

National College of Ireland

BSc(Hons) in Computing

Cybersecurity

2021/2022

Dawid Marais

x18133282

x18133282@student.ncirl.ie

Vault Knox Password Manager

Technical Report



[Contents](#)

Introduction	3
Background	3
Aims	4
Technology	5
Structure	6
System	7
Functional Requirements	7
Use Case Diagram	7
Requirement 1 <Password Vault Registration/Login>	7
Description & Priority	7
Use Case: PVRLREQ1	8
Requirement 2 <Password Vault Modification>	10
Description & Priority	10
Use Case: PVMREQ2	10
Requirement 3 <Data Processing>	13
Description & Priority	13
Use Case: DPREQ3	13
Requirement 4<Password Vault Recovery>	15
Description & Priority	15
Use Case: PVRREQ4	15
Requirement 5 <Password Vault Settings>	18
Description & Priority	18
Use Case: PVSREQ5	18
Requirement 6 <Password Feedback>	20
Description & Priority	20
Use Case: PFREQ6	21
Requirement 7 <Inactivity Termination>	23
Description & Priority	23
Use Case: ITREQ7	23
Requirement 8 <Generate Password>	25
Description & Priority	25
Use Case: GPREQ8	25

Non-Functional Requirements	27
Security Requirements	27
User Requirements	28
Environmental Requirements	28
Usability Requirements	29
Availability Requirement	29
Performance/Response time requirement	29
Robustness requirement	30
Data Management requirement	30
System requirement	30
Reliability requirement	31
Maintainability requirement	31
Reusability requirement	31
Extendibility requirement	32
Design & Architecture	32
Class Diagram	33
Master Password Class	34
Password Entry Class	34
Settings Class	34
Bruteforce Class	35
IP Class	35
Generate Password Class	36
Inactivity Class	36
Implementation	37
Graphical User Interface (GUI)	50
Testing	55
Conclusions	67
Further Conclusions or Research	68
References	69
Appendices	71
Project Plan	71
Mid Point Project Plan	71

Final Project Plan	71
Project Proposal	71
Objectives	74
Background	74
State of the Art	75
Technical Approach	76
Technical Details	77
Special Resources Required	78
Project Plan	78
Testing	81
Bibliography	81
Reflective Journals	82
Reflective Journals Semester 1:	82
Reflective Journals Semester 2:	87

1.0 Introduction

1.1. Background

For my third-year internship, I interned as a cyber security analyst for a security company called BCC Edgescan. During my employment, we used a password manager called KeePass which served as our basic password manager. After my experience with this program, I saw the value that it contained but I felt like I could further improve on the functionality it provided.

We live in a digital age where all user information is backed up and stored behind a user profile for seemingly every application and service on the internet. With the incline of processing power, it has become much easier and faster for hackers to crack passwords located on the internet. For this reason, it is important that users have a unique password for each of their user accounts that utilise strong password protocols. However, it is impossible to remember a unique password for each service. This is where the value of a password manager lies.

With a password manager, it is no longer required to remember unique strong passwords for each of your accounts. A password manager conveniently stores all user passwords in a centralised location. Total access is granted to the password vault through a master password key. This is the only password the user is required to remember.

My project idea arose from the same issue I kept encountering while using web-based applications. I found myself having to reset passwords for services each time I would get signed out due to the sheer volume of accounts I continually use. The issue is that different platforms have different requirements for setting passwords. This makes it difficult to remember each unique password for a specific service and using a single password for multiple accounts leads to greater security threat exposure. My goal is to develop an application that combines the best features of already existing leading password managers and completely novel capabilities.

1.2. Aims

The aim of this project is to create a secure offline password manager that is unrivalled in the compilation of unique user functionalities, security and ease of use. This project aims to improve upon current password managers by providing greater flexibility of use, clarity of feedback and seamless maintenance without compromising on security. To facilitate all the before mentioned, the primary objectives further illustrate how Vault Knox Password Manager accomplishes the project vision.

Vault Knox Password Manager:

- utilizes a decentralized encrypted offline database to maintain and store passwords that utilize strong password protocols
- can generate new secure random passwords that satisfy industry-leading strong password protocols.
- facilitates maintenance of security through continuous report feedback, inactivity termination and update of outdated or compromised passwords
- can be utilized for all offline and online applications, wherever passwords are used for user profiles.
- maintains secure and fail-safe recovery of credentials using a two-factor authentication mechanism
- facilitates security using leading industry protocols for hashing, encryption, decryption and true random value generation.
- places the responsibility of the extent to which security of passwords is applied, on the user.
- prioritises functional simplicity and ease of use regardless of the users' technical proficiency.
- mitigates the threat exposure to the overall application system, by implementing secure application programming and principles.

1.3. Technology

Python is used as the primary development language. The main benefits of using Python is extreme streamlined effectivity and simplicity when writing code without compromising on performance or quality. Python contains many powerful native libraries that are necessary for the development of nearly all functionalities. Furthermore, importing external libraries using the pip package-management system is seamless and time-effective. Storing passwords online exposes them to any hacker willing to try to hack the database that contains all the data of users of the password manager. Sqlite3 is used to generate an offline database. Because a single user's password database is decentralized, a hacker is discouraged to try to hack a single offline user database. SQLite is a local file that only allows for only one writable connection at a time. All information saved to the database is encrypted in non-human readable byte format. Only the master user with their random and unique derived key is able to decrypt the data at run time.

Hashlib is a library that supports hashing. The hashing of passwords ensures the integrity of stored data and is facilitated using the most secure protocol, SHA-512. The long character length of the hash is best equipped to prevent hash collisions. By adding a salt value to the hashing algorithm we achieve protection against rainbow tables and greater complexity. The salt value uses the operating system UUID to generate a truly random value. It is necessary to encode user password input to UTF-8 using the collection of encoding protocols from the base64 library before applying the hashing algorithm.

Encryption is implemented using the fernet cryptography library. The library contains a key derivation function called pbkdf2_hmac. The key function generates a derived key that is unique to the master user. The same key is used to decrypt values that are encrypted. The function accepts 5 parameters to generate the function: hash name, password value, salt value, number of hash iterations and the key length. Users can copy generated and encrypted passwords to their clipboard without exposing the character values utilizing the pyperclip library.

The native python time library facilitates the functionality of scheduling and maintaining up to date passwords, as well as identifying when to terminate the application when remaining in a state of inactivity, for a prolonged period of time. The zxcbn package is used to identify compromised passwords and generate remediation feedback for vulnerability reports. The selenium package is utilised to create new browser sessions that open the service for outdated passwords that have been updated. This allows the user to quickly navigate to the respective service that requires remediation of an updated password value. The threading package handles all concurrent processes that run in the background supplementary to the main application process. Instances of threads include the inactivity check feature that runs in the background of the application.

The Urllib package makes get requests to web addresses to determine whether the user-supplied service name of a password entry generates a valid service application for the above-mentioned update feature. System credentials such as the IP address and geographical location is determined with the same package. These credentials are utilised when advertising the VPN affiliate service on the login screen. The Python Pillow package generates the canvas on which the loading screen of the application is generated and displayed to the user on the application startup. The functools library contains necessary modules partial and lambda to facilitate greater function flexibility to develop more complex classes and methods in Python that follow OOP principles.

The Tkinter framework is used to generate GUI components such as windows, frames and widgets for handling application styling and user control/input. Automated application testing is carried out using the native python unittest library. Unit testing will identify application bugs and unhandled exceptions to improve overall design robustness. NSIS is software that enables the final python script to be converted into a single application executable with an installer client that configures the application installation.

1.4. Structure

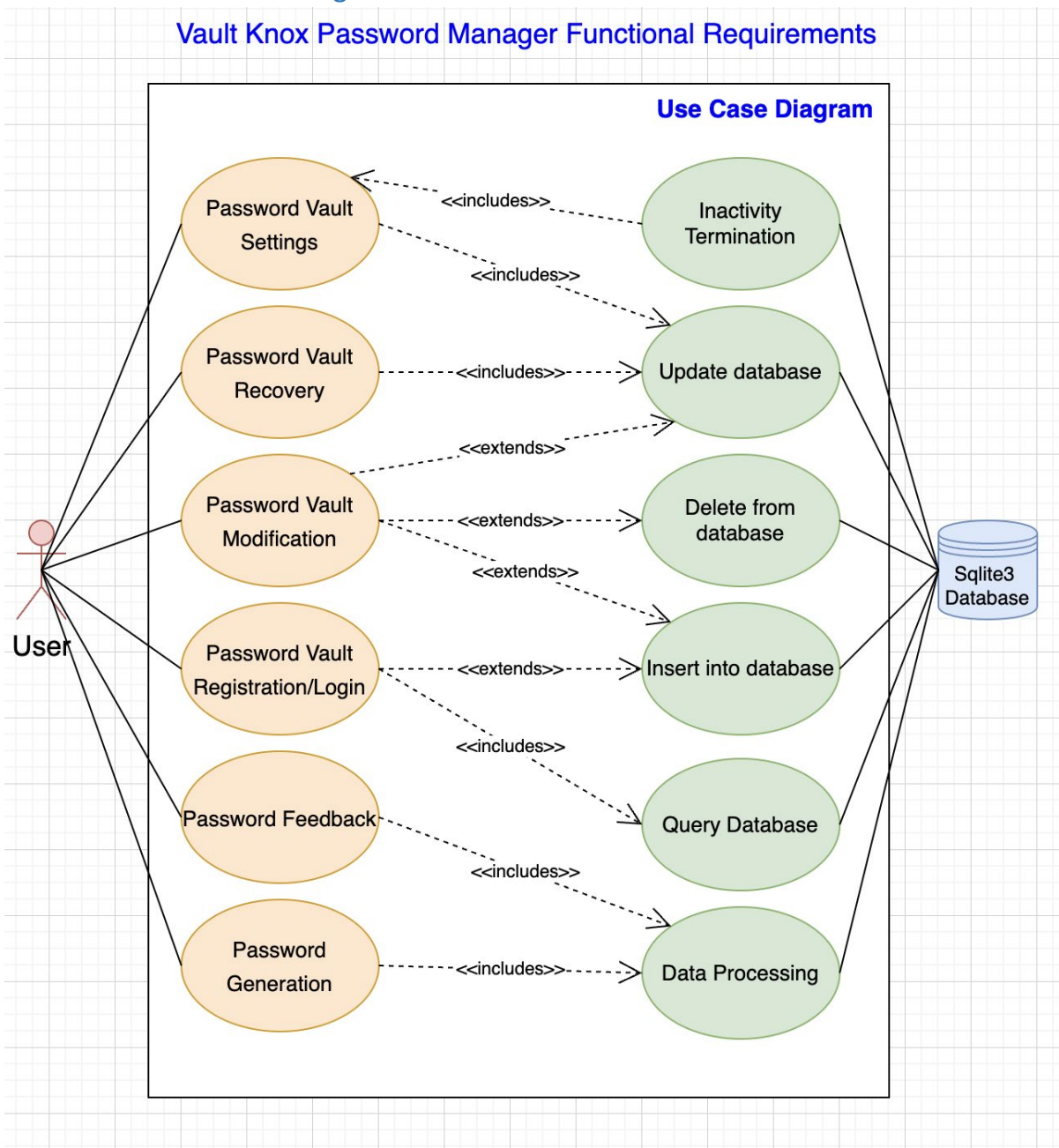
The purpose of this document is to clarify the requirements and technical details set out for the creation of the Vault Knox Password Manager application. The document is broken up into 6 main sections. These sections, in order, include the Introduction, System, Conclusions, Further Development or Research, References and Appendices. Each main heading contains descriptive subheadings that give further structure and illustration. The introduction aims to give a broad overview of the project as a whole and utilise summaries to effectively convey the bigger picture. The system section is the substance of the report and showcases the application and its requirements. It describes in detail the specific requirements and functionality that work together in the application. It is an in-depth investigation and analysis of the blueprint of the program. The system sections also includes the system design and architecture, implementation, graphical user interfaces and testing. The Implementation section examines relevant code snippets that describe the project's main algorithms, classes, and functions. The GUI section contains screenshots of key screens with explanations of what is displayed to the user. The Testing section contains evidence for all Unit, Integration, and End User testing that was performed. The project's advantages/disadvantages, strengths, and limitations are described in the Conclusions section. The section on Further Development and Research explores the project's future trajectory if given more time and resources. The appendices portion of the report comprises material that that supports the main body of the document. The conclusion The appendices section includes the project plans, project proposal and monthly journals.

2.0 System

2.1. Functional Requirements

Use Case Diagram

Vault Knox Password Manager Functional Requirements



2.1.1. Requirement 1 <Password Vault Registration/Login>

2.1.1.1. Description & Priority

The basis of this project is the essential requirement for the registration and subsequently login of a password vault. The login and registration requirement lends the characteristic of a secure, unique identity to the password vault. A single user has a master password that grants access to their password vault. If a user does not yet own a password vault they are prompted to register a new

password vault. Both the login and register methods use the generation of a master password. The master password credential requires strong security protocols that need to be satisfied to ensure integrity, confidentiality and authenticity for the generation of a new password vault.

Registration and login of a password vault is the top priority as it is used to facilitate all other requirements. [Priority 1]

2.1.1.2. Use Case: PVRLREQ1

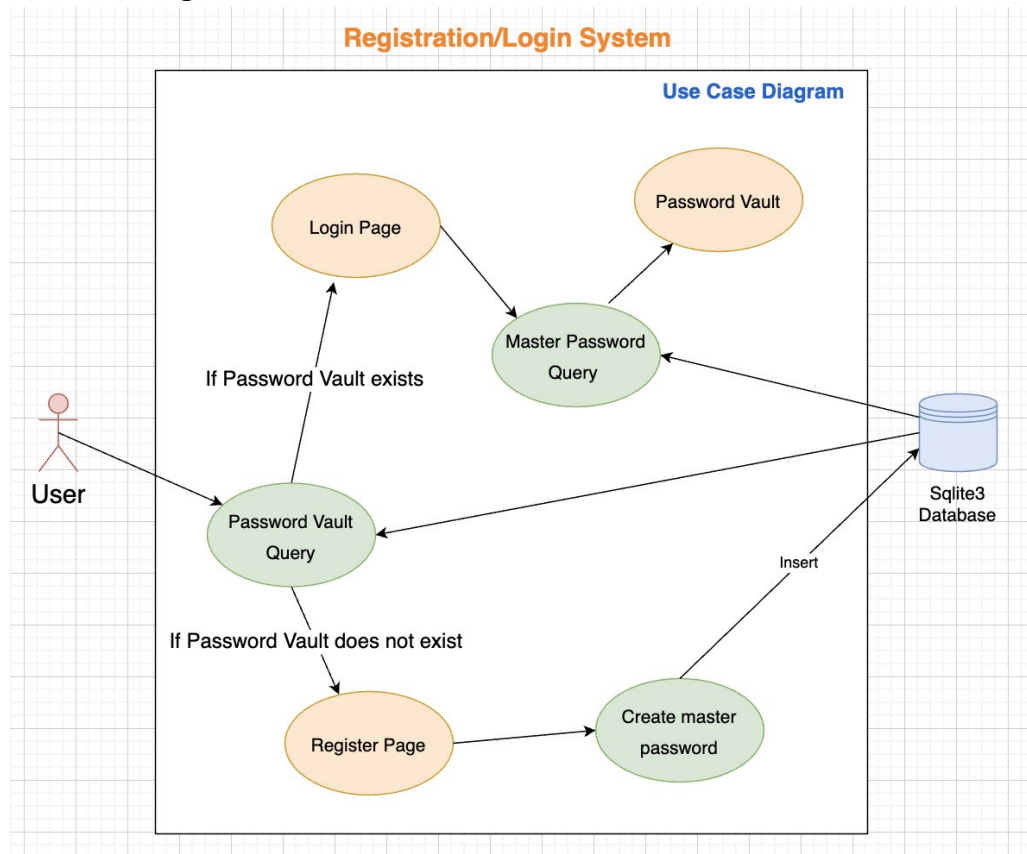
Scope

The scope of this use case is to demonstrate the interaction of the user and the login/register system of the password vault.

Description

The following use case describes a single user registering as well as logging in to their password vault and how user input data is processed by the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialisation mode, the user is first brought to the register/login screen to initiate access to their password vault.

Activation

The user begins on the login screen where they enter their master password to sign into their password vault. If the user does not own a password vault, they are directed to the registration screen to create a new master password.

Main flow

1. The system determines if a password vault exists (see A1)
2. The system directs the user to the password vault login screen if a password vault exists
3. The user enters their master password in the text field and presses the login button (see A2)
4. The system processes and checks the login request (see E1)
5. The system directs to the password vault screen if the login request is successful

Alternate flow

A1: <Password vault does not exist>

1. The system directs the user to the password vault register screen if a password vault does not exist
2. The user enters a new master password in the first text field (see A2)
3. The user repeats the entering of the same master password in the second text field and presses the login button (see A2)
4. The system processes and checks the register request (see E2)
5. The use case continues at position 2 of the main flow if the registration request is successful

A2: <User hides master password characters>

1. The user presses the hide password button
2. The system hides the password characters in the text field
3. The use case returns to the same position of the parent flow

Exceptional flow

E1: <Master password does not match>

1. The system compares the master password entered by the user with the master password in the database
2. If passwords do not match the system will output an error message stating, that they do not match.

3. The system clears the entered master password
4. The use case returns to the same position of the parent flow

E2: <Master password does not satisfy requirements>

1. The system checks whether the entered master password satisfies the list of password requirements
2. If the master password does not satisfy the password requirements the system will output an error message stating the unsatisfied requirement, eg. "master password length should be greater than 8 characters".
3. The use case returns to the same position as the parent flow.

Termination

Once the user logs in or registers successfully using their master password, the user is granted access to the application and is directed to their password vault.

Post condition

The system is in an idle wait state on the user password vault screen after user input is validated.

2.1.2. Requirement 2 <Password Vault Modification>

2.1.2.1. Description & Priority

An essential requirement for the project is modifying password entries stored in the password vault. A user has the ability to create a new password entry by clicking on the add password button and supplying the password service name, their username and the password. A user has the ability to remove an existing password entry by clicking on the delete button next to the respective password entry. A user has the ability to update an existing password entry by clicking on the update button next to the respective password entry and supplying the new password service name, username and password values.

The second highest priority is for a user to be able to modify password entries stored in their password vault. [Priority 2]

2.1.2.2. Use Case: PVMREQ2

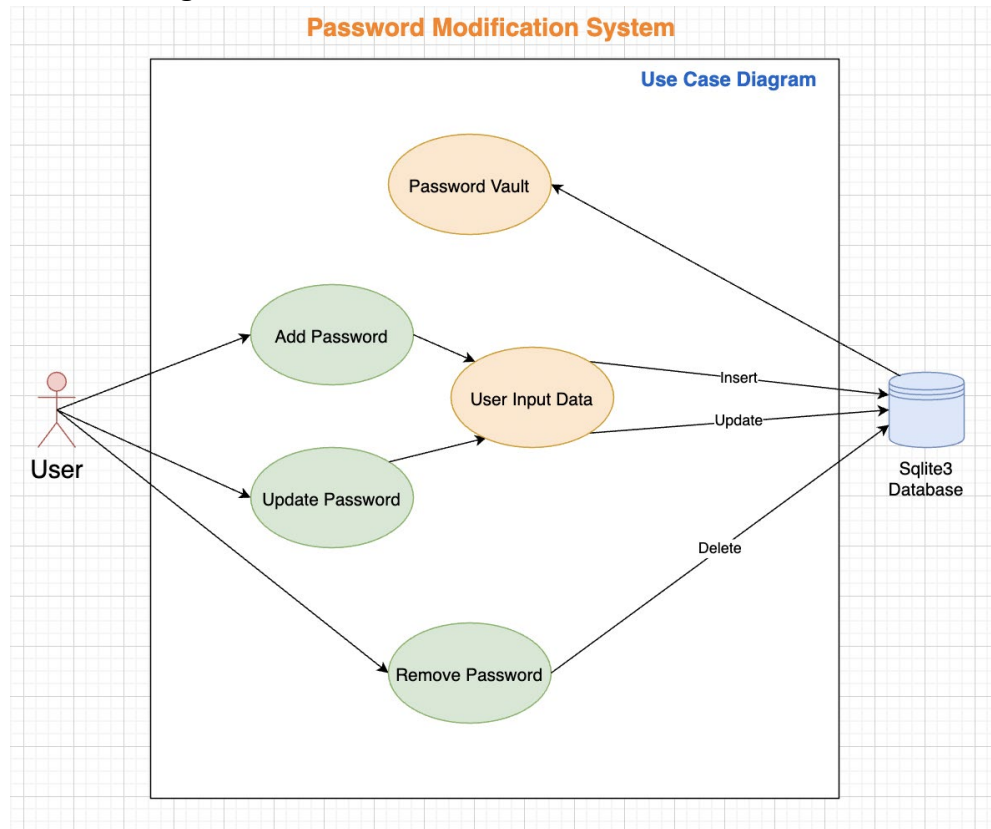
Scope

The scope of this use case is to demonstrate the interaction of the user and password entries stored in the password vault.

Description

The following use case describes a single user adding, removing and updating password entries stored in the password vault.

Use Case Diagram



Flow Description

Precondition

The system is in initialisation mode when the system directs to the password vault screen after a user logs in. The system displays all the password entries present in the password vault.

Activation

The user starts on the password vault screen where they can either add a new password entry, remove an existing password entry or update an existing password entry by clicking on each of the respective add, remove or update buttons.

Main flow

1. The user presses the add password entry button
2. The system directs to the add password screen
3. The user enters the service name, the username and the password in the text fields and presses the ok button
4. The system checks the add password request(see E1,E2,E3)
5. The system inserts the password entry to the password vault database
6. The system directs to the password vault screen

Alternate flow

A1: <The user deletes a password entry>

1. The user presses the delete password entry button
2. The system deletes the password from the password vault database

A2: <The user updates a password entry>

1. The user presses the update password entry button
2. The system directs to the update password screen
3. The user enters the service name, the username and the password in the text fields and presses the ok button
4. The system checks the update password request(see E1, E2, E3)
5. The system updates the password entry to the password vault database
6. The use case resumes at position 6 of the main flow

Exceptional flow

E1: <Service name does not satisfy requirements>

1. The system checks whether the entered service name satisfies the list of service name requirements
2. If the service name does not satisfy the service name requirements the system will output an error message stating the unsatisfied requirement, eg. "service name can not be empty".
3. The use case returns to the same position as the parent flow.

E2: <Username does not satisfy requirements>

1. The system checks whether the entered username satisfies the list of username requirements
2. If the username does not satisfy the username requirements the system will output an error message stating the unsatisfied requirement, eg. "username can not be empty".
3. The use case returns to the same position as the parent flow.

E3: <Password entry does not satisfy requirements>

1. The system checks whether the entered password satisfies the list of password requirements
2. If the master password does not satisfy the password requirements the system will output an error message stating the unsatisfied requirement, eg. "password length should be greater than 8 characters".
3. The use case returns to the same position as the parent flow.

Termination

The system updates the database once the respective password entry modification operation has successfully been carried out and updates the password vault screen to present the changes.

Post condition

The system is in an idle wait state on the password vault screen after an operation by the user has been carried out.

2.1.3. Requirement 3 <Data Processing>

2.1.3.1. Description & Priority

All data stored in the password vault table and master password table is required to be encrypted. Data encryption is done using the strongest SHA-512 protocol to ensure that sensitive data such as passwords remain secure and confidential even if a data breach or leak were to occur. Non-human readable encrypted data stored in byte format in the database is decrypted at run time before it is presented to the user. The encryption and decryption method is derived from the unique master key. Data remains confidential without the corresponding master key.

