A REPORT ON PYTHON SOFTWARE DEVELOPMENT PROJECT

NO. 6 - PASSWORD MANAGER WITH ENCRYPTION

BY

GROUP 19

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY,

COLLEGE OF NATURAL AND APPLIED SCIENCES,

SUBMITTED TO

AYUBA MUHAMMAD (SENIOR SOFTWARE/AI ENGINEER)

NEW HORIZONS - BELLS UNIVERSITY OF TECHNOLOGY, OTA, OGUN STATE,

NIGERIA

COURSE CODE: ICT 323

JANUARY 2026

MEMBERS OF GROUP 19

- **Ojo Ifeoluwa Wonders (2023/12197)**
- Oladokun Bolade James **(2023/12328)**
- Vera Edet Teyojesam **(2023/12387)**
- Abiola Taiwo Daniel **(2023/12669)**
- Jimoh Ifeoluwa Fathia **(2023/12423)**

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Background of the Study

Today, almost everyone uses the Internet for various purposes, from social media and banking to online classes. Because of this, we all have so many accounts that it has become really difficult to remember every single password. To make matters easier, most human beings use a simple password like their first name or use the same password for every website. This is a big problem due to the fact that if a hacker receives one password, they have access to various accounts, which could cause threats like identity theft or financial loss.

Keeping passwords steady is now a main part of cybersecurity. Previously, people used to write down passwords in notebooks or in a simple text document on their computer systems; those strategies aren't secure in any respect. That's why we need password managers – applications that can keep passwords securely and ensure users are choosing strong passwords that can be difficult to guess.

For this challenge, Python 3 was used as it is a very flexible language for creating desktop applications. Python has unique libraries that cope with encryption (cryptography, fernet) to maintain records secret and document control thoroughly. To make the application modern and easy to use, a proper GUI was developed using ttkbootstrap, which is an advanced version of Tkinter that provides professional themes. Furthermore, to provide advanced data visuals and analytics, Matplotlib and Seaborn were utilized for generating professional security charts and visualizations, while Pandas was used for efficient data structuring and management within the application.

CyberVault was created to solve the problem of poor password habits. It doesn't just store passwords; It uses encryption to lock them and has a system to check if a password is "strong" or "weak" before saving it. Creating this app helped us understand how to take security problems and turn them into a working software solution using Python.

1.2 Problem Statement

Everyone gets that cyberattacks are a real problem, but most people just don’t have what they need to actually stay safe online. The truth is, the digital world right now makes it nearly impossible for anyone to keep themselves secure, and there are three main reasons.

"Password fatigue" would be the first reason because one must balance the benefits against the potential threats posed by a password, resulting in setting a weak password that can be easily guessed, such as "12345."

Secondly, reusing the password will pose a major threat to the individual's identity, as using a particular password may jeopardize the identity.

Third, the level of security provided by storing passwords would be almost zero.

Some cloud password managers can be used, but many people do not want to store any personal, confidential data on a third-party server. In this regard, there is a specific need that is met by a solution that includes not only protecting passwords, but also using real-time analysis for passwords to help break the cycle of weak passwords.

1.3 Aim & Objectives

Aim: Build *CyberVault*, a secure Python-based desktop app that helps users store and manage their passwords offline. It will feature strong encryption and include visual tools to help users understand and manage their data easily.

General Objective: To design and develop a fully functional Python desktop application with a GUI, data processing, and visualization.

Specific Objectives:

1. To create an intuitive graphical user interface
2. To implement file handling and data persistence
3. To perform real-time calculations and data analysis
4. To generate interactive charts and reports
5. To package the application with proper error handling
6. To deploy on GitHub with complete documentation

1.4 Significance of the Project

The importance of this study is to expose how critical it is to have a steady way to manage digital identities in a global where cyber-attacks are becoming more common. Many users are at high risk due to the fact that they do not know the way to create strong passwords or where to keep them safely without exposing them. By developing CyberVault, this challenge gives a practical tool that solves those issues for the user.

Furthermore, this project is essential as it indicates that Python can be used to effectively solve real-world security issues. This proves that you don't constantly want complex, expensive cloud structures to stay safe; A nicely-designed local application can provide high level security and privacy.

This project was essential as it allowed us as computer science students to move beyond simple coding and discover ways to use advanced, professional libraries. Learning to put use Fernet for encryption, Pandas for data management, and Seaborn for analytics has given us experience that is valuable for future careers in software development and cybersecurity.

1.5 Scope and Limitations

Scope: This project involves developing a desktop-based password manager that lets users add, view, edit, and delete their account information. It also includes a security dashboard that visually displays password strength using clear, easy-to-read charts and graphs created with Matplotlib and Seaborn. The application suggests passwords that are tough to crack. Everything you store gets locked down with AES encryption and stays local on your device, which means nothing goes to the cloud.

Limitations:

Platform: For now, it's designed to run on a computer. There's no mobile version yet.

Recovery: Privacy comes first, so the app works fully offline Because the app runs entirely offline, there's no "forgot password" feature or cloud storage. If the master key is lost, all data stays locked and can't be retrieved.
Storage: CyberVault is intended for personal use on a single device.

Non-Responsive Interface:The app sticks to a fixed-size window, so dashboards and tables always look tidy and in place.

CHAPTER 2: LITERATURE REVIEW

2. 1 Review of Existing Similar Applications

This section reviews their operating models and key features of existing solutions for managing passwords. These programs are usually categorized depending on how they hold their data and depending on their distribution channels.

2.1.1 Cloud managers (e.g. 1Password, Bitwarden)

The primary objective of these managers is password storage in the cloud so that they can be accessed from any computer.

Key Functions: The key function here is the capability of these managers to work on various platforms, so your passwords remain accessible to you no matter where you are. Thus, there are browser extensions, mobile apps, and desktop clients that all access the same central database.

Security Model: Typically, a "Zero Knowledge" architecture is employed, meaning the service provider does not possess the master key, although the encrypted data is stored on their servers.

2. 1. 2 Integrated Browser Managers (e. g., Google Password Manager, iCloud Keychain)
These are directly integrated into the web browser's source code.

Core Functions: The main feature is their "deep integration" with web forms. Heuristic analysis is used to detect login fields and thus facilitate one-click "Autofill" capabilities.

Storage Logic: Most of the time, credentials are associated with a main identity provider account (like a Google or Apple ID) and are synchronized via the browser's internal profile sync settings.

2. 1. 3 Independent Local Clients (e. g., KeePass, VaultMan)
Local clients refer to standalone executable files that do not need an internet connection to work.

Core Functions: The primary function of local clients is to generate a single, encrypted database file that is saved directly on the user's hard drive. These systems rely heavily on manual input and "Auto Type" sequences, with no emphasis on cloud synchronization.

Interface Style: Most of these applications rely on an old school design element by making use of typical Windows API forms and tree views to control the data associated with accounts.

2.2 Strengths and Weaknesses of Existing Projects

Tool Category	Primary Strength	Primary Weakness	Privacy Level
Cloud-Based	Universal access and auto sync	Target for server-side breaches	Low (Third party)
Browser-Based	Seamless autofill integration	Single point of failure through main account	Medium (Corporate)
Legacy Desktop	Total local data control	Clunky UI and lack of analytics	High (User Owned)
CyberVault (Proposed)	Local Encryption and Analytics	No cloud-sync by design	Maximum (Offline)

2. 3 Python Libraries Used in Similar Projects

To deliver a feature-rich and safe password manager, software engineers generally choose a defined set of Python libraries. The libraries take care of all, from light GUI to complex mathematics and data security.

