

# Offline Password Vault

Student Number: 3807942

Deadline: Friday 13th of November 2020, by 5 pm

## Contents

Introduction.....	5
Aims .....	6
Objectives .....	6
Similar solutions.....	6
Constraints .....	7
Software development process.....	7
Requirements .....	8
Stakeholders identification table .....	8
Onion diagram.....	11
User stories.....	11
Use case diagram.....	12
Use case descriptions.....	13
Activity diagrams .....	14
Design.....	16
Low fidelity.....	16
High fidelity.....	24
Technical Review .....	29
Similar systems .....	29
Justification .....	29
Non-functional requirements.....	30
Implementation .....	31
Class diagram .....	31
Start page .....	32
Generate password page .....	37
View all accounts page .....	41
Add account page .....	46
View account page .....	48
Change password page .....	53
Import accounts page.....	55
Export accounts page.....	59
Critical Evaluation.....	62
Deciding what system to develop.....	62
Aims of the system .....	62
Further development.....	62
Developing the system.....	63

Personal reflection .....	63
Appraisal .....	63
Bibliography .....	64
Appendix A – main.py (Bhatti, 2020) .....	69
Appendix B – Project Proposal .....	89
Figure 1 .....	11
Figure 2 .....	12
Figure 3 .....	14
Figure 4 .....	15
Figure 5 .....	16
Figure 6 .....	17
Figure 7 .....	18
Figure 8 .....	19
Figure 9 .....	20
Figure 10 .....	21
Figure 11 .....	22
Figure 12 .....	23
Figure 13 .....	24
Figure 14 .....	25
Figure 15 .....	25
Figure 16 .....	26
Figure 17 .....	26
Figure 18 .....	27
Figure 19 .....	27
Figure 20 .....	28
Figure 21 .....	31
Figure 22 .....	32
Figure 23 .....	32
Figure 24 .....	33
Figure 25 .....	33
Figure 26 .....	34
Figure 27 .....	34
Figure 28 .....	35
Figure 29 .....	35
Figure 30 .....	36
Figure 31 .....	37
Figure 32 .....	37
Figure 33 .....	38
Figure 34 .....	39
Figure 35 .....	40
Figure 36 .....	40
Figure 37 .....	41
Figure 38 .....	41
Figure 39 .....	42

Figure 40.....	43
Figure 41.....	44
Figure 42.....	45
Figure 43.....	46
Figure 44.....	46
Figure 45.....	47
Figure 46.....	47
Figure 47.....	48
Figure 48.....	49
Figure 49.....	49
Figure 50.....	50
Figure 51.....	50
Figure 52.....	51
Figure 53.....	52
Figure 54.....	53
Figure 55.....	53
Figure 56.....	54
Figure 57.....	55
Figure 58.....	55
Figure 59.....	56
Figure 60.....	57
Figure 61.....	58
Figure 62.....	59
Figure 63.....	59
Figure 64.....	60
Figure 65.....	61
Figure 66.....	63
Table 1 .....	6
Table 2.....	10
Table 3 .....	12
Table 4.....	13
Table 5.....	14
Table 6.....	89

## Introduction

Online password managers are likely to be compromised by hackers in contrast to individuals storing their account data on their machines as online password managers store their users' data on centralised servers where, if they were hacked, you and thousands of individuals would have their data compromised. Additionally, (Wagenseil, 2020) illustrates how password managers can be compromised easily due to human error as most password managers utilise a master password to secure the password manager. If the master password was subjected to a brute force attack, the password manager could be compromised if no other authentication or security techniques were implemented to secure the vault. Although the source highlights a multitude of attack vectors which could compromise password managers which have since been fixed, the master password of a users password manager is a critical ingress point which could render the accounts stored in the password manager vulnerable. If the password manager checks that the master password is relatively complex then the likelihood of brute force attacks working becomes minuscule, however, for passwords which are extremely simplistic a brute force attack would allow a hacker to gain access to the users' password manager effortlessly.

(Trend Micro Incorporated, 2018) depict a timeline of security vulnerabilities in which companies that utilise servers to store their users' data offer no encryption on the drive that the data is being stored on or may store the data in plain text files making the data stored on the servers vulnerable to hackers which have gained access to the company's systems. The source also illustrates the type of data that was accessed during these data breaches such as old databases with user information which contain names, addresses, emails, phone numbers and credit card information. Although GDPR allows an end-user to request for their data to be deleted from a platform, most end-users are unaware of this which allows companies to store and utilise an end-users' data as they see fit for up to 20 years.

Thus, the only way to ensure that an end-users' data is secure is by an end-user being the only individuals with physical access to their data. This project aims to solve this by making password storage and generation offline in encrypted or non-plain text files stored locally on the users' machine.

One basic requirement for the software to work as intended would be unique and random password generation while allowing users to store these passwords with additional information such as an identifier to associate the password to an account and a username for the account. Another requirement would be for an end-user to view all of their accounts with any associated information for these accounts such as usernames and passwords.

The project will require the usage of encryption algorithms to ensure that the application complies with the "Integrity and confidentiality" principle illustrated by (Information Commissioners Office, 2020) concerning GDPR to ensure the security of data that is collected and stored by a system. Furthermore, the application will implement a graphical user interface to allow users to interact with the application efficiently and to act as a feasible alternative to systems which are accessed through browsers thus improving the usability of the application and helping entice individuals due to any unique design choices made during the development of the application.

## Aims

- Utilising an encryption algorithm and understand methods used to crack encryption algorithms
- Developing skills for creating a program with a graphical user interface
- For the developed application to reside in a smaller digital footprint than alternative systems

## Objectives

The program needs to be able to:

- Store all account details in a file
- Generate passwords
  - Generate a random sequence of characters to create a password where the characters used to generate the password are specified by the user
  - Storing generated passwords with an identifier and the username for the account
- Read all saved accounts from the account details file
  - Filtering displayed accounts by a specific keyword
  - Displaying a specific account with its username and password
  - Remove accounts from the account details file
- Import account details from another password manager in a CSV or JSON format
- Export account details from the application in a readable CSV or JSON format
- Provide dialogue boxes for any errors or actions which require user intervention

## Similar solutions

PLATFORM	FEATURES
<b>LASTPASS BY (LOGMEIN, INC., 2020)</b>	As a free customer, individuals get to save and generate as many passwords as they want and store notes, addresses, payment cards, bank accounts, drivers' licenses, passports and WIFI passwords. Paying users are entitled to 1GB of encrypted file storage or premium licenses for others.
<b>(DASHLANE INC., 2020)</b>	As a free customer, users can store up to 50 passwords but can only access these accounts on 1 device. Additionally, users can secure their account with two-factor authentication. A paying customer can store an unlimited number of passwords and can access their account on an unlimited number of devices while getting access to Dashlane's VPN service.
<b>(BITWARDEN, INC., 2020)</b>	As a free customer, users can store as many passwords as they want and can access their passwords on any device and can choose to host the service on their server. Paying users can use the services' two-factor authenticator and get access to 1GB of encrypted storage.
<b>KEEPPASS BY (REICHI, 2003)</b>	The software offers database encryption for user accounts, multiple keys to access the database so multiple users can see passwords in the database but not all of them and portability of running the software without installation.
<b>(KEEPPASSXC TEAM, 2016)</b>	The software offers database creation to store and search through accounts with password generation. Furthermore, the system allows the generation and storage of two-factor authentication keys and sharing of a user's password database.

Table 1

LastPass and Dashlane were created to allow users to create and store their passwords for their accounts securely from multiple devices and with the ability to access their passwords from any device with an internet connection. However, due to the closed source nature of these systems, they appear less reliable as an individual is unable to clarify the validity of the security measures that are implemented. The only evidence to justify these claims are security reports released every year by the service which details the methods they utilise to keep an individual's account safe.

In contrast, Bitwarden offers similar features to LastPass and Dashlane but operates with an opensource methodology in which all code used to develop the system is viewable by the public so end users can actively contribute to the development of the system. Furthermore, due to Bitwarden being opensource, users can choose to host an instance of Bitwarden on a local machine to store all of their data locally instead of on the service's servers, unlike LastPass and Dashlane.

However, there are platforms like KeePass and KeePassXC which operate as offline systems which store your account information on your local machine without the need for an internet connection. Although Dashlane, LastPass and Bitwarden work on multiple devices, KeePass and KeePassXC will only work on machines running macOS, Windows or a Linux distribution with unofficial support for mobile operating systems. Furthermore, as KeePass and KeePassXC do not utilise a server, manual syncing is required if any passwords are changed or if a user adds a new account to their KeePass database.

## Constraints

Although the system could store an end user's data on a server where the application would access the server each time a user wants to get their account details, this will leave the account details vulnerable if the IP address of the server was discovered thus an internet connection shouldn't be necessary for the system to function. Additionally, the system could be developed for multiple platforms, however, due to time constraints and the security vulnerabilities associated with storing passwords on mobile operating systems in which the device could be stolen or tampered with, the system will be developed for a desktop environment. Furthermore, the system is intended for only one user therefore, there should be only one password or key which allows access to the application. Lacking the technical expertise to develop an encryption algorithm, I will have to rely on the encryption algorithm supplied by the programming language deemed most appropriate for the development of the application.

## Software development process

(Sommerville, 2016) demonstrates three general process models which can be adapted for use in software development: the waterfall model, incremental development and the integration and configuration model. With detailed consideration of the individual characteristics of each software development model, the incremental model would be most suitable for the project. Although the overall structure of the source code will degrade as new increments are added to the system, this model prioritises delivery and deployment of the software whilst allocating enough time for detailed testing and with regular refactoring, source code degradation could be avoided. Even though the waterfall model would allow for more detailed planning before the development of the system, the model is more suitable for large scale systems, however, a similar structure of the waterfall process will be implemented to mend any flaws in incremental development. This allows for visualisation of how the system should function before development takes place which will incorporate requirements gathering techniques similar to those in software processes which follow agile methodologies, such as user stories, to assess how end-users and those in the wider community may interact with the system.

Agile software processes would be inadequate when developing the system as they aren't suitable for developing systems which require robust security, however, if the system had a client to gather requirements from then most likely an agile software process would have been used. The integration and configuration model is primarily used when developing a system that is modelled after a pre-built system or developing new features for a pre-built system thus, it's not considered a viable software process for use when developing the project as we may only reference any code used as inspiration when developing the project, not building upon another's work while submitting it as our own. However, requirements refinement, which is mainly implemented in integration and configuration, will be utilised within the project to evaluate similar software systems in hopes of identifying features that end-users require when using the system. Overall, the optimal process model for the project

would be incremental development as it allows for detailed testing of the system before deployment whilst providing similar speeds to agile software models thus accomodating time constraints set for the project.

## Requirements

### Stakeholders identification table

(Lemac & Phillips, 2011) is used to identify stakeholders that are interested in the system, represented in the template provided by (Lemac, 2016). This will aid in identifying users of the system and any features that should be implemented according to how a stakeholder envisages how the system should operate.

Stakeholder	Stakeholder Role/Responsibility	Importance	Influence	Interests/ Positive Impacts	Concerns
End-user	They use the product directly to store their passwords or generate new passwords for accounts.	They are of low importance as they aren't able to identify how any technical aspects of the system should be implemented.	They have high influence as they decide how the system should function and what features the system should have.	The product will ensure that only they can view or have access to their account information	They will be concerned about how secure the system is and if the system is easy to use.
Maintenance operator	They use the system directly to fix any issues with the system or to ensure that the system continues to function if any code needs to be changed in the software.	They are of medium importance as they identify any technical issues of the system and release updates to fix these issues.	They have low influence as they don't dictate how the system should function or how the system should be developed.	They are interested in how many individuals will use the system.	They are worried about any issues which would occur if the software libraries used to create the system were to change drastically in the future.
Project manager	They don't use the system directly as they organise how the system should be developed.	They are of high importance as they identify any time constraints in developing the system.	They are of medium influence as they dictate how long it will take for the development of the software to finish and they also direct how the software should be developed.	They are interested in how quickly they can complete the project development cycle.	They will be concerned about whether the system is ready for use by a consumer.
Negative stakeholder	They don't use the system directly as they compare the	They are of high importance as they have	They are of low influence as they don't	They are interested in if the system	They would be concerned about clients



	system to their software implementation.	implemented similar systems successfully.	dictate how the system should operate.	has implemented any features that could be implemented in their software.	of their software solutions using this software or similar software systems thus losing them revenue or relinquishing their access to that client's data.
Requirements analyst	They don't use the system as they collect information on the features end-users want the system to be able to carry out.	They are of medium importance as they gather information on how end-users want the system to function.	They are of low influence as they identify how an end-user wants the system to function whilst omitting any personal bias for features, they want to be implemented in the system	They are interested in any features the system requires to be functional or considered complete.	They are concerned about the requirements they have gathered not being detailed enough and not suiting what the end-users need from the system.
Designer	They don't use the system directly as they design how the system will look.	They are of low importance as they design how the system should look.	They are of high influence as they decide how the system should look and how an end-user will navigate through the system.	They are interested in how an end-user would perceive the system.	They are concerned about whether the system is intuitive enough for an end-user to navigate through.
Programmer	They don't use the system directly as they create the system.	They are of high importance as they identify the best way to implement the software.	They have low influence as they implement features identified by the requirements analyst.	They are interested in the mechanics of building a software solution suitable for an end-user.	They would be worried that the solution they used to implement the system isn't suitable enough for an end-user or isn't optimal in terms of performance on a user's machine.
Tester	They use the system directly to identify	They are of medium	They have medium	They are interested in	They would be worried about

	any issues with the operation of the system before deployment.	importance as they identify any issues while using the system.	influence as they decide when the system is functional without any identifiable issues.	how their feedback has impacted the development of the system.	the issues they discovered during the development of the system not being resolved before deployment of the system.
Security engineer	They may use the system directly to test how secure the system is at keeping an end-users' account information safe.	They are of medium importance as they can assess how secure the system is for an end-user.	They have high influence as they decide whether the system is secure enough for deployment.	They are interested in how the security of the system was designed and how these security practices could be used by other systems	They would be concerned about how secure the system would be on an end-users machine.
Software expert (consultant)	They don't use the system directly as they identify how similar systems function.	They are of medium importance as they identify how other companies have implemented similar features in their software.	They have low influence as they don't decide how the system should function or what features should be implemented in the system.	They are interested in the use of the system and how they could gain commission from being referred by the developers of this system.	They would be concerned that if the system were to fail, they wouldn't receive any commission or royalties after providing their services.
Government	They don't use the system directly.	They are of low importance as they don't provide any technical expertise into the operation of the system.	They have high influence as they decide if the system abides by the Data Protection Act.	They are interested in the adoption of the system by individuals which choose to use this system	They would be concerned with whether the system complies with the Data Protection Act and if the system can keep an end-users data secure.

Table 2

## Onion diagram

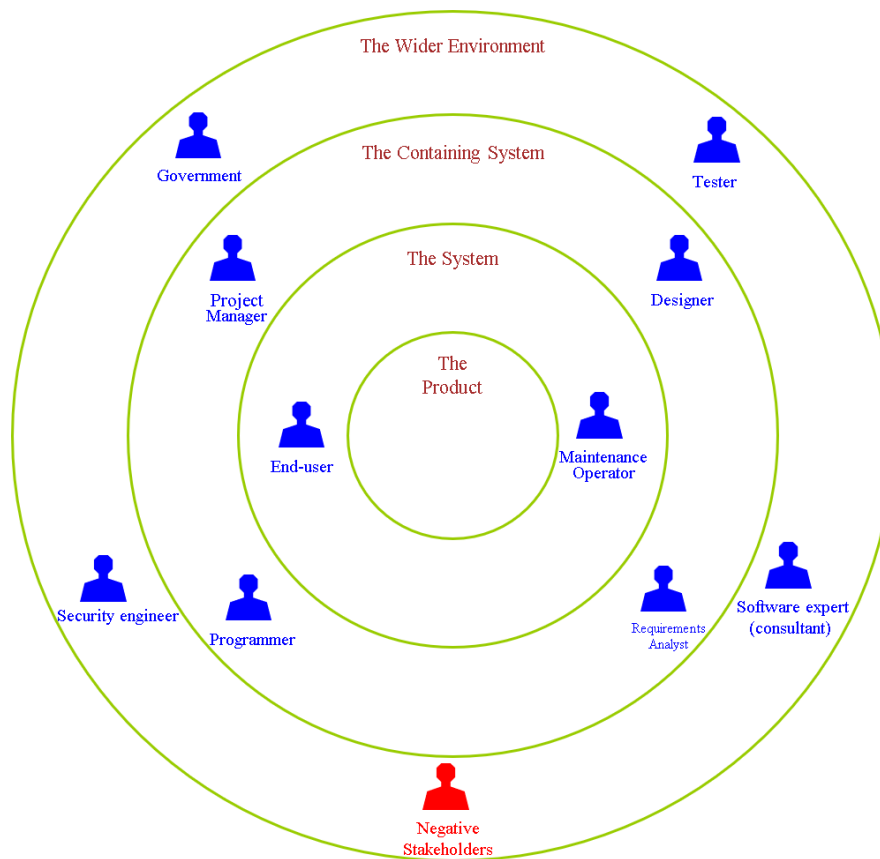


Figure 1

(Alexander & charbison, 2010) is used as a template in Figure 1 to illustrate the position stakeholders will take in regards to their interaction with the system. Stakeholders which are considered direct users of the system have been addressed in Table 2.

## User stories

User	User story
End-user	As an end-user, I want to create a vault to store my accounts
End-user	As an end-user, I want to open the vault I created to store my accounts
End-user	As an end-user, I want to generate passwords for any accounts I create
End-user	As an end-user, I want to save the passwords I generate for the accounts I create
End-user	As an end-user, I want to view all my accounts so I can see the accounts I have
End-user	As an end-user, I want to search through my accounts to find the details for a specific account
End-user	As an end-user, I want to view the details of any account
End-user	As an end-user, I want to delete any accounts I no longer need
End-user	As an end-user, I want to be able to change the passwords of any of my accounts

End-user	As an end-user, I want to manually add accounts that I have
End-user	As an end-user, I want to import any of my accounts from other password managers
End-user	As an end-user, I want to export my accounts from my vault to be used in other password managers.

Table 3

Table 3 depicts user stories from the perspective of an end-user which will be used to identify the main features necessary for the application to function.

### Use case diagram

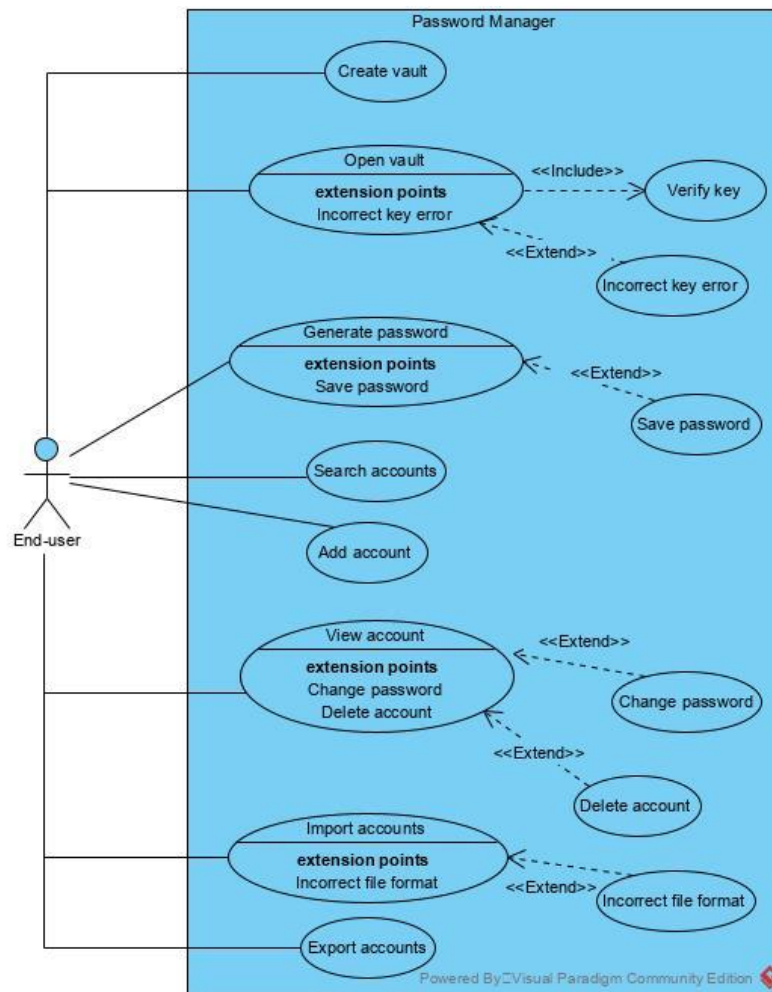


Figure 2

Figure 2 can be used to illustrate the use cases an end-user may execute which could be used as a model for any functions in the programs source code. (Lemac & Phillips, 2011) was used to illustrate the include and extend relationships between use cases as depicted in Figure 2. The only user of the system would be an end-user directly interacting with the application. “Verify key” is identified as the only include relationship in Figure 2 as it is triggered after a user attempts to open the vault as a means of validation before viewing the users’ accounts. “Incorrect key error” is considered as an extend relationship as this use case is only triggered if a user attempts to access the program without a valid key which will trigger an error message to be displayed. Furthermore, “Save password” has been associated to “Generate password” as an extend relationship as users may choose to save the

generated password or to generate a new password. “Change password” and “Delete account” are illustrated as extended use cases to “View account” as they do not have to be executed when viewing an account, however, the option to trigger these use cases is only accessible while viewing an account. “Incorrect file format” is an extended relationship of “Import accounts” as it triggers an error message if the file used is formatted incorrectly. Lastly, the “Add account” use case allows users to add account information manually by entering a label, username and password for an account.