The third-highest priority is that all data saved in the password manager such as password entries and master passwords should be encrypted. [Priority 3]

2.1.3.2. Use Case: DPREQ3

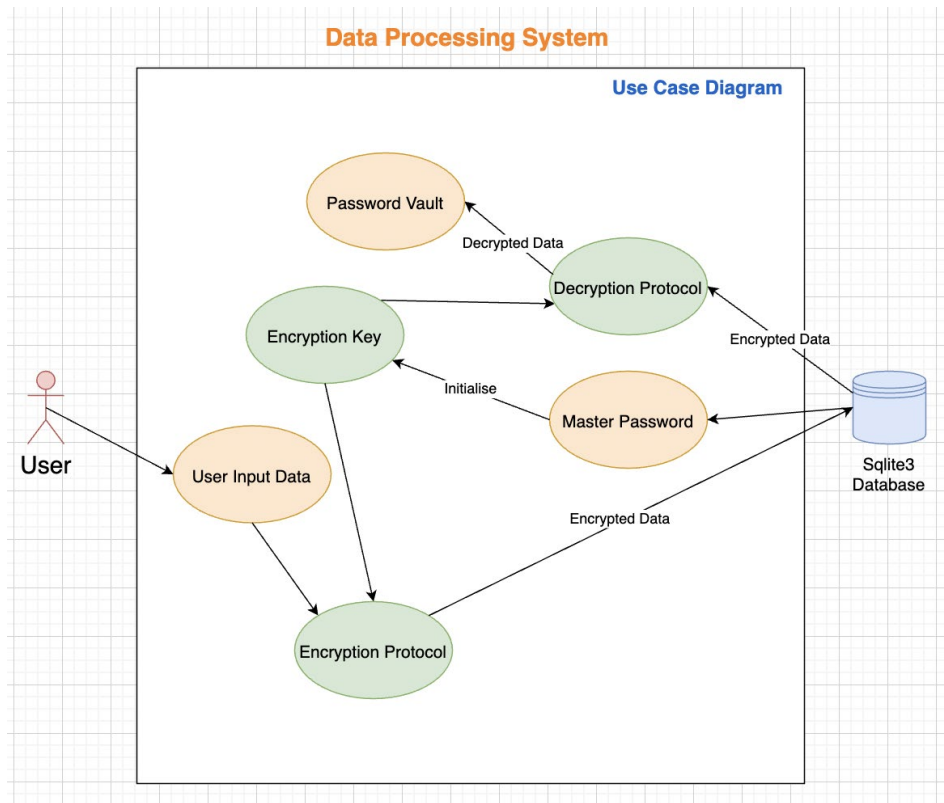
Scope

The scope of this use case is to illustrate the processing by the system of the data stored in the password manager database.

Description

The following use case describes how data is first processed by the system before storing it in the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialisation mode, the system awaits input from the user.

Activation

The system receives unencrypted input from the user. The user directs the system to the password vault screen.

Main flow

1. The system sets the configuration of the encryption algorithm
2. The system encodes the master password to UTF-8
3. The system sets the encryption key to the encryption derivative of the encoded master password
4. The system encrypts the user input using the encryption key (see A1)
5. The encrypted user input is returned

Alternate flow

A1: <The system decrypts encrypted data>

1. The system decrypts the encrypted data using the encryption key
2. The system presents the decrypted data in the password vault

Termination

Once the system has finished processing the encrypted data of the user, the system can perform the respective operation on the encrypted data in the database.

Post condition

The system is in an idle wait state after processing the encrypted data of the user in the database.

2.1.4. Requirement 4<Password Vault Recovery>

2.1.4.1. Description & Priority

The password vault recovery requirement ensures that a user is able to recover their password vault even if they forget their master password or it is unknown. A recovery key is automatically generated whenever a user creates or updates their master password. This true random recovery key is stored along with the master password. The user is prompted to copy this key into their clipboard, to paste and store it in an undisclosed secure location. The user can click the forgot password button to enter their recovery key to reset their master password. If the recovery key matches, the password vault can be retrieved.

The fourth-highest priority is that a user should be able to recover their password vault if their master password is unknown. [Priority 4]

2.1.4.2. Use Case: PVRREQ4

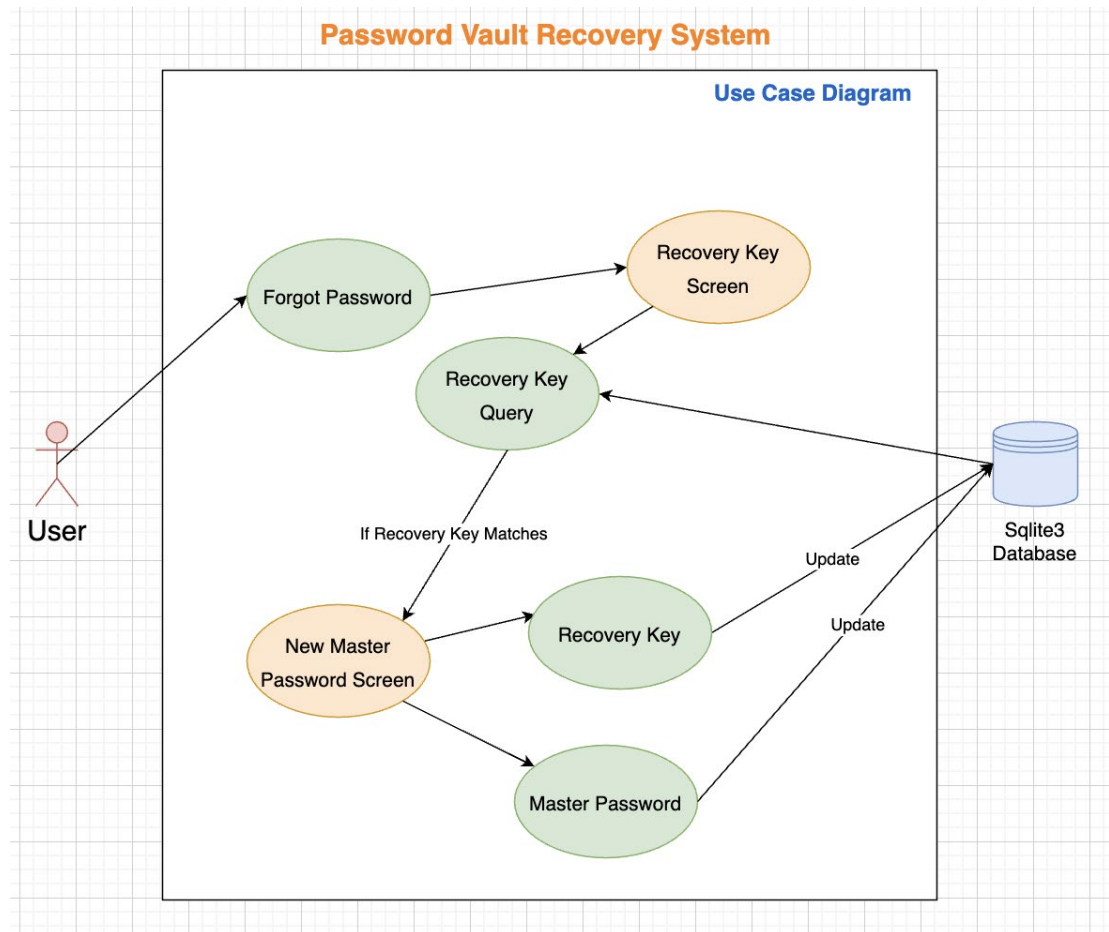
Scope

The scope of this use case is to demonstrate the interaction of the user and the password vault recovery of the system.

Description

The following use case describes a single user recovering their password vault using their respective recovery key to reset their master password stored in the database.

Use Case Diagram



Flow Description

Precondition

The system is in initialisation mode, the user is on the master password login screen of the password vault.

Activation

The user starts on the master password login screen where they press the forgot password button. The system directs to the recovery key screen.

Main flow

1. The user enters the recovery key in the text field and presses the check key button
2. The system checks if the entered recovery key matches the recovery key stored in the database (see E1)
3. If the recovery key matches the recovery key stored in the database the system directs to the create new master password screen
4. The user enters a new master password in the first text field
5. The user repeats the entering of the same master password in the second text field and presses the login button

6. The system processes and checks the register request (see E2, E3)
7. The system directs to the recovery key screen
8. The system generates a new recovery key and stores it in the database
9. The user presses the copy recovery key button and ok button
10. The system directs to the password vault

Exceptional flow

E1: <Recovery key does not match>

1. The system compares the recovery key entered by the user with the recovery key stored in the database
2. If recovery keys do not match the system will output an error message stating, that they do not match.
3. The system clears the entered master password
4. The use case returns to the same position of the parent flow

E2: <Master password does not match>

1. The system compares the master password entered by the user with the master password in the second text field
2. If passwords do not match the system will output an error message stating, that they do not match.
3. The system clears the entered master passwords
4. The use case returns to the same position of the parent flow

E3: <Master password does not satisfy requirements>

1. The system checks whether the entered master password satisfies the list of password requirements
2. If the master password does not satisfy the password requirements the system will output an error message stating the unsatisfied requirement, eg. "master password length should be greater than 8 characters".
3. The use case returns to the same position of the parent flow.

Termination

Once the system has generated the new recovery key and the user presses the done button, they are directed to their password vault.

Post condition

The system is in an idle wait state on the password vault screen after the user-created their new master password.

2.1.5. Requirement 5 <Password Vault Settings>

2.1.5.1. Description & Priority

The password vault settings requirement describes that the password vault should have configured settings according to the requirements of the user for the maintenance of passwords. Each time a password entry is created or updated the time of the modification is also saved. A user specifies after how many days they should be notified if a password entry has been created or has not been changed. Each time the user logs into their password vault the system checks if a password entry is outdated according to the setting configuration. The user is prompted to update the password to a new value. The new value can then be copied from the password vault to use when updating the password of the service.

The fifth priority is that users should be able to configure the maintenance settings of the password vault. [Priority 5]

2.1.5.2. Use Case: PVSREQ5

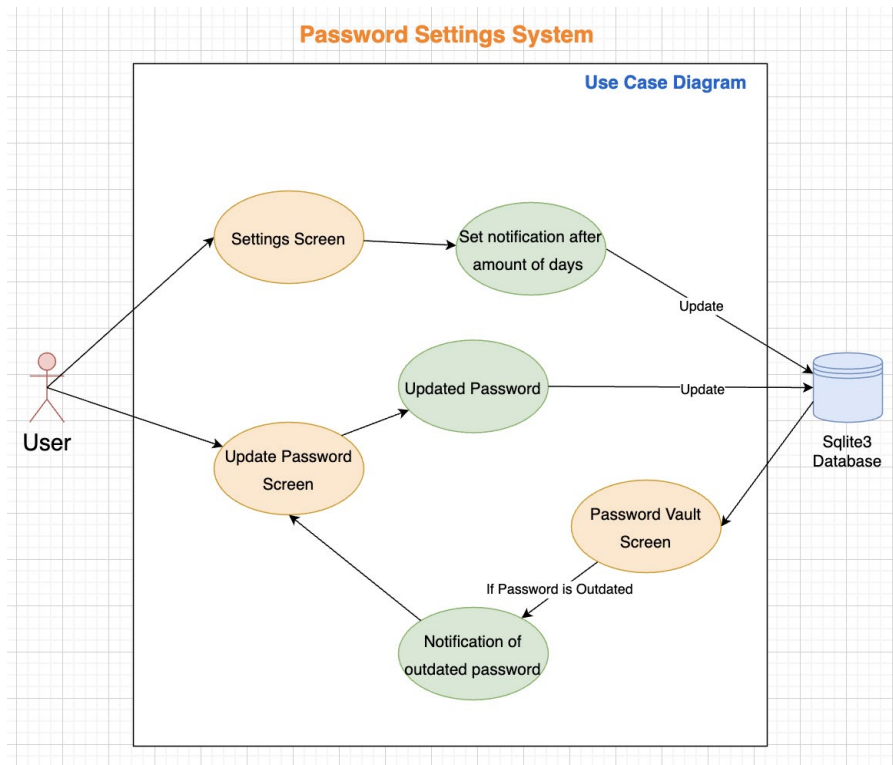
Scope

The scope of this use case is to demonstrate the interaction of the user and the settings of the system for the password vault.

Description

The following use case describes a single user configuring the settings for the password vault system that maintains the stored password entries.

Use Case Diagram



Flow Description

Precondition

The system is in initialisation mode, the system is on the password vault screen and awaits input from the user.

Activation

The user presses the settings button on the password vault screen and the system directs to the password vault settings screen.

Main flow

1. The user enters the amount in days after which they are notified that a password should be updated
2. The user presses the save configuration button
3. The system checks the configuration request (see E1)
4. The system updates the stored settings data in the database
5. The system redirects to the password vault (see A1)

Alternate flow

A1: <Password is outdated>

1. The system checks if the list of password entries stored in the password vault has been updated within the time period of the settings configuration
2. If a password entry is outdated the system notifies the user
3. The system directs the user to the update password screen

4. The user updates the password value of outdated password entry
5. The system updates the date of password update stored in the database
6. The use case resumes at position 5 of the main flow

Exceptional flow

E1: <Configuration does not satisfy requirements>

1. The system checks whether the entered configuration satisfies the list of configuration requirements
2. If the configuration does not satisfy the configuration requirements the system will output an error message stating the unsatisfied requirement, eg. "days can not be greater than 365".
3. The use case returns to the same position of the parent flow.

Termination

The user presses the save settings button and the system updates the password vault settings with the new configuration, the system directs to the password vault screen.

Post condition

The system is in an idle wait state on the password vault screen after saving the updated password vault settings configuration in the database.

2.1.6. Requirement 6 <Password Feedback>

2.1.6.1. Description & Priority

The Password Feedback requirement encapsulates the user's ability to generate a password feedback report for any one password entry that is saved within their password vault. The password feedback screen requires the users to select a single password entry from the list of all entries pulled from their password vault. Once a password entry is selected, the entire password entry is examined by the password feedback function to produce a report displayed to the user. The report includes the password entry, the strength of the password value, the entry creation date, the entry expiration date, the number of uses of the entry, remediation if the password is found with vulnerabilities, how many days till the expiration of the password and the last time that the password vault was used.

The sixth priority is that a user can generate a password feedback report for any password saved within their password vault. [Priority 6]

2.1.6.2. Use Case: PFREQ6

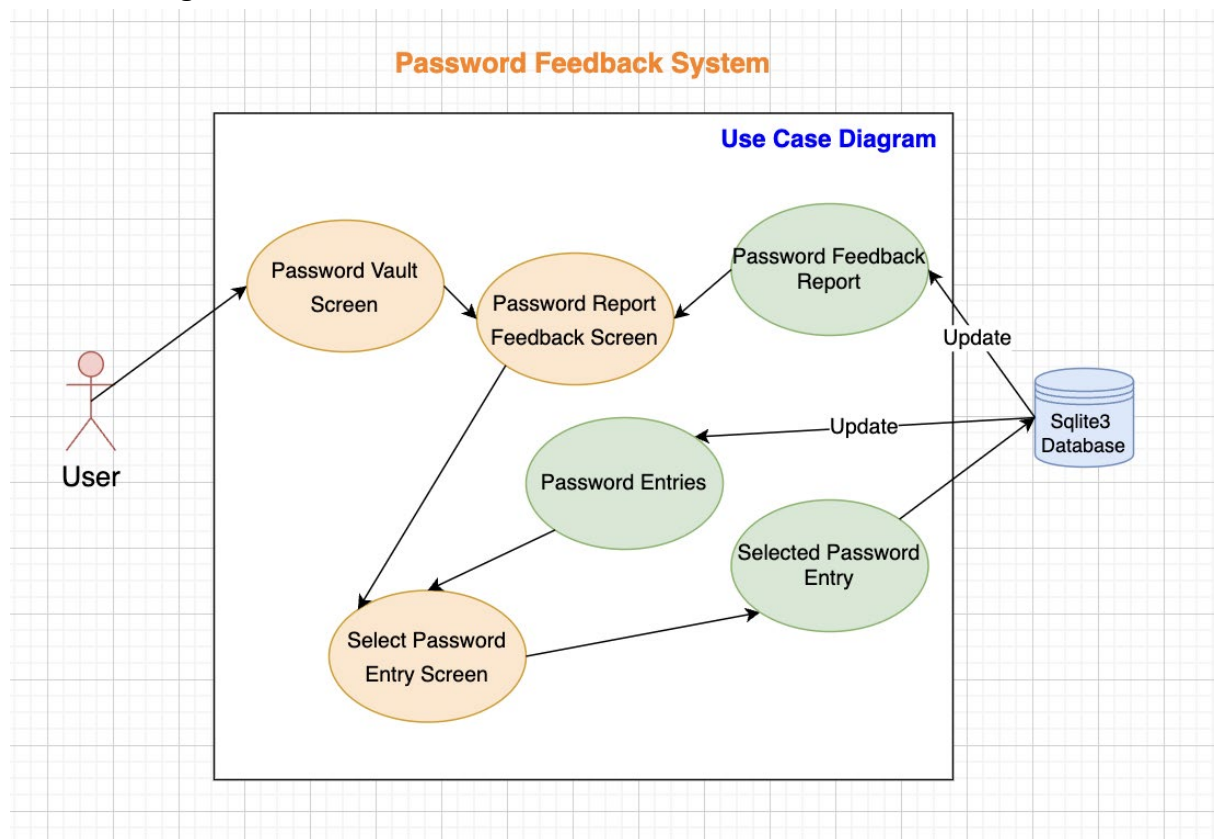
Scope

The scope of this use case is to demonstrate the interaction of the user and the password report feedback mechanism for a single password retrieved from the user password vault.

Description

The following use case describes a single user generating a password feedback report for a password stored in the password vault system that maintains the stored password entries.

Use Case Diagram



Flow Description

Precondition

The system is in idle mode, the system is configured on the password vault screen and awaits input from the user.

Activation

The user presses the report feedback button on the password vault screen and the system directs to the password report feedback screen.

Main flow

1. The system checks the last time the password vault was updated

2. The system populates the last accessed vault field
3. The system sets all the fields to the value: <None>
4. The user presses the select password entry button
5. The system navigates to the select password entry screen
6. The user selects a password entry from the password entry list
7. The user presses the select entry button (E1)
8. The system terminates the select entry screen
9. The system navigates to the password report feedback screen (A1)
10. The system generates a password feedback report for the selected entry
11. The system populates the fields in the password feedback report screen

Alternate flow

A1: <Password entry is vulnerable>

1. The system checks whether the selected password has expired
2. The date of expiry field is marked red if the password has already expired
3. The system checks whether the password satisfies the specifications of a healthy password
4. The password entry field is set to green if the password is safe, is set to orange if the password health is medium, and is set to red if the password health is bad
5. The use case resumes at Step 11 of the Main flow

Exceptional flow

E1: <Password entry is not selected>

1. The system checks whether a password selection has been made from the password entry list
2. If a password entry is not selected, the system will notify the user to select a valid password entry from the list
3. The system will reset its state to select password entry screen without a selection made by the user
4. The use case returns to step 5 of the Main flow

Termination

The user presses the exit button of the password report feedback screen after concluding the generation of password feedback reports for password entries, and the system directs to the password vault screen.

Post condition

The system is in an idle wait state on the password vault screen and awaits input from the user.

2.1.7. Requirement 7 <Inactivity Termination>

2.1.7.1. Description & Priority

The Inactivity Termination requirement enforces a security mechanism employed by the Vault Knox application. When the application is launched and first initialized, a thread is created to check user inactivity. The thread is put to sleep for the interval specified within the user settings configuration. After this period has passed, the thread will prompt the user to specify whether they are still actively using the application. The user is blocked from using any functionality from the application without answering the prompt. If the user confirms that they are still using the application, the thread is reset. If the user denies that they are still using the application, the thread will terminate the processes of the entire application. The user can change the interval of the inactivity check within the user settings screen.

The seventh priority is that the system will prompt the user to specify whether they are still using the application according to intervals specified within the password settings. [Priority 7]

2.1.7.2. Use Case: ITREQ7

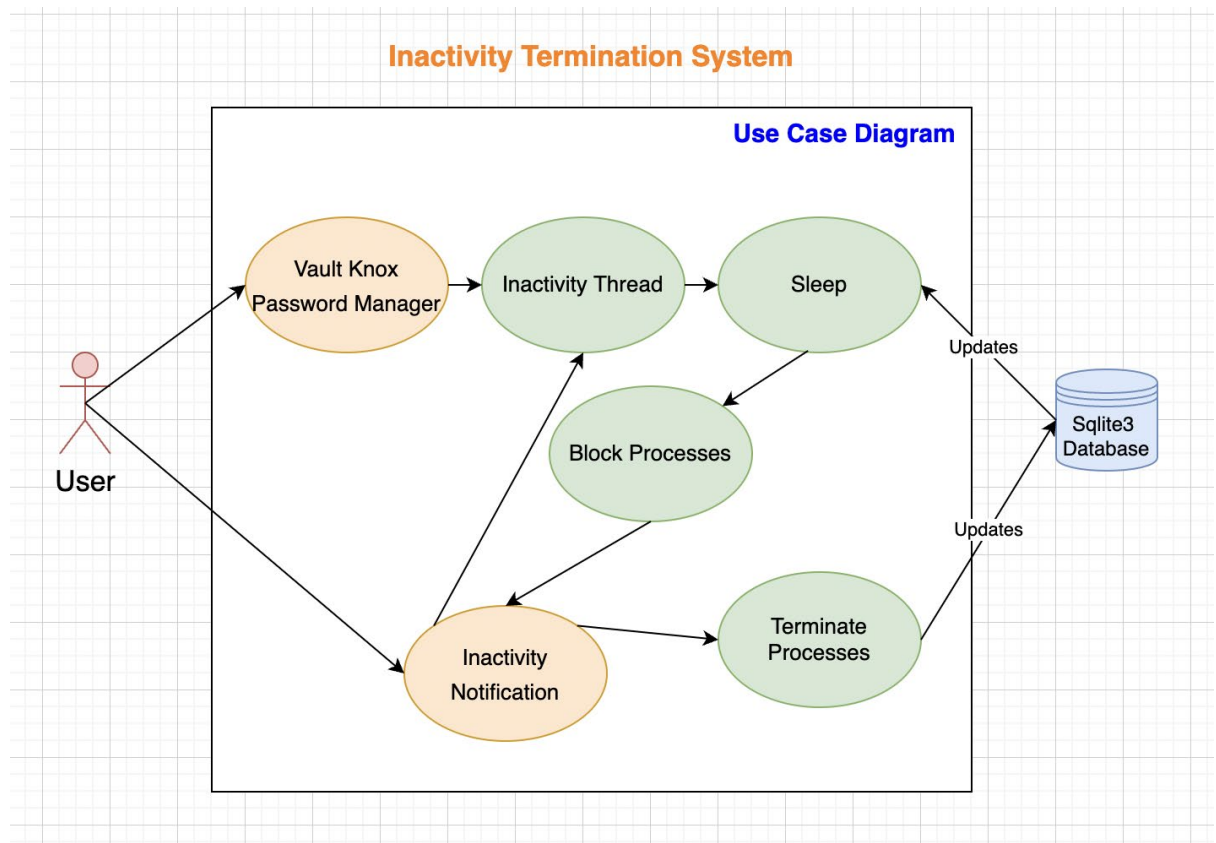
Scope

The scope of this use case is to demonstrate the interaction of the system and the user when the inactivity thread has been executed.

Description

The following use case describes the inactivity thread of the system and the user interaction when the inactivity thread is executed.