2. 3. 1 Encryption and Security Libraries

- Cryptography (Fernet): The library is the most popular choice when it comes to "Zero-Knowledge" encryption. It comes with the Fernet implementation of AES (Advanced Encryption Standard). The reason for the preference is that the library is designed to be "footgun-free, " so it is very difficult for a developer to unintentionally create an insecure system. Real World Example: The logic implemented via the Fern et recipe is similar to that of the 256-bit AES encryption utilized by 1Password as the core security building block.

- Hashlib: A standard Python library that can be used to create secure message digests. Most alike projects use it for hashing the "Master Password" with the help of algorithms, so that passwords in plain text are not stored at all Real World Example: The PBKDF2 key derivation algorithm, along with its HMAC and SHA-256 algorithms is utilized by Bitwarden in order to encrypt keys from the master password. The raw password does not reach their server.

- Secrets: The secrets module, in contrast to the standard random library (which is predictable), is used for generating cryptographically strong random numbers in very high-security systems.

2. 3. 2 Graphical User Interface Frameworks

- Tkinter: Python's default, built-in library for GUIs. Although efficient and minimal, the general remark given about this library's interface is that it is of the older style and thus appears somewhat out-of-date on newer operating systems.

- CustomTkinter & Ttkbootstrap: These are contemporary wrappers for Tkinter. They permit developers to generate modern and trendy interfaces with the help of pre-built themes (like Dark Mode) without the difficulty of bigger frameworks such as PyQt.

- PyQt / PySide: These are robust, heavy, industrial-grade frameworks designed for the creation of complex desktop applications. They are, however, less straightforward to learn and result in a larger file size as compared to Tkinter, based apps.

2. 3. 3 Data Management and Visualization

- Pandas: Often, security tools that handle large sets of credential data are designed to work with pandas. This library provides functionalities for easy filtering, sorting, and searching within the password vault.

- Matplotlib & Seaborn: These two libraries are considered the best for data visualization. In today's security applications, they are employed for converting raw password data into visual Security Dashboards, which enable the fast recognition of weak or repeated passwords through graphical representation and heatmaps.

- JSON / SQLite: These are two different methods that serve the purpose of "Data Persistence. " JSON is chosen for lightweight local storage, and SQLite is employed in projects with more complex relational databases. For example, KeePass works with a local database file (similar to JSON storage) to make sure that the user has full control over their data without requiring an internet connection.

2. 4 Gap Your Project Fills

From the review of current apps, a research and functional void became pretty obvious. The existing password managers are many, but they, in general, trade privacy, ease of use, and visual insight against each other. To address these issues, CyberVault comes with the following functionalities:

Cloud Dependence Gaps: The applications that are mostly in use, such as 1Password, require internet access to work properly. What it means is that the data is stored online on the servers of the applications and are subject to hacking attacks, as happened to LastPass. CyberVault fills the gap as it works entirely offline.

The Problem with Old Desktop Apps: Tools like KeePass are offline and safe, but are very old and difficult to use design is well-known. They do not provide the user with any feedback regarding their security. CyberVault closes this gap by utilizing Python's Matplotlib and Seaborn

to generate a dashboard. Rather than just a list of passwords, it displays charts that assist the user in identifying weak or reused passwords.

The Privacy Gap: A large number of individuals are hesitant to entrust major companies like Google or Apple with all their passwords. In contrast, since CyberVault is a personal project with a straightforward structure, users are fully aware of the fact that their data is right there with a local JSON file on their own hard drive, and not in some unknown database owned by a big corporation.

Simplicity and Ease of Use: Some professional software programs are packed with too many menus and settings, which a student or worker would not know or need. CyberVault has been created to be not only simple but also straight to the point. It discards the confusing menus of old software and concentrates on the functionalities that people use most: saving, generating, and checking the strength of passwords.

CHAPTER 3: METHODOLOGY

3.1 System Architecture (Block Diagram)

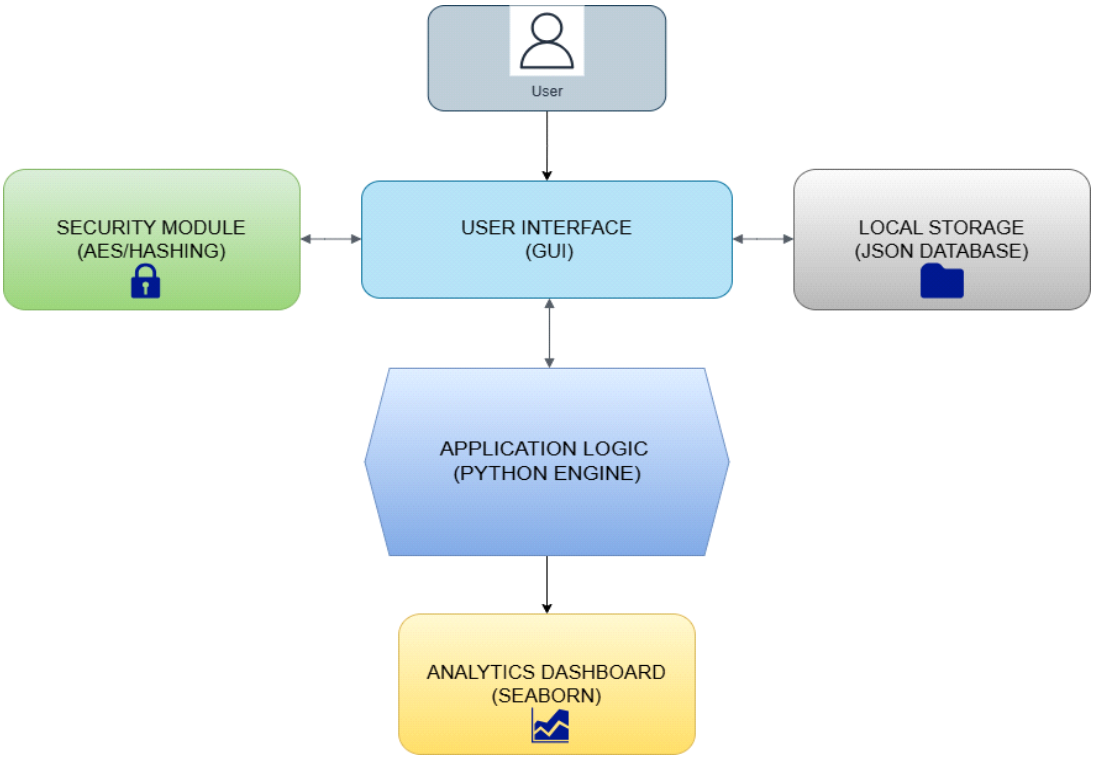


Figure 3.1: CyberVault System Architecture - A Modular Offline Design

3. 1. 1 Description of System Components

The CyberVault system architecture consists of five main parts that interact with each other to ensure that the user's data is safe yet simple to manage:

- The User: It is the individual who uses the software. The whole operation starts with the user giving input, such as their master password or the login details they want to save.
- User Interface (GUI): It is the "front end" of the app created with ttkbootstrap. It receives the clicks and typing from the user and, after that, communicates them to the "brain" of the app.
- Application Logic (The Brain): It is the central Python engine. It determines the next step. For instance, when you click "Save, " the logic instructs the Security Module to first encrypt the text and then the Storage Module to write it to the file.