### Use case descriptions

(Lemac, 2016) has been used as a template to represent two use cases in Figure 2 which require an explanation into their functionality. This will aid in the development of these algorithms by giving insight into how each of the algorithms should function.

<b>Use case name:</b>	Open vault	
<b>Scenario:</b>	End-user tries to open password vault	
<b>Triggering event:</b>	End-user want to view their accounts or generate a new password	
<b>Brief description:</b>	End-user isn't able to look at their accounts or generate a new password without opening their password vault and validating that it is their vault	
<b>Actors:</b>	End-user	
<b>Related use cases:</b>	Verify key Incorrect key error	
<b>Stakeholders:</b>	End-user Security engineer	
<b>Preconditions:</b>	End-user has a vault already created End-user has the decryption key	
<b>Postconditions:</b>	The program displays the password generation screen The program can show the accounts screen	
<b>Flow of activities:</b>	<b>Actor</b>	<b>System</b>
	1. End-user selects the decryption key file 2. End-user selects vault file 3. End-user clicks open button	3.1 System checks decryption key and vault file 3.2 System displays the home page of the program
<b>Exception conditions:</b>	3.1 If there is no decryption key file selected when clicking the open button, display an appropriate error 3.1 If there is no vault file selected when clicking the open button, display an appropriate error 3.1 If decryption key file doesn't decrypt the vault file, display an error message that the wrong key or vault was used	

Table 4

The security engineer is depicted as a stakeholder in Table 4 as they would be interested in the software mechanisms used to identify how securing the program and how the validation used in the program would work.

<b>Use case name:</b>	Export accounts
<b>Scenario:</b>	End-user exports accounts
<b>Triggering event:</b>	End-user decides to use a different password manager
<b>Brief description:</b>	End-user wants to export their accounts from the program to use in a different password manager
<b>Actors:</b>	End-user
<b>Related use cases:</b>	

<b>Stakeholders:</b>	End-user Programmer	
<b>Preconditions:</b>	End-user has unlocked their vault	
<b>Postconditions:</b>	Vault continues to display accounts Account details file is outputted to the desktop in a readable format	
<b>Flow of activities:</b>	<b>Actor</b>	<b>System</b>
	1. End-user clicks the export button 2. End-user chooses to export as CSV or JSON	2.1 System reads encrypted vault file 2.2 System writes vault file into a readable CSV or JSON file 2.3 System displays a confirmation message that the process is complete
<b>Exception conditions:</b>	2. If the end-user chooses to, they can cancel exporting	

Table 5

A programmer has been considered as a stakeholder in Table 5 as they would be interested in whether the account details stored in the program were outputted to a CSV or JSON file with correct formatting.

### Activity diagrams

(Lemac & Phillips, 2011) has been used to illustrate the flow of activities in Table 4, represented in Figure 3, and Table 5, represented in Figure 4. This further illustrates the flow of data while an algorithm is being executed.

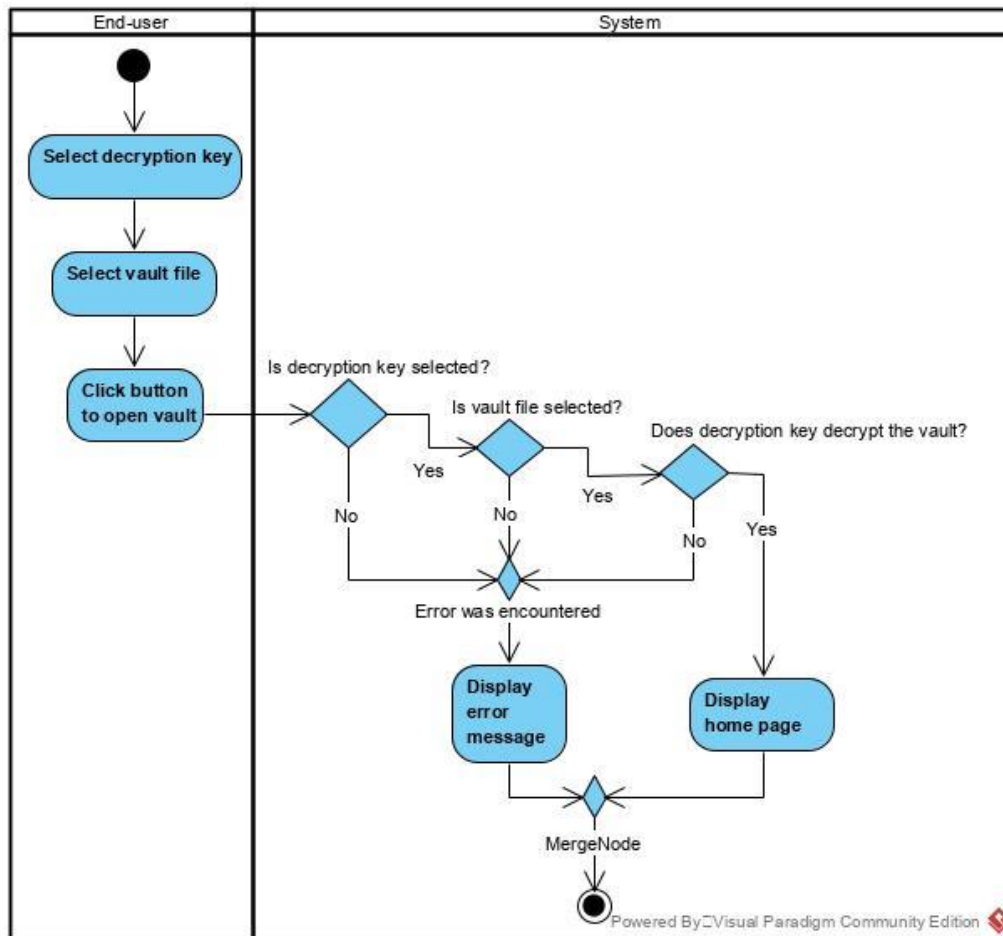


Figure 3

Although there is only one activity to display an error message in Figure 3, the error that will be displayed will differ depending on the event which triggers the activity.

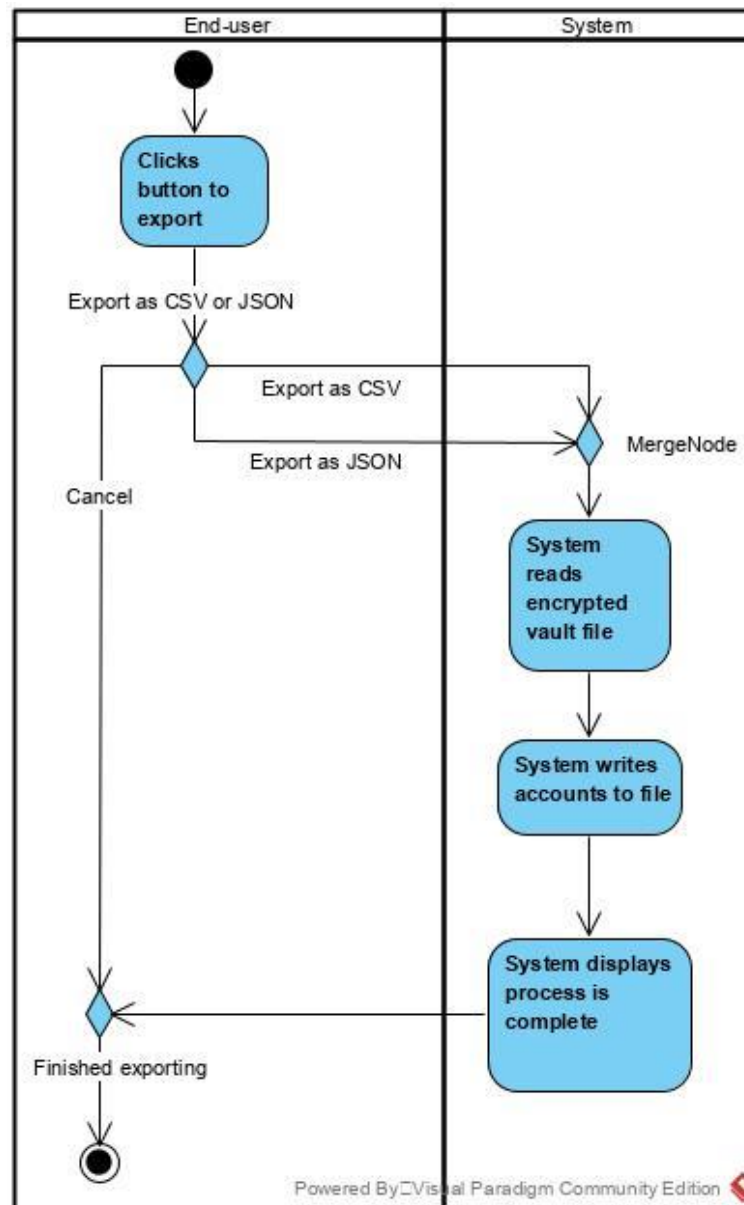


Figure 4

## Design

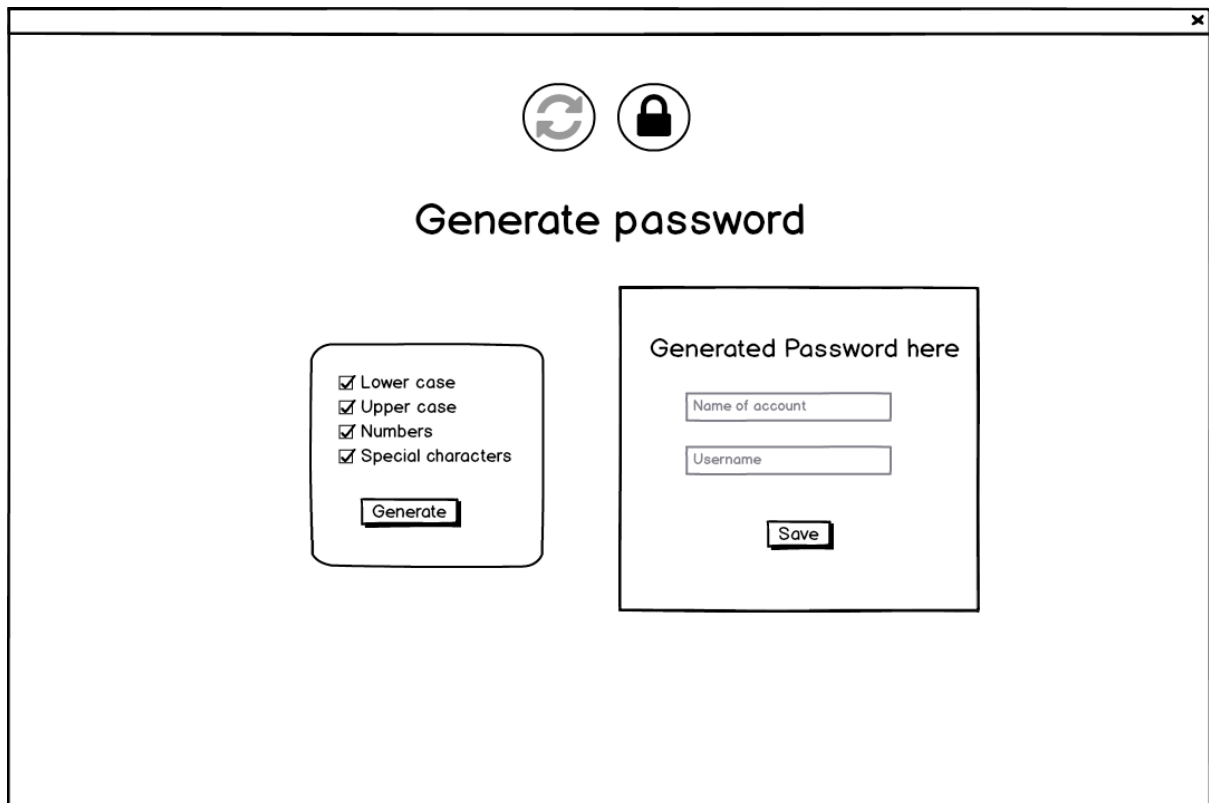
### Low fidelity

<h3>Create Vault</h3> <p>This will create an encrypted file to store your account details and a key file used to access the vault which can be found on the Desktop as vault and key. The next time you use the application you will need these files to access your accounts. If you wish to proceed, press start.</p> <p><b>Start</b></p>	<h3>Open Vault</h3> <p>Key file <input type="text"/></p> <p>Vault file <input type="text"/></p> <p><b>Open</b></p>
---	--

Figure 5

Figure 5 will be displayed when the user executes the program. An end-user will select to either create a vault to use with the application or they will open their vault by selecting their vault and key files from the file selection buttons. After selecting their vault and key files an end-user will click the open button to open the application to view their accounts or add accounts.





The screenshot shows a web browser window with a title bar containing a close button (X). The page has a light gray background. At the top center, there are two circular icons: a refresh icon (two curved arrows) and a padlock icon. Below these icons is the heading "Generate password" in a bold, black, sans-serif font. The main content area is divided into two sections. On the left, there is a rounded rectangular box containing four checked checkboxes: "Lower case", "Upper case", "Numbers", and "Special characters". Below these checkboxes is a "Generate" button. On the right, there is a rectangular box with the heading "Generated Password here". Inside this box, there are two text input fields: "Name of account" and "Username". Below these fields is a "Save" button.

*Figure 6*

Figure 6 will be displayed after an end-user opens the vault. A user can choose to view all of the accounts by clicking the padlock button otherwise, a user will generate a password and save the generated password on this page. The end-user must select the character sets they want in their generated password and must click the generate button to generate a password. If no values are entered into the name of account or username field, the password will not be saved.

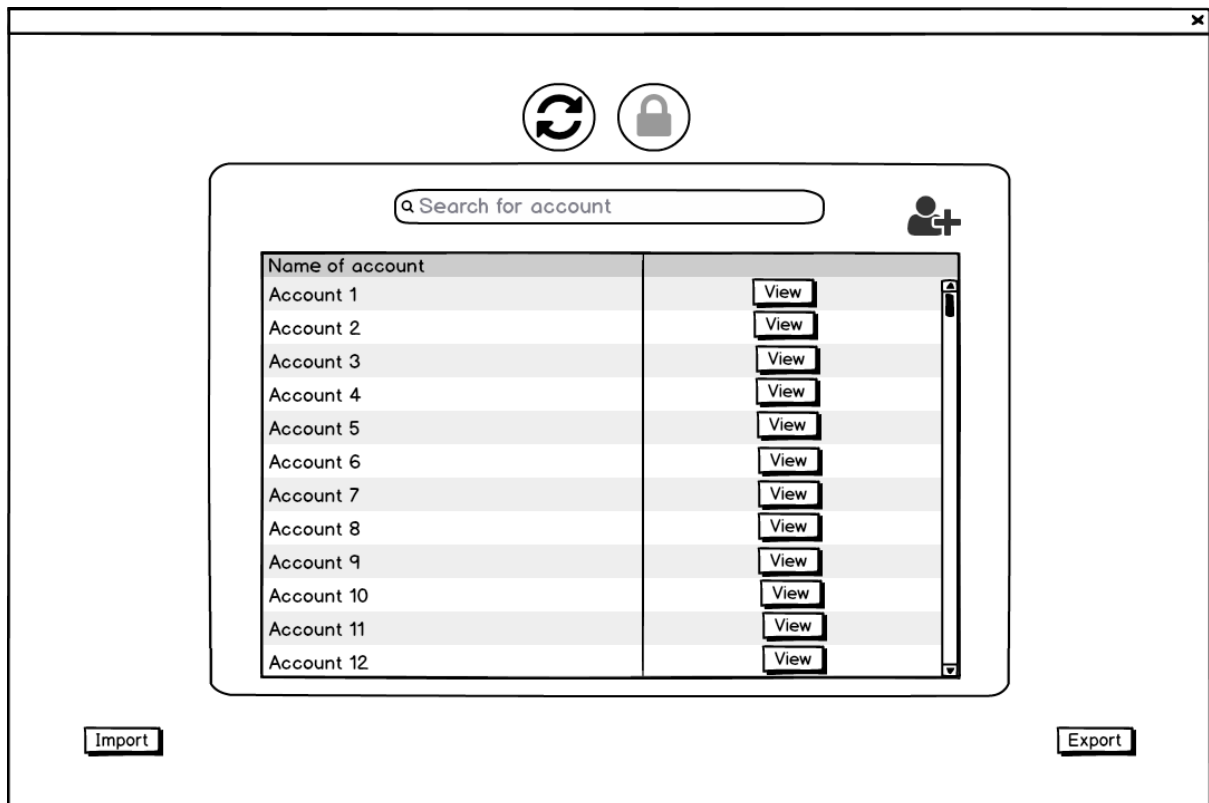
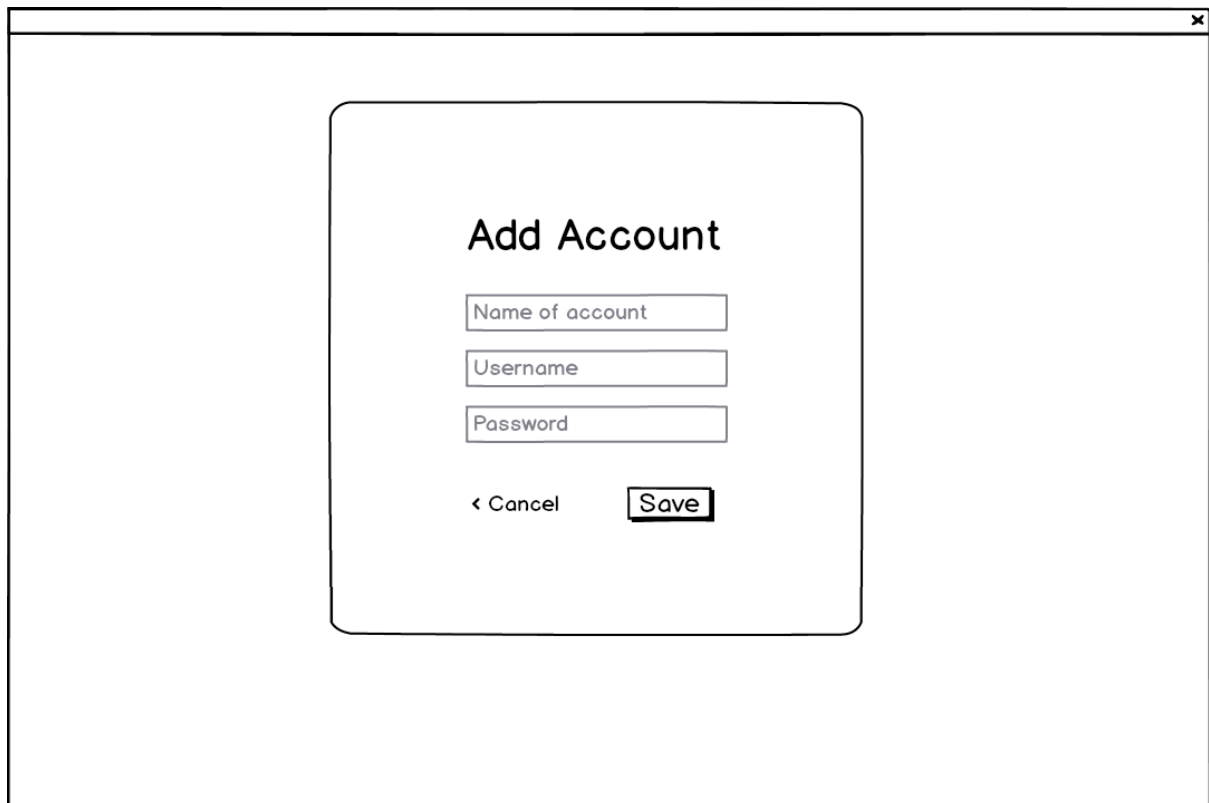


