**GROUP 12 PYTHON ADVANCED PROJECT – A PASSWORD VAULT**

|  |  |
| --- | --- |
| GROUP MEMBERS | ROLES |
| Chorio-Philip Riyeh | Leader and Coordinator |
| Mathias Aboh Job | Docs/Presenter |
| Zainab Musa | Developer |
| Ayeoyenikan Oluwatimileyin | Docs/Presenter |
| Francis Chidiebele Dibigbo | Docs/Presenter |
| Abdulrahman Yahaya | Docs/Presenter |
| Khalil Abubakar Mustapha | Developer |
| Paul Soja | Tester |
| Jatto Favour Ogeneomamuzo | Tester |
| Adamu Murtala | Docs/Presenter |

SHORT DESCRIPTION OF THE APPLICATION

The Password Vault is a locally encrypted password manager with a master password, add/get/list entries, and a copy-to-clipboard feature.

KEY FEATURES

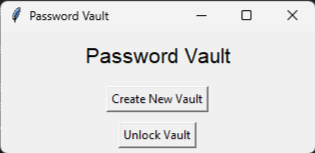
* Secure storage and Retrieval of entries using authenticated encryption
* Denial of access if the wrong master password is provided.
* A Password Generator

LIBRARIES USED

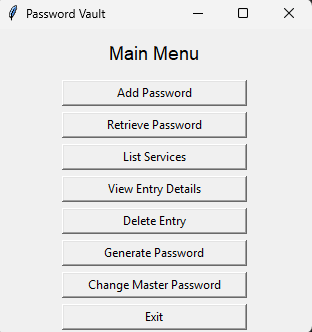
* os
* pytest
* sqlite3
* hashlib
* secrets
* base64
* pyperclip
* cryptography
* getpass
* tkinter

DESIGN(GUI)

* The Login Prompt



* The Main Menu



THE TESTS THAT WERE CARRIED OUT

1. The Vault Creation and Setup Test

def test\_vault\_setup():

    """Test vault creation and setup"""

    test\_db = setup\_db()

    vault = PasswordVault(test\_db)

    try:

        vault.initialize\_db()

        assert vault.setup\_vault("test\_password") is True

    finally:

        vault.close()

        if os.path.exists(test\_db):

            os.remove(test\_db)

1. The Vault Unlock Test

def test\_vault\_unlock():

    """Test unlocking the vault"""

    test\_db = setup\_db()

    vault = PasswordVault(test\_db)

    try:

        vault.initialize\_db()

        vault.setup\_vault("test\_password")

        assert vault.unlock\_vault("test\_password") is True

        assert vault.unlock\_vault("wrong\_password") is False

    finally:

        vault.close()

        if os.path.exists(test\_db):

            os.remove(test\_db)

1. The Adding and Receiving Password Test

def test\_add\_and\_get\_password():

    """Test adding and retrieving a password"""

    test\_db = setup\_db()

    vault = PasswordVault(test\_db)

    try:

        vault.initialize\_db()

        vault.setup\_vault("test\_password")

        vault.unlock\_vault("test\_password")

        assert vault.add\_password("test\_service", "test\_user", "test\_password") is True

        assert vault.get\_password("test\_service", "test\_user") == "test\_password"

        assert vault.get\_password("wrong\_service", "test\_user") is None

    finally:

        vault.close()

        if os.path.exists(test\_db):

            os.remove(test\_db)

1. The Update Password Test

def test\_update\_password():

    """Test updating an existing password"""

    test\_db = setup\_db()

    vault = PasswordVault(test\_db)

    try:

        vault.initialize\_db()

        vault.setup\_vault("test\_password")

        vault.unlock\_vault("test\_password")

        vault.add\_password("test\_service", "test\_user", "old\_password")

        assert vault.get\_password("test\_service", "test\_user") == "old\_password"

        # Update

        assert vault.add\_password("test\_service", "test\_user", "new\_password") is True

        assert vault.get\_password("test\_service", "test\_user") == "new\_password"

    finally:

        vault.close()

        if os.path.exists(test\_db):

            os.remove(test\_db)

CHALLENGES ENCOUNTERED

* We found it hard choosing the appropriate libraries needed to accomplish our project’s MVP and some of the Stretch goals but we solved the problem by making research.