- Security Module: It is the part that makes the whole operation safe the most. The above module uses AES-256 for encryption and SHA-256 as its hashing algorithm. The primary purpose of the module is to ensure the encryption of the entire password set that is stored in plain form to unreadable texts to avert guessing of the current password.

- Local Storage (JSON): CyberVault chooses a straightforward local JSON file instead of a complicated online database to save all the data. The significant factor to take note of here is to ensure that the data is being recorded on the personal computer, not on the internet.

- Analytics Dashboard: The component is equipped with Seaborn to fetch the data from the vault and convert it into visual representations. It supports the user in understanding, with a quick look, the distribution of their passwords categorized as "Weak, " "Medium, " or "Strong. "

3.2 Tools and Libraries

3.2.1 Development Tools

- PyCharm (v2025. 2. 2): The IDE used for writing and debugging scripts.

- Python (v3. 13, 64-bit): The programming language used for the whole project.

3. 2. 2 Graphical User Interface Libraries

- Tkinter (Built-in): The basic Python library utilized for creating the windows and the pop-up dialogs (messagebox, simpledialog) in the application.

- Ttkbootstrap (v1.10.1): The app is styled by this library that brings to the table ready-to-use widgets and themes with a professional look.

3. 2. 3 Security and Encryption Libraries

- Cryptography (Fernet) (v46.0.3): It is a security library in charge of AES-256 encryption, thus user passwords are definitely made unreadable with ciphertext before being stored.

- Re (Regex) (Built-in): A module, which is used for Input Validation. It also makes sure that the master password and the saved information are in compliance with the security policy requirements (like length and type of characters).

3. 2. 4 Data Analytics Libraries

- Pandas (v2.2.3): The library that is responsible for reshaping the vault data into "DataFrames, " which is a format that allows easy searching, sorting, and organizing password records.

- JSON (Built-in): The library for Local Storage .The encrypted information is taken and stored as a small file on the computer.

3. 2. 5 Data Visualization Libraries

- Seaborn (v0.13.2): A great library with the main aim of creating the most beautiful visuals for the Analytics Dashboard.

- Matplotlib (v3.10.8): The tool that draws the visuals. In particular, FigureCanvasTkAgg is the element that links these visuals with the Tkinter window.

3. 2. 6 Utility Libraries

- Pyperclip (v1.9.0): This is a tool that enables the user to simply click a button and have the password copied straight to the system clipboard for convenient pasting.

- Random & String (Built-in): These were employed to develop the Password Generator feature that can fashion robust, hard, to, predict passwords.

3.3 GUI Design (Wireframe and Interface Layout)

The CyberVault interface was built around a minimalist theme. The UI of the default state for new users is decluttered and shows only the necessary information, thus ensuring both security and user friendliness.

3.3.1 Authentication Screen (Master Password)

Users have to access their account using the master password screen that is right before them.

Target: The objective of this screen is to confirm that authorized personnel only are able to access the local DB.

Design: The design consists of the CyberVault brand logo on the top of the master password page, a password entry box that masks the entries, and an "Unlock Vault" button.

Functioning: This page is like a wall, and in case the Master Password is not correct, then the main dashboard will remain closed.

3. 3. 2 Architectural Modules (Wireframe Zones)

The GUI of the implementation is divided into four functional zones:

- Top Navigation Bar: The top navigation bar has the "Hamburger" menu for toggling the sidebar and a universal Search Bar for instantly searching through the saved accounts.

- Status Summary Area: The area just below the navigation bar in the main frame shows numbers for the Total Accounts, Weak Passwords, and Strong Passwords. The numbers are live and thus constantly changing.

- Dynamic Content Area: The Empty State displays a "Welcome" "Create Your First Account" prominently graphic and call to action (CTA) while the Populated State displays the accounts list (Website and Username only), thus the sensitive data is not exposed to "shoulder surfing. "

- Collapsible Navigation Sidebar: A menu on the right-hand side with the main application modules: Accounts (Add, View, Delete), Analytics, Settings, and About

3. 3. 3 Functional Modules

Account Management (Add/Modify): It causes a modal pop-up window. The inputs are validated with Regex, there is a strong password toggle, and there is a live Password Strength Meter that is based on the policies set in the Settings module.

Security View: To visually see the saved account details, the user must select an account and click "View Account" only. Therefore, privacy is improved.

Analytics Module: The main content area is replaced with a Seaborn Bar Chart, which visually shows the vault in Weak, Medium, and Strong segments.

Settings Module: The UI theme (Light/Dark) and functional aspects (Password Policy complexity) can be changed.