Figure 7

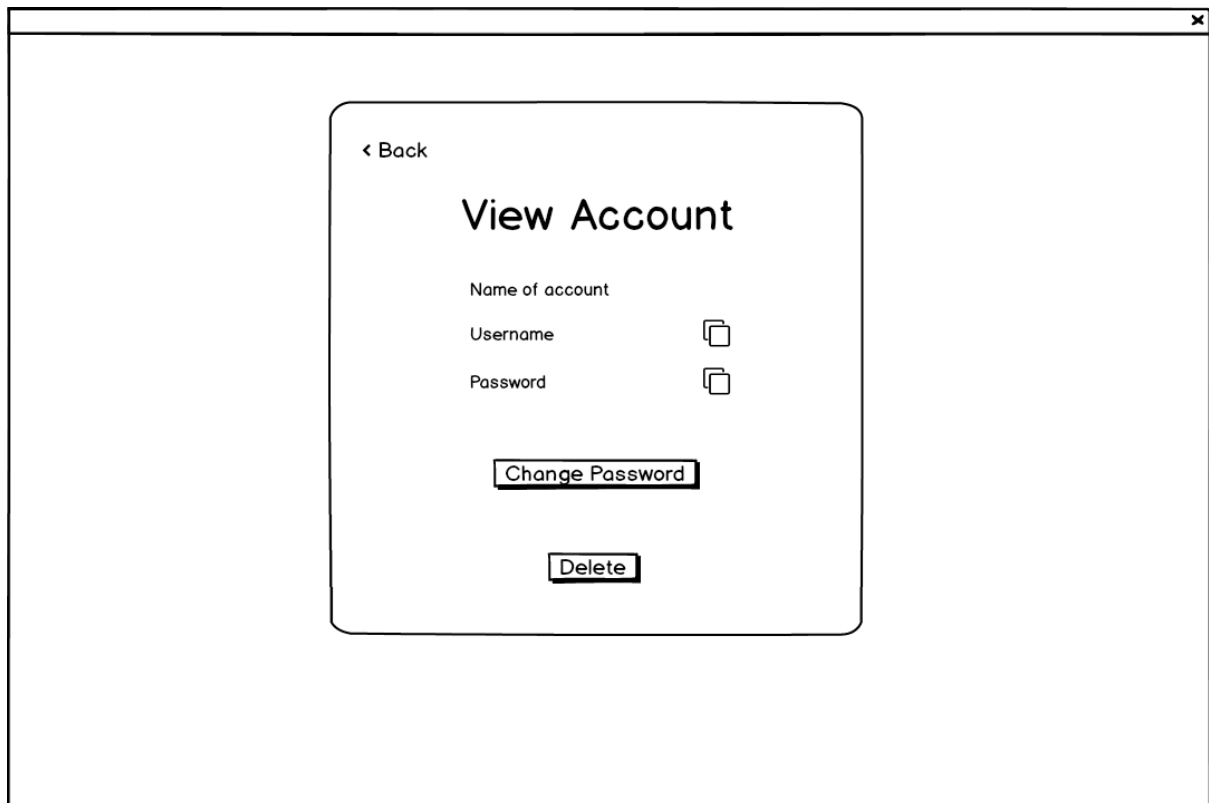
Figure 7 allows users to see all of the accounts they have in their accounts file. They can search for accounts via the search field, add accounts manually via the button with a person icon, view information for an account via the view button for each account on an individual row, import accounts via the import button and export the accounts in the application via the export button.



The image shows a web browser window with a modal dialog box in the center. The dialog box has a title "Add Account" and three input fields labeled "Name of account", "Username", and "Password". Below the input fields are two buttons: "< Cancel" and "Save". The "Save" button is highlighted with a black border. The dialog box is centered on the page and has a white background with a thin black border.

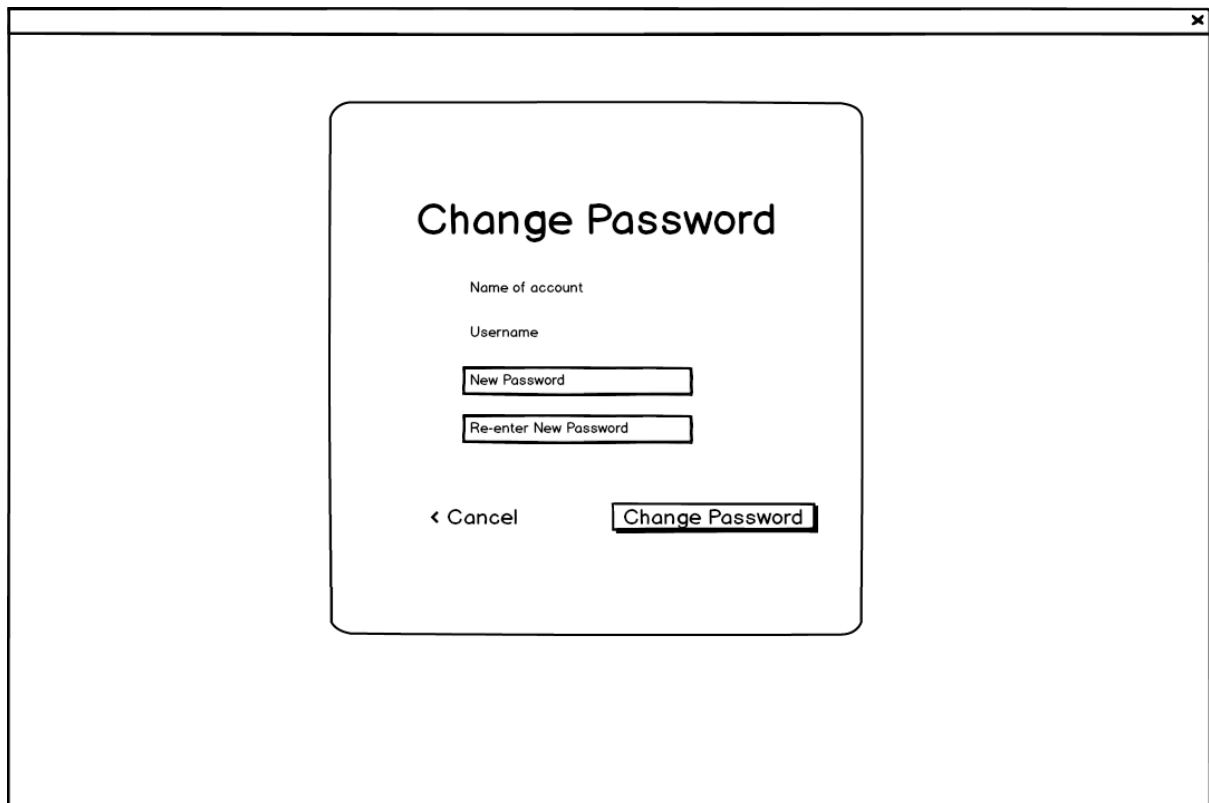
*Figure 8*

Figure 8 allows a user to enter account information manually. This will have error checking to ensure that fields can't be submitted if they contain erroneous data. If an end-user wants to save an account, they must click the save button otherwise, if the cancel button is clicked, they are taken back to the all accounts page.



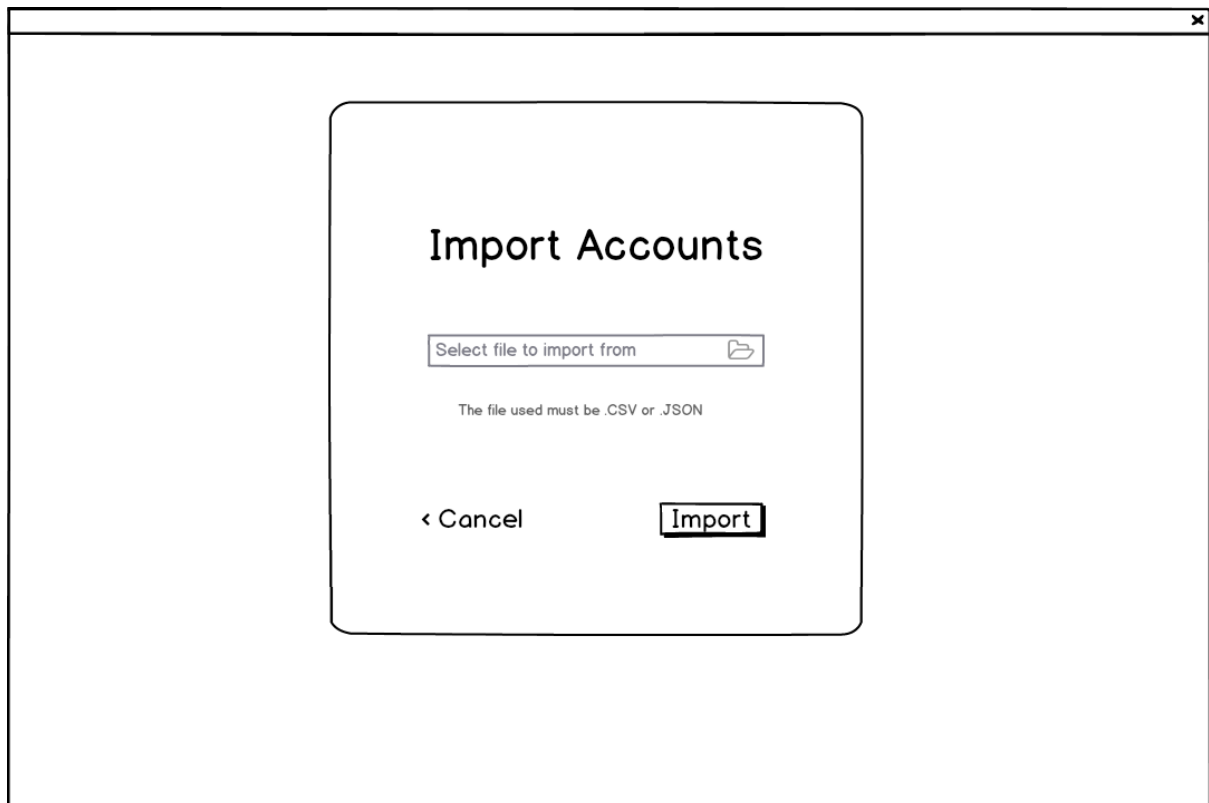
*Figure 9*

Figure 9 retrieves and displays the information for an individual account. The details of the account will be displayed allowing an end-user to copy the username or password of the account via the copy buttons. A user may delete the account via the delete button or change the password for the account by clicking the change password button. To return to the all accounts page the end-user would click the back button.

A screenshot of a web application window titled "Change Password". The window has a standard macOS-style title bar with a close button (X) in the top right corner. The main content area is a light gray rectangle. Inside this rectangle is a white rounded rectangle containing the form. The form has a title "Change Password" in bold black text. Below the title are two labels: "Name of account" and "Username", each followed by a text input field. Below these are two more text input fields labeled "New Password" and "Re-enter New Password". At the bottom of the form are two buttons: a "Cancel" button with a left-pointing arrow and a "Change Password" button.

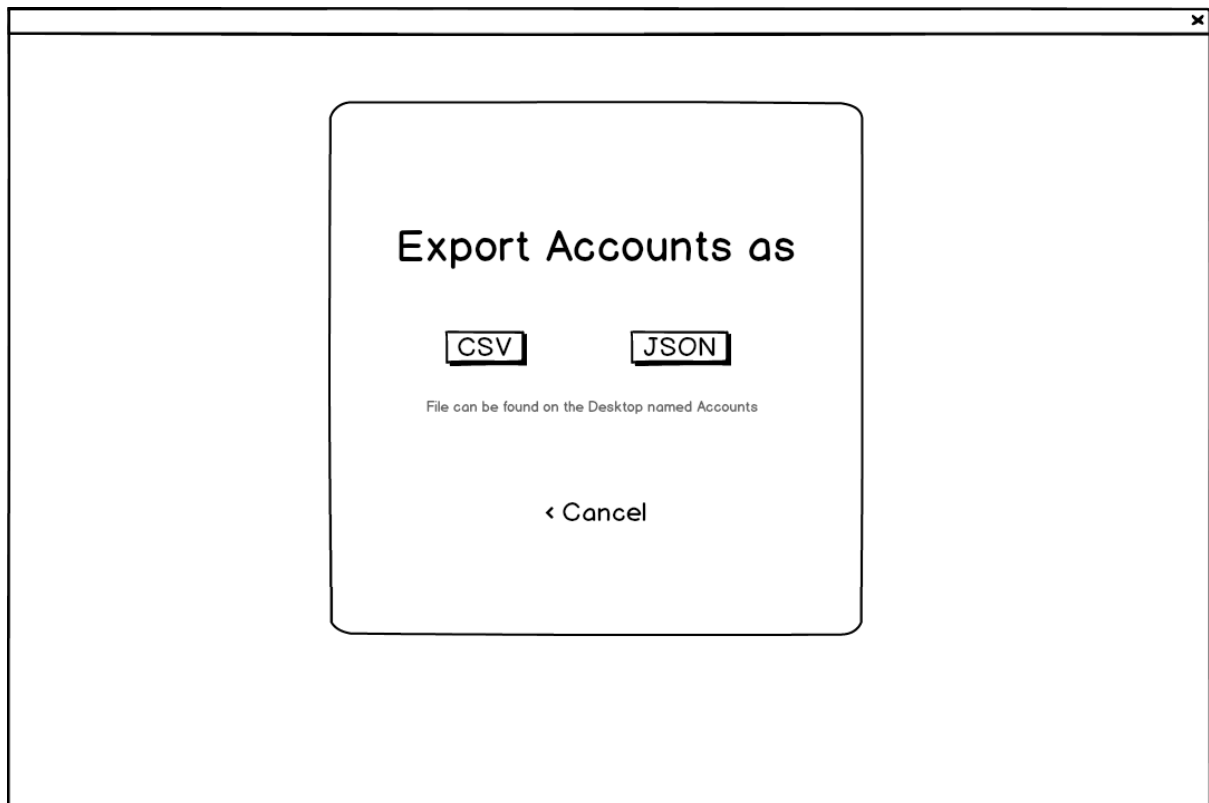
*Figure 10*

Figure 10 is displayed when a user wants to change a password on an account in which they are required to enter the password twice, if the passwords match and don't contain erroneous data then the password will be updated and the user will be brought back to the individual account view where the updated details will be displayed.



*Figure 11*

Figure 11 is displayed when a user wants to import accounts from a file that was exported from a password manager. The user will select a file from the file chooser button, if the file is supported and contains accounts then the accounts will be imported into the application. If the user chooses not to import accounts, they are brought back to the page listing all of the accounts in their vault via the cancel button.



*Figure 12*

Figure 12 is displayed when a user wants to export their accounts from the application. The user can choose to export their accounts as a CSV or JSON document which will be created on the desktop. If a user doesn't want to export their accounts from the application, they are brought back to the all accounts page via the cancel button.

(Balsamiq Studios, LLC, 2008) has been used as a wireframing tool to design the low fidelity diagrams representing the majority of the use cases found in Figure 2. The "Verify key" use case would be an algorithm implemented into the backend of the application which isn't represented in any of the diagrams. Furthermore, the use cases for "Incorrect key error", "Delete account" and "Incorrect file format" are not represented with individual designs as they would be represented by error messages.

## High fidelity

(Adobe Inc., 2020) the software has been used to develop high fidelity wireframes of the application to provide a detailed depiction of how the application should look thus, the development of the application will strive to replicate the style shown in the following diagrams.

