    print("[-] Record deleted")

def show_details():
    """
    Function to display all password entries stored in the
database in a tabular format.
    """
    conn = connect_database()
    myCur = conn.cursor()
    sqlQuery = "SELECT Id, Website, URL, Username, Email,
Password, Description FROM UserDataBase"

    myCur.execute(sqlQuery)
    rows = myCur.fetchall()

    myTable = PrettyTable.PrettyTable(["Id", "Website", "URL",
"Username", "Email", "Password", "Description"])
    for row in rows:
        myTable.add_row(row)
    print(myTable)

def clear_data():
    """
    Function to clear all records from the database.
    """
    conn = connect_database()
    myCur = conn.cursor()
    sqlQuery = "DELETE FROM UserDataBase"
    myCur.execute(sqlQuery)
    conn.commit()
    conn.close()
    print("[-] All records cleared")
```

_passmag_menu.py

```
from Password_Manager.user._db_manager import store_password,
show_details, delete_password

def menu():
    """
    Display a menu with options for the user.

    Returns:
    - str: User's choice input
    """
    print("""
        1. Store Password
        2. Show Details
        3. Delete Password
        4. Exit
        """)
    return input(" :")

def save_password():
    """
    Function to save a new password by taking input from the
user.

    The user is prompted to enter information such as website,
URL, username, email, password, and description.
    The entered information is then passed to the store_password
function.
    """
    print(f"\nWebsite:  URL:   Username:    Email:   Password:
")
```

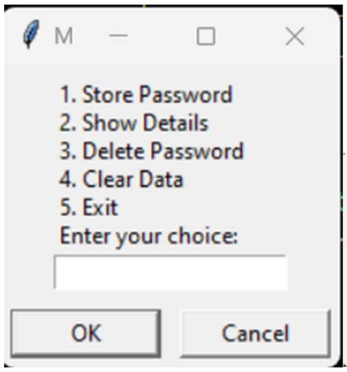
```
    website = input('\n Website:')
    print(f"\nWebsite: {website}
URL:   Username:    Email:   Password:  ")
    url = input('\n URL:')
    print(f"\nWebsite:
{website}  URL:{url}   Username:    Email:   Password:  ")
    username = input('\n Username:')
    print(f"\nWebsite:{website}  URL: {url}  Username:
{username}   Email:   Password:   ")
    email = input('\n Email:')
    print(f"\nWebsite:{website}  URL: {url}  Username:
{username}  Email:{email}   Password:   ")
    password = input('\n Password:')
    print(f"\nWebsite:{website}  URL:{url}   Username:{username}
   Email: {email} Password: {password}")
    description = input('\n Description:')

    store_password(website, url, username, email, password,
description)

def remove_password():
    """
    Function to remove a password entry based on user input.

    The function displays details using show_details, prompts the
user for the ID to delete,
    and asks for confirmation before calling delete_password.
    """
    show_details()
    del_id = input("\n [-] Id That you want to delete: ")
    warn_check = input("Are you sure (y/n):")
    if warn_check == "y" or warn_check == "Y":
        delete_password(del_id)
        print("\n[-] Successfully deleted")
    else:
        print("\n[+] Cancelled Successfully")

def show_entries():
    """
    Function to display all password entries stored in the
database using show_details.
    """
    show_details()
```

iThack.py

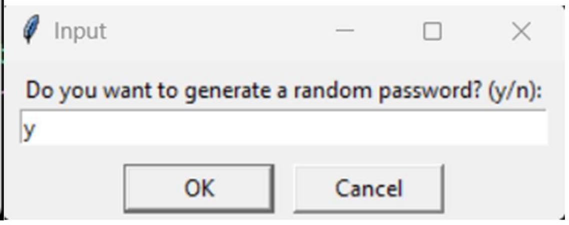
```
import tkinter as tk
import secrets
import string
from tkinter import simpledialog
from Password_Manager.user._db_manager import store_password,
show_details, delete_password, clear_data

def generate_random_password(length=12):
    """
    Generate a random password with a specified length.

    Parameters:
    - length (int): Length of the generated password. Default is
12.

    Returns:
    - str: Randomly generated password.
    """
    alphabet = string.ascii_letters + string.digits +
string.punctuation
    password = ''.join(secrets.choice(alphabet) for _ in
range(length))
    return password

def save_password_gui():
    """
    Function to interactively save a new password using a
graphical user interface.

    The user is prompted to enter information such as website,
URL, username, email, and password.
    The option to generate a random password is also provided.
    The entered information is then passed to the store_password
function.
    """
    website = simpledialog.askstring("Input", "Enter Website:")
    url = simpledialog.askstring("Input", "Enter URL:")
    username = simpledialog.askstring("Input", "Enter Username:")
    email = simpledialog.askstring("Input", "Enter Email:")

    # Ask if the user wants to generate a random password
    generate_password = simpledialog.askstring("Input", "Do you
want to generate a random password? (y/n):")
    if generate_password.lower() == 'y':
        password = generate_random_password()
    else:
        password = simpledialog.askstring("Input", "Enter
Password:")

    description = simpledialog.askstring("Input", "Enter
Description:")

    store_password(website, url, username, email, password,
description)

def remove_password_gui():
    """
    Function to interactively remove a password entry using a
graphical user interface.