Use Case Diagram



Flow Description

Precondition

The system has been initialised and is in an idle state for the duration of the inactivity interval specified within the user settings.

Activation

The user launches the Vault Knox application.

Main flow

1. The system initializes the inactivity thread
2. The system executes the inactivity thread
3. The system puts the inactivity thread to sleep for the interval specified in the user settings
4. The system prompts the inactivity notification to the user
5. The system blocks all features from interaction with the user
6. The user confirms the inactivity notification (A1)
7. The system resets the inactivity thread

Alternate flow

A1: <The user denies the inactivity notification>

1. The user denies the inactivity notification
2. The system terminates all the processes of the application

3. The application is terminated

Termination

The user confirms the activity notification prompted by the system and the inactivity thread is reset or the user denies the activity notification prompted by the system and all the processes of the application are terminated.

Post condition

The application has been terminated by the system if the user denies activity or the system is in an idle state awaiting input from the user if the user confirms activity.

2.1.8. Requirement 8 <Generate Password>

2.1.8.1. Description & Priority

The final requirement is that the user can generate a random password value when updating or creating a password entry. The user can specify the character length of the password they wish to generate. The user can specify whether the generated characters should include uppercase, digits and special case characters. The user can regenerate the password value until they are satisfied with the result. The randomly generated password is saved as the created or updated password value of a single password entry within the password vault.

The last priority is that users can generate a random password when updating or creating a password entry in the password vault. [Priority 8]

2.1.8.2. Use Case: GPREQ8

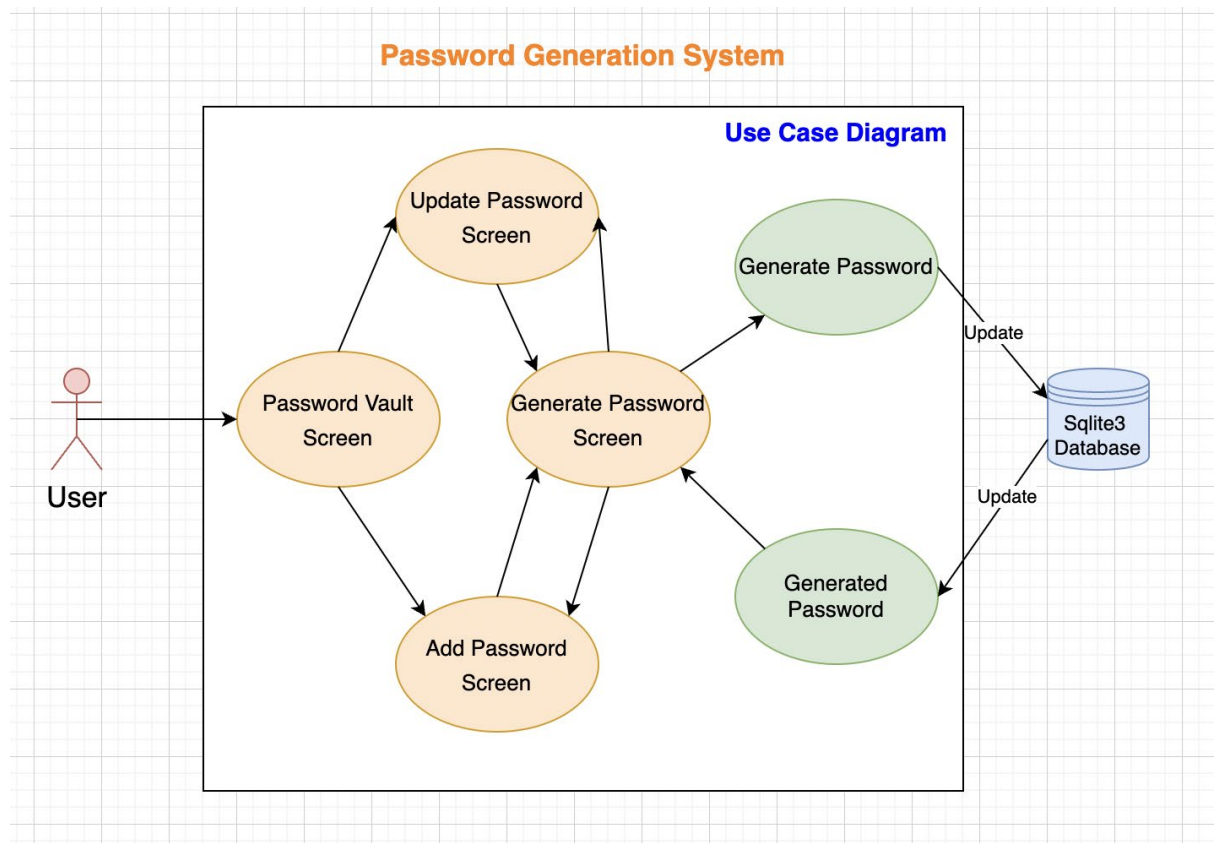
Scope

The scope of this use case is to demonstrate the interaction of the user and the mechanism for generating a random password by the system when updating or adding a password entry in the password vault.

Description

The following use case describes a single user generating a random password and storing it in their password vault system that maintains the stored password entries.

Use Case Diagram



Flow Description

Precondition

The system is on the update password entry screen or on the add password screen and in an idle state awaiting input from the user.

Activation

The user presses the generate password button on the update password screen or the add password screen and the system directs to the generate password screen.

Main flow

1. The user selects the character length of the generated password value from the scroll box
2. The user selects the include digits checkbox
3. The user selects the include uppercase checkbox
4. The user selects the include special character checkbox
5. The user presses the generate password button (A1)
6. The system generates a random password string according to the user configurations
7. The system populates the password entry field with the generated password value

8. The user presses the save password value button

Alternate flow

A1: <The user regenerates a password value>

1. The user presses the generate password button again
2. The use case resumes at Step 6 of the Main flow

Termination

The user presses the save generated password button and the system updates the password entry field in the add password screen or the system updates the password entry field in the update password screen with the newly generated password entry, the system directs to the add password screen or the system directs to the update password screen respectively.

Post condition

The system is in an idle wait state on the add password screen or the update password screen and awaiting the input from the user to store the randomly generated password value in the database.

2.2. Non-Functional Requirements

2.2.1. Security Requirements

The security requirements of the password manager app are the most important non-functional requirements. They are the basis upon which all the rest of the requirements are built. The security of the application aims to balance the effectiveness of performance with the complexity and degree of security/encryption. Login and register functionality was developed with security in mind. Login and register functionality is incorporated to fulfil the majority of the application security requirements in terms of authentication. Users have specified boundaries when inputting data into the master password field in the register section of the application. Restrictions include that passwords must be over 8 characters long and must have at least 1 uppercase character followed by at least one number and special character. Users have the ability to toggle between hiding and showing the character values of the password they input into the master password field. Sensitive data such as passwords are salted and hashed before insertion into the database using SHA-512. All data saved in the database is encrypted using a unique master key specific to the user. Protocols enforce the confidentiality of the application. Sanitization is another security feature implemented to user inputs to strip any unwanted commands that can lead to vulnerabilities such as SQL injection. The application will terminate the session if it remains in a state of inactivity, for a prolonged period of time. Users will have the ability to generate feedback through reports of passwords in their password vault. These reports incorporate

NIST password guideline standards as well as notifications of known compromised user passwords.

2.2.2. User Requirements

In order to gather user requirements, I tested many password managers that are free and available to the public. I demoed each application for at least a week of use to determine the functionalities and aspects of the application that I found the most useful. I also documented all the functionalities and aspects of the application that I did not enjoy to gather a better understanding of how I wanted my application to function. After demoing a total of six password managers I had a list of user requirements that I deemed necessary. I combined all the functionalities that I recorded to be the most useful and incorporated functionality that I wish I had but has not yet been developed. The list of user requirements included: encryption using the strongest encryption protocol(SHA-512), automating the generation of new secure random passwords that satisfy industry-leading strong password protocols and custom user parameters, automated continuous password report feedback and the update of outdated or compromised passwords, offline and online application use, inactivity termination, password recovery and ease of use such as copying passwords to clipboard.

2.2.3. Environmental Requirements

The environmental requirements that are necessary to develop the application are:

Device-MacBook: I use an Apple MacBook to develop this project in the PyCharm IDE.

IDE-PyCharm: I use the PyCharm IDE to write the code for the application. PyCharm IDE is designed specifically for Python development. I find the IDE extremely effective in terms of debugging and troubleshooting exceptions, as well as configuring version control. This IDE also allows me to sync my online GitHub account to push my application to the cloud. I am able to continue development regardless of my device so long as I have an internet connection.

Internet Access: I use internet resources to research methods I can use to develop certain functionalities. Internet Access is however not required for use, as the application runs on an local, offline and secure environment.

Interpreter-Python: To run and test my project I require the python interpreter to compile and run my code from the terminal.

2.2.4. Usability Requirements

In order to use the GUI based application, the user is required to have a rudimentary level of computer skill to login/register using a password, as well as, enter basic selected choices. The application is intended for all users and as such, to be user friendly and customer-centric. Vault Knox Password Manager is a python application with multi-platform support, this means the program can be viewed and accessed by laptops or desktops regardless of the operating system as long as it supports the python interpreter. A user does not require Internet Access as the application runs in a local, offline and secure environment.

2.2.5. Availability Requirement

Availability is important when offering any customer-based service. User's access the Vault Knox Password Manager application using their personal or corporate devices. An offline approach to the program system has been facilitated to prevent the influence of the availability of online networks and the internet to grant availability to the service 24/7. Having access to password information is often a necessity to log in to important services or applications. Users also have the backup ability to recover their password vault in the event that they forget their master password or it is unknown. Availability is dependent on the expected performance of the physical device of the user and nothing else.

2.2.6. Performance/Response time requirement

Performance and response time is another important aspect considering application design for any customer-centric service or application. The application performs without any delay or buffering in responses or operations. The performance and response time are influenced by the hardware specification of the user device. The internet connection of the user does not influence the performance of the application whatsoever. Performance and response time is totally dependent on the hardware requirement on which the application runs. Owners of the hardware are able to make upgrades or replacements to these requirements. Furthermore, the database is designed with sqlite3 which is the best lightweight solution for data storage as the database runs on a single local file. The database is not hosted online which streamlines the reading and writing database processes. The database file remains extremely compact as it only stores byte format text of limited keywords. Performance was kept in mind when developing the encryption protocols to ensure that security requirements were maximised without compromising on response time performance when processing data. Overall the performance efficiency of the application is optimized for processing encrypted data and lighting fast usability.

2.2.7. Robustness requirement

As for all applications, reliability and seamless uninterrupted service is important for user experience. For the development of my application, it is important to add all the necessary functionalities that benefit the overall project, while focusing on keeping it as simplistic as possible, in order to reduce exposure to vulnerability threats as well as bugs. For testing measures, I will write automated unit tests to test multiple scenarios imitating user behaviour to catch unexpected and unwanted exceptions or hidden bugs created by alterations to the current version build. These behaviour actions include registration and login, password entry modification, error handling, maintenance scheduling, form sanitisation, feedback reports and recovery. Tests ensure that checks are correctly implemented to prohibit the malicious influence or access of program structure, protocols and sensitive data. A testing methodology supports an approach of prevention rather than reaction. Data breaches and leaks have untold consequences on monetary, time and manpower resources.

2.2.8. Data Management requirement

Vault Knox Password Manager contains and manages confidential and sensitive user data such as credentials. If third parties were able to access and penetrate the security mechanisms of the application, consequences could be dire. Data saved within the application should remain in an eternal secure state. Users have the ability to wipe all their data and their correlating user account from the system. This will erase all evidence of the user profile and as a result, a new vault can be registered. Furthermore, users can specify the amount of time after which they should be notified that a password is out of date. Finally, the entire application data will prompt a self-destruct function if it is identified that a user is trying to brute force attack the recovery mechanism by entering multiple invalid recover key values. These mechanisms provide users with peace of mind knowing that their data will remain secure and have the flexibility to perform data maintenance.

2.2.9. System requirement

The entire Vault Knox password manager application is built using the Python language. Python is a very versatile language, allowing files to be executed on a system indiscriminately of the operating system. All that is required is an up to date Python compiler such as Python 3.10. Furthermore, the application has been converted and finalised into a single executable file. This executable file creates a virtual environment with all the necessary dependencies already set up, without the user needing to install a valid Python version, package or library.

2.2.10. Reliability requirement

Testing is a vital practice to ensure the reliability of an application. Once changes or improvements have been made, rigorous testing protocols will identify undesired behaviour of previously developed code. The scope of testing is however restricted to automated tests that might not be intuitive enough to replicate the interaction of human behaviour. Testing is directly correlated to the quality of written tests and is highly dependent on the expertise of the developer. To better manage the reliability of the application I have involved fellow students to stress test each iteration of the development cycle. The goal of this practice is to try and break or create unwanted behaviour in established application features. This is best practice to simulate the interaction of users within a live environment before the application is deployed. Different users will also provide the reliability requirement with a difference in physical hardware and software specifications. These specifications that result in unwanted behaviour can not be determined by the developer otherwise, making it an invaluable asset. The process of identifying bugs and errors is exponentially improved by increasing the interaction of user input on the system.

2.2.11. Maintainability requirement

Maintainability of the VaultKnox application has been facilitated to ease the development process. This is a solo project that requires a lot of development time and having a clear cut development process really aids in efficiently maintaining the quality of the project. Practices such as labelling the purpose of functions and classes allow me as a developer to quickly pick up where I left off, without having to research functionality design in previous development sessions. Classes also facilitate a clear process flow for system interactions with the user. Designing classes with design principles in mind has enabled me to prevent unwanted discrepancies in behaviour when modifying existing or creating new features and affecting other aspects of the project.

2.2.12. Reusability requirement

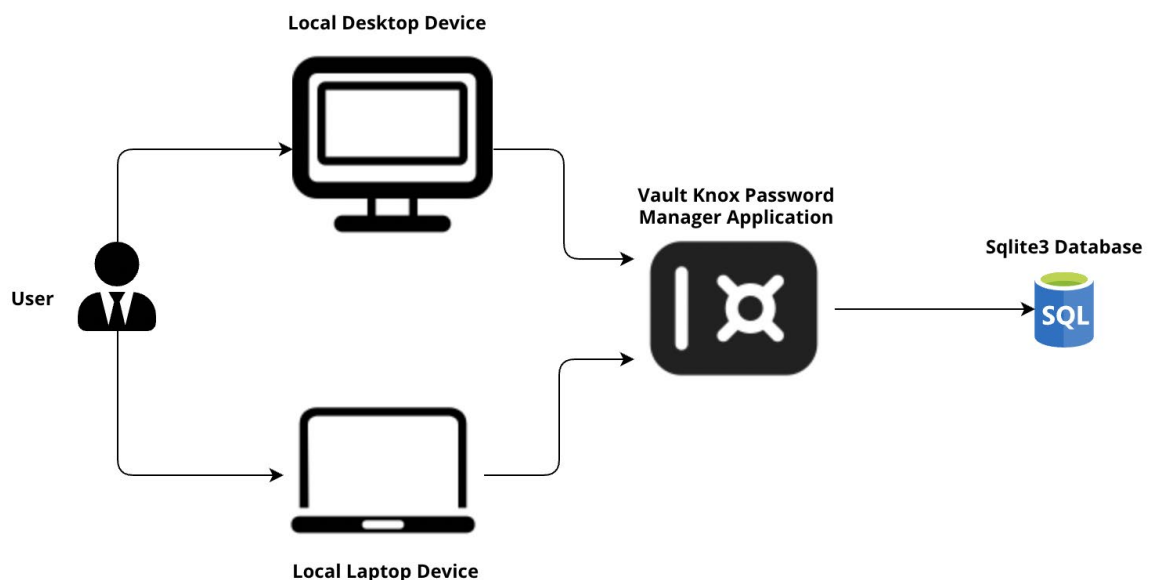
Design principles that utilize mechanisms such as encapsulation of functionality in classes and objects have been implemented to streamline the reusability of code. The application is coded in an Object Orientated Programming standard to facilitate this requirement. Coding new features are seamless due to the foundation of classes and functions that can be reused. Reusability allows for greater maintainability of code by creating a more clean and easily readable program as well as limiting the duplication of code. Python is a dynamically interpreted language and gains performance benefits from interpreting the minimum amount of lines of code required to execute functionality.

2.2.13. Extensibility requirement

VaultKnox as a password manager has a lot of scope for collaborating with security services and affiliates. For this reason, with the development of security practices and protocols, the features of VaultKnox can be extended and improved indefinitely. Security is an ever-changing field and applications require a degree of malleability to satisfy changing security standards. A newer version of VaultKnox password manager can be deployed with the same mechanism.

2.3. Design & Architecture

System Architecture Diagram

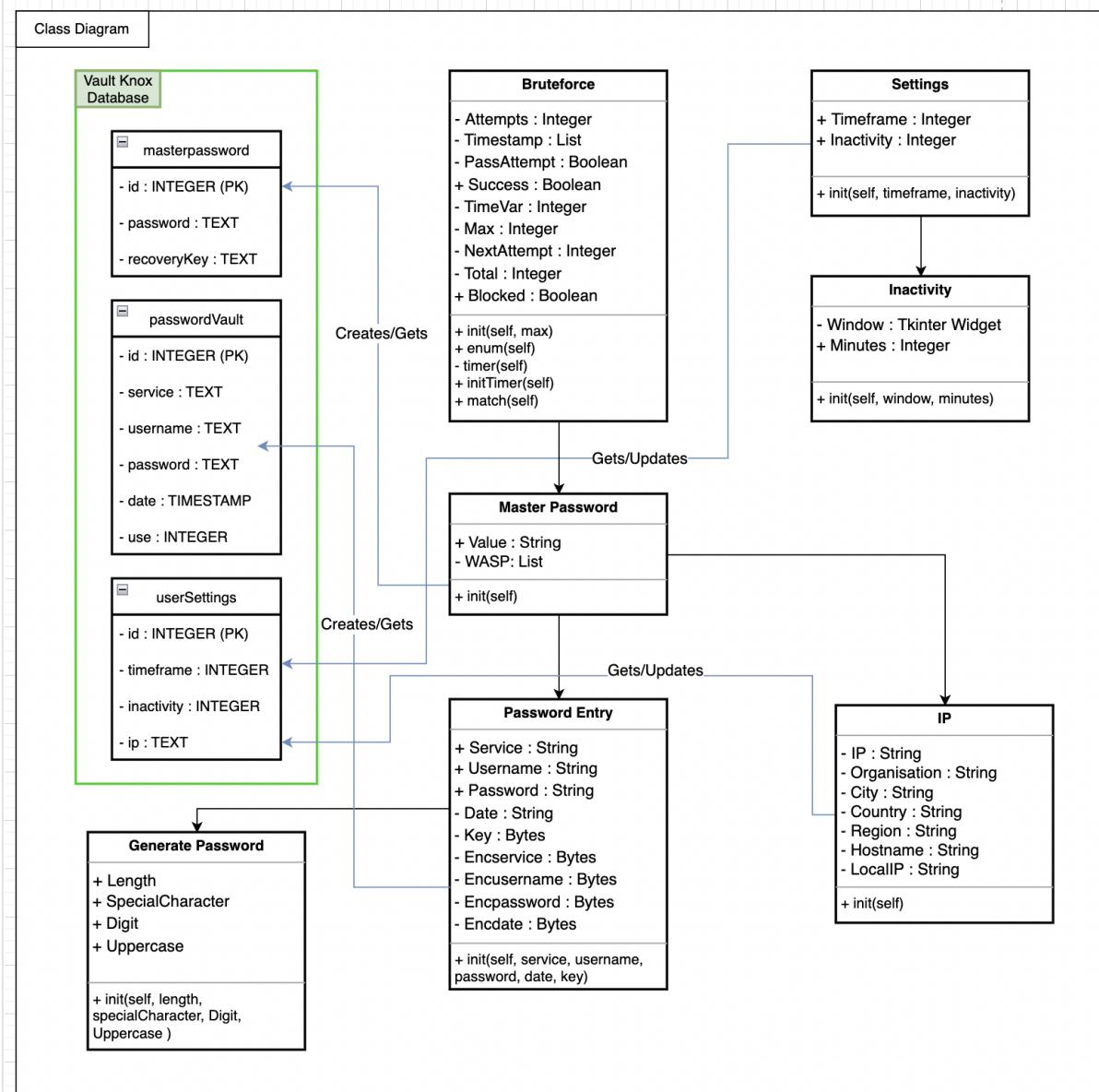


The above diagram displays how the system architecture is designed. The user can access the application from either a desktop or laptop device client. Vault Knox Password Manager is a python application with multi-platform support, this means the program can be viewed and accessed by laptops or desktops regardless of the operating system as long as it supports the python interpreter. A user does not require Internet Access as the application runs in a local, offline and secure environment. The application was developed using the Python scripting language. The design aspect itself has been coded using the Tkinter framework to generate GUI components such as windows, frames and widgets for handling application styling and user control/input and interfacing such as the use of buttons and popups. The entire interface was designed from the ground up and functionality was coded for each component.

The application itself is accessed through the user's device that contains the Sqlite3 database. The database is specific to the device that the user runs the application on. The application communicates with the database via encryption, decryption and hashing protocols. The user gains access to the database with their respective master password. The master password derives the encryption master key that is used to process all the data that is stored and displayed on the application.

My main design goal from the start of the planning phase is to create a user experience that prioritizes logical flow and ease of use. I desire to create a seamlessly flowing interface. To achieve this, I created a navigation menu allowing the user to easily move between screens. I kept the design choice simplistic with distinguishable icons and basic colour tone to give a professional and structured feel.

Class Diagram



Master Password Class

When the user first launches the application they are prompted to register a new master password. This master password is used to log in to the user password vault. The master password class initializes a master password object. The master password object takes the value of the entered password. Furthermore, the master password class contains the "WASP" parameter that stands for: with advanced security principles of type list. An initialised master password object initializes an empty WASP list variable. The class handles advanced security principle checks to determine any vulnerabilities found within the entered value parameter. Remediation of found vulnerabilities is created and appended to the WASP variable. If there are no items within the WASP variable, the password is accepted. If there are items within the WASP list, it is displayed to the user for remediation purposes and is not accepted or stored within the database. The master password controls the access to a password vault as well as is utilized for the encryption mechanism. The encryption key employs the master password value to derive a key function that is used to encrypt and decrypt values stored within the local database file. Rather than encrypting the master password, it is hashed to create a 32 byte hashed value and stored in the database. This hashed value is compared with the hashed value of the user-supplied master password field to determine whether a user has access to the configured vault.

Password Entry Class

Password entries that are stored within the password vault are handled using the password entry class. The password entry class initiates objects for creating or modifying password entries. Password entry type objects are created that accept parameters for the service name, username, password, date of creation and finally, the encryption key. The password entry class handles the checking of password entries against security protocols. If the password entered does not meet anyone security protocol the password entry is abandoned and the failed security measure is displayed to the user for remediation purposes. If the password value of the password entry object passes all the security protocols, each parameter field is encrypted using the encryption key. Encrypted values are set to the encservice, encusername, encpassword and encdate variables within the password entry object respectively. These encrypted fields are utilized as the stored values when creating or updating the password entry to the database.

Settings Class

The settings class is responsible for instantiating the settings configuration of the user for their respective password vault. The settings class accepts fields timeframe and inactivity when creating user settings objects. The timeframe parameter specifies the period after the creation of a password entry in which the user wishes to set a password as outdated. The inactivity parameter specifies the interval after which the application will notify the user if they are still using the application. The timeframe parameter is used for checking outdated

passwords and the inactivity parameter is used for checking user inactivity when using the application. If there is no prior evidence of data stored specifically to the user settings, the settings class will instantiate a settings type object with 1 month for the timeframe parameter and 5 minutes for the inactivity parameter. However, if evidence of stored setting data is found, an object is created using the stored user-supplied values from the database. The settings class can be accessed and modified from the user settings screen.