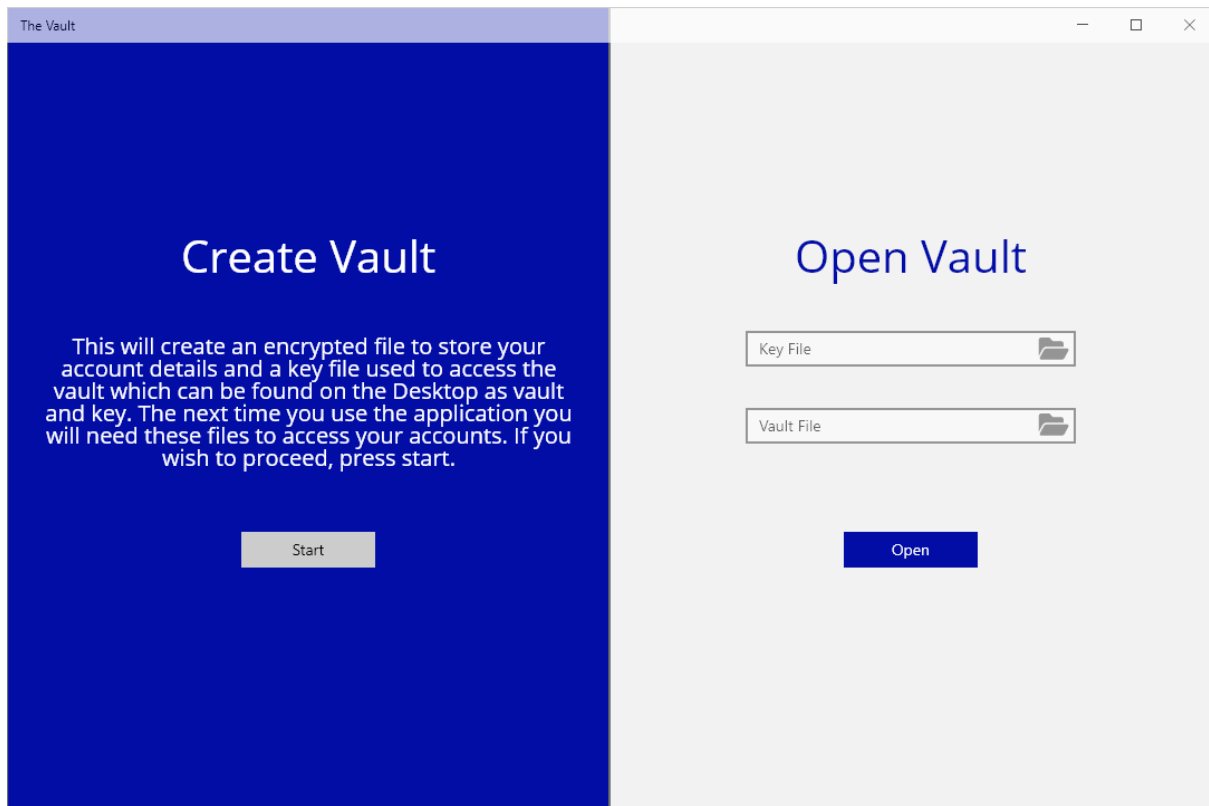


Figure 13



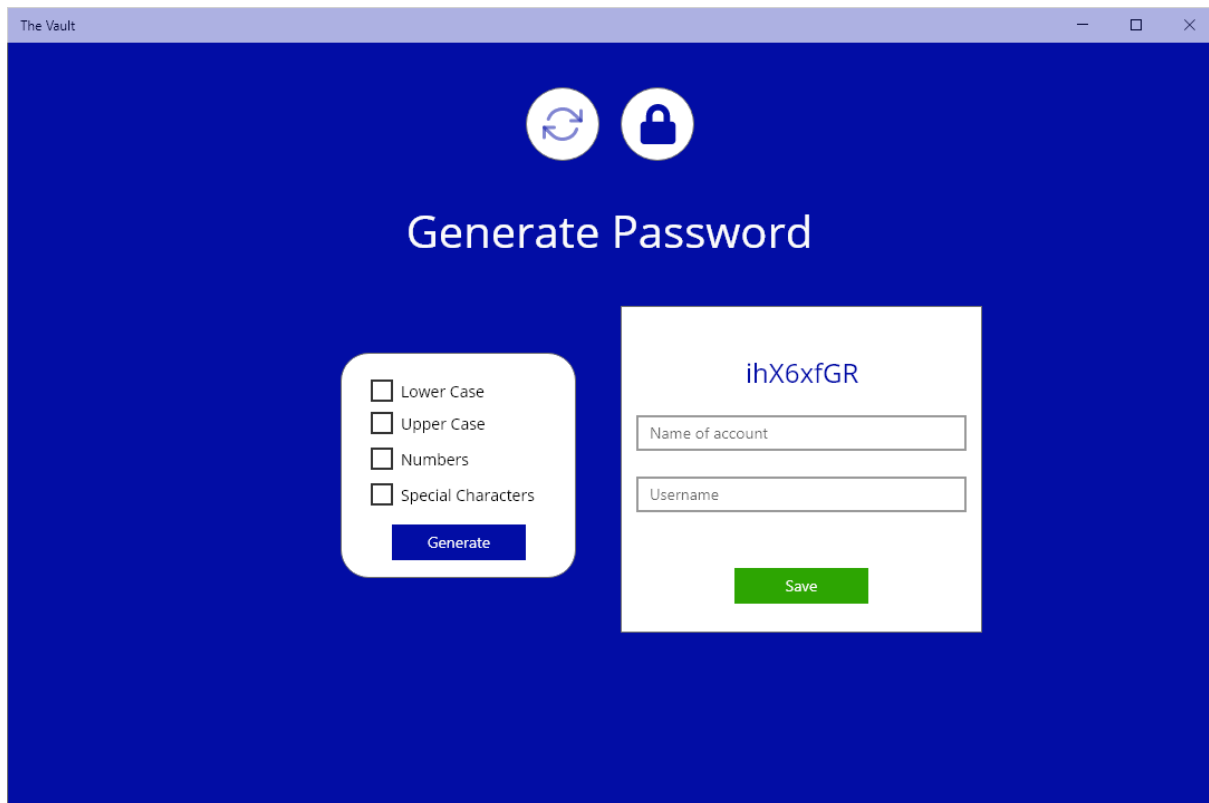


Figure 14

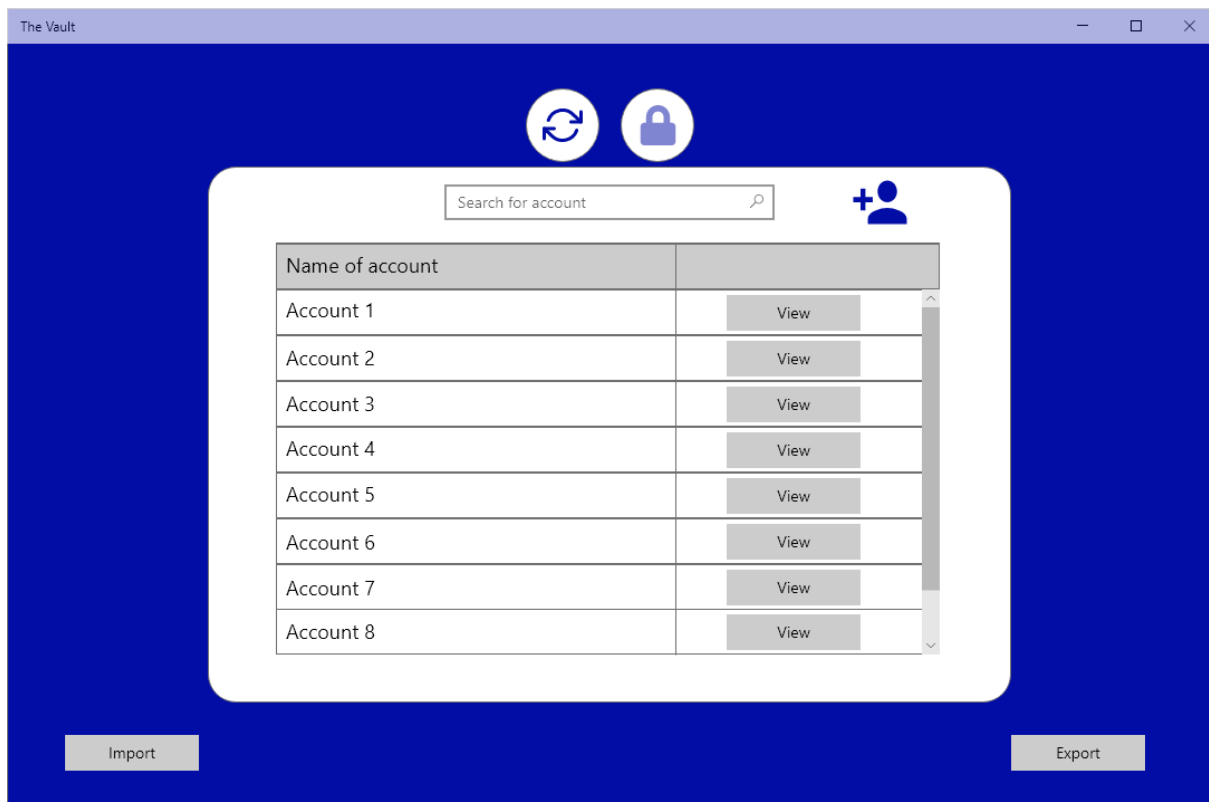


Figure 15

The screenshot shows a window titled "The Vault" with a dark blue background. In the center is a white rounded rectangle titled "Add Account". It contains three input fields: "Name of account", "Username", and "Enter password". Below these fields are two buttons: a red "Cancel" button and a green "Save" button.

Figure 16

The screenshot shows a window titled "The Vault" with a dark blue background. In the center is a white rounded rectangle titled "View Account". At the top left of this rectangle is a grey "Back" button. Below the title are three labels: "Name of account", "Username", and "Password". To the right of "Username" and "Password" are copy icons (two overlapping squares). Below these labels are two buttons: a blue "Change Password" button and a red "Delete" button.

Figure 17

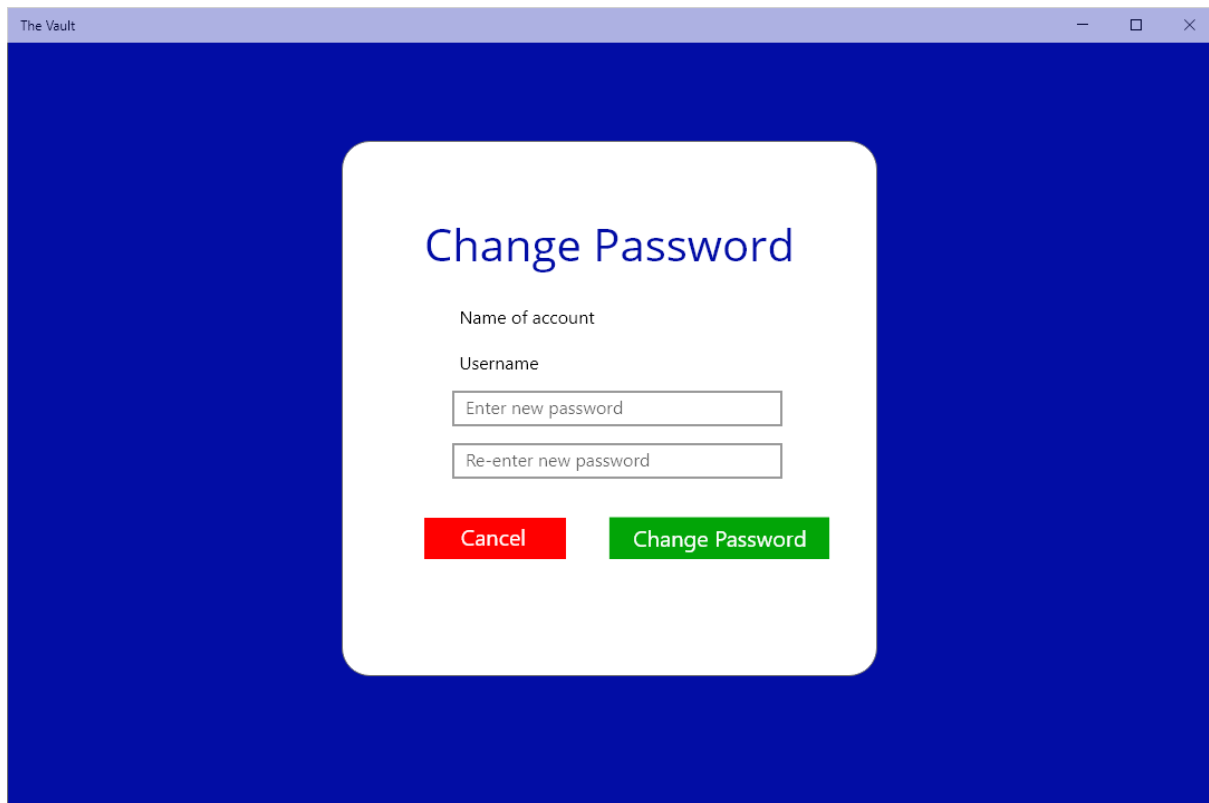


Figure 18

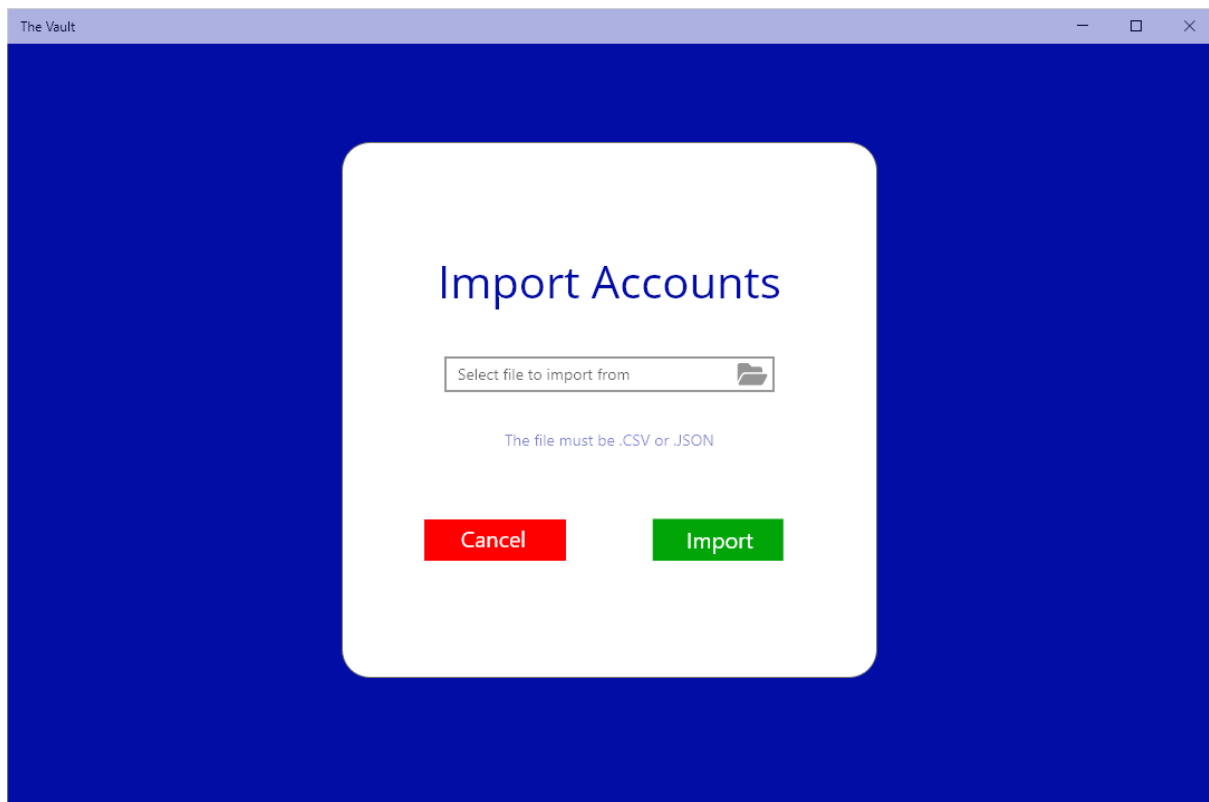
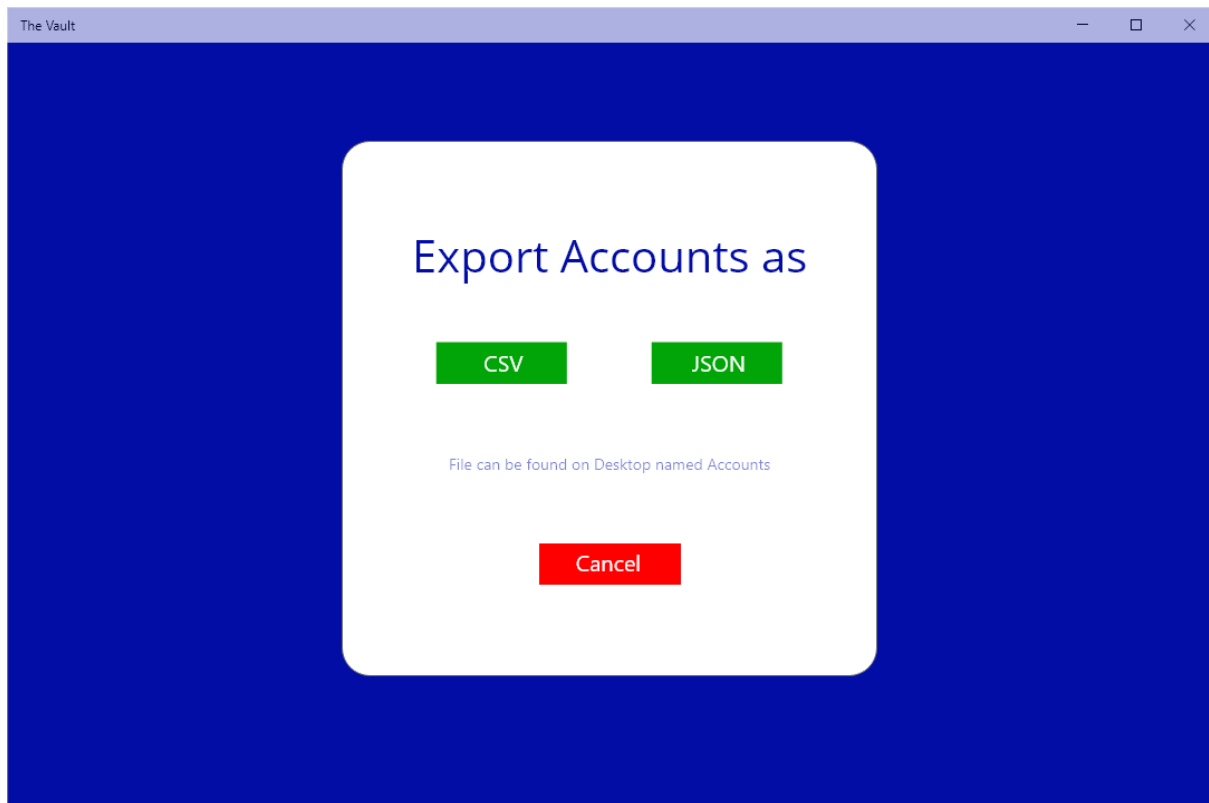


Figure 19



*Figure 20*

## Technical Review

### Similar systems

Bitwarden is mainly written in TypeScript, C# and HTML for use on desktop and mobile operating systems. Furthermore, the platform uses AES 256-bit encryption to ensure that an end user's data is kept secure as AES encryption is considered unbreakable. Additionally, PBKDF2 is used to hash the encryption key used to encrypt and decrypt the data an end-user stores with the platform. Due to the software catering to multiple platforms, a network connection is necessary to access any account details on their server which could inconvenience an end-user that chooses to access their account details while lacking an internet connection. However, the user interface used by the software is simple to interact with and an end-user has a choice to install the system on their server so that all the data an end-user generates is only accessible by users with a connection to the server.

KeePass and KeePassXC operate similarly to Bitwarden as they are both developed using C++ and C# with the main difference between the two platforms being that KeePassXC has a modern user interface which feels intuitive to use whilst KeePass feels dated and slightly difficult to navigate. Both solutions offer the same features while offering more features than their online counterparts like Bitwarden which makes KeePass and KeePassXC difficult to use if an end-user has the intention to only use the software to store and generate passwords. All these software solutions offer AES 256-bit encryption algorithms to store an end user's data and they follow an open-source development methodology so that the wider community can review the code used to develop the systems or contribute directly to the development of the systems. Individuals which review the code used to develop these systems thus legitimise any claims made by the developers of the systems by identifying any flaws which could be used to compromise the system or any data the system accesses.

### Justification

To combat the shortcomings of server solutions such as Bitwarden, the application won't rely on a server connection. Although this will reduce the overall accessibility of the application as users will only be able to use the application on the device with access to the end user's data, this will improve the overall security of an end user's data by limiting the ingress points a hacker could use to get the end-users account data. To improve the usability of the application, features such as secure note storage and RSA key generation shall be omitted from development to prioritise the main functions of the system such as password generation and account storage with import and export capabilities so that new users aren't intimidated by a multitude of features when using the system. Even though the system would be perceived as more customisable if the password to the vault was decided by an end-user, an end-user choosing an adequately complex password is unlikely as they would tend to favour a simplistic password which is more memorable. Therefore, to reduce the risk of a brute-force attack being used to guess the password to the vault, the password to the system will be randomly generated and stored in a "key file" given to the end-user during the generation of the vault.

Although Visual Studio Code provides more features than Atom without extensions and starts faster than Atom as insinuated by (Slant, 2020), the latter is less resource-intensive than Visual Studio Code whilst offering comparative features if plugins are used. The source, whilst lacking reliability of the information advertised, eliminates bias in its depiction of content by depicting the opinions of individuals in the wider community that have reviewed the development environments thus, Atom text editor will be used for the development of the project. Instead of keeping multiple instances of the application locally whilst developing new iterations or features, a version control system such as git will be used to reduce confusion whilst developing the system and will ensure that all copies of the system can be accessed from a remote location, should the system used to develop the application suffer faults.

A programming language such as C would compile and run faster whilst utilising fewer resources, according to (Kamaruzzaman, 2020), Python is in more demand than C therefore, Python shall be used during the development of the application. Furthermore, a Python module such as Tkinter could be used for the development of the graphical user interface as it distributed with most instances of Python and is simple to use, however, as dictated by (amigos-maker, 2020), Tkinter offers fewer features than PyQt and the latter offers a tool, Qt Designer, which allows developers to create the interface without code but developers must code the logic of how the interface should function. Therefore, PyQt will be used for the development of the user interface as it offers detailed documentation and tutorials for creating user interfaces in contrast to Tkinter. Although Python offers a cryptography module to encrypt data, the minimum encryption standard that must be implemented in the application to ensure the security of end-users is AES-256-bit encryption, rendering the cryptography module inadequate, therefore, the Pycryptodome module shall be used.

After comparing the features found in alternative systems, additional requirements have been elicited in means of improving the overall functionality of the project:

#### Non-functional requirements

- The application should have no dependency on a server or internet connection
- The application should generate a secure key for a user to access their accounts
- The application should implement AES-256-bit encryption to ensure the security of the end-users accounts

## Implementation

The programming language used in the project is Python with PyQt 5 being used to generate all graphical user interfaces. The Qt Designer tool was used to develop the user interfaces to ensure they were designed correctly thus, by using the command line *pyuic5* tool, the designs created in Qt Designer were exported to Python code for use in the application. All of the source code can be found at (Bhatti, 2020) with the custom created code under the main.py file while the code generated by *pyuic5* in the following files: AddAccountPage.py, allAccountsPage.py, changePassPage.py, exportAccountsPage.py, genPassPage.py, importAccountsPage.py, startPage.py and viewAccountPage.py. The screencast illustrating the features an end-user would regularly interact with can be found at (Bhatti, 2020) whilst the A3 poster showcasing the software can be found at (Bhatti, 2020).

## Class diagram

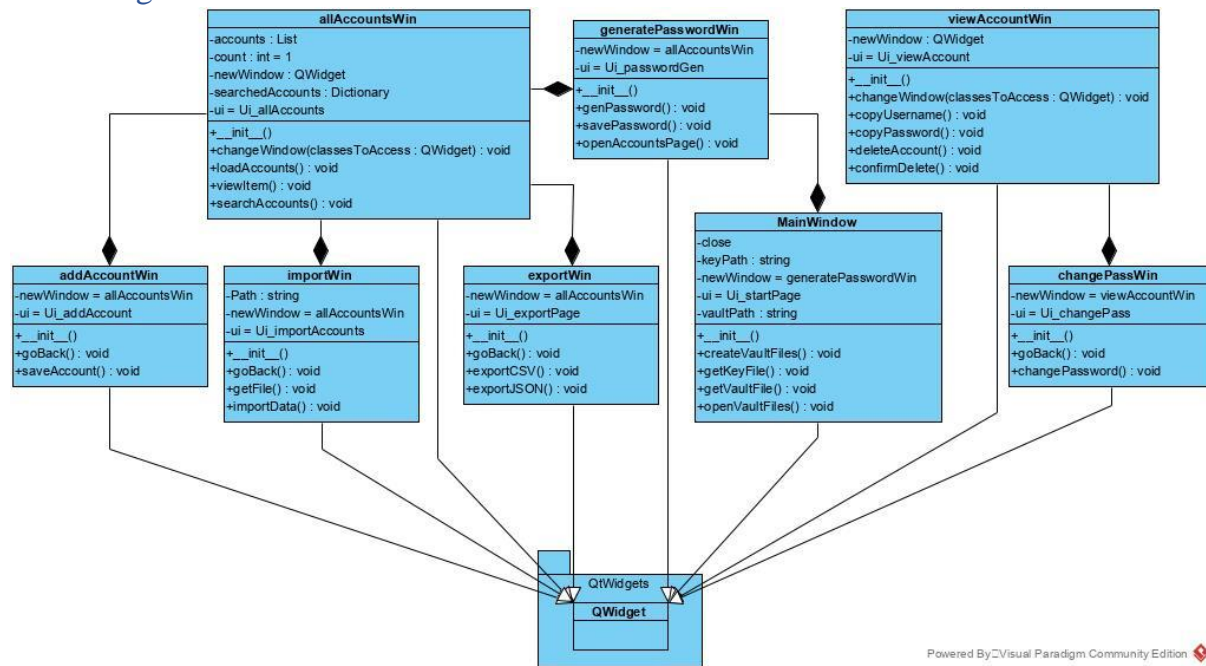


Figure 21

Figure 21 illustrates the classes found in the main.py file to depict the relationships between each of the classes and the interfaces which will be displayed. All of the classes extend the “QWidget” object found inside the “QtWidgets” module and all of the “ui” variables in each of the classes are related to a “ui” function specified in the separate user interface files. As described in (Child, 2020), a composition relationship has been used between each of the classes as a new interface is created after the function in the previous class calls for the new class to generate its related interface.