    The user is prompted to enter the ID of the password entry to
be deleted,
    and confirmation is sought before calling the delete_password
function.
    """
    del_id = simpledialog.askstring("Input", "Enter Id to
delete:")
    warn_check = simpledialog.askstring("Input", "Are you sure?
(y/n):")

    if warn_check.lower() == "y":
        delete_password(del_id)
        print("\n[-] Successfully deleted")
    else:
        print("\n[+] Canceled Successfully")

def show_entries_gui():
    """
    Function to display all password entries stored in the
database using a graphical user interface.
    """
    show_details()

def clear_data_gui():
    """
    Function to interactively clear all data from the database
using a graphical user interface.

    Confirmation is sought before calling the clear_data
function.
    """
    # Ask for confirmation before clearing all data
    confirm_clear = simpledialog.askstring("Input", "Are you sure
you want to clear all data? This action cannot be undone.
(y/n):")

    if confirm_clear.lower() == "y":
        clear_data()
        print("\n[-] All records cleared")
    else:
        print("\n[+] Clear Data Canceled")

def main_gui():
    """
    Main function to run the graphical user interface for the
password manager.

    The user is presented with a menu to choose options like
storing passwords, showing details, deleting passwords, clearing
data, and exiting.
    The user can interactively perform these operations.
    """
    root = tk.Tk()
    root.withdraw()  # Hide the main window

    while True:
        choose = simpledialog.askstring("Menu", "1. Store
Password\n2. Show Details\n3. Delete Password\n4. Clear Data\n5.
Exit\nEnter your choice:")
```

```python
        if choose == "1":
            save_password_gui()
        elif choose == "2":
            show_entries_gui()
        elif choose == "3":
            remove_password_gui()
        elif choose == "4":
            clear_data_gui()
        elif choose == "5":
            break

if __name__ == "__main__":
    main_gui()
```

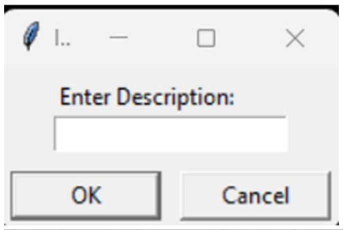
Output:



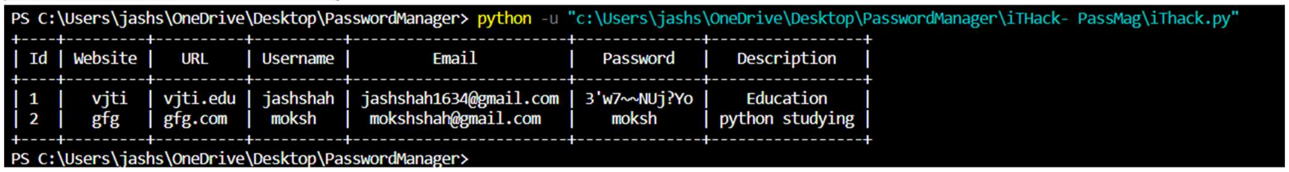
1. Store Password
2. Show Details
3. Delete Password
4. Clear Data
5. Exit
Enter your choice:



Do you want to generate a random password? (y/n):
y



Enter Description:



```
PS C:\Users\jashs\OneDrive\Desktop\PasswordManager> python -u "c:\Users\jashs\OneDrive\Desktop\PasswordManager\iTHack- PassMag\iThack.py"
+----+---------+----------+----------+-------------------------+---------------+------------------+
| Id | Website |   URL    | Username |          Email          |   Password    |   Description    |
+----+---------+----------+----------+-------------------------+---------------+------------------+
| 1  |  vjti   | vjti.edu | jashshah | jashshah1634@gmail.com  | 3'w7~~NUj?Yo  |    Education     |
| 2  |   gfg   | gfg.com  |  moksh   |  mokshshah@gmail.com    |    moksh      | python studying  |
+----+---------+----------+----------+-------------------------+---------------+------------------+
PS C:\Users\jashs\OneDrive\Desktop\PasswordManager>
```

Discussion:

Graphical User Interface (GUI) Design:

The code employs Tkinter, a popular Python GUI library, to create a user interface for the application. The use of labels, buttons, widgets contributes to a clean and intuitive design.

Database Integration:
For robust data storage and retrieval, we employ SQLite as the backend database. This choice ensures a lightweight and user-friendly solution for securely managing passwords. The code includes functionalities to create the necessary tables, establish connections, and interact with the database, bolstering the application's data management capabilities.

Modular Approach:

The code is organized into classes (db_manager and passmag_menu) and functions (store_password,delete_password,show_details) to promote a modular and maintainable structure. This design facilitates code readability and reusability.

Error Handling:

Robust error handling is implemented using try-except blocks, particularly around database connection and query execution. This practice enhances the application's resilience by providing informative error messages for developers and users alike.

User Interaction:

The application allows users to upload new passwords as well as generate new passwords with the help of a menu. It also allows users to delete a password that is no longer required or to change a password. The design of these windows and the associated buttons contributes to a user-friendly experience enhances the overall usability of the application.

Potential Enhancements:

Add a feature to assess the strength of the user-entered password or the generated password, providing feedback to the user.
Add a feature to copy passwords to the clipboard with a button click, making it convenient for users to use the generated or stored password.
Include a search functionality to allow users to search for specific entries in the password database.
Introduce a master password to secure access to the password manager application.
Provide options for users to backup and restore their password database, ensuring data integrity.

Usability and User Feedback:

The usability of the application is largely influenced by the straightforward design and functionality. Collecting user feedback on the current features can inform future updates and improvements, ensuring that the platform aligns with user expectations.

Testing and Debugging:

Adequate testing and debugging are essential for identifying and resolving issues. Developers should conduct thorough testing, including edge cases, to ensure the robustness and reliability of the application.