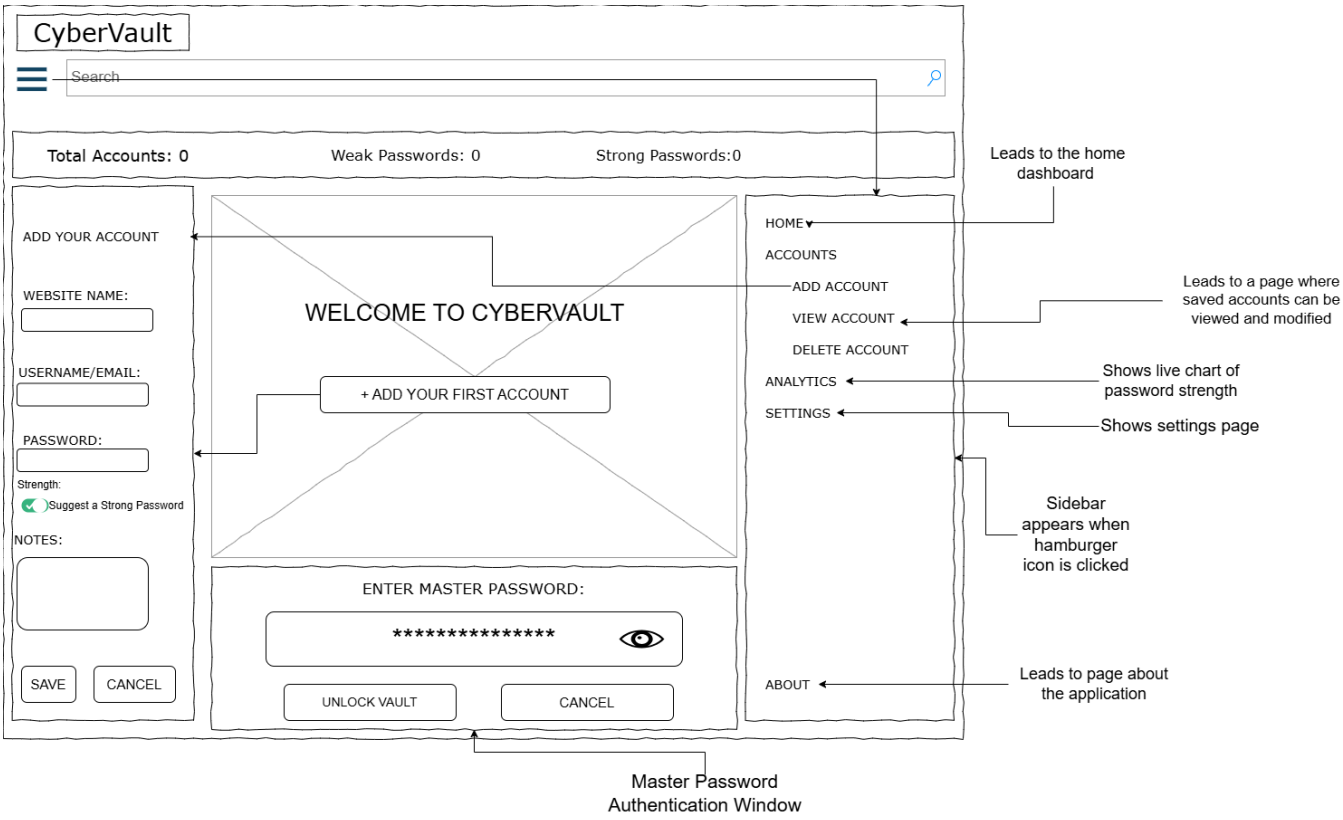


Figure 3.3: Wireframe of CyberVault Dashboard

3. 4 Algorithm and Flowchart of Core Features

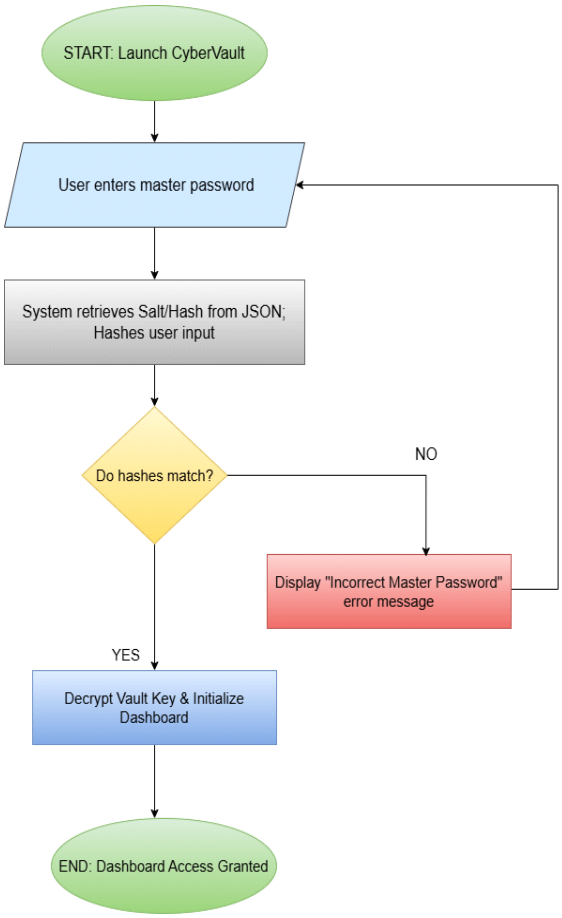
These are the procedures associated with the two most important features of CyberVault, which consist of the User Authentication and Secure Account Creation processes.

3. 4. 1 Master Password Authentication

A. Algorithm

1. Initialize: The program opens the Master Password input window (Modal) upon launch.
2. Input: The user inputs the Master Password.
3. Key derivation: First, the system retrieves the salt from storage. Then, it runs your password through a key derivation function to create a hash.
4. Comparison: Now, the system takes that hash and checks it against the one saved in the local config file.
5. Decision: The system decrypts the vault and displays the dashboard if the generated hash and the stored hash match.
   An "Incorrect Master Password" error is displayed if the hash does not match.
6. Termination: The system stays locked, and the dashboard is unavailable if this fails despite multiple attempts.

B. Flowchart



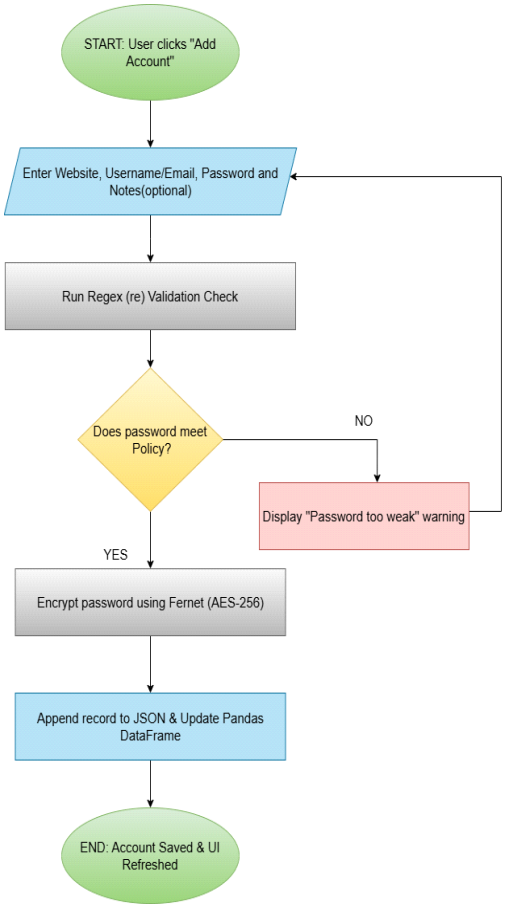
3. 4. 2 Secure Account Creation &Password Policy Validation

This functionality uses the Re (regex) library for policy enforcement and the Cryptography library for data security.

A. Algorithm:

1. Trigger: By clicking the "Add Account" button, the login form is presented.
2. Data Entry: User fills in website name, username/email, password, and notes (optional).
3. Power Meter: The GUI of the real-time view indicator is changed to correspond with the result of the policy check saved in the settings.
4. Encryption: Fernet (AES-256) algorithm encrypts the plain text password with the help of the obtained vault key.
5. Storage: The JSON database has been updated with encrypted records and refreshed Pandas Data Frames.

B. Flowchart:



3.5 File structure and database

The CyberVault system is structured around a modular layout, which shows the clear separation of security, logic, user interface components, and data storage.

3. 5. 1 Project Directory Structure

The project is organized in the CyberVault root directory. File analysis is as follows:

data/: The main storage location.

- vault.enc: An encrypted database file that stores all account credentials.
- settings.json: A configuration file that saves user preferences like themes and password policies.

Logic Module:

- auth.py: Handles authentication of the master password and session management.
- crypto_utils. py: Contains Fernet encryption and decryption methods.
- vault. py: Manages backend CRUD operations (create, read, update, delete).
- analytics. py: Prepares data for bar charts using Seaborn/Matplotlib.

UI Module:

- gui_dashboard. py: Primary application interface.
- main.py: Runs gui_dashboard.py

Documentation:

- README.md: Overview of the project and instructions for the setup.
- requirements. txt: External libraries list (e. g., ttkbootstrap, cryptography, pandas).

3.6 Development Workflow

CyberVault's codebase is secured by implementing version control with Git. Therefore, they recorded every change to the cryptographic logic and UI, resulting in a clean, professional development history.