## Start page

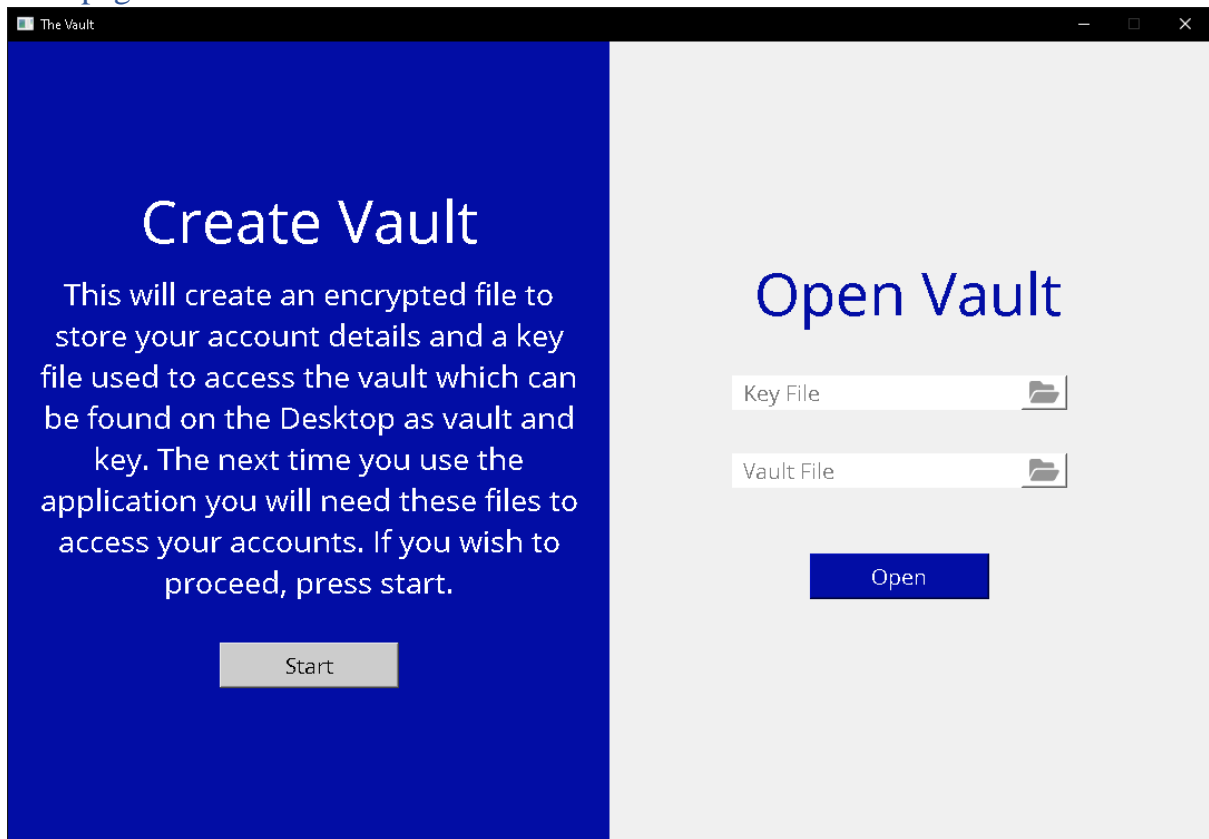


Figure 22

```
class MainWindow(QMainWindow):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.ui = Ui_startPage()
        self.ui.setupUi(self)  # initializes start page
        self.ui.startButton.clicked.connect(self.createVaultFiles)
        self.ui.selectKeyFile.clicked.connect(self.getKeyFile)
        self.ui.selectVaultFile.clicked.connect(self.getVaultFile)
        self.ui.openButton.clicked.connect(self.openVaultFiles)
        # button variables which execute a specific function
```

Figure 23

The code in the constructor generates a user interface using the code in the startPage.py file with each button linking to a separate function. When the “startButton” is clicked, the “createVaultFiles” function under the same class is executed.



```

def createVaultFiles(self):

    key = get_random_bytes(32) # 32 bytes is 256 bits
    data = ''.encode('utf-8') # basic data for file to encrypt
    desktopPath = getPathToDesktop() # gets path to desktop
    keyFile = open(desktopPath + "\\key.bin", "wb")
    keyFile.write(key) # writes encryption key to file
    keyFile.close

    cipher = AES.new(key, AES.MODE_CBC)
    ciphered_data = cipher.encrypt(pad(data, AES.block_size))

    vaultFile = open(desktopPath + "\\vault.bin", "wb") # creates vault file
    vaultFile.write(cipher.iv)
    vaultFile.write(ciphered_data)
    vaultFile.close()

    Alert("Process Completed", QtWidgets.QMessageBox.Information, "Created vault.bin and key.bin")
    # Alert function to reuse the code to generate a QMessageBox

```

*Figure 24*

The encryption algorithm, as illustrated in (Vollebregt, 2019), utilises the PyCryptodome module to write the key to the “keyFile” and create a “vaultFile” thus creating 2 binary files encoded in UTF-8 format. Although a different number of bytes could be used in generating the key which would reduce the overall time complexity in generating the key and vault files, using 32 bytes for the key allows us to implement AES-256-bit encryption which increases the overall security of the accounts stored in the application. Furthermore, the “cipher.iv” variable is used to ensure that if data is written to the “vaultFile”, the hash of the data will always be different thus reducing the likelihood of visibly distinguishing accounts in the “vaultFile”.

```

def getPathToDesktop():

    # path to desktop is different on windows and unix systems as on windows the drive the
    # desktop is on can be changed

    if system() == 'Windows':

        desktopPath = os.environ["HOMEPATH"] + "\\Desktop" # finds path to desktop
        for driveLetter in ascii_uppercase: # find drive desktop folder is on
            if os.path.exists("{0}:{1}".format(driveLetter, desktopPath)):
                desktopPath = "{0}:{1}".format(driveLetter, desktopPath)

    else:

        desktopPath = os.path.join(os.path.join(os.path.expanduser('~')), 'Desktop')

    return desktopPath

```

*Figure 25*

Figure 25 is a global function that utilises the “system” function from the “platform” module to identify the operating system an end-user is executing the program on as indicated by (Dunn, 2020). The “driveLetter” the Desktop is stored on is determined utilising methods in (thefourtheye, 2015) as on Windows the Desktop can be moved to a different drive which, to my knowledge, isn’t possible on other operating systems. Although the code used to validate the path to the Desktop and generate the path to the Desktop vary depending on the hosts operating system, both implementations are dependent on the “os” module as depicted in (user559633, 2015).

```
def Alert(title, icon, text):
    # creates QMessageBox based on arguments in function
    message = QtWidgets.QMessageBox()
    message.setWindowTitle(title)
    message.setIcon(icon)
    message.setText(text)
    message.exec_()
```

*Figure 26*

Figure 26 is a globally accessible function which allows for the reuse of QMessageBoxes which generate dialogue boxes during program execution according to the “title”, “icon” and “text” passed as arguments to the functions’ procedure call.

```
def getKeyFile(self):
    file = QtWidgets.QFileDialog.getOpenFileName(
        self, 'Open file', "", "All Files (*)") # lets user choose files from explorer
    url = QtCore.QUrl.fromLocalFile(file[0]) # gets path to file and stores it as an object
    self.ui.keyFileLabel.setText(url.fileName()) # adjusts file name in gui
    self.ui.keyFileLabel.adjustSize() # adjusts size of text wrapper for file name in gui
    self.keyPath = file[0] # makes keyPath accessible in all of MainWindow class
```

*Figure 27*

Figure 27 is executed when the “selectKeyFile” button is clicked. (eyllanesc, 2018) was applied to get the file name of the key file while (rakshitarora, 2020) was used as a point of reference in updating the label for the key file on the user interface. This is very similar to the implementation of Figure 28 where the vault file is selected in the file manager and the label for the vault file is updated. There was some difficulty trying to implement this section as an attempt to reuse the same function to update the individual vault and key paths didn’t work as intended thus the functionality to get the files and update the labels and variables associated to these files were split into their separate functions:

```

def getVaultFile(self):

    file = QtWidgets.QFileDialog.getOpenFileName(

        self, 'Open file', "", "All Files (*)") # lets user choose files from explorer

    url = QtCore.QUrl.fromLocalFile(file[0])    # gets path to file and stores it as an object

    self.ui.vaultFileLabel.setText(url.fileName()) # adjusts file name in gui

    self.ui.vaultFileLabel.adjustSize()        # adjusts size of text wrapper for file name in gui

    self.vaultPath = file[0]    # makes vaultPath accessible in all of MainWindow class

```

Figure 28

```

def openVaultFiles(self):

    keyFile = self.ui.keyFileLabel.text()

    vaultFile = self.ui.vaultFileLabel.text()

    if (keyFile == "Key File") or (vaultFile == "Vault File"):

        # checks that a Key File or Vault file have been selected

        Alert("Error", QtWidgets.QMessageBox.Critical,

            "Either one or no files were selected. Please select files to open the vault")

        # Alert function to display error QMessageBox

    else:

        # exception handling

        try:

            key, iv, data = getData(self.keyPath, self.vaultPath)

            # display new window for generating password or viewing accounts

            self.newWindow = generatePasswordWin()

            self.newWindow.show() # show new window

            self.hide() # close old window

        except (ValueError, FileNotFoundError) as e:

            Alert("Error", QtWidgets.QMessageBox.Critical, "Incorrect files selected")

            # Alert function to show error message

```

Figure 29

Figure 29 is executed when the “openButton” is clicked. (ekhumoro, 2017) was utilised to retrieve the text from the key and vault file labels. This function checks that a user has selected a key and vault file if they haven't or the key and vault file they have selected couldn't be opened then an error is raised in the exception block, else the application refers the user to the genPassPage.py interface in the “generatePasswordWin” class under the try block. Multiple exceptions were implemented as illustrated in (mechanical\_meat, 2011).

```

def getData(pathToKey, pathToVault):    # allows me to access Paths throughout document
    global KEYPATH, VAULTPATH
    KEYPATH, VAULTPATH = pathToKey, pathToVault
    readVaultFile = open(VAULTPATH, 'rb') # Open the file to read bytes
    iv = readVaultFile.read(16) # Read the iv out - this is 16 bytes long
    ciphered_data = readVaultFile.read() # Read the rest of the data
    readVaultFile.close()
    readKeyFile = open(KEYPATH, 'rb')
    key = readKeyFile.read()
    readKeyFile.close()
    cipher = AES.new(key, AES.MODE_CBC, iv=iv) # Setup cipher
    # Decrypt and then up-pad the result
    data = unpad(cipher.decrypt(ciphered_data), AES.block_size)
    return key, iv, data

```

*Figure 30*

Figure 30 is a globally accessible function which opens the key and vault files by utilising the methods in (Vollebregt, 2019) to retrieve the encryption “key” in the key file and the “iv” and “data” in the vault file. The “iv” ensures that if the same value is written to the vault file, the hash produced for the value is always different.

## Generate password page

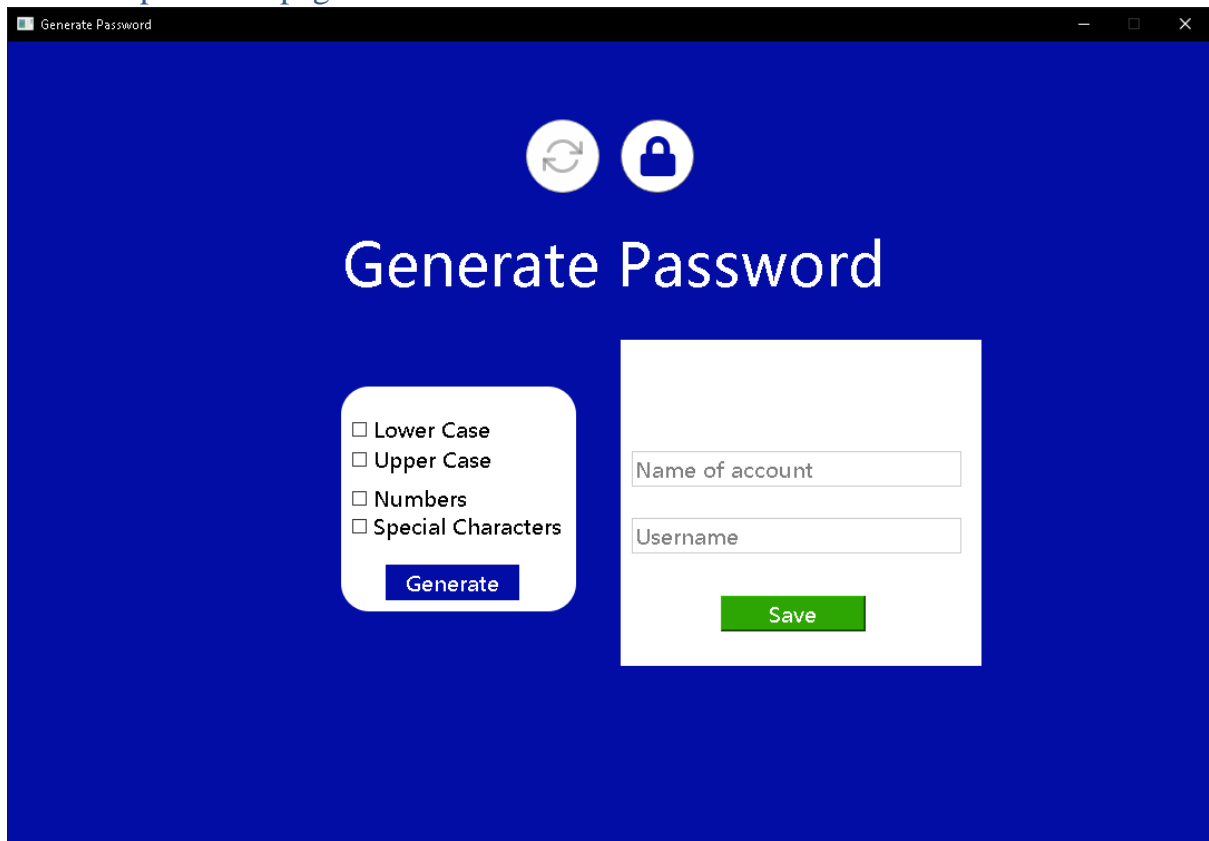


Figure 31

```
class generatePasswordWin(QtWidgets.QWidget):  
    # displays generate password window when vault is open  
    def __init__(self, *args, **kwargs):  
        super().__init__(*args, **kwargs)  
        self.ui = Ui_passwordGen()  
        self.ui.setupUi(self)  
        self.ui.genBtn.clicked.connect(self.genPassword)  
        self.ui.saveBtn.clicked.connect(self.savePassword)  
        self.ui.viewAccountsTab.clicked.connect(self.openAccountsPage)
```

Figure 32

The code in the constructor utilises the user interface generated in the genPassPage.py file.

```

def genPassword(self):
    passwordOptions = ""
    if self.ui.lowerCaseCheck.isChecked() or self.ui.upperCaseCheck.isChecked() or
self.ui.numbersCheck.isChecked() or self.ui.specialCharsCheck.isChecked():
        if self.ui.lowerCaseCheck.isChecked():
            passwordOptions += ascii_lowercase
        if self.ui.upperCaseCheck.isChecked():
            passwordOptions += ascii_uppercase
        if self.ui.numbersCheck.isChecked():
            passwordOptions += digits
        if self.ui.specialCharsCheck.isChecked():
            passwordOptions += punctuation.replace(',', '')
        lengths = [i for i in range(8, 17)]
        passLength = random.choice(lengths)
        password = ""
        for i in range(0, passLength):
            password += random.choice(passwordOptions)
        self.ui.generatedPassLabel.setText(password)
        self.ui.nameOfAccountEdit.setEnabled(True)
        self.ui.usernameEdit.setEnabled(True)
        self.ui.saveBtn.setEnabled(True)
    else:
        Alert("Error", QtWidgets.QMessageBox.Critical, "No options to generate password from")

```

*Figure 33*

Figure 33 is an embedded function in the “generatePasswordWin” class which is called when the end-user selects options from the checkboxes to be used in the generated password and clicks the “genBtn”. The punctuation values and the ASCII values from the “string” module, as dictated by (user2555451, 2015), have been used to add the individual values that would be added to the possible password generation values. The comma value has been removed from the punctuation values to reduce erroneous results when writing to CSV or JSON files in the future. Methods in (rakshitarora, 2020) were adopted to check if the checkboxes have been checked. If any of the checkboxes have been checked, a password is generated for a random length between 8 and 17 characters where the generated password is displayed on the screen and (Luengo, 2015) has been referenced to enable the disabled text and button fields to save a generated password with account information. However, if none of the checkboxes are selected on submission of the “genBtn”, an error message is passed to the “Alert” function to display an appropriate dialogue box.

```

def savePassword(self):
    if (self.ui.nameOfAccountEdit.text() == (None or "")) or (self.ui.usernameEdit.text() == (None
or "")):
        Alert("Error", QtWidgets.QMessageBox.Critical,
            "Account name or Username has been left empty")
    else: # displays any error message if the user input fields are empty or incorrectly entered
        if (self.ui.nameOfAccountEdit.text()[0] == " ") or (self.ui.nameOfAccountEdit.text()[-1]
== " "):
            Alert("Error", QtWidgets.QMessageBox.Critical,
                "Please remove spaces from the beginning or end of Account name")
        elif " " in self.ui.usernameEdit.text():
            Alert("Error", QtWidgets.QMessageBox.Critical,
                "Please remove spaces from Username")
        elif ("," in self.ui.nameOfAccountEdit.text()) or ("," in self.ui.usernameEdit.text()):
            Alert("Error", QtWidgets.QMessageBox.Critical,
                "Please remove commas from name of account or username")
        else:
            nameOfAccount = self.ui.nameOfAccountEdit.text()
            username = self.ui.usernameEdit.text()
            password = self.ui.generatedPassLabel.text()
            writeData(nameOfAccount, username, password)
            Alert("Process Completed", QtWidgets.QMessageBox.Information, "Account saved")
            # reset check boxes after saving accounts
            self.ui.lowerCaseCheck.setChecked(False)
            self.ui.upperCaseCheck.setChecked(False)
            self.ui.numbersCheck.setChecked(False)
            self.ui.specialCharsCheck.setChecked(False)
            # the code below resets that generatedPassLabel, nameOfAccount input and username
input after saving
            self.ui.generatedPassLabel.setText("")
            self.ui.nameOfAccountEdit.setText("")
            self.ui.usernameEdit.setText("")
            self.ui.nameOfAccountEdit.setEnabled(False)
            self.ui.usernameEdit.setEnabled(False)

```

*Figure 34*

Figure 34 is executed when the “saveBtn” is clicked in which the name of account and username fields are checked for any erroneous data to which an error message would be displayed using the “Alert” function, however, if the name of account and username fields pass the checks in the function, the name of the account, username and password are passed to the “writeData” function to write the

account to the vault file. After writing the accounts to the file the information on the page is reset and a message is displayed confirming that the account has been saved through the “Alert” function.

```
def writeData(nameOfAccount, username, password): # writes name of account, username and
password to vaultFile

    global KEYPATH, VAULTPATH

    key, iv, data = getData(KEYPATH, VAULTPATH)

    data += ("{}", {}, {} \n".format(nameOfAccount, username, password)).encode('utf-8')

    cipher = AES.new(key, AES.MODE_CBC, iv=iv)

    ciphered_data = cipher.encrypt(pad(data, AES.block_size))

    vaultFile = open(VAULTPATH, "wb") # creates vault file

    vaultFile.write(cipher.iv)

    vaultFile.write(ciphered_data)

    vaultFile.close()
```

*Figure 35*

Figure 35 is a global function which writes the name of the account, username and password of an account to the vault file. The vault file is read to retrieve the accounts in the file then the new account details are added to the end of the variable used to store the accounts which are then rewritten to the vault file. As the vault file is encoded in UTF-8, (kame, 2016) was implemented to encode the name of account, username and password into a UTF-8 format.

```
def openAccountsPage(self): # opens window to view all accounts

    self.newWindow = allAccountsWin()

    self.newWindow.show() # show new window

    self.hide() # close old window
```

*Figure 36*

Figure 36 is executed if the “viewAccountsTab” button is clicked to which the code in the “allAccountsWin” class generates the user interface for the allAccountsPage.py file.