Bruteforce Class

The Bruteforce class is responsible for prohibiting brute force attacks aimed at the system access mechanisms. Access requests are sanitized and checked with security measures set out in the Bruteforce class. The Bruteforce class initiates object of type Bruteforce that accepts field maximum as a parameter to specify the maximum attempts a user can make when requesting access to their password vault. A Bruteforce object initiates a thread to start the execution of a timer. The timer thread keeps track of the number of current attempts requested within a 60-second timeframe. Each time the user makes a request attempt, the object calls the enum function which will check whether the current total requests are less than the maximum allowed requests. If true, the number of attempts is increased and the timestamp of the request is appended to a list variable. Each item in the timestamp list is popped after 60 seconds have passed. While the timestamp list has the maximum number of elements, all request attempts are denied until an element has been popped after 60 seconds have passed. For the reset master password mechanism that utilizes the checking of a recovery key, the Bruteforce class will only a maximum of 5 requests before terminating all the user data and account. In this case, the brute force class identifies the excessive recovery request as malicious Bruteforce activity. User data is terminated to protect it from being accessed by a malicious attacker. For the login mechanism, the application will accept a maximum of 10 login request attempts for locking a user account. The user can only continue attempting login requests by resetting their master password using the appropriate recovery key for the password vault.

IP Class

The IP class handles the configuration details of the system that runs the Vault Knox application. The details are utilized for advertising purposes when marketing the affiliate link to the Virtual Private Network service, NordVPN. The IP class instantiates IP type object that import modules socket and urllib. The socket library is employed to gather information regarding the system hostname and the local system IP address. The urllib library is used to make an API call to identify the details of the system's online IP. The response is captured using JSON data and saved to the field IP, organisation, city, country and region. This call identifies the geographical location of the system and is displayed to the user as a warning of vulnerability. A VPN service will spoof these details with the details of the proxy server, protecting the user. If either one of the libraries can not determine a value for the

instantiated fields, the field is set to the value “Unknown”. For instance, if the system does not have an active internet connection and is offline, all the values retrieved by the urllib module will be set to unknown. In this case, an unknown value constitutes the system being secure from determining the geographical location of the system.

Generate Password Class

The generate password class handles the creation of a new randomly generated password configured by the user. The generate password class instantiates generated password objects that accept parameter length, special characters, digits and uppercase. The length parameter specifies the total character length of the password the user wishes to generate. The length parameter is retrieved from the update and add password entry screens where the user can select an integer value between 4 and 64. The following parameters are of type boolean and specify whether the generated password should include certain characters. If the boolean variable is true, the type of character will be utilized in creating the generated password. If the boolean variable is false, the type of character will not be utilized to create the generated password. These boolean fields are set within the same above mentioned screens. If the user is satisfied with the generated password value, the generated password object will be sent to the respective screen for storage procedures.

Inactivity Class

The inactivity class is responsible for instantiating the thread that checks the user activity when the application is launched. An inactivity type object is instantiated once only when the program is first executed. The inactivity class takes parameters window and minutes as fields for instantiating an inactivity type object. The window parameter specifies the main GUI responsible for the main application flow. This window variable will be destroyed if the user denies the activity check, resulting in terminating all the processes of the application. All processes and threads are dependent and built upon the main GUI component and set to a Daemon type. Daemon threads will self destruct when dependent threads are terminated. The minutes parameter takes the interval after which the inactivity notification will be executed and displayed to the user for input. If the user confirms their activity, the inactivity object will be reset and put to sleep for the duration specified in the minutes field. The inactivity class can be accessed and modified from the user settings screen. The minutes field retrieves an integer value between 5 - 60 from the scrollbox configured by the user in the user settings.

2.4. Implementation

Encryption:

```
#encryption

backend = default_backend()
salt = b'2444'

kdf = PBKDF2HMAC(
    algorithm=hashes.SHA512(),
    length=32,
    salt=salt,
    iterations=100000,
    backend=backend
)

encryptionKey = 0

def encrypt(message: bytes, key: bytes) -> bytes:
    return Fernet(key).encrypt(message)

def decrypt(message: bytes, token: bytes) -> bytes:
    return Fernet(token).decrypt(message)
```

Encryption is implemented using the fernet cryptography library. The library contains a key derivation function called pbkdf2_hmac. The key function generates a derived key that is unique to the master user. The same key is used to decrypt values that are encrypted. The function accepts 5 parameters to generate the function: hash name, password value, salt value, number of hash iterations and the key length. In this case, the function will contain: the hash name SHA-512, the user-supplied password value, the unique salt value generated from the operating system UUID function and the standard key length of the SHA-512 hash which is 64 bytes.

```
# Insert password entry to offline database

service = encrypt(ser.encode(), encryptionKey)
username = encrypt(user.encode(), encryptionKey)
password = encrypt(passw.encode(), encryptionKey)
currentDateTime = datetime.now()
date = encrypt(str(currentDateTime).encode(), encryptionKey)

insert_fields = """INSERT INTO passwordVault(service, username, password, date, use)
VALUES(?, ?, ?, ?, ?)"""

cursor.execute(insert_fields, (service, username, password, date, 0))
db.commit()
pwFrame.destroy()
passwordVault()
```


Hashing:

Hashing produces a unique one-size signature. This is an irreversible one-way function that ensures integrity. Many weak hashing algorithms such as MD5 have already documented exploitable rainbow tables. A rainbow table is a precomputed table used to cache the output of cryptographic hash functions, which is typically used to brute-force crack password hashes. Tables are commonly used to reconstruct the key derivation function that is made out of a limited number of characters. Small hashing algorithms have a greater chance to produce hash collisions. Hash collisions occur when two distinct values produce the same hash signature due to the finite amount of characters.

```
if savePW:

    sql = "DELETE FROM masterpassword WHERE id = 1"

    cursor.execute(sql)

    hashedPassword = hashPassword(txt.get().encode('utf-8'))

    key = str(uuid.uuid4().hex)
    recoveryKey = hashPassword(key.encode('utf-8'))

    global encryptionKey
    encryptionKey = base64.urlsafe_b64encode(kdf.derive(txt.get().encode()))

    insert_password = """INSERT INTO masterpassword(password, recoveryKey)
                        VALUES(?, ?)"""
    cursor.execute(insert_password, ((hashedPassword), (recoveryKey)))
    db.commit()

    btn2.config(text="Success")
    btn2.after(500, partial(recoveryScreen, key))

    pyperclip.copy(entPW.value)

else:
    txt.focus()
```

The hashing of passwords is facilitated using the hashlib library and the SHA-512 algorithm, which is considered to be the most secure protocol. The long character length of the hash also is best equipped to prevent collisions. Adding a salt value to the hashing algorithm further increases complexity. Most importantly we achieve protection against rainbow tables. Salt values should be user-specific and unique for the best results. Because of the unique nature of the salt value, derived keys no longer mimic rainbow tables or produce

known hash collisions. Popular salt generation methods include utilizing system date and time, IP addresses, user input, etc. It is possible for a very skilled individual to acquire such information. To best protect against such practices we can use a randomly generated salt value utilizing the operating system UUID to generate a true random value. It is necessary to encode user password input to UTF-8 so that the hashing algorithms - hex digest can be applied.

Input validation and representation:

```
while True:
    cursor.execute("SELECT * FROM passwordVault")
    array = cursor.fetchall()

    if(len(array) == 0):
        break

    lblOne = Label(window, text=(decrypt(array[i][1], encryptionKey)), font=("Helvetica", 12))
    lblOne.grid(column=0, row=i+3)

    lblOne = Label(window, text=(decrypt(array[i][2], encryptionKey)), font=("Helvetica", 12))
    lblOne.grid(column=1, row=i+3)

    lblOne = Label(window, text=(decrypt(array[i][3], encryptionKey)), font=("Helvetica", 12))
    lblOne.grid(column=2, row=i+3)

    btn = Button(window, text="Delete entry", command=partial(removeEntry, array[i][0]))
    btn.grid(column=3, row=i+3, pady=10)

    btnOne = Button(window, text="Update entry", command=partial(updateEntry, array[i][0]))
    btnOne.grid(column=4, row=i + 3, pady=10)

    i += 1

    cursor.execute("SELECT * FROM passwordVault")
    if len(cursor.fetchall()) <= i:
        break
```

With the development of the Vault Knox password manager It is of vital importance that accounting for validation of user input before manipulating or utilising values during runtime is made. Improper validation of such user-supplied input values can lead to vulnerabilities such as SQL injection and breaking the logic flow of the application, producing unwanted output.

The zxcvbn package is a password strength estimator package that tests passwords against predefined rainbow tables to find vulnerable values. Pattern matching password values help identify known and easily guessable passwords. The package produces a score from 1 - 4 to rate the security of any given supplied password. Passwords that do not have a security score of 1, produce remediation suggestions in the feedback dictionary field. In this project, remediation methods for unsatisfied passwords with a security score higher than 1 are projected to the user through the GUI. This process is repeated until the validation of password input is satisfied.


```
# Password remediation zxcvbn package

if len(entPW.WASP) == 0:
    results = zxcvbn(txt.get())
    suggestions = results['feedback']['suggestions']

    if suggestions == []:
        val = txt.get()
        txt.delete(0, "end")
        txt.config(fg="light green")
        txt.insert(0, val)
        val1 = txt.get()
        txt1.delete(0, "end")
        txt1.config(fg="light green")
        txt1.insert(0, val1)
        savePW = messagebox.askyesno("Password OK",
                                     f"Are you sure you want to set your master password to:\n\n{entPW.value}\n\n"
                                     f"Your password will be copied to your clipboard if you wish to "
                                     f"securely save it locally.")
```

During run-time, the user is prompted to enter sensitive and confidential values such as a master password. They have the capability to hide the actual character values. This accounts for the fact that a user might not be acting in a confidential environment and passwords could be compromisable by projecting them to the graphical user interface.

```
if not showing:
    btnHide.config(text="Show Passwords")
    btnThree = Button(window, text="*****", font=("Helvetica", 12))
else:
    btnThree = Button(window, text=(decrypt(array[i][3], encryptionKey)), font=("Helvetica", 12))

btnOne = Button(window, text=(decrypt(array[i][1], encryptionKey)), font=("Helvetica", 12))
btnOne.grid(column=0, row=i + 3)
btnOne.config(command=partial(copyIn, array[i][1], btnOne))

btnTwo = Button(window, text=(decrypt(array[i][2], encryptionKey)), font=("Helvetica", 12))
btnTwo.grid(column=1, row=i + 3)
btnTwo.config(command=partial(copyIn, array[i][2], btnTwo))
```

Furthermore, during input validation, all abilities such as SQLi are accounted for by utilising a python context manager called cursor. because the context manager binds to an instance of a database, in this case, an SQLite3 file. Passing variables that are intended to be stored in the database can be used with the (?) - operator. This prohibits the user from injecting commands through the graphical user interface and producing unwanted behaviour in the database.

```
insert_password = """INSERT INTO masterpassword(password, recoveryKey)
                    VALUES(?, ?)"""
cursor.execute(insert_password, ((hashedPassword), (recoveryKey)))
```

Example of vulnerable SQLi in python:

```
def add(table,*args):
    statement="INSERT INTO %s VALUES %s" % (table,args)
    cursor.execute(statement)
```

Errors:

Undefined and unhandled errors can occur in an application that does not account for the complete spectrum of inputs or specifications of user systems. This project is developed in Python and as such, flexible in terms of cross-platform support. The graphical user interface is initialised and set up according to the specifications of the user's system by querying dimensions such as the screen width and height for configuring resolution and size.

```
window = Tk()

screen_width = window.winfo_screenwidth()
screen_height = window.winfo_screenheight()
```

```
app_width = int(screen_width / 2)
app_height = int(screen_height)
xCor = (screen_width / 2) - (app_width / 2)
yCor = (screen_height / 2) - (app_height / 2)
window.geometry(f'{app_width}x{app_height}+{int(xCor)}+{int(yCor)}')
```

Furthermore, logic blocks account for all possible inputs that include undefined or unwanted formats. Logic is clearly defined by creating a flow of decision making specific to input specifications such as length and type. Undefined input produces remediation methods that instruct users to recalibrate or resubmit user-supplied input.

```
if ser == "":
    messagebox.showerror("No service name", "Please enter a valid service name.")
    return
elif len(ser) > 64:
    messagebox.showerror("Service name too long", "The service name is too long, please enter a valid service name.")
    return
elif user == "":
    messagebox.showerror("No username", "Please enter a valid username.")
    return
elif len(user) > 64:
    messagebox.showerror("Username too long", "Your username is too long, please enter a valid username.")
    return
elif passw == "":
    messagebox.showerror("No password", "Please enter a valid password.")
    return
elif len(passw) > 64:
    messagebox.showerror("Password too long", "The password is too long, please enter a valid password.")
    return
else:
```

Encapsulation:

Encapsulation is the practice of hiding or quarantining logic within classes and instantiating objects to define the behaviour of variables. In doing so, confidential data and parameters are secluded from other logic that could attempt to manipulate unwanted output. For instance, passwords remain within the scope of classes and are inaccessible by the graphical user interface. The values to the behaviour of such password parameters can be influenced or stored permanently in the database.

```
class MasterPassword:
    #Init Masterpassword object
    def __init__(self, value, WASP):
        self.value = value
        # WASP: With Advanced Secure Programming Principles
        self.WASP = WASP

    # Validation and remediation of master password
    if len(value) < 8:
        self.WASP.append(". Password should not be shorter than 8 characters.")
    if len(value) > 64:
        self.WASP.append(". Password should not be longer than 64 characters.")
    if not re.search("[a-z]", value):
        self.WASP.append(". Password should contain at least one lower case character.")
    if not re.search("[A-Z]", value):
        self.WASP.append(". Password should contain at least one upper case character.")
    if not re.search("[0-9]", value):
        self.WASP.append(". Password should contain at least one digit.")
    if not re.search("[^A-Za-z0-9]", value):
        self.WASP.append(". Password should contain at least one special character.")
```

Through the graphical user interface, we further establish an encapsulated methodology by streamlining and restricting the possible behaviour of the user. Input Fields restrict the user in their capabilities to produce unwanted behaviour. Functionality from one graphical user interface is secluded and unmodifiable by methods present in another.

```

class PasswordEntry:
    #Init Password entry object
    def __init__(self, service, username, password, key):
        print(key)
        self.service = service
        self.username = username
        self.password = password
        self.enservice = ""
        self.encusername = ""
        self.enpassword = ""
        # Validation and remediation of password entry object
        if self.service == "":
            messagebox.showerror("No service name", "Please enter a valid service name.")
            return
        elif len(self.service) > 64:
            messagebox.showerror("Service name too long",
                                "The service name is too long, please enter a valid service name.")
            return
        elif self.username == "":
            messagebox.showerror("No username", "Please enter a valid username.")
            return
        elif len(self.username) > 64:
            messagebox.showerror("Username too long",
                                "Your username is too long, please enter a valid username.")
            return
        elif self.password == "":
            messagebox.showerror("No password", "Please enter a valid password.")
            return
        elif len(self.password) > 64:
            messagebox.showerror("Password too long",
                                "Your password is too long, please enter a valid password.")
            return
        else:
            # Password entries encrypted for storage in database
            self.enservice = encrypt(self.service.encode(), key)
            self.encusername = encrypt(self.username.encode(), key)
            self.enpassword = encrypt(self.password.encode(), key)

```

Security Features:

Security is the main benefit of utilising a service such as a password manager. Functionality such as encryption, decryption, hashing, random value generation and character masking are all protocols used to manage data in the most secure and robust fashion. The strongest SHA-512 encryption protocol is used to encrypt data, ensuring that sensitive information like passwords remains secure and secret even if a data breach or leak occurs. Before it is displayed to the user, non-human readable encrypted data stored in byte format in the database is decrypted at run time. The encryption and decryption methods are based on the master key's uniqueness. Without the accompanying master key, data stays private.

Rainbow table dictionary attacks are utilised to brute-force known hash or encrypted data. To best protect against such practices we can use a randomly generated salt value utilizing the operating system UUID to completely randomize the value. The salt value is incorporated into the key derivation function to further encrypt and randomise produced encrypted values. After the encryption protocol is established we iterate the mechanism a 100,000 times.

```

backend = default_backend()
salt = b'2444'

kdf = PBKDF2HMAC(
    algorithm=hashes.SHA512(),
    length=32,
    salt=salt,
    iterations=100000,
    backend=backend
)

```

The key derivation function is then utilised to create the unique encryption key into lives for all encryption and decryption methods.

```

global encryptionKey
encryptionKey = base64.urlsafe_b64encode(kdf.derive(userpw.encode()))

service = encrypt(ser.encode(), encryptionKey)
username = encrypt(user.encode(), encryptionKey)
password = encrypt(passw.encode(), encryptionKey)

```

Encrypted values of encrypted data stored in the database can be hidden using character masking in runtime. This does not influence the behaviour of copying values from the database or to the clipboard. The representation of such values or however hidden.

```

def hideT():
    txt.config(show="*")

def showT():
    txt.config(show=" ")

def checkT():
    global x
    x += 1
    if x % 2 == 1:
        btn2['text'] = "Show"
        hideT()
    else:
        btn2['text'] = "Hide"
        showT()

```

The UUID system value is a true random representation of number generation that is made specifically for anyone single-user system. It serves as the perfect candidate for creating a unique recovery key specific to the user password vault and respective system. This ensures recovery key security and generation specifically to systems of users. Malicious users are

unable to guess this encoded hash value or executed a brute force dictionary attack.

```
key = str(uuid.uuid4()).hex
recoveryKey = hashPassword(key.encode('utf-8'))
```

Loadscreen:

A load screen is instantiated when the application is first launched. The load screen is represented as a splash canvas that displays the Vault Knox password manager logo to the user. The load screen is displayed to the user for a period of 2-seconds and serves as the mechanism for instantiating functions of the main class in the background. When the load screen has terminated, the login screen will have been instantiated with the necessary values such as the system IP configuration details and utilised for marketing the virtual private network affiliate link.

```
def resource_path0(relative_path):

    base_path = getattr(
        sys,
        '_MEIPASS',
        os.path.dirname(os.path.abspath(__file__)))
    return os.path.join(base_path, relative_path)

canvas = Canvas(window, width=600, height=590)
canvas.pack()
path = resource_path0("vaultKnox.png")
img = ImageTk.PhotoImage(Image.open(path))
canvas.create_image(0, 0, anchor=NW, image=img)

def initIPObj():
    global ipdetails
    ipdetails = IPDetails()

initThread = threading.Thread(name='initThread', target=initIPObj,
                               daemon=True)
initThread.start()
```



```
# Inactivity thread
def active(window, mins):
    while True:
        secs = mins * 60
        time.sleep(secs)
        userAns = messagebox.askyesno("Inactivity notice", f"You have been using Securepass for {mins} minutes. "
                                     "\n\n Are you still using the application?")

        if userAns == True:
            continue
        else:
            window.destroy()
```

Inactivity thread:

The inactivity thread is instantiated once only when the application is first launched. The purpose of this thread is to create an IP details type object that will keep track of user activity within the application. The user settings table is retrieved from the local offline SQLite3 database by instantiating a UserSettings type object that will determine the configuration of the inactivity thread. The interval after which the user will be notified to confirm their activity is specified within the user settings. If the user confirms their activity the inactivity thread will be reset or otherwise if the user denies the activity the inactivity thread will terminate the main process, resulting in the overall termination of the application.

```
# Inactivity thread
cursor.execute("SELECT * FROM userSettings")
settings = cursor.fetchall()

if settings == []:
    userSettings = UserSettings(1, 300)
    cursor.execute("INSERT INTO usersettings(timeframe, inactivity) VALUES (?, ?)", (1, 5))
    db.commit()

else:
    userSettings = UserSettings(settings[0][1], settings[0][2])

b = threading.Thread(name='background', target=partial(active, window, userSettings.inactivity),
                    daemon=True)
b.start()

window.protocol("WM_DELETE_WINDOW", partial(on_closing, window))
```

Database self-destruct:

The reset master password mechanism contains a function that will detect whether a malicious user is currently opposing a brute force attack on the application. If the user fails to guess the recovery key after a total of five attempts, the local offline SQLite3 database is deleted from the user system. This mechanism protects the sensitive user data from access attacks of an identified malicious attacker.

```

def checkRecoveryKey():
    checked = getRecoveryKey()

    resetObj.enum()

    if resetObj.attempts == 1:
        resetObj.initTimer()

    if not resetObj.passAttempt:
        lbl1.config(text="Recovery request blocked. You are permitted 2 recovery attempts per minute.")

    elif resetObj.blocked:
        lbl1.config(text="You have exceeded recovery attempts.\nUser account and data will self destruct.")

        def delData():
            if os.path.exists("vaultKnox_passwords.db"):
                os.remove("vaultKnox_passwords.db")
            window.destroy()
            main()
        lbl1.after(2000, delData)

    else:
        if checked:
            resetObj.match()
            sql = "DELETE FROM masterpassword WHERE id = 1"
            cursor.execute(sql)

```

System attempting access from a different IP address:

If the system is connected to the internet a note of the system IP address is documented within the offline database. Once a user attempts to log into the password vault the application will determine whether the system is currently connected to the internet and if so, the system IP address corroborates with the previously configured address. If it is found that there is a discrepancy in these values, the user will be required to enter their recovery key as security measure to access their password vault from a different geographical location. This prohibits a malicious attacker from illicitly attaining the system of a user and attempting to login to the user password vault from a remote location. The malicious attacker cannot access the system without the recovery key in the user's possession.

```

cursor.execute("SELECT IP FROM userSettings")
IP = cursor.fetchall()[0][0]

if IP == None:
    if not ipdetails.IP == "Unknown":
        cursor.execute("UPDATE usersettings SET IP = ? WHERE id = ?",
            (hashPassword(ipdetails.IP.encode('utf-8')), 1))
        db.commit()
else:
    if not ipdetails.IP == "Unknown":
        encIP = hashPassword(ipdetails.IP.encode('utf-8'))
        if not IP == encIP:
            while True:
                userRecovery = simpleDialog.askstring("Logging in from a different IP address.", "The system IP address does not "
                    "match the configured IP of the password vault.\n\n"
                    "Please enter your recovery key to continue with the login process.\n")

                if userRecovery == None:
                    continue
                else:
                    def getRecoveryKey():
                        recoveryKeyCheck = hashPassword(userRecovery.encode('utf-8'))
                        cursor.execute("SELECT * FROM masterpassword WHERE id = 1 AND recoveryKey = ?",
                            [(recoveryKeyCheck)])
                        return cursor.fetchall()

```

Generate Random Password:

The user can generate a random password within the functionality of updating or creating a new password entry. The user will be prompted to specify whether the randomly generated password value should include special case characters, digits and/or uppercase characters. The user also specifies the password length they wish to generate.

```
if upperCaseV == 1 and digitsV == 1 and specialCharacterV == 1:
    for x in range(pwLength):
        ran = randint(1, 3)
        if ran == 1:
            ranPW += chr(randint(33, 64))
        elif ran == 2:
            charac = randint(97, 122)
            ranPW += chr(charac)
        else:
            ranPW += chr(randint(33, 126))
elif upperCaseV == 0 and digitsV == 1 and specialCharacterV == 1:
    for x in range(pwLength):
        if randint(1, 2) == 1:
            ranPW += chr(randint(33, 64))
        else:
            ranPW += chr(randint(91, 126))
elif upperCaseV == 0 and digitsV == 0 and specialCharacterV == 1:
    for x in range(pwLength):
        if randint(1, 3) == 1:
            ranPW += chr(randint(91, 126))
        elif randint(1, 2) == 1:
            ranPW += chr(randint(33, 47))
        else:
            ranPW += chr(randint(58, 64))
elif upperCaseV == 0 and digitsV == 0 and specialCharacterV == 0:
```

Update outdated password entries:

When the user logs in to the password vault the system will check the saved password entries to determine whether the current system time has passed any of the expiration dates of the password entries. Expired passwords are presented to the user for remediation. The user can also use the generate random password functionality to update the password value of a outdated password entry. The service name of the password entry is used to create a new browser session that will attempt to access the service whereafter the user can update the remediated password entry.

```

def passwordCheck():
    date = decrypt(array[i][4], encryptionKey).decode("utf-8")
    dateobj = datetime.strptime(date, '%Y-%m-%d %H:%M:%S.%F')
    outdated = dateobj + timedelta(days=(userSettings.timeframe * 30))
    currentDateTIme = datetime.now()

    if currentDateTIme > outdated:
        updatePW.append([])
        updatePW[len(updatePW)-1].append(array[i][0])
        updatePW[len(updatePW)-1].append(array[i][1])
        updatePW[len(updatePW)-1].append(array[i][2])
        updatePW[len(updatePW)-1].append(array[i][3])

passwordCheck()

i += 1

cursor.execute("SELECT * FROM passwordVault")
if len(cursor.fetchall()) <= i:
    break
if len(updatePW) > 0:
    y = 0
    outdatedEntry(updatePW, y)
    messagebox.showwarning("Passwords outdated",
        "You have outdated passwords, please update these values and the correlating service.")