3.6.1 Git Implementation in PyCharm

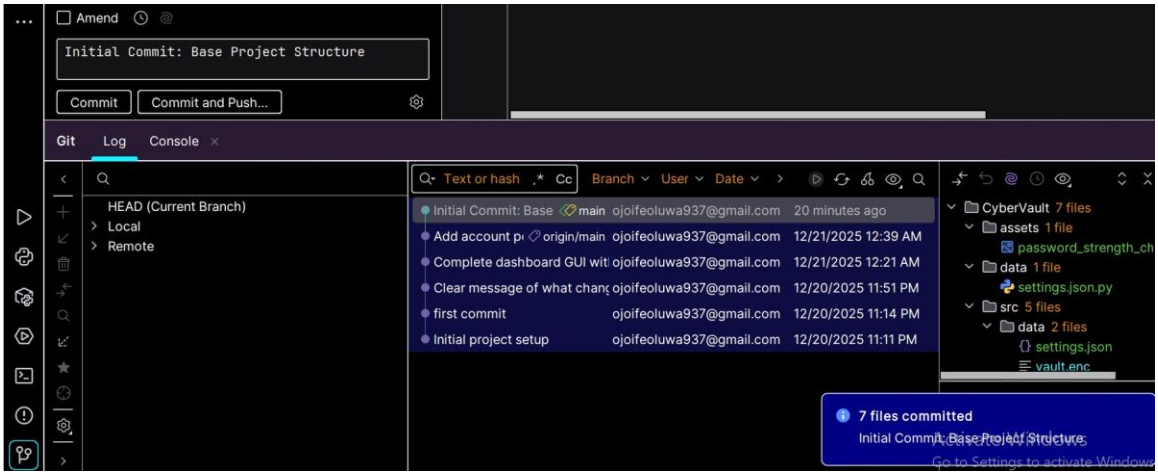
The source code was converted to the Git repository directly in the PyCharm IDE, thus allowing the state of the files to be followed at once through color coding: Red (unversioned), which means the files were in their first state before being included in the repository and Gray

(Committed) which is the current state of the project, indicating that all code has been properly saved to the local repository.

3.6.2 Branching Strategy

Main branch: This contained the stable, production-ready version of the vault.

Feature Branches: These branches are developed for high-risk modules, such as crypto_utils. py and auth.py. The code used here did not cause the main application to break during the build phase.



CHAPTER 4: IMPLEMENTATION AND RESULTS

4.1 Screenshots of all major windows/pages

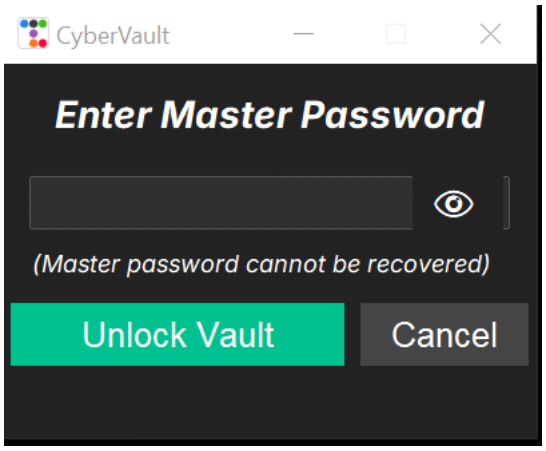


Figure 4.1.1: The initial authentication window in dark theme. The system remains locked until the Master Password hash is verified.

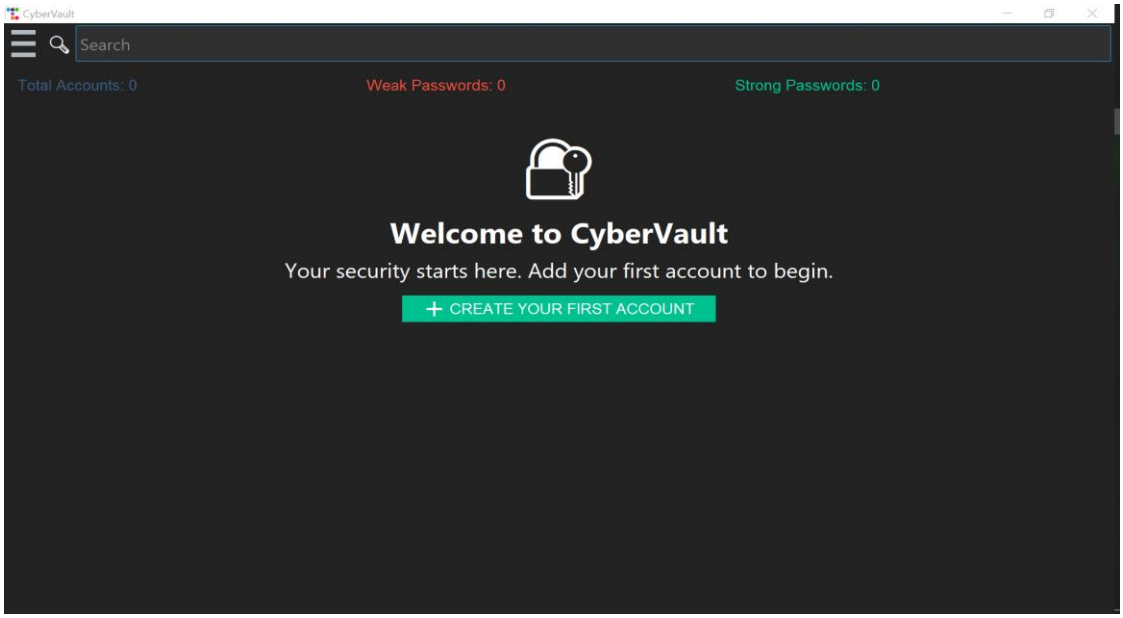


Figure 4.1.2: The empty-state dashboard in dark theme provides clear 'Call-to-Action' prompts for the first-time user.