## View all accounts page

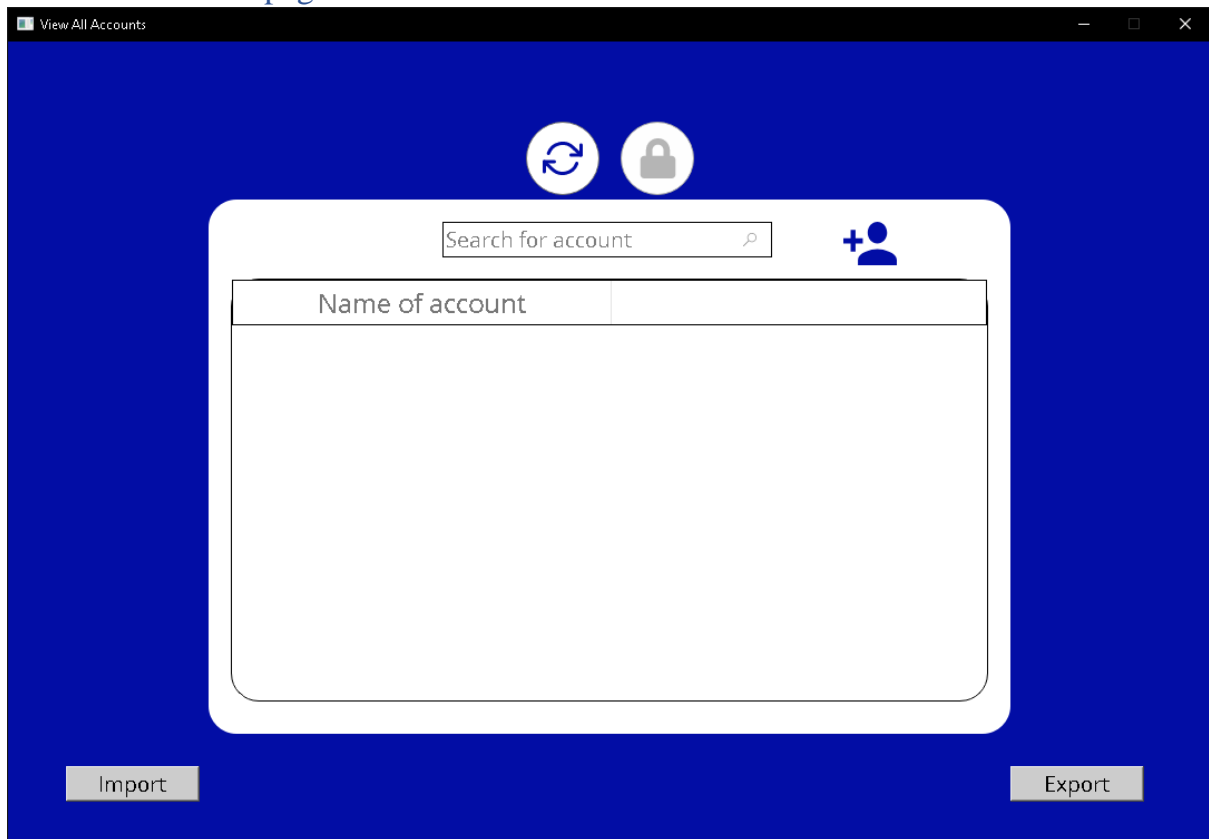


Figure 37

```
class allAccountsWin(QtWidgets.QWidget):    # view all accounts window

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

        self.ui = Ui_allAccounts()
        self.ui.setupUi(self)

        # button which links to generate password window
        self.ui.genPassTab.clicked.connect(lambda: self.changeWindow(generatePasswordWin()))

        self.loadAccounts()

        self.ui.accountsTable.itemClicked.connect(self.viewItem)

        self.ui.addAccountBtn.clicked.connect(lambda: self.changeWindow(addAccountWin()))

        self.ui.searchBox.returnPressed.connect(self.searchAccounts)

        self.ui.importBtn.clicked.connect(lambda: self.changeWindow(importWin()))

        self.ui.exportBtn.clicked.connect(lambda: self.changeWindow(exportWin()))
```

Figure 38

“allAccountsWin” class implements the user interface found in the allAccountsPage.py file. The “genPassTab” button works similarly to the implementation of the “generatePassWin” however, it points to the “generatePassWin” class instead of the “allAccountsWin” class which is similar to the “addAccountBtn”, “importBtn” and “exportBtn” which points to “addAccountWin” class, the “importWin” class and “exportWin” class respectively. All of the buttons which direct to a different

window use the “lambda” function to pass the class that the button will use to create a new window to the “changeWindow” function.

```
def changewindow(self, classToAccess): # takes new window argument
    self.newWindow = classToAccess
    self.newWindow.show() # show new window
    self.hide() # close old window
```

*Figure 39*

Figure 39 is a method in the “allAccountsWin” class which takes a class passed to the “classToAccess” variable to change the window that is displayed to an end-user.

```

def loadAccounts(self): # added feature to read accounts from file

    global KEYPATH, VAULTPATH

    self.searchedAccounts = {}

    self.ui.accountsTable.setEditTriggers(QtWidgets.QTableWidget.NoEditTriggers)

    key, iv, data = getData(KEYPATH, VAULTPATH)

    data = data.decode('utf-8')

    self.count = 1 # count for resetting all accounts view

    if data != "":

        row = data.split('\n')

        self.accounts = {}

        i = 0

        for value in row:

            if value != "":

                self.accounts[i] = value.split(',')

                i += 1

    self.ui.accountsTable.setRowCount(0) # removes all data in table before making table

    for n, key in enumerate(sorted(self.accounts.keys())): # displays code in table in window

        self.ui.accountsTable.insertRow(n)

        newitem = QtWidgets.QTableWidgetItem(self.accounts[key][0])

        viewLabel = QtWidgets.QTableWidgetItem("View")

        viewLabel.setTextAlignment(QtCore.Qt.AlignCenter)

        self.ui.accountsTable.setItem(n, 0, newitem)

        self.ui.accountsTable.setItem(n, 1, viewLabel)

        viewLabel.setBackground(QtGui.QColor(210, 210, 210))

        viewLabel.setFlags(viewLabel.flags() ^ QtCore.Qt.ItemIsEditable)

    else: # else disables table

        self.ui.accountsTable.setEnabled(False)

        self.ui.searchBox.setEnabled(False)

```

*Figure 40*

Figure 40 reads all of the accounts in the vault file by calling the “getData” function and refers to (<https://pythonbasics.org>, 2020) to display all of the accounts as separate elements in a table displayed in the user interface. If no accounts are read from the vault file, the search box field is disabled.

```

def viewItem(self):
    global VIEWEDITEM

    if (self.ui.accountsTable.currentItem().text() == "View") and
(self.ui.accountsTable.currentColumn() == 1):

        row = self.ui.accountsTable.currentRow()

        if not(self.searchedAccounts): # checks if searchedAccounts is empty
            VIEWEDITEM = self.accounts[row]
        else:
            for n, key in enumerate(sorted(self.searchedAccounts.keys())):
                if row == n:
                    VIEWEDITEM = self.accounts[key]

        self.changeWindow(viewAccountWin())

```

*Figure 41*

Figure 41 is executed when an item in the accounts table is selected to be viewed. (The Qt Company, 2020) has been referenced to get the current item a user has selected in the accounts table to identify if the value that they have selected is “View” and in the second column. If the condition is true, a new window is created displaying the individual accounts details otherwise, nothing happens. The “self.changeWindow” function is passed the “viewAccountWin” class to change the window to an individual window displaying the details of an account.

```

def searchAccounts(self):
    term = self.ui.searchBox.text()
    if term != (None or ""):
        self.searchedAccounts = self.accounts.copy() # copy sets values to new variable to edit
        self.count -= 1 # decreases count for table to reset when nothing in searchBox
        self.ui.accountsTable.setRowCount(0) # deletes tables contents
        for n, key in enumerate(sorted(self.accounts.keys())): # displays code in table in window
            if not(term.lower() in self.accounts[key][0].lower()):
                self.searchedAccounts.pop(key) # removes values not in search
        # code below works just like in loadAccounts but with search terms
        for n, key in enumerate(sorted(self.searchedAccounts.keys())):
            self.ui.accountsTable.insertRow(n)
            newitem = QTableWidgetItem(self.searchedAccounts[key][0])
            viewLabel = QTableWidgetItem("View")
            viewLabel.setTextAlignment(QtCore.Qt.AlignCenter)
            self.ui.accountsTable.setItem(n, 0, newitem)
            self.ui.accountsTable.setItem(n, 1, viewLabel)
            viewLabel.setBackground(QtGui.QColor(210, 210, 210))
            viewLabel.setFlags(viewLabel.flags() ^ QtCore.Qt.ItemIsEditable)
    else: # if search box is empty
        if self.count <= 0: # comparison to make sure you only run loadAccounts after a search
            self.searchedAccounts = {}
            self.loadAccounts()

```

*Figure 42*

Figure 42 is a function in the “allAccountsWin” class which retrieves the data that was inputted in the search box and cross-references the searched term with the values in the table. If the term is found in the accounts table, the table is updated to only display the accounts which share similar data to the search term. As the search box has no submission button, methods in (user1006989, 2013) were exercised to detect if the enter button was pressed thus, to trigger a search a user must press the enter button after entering a search term. (Striver, 2018) was referenced to copy the “accounts” set to another variable which can be manipulated. To remove all of the accounts that were in the table before the search, (alexisdsm, 2013) was implemented. To display the new table with contains accounts that are referenced by the search term, (<https://pythonbasics.org>, 2020) was cited to create the new table whilst methods in (ekhumoro, 2017) were used to align the “View” label in the table alongside each of the account items.

## Add account page

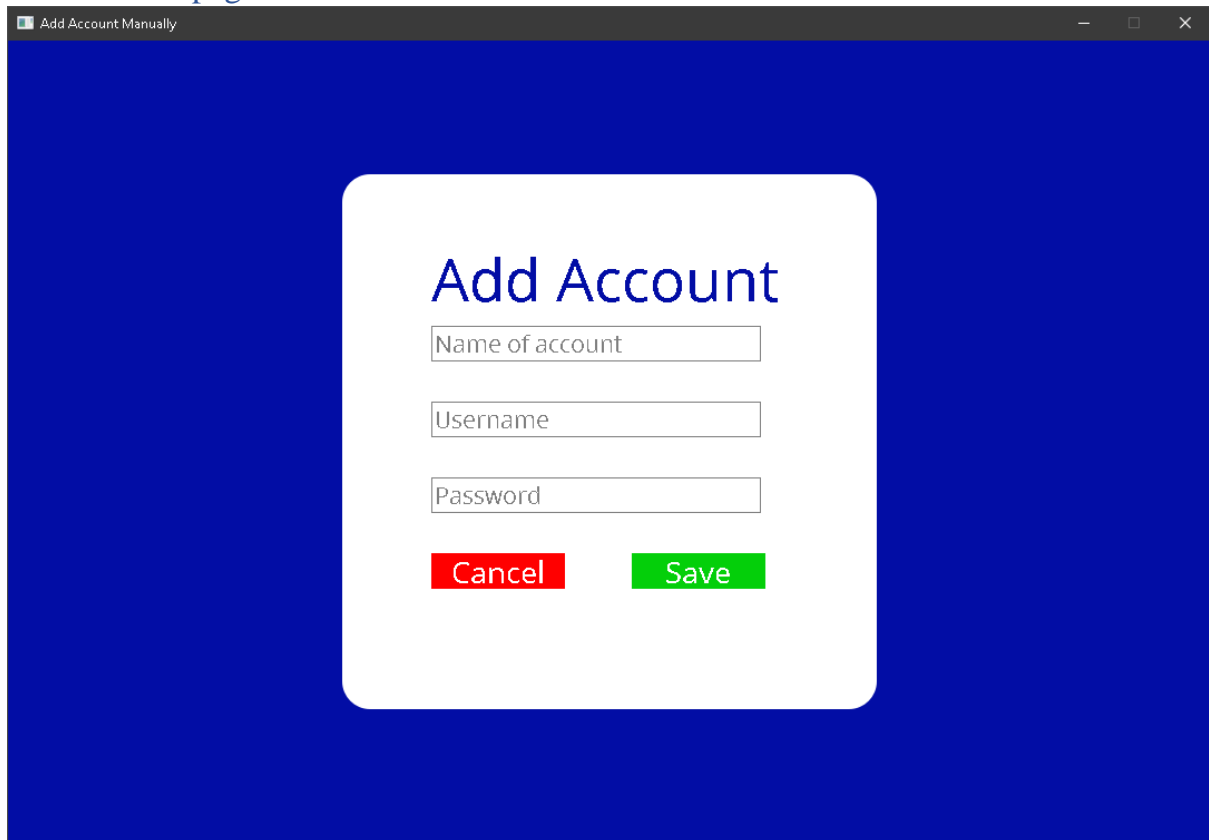


Figure 43

```
class addAccountWin(QtWidgets.QWidget):  
    def __init__(self, *args, **kwargs):  
        super().__init__(*args, **kwargs)  
        self.ui = Ui_addAccount()  
        self.ui.setupUi(self)  
        self.ui.cancelBtn.clicked.connect(self.goBack)  
        self.ui.saveBtn.clicked.connect(self.saveAccount)
```

Figure 44

Figure 44 is a class which creates the user interface in the addAccountPage.py file.

```
def goBack(self):

    self.newWindow = allAccountsWin()

    self.newWindow.show()

    self.hide()
```

Figure 45

When the “cancelBtn” is clicked, the “goBack” method in the “addAccountWin” class is executed which closes the “addAccountPage” and creates an interface by calling the “allAccountsWin” class. This is similar to the “changeWindow” method under the “allAccountsWin” class, however, instead of being called from a button which utilises a lambda function to pass the class that will generate the new window, the class that generates the new window is already specified in the function.

```
def saveAccount(self):

    if (self.ui.nameOfAccountEdit.text() == (None or "")) or (self.ui.usernameEdit.text() == (None or "")) or (self.ui.passwordEdit.text() == (None or "")):

        Alert("Error", QtWidgets.QMessageBox.Critical,

            "Account name, Username or the Password field has been left empty")

    else: # displays any error message if the user input fields are empty or incorrectly entered

        if (self.ui.nameOfAccountEdit.text()[0] == " ") or (self.ui.nameOfAccountEdit.text()[-1] == " "):

            Alert("Error", QtWidgets.QMessageBox.Critical,

                "Please remove spaces from the beginning or end of Account name")

        elif (" " in self.ui.usernameEdit.text()) or (" " in self.ui.passwordEdit.text()):

            Alert("Error", QtWidgets.QMessageBox.Critical,

                "Please remove spaces from Username or Password")

        elif ("," in self.ui.nameOfAccountEdit.text()) or ("," in self.ui.usernameEdit.text()) or ("," in self.ui.passwordEdit.text()):

            Alert("Error", QtWidgets.QMessageBox.Critical,

                "Please remove commas from Name of account, Username or Password")

    else:

        nameOfAccount = self.ui.nameOfAccountEdit.text()

        username = self.ui.usernameEdit.text()

        password = self.ui.passwordEdit.text()

        writeData(nameOfAccount, username, password)

        Alert("Process Completed", QtWidgets.QMessageBox.Information, "Account saved")

        self.goBack()
```

Figure 46

Figure 46 is a method under the “addAccountWin” class which is executed when the “saveBtn” is clicked. This function checks that all input fields aren’t empty and that there is no erroneous data in any of the input fields such as spaces in a username or password and that there are no commas in any

of the input fields which ensures that there are no issues when writing to CSV files in the future. If the input fields pass all of the error checking implemented in this function, the name of the account, username and password entered are passed to the “writeData” function to be written to the vault file. Dialogue messages are implemented via the “Alert” function to give the end-user visual feedback for any errors that have been encountered or to confirm that the account has been saved.

## View account page

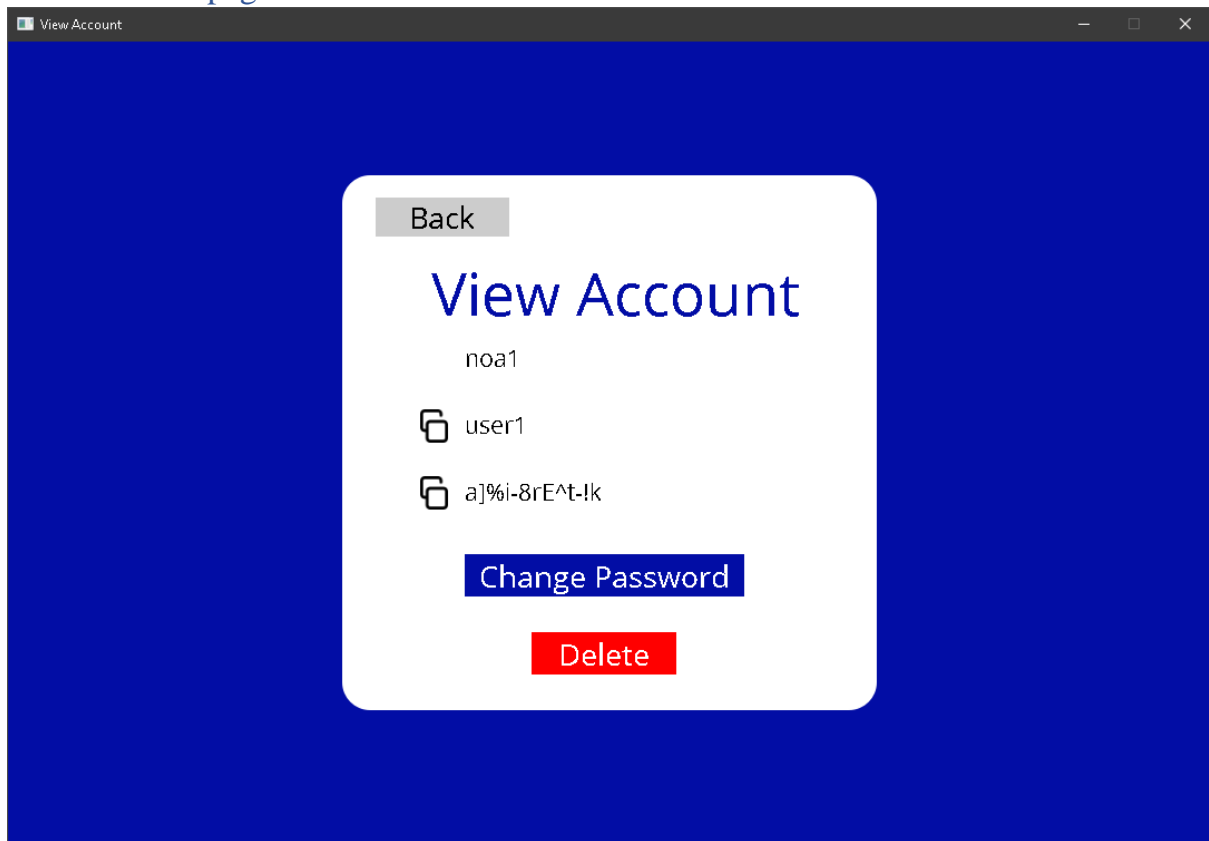


Figure 47



```

class viewAccountWin(QtWidgets.QWidget):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.ui = Ui_viewAccount()
        self.ui.setupUi(self)
        self.ui.backBtn.clicked.connect(lambda: self.changeWindow(allAccountsWin()))
        self.ui.nameOfAccountLbl.setText(VIEWEDITEM[0])
        self.ui.nameOfAccountLbl.adjustSize()
        self.ui.usernameLbl.setText(VIEWEDITEM[1])
        self.ui.usernameLbl.adjustSize()
        self.ui.passwordLbl.setText(VIEWEDITEM[2])
        self.ui.passwordLbl.adjustSize()
        self.ui.copyUserBtn.clicked.connect(self.copyUsername)
        self.ui.copyPassBtn.clicked.connect(self.copyPassword)
        self.ui.changePassBtn.clicked.connect(lambda: self.changeWindow(changePassWin()))
        self.ui.deleteBtn.clicked.connect(self.deleteAccount)

```

*Figure 48*

The “viewAccountWin” class implements the user interface defined in the viewAccountPage.py file. Similarly, the lambda function has been used to pass any classes which generate an interface to the “changeWindow” method located in the “viewAccountWin” class as seen in the “allAccountsWin” class. Furthermore, the selected account is stored in the global variable “VIEWEDITEM” to retrieve the details of the account.

```

def copyUsername(self):
    cb = QtGui.QGuiApplication.clipboard()
    cb.setText(self.ui.usernameLbl.text(), mode=cb.Clipboard)
    Alert("Confirmed", QtWidgets.QMessageBox.Information,
          "Username copied to clipboard")