```

System IP details:

The application utilizes the `Urllib.request` package to generate new get requests. A get request is created to determine the IP configuration details of the connected system. These details are saved within an `IPdetails` type object. Fields of the `IPdetails` object are retrieved when displaying the advertisement for the virtual private network affiliate link. Details of the IP address such as the city, country, region and organisation name are retrieved. The `socket` package is utilised to retrieve the local system name and local IP address. These values make up the title of the system IP details summary.

```

try:
    url = 'http://ipinfo.io/json'
    response = urlopen(url)
    data = json.load(response)

    try:
        IP = data['ip']
    except:
        IP = "Unknown"

    try:
        org = data['org']
    except:
        org = "Unknown"

    try:
        city = data['city']
    except:
        city = "Unknown"

    try:
        country = data['country']
    except:
        country = "Unknown"

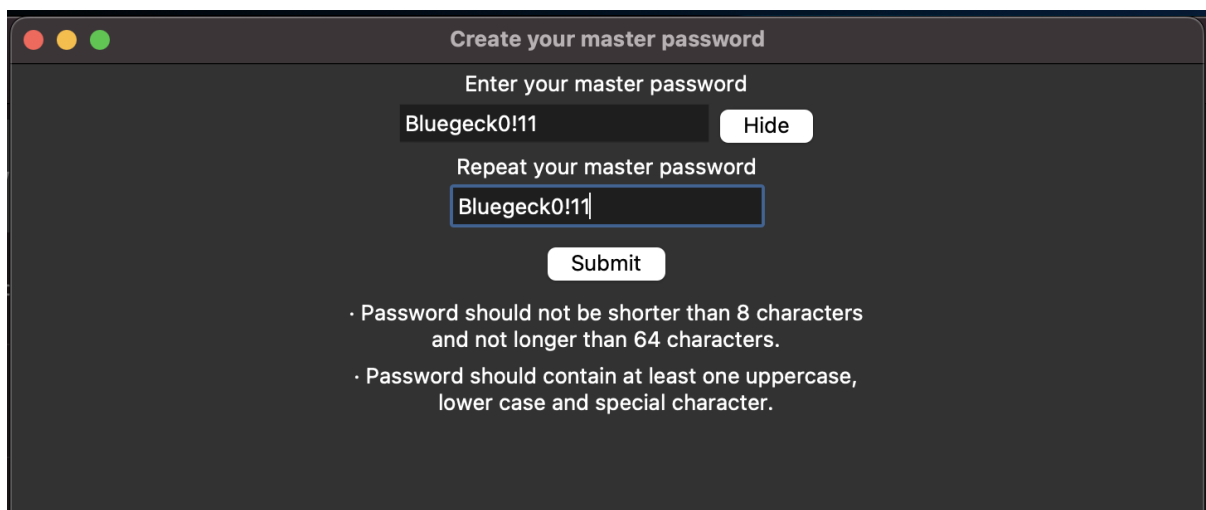
    try:
        region = data['region']
    except:
        region = "Unknown"

```

2.5. Graphical User Interface (GUI)

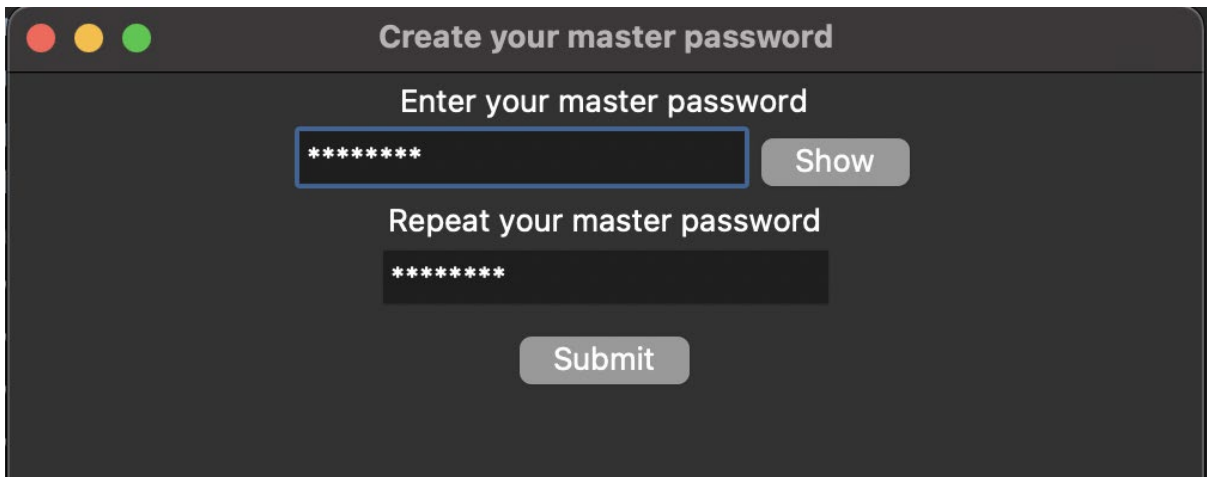
Create New Master Password Screen:

When the user runs the application for the first time and does not yet have a password vault or the user requests a new master password using a recovery key, the system will direct to the below screen. The user is prompted to create a new master password. The user can hide the character values of the entered master password in the text field as seen in the second figure. The user is required to repeat an identical password in each field that uses strong password protocols. The user is directed to the new recovery key screen if the submit request is successful.



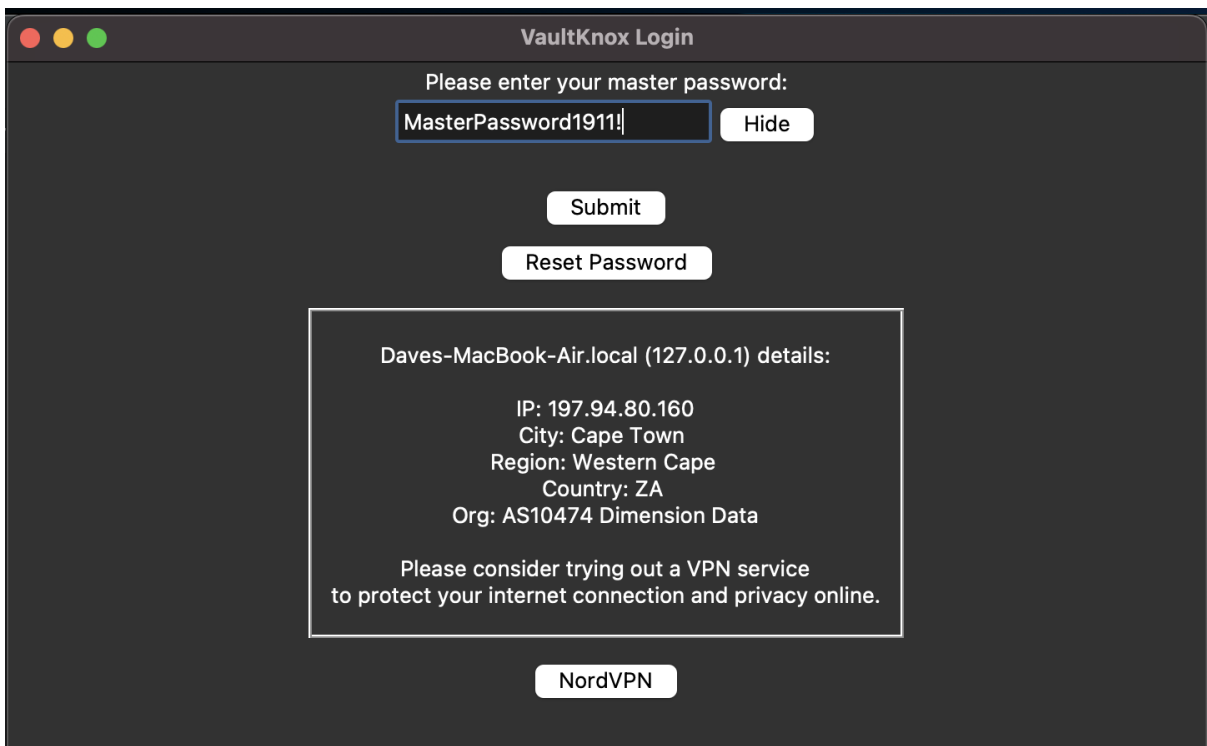
Create your master password
 Enter your master password
 Bluegeck0!11
 Repeat your master password
 Bluegeck0!11

- Password should not be shorter than 8 characters and not longer than 64 characters.
- Password should contain at least one uppercase, lower case and special character.



Master Password Login Screen:

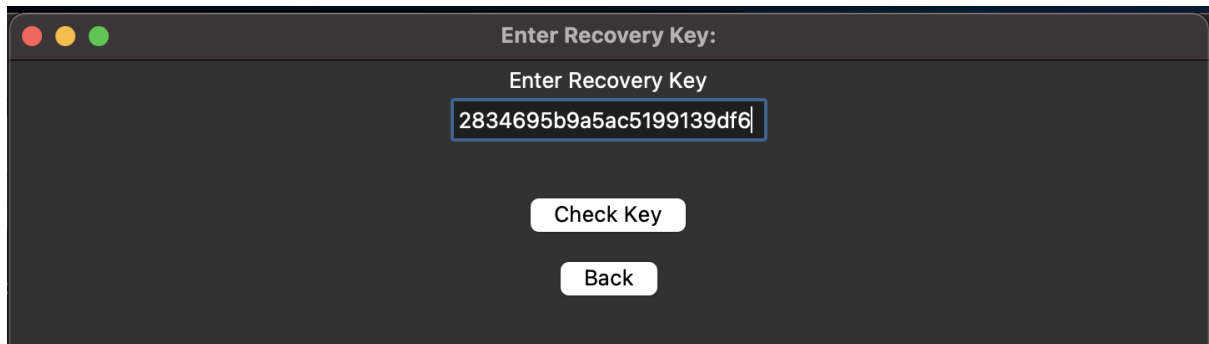
When the user has run the application more than once or has created a new master password the system will direct to the below screen. The user is prompted to enter their password vault master password. The user can hide the character values of the entered master password in the text field. The user is directed to the master password login screen if the submit request is successful and the entered master password matches the database entry. A user can also request to reset their password.



Recovery key screen:

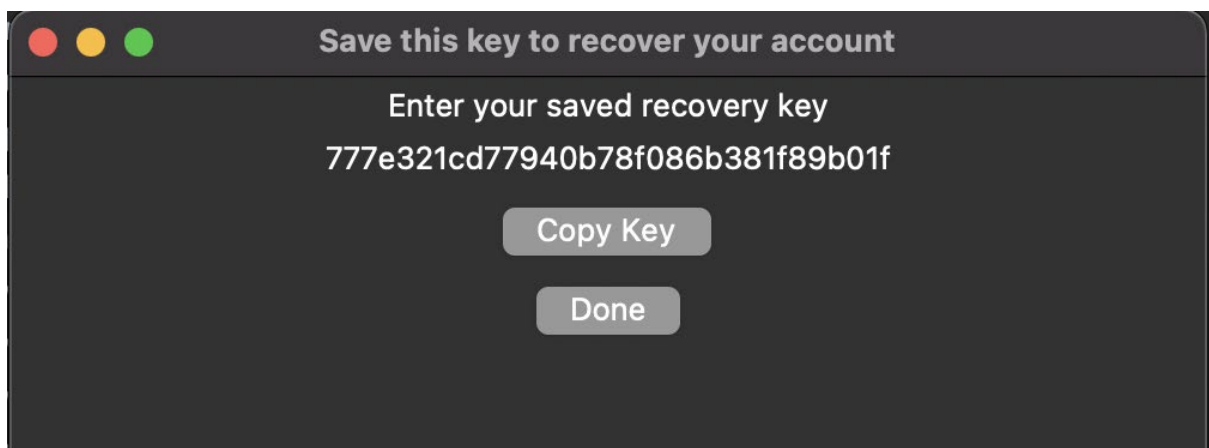
When the user presses the reset password button located on the master password login screen the system will direct to the below screen. The user is prompted to enter their password vault recovery key. The user is directed to the create new master password login

screen if the submit request is successful and the entered recovery key matches the database entry.



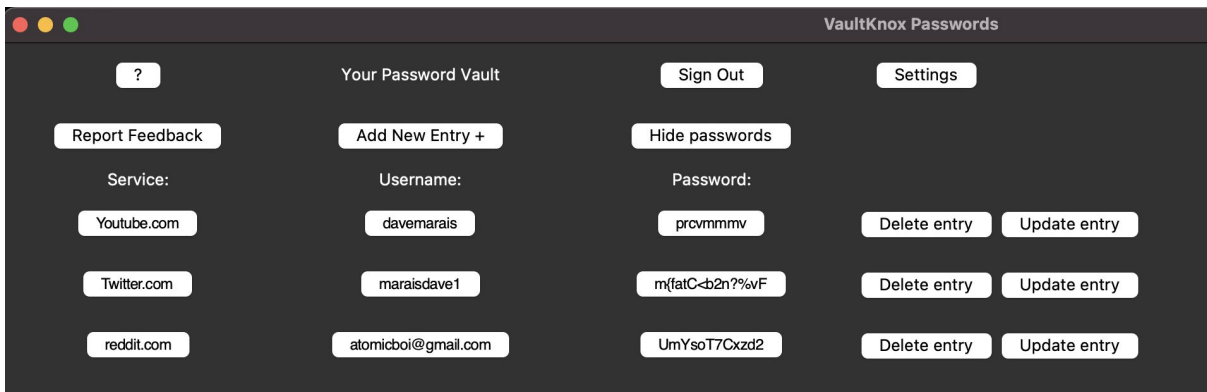
New Recovery Key Screen:

When the user has created a new master password the system will direct to the below screen. The system generates a new random recovery key. The user can press the copy key button to copy the key to their clipboard. After saving the copied recovery key the user can press the done button. The user is then directed to the master password login screen.



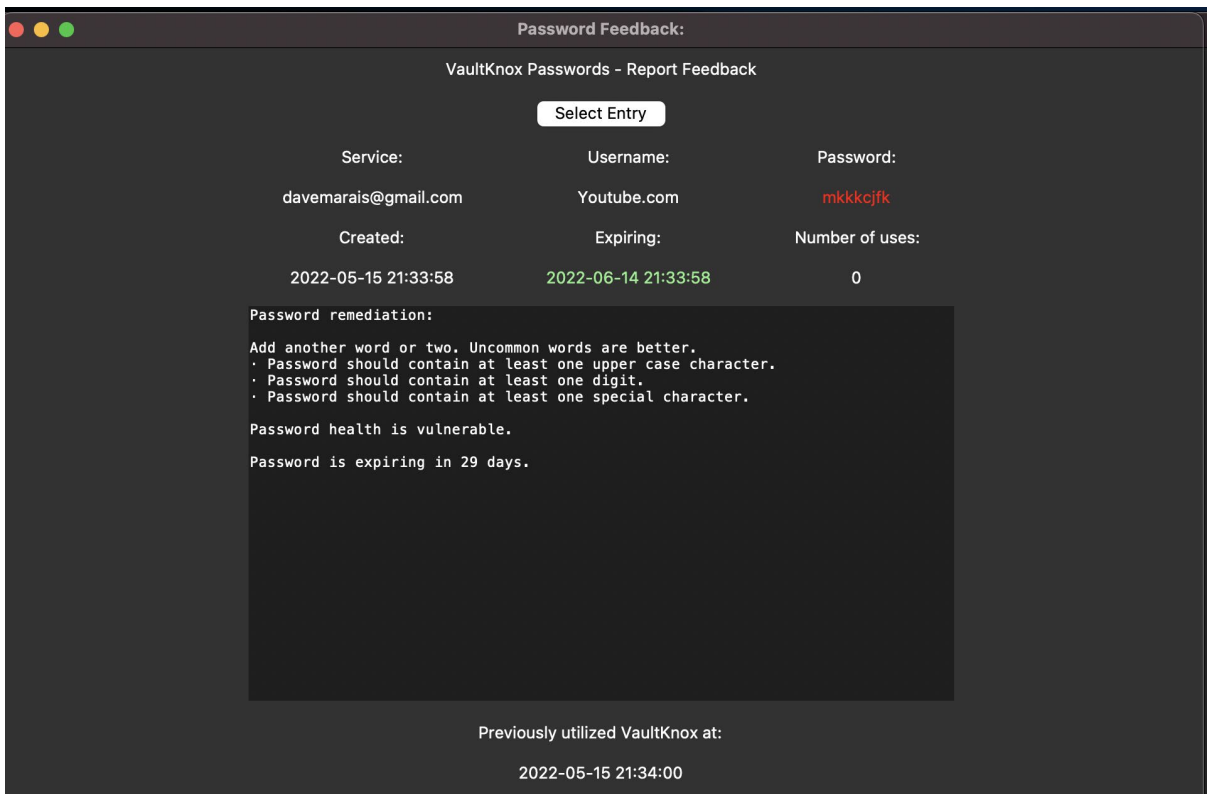
Password Vault Screen:

When the user has logged into the application from the master password login screen the application will direct to the below screen. The password vault is presented to the user. The user can add a new password entry, remove an existing password entry or update an existing password entry. The user can press the respective buttons: add entry, delete entry or update entry to initialize the before-mentioned operation.



Password Feedback Screen:

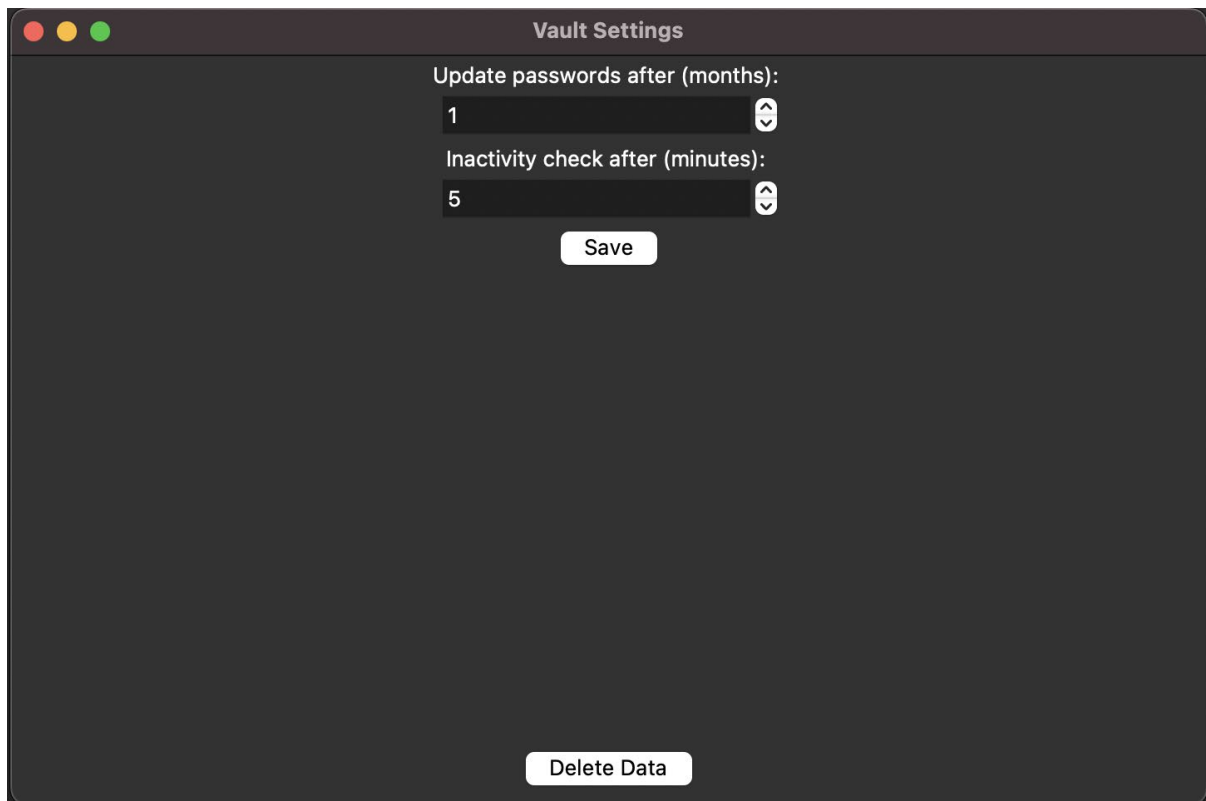
Users must choose a single password entry from a list of all entries fetched from their password vault on the password feedback screen. When a password entry is selected, the password feedback function examines the full password entry and generates a report for the user. The password entry, the password strength, the entry creation date, the entry expiration date, the number of uses of the entry, remediation if the password is found to have vulnerabilities, how many days until the password expires, and the last time the password vault was used are all included in the report.



Vault Settings Screen:

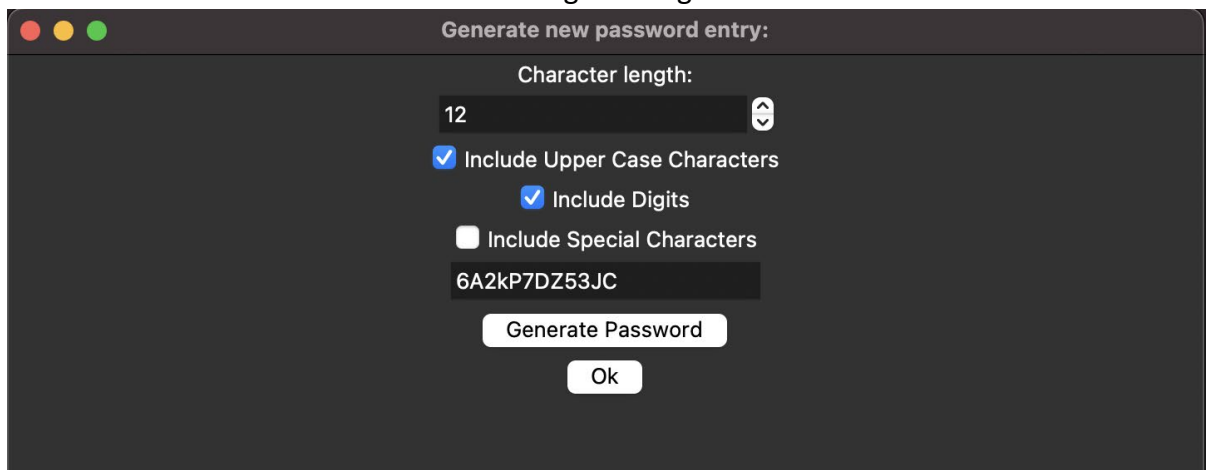
The Vault Settings Screen allows the user to configure the settings of the application. Settings include the ability to modify the period after which a password entry is classified as expired and outdated. Furthermore, users can modify the interval after which the inactivity notification will prompt the user to confirm the activity within the application. lastly the

user can press the delete data button to completely wipe all the user and account data from the system. This process will reset the application, requiring the user to create a new master password entry and vault that will be empty.