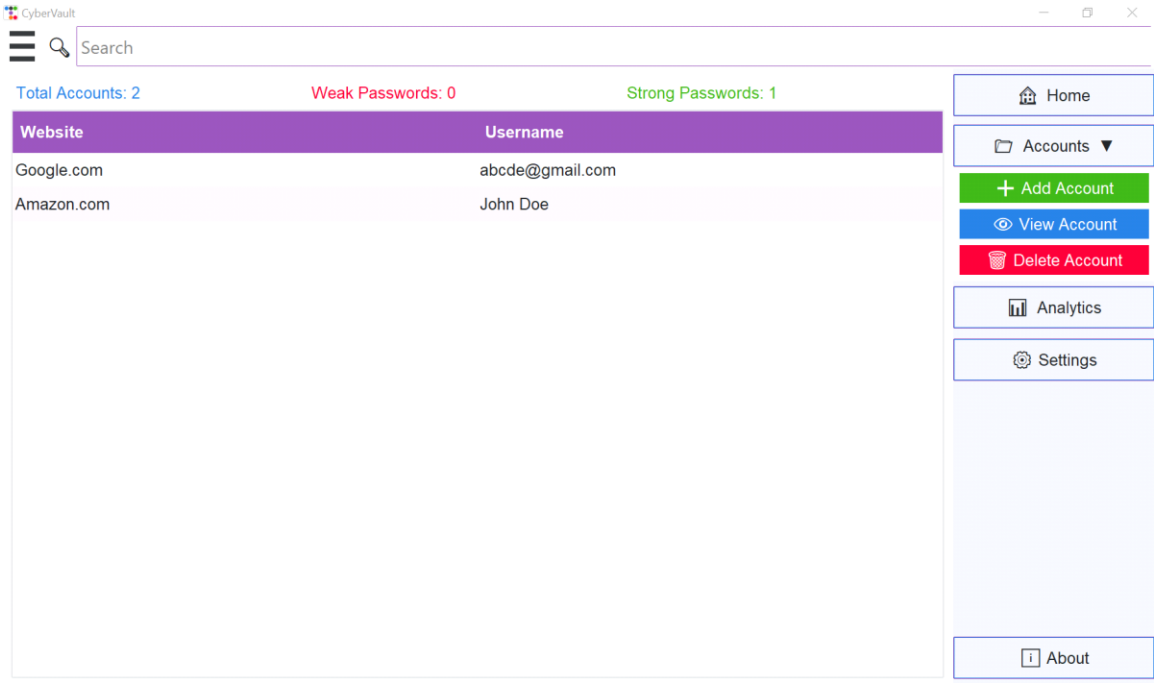


Figure 4.1.3: The Main Dashboard in light theme. Users can view, search, and manage their encrypted vault here.

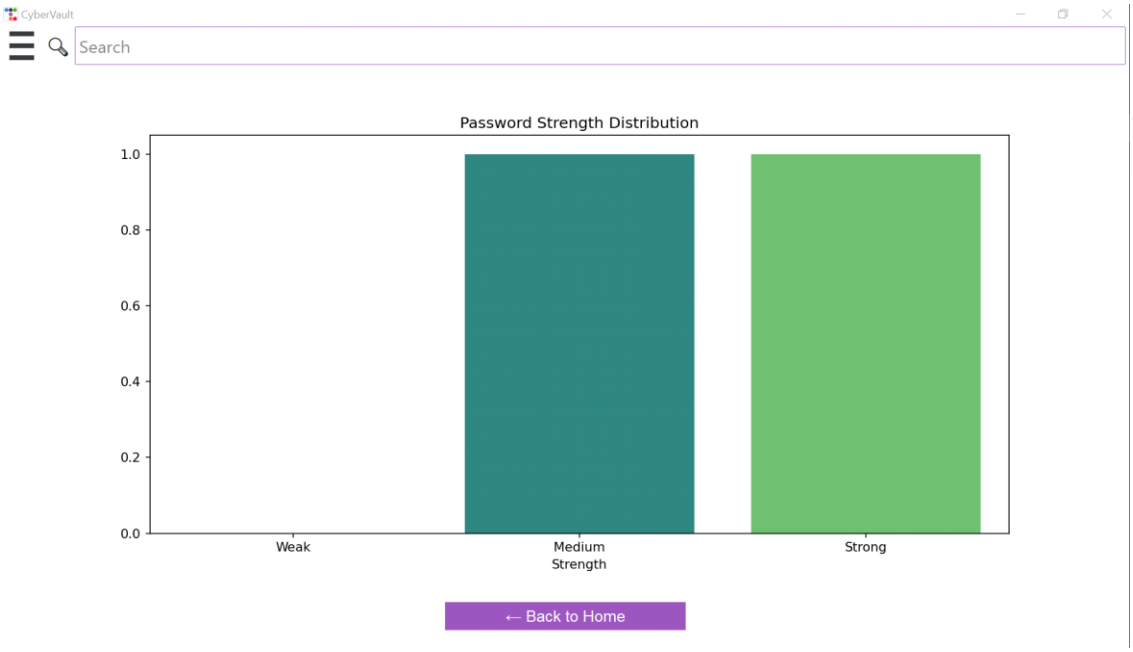


Figure 4.1.4: Data Analytics display showing the distribution of password strengths across the vault.

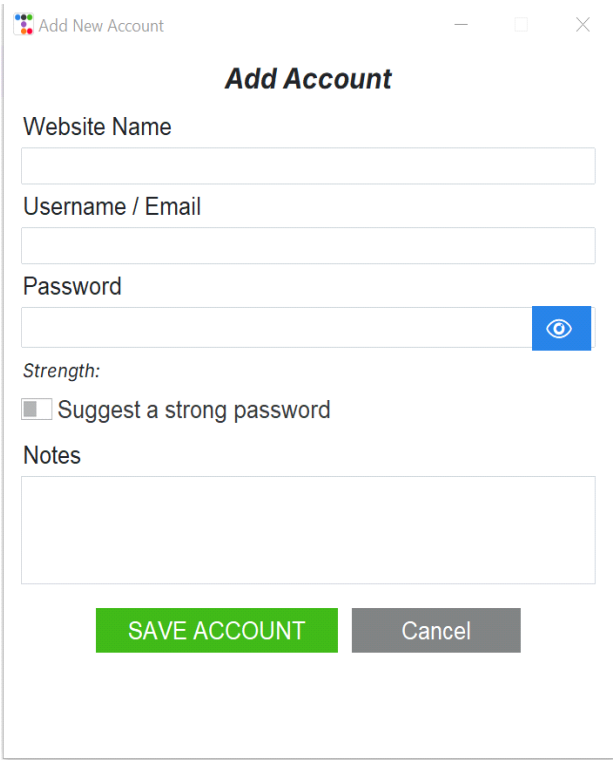


Figure 4.1.4: The Add Account interface in light theme. This window handles the real-time validation for password strength.

4. 2 Sample Input and Output Examples

This section shows the flow of data through CyberVault by following one piece of information from the interface, through validation and encryption stages, and finally, to its representation on the disk.