```

*Figure 49*

When the “copyUserBtn” is clicked, the “copyUsername” method in the “viewAccountWin” class is executed which copies the text in the username label to the systems clipboard as depicted in (The Qt Company, 2020). This is similar to the implementation of “copyPassword” which is executed after the “copyPassBtn” is clicked. In both instances, the “Alert” function is used to display confirmation messages that the associated text field has been copied.

```
def copyPassword(self):  
    cb = QtGui.QGuiApplication.clipboard()  
    cb.setText(self.ui.passwordLbl.text(), mode=cb.Clipboard)  
    Alert("Confirmed", QtWidgets.QMessageBox.Information,  
          "Password copied to clipboard")
```

*Figure 50*

```
def deleteAccount(self):  
    message = QtWidgets.QMessageBox()  
    message.setWindowTitle("Warning")  
    message.setIcon(QtWidgets.QMessageBox.Warning)  
    message.setText("Are you sure you want to delete the account?")  
    message.setStandardButtons(QtWidgets.QMessageBox.Yes |  
QtWidgets.QMessageBox.Cancel)  
    message.setDefaultButton(QtWidgets.QMessageBox.Cancel)  
    message.buttonClicked.connect(self.confirmDelete)  
    message.exec_()
```

*Figure 51*

Figure 51 is executed when the “deleteBtn” is clicked which creates a dialogue box to confirm the deletion of the account that is being viewed. This implementation of QMessageBox is similar to the implementation used in the “Alert” function, however, this instance displays multiple buttons instead of one button when the message is displayed. Also, the cancel button is set as the default button so that a user which clicks the delete button when trying to change a password doesn’t accidentally delete the account they are viewing. This in turn makes the process of deleting an account more time consuming as a user has to actively select the “Yes” button on the dialogue box instead of the option automatically being selected.

```

def confirmDelete(self, clickedBtn):

    if clickedBtn.text() == "&Yes":

        key, iv, data = getData(KEYYPATH, VAULTPATH)

        data = data.decode('utf-8')

        row = data.split('\n')

        accounts = []

        for value in row:

            if value != "":

                # stores accounts as nested lists seperated by value

                accounts.append(value.split(','))

        for account in accounts:

            if account == VIEWEDITEM:

                index = accounts.index(account)

                accounts.pop(index)

                # when this code was a for loop in range len(accounts) sometimes it would give

                # a random error when lots of accounts were added and then someone attempts to

delete an account

                # although the code is now longer, this fixes the index error issue

                updateAccounts(accounts)    # calls updateAccounts

                self.changeWindow(allAccountsWin())

```

*Figure 52*

When the dialogue box for confirming deleting appears the “confirmDelete” function is executed when an end-user selects to cancel or confirm the deletion of an account. If the user selects the cancel button then the dialogue box closes and the account isn’t deleted. However, if the user selects to delete the account, the account is found in the vault file by calling the “getData” method to get all of the accounts, then the account is removed from a variable which stores all of the accounts in the vault file. The variable that has been manipulated is then passed to the globally accessible “updateAccounts” function to write the change to the vault file. After the change has been written the “changeWindow” method in the “viewAccountWin” class is passed the “allAccountsWin” class to update the interface displayed to an end-user. Some issues were encountered when implementing this section as an index error would occur in random intervals of repeated program execution. To resolve this (Coventry, 2017) was used to rectify the error caused by the index of the account that was confirmed for deletion not being retrieved correctly.

```

def updateAccounts(data):
    global KEYPATH, VAULTPATH
    key, iv, oldData = getData(KEYPATH, VAULTPATH)
    accounts = []
    for value in data:
        row = ','.join(value)
        accounts.append(row)
    newData = b''
    for line in accounts:
        newData += ("{}\n".format(line)).encode('utf-8')
    cipher = AES.new(key, AES.MODE_CBC, iv=iv)
    ciphered_data = cipher.encrypt(pad(newData, AES.block_size))
    vaultFile = open(VAULTPATH, "wb")      # creates vault file
    vaultFile.write(cipher.iv)
    vaultFile.write(ciphered_data)
    vaultFile.close()

```

*Figure 53*

Figure 53 is a globally accessible function which takes a list parameter containing all of the accounts in the vault file and overwrites the data stored in the vault file. This is similar to the “writeData” function, however, instead of writing the data that has been retrieved from the “getData” function, the data passed to the function when the function is called is written to the vault file.

## Change password page

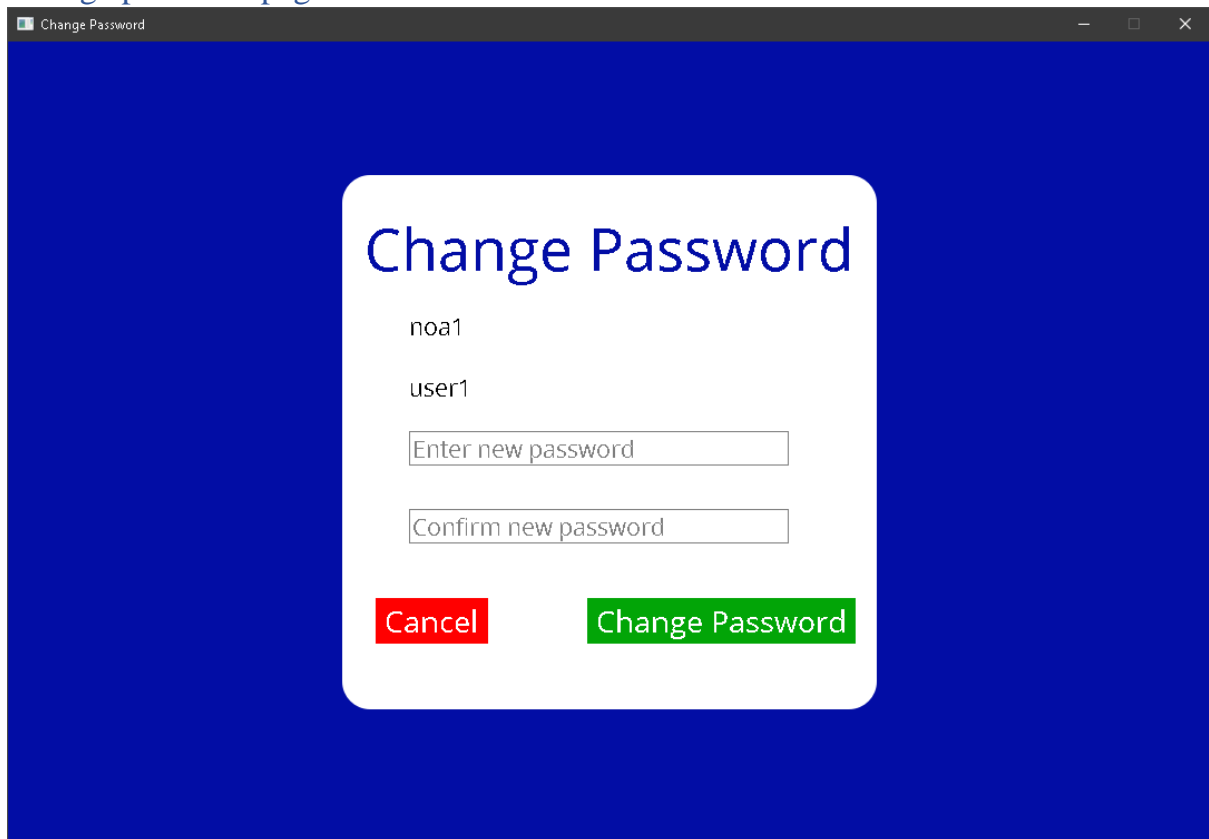


Figure 54

```
class changePassWin(QtWidgets.QWidget):  
    def __init__(self, *args, **kwargs):  
        super().__init__(*args, **kwargs)  
        self.ui = Ui_changePass()  
        self.ui.setupUi(self)  
        self.ui.nameOfAccountLbl.setText(VIEWEDITEM[0])  
        self.ui.usernameLbl.setText(VIEWEDITEM[1])  
        self.ui.cancelBtn.clicked.connect(self.goBack)  
        self.ui.changePassBtn.clicked.connect(self.changePassword)
```

Figure 55

The “changePassWin” class implements the user interface described in the changePassPage.py file. The global VIEWEDITEM variable is used to retrieve the name and username of the viewed account. Additionally, the “goBack” method in the “changePassWin” class is equivalent to the implementation used in the “addAccountWin” class, however, the implementation used in the “changePassWin” class creates the interface for the “viewAccountWin” class.

```

def changePassword(self):
    if (self.ui.passwordEdit.text() == (None or "")) or (self.ui.confirmPassEdit.text() == (None
or "")):
        Alert("Error", QtWidgets.QMessageBox.Critical,
            "One or Both of the password fields are empty")
    else:
        if self.ui.passwordEdit.text() != self.ui.confirmPassEdit.text():
            Alert("Error", QtWidgets.QMessageBox.Critical, "Passwords dont match")
        elif (" " in self.ui.passwordEdit.text()) or (" " in self.ui.confirmPassEdit.text()):
            Alert("Error", QtWidgets.QMessageBox.Critical, "Remove spaces from password fields")
        elif ("," in self.ui.passwordEdit.text()) or ("," in self.ui.confirmPassEdit.text()):
            Alert("Error", QtWidgets.QMessageBox.Critical, "Remove commas from password fields")
        else:
            key, iv, data = getData(KEYPATH, VAULTPATH)
            data = data.decode('utf-8')
            row = data.split('\n')
            accounts = []
            for value in row:
                if value != "":
                    # stores accounts as nested lists seperated by value
                    accounts.append(value.split(','))
            for i in range(len(accounts)):
                if accounts[i] == VIEWEDITEM:
                    VIEWEDITEM[2] = self.ui.passwordEdit.text() # updates the item being viewed
                    accounts[i] = VIEWEDITEM # updates the item in the accounts nested list
            updateAccounts(accounts) # calls updateAccounts
            Alert("Confirmed", QtWidgets.QMessageBox.Information, "Password Changed")
            self.goBack() # go to view account page after password is changed successfully

```

*Figure 56*

When the “changePassBtn” is clicked, the “changePassword” function is executed. First, the password entry fields are checked for erroneous data and to ensure that they match, if the test cases fail then an appropriate error message is displayed via the “Alert” function. If the test cases pass, all the accounts are retrieved from the vault file via the “getData” function. The account is then updated in the variable storing all of the accounts and the manipulated variable is passed to the “updateAccounts” function. Upon completion, a confirmation message is displayed via the “Alert” function and the “goBack” method is executed to display the viewed account with updated details.

## Import accounts page

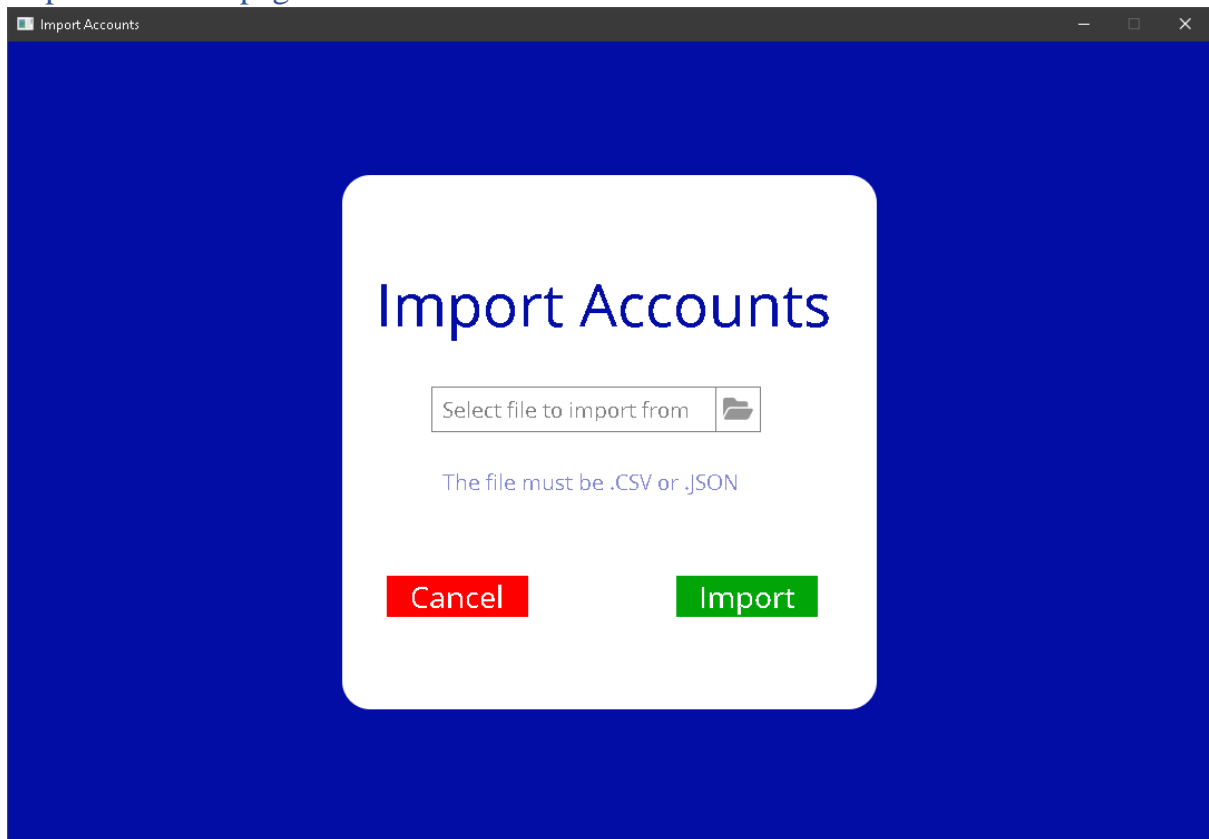


Figure 57

```
class importWin(QtWidgets.QWidget):  
    def __init__(self, *args, **kwargs):  
        super().__init__(*args, **kwargs)  
        self.ui = Ui_importAccounts()  
        self.ui.setupUi(self)  
        self.ui.cancelBtn.clicked.connect(self.goBack)  
        self.ui.selectFileBtn.clicked.connect(self.getFile)  
        self.ui.importBtn.clicked.connect(self.importData)
```

Figure 58

The “importWin” class generates the interface specified in the ImportAccountsPage.py and is executed when clicking the import button on the view all accounts page. The “goBack” method implemented in this class also creates an interface for the “allAccountsWin” class which is equivalent to the “goBack” function in the “addAccountWin” class.

```
def getFile(self):  
    file = QtWidgets.QFileDialog.getOpenFileName(  
        self, 'Open file', "", "All Files (*)") # lets user choose files from explorer  
    url = QtCore.QUrl.fromLocalFile(file[0])    # gets path to file and stores it as an object  
    self.ui.fileLbl.setText(url.fileName())     # adjusts file name in gui  
    self.ui.fileLbl.adjustSize()                # adjusts size of text wrapper for file name in gui  
    self.Path = file[0]                        # makes path accessible in importWin
```

*Figure 59*

Figure 59 in the “importWin” class is executed when an end-user clicks the “selectFileBtn” which is similar to the implementation of “getKeyFile” and “getVaultFile” in the “MainWindow” class.



```

def importData(self):

    if self.ui.fileLbl.text() == "Select file to import from":

        # checks that a Key File or Vault file have been selected

        Alert("Error", QtWidgets.QMessageBox.Critical,

              "No file was selected. Please select a file to import from")

        # Alert function to display error QMessageBox

    else:

        accounts = []

        if self.ui.fileLbl.text().lower().endswith(".csv"):

            with open(self.Path, 'r') as csvFile:

                reader = csv.DictReader(csvFile, delimiter=',')

                for row in reader:

                    if ('name' in row) and ('username' in row) and ('password' in row): #
lastpass format

                        if (row['username'] != "") and (row['password'] != "") and (row['name'] !=
""):

                            values = [row['name'], row['username'], row['password']]

                            accounts.append(values)

                        elif ('name' in row) and ('login_username' in row) and ('login_password' in
row): # bitwarden format

                            if (row['name'] != "") and (row['login_username'] != "") and
(row['login_password'] != ""):

                                values = [row['name'], row['login_username'], row['login_password']]

                                accounts.append(values)

            if len(accounts) < 1:

                Alert("Error", QtWidgets.QMessageBox.Critical,

                      "CSV file format not supported or no data to import was found")

            else:

                for item in accounts:

                    writeData(item[0], item[1], item[2])

                Alert("Confirmed", QtWidgets.QMessageBox.Information,

                      "Imported accounts from .CSV")

                self.goBack()

```

Figure 60

```

elif self.ui.fileLbl.text().lower().endswith(".json"):

    with open(self.Path) as jsonFile:

        data = json.load(jsonFile)

        if 'items' in data:

            for item in data['items']: # checks for bitwarden format

                if 'login' in item:

                    if ('username' in item['login']) and ('password' in item['login']):

                        if (item['login']['username'] is not None) and
(item['login']['password'] is not None):

                            values = [item['name'], item['login']

                                    ['username'], item['login']['password']]

                            accounts.append(values)

                else:

                    Alert("Error", QtWidgets.QMessageBox.Critical,

                        "JSON file format not supported")

            if len(accounts) < 1:

                Alert("Error", QtWidgets.QMessageBox.Critical,

                    "JSON file has no data to import")

            else:

                for item in accounts:

                    writeData(item[0], item[1], item[2])

                Alert("Confirmed", QtWidgets.QMessageBox.Information,

                    "Imported accounts from .JSON")

                self.goBack()

        else:

            Alert("Error", QtWidgets.QMessageBox.Critical, "File type not supported")

```

Figure 61

Figure 60 and Figure 61 refer to the “importData” function under the “importWin” class. In Figure 60, the label of the file is checked to ensure an end-user has selected a file as if a file isn’t selected an error message will be displayed via the “Alert” function. When a file is selected, if the file is in a “.csv” format, the file is read using the CSV module as dictated in (EDUCBA, 2020). If there are accounts in the file they are stored and passed to the “writeData” function, however, if the file has no accounts or the formatting of the file is incorrect, an error message is displayed via the “Alert” function. After the accounts have been imported a confirmation message is displayed via the “Alert” function and the “goBack” function is executed to return to the view all accounts page. The LastPass and Bitwarden CSV formats are the only CSV formats supported by the application, with the sample format for LastPass originating from (LogMeIn, 2020) and the format for Bitwarden was exported from an exported CSV file created by the service.

Figure 61 depicts the implementation for importing from a JSON file format with the format being extrapolated from an exported JSON file created by the service. This functions similarly to the implementation for reading from a CSV where accounts are imported from the file by using the JSON

python module as indicated by (Tejashwi5, 2019). To check that the JSON file was in the correct format, (Varun, 2018) was used on the dictionary variable storing the data from the JSON file, however, to check if there were any accounts in the accounts variable, (STechies, 2018) was used.

