Password manager web application

This web application provides a platform for users to manage their passwords across various services. Built using Flask, a lightweight Python web framework, the application features a user-friendly interface that facilitates easy registration, login, and management of passwords.

Key features:

- **User registration and login:** Users can create account securely and log in to access their stored passwords.
- **Dashboard:** After login, users are greeted with a dashboard that displays all their password groups, allowing for easy navigation and management.
- **Password Groups:** Users can create groups to categorize passwords and make passwords easier to handle.
- **Password generating, adding, editing, and retrieving:** Passwords are stored securely to database after you add them. You can add new passwords or edit existing ones. While adding or editing passwords you can generate new secure random passwords.
- **Possibility to be deleted:** You are free to delete your passwords, password groups or whole account and no data about you is left in database after account is deleted.

Usage:

1. Register a new account or login to an existing one.
2. After login you are in the dashboard where you can see your account's existing password groups and you can also create new groups if you want.
3. After you have the password group created you can open it and start adding or editing password inside the group. You can also delete passwords you don't need.
4. After you are ready you can log out or delete your account.

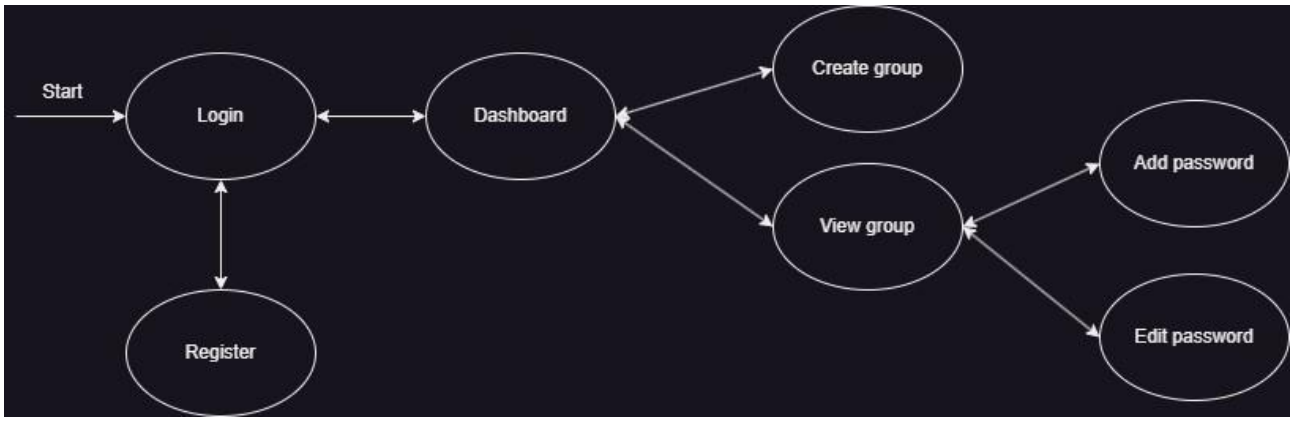
Here was a description about application, its key features, and its usage. If you want to see the application user interface structure in diagram form you can check picture 2 below.

Program structure:

I used a python web framework Flask for this web application. For user interface I used HTML for structure and CSS for styling. All the data is stored in SQL-database. Below is diagram about program structure and technologies I used.

Picture 1: Program technologies structure

Here is also chart to visualize program usage.



Picture 2: Program UI structure

Secure programming solutions:

User password hashing:

Master password used to log in to account is hashed before stored to database. This makes program more secure against data breaches and multiple kind of attacks, for example brute-force, rainbow-table, and dictionary attacks.

In code: Generated in app.py file at row 75. Hash checked at login in row 45.

Password encryption:

Passwords added to this program are encrypted with AES-256 encryption before they are stored in the database. This limits the damage if someone gets to database unauthorized and steals data. Passwords are decrypted after you view some password group to make it possible for user to copy real password.

In code: Password encrypting functions are in password_encrypting.py -file. Encryptions are called in app.py rows 160 and 192 while adding or editing password. Single password decryption is called in app.py row 203 while editing password. Multiple password decryption function is called in app.py row 110.

!ATTENTION!: In password encryption I have used system environment variable for secure AES key storage. In password_encrypting.py -file there is one already generated key for test using, but that is not secure way to use encryption.

Input handling:

Input handling makes XSS-attacks and SQL injections harder. User input of program has been handled by blocking some suspicious characters which are not needed in input forms and then could be blocked. This is probably not recommended way to do it, but I had no time to do input handling better now and it should be improved before using this program in real scenario.

In code: function in app.py line 26. Function is called in multiple app routes where user input has been read.

User authentication

User can only access passwords of his own account. Passwords are also stored behind the master password of the user.

In code: login and register html-files and their routes in app.py.

CSRF-protection:

CSRF-tokens are used in program to prevent CSRF-attacks.

In code: At every html-file where is form used. In app.py rows 269 and 17.

FLASK secret key:

Flask is using secret key for secure session management and cookie security.

In code: app.py line 13-17

Testing:

I have tested the program manually so that all the buttons and routes in UI work as they should. I have also tried to manually check different user inputs to try to crash the program. According to my manual tests the user input limits and requirements seem to be working as intended. Also, encryption seems to be working as it should and master password hashing is working.

I also run my python scripts with Bandit security vulnerability checker and here are the first results:

```
(.venv) PS C:\Yliopisto\Secureprogramming\password_manager> bandit -r application/database/
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.12.2
Run started:2024-05-05 16:28:43.886945

Test results:
        No issues identified.

Code scanned:
        Total lines of code: 93
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
Files skipped (0):
```

Picture 3: Python script security vulnerability testing under database directory.

Here I did run scanner for python files database_handler.py and password_encrypting.py and no issues were found. Let's run same test for app.py:

```
(.venv) PS C:\Yliopisto\Secureprogramming\password_manager> bandit -r application/app.py
[main]  INFO    profile include tests: None
[main]  INFO    profile exclude tests: None
[main]  INFO    cli include tests: None
[main]  INFO    cli exclude tests: None
[main]  INFO    running on Python 3.12.2
Run started:2024-05-05 16:31:59.337914

Test results:
        No issues identified.

Code scanned:
        Total lines of code: 203
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
        Total issues (by confidence):
                Undefined: 0
                Low: 0
                Medium: 0
                High: 0
Files skipped (0):
```

Picture 4: app.py vulnerability scan

Here scanner also did not find any vulnerabilities.

Ideas to improve program:

- Possibility to search passwords from program.
- In password generating possibility for the user to decide what characters can be used and how long password to generate.
- Password strength analyzer.
- Browser extension to automatically fill password to websites so don't need to copy every time from program.
- Logs for user activities.