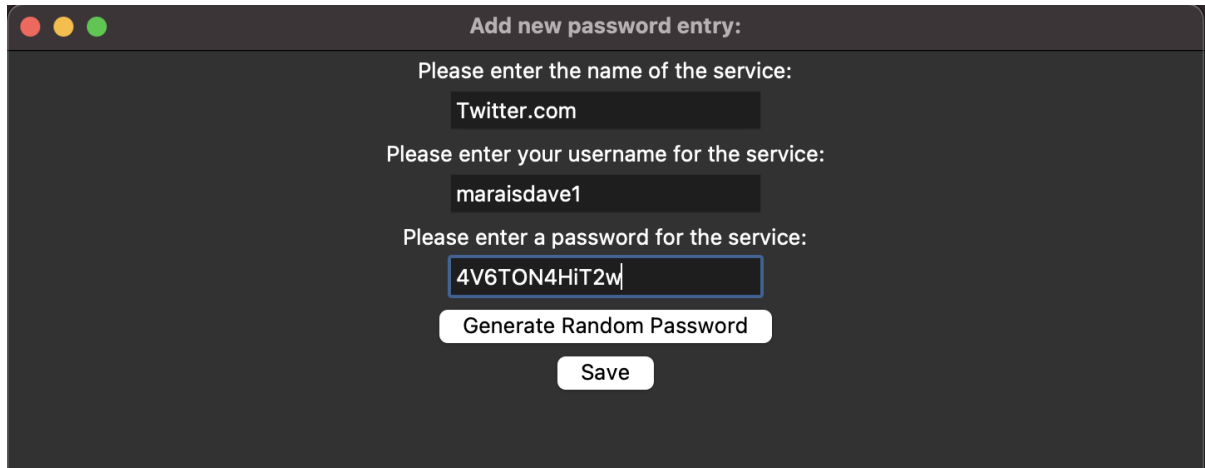
Generate Random Password Screen:

Within the feature of updating or establishing a new password entry, the user can generate a random password. The user will be asked if they want special case characters, numbers, or uppercase characters in the randomly generated password value. The user may also specify the length of the password they want to generate. Each of these values are controlled by the GUI widgets. The character length is specified by a scrollbox and the character types are controlled by the checkboxes. If the user presses the generate password button a password value will be created with their chosen widget configuration.



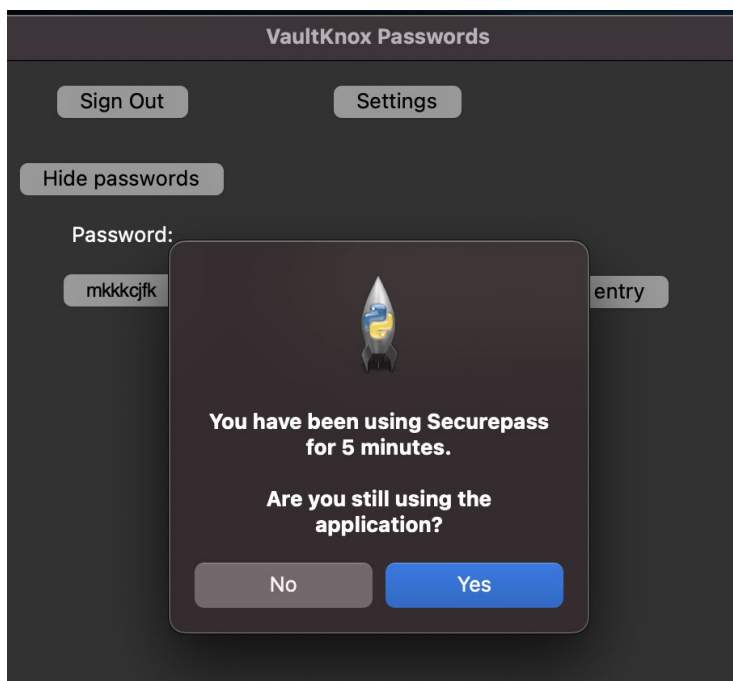
Add Password Screen:

A user has the ability to create a new password entry by clicking on the add password button and supplying the password service name, their username and the password. The application will notify the user if the entered service name does not produce a valid URL address. The user can also use the above-mentioned generate around a password feature to automatically generate the password value stored within the password field.



Inactivity Notification:

If the inactivity thread is executed the user will be prompted with the below notification. The user is asked to either confirm or deny whether they are still using the application. The inactivity thread will be reset if the user confirms the notification we are after they can continue using the application. If the user denies the notification the application will terminate.



2.6. Testing

The testing of the program was carried out in two stages:


1 - Manual testing in the form of exploratory testing. It was carried out throughout the development stage to ensure that each function performs as expected and that no unexpected output or actions are elicited. Following the implementation of a feature, testing will be performed to ensure that previously built features do not fail and that the desired output is produced exactly as intended.

Fellow students were involved during the manual testing phase by sending them the application to demo different functionalities. The objective of the users was to find bugs in the application by using the application for their own personal use to simulate user interaction in a live environment. They were also challenged to break the application to produce unwanted behaviour. This testing phase was carried out in a 3 day period and their results were documented. The undesired behaviour was then examined to determine which file was responsible for the error-prone code. Once the location of the code was determined, remediation of broken functionality was implemented. The fix of broken code was also documented. The results were evaluated as follows.

Tester Name: Shaurya Kumar

Tester Student ID: x18138284

Tester Course and specialization: BSc(Hons) in Computing, specializing in Cybersecurity

Issue	
Bug:	Multiple Inactivity Threads
Description:	I receive multiple, duplicate notifications when I am prompted to answer whether I am still using the application. The notifications execute one after the other. The number of notifications is variable.
Evaluation	
Location:	File: main.py Function: passwordVault() Line: 605-607 Screenshot:  <pre># Access user password vault def passwordVault(): for widget in window.winfo_children(): widget.destroy() b = threading.Thread(name='background', target=partial(active, window, userSettings.inactivity), daemon=True) b.start()</pre>
Description	Python uses spacing to differentiate the flow of code. In this case, the inactivity thread is launched within the for loop that checks the number of widgets on the current screen. For each widget, an inactivity thread is created. Furthermore, the inactivity mechanism is launched every time the user navigates back to their password vault.

Remediation	
Fix:	<p>File: main.py Function: main() Line: 128-138 Screenshot:</p>  <pre># Inactivity thread cursor.execute("SELECT * FROM userSettings") settings = cursor.fetchall() if settings == []: userSettings = UserSettings(1, 300) cursor.execute("INSERT INTO usersettings(timeframe, inactivity) VALUES (?, ?)", (1, 5)) db.commit() else: userSettings = UserSettings(settings[0][1], settings[0][2]) b = threading.Thread(name='background', target=partial(active, window, userSettings.inactivity), daemon=True) b.start()</pre>
Description:	<p>The inactivity thread was moved to the initialization stage of the application to ensure that only one instance is created when the application is first launched. Furthermore, the inactivity thread was also moved out of a for loop, preventing it from getting called multiple times. This solution also changes the inactivity check to start when the application is first launched and not when the password vault is accessed. This means that the check is also active when the user is in an idle state on the register and login screen.</p>

Tester Name: Bryan Rose

Tester Student ID: x18100180

Tester Course and specialization: BSc(Hons) in Computing, specializing in Software Development

Issue	
Bug:	Generate Random Password feature produces a string longer than the character length specified
Description:	When a password is generated and the special character, uppercase and digits checkboxes are selected, the character length of the password is 3 times longer than the character length specified in the scrollbox.
Evaluation	
Location:	<p>File: main.py Function: genPWScreen() Line: 1660-1668 Screenshot:</p>

	<pre> if upperCaseV == 1 and digitsV == 1 and specialCharacterV == 1: for x in range(pwLength): ranPW += chr(randint(33, 64)) for x in range(pwLength): charac = randint(97, 122) ranPW += chr(charac) for x in range(pwLength): ranPW += chr(randint(33, 126)) </pre>
Description	<p>For the configuration of generating a password value with all the checkbox selected, 3 separate for loops are generated that span the length of the password configured in the scrollbox. This leads to the program populating the password value with the different types of characters with the length of the total password for each. As a result, the password value is 3 times longer than is specified in the scrollbox. The 3 For loops generate and add 3 characters of different types for each iteration that spans the length specified of the password.</p>
Remediation	
Fix:	<p>File: main.py Function: genPWScreen() Line: 1660-1669 Screenshot:</p> <pre> if upperCaseV == 1 and digitsV == 1 and specialCharacterV == 1: for x in range(pwLength): ran = randint(1, 3) if ran == 1: ranPW += chr(randint(33, 64)) elif ran == 2: charac = randint(97, 122) ranPW += chr(charac) else: ranPW += chr(randint(33, 126)) </pre>
Description:	<p>This error was fixed by correctly delegating a single for loop that spans only the total length of the password characters specified. For each iteration of the loop, the program will randomly select a character type and add a random type of the character to the password value. The password value will only span the length specified, by randomly selecting a character type of the 3 options at a time.</p>

Issue	
Bug:	Variability in initializing IP details
Description:	Sometimes the IP details used for advertising the VPN solution as an affiliate link

	do not display on the loginscreen. The console displays that the socket request fails. The application continues execution without the above feature. The error occurs indiscriminately to environment variables.
Evaluation	
Location:	File: IP.py Function: __init__(self) Line: 6-10 Screenshot: <pre>class IPDetails: def __init__(self): hostname = socket.gethostname() localIP = socket.gethostbyname(hostname)</pre>
Description	The socket library is employed to determine the system hostname as well as the system host IP address. An IPDetails object is initialized when the application navigates to the loginscreen. This bug is difficult to replicate with the normal process flow as its execution is dependent on the user's internet connection speed and system specifications. By disabling the system internet connection the error can be reproduced because the socket library can not request the host details if the system is not online. This will produce an error prompting that the socket library failed to make the request. The error terminates the entire process flow of initializing the IPDetails object. Consequently, the program does not load and display the VPN affiliate feature to the user.
Remediation	
Fix:	File: IP.py Function: __init__(self) Line: 6-15 Screenshot: <pre>class IPDetails: def __init__(self): try: hostname = socket.gethostname() except: hostname = "Unknown" try: localIP = socket.gethostbyname(hostname) except: localIP = "Unknown"</pre>
Description:	To fix this bug, both socket requests are relocated within a try-except block. Provision is made for occasions when the loginscreen is executed before a valid hostname and correlating IP address can be determined. The socket request is tried, if the socket library can not determine the credentials for the system

hostname and IP address, the process flows delegates to the except clause, the hostname or localIP variable is consequently configured to the "Unknown" string variable. If the loginscreen is displayed before the loginscreen is executed, the application will replace these values with the Unknown value. The error prompted to the console is thus resolved and can not produce output displayed to the user.

2 - Automated testing - I utilized a python unit testing library called unittest. This enables developers to construct unit tests to see if functions perform as intended and produce accurate results. Unit tests aid in the automation of test cases, ensuring that functionality is tested correctly and that any modifications or new features added to the project do not create mistakes in other parts of the program, resulting in the failure of the started unit tests. When it comes to documenting modifications and precisely changing existing logic, it's an invaluable tool. Furthermore, integration tests can be carried out by specifying and running tests from the command line that function incongruancy with eachother. This ensures that the results of tests are documented in an environment rather than a vacuum. Finally all the tests located in the test class is run as the final End to End test, that will document whether the entire application functions as intended. Passing the End to End test, requires all tests to be satisfied.

Unit Testing Data Processing Calculations:

Testing the security protocols for this application is the top priority. This being said, the first test case is for the encryption protocol used throughout the data processing workflows. All data saved within the local offline database file is encrypted and non-human readable. A correctly implemented SHA-512 encrypted value is converted to a 64 bytes value. In the test_encryptionProtocol a random string is supplied. It is then encoded into a UTF-8 byte format. The testBytes variable can then be hashed using the hashPassword method. If implemented correctly, the method returns a 64-byte byte value of 128 characters. Using the assertEquals test method we check that the returned hash value is equal to 128 characters. If so, the test case will produce a successful response.

```

import unittest

from Hashing import *
from Encryption import *
from Masterpassword import MasterPassword
from Passwordentry import PasswordEntry

class TestCalc(unittest.TestCase):

# Valid SHA-512 encrypted value is 64-bytes or 128 characters long
def test_encryptionProtocol(self):
    testString = "TestString123"
    testBytes = testString.encode('utf-8')
    self.assertEqual(len(hashPassword(testBytes)), 128)

```

Secondly, we test the variable type for the returned values of both the encryption and decryption methods. The Fernet cryptography library accepts an encryption key value and the message to be encrypted as input. The testMessage variable serves as the value of the user-supplied input which is an encrypted SHA-512 value. It is then encoded to UTF-8 byte format. The testToken variable serves as the encryption key value derived from the user's master password. It is also converted to UTF-8 byte format. The encrypt method is called and both values are passed as the encryption key and message respectively. If the encryption method is correctly implemented the encrypted byte value is returned. The isinstance test method is used to verify whether the output of the encrypt method is of the byte variable type. The test case will produce a successful response for accurately encryption with the encryption key of message values.

```

# Valid encrypted output type is in bytes
def test_encryptionOutputType(self):
    testMessage = "qAAAAABiTGs3XYwhs_Jqu-2jii97Hzpyh1vQWP0Q95K8xfqRegg8DaYtNdMnSxnYPZ0em0d_DM5RHTJkDki-VRNXMj5Wn6BFLq=="
    testBytesMessage = testMessage.encode('utf-8')
    testToken = "Q9jeh5C4IBuAGa78o0kXVChgBxE_qV3Fi07HJyDaWPA="
    testBytesToken = testToken.encode('utf-8')
    self.assertIsInstance(encrypt(testBytesMessage, testBytesToken), bytes)

# Valid decrypted output type is in bytes
def test_decryptionOutputType(self):
    testMessage = "qAAAAABiTGs3XYwhs_Jqu-2jii97Hzpyh1vQWP0Q95K8xfqRegg8DaYtNdMnSxnYPZ0em0d_DM5RHTJkDki-VRNXMj5Wn6BFLq=="
    testBytesMessage = testMessage.encode('utf-8')
    testToken = "Q9jeh5C4IBuAGa78o0kXVChgBxE_qV3Fi07HJyDaWPA="
    testBytesToken = testToken.encode('utf-8')
    self.assertIsInstance(decrypt(testBytesMessage, testBytesToken), bytes)

```

Vice versa, The second testMessage variable serves as the value of the run time supplied input which is stored in encrypted SHA-512 format. It is then encoded to UTF-8 byte format. The testToken variable serves as the encryption key value derived from the user's master

password. It is also converted to UTF-8 byte format. The decrypt method is called and both values are passed as the token and message respectively. If the decryption method is correctly implemented the decrypted byte value is returned. The `assertIsInstance` test method is used to verify whether the output of the decrypt method is of the byte variable type. The test case will produce a successful response for accurately decryption with the token of message values.

The user-supplied master password value is only hashed to SHA-512 format and is stored in the database if the validation parameters are satisfied. Remediation of unsatisfied parameters is saved within the master password object in the WASP (with advanced secure programming principles) field. In the below test case the `testMasterPassword` variable serves as the user-supplied master password input. It is passed as the value parameter for the `MasterPassword` object and saved to the `testMasterPasswordObject` variable instance. The `MasterPassword` is initialised and populated with possible remediation suggested. If all validation parameters are satisfied the WASP field containing the list of remediations will remain empty. In the `assertEqual` test case, we check that the WASP list field is empty to signify that the master password is accepted for encryption. The test case will produce a successful response if the WASP list field remains empty, suggesting no further remediation is required.

```
# Valid master password has no remediation list items
def test_masterPasswordObject(self):
    testMasterPassword = "ThisisaTestMP99101!"
    testMasterPasswordObject = MasterPassword(testMasterPassword, [])
    self.assertEqual(len(testMasterPasswordObject.WASP), 0)
```

Likewise to the masterpassword value, the user-supplied password entry values are only hashed to SHA-512 format and stored in the database if all the validation parameters are satisfied. Unsatisfied parameters will save the encrypted fields of the password entry object to empty string values. In the below test case the input parameters of the `testPasswordEntryObject` serve as the user-supplied password entry fields. They are passed as the service, username and password parameters for the `PasswordEntry` object and saved to the `testMasterPasswordObject` variable instance. The `PasswordEntry` is initialised and populated with the parameters and if successful encrypted using the `encryptionKey`. If all validation parameters are satisfied by the encryption fields, the 3 respective parameters are populated with the converted encrypted values. In the `assertNotEqual` test case, we check that each encrypted field of the `testPasswordEntryObject` is not an empty string, signifying that the validation parameters have been met. The test case will produce a successful response if all the encrypted fields are not equal to an empty string, suggesting the

password entry was accepted.

```
# Valid password entry with expected values and encryptionKey has encrypted fields
def test_passwordEntryObject(self):
    testService = "Youtube.com"
    testUsername = "davemarais@gmail.com"
    testPassword = "WihsbEIL!"
    testdate = "2022-05-10 20:06:12.319494"
    testEncryptionKey = b'Q9jeh5C4IBuAGa78o0kXVChgBxE_qV3Fi07HJyDaWPA='
    testPasswordEntryObject = PasswordEntry(testService, testUsername, testPassword, testdate, testEncryptionKey)
    self.assertNotEqual(testPasswordEntryObject.encservice, "")
    self.assertNotEqual(testPasswordEntryObject.encusername, "")
    self.assertNotEqual(testPasswordEntryObject.enccpassword, "")
    self.assertNotEqual(testPasswordEntryObject.enccdate, "")
```

Master Password Integration Testing:

```
class TestMasterPassword(unittest.TestCase):
    def test_masterPasswordWASP(self):
        testValue = "ThisisaTestMP99101!"
        testWASP = []
        testMasterPasswordObject = MasterPassword(testValue, testWASP)
        self.assertIsInstance(testMasterPasswordObject.WASP, list)

    def test_masterShortPasswordValue(self):
        shortValue = "ThMP12!"
        testMasterPasswordObject = MasterPassword(shortValue, [])
        self.assertEqual(testMasterPasswordObject.WASP[0], ". Password should not be shorter than 8 characters.")

    def test_masterLongPasswordValue(self):
        longValue = "ThMP12!hasgdhskagdhsgsdgsvdghgasdasidydsdqasidydsadisavgdaysugdasdydvvasgdaysgdviaus"
        testMasterPasswordObject = MasterPassword(longValue, [])
        self.assertEqual(testMasterPasswordObject.WASP[0], ". Password should not be longer than 64 characters.")

    def test_masterUpperCasePasswordValue(self):
        noLowerCaseValue = "THMP1234!"
        testMasterPasswordObject = MasterPassword(noLowerCaseValue, [])
        self.assertEqual(testMasterPasswordObject.WASP[0], ". Password should contain at least one lower case character.")

    def test_masterLowerCasePasswordValue(self):
        noUpperCaseValue = "hjdSa1234!"
        testMasterPasswordObject = MasterPassword(noUpperCaseValue, [])
        self.assertEqual(testMasterPasswordObject.WASP[0], ". Password should contain at least one upper case character.")
```

The Test master password class contains 7 functions that each check the remediation of differently configured passwords values. Each string is set up in a way to trigger the function assert operator. If the test value creates and appends to the list field object, it indicates the test to be successful.

```
def test_masterNoDigitPasswordValue(self):
    noDigitsValue = "hjdSaUHDSA!"
    testMasterPasswordObject = MasterPassword(noDigitsValue, [])
    self.assertEqual(testMasterPasswordObject.WASP[0], ". Password should contain at least one digit.")

def test_masterNoSpecialPasswordValue(self):
    noSpecialCharacterValue = "hjdSaUHDSA123"
    testMasterPasswordObject = MasterPassword(noSpecialCharacterValue, [])
    self.assertEqual(testMasterPasswordObject.WASP[0], ". Password should contain at least one special character.")
```


Unit Testing Settings:

```
class TestUserSettings(unittest.TestCase):
    def test_settingsFieldType(self):
        testTimeframe = 1
        testInactivity = 5
        testUserSettingsObject = UserSettings(testTimeframe, testInactivity)
        self.assertIsInstance(testUserSettingsObject.timeframe, int)
        self.assertIsInstance(testUserSettingsObject.inactivity, int)

    def test_settingsFieldValues(self):
        testTimeframe = 10
        testInactivity = 50
        testUserSettingsObject = UserSettings(testTimeframe, testInactivity)
        self.assertEqual(testUserSettingsObject.timeframe, 10)
        self.assertEqual(testUserSettingsObject.inactivity, 50)
```

The user settings class contains two functions that each determine different values for the accepted user-supplied parameters. One of the prerequisites is that the parameters should be of type integer and secondly that the correct value supplied is stored within the user settings object.

IP Integration Testing:

```
class TestOnlineSystem(unittest.TestCase):
    def test_SystemInternetConnection(self):
        testIPObj = IPDetails()
        self.assertNotEqual(testIPObj.IP, "Unknown")
        self.assertNotEqual(testIPObj.org, "Unknown")
        self.assertNotEqual(testIPObj.city, "Unknown")
        self.assertNotEqual(testIPObj.country, "Unknown")
        self.assertNotEqual(testIPObj.region, "Unknown")

    def test_IP(self):
        testIPObj = IPDetails()
        self.assertIn(".", testIPObj.IP)

class TestOfflineSystem(unittest.TestCase):
    def test_OfflineSystemInternetConnection(self):
        testIPObj = IPDetails()
        self.assertEqual(testIPObj.IP, "Unknown")
        self.assertEqual(testIPObj.org, "Unknown")
        self.assertEqual(testIPObj.city, "Unknown")
        self.assertEqual(testIPObj.country, "Unknown")
        self.assertEqual(testIPObj.region, "Unknown")

class TestLocalSystem(unittest.TestCase):
    def test_LocalSystemDetails(self):
        testIPObj = IPDetails()
        self.assertNotEqual(testIPObj.localIP, "Unknown")
        self.assertNotEqual(testIPObj.hostname, "Unknown")
```

The IPClass is tested by 3 separate classes. The first class checks whether a system that is connected to a valid internet connection produces known credential values for each of the IP details object fields. If a valid credential is found it will not be set to the Unknown variable. Vice-versa the test offline system class will test a system that does not have an active internet connection

to determine whether the IP details object Fields are set to the Unknown variable. lastly the test local system class will check whether the socket Library can determine the local credentials of the system that is currently running the Vault Knox password manager application. if valid credentials are found the fields of the IP object will not be said to unknown.