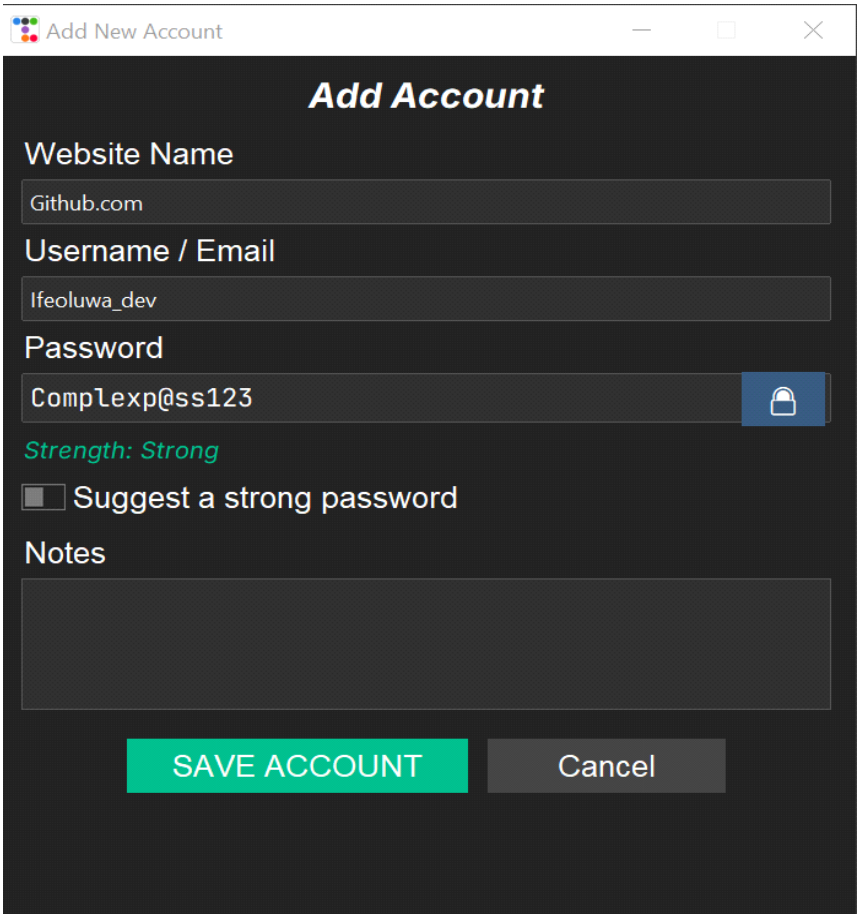
4. 2. 1 Input: User Entry through GUI

The user opens the Add Account window and provides this sample data:

Website: Github.com

Username: Ifeoluwa_dev

Password: ComplexP@ss123



A screenshot of the "Add Account" pop-up window filled with the data above.

Logic: The system implements a scoring function located in analytics. py (called by the GUI) to carry out a Regular Expression (Regex) check. By looking for uppercase letters, lowercase letters, numbers, and special characters, the system generates a score of 5/5, labeling this input as "Strong. "

4. 2. 2 Internal Processing: Hashing and Encryption
Before writing the record to the local database, the software performs two essential silent operations:

JSON Structuring: The separate input strings are converted into a structured Python dictionary.

Fernet Encryption: The crypto_utils.py module employs the AES-256 standard to encrypt the entire dictionary with the Master Key. This operation guarantees that even if the file is taken, the information is of no use to anyone without the user's master password.

4. 2. 3 Output: Encrypted Storage

The output was produced in src/data/vault.enc, which is essentially a very long random string. Here is the actual encrypted content generated by the system after the account was saved:

"gAAAAABpVlPFWPIiQS179Oaba2Y_obNWf7mvKLbk92mfJMeEgUD8pZnHRHoDzuec-SjcwLZqB4mE52EUWwUxUOtkCuTdBYD3-Q=="

The account name "GitHub.com" and the password are no longer visible. The effect of the Fernet algorithm is what causes the whole file content to be changed drastically every time a new account is added; pattern analysis is not possible for an unauthorized party.

4. 3 Key Code Snippets and Explanations

The core logic of the CyberVault system is shown here. This includes the cryptographic security, data persistence, and user interface design parts.

4. 3. 1 Key Derivation Logic (from crypto_utils. py)

In the case of encryption of data, the network must encrypt the user's plain-text Master Password into a strong key by the system.

crypto_utils.py

```python
from cryptography.fernet import Fernet
import os
import base64
import hashlib


def generate_key(master_password: str) -> bytes:  4 usages  ojoifeoluwa93
    """
    Derive encryption key from master password
    """
    hashed = hashlib.sha256(master_password.encode()).digest()
    return base64.urlsafe_b64encode(hashed)

```

Explanation: To create the 32-byte hash, we use the SHA-256 algorithm. After that, we encode the hash with Base64 to get the Fernet hash.

4. 3. 2 Secure Data Persistence (from vault. py)

This snippet demonstrates the process of "Data at Rest" protection, saving the user's accounts to a physical file safely.

vault.py

```python
import json
import os
from cryptography.fernet import Fernet

VAULT_FILE = "data/vault.enc"

def save_vault(data: dict, fernet: Fernet):  4 usages  ojoifeoluwa9
    """Encrypts and saves the vault data to disk."""
    encrypted = fernet.encrypt(json.dumps(data).encode())

    if not os.path.exists("data"):
        os.makedirs("data")

    with open(VAULT_FILE, "wb") as f:
        f.write(encrypted)
```

Explanation: The process takes three very strict steps:

- Serialization: Turning the Python dictionary into a JSON string.
- Encryption: Encrypting the data with the Fernet key created from the password, producing a ciphertext.
- Binary Storage: Saving the result as a binary file (. enc) so that it cannot be tampered with by normal text editors.

4. 3. 3 Password Strength Analytics (from analytics. py)

It's the "brain" of the app where it determines the strength of password security.

analytics.py

```
3   def password_strength(password):   ojoifeoluwa937@gmail.com <your.email@
8       # Length check
9       if len(password) >= 8:
10          score += 1
11      # Uppercase
12      if re.search( pattern: r"[A-Z]", password):
13          score += 1
14      # Lowercase
15      if re.search( pattern: r"[a-z]", password):
16          score += 1
17      # Numbers
18      if re.search( pattern: r"[0-9]", password):
19          score += 1
20      # Special characters
21      if re.search( pattern: r"[!@#$%^&*(),.?\":{}|<>]", password):
22          score += 1
23
24      # Message based on score
25      if score <= 2:
26          msg = "Weak"
27      elif score == 3 or score == 4:
28          msg = "Medium"
29      else:
30          msg = "Strong"
31
32      return score, msg
```

Explanation:

The function uses regex to test the password according to the complexity rules. The result obtained is a score, which is further utilized to create the graphs on the dashboard.

4. 3. 4 Modern UI Implementation (from gui_dashboard. py)
The interface is designed to be user-friendly by utilizing the ttkbootstrap framework.

gui_dashboard.py

```
130  # 3. Translate the saved preference into a valid theme name
131  saved_theme_pref = saved_data.get("theme", "darkly")
132
133  if saved_theme_pref == "light":
134      actual_theme = "cosmo"
135  elif saved_theme_pref == "dark":
136      actual_theme = "darkly"
137  else:
138      # If it's already "cosmo" or "darkly", use it as is
139      actual_theme = saved_theme_pref
140
141  # 4. Start the window with the VALID theme name
142  root = tb.Window(themename=actual_theme)
143  root.title("CyberVault - Secure Password Manager")
144  root.geometry("950x600")
145
146  style = tb.Style(theme=actual_theme)
```

Explanation: This logic serves as a bridge between settings that are easy for the user to understand and the technical requirements of the ttkbootstrap library. The application guarantees a consistent appearance. This fragment is evidence that the program communicates with a settings configuration file, thus it can keep track of user selections from one session to another, which is a feature of software of a professional standard.

4. 4 Performance and Resource Evaluation

The section examines CyberVault's resource management and user interface performance through live data from Windows Task Manager.

4. 4. 1 Resource Monitoring
As an app, it is built to be minimal and completely offline (0 Mbps network usage).

Operation State	CPU Usage	Memory (RAM)	Network
Initial Startup	0.0%	84.5 MB	0 Mbps
Login Screen	0.0%	85.1 MB	0 Mbps
Adding Account	0.1%	85.9 MB	0 Mbps
Post-Entry Idle	0.0%	85.9 MB	0 Mbps

4.4.2 Response Times
App Boot: 5-10 Seconds (Time to load Python and the themes).

Vault Access: 1 Second (Time to hash the password and decrypt the file).

4.5 User Testing

The usability of CyberVault was then tested by recruiting five testers to conduct basic tasks within the interface and commenting on usability.