Encryption Integration Testing:

```
class TestEncryption(unittest.TestCase):
    def test_EncryptionMechanism(self):
        testMessage = "Unencrypted test message"
        testBytesMessage = testMessage.encode('utf-8')
        testToken = "Q9jeh5C4IBuAGa78o0kXVChgBxE_qV3Fi07HJyDaWPA="
        testBytesToken = testToken.encode('utf-8')
        encryptedValue = encrypt(testBytesMessage, testBytesToken)
        decryptedValue = decrypt(encryptedValue, testBytesToken).decode('utf-8')
        self.assertEqual(decryptedValue, testMessage)

    def test_DecryptionMechanism(self):
        testMessage = "qAAAAABiqTigTtpJkqSxrBzI_QIPpNajk_DyHQ6mipJHVHiv-gJ17z3niB-gqiB0VzD"
        testBytesMessage = testMessage.encode('utf-8')
        testToken = "Q9jeh5C4IBuAGa78o0kXVChgBxE_qV3Fi07HJyDaWPA="
        testBytesToken = testToken.encode('utf-8')
        decryptedValue = decrypt(testBytesMessage, testBytesToken)
        encryptedValue = encrypt(decryptedValue, testBytesToken)
        decryptedValue = decrypt(encryptedValue, testBytesToken).decode('utf-8')
        self.assertEqual(decryptedValue, "Unencrypted test message")
```

In the test encryption class two functions are created. The first function tests the mechanism of the encryption protocol to determine whether an encrypted message will produce the correct encrypted response. The assertequals operator checks that when the test message is encrypted and thereafter decrypted, it will produce the same original test message string. This test is justified as an integration test because the encryption and decryption protocols are dependent upon each other and require the correct functioning of both. The second function test the decryption mechanism in the same fashion by starting with an encrypted string value which is decrypted, encrypted and once again decrypted thereafter to compare with the expected decrypted value. This test is also justified in a similar fashion as an integration test.

Hashing Unit Testing:

```
class TestHashing(unittest.TestCase):
    def test_HashingMechanism(self):
        testMessage = "Unhashed test message"
        encodedMessage = testMessage.encode('utf-8')
        hashedMessage = hashPassword(encodedMessage)
        self.assertEqual(hashedMessage, "8c15e014010818050415275204aa0779628bfe6ad5d543979")
```

The Test hashing class will check whether the unhashed test message produces a valid sha512 variable. the expected variable as compared to the test message that is hashed. The hashed value is encoded to utf-8 format.

End to End Application Testing:

```
Testing started at 21:59 ...
Launching unittests with arguments python -m unittest Testing in /Users/atomicboi/PycharmProjects/securePass

Process finished with exit code 0

Ran 20 tests in 1.188s

OK
```

In the above-mentioned screenshot all the tests specified and instantiated within the testing class is called from the command line to serve as the end-to-end application testing protocol. As seen in the screenshot all of the 20 tests that run for a duration of 1.2 seconds were completed and pass successfully indicating that the application is fully functional and operational as expected. If one test were to fail it would indicate that the end-to-end application testing protocol did not pass and functionality has produced errors when the testing was run while configured with the end to end environment of the entire application.

3.0 Conclusions

Coding Vault Knox password manager has been the most daunting project I have challenged myself to tackle. There are many reasons for this, but at the top of the list, it is the fact that I chose to develop it with a completely Python-based approach. I was interested in Python development after researching its flexibility, ease of use and popularity. I saw it fit that for my final year project I would challenge myself by learning a completely novel language, that if mastered, would prove fruitful in real-world applications and scenarios. Secondly, it was a priority to make a password manager that was entirely independent of user system specifications, environmental access and technical savvy. To facilitate this, I realized early on that I would have to create all functionality from the bottom up, independent of frameworks and prebuilt solutions. Environmental factors of users such as location and access to the internet were considered and consequently, I made the decision to make the application with a 100 percent offline approach and develop functionality to complement security in a local environment without sacrificing application robustness or quality. The tradeoff between functionality that online services could provide and the availability and security of an offline approach were extensively analysed in this case.

The advantage of my comprehensive approach is that I learnt extensively about fundamental security concepts (specific to my specialization) such as hashing and

encryption. I designed and implemented the entire cryptography utilized in the application and as such have a high level of understanding of these concepts. Designing security protocols requires the developer to have a fundamental scope of security and ensure protocols and practices. At the time of its development, Vault Knox was designed with the strongest encryption algorithm namely, SHA 512 at my disposal.

The disadvantage of my approach is that utilizing an established framework for sophisticated security concepts such as encryption and data processing would cut down considerably on development time. Frameworks are additionally also tried and tested through rigorous testing protocols. It can be the case that utilizing such services would better protect against unexpected behaviour. It is the nature of online vulnerabilities to be everchanging and evolving, by delegating security to a third party it makes it more manageable to maintain up to date security protocols since these frameworks are continuously updated to improve their efficiency. Such an approach would enable the developer to focus on more pressing matters specific to the application use case and functionality, rather than accounting for unknown variables with each iteration of the application update.

In conclusion, I really pushed myself in terms of the expectations of the project and executing the planned deliverables. It can be said with conviction that, Vault Knox exceeds the application of industry offline password managers when comparing functionality and ease of use. Thus, the objectives laid out in the project proposal have been met. The strengths of the project include that it is extremely lightweight and efficient in terms of utilizing the system resources required to run the application. This means it can perform well on any system and maintain a robust idle state as a background application indefinitely. Modern-day vulnerability mechanisms of SQLi and Brute force attacks are accounted for and meet the security expectations of a reliable password manager. In its current state, the project can be deployed and function in real-world applications and environments.

4.0 Further Development or Research

In the case of extending the development timeframe of this project, I would choose to further refine the application process flow and cosmetics. I would completely overhaul the Graphical User Interface package, namely Tkinter to a more flexible and modern framework. Determined approach to configuration to propose an error-free process flow from different screens with navigation and functionality specific to the currently configured interface scope. There is great potential in improving the overall appearance and ergonometry of the application with an influx of time. Vault Knox in its current state is also specific to the desktop environment, restricting its use case from popular mobile devices such as smartphones and tablets. Fortunately, Python has a very flexible application to different system environments and with the implementation of mobile frameworks such as Kivy can

be utilized to develop a mobile version of the application with an increase in the available development timeframe.

With additional resources, it would allow the project to contain external API services that could further improve security and functionality. Such an example is the HIBP API which can determine whether user credentials have been compromised through a data leak by searching through credentials of known database repositories. Such services either charge a monthly subscription fee (HIBP = \$3.50/month) or charge per use case. Scaling such an application with an active user base would require additional resources. Other avenues such as marketing and deployment strategies could also be pursued to increase the exposure of the application to the public and increase revenue generated through in application affiliate purchases, such as the VPN service advertised on the login screen. Monetary resources can also allow for contracting an independent third party for stringent penetration and security testing specifically to the application before its deployment to a live environment.

5.0 References

Docs.python.org. 2021. *hashlib — Secure hashes and message digests — Python 3.10.0 documentation*. [online] Available at: <<https://docs.python.org/3/library/hashlib.html>> [Accessed 7 November 2021].

Cryptography.io. 2021. *Key derivation functions — Cryptography 36.0.0.dev1 documentation*. [online] Available at: <<https://cryptography.io/en/latest/hazmat/primitives/key-derivation-functions/#pbkdf2>> [Accessed 7 November 2021].

Teamgantt.com. 2021. *Online Gantt Chart Maker - Try for Free! | TeamGantt*. [online] Available at: <<https://www.teamgantt.com/>> [Accessed 7 November 2021].

Docs.python.org. 2021. *unittest — Unit testing framework — Python 3.10.0 documentation*. [online] Available at: <<https://docs.python.org/3/library/unittest.html>> [Accessed 7 November 2021].

Draw.io. 2021. *Flowchart Maker & Online Diagram Software*. [online] Available at: <<https://draw.io/>> [Accessed 7 November 2021].

PyPI. 2021. *auto-py-to-exe*. [online] Available at: <<https://pypi.org/project/auto-py-to-exe/>> [Accessed 7 November 2021].

Sqlite.org. 2021. *SQLite Home Page*. [online] Available at: <<https://sqlite.org/index.html>> [Accessed 7 November 2021].

Adobe.com. 2021. *Free Logo Maker: Design Custom Logos | Adobe Spark*. [online] Available at: <<https://www.adobe.com/express/create/logo>> [Accessed 7 November 2021].

Selenium-python.readthedocs.io. 2022. *Selenium with Python — Selenium Python Bindings 2 documentation*. [online] Available at: <<https://selenium-python.readthedocs.io/>> [Accessed 12 May 2022].

Python, R., 2022. *Working With JSON Data in Python – Real Python*. [online] Realpython.com. Available at: <<https://realpython.com/python-json/>> [Accessed 12 May 2022].

Python, R., 2022. *Socket Programming in Python (Guide) – Real Python*. [online] Realpython.com. Available at: <<https://realpython.com/python-sockets/>> [Accessed 12 May 2022].

Python, R., 2022. *Python's urllib.request for HTTP Requests – Real Python*. [online] Realpython.com. Available at: <<https://realpython.com/urllib-request/>> [Accessed 12 May 2022].

Python, R., 2022. *Python GUI Programming With Tkinter – Real Python*. [online] Realpython.com. Available at: <<https://realpython.com/python-gui-tkinter/>> [Accessed 12 May 2022].

Python, R., 2022. *Image Processing With the Python Pillow Library – Real Python*. [online] Realpython.com. Available at: <<https://realpython.com/image-processing-with-the-python-pillow-library/>> [Accessed 12 May 2022].

Python, R., 2022. *An Intro to Threading in Python – Real Python*. [online] Realpython.com. Available at: <<https://realpython.com/intro-to-python-threading/>> [Accessed 12 May 2022].

W3schools.com. 2022. *Python Requests Module*. [online] Available at: <https://www.w3schools.com/python/module_requests.asp> [Accessed 12 May 2022].

GeeksforGeeks. 2022. *Pyperclip module in Python - GeeksforGeeks*. [online] Available at: <<https://www.geeksforgeeks.org/pyperclip-module-in-python/>> [Accessed 12 May 2022].

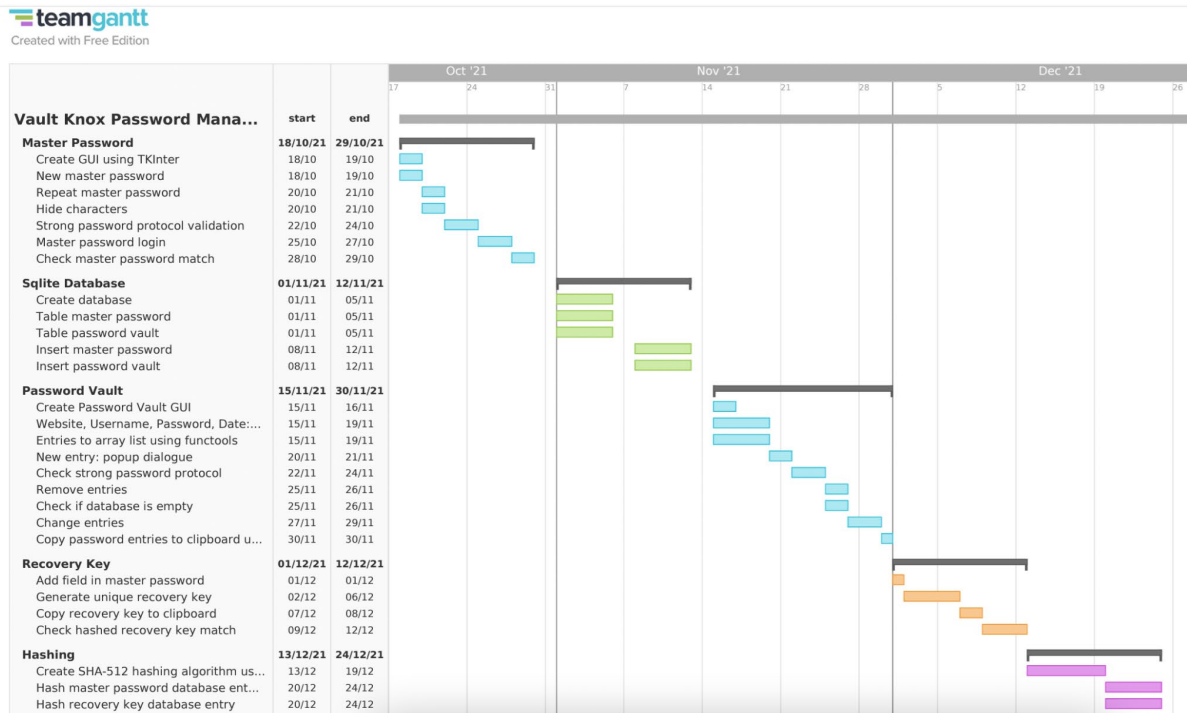
Marais, D., 2022. *Vault Knox Project Board*. [online] Trello.com. Available at: <<https://trello.com/b/dhfm9B3/vaultknox>> [Accessed 12 May 2022].

Pythonlang.dev. 2022. *Zxcvbn Python - Python Repo*. [online] Available at: <<https://pythonlang.dev/repo/dwolfhub-zxcvbn-python/>> [Accessed 12 May 2022].

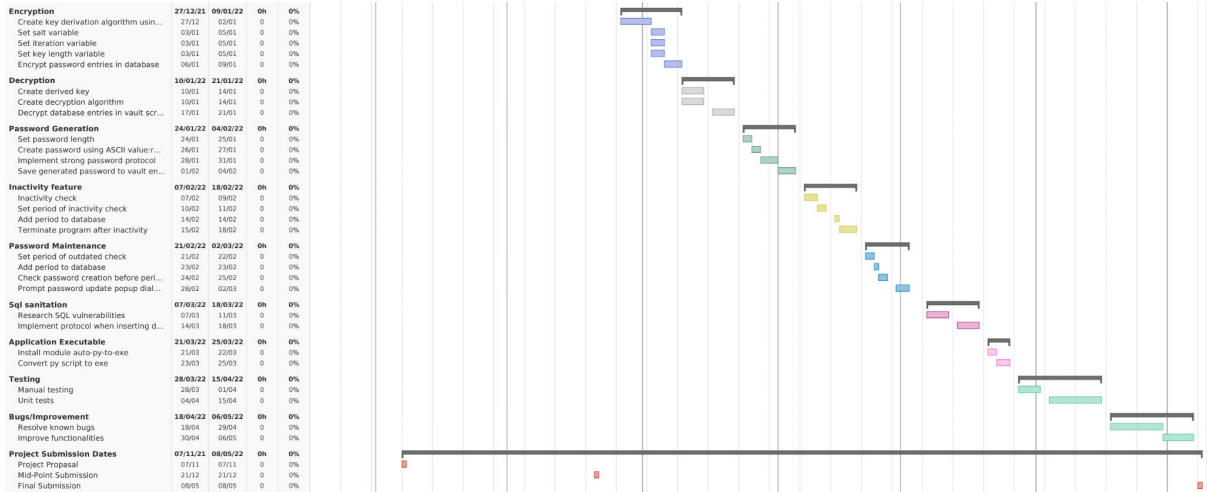
6.0 Appendices

6.1. Project Plan

Mid Point Project Plan



Final Project Plan



6.2. Project Proposal

National College of Ireland

Project Proposal

Vault Knox Password Manager

03/11/2021



Bsc(Hons) in Computing

Cybersecurity

2021/2022

Dawid Marais

x18133282

x18133282@student.ncirl.ie

Contents

Objectives	74
Background	74
State of the Art	75
Technical Approach	76
Technical Details	77
Special Resources Required	78
Project Plan	78
Testing	81
Bibliography	81

1.0 Objectives

The objective of this project is to create a secure offline password manager that:

- will utilize a decentralized encrypted offline database to maintain and store passwords that utilize strong password protocols
- can generate secure random passwords that satisfy strong password protocols.
- will facilitate maintenance of security through continuous report feedback, inactivity termination and update of outdated or compromised passwords
- can be utilized for all offline and online applications, wherever passwords are used for user profiles.
- will maintain secure and fail-safe recovery of credentials using two-factor authentication
- will facilitate security using leading industry protocols for hashing, encryption, decryption and true random value generation.
- places the responsibility of the extent to which security of passwords is applied, on the user.
- will prioritise functional simplicity and ease of use regardless of the users' technical proficiency.
- will mitigate the threat exposure to the overall application system, by implementing secure application programming and principles.

2.0 Background

For my third-year internship, I interned as a cyber security analyst for a security company called BCC Edgescan. During my employment, we used a password manager called KeePass which served as our basic password manager. After my experience with this program, I saw the value that it contained but I felt like I could further improve on the functionality it provided.

We live in a digital age where all user information is backed up and stored behind a user profile for seemingly every application and service on the internet. With the incline of processing power, it has become much easier and faster for hackers to crack passwords located on the internet. For this reason, it is important that users have a unique password for each of their user accounts that utilise strong password protocols. However, it is impossible to remember a unique password for each service. With a password manager, it is no longer required to remember unique strong passwords for each of your accounts. A password manager conveniently stores all user passwords in a centralized location. Total access is granted to the password vault through a master password key. This is the only password the user is required to remember.

My project idea arose from the same issue I kept encountering while using web-based applications. I found myself having to reset passwords for services each time I would get signed out due to the sheer volume of accounts I continually use. The issue is that different platforms have different requirements for setting passwords. This makes it difficult to remember each unique password for a specific service and using a single password for multiple accounts leads to greater security threat exposure. My goal is to develop an application that will combine the best features of already existing leading password managers and completely novel capabilities.

3.0 State of the Art

The motivation behind Vault Knox password manager is to create an approach that stores passwords offline. Storing passwords online exposes them to any hacker willing to try to hack the database that contains all the data of users of the password manager. It becomes your job to physically protect your passwords. Because a single user's password database is decentralized, a hacker is discouraged to try to hack a single offline user database. Sqlite will be used to generate the database. SQLite is a local file that only allows for one writable connection at a time. All information saved to the database will be encrypted in non-human readable byte format. Only the master user with their random and unique derived key is able to decrypt the data at run time.

On top of the standard features of an offline encrypted password manager, I will incorporate novel functionality such as a strong password generator that utilizes an algorithm to automatically generate a random password that is designed with numerous strategies to maximise protection against password cracking as well as user-supplied parameters. It is of utmost importance to maintain and keep your passwords up-to-date. To facilitate this functionality, a user will also be advised when a password should be updated after a scheduled amount of time has transpired. The program will terminate after a certain amount of inactivity has transpired, configured by the user. Users can recover their password vault using a recovery key if their master password is unknown. Users can request feedback reports on passwords stored in their password vault that incorporate strong password guidelines as well as informing the user if a currently stored password has been compromised. Users can simply copy and paste passwords from the password manager into online or offline input fields without exposing their character values. Furthermore, protection against common vulnerabilities like SQL injection, decryption and user privilege attacks will be accounted for.

Cloud Based Password Manager		Vault Knox Password Manager	
Pros	Cons	Pros	Cons
Cloud backup	Requires internet connection	Does not require internet connection	Manual backups
Sync devices	Centralised password database	Decentralised password database	Manual device syncing
	Subscription service	Free	
	Requires constant security updates	Only requires physical security	
	Subject to all network attacks	Subject to ineffective brute force attack	

4.0 Technical Approach

An iterative approach will be taken in the development of this project. The reason for this approach is because it enables a phased implementation strategy, with the highest priority features being developed first. It lowers overall project risk and focuses on improving feature quality while allowing for major changes throughout the project. It also allows for the proactive elimination of unforeseen design flaws.

The project scope and requirements are very fluid and ever-changing throughout the software development cycle. It is however of utmost importance that the main encryption features work well and are of high quality since it is the foundation of the project. All other features are built upon the hashing and encryption protocols. A password manager program is a special candidate when considering the main requirements. The overall project can not succeed without the implementation of the main requirements. Vice versa, it may be the case that the timeframe to develop features are overestimated. If this is the case, an iterative approach allows for the best flexibility when making new or improved features.

The best way to identify requirements for a project of this nature is to research and inspect already existing software that is most popular in the password manager niche. By testing and trialling existing software, it is possible to gauge its shortcomings of the software. The bare minimum will be to have industry-standard features and build further functionality on top of them. Otherwise, there is no incentive from a user's perspective to adopt new software. For novel requirements, newly developed python libraries will be inspected to formulate and brainstorm possible features that do not yet exist.

To describe requirements, use cases will be designed in the form of a use case diagram to illustrate the scope of the requirement. Each requirement will also be described with a use case description. Use case descriptions will illustrate the entire flow of the requirement throughout each description stage.

Gant team software has been used subsequently in this proposal to document a concise project plan in terms of project tasks, activities and milestones. Gant team is a very powerful software tool that visually illustrates the full timeframe of the software development life cycle. Each iteration of the approach is broken up into separate features. Each feature contains project tasks, activities to carry out to complete these tasks and milestones to realise whether development is on/ahead or behind schedule. Each feature has a timeframe of completion as well as the individual tasks that make up the feature.

The features are developed in chronological order. The order of the features is built in the same way a pyramid is built. The first features make up the basis of the project. These features act independently from one another. The next level of features adds extra functionality on top of the already established base. The features at the last level make up the detail of the overall project. This level includes practices such as testing and bug resolving/identification. Finally, the finished project is converted into a single executable application.

5.0 Technical Details

Python will be used as the implementation language. The benefits of using Python is extreme streamlined effectivity and simplicity without compromising on performance or quality. Python contains many useful native libraries and importing external libraries using the pip package-management system is seamless. Auto-py-to-exe is an external module that enables the final python script to be converted into a single application executable.

The golden standard used by most secure applications is the encryption of data. Ultimately if a password manager cannot back up and store sensitive data using the necessary encryption protocols the program proves to be useless. Encryption is a method to encode content to keep it an independent secret from third parties. The purpose of the encryption method is to ultimately decrypt values so that the encrypted values are human-readable and usable by the user with permissions when necessary. Encryption and cryptography are only as effective to the extent to which it is applied. A combination of encryption-decryption, random value generation and hashing will be implemented. Hashing will be used when generating the master key. Encryption will be used wherever the actual character value of entries i.e. password entries are required.

Hashing produces a unique one-size signature. This is an irreversible one-way function that ensures integrity. Many weak hashing algorithms such as MD5 have already documented exploitable rainbow tables. A rainbow table is a precomputed table used to cache the output of cryptographic hash functions, which is typically used to brute-force crack password hashes. Tables are commonly used to reconstruct the key derivation function that is made out of a limited number of characters. Small hashing algorithms have a greater chance to produce hash collisions. Hash collisions occur when two distinct values produce the same hash signature due to the finite amount of characters.

The hashing of passwords will be facilitated using the SHA-512 algorithm, which is considered to be the most secure protocol. The long character length of the hash also is best equipped to prevent collisions. Adding a salt value to the hashing algorithm further increases complexity. Most importantly we achieve protection against rainbow tables. Salt values should be user-specific and unique for the best results. Because of the unique nature of the salt value, derived keys no longer mimic rainbow tables or produce known hash collisions. Popular salt generation methods include utilizing system date and time, IP addresses, user input, etc. It is possible for a very skilled individual to acquire such information. To best protect against such practices we can use a randomly generated salt value utilizing the operating system UUID to completely randomize the value. It is necessary to encode user password input to UTF-8 so that hashing algorithm can be applied.

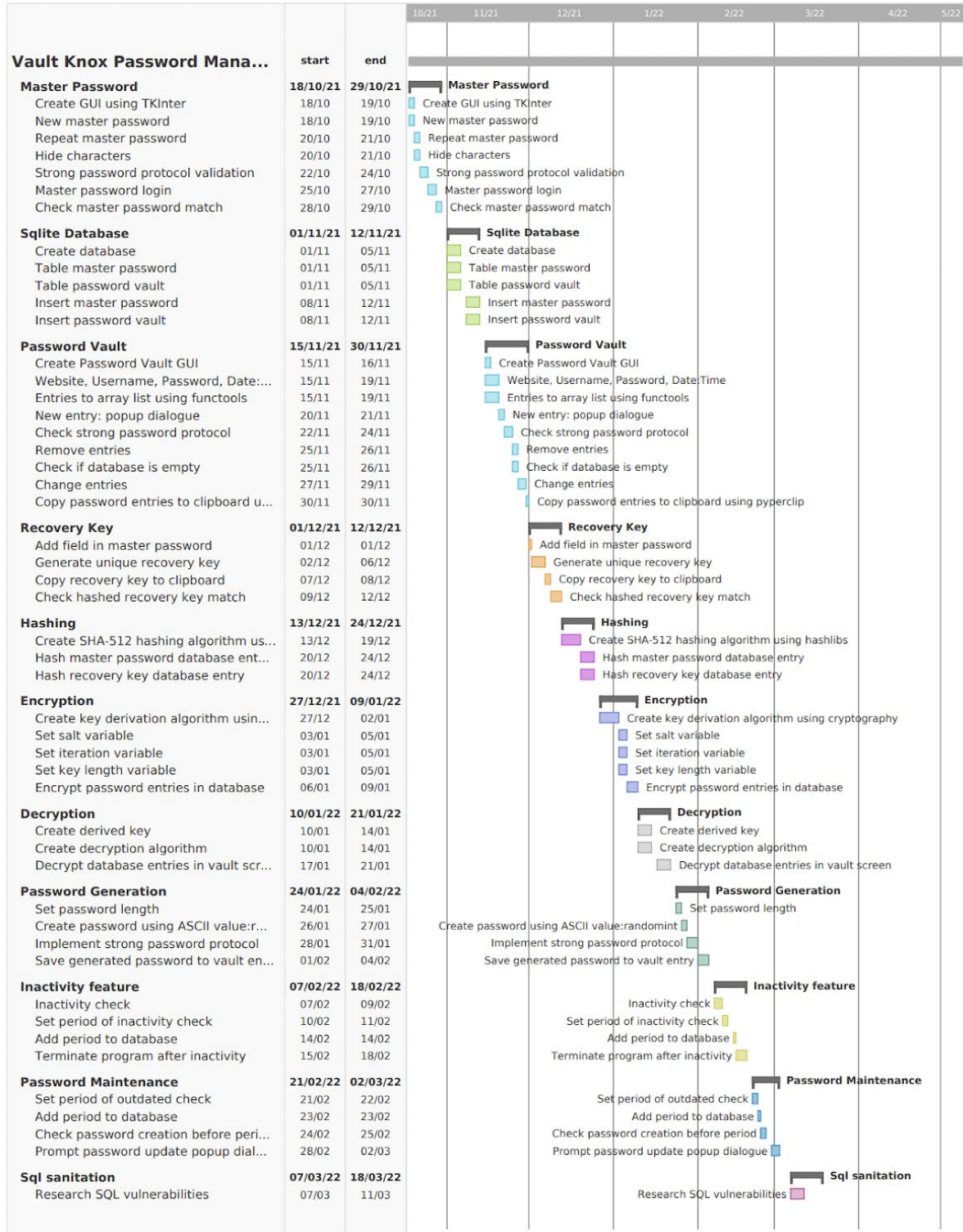
Encryption will be implemented using the fernet cryptography library. The library contains a key derivation function called pbkdf2_hmac. The key function generates a derived key that is unique to the master user. The same key is used to decrypt values that are encrypted. The function accepts 5 parameters to generate the function: hash name, password value, salt value, number of hash iterations and the key length. In this case, the function will contain: the hash name SHA-512, the user-supplied password value, the unique salt value generated from the operating system UUID function and the standard key length of the SHA-512 hash which is 64 bytes. Users can copy generated and encrypted passwords to their clipboard without exposing the character values using the pyperclip library.

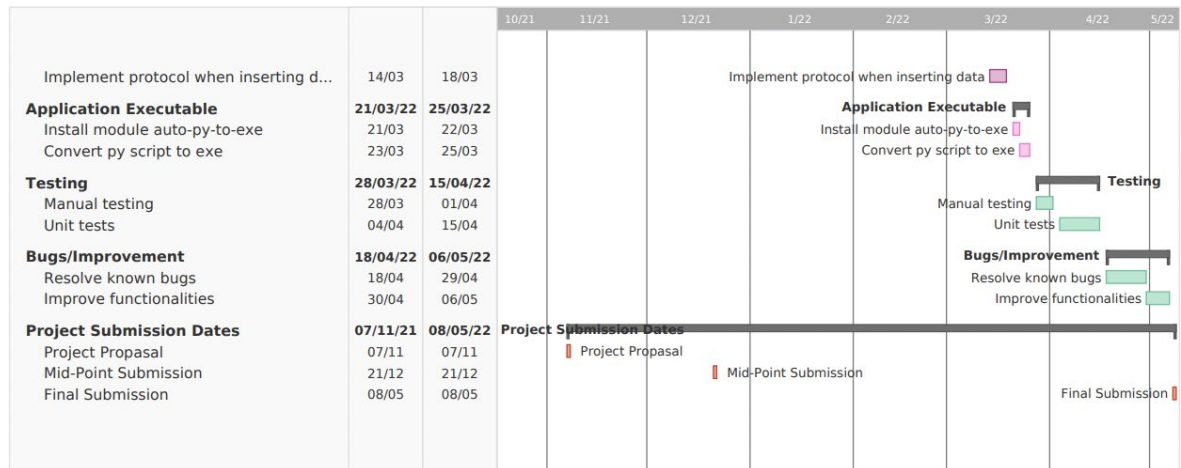
6.0 Special Resources Required

This project will not require any special resources to facilitate the development of the program as it will be hundred percent software-based. The basis of the program will be written in Python. The majority of the libraries that are imported are native to python. The only external package is the cryptography library imported from Fernet. Additionally, to convert the python script to an executable application the auto-py-to-exe module is installed using pip. However, this is a once-off process that the end-user does not have to fulfil to run the executable. Once the executable is created in the development stage, it can freely be shared along with the necessary program files, thereafter.

7.0 Project Plan

A project plan has been set up using software called Teamgantt. This project plan documents details on the implementation steps and timelines that will be followed to develop the application.





8.0 Testing

Testing of the application will be carried out in primarily two stages:

1 - **Manual testing** in the form of exploratory testing will be conducted throughout the development stage to ensure each function works as intended and does not elicit unexpected output or actions. After implementing a feature, testing will be done to ensure that previously developed features do not break.

As part of the project development stage, I will also send my project to three to five colleagues. They will have the opportunity to run the application, and subsequently, try to break it and identify any obvious bugs that may occur. Ultimately, they will give overall feedback on the quality, flow or any input on possible modifications that can improve the product, through the filling out of a standardized survey.

2 - I will utilize a python **unit testing** library called unit test. This allows programmers to write unit tests to determine if functions run as expected and return their respective accurate values. Unit tests help automate test cases to ensure functionality runs correctly and any changes or new features made to the project do not cause errors in other areas of the program without prompting failure of initialized unit tests.

9.0 Bibliography

Docs.python.org. 2021. *hashlib — Secure hashes and message digests — Python 3.10.0 documentation*. [online] Available at: <<https://docs.python.org/3/library/hashlib.html>> [Accessed 7 November 2021].

Cryptography.io. 2021. *Key derivation functions — Cryptography 36.0.0.dev1 documentation*. [online] Available at: <<https://cryptography.io/en/latest/hazmat/primitives/key-derivation-functions/#pbkdf2>> [Accessed 7 November 2021].

Teamgantt.com. 2021. *Online Gantt Chart Maker - Try for Free! | TeamGantt*. [online] Available at: <<https://www.teamgantt.com/>> [Accessed 7 November 2021].

Docs.python.org. 2021. *unittest — Unit testing framework — Python 3.10.0 documentation*. [online] Available at: <<https://docs.python.org/3/library/unittest.html>> [Accessed 7 November 2021].

Draw.io. 2021. *Flowchart Maker & Online Diagram Software*. [online] Available at: <<https://draw.io/>> [Accessed 7 November 2021].

PyPI. 2021. *auto-py-to-exe*. [online] Available at: <<https://pypi.org/project/auto-py-to-exe/>> [Accessed 7 November 2021].

Sqlite.org. 2021. *SQLite Home Page*. [online] Available at: <<https://sqlite.org/index.html>> [Accessed 7 November 2021].

Adobe.com. 2021. *Free Logo Maker: Design Custom Logos | Adobe Spark*. [online] Available at: <<https://www.adobe.com/express/create/logo>> [Accessed 7 November 2021].

6.3. Reflective Journals

Reflective Journals Semester 1:

Supervision & Reflection

Student Name	Dawid Marais
Student Number	x18133282
Course	BSc(Hons) in Computing, Specialising in Cybersecurity

Month: October

What?

This month I started my development on the login functionality. My planned functionality for the login pages included

- Create master password Page
- Master password login page
- Create Sqlite3 database
- Create tables and fields for data saved to the database
- Hide character functionality
- Link to Password Vault

So far, I have successfully completed the basic coding of the initial login screens. These screens include registering a new master password and logging in using the master login screen. These screens have been relatively easy to develop since they use the TKInter python GUI framework. To save time I kept the style minimal and simplistic because I want to focus on the functionality rather than style / look, in order to check off the hard tasks first then go back and make my tweaks and changes to my design choice and layout.

Furthermore, I have created the SQLite3 database for data saved by the application. Currently, I have functionality that adds the generated master password by the user, to the database. If the master password does not exist, a new database is created dynamically. Before saving the master password to the database, it is encrypted using a hashing

algorithm that prohibits the value from being human-readable. I have also hidden the characters that the user inputs into the master password text field. Once the user inputs the correct master password the application links to the blank password vault screen. So far I have not encountered any issues I could not resolve using the internet or my intuition.

So What?

In terms of the development of my project, November has been a successful month. I had previously prepared a Gantt chart to clearly represent the time allotted for each phase of my project. I utilized this chart to design the development timeline, and I am currently on track with the set-out schedule. I've been able to remain on track since I've been effectively allocating my time across each of my modules.

Now What?

Now that I know what a successful month is like, I'll make weekly objectives for the tasks I need to finish in order to stay on track with the completion of my final year project. This will make me more aware of when I'm falling behind on my goals and will motivate me to accomplish the work before the end of the week. If I don't do the tasks for the week, it will have a domino effect on my development in the following weeks. As a consequence, I'll be more motivated to finish what I set out to do that week.

Student Signature



Supervision & Reflection

Student Name	Dawid Marais
Student Number	x18133282
Course	BSc(Hons) in Computing, Specialising in Cybersecurity

Month: November

What?

This month I started my development on the login functionality. My planned functionality for the login pages included

- Create master password Page
- Master password login page
- Create Sqlite3 database
- Create tables and fields for data saved to the database
- Hide character functionality
- Link to Password Vault

So far, I have successfully completed the basic coding of the initial login screens. These screens include registering a new master password and logging in using the master login screen. These screens have been relatively easy to develop since they use the TKInter python GUI framework. To save time I kept the style minimal and simplistic because I want to focus on the functionality rather than style / look, in order to check off the hard tasks first then go back and make my tweaks and changes to my design choice and layout.

Furthermore, I have created the SQLite3 database for data saved by the application. Currently, I have functionality that adds the generated master password by the user, to the database. If the master password does not exist, a new database is created dynamically. Before saving the master password to the database, it is encrypted using a hashing algorithm that prohibits the value from being human-readable. I have also hidden the characters that the user inputs into the master password text field. Once the user inputs the correct master password the application links to the blank password vault screen. So far I have not encountered any issues I could not resolve using the internet or my intuition.

So What?

In terms of the development of my project, November has been a successful month. I had previously prepared a Gantt chart to clearly represent the time allotted for each phase of my project. I utilized this chart to design the development timeline, and I am currently on track with the set-out schedule. I've been able to remain on track since I've been effectively allocating my time across each of my modules.

Now What?

Now that I know what a successful month is like, I'll make weekly objectives for the tasks I need to finish in order to stay on track with the completion of my final year project. This will make me more aware of when I'm falling behind on my goals and will motivate me to accomplish the work before the end of the week. If I don't do the tasks for the week, it will have a domino effect on my development in the following weeks. As a consequence, I'll be more motivated to finish what I set out to do that week.

Student Signature



Supervision & Reflection

Student Name	Dawid Marais
Student Number	x18133282
Course	BSc(Hons) in Computing, Specialising in Cybersecurity

Month: December

What?

This month I worked on the development of my prototype for the midterm presentation. The aim of the prototype is to showcase the core functionality of the Vault Knox Password Manager. The core functionality of the prototype included:

- Password Vault
- Encryption protocol
- Decryption protocol

- Hashing
- Vault Recovery

So far, I have successfully completed a preliminary basic structure for all of the above requirements. The password vault screen allows a single user to save password entries to their repository of passwords. Each password contains a name for the service of the password, a username and the password value. Password entries can be added, updated or removed. Each password entry is tied to a single id and button illustrated through the GUI that a user can select to perform each of the previously mentioned operations. Furthermore, I have implemented and designed an encryption protocol for the data such as password entries and user credentials stored in the application. The encryption protocol allows for a reversed engineered decryption function. During run time the entries saved within the password vault is decrypted and presented to the user in a human-readable format. The decryption algorithm uses a unique master key that is concurrently implemented with the encryption protocol. Hashing was incorporated to ensure the integrity of the user master password and recovery key. The recovery key is a once-off generated code that is equally as unique as the ID value of the master password. Each time a master password is created a corresponding recovery key is automatically generated.

So What?

The development of the prototype for the midterm submission has pushed me to deliver a project past own expectations I set out. From this perspective, the total development of the project can allow for more detail and better developed robust strategies. Upon inspection of my initial project plan that I designed using Gantteam, it can be derived that I am currently slightly ahead of the scheduled project plan. Completing some of the more challenging requirements has enabled me to pursue better revision and refinement of issues later down the line.

Now What?

Looking forward, my main focus is now to incorporate a rudimentary implementation of outstanding functionality as soon as possible. I have found it more helpful to have a basic understanding and layout of the application that with time, I can refine and improve. This

approach allows me to inspect and determine the best course of action going forward. My project plan has also proven to be extremely valuable when managing my time and goals of the development process. I plan to further refine my project plan according to a more accurate and coherent methodology after attaining the experience of developing this project.

Student Signature



Reflective Journals Semester 2:

Supervision & Reflection

Student Name	Dawid Marais
Student Number	x18133282
Course	BSc(Hons) in Computing, Specialising in Cybersecurity

Month: January

What?

This month I was actively engaged in developing additional functionalities described in my project plan. After finishing the development of the minimum viable product in the form of a prototype for the midterm presentation my focus was aimed on creating features that would complement and improve upon the basic password manager.

Features included:

- Random Password Generation
- User-supplied generation parameters
- Application Inactivity check
- Application inactivity termination

Firstly, I identified that the random password generator feature would serve best for creating a new password entry and updating an existing password entry. When the user is prompted to modify a password entry they can choose the password for the generation feature by clicking on the appropriate button. The random password generator screen is

then launched where the user can select the length of the password they wish to generate in characters by from a scroll box widget. Furthermore, the user can select whether they wish to include different types of characters found in passwords such as special characters, digits and uppercase characters. Once the randomly generated password is created the user can save it to their password entry or generate a new one.

Another security feature that I was actively working on was the inactivity termination of the application after a certain amount of time had transpired. The user is prompted within the settings screen for the period that the application will prompt a notification asking the user whether they are still actively engaged in a session within the application. For this feature, it was required to create a new independent thread that would run in the application background. After the user-specified time frame has transpired a notification will prompt the user to answer yes or no, as to whether they are still using the application. If answered yes, the inactivity thread will reset otherwise the application will terminate all processes and threads and as a result self-terminate.

So What?

Now that the basic foundation for the password manager has been laid out and implemented, full focus can be contributed to developing interesting features and novel capabilities that will differentiate the application from its competitors. This process is trial by error to determine whether certain features are achievable and realistic. For this month, I had to spend a lot of time researching the mechanisms that Python use to create new processes and terminate them. Having done so I have a better understanding of the capabilities within the Python language for the development of future functionality.

Now What?

Next month will be spent on researching various services such as libraries and APIs to determine novel capabilities that would improve the overall use case scope of the application. Novel features require an exorbitant amount of planning and research to ensure their comprehensive implementation. This being said, I will continue to try and stay ahead of the project plan and analyse whether features are viable within the offline environment of Vault Knox. In terms of the development of features aim to increase my work output considerably compared to the month of January. Developing the foundation

for features is the aspect that takes the most amount of time and therefore it is optimal to have a basic outline to refine in the future.

Student Signature



Supervision & Reflection

Student Name	Dawid Marais
Student Number	x18133282
Course	BSc(Hons) in Computing, Specialising in Cybersecurity

Month: February

What?

This month I continued to remediate bugs and build upon implemented functionality. The goal of this month was to refine the core functionalities that I developed in the previous month which included:

- Backup of user data
- Duplication method of backed up data
- Refined encryption modules
- Ease of use, using Pyperclip

The biggest obstacle that I have encountered this far was the implementation of the encryption library of fernet. This module has built-in error remediation to facilitate the integrity of encryption. However, this has proven detrimental to the use case of the project as it assumes manipulating methods according to the requirements and processes of user action. This required me to alter the documentation of encryption methods according to the requirements of the project.

I created a backup mechanism for all the user data saved within the password vault. This proved to be a vital functionality that helped me during the debugging process. I initially had this requirement as a lower priority and scheduled its development later in the project plan. I encountered obstacles when deleting and updating data to the user data which required me to reset iterations of the password vault. This slowed down the development process and as such, I thought it best to improve my ability to reroll an iteration of the password vault sooner rather than later.

I also utilized the Pyperclip library to copy data to a device's clipboard without revealing the actual values during run time. This functionality greatly improve the user experience and added to the ease of use requirement that was set out in the project documentation. The copy to clipboard function allows users to enter confidential data to any field(online or offline) with a single click of a button.

So What?

Managing the ongoing development of the final year project has proven to be challenging this month. I took a break from development at the start of this year to pursue other priorities that I had in my personal life. These included applying for jobs after my graduation which is set in August. I am an international student and as such, it is more challenging to find graduate programs with requirements that I can meet. I have been contacted by Helen Conway regarding showcasing my final year project for marketing purposes. I gladly accepted as this would prove to be very beneficial as exposure for my programming skills from aspiring recruiters.

Now What?

Looking forward, my main focus is now to catch up with the project plan that I set out initially. Luckily, throughout the application process for multiple roles, I was requested to showcase my programming skills in the form of a coding assessment. These coding assessments are designed to establish the problem-solving capabilities of candidates and as such, even though I have not focused primarily on the development of my overall final year project as a product, I have still continued to improve my programming and problem-solving skills. After exposure to different algorithms that optimise for effectivity, I have realised that some of my encryption and processing logic have room for improvement.

Student Signature	
-------------------	---

Supervision & Reflection

Student Name	Dawid Marais
Student Number	x18133282
Course	BSc(Hons) in Computing, Specialising in Cybersecurity

Month: March

What?

For the month of March, I continued to add novel features to the application. these functionalities aim to enable the user greater flexibility and control over maintaining and managing data saved within the application.

Features included:

- Password Maintenance
- Password Report Feedback
- User Settings
- Password Strength Check
- Delete User data
- Load Screen

The feature of password maintenance was incorporated by specifying a time of the creation for any given password entry that is created or updated by the user. The user can specify within the user setting screen the period after which the program will prompt the update of an outdated password entry. When the user logs into the application password entries are checked and displayed to the user if outdated. Once the user changes the password value of the entry, the date of the correlating entry is updated and reset. The password report feedback section enables the user to select a single password entry and generate a correlating report based upon the strength and health of its values. Remediation of vulnerable passwords is prompted to the user if found. The user can also inspect data such as the number of uses a single password has accumulated. Finally, the days after which the password will expire are also added to the report.

In the user settings section, the user can wipe all the user data as well as the account from the application. This will enable a user to create a new user account and password vault on their system. When the application is first launched a loading screen is displayed to the user. This enables background processes such as initialising API calls and objects to be complete before the main application is shown to the user.

So What?

This month has proven to be very fruitful in terms of further evolving the application of the password manager and features that can be imposed upon data saved within the Vault. Creating the user settings and password report feedback sections required me to think carefully and plan the appropriate structure of classes that can access data saved within the encrypted database and modify data with functionality according to these user settings. Aspects such as determining the execution of feature checks were a challenge to map out since for each feature there are many moving parts. I further researched and refined threading capabilities and the implementation of processes with synchronicity to better my solutions.

Now What?

The project as a whole is starting to come together and as a result, its development continues to be a rewarding practice. I am currently working exactly in tandem with the timeframe of my project plan set out at the start of my project. I have however updated and added features that I wished to develop in the future, to the project plan with the appropriate time frames. The development of the project will taper down for the next month due to the influx of different assignments and exams I am engaged in, in the month of April. For this reason, I'm planning my time management meticulously to prepare for this fact.

Student Signature



Supervision & Reflection

Student Name	Dawid Marais
Student Number	x18133282
Course	BSc(Hons) in Computing, Specialising in Cybersecurity

Month: April

What?

The month of April marks the last reflective journal for my final year project. This month's focus was aimed to progress the application to its final state as much as possible. Features specified within the project plan were implemented for the final time with the intention to finalize them in the last weeks leading up to the project submission date.

Features included:

- SQLi prevention
- Brute force prevention
- Geometric identification
- Service identification
- Data self-destruction

Security of the VaultKnox password manager application is of utmost importance and thus this month, vulnerabilities such as SQL injection were addressed by implementing a context manager. The context manager will convert all supplied parameters of executed SQL operations and convert them to a secure string format. This prohibits any type of SQL syntax or operations to be injected into the backend of the application. Brute force attacks are another vulnerability that was addressed during the login and recovery process. Preventing brute force attacks means prohibiting the number of requests users can make to sensitive mechanisms such as login and recovery. In the case of the login mechanism, the user is only allowed to attempt 5 login requests per minute. For the Recovery mechanism, the user is only allowed 2 login requests per minute. Additionally, in total the user account will be locked if requests to login exceed 10 attempts or requests to recover exceed 5 attempts in total. A locked user account can only be unlocked by entering the valid recovery key for the

password vault. In the case of exceeding recovery attempts, the application will wipe all user data if it is determined that a user is trying to brute-force the recovery mechanism. When a user creates a new password entry the program will now determine whether the entered service name produces a valid response by getting the URL. This mechanism is utilized when password entries are outdated, consequently updated and the service is launched in a new tab where the user can update the new password details online.

So What?

This month proved to be extremely difficult in managing a healthy work-life balance due to the sheer amount of work that was necessary to be completed. Work included finishing up projects as well as the final exam period of my degree. In the interest of performing well in my modules, I had to sacrifice some of the time available for the development of the Vault Knox password manager. Fortunately, I was ahead of schedule following the previous month's progression and will be able to complete the project in accordance with my project plan moving forward. I also accounted for a period of improving the existing functionality of the project into the project plan that will enable me to finish features on time.

Now What?

Looking forward, I can luckily focus all my attention and energy on the sole pursuit of developing my final year project as I have completed the other modules that make up my degree. In conclusion, my plan is to design a comprehensive testing methodology that will include unit, integration and end-to-end mechanisms for the project as a whole. Once these testing mechanisms have been established as the final step I will convert the entire Python application into a single executable file that can run independently of the operating system.

Student Signature