4.5.1 Tester Profiles
The group represents a section of typical users, ranging from basic to advanced technical skills.

Tester ID	Profile	Technical Level
U1	Home User	Basic
U2	Home User	Basic
U3	Student	Intermediate
U4	Student	Intermediate
U5	Developer	Advanced

4.5.2 Test Tasks & Results
Testers were asked to login and interact with the application

Tester	User Feedback
U1	"Very interesting and secure; perfect for users with bad memory and “Suggest Strong Password” button makes the creation of passwords effortless."
U2	“The layout is great, and the transition from the authentication process to the dashboard is extremely fast”
U3	"The 'View Account' process feels a bit long, though the feature works perfectly.
U4	"The theme switch is nice, and the app is very easy to navigate."
U5	"The interface is minimalistic and simple; it gets the job done without being clunky."

CHAPTER 5: CONCLUSIONS AND RECOMMENDATIONS

5. 1 Summary of Achievements

The key points of advancement are:

- Data Security: The security of user data is ensured by the implementation of AES for encryption and SHA for key derivation, with the use of 256-bit encryption.
- Adaptive UI: Use of two themes (Cosmo and Darkly) that not only help it adapt to different lighting conditions but also cater to the user's preference.
- Visual Analytics: A security audit system that provides a real-time view of the security level of passwords held in a vault.
- Resource Efficiency: A compact application with a small memory footprint (~86MB) and no network dependency.

5.2 Conclusion

CyberVault is able to successfully accomplish the protection of passwords in a minimalist manner and to a high level of security. There is certainly room to improve the workflow of "View Account," but overall, CyberVault addresses all of the elements needed to successfully accomplish a safe and easy-to-use security solution. In particular, it is evident that the “Strong Password Generator and Theme Toggle” is one of the most appealing elements of the functionality of this app, as it addresses the issue of password fatigue and security practices.

5. 3 Challenges faced and lessons learned

- Importance of planning: I was so eager to get to work that I started coding without a proper design or prototype. This resulted in a disorganized codebase that ultimately made me decide to do a complete restart. I learned my lesson that architectural design cannot be compromised before coding.
- UI/UX Barriers: In the initial stage, the size of the analytics chart was such that there was no space for navigation buttons. I learned to use responsive layout management so that the "back to home" buttons are always accessible and functional.
- Color Theory and Approach: In the beginning, I was focused on the idea of using certain colors like purple, blue, or green. But then I changed my mind and moved towards an adaptive minimalist design, which not only looks good but is also usable by people of all age groups from kids to elderly.
- Feature Creep & Technical Debt: Attempting to add the "Delete" button to the Edit page, code stability issues kept occurring. I decided not to break the feature and instead took the initiative to stabilize the vault, which was a valuable lesson in software management.

5. 4 Future Improvements

Further updates are planned for CyberVault to make it even better:

- Categorization: Accounts could be grouped into Work, Social, Personal, and Business categories for easier account management.
- Advanced Analytics: Chart updates to show grouped data (e. g., "4 Weak Social Accounts","5 Strong Personal Accounts").
- Enhanced CRUD Operations: Reworking database logic to allow a safe "Delete Account" button in the edit interface.
- Cross-Platform Portability: Creating a mobile, responsive version of the app so users can access their vault anywhere

5. 5 Contribution and Usefulness

CyberVault builds on a local, open-source project aimed at the secure functioning of the online life of citizens. It comes at a time when the problem of cloud data breaches as well as cyber threats persists. It gives users control of their data.

The main strength of the tool is its simplicity: it acts as a bridge between advanced cryptography and everyday usability. CyberVault encrypts data with cutting-edge technology and makes sophisticated security simple and accessible to the average user by providing helpful feedback.

References

Cryptography.io (2025). Fernet (Symmetric Encryption) Documentation.https://cryptography.io/en/latest/fernet/

Python Software Foundation (2026). The Python Standard Library: 'hashlib' and 're' modules https://docs.python.org/3/

Schlobohm, S. (2025). ttkbootstrap: A modern extension for Tkinter.https://ttkbootstrap.readthedocs.io/

Python Software Foundation (2025). The secrets and hashlib module documentation.docs.python.org.

1Password Security Blog. Why do we use 256-bit AES encryption? 1password.com/blog.

Bitwarden Security Whitepaper. Key Derivation and Master Password Hashing (PBKDF2 SHA-256). Retrieved from bitwarden.com/help.

Ttkbootstrap Documentation. Theming and Modern GUI widgets for Python. ttkbootstrap.readthedocs.io.

NeuralNine (2023). Python Encryption & Decryption - Cryptography Tutorial. Youtube

Tech With Tim (2024). How to build a Password Manager in Python. Youtube

Appendices

Appendix A: Full Source Code

vault.py

```
import json
import os
from cryptography.fernet import Fernet

VAULT_FILE = "data/vault.enc"

def save_vault(data: dict, fernet: Fernet):
    """Encrypts and saves the vault data to disk."""
    encrypted = fernet.encrypt(json.dumps(data).encode())

    if not os.path.exists("data"):
        os.makedirs("data")

    with open(VAULT_FILE, "wb") as f:
        f.write(encrypted)

def load_vault(fernet: Fernet) -> dict:
    """Loads and decrypts the vault data from disk."""
    if not os.path.exists(VAULT_FILE):
        return {}

    with open(VAULT_FILE, "rb") as f:
        encrypted = f.read()

    decrypted = fernet.decrypt(encrypted)
    return json.loads(decrypted.decode())
```

analytics.py

```
import re

def password_strength(password):
    """
    Returns a score and message for password strength
    """
    score = 0
    # Length check
    if len(password) >= 8:
        score += 1
    # Uppercase
```

```
    if re.search(r" [A-Z]", password):
        score += 1
    # Lowercase
    if re.search(r" [a-z]", password):
        score += 1
    # Numbers
    if re.search(r" [0-9]", password):

score += 1
    # Special characters
    if re.search(r" [!@#$%^&*(),.?\":{}|<>]", password):
        score += 1

    # Message based on score
    if score <= 2:
        msg = "Weak"
    elif score == 3 or score == 4:
        msg = "Medium"
    else:
        msg = "Strong"

    return score, msg
```

Due to the length of the main Python file, the full version is available at the GitHub link provided in Appendix D.

Appendix B: requirements.txt

```
cffi==2.0.0
contourpy==1.3.3
cryptography==46.0.3
cycler==0.12.1
fonttools==4.61.1
kiwisolver==1.4.9
matplotlib==3.10.8
numpy==2.4.0
packaging==25.0
pandas==2.3.3
pillow==12.0.0
pycparser==2.23
pyparsing==3.2.5
python-dateutil==2.9.0.post0
pytz==2025.2
seaborn==0.13.2
six==1.17.0
tzdata==2025.3
```

Appendix C: User Manual

1. First Run: Launch main.py and set your Master Password.

2. Managing Accounts: Click "Add Account" to store information.

3. Security Check: View the Dashboard chart to see your password health.

4. Settings: Use the settings menu to toggle between "Darkly" and "Cosmo" modes. Password policy preferences can be set here.

Appendix D: GitHub link
https://github.com/Ifeoluwa000/CyberVault