## Export accounts page

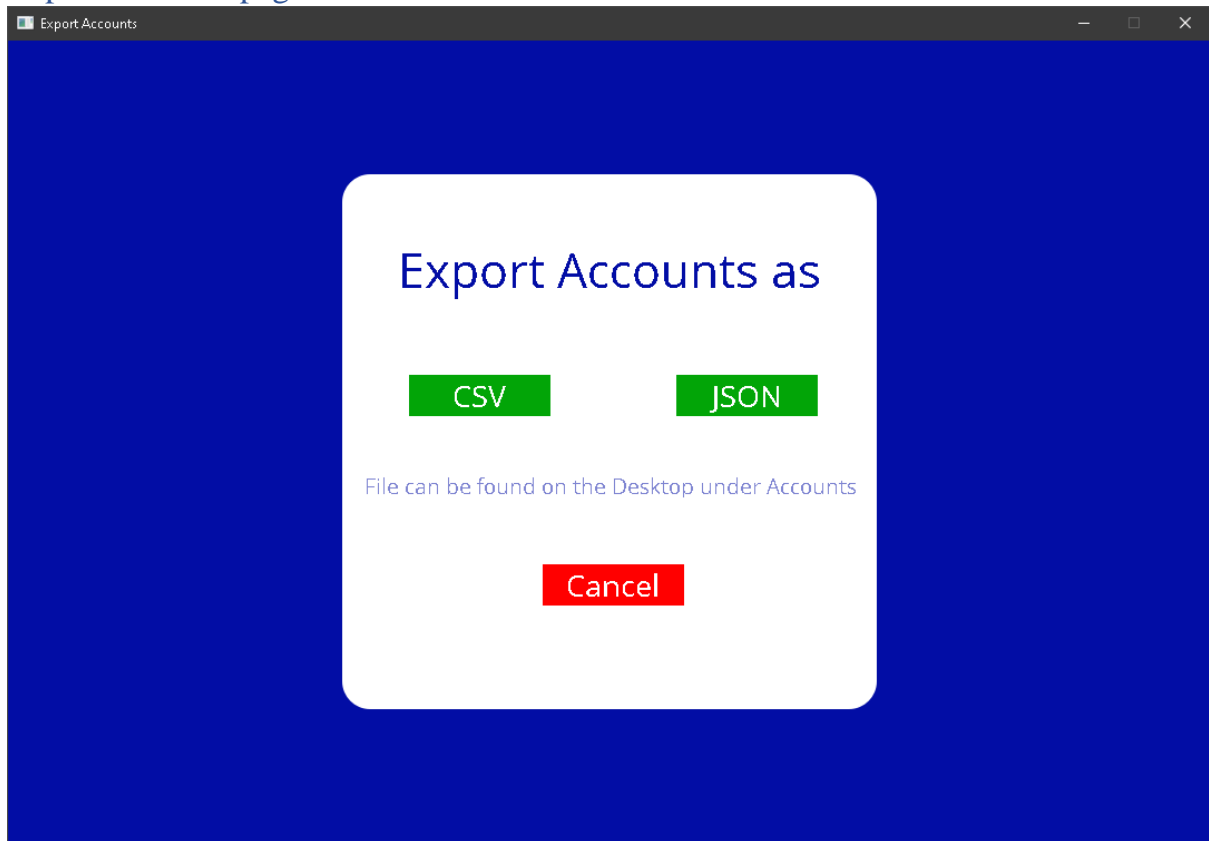


Figure 62

```
class exportWin(QtWidgets.QWidget):  
    def __init__(self, *args, **kwargs):  
        super().__init__(*args, **kwargs)  
        self.ui = Ui_exportPage()  
        self.ui.setupUi(self)  
        self.ui.cancelBtn.clicked.connect(self.goBack)  
        self.ui.csvBtn.clicked.connect(self.exportCSV)  
        self.ui.jsonBtn.clicked.connect(self.exportJSON)
```

Figure 63

The “exportWin” class creates the interface described in the exportPage.py file This class is executed when the export button on the window generated by the “allAccountsWin” class is clicked. “goBack” in this instance is implemented identically to the implementation in the “importWin” class.

```

def exportCSV(self):
    key, iv, data = getData(KEYPATH, VAULTPATH)
    data = data.decode('utf-8')
    path = getPathToDesktop()
    path += "\\Accounts.csv"
    if data != "":
        row = data.split('\n')
        accounts = []
        for value in row:
            if value != "":
                terms = value.split(',')
                temp = {}
                temp["name"], temp["username"], temp["password"] = terms[0], terms[1], terms[2]
                accounts.append(temp)
        with open(path, 'w') as file: # writes to csv file in lastpass format as lastpass'
format is widely supported
            columns = ['url', 'username', 'password', 'extra', 'name', 'grouping', 'fav']
            writer = csv.DictWriter(file, fieldnames=columns, lineterminator='\n')
            writer.writeheader()
            writer.writerows(accounts)
        Alert("Confirmed", QtWidgets.QMessageBox.Information, "CSV file successfully created")
    else:
        Alert("Error", QtWidgets.QMessageBox.Critical, "No accounts to export")
    self.goBack()

```

*Figure 64*

Figure 64 is executed when the “csvBtn” is clicked on the export page. Accounts are read from the vault file via the “getData” function, if there are accounts in the file then the accounts are written to an “Accounts.csv” file that will be written to the Desktop (the Desktop is retrieved via the “getPathToDesktop” function). (Lanaru, 2012) was used to write the accounts from the vault file to the CSV file while they were stored in a dictionary variable. An error was encountered when implementing this section as the CSV file would write all of the data on the same line, to remedy this (Krunal, 2019) was used to implement the new line terminator at the end of each account. The LastPass CSV format was used as a template for writing to the CSV file where all of the accounts are written to the CSV file after being stored in a dictionary variable. Confirmation messages are displayed via the “Alert” function after the process is complete, however, if there are no accounts in the vault file an error message is displayed instead.

```

def exportJSON(self):

    key, iv, data = getData(KEYPATH, VAULTPATH)

    data = data.decode('utf-8')

    path = getPathToDesktop()

    path += "\\Accounts.json"

    if data != "":

        row = data.split('\n')

        accounts = {}

        account = []

        for value in row:

            # json uses None for null and False for false when writing to a json

            if value != "":

                terms = value.split(',')

                loginValues = {}

                uris = [{"match": None, "uri": "http://"}]

                loginValues['uris'], loginValues['username'], loginValues['password'],
loginValues['totp'] = uris, terms[1], terms[2], None

                temp = {}

                temp['id'], temp['organizationId'], temp['folderId'], temp['type'], temp['name'],
temp['notes'], temp[

                    'favorite'], temp['login'], temp['collectionIds'] = "", None, None, 1,
terms[0], None, False, loginValues, False

                account.append(temp)

            accounts['items'] = account

            with open(path, 'w') as file:      # writes to csv in lastpass format

                json.dump(accounts, file, indent=4)

            Alert("Confirmed", QtWidgets.QMessageBox.Information, "JSON file successfully created")

        else:

            Alert("Error", QtWidgets.QMessageBox.Critical, "No accounts to export")

    self.goBack()

```

Figure 65

Figure 65 is executed when the “jsonBtn” is clicked. This function operates similarly to the “exportCSV” function, however, to write to the JSON file the Bitwarden format was used as a template to generate the headings for each field. Additionally, (AbhishekAgarwal4, 2020) was used to implement the functionality to write the accounts to the JSON file. Furthermore, when writing to the JSON file initially null values weren’t implemented correctly as when trying to write null values to the JSON document, they would be treated as string values encapsulated in quotation marks instead of null values, (user2555451, 2014) was used to rectify this as null values are equivalent to the “None” keyword in Python.

## Critical Evaluation

### Deciding what system to develop

Choosing a system to develop was quite difficult as many students chose to create projects because they wanted to solve a problem they had experienced or that they know of which I felt I couldn't relate to. Instead, I chose to develop a system so that I could learn about a topic of my interest, security and data collection. In Year 1 of University, I write a paper detailing how large companies utilise the data that individuals provide to which most individuals are unaware of how their data is used or what data is being collected. Writing that paper helped me understand that I have a passion for wanting to understand how data collection works and how individuals can go about securing their data. Developing the system helped me understand the security practices required in securing end-user data.

### Aims of the system

I believe that I have sufficiently implemented an encryption algorithm to encrypt the account details stored in the application while understanding some attack vectors used to break encryption methods such as brute force attacks. Furthermore, my skills for developing user interfaces have improved as the design of the application conforms to many major usability principles. For instance, the simplistic design of the application improves learnability as an end-user can familiarise themselves with the functions of the system quickly. Additionally, the visibility of elements on the interface allows for navigation and items of the interface to be easily identifiable with visual feedback being delivered to an end-user in the form of dialogue boxes which, although rather brief in the information conveyed, do not appear intrusive when using the application. However, the use of bright colours in the application may contribute to eye-strain when used in un-ideal lighting conditions or may reduce usability for colour-blind individuals. Concerning the digital footprint of the application, the entire system excluding the files produced by the application such as the vault and key files occupies 820 kilobytes on the disk which is significantly smaller than the compressed size of KeePass which occupies 5.4 megabytes (Slashdot Media, 2020).

### Further development

- Implement multithreading or multiprocessing
- Implement a theming option
- Implement random password generation on the Add Account and Change Password page

Implementing multithreading or multiprocessing would allow functions in the application to execute faster which would reduce the number of system resources being utilised when IO-bound operations occur. Furthermore, implementing random password generation as a button on the Add Account and Change Password page would allow end-users to choose between using a password the system generates or to use a password of their choice. Although beyond the necessary functional requirements of the application, implementing a "theme" feature would improve customizability and would allow an end-user to choose the colours of the interface to suit their preference which could benefit colour-blind users and could reduce the risk of developing eye-strain while using the application.

## Developing the system

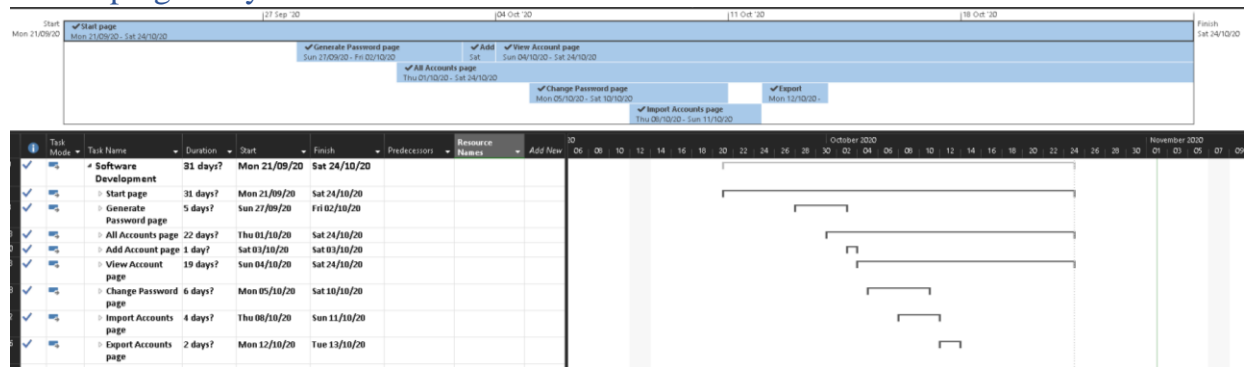


Figure 66

Figure 66 is a Gantt chart which was used to record the time spent during development on each task which is treated as subtasks under each interface. Although the main development process used was incremental development, I found finishing the development of each page somewhat difficult as some algorithms needed to be defined before proceeding with the implementation of the program would generate errors which would hinder development. If an algorithm needed to be developed or an error occurred, I would attempt to resolve the error or create the algorithm. In most situations, this would comply with incremental development, however, if the implementation of the algorithm wasn't clear or the error couldn't be rectified during that increment, I tried moving onto another task and then coming back to the problem to progress the development of the system which caused the time allocated to some tasks to overlap. The time allocated for the start page, all accounts page and view account page overlap the most due to code refactoring after all the interfaces had been implemented.

## Personal reflection

During the development of the project, I have learnt that systems which accommodate change are less costly to maintain. To account for future changes such as modules becoming deprecated for instance, Pycryptodome was used in place of the Cryptography module but the "import" parameter for the module remained the same thus uninstalling the deprecated package and installing Pycryptodome wouldn't affect systems using the older module unless the functionality of the module had changed. Furthermore, development of a system should be perceived from an end-user perspective as a developer which aided in building the system expects the system to operate in a specific way whereas, an end-user must learn to use a system as they don't expect how a feature implemented in the system should function. Similarly, thorough testing should be conducted before deployment of a system to accommodate any errors the system must handle while an end-user utilises the application such as exception handling.

## Appraisal

Although the development of the application in Python was quite fast, if C++ were used, I could have implemented the same user interface using the Qt module while providing optimised compilation and execution times. Instead of using an encrypted binary file to store all of a user's accounts, if PBKDF2 was used in cohesion with the AES encryption applied to the encrypted binary file then the overall security of the accounts file would improve. Overall, the application meets the requirements described in the project, however, if an end-user favours accessibility from multiple devices then the system isn't viable to which Bitwarden or another cloud service solution would be more suitable. Furthermore, if the end-user requires that their password manager stores notes, credit card details or to generate and store SSH keys, a system like KeePass or KeePassXC would be favourable as the developed system doesn't offer support for these formats. Nonetheless, the system can be perceived as a simplistic alternative to KeePass or KeePassXC given that the end-user requires the system to only store account details.

## Bibliography

- AbhishekAgarwal4, 2020. *How To Convert Python Dictionary To JSON?*. [Online]  
Available at: <https://www.geeksforgeeks.org/how-to-convert-python-dictionary-to-json/>  
[Accessed 13 October 2020].
- Adobe Inc., 2020. *Adobe XD*, San Jose: Adobe Inc..
- Alexander, I. & Charbison, 2010. *Onion Diagram*. [Online]  
Available at: [https://vle.lsbu.ac.uk/pluginfile.php/1738330/mod\\_resource/content/1/onion.ppt](https://vle.lsbu.ac.uk/pluginfile.php/1738330/mod_resource/content/1/onion.ppt)  
[Accessed 22 March 2020].
- alexisdm, 2013. *How to delete all rows from QTableWidgetItem*. [Online]  
Available at: <https://stackoverflow.com/questions/15848086/how-to-delete-all-rows-from-qtablewidget>  
[Accessed 3 October 2020].
- amigos-maker, 2020. *Python GUI, PyQt vs Tkinter*. [Online]  
Available at: <https://dev.to/amigosmaker/python-gui-pyqt-vs-tkinter-5hdd>  
[Accessed 18 October 2020].
- Balsamiq Studios, LLC, 2008. *Balsamiq Wireframes*, Sacramento: Balsamiq Studios, LLC.
- Bhatti, K., 2020. *Offline Password Vault*. [Online]  
Available at: <https://youtu.be/mEqxIAC4rQM>  
[Accessed 27 October 2020].
- Bhatti, K., 2020. *Poster*. [Online]  
Available at: [https://drive.google.com/file/d/1IIEGvZhIhkk7QE8quRs-elgmeRBGs8\\_h/view?usp=sharing](https://drive.google.com/file/d/1IIEGvZhIhkk7QE8quRs-elgmeRBGs8_h/view?usp=sharing)  
[Accessed 27 October 2020].
- Bhatti, K., 2020. *The Vault*. [Online]  
Available at: <https://github.com/k5924/TheVault>  
[Accessed 23 October 2020].
- Bitwarden, Inc., 2020. *Plans and Pricing*. [Online]  
Available at: <https://bitwarden.com/pricing/>  
[Accessed 11 September 2020].
- Child, M., 2020. *04 - Abstract classes, interfaces and UML*. [Online]  
Available at: <https://vle.lsbu.ac.uk/mod/resource/view.php?id=1454658>  
[Accessed 14 October 2020].
- Coventry, A., 2017. *Finding the index of an item in a list*. [Online]  
Available at: <https://stackoverflow.com/questions/176918/finding-the-index-of-an-item-in-a-list>  
[Accessed 8 October 2020].
- Dashlane Inc., 2020. *Choose the plan for you*. [Online]  
Available at: <https://www.dashlane.com/plans>  
[Accessed 11 September 2020].
- Dunn, N., 2020. *How to Check the Operating System with Python*. [Online]  
Available at: <https://www.webucator.com/how-to/how-check-the-operating-system-with->



python.cfm

[Accessed 23 October 2020].

EDUCBA, 2020. *Python Read CSV File*. [Online]

Available at: <https://www.educba.com/python-read-csv-file/>

[Accessed 11 October 2020].

ekhumoro, 2017. *Get Plain text from a QLabel with Rich text*. [Online]

Available at: <https://stackoverflow.com/questions/8890320/get-plain-text-from-a-qlabel-with-rich-text>

[Accessed 13 September 2020].

ekhumoro, 2017. *How to align the text to center of cells in a QTableWidgetItem*. [Online]

Available at: <https://stackoverflow.com/questions/47971275/how-to-align-the-text-to-center-of-cells-in-a-qtablewidget>

[Accessed 3 October 2020].

eyllanesc, 2018. *PyQt5 - Get file name from an opened file, not a file path*. [Online]

Available at: <https://stackoverflow.com/questions/49217347/pyqt5-get-file-name-from-an-opened-file-not-a-file-path>

[Accessed 13 September 2020].

<https://pythonbasics.org>, 2020. *How to use Tables in PyQt*. [Online]

Available at: <https://pythonbasics.org/pyqt-table/>

[Accessed 2 October 2020].

Information Commissioners Office, 2020. *Principle (f): Integrity and confidentiality (security)*. [Online]

Available at: <https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/principles/integrity-and-confidentiality-security/>

[Accessed 16 October 2020].

Kamaruzzaman, M., 2020. *Top 10 In-Demand programming languages to learn in 2020*. [Online]

Available at: <https://towardsdatascience.com/top-10-in-demand-programming-languages-to-learn-in-2020-4462eb7d8d3e>

[Accessed 18 October 2020].

kame, 2016. *How to convert 'binary string' to normal string in Python3?*. [Online]

Available at: <https://stackoverflow.com/questions/17615414/how-to-convert-binary-string-to-normal-string-in-python3>

[Accessed 29 September 2020].

KeePassXC Team, 2016. *The Project*. [Online]

Available at: <https://keepassxc.org/project/>

[Accessed 11 September 2020].

Krunal, 2019. *How To Write Python Dictionary To CSV File Example*. [Online]

Available at: <https://appdividend.com/2019/11/14/how-to-write-python-dictionary-to-csv-example/>

[Accessed 13 October 2020].

Lanaru, 2012. *csv.write skipping lines when writing to csv*. [Online]

Available at: <https://stackoverflow.com/questions/11652806/csv-write-skipping-lines-when-writing-to-csv>

[Accessed 13 October 2020].

Lemac, M., 2016. *Stakeholder Identification - Template*. [Online]  
Available at:  
[https://vle.lsbu.ac.uk/pluginfile.php/1738329/mod\\_resource/content/2/Stakeholders%20Identification.docx](https://vle.lsbu.ac.uk/pluginfile.php/1738329/mod_resource/content/2/Stakeholders%20Identification.docx)  
[Accessed 22 March 2020].

Lemac, M., 2016. *Use Case Descriptions - Template*. [Online]  
Available at:  
[https://vle.lsbu.ac.uk/pluginfile.php/1725332/mod\\_resource/content/2/UseCaseDescriptions-template.docx](https://vle.lsbu.ac.uk/pluginfile.php/1725332/mod_resource/content/2/UseCaseDescriptions-template.docx)  
[Accessed 22 March 2020].

Lemac, M. & Phillips, N., 2011. *Stakeholders*. [Online]  
Available at:  
[https://vle.lsbu.ac.uk/pluginfile.php/1738328/mod\\_resource/content/1/Stakeholders.pptx](https://vle.lsbu.ac.uk/pluginfile.php/1738328/mod_resource/content/1/Stakeholders.pptx)  
[Accessed 17 February 2020].

Lemac, M. & Phillips, N., 2011. *Week 5: Activity Diagrams*. [Online]  
Available at:  
[https://vle.lsbu.ac.uk/pluginfile.php/1725334/mod\\_resource/content/3/ActivityDiagrams.pptx](https://vle.lsbu.ac.uk/pluginfile.php/1725334/mod_resource/content/3/ActivityDiagrams.pptx)  
[Accessed 22 March 2020].

LogMeIn, Inc., 2020. *PLans & Pricing*. [Online]  
Available at: <https://www.lastpass.com/pricing>  
[Accessed 11 September 2020].

LogMeIn, 2020. *How do I import stored data into LastPass using a generic CSV file?*. [Online]  
Available at: <https://support.logmeininc.com/lastpass/help/how-do-i-import-stored-data-into-lastpass-using-a-generic-csv-file>  
[Accessed 11 October 2020].

Luengo, I., 2015. *How to make push button immediately disabled?*. [Online]  
Available at: <https://stackoverflow.com/questions/33954886/how-to-make-push-button-immediately-disabled>  
[Accessed 29 September 2020].

mechanical\_meat, 2011. *Catch multiple exceptions in one line (except block)*. [Online]  
Available at: <https://stackoverflow.com/questions/6470428/catch-multiple-exceptions-in-one-line-except-block>  
[Accessed 13 September 2020].

Phillips, N. & Lemac, M., 2011. *Week 4: Use Case Descriptions*. [Online]  
Available at:  
[https://vle.lsbu.ac.uk/pluginfile.php/1725329/mod\\_resource/content/5/UseCaseDescriptions.pptx](https://vle.lsbu.ac.uk/pluginfile.php/1725329/mod_resource/content/5/UseCaseDescriptions.pptx)  
[Accessed 17 February 2020].

rakshitarora, 2020. *PyQt5 – How to auto resize Label / adjustSize QLabel*. [Online]  
Available at: <https://www.geeksforgeeks.org/pyqt5-how-to-auto-resize-label-adjustsize-qlabel/>  
[Accessed 13 September 2020].

rakshitarora, 2020. *PyQt5 – isChecked() method for Check Box*. [Online]  
Available at: <https://www.geeksforgeeks.org/pyqt5-ischecked-method-for-check-box/>  
[Accessed 28 September 2020].

Reichi, D., 2003. *Keepass features*. [Online]  
Available at: <https://keepass.info/features.html>  
[Accessed 11 September 2020].

Satzinger, J. & Lemac, M., 2020. *Week 7: Domain Class Diagram*. [Online]  
Available at:  
[https://vle.lsbu.ac.uk/pluginfile.php/1725343/mod\\_resource/content/8/ClassDiagram.ppt](https://vle.lsbu.ac.uk/pluginfile.php/1725343/mod_resource/content/8/ClassDiagram.ppt)  
[Accessed 22 March 2020].

Slant, 2020. *Atom vs Visual Studio Code*. [Online]  
Available at: [https://www.slant.co/versus/48/5982/~atom\\_vs\\_visual-studio-code](https://www.slant.co/versus/48/5982/~atom_vs_visual-studio-code)  
[Accessed 18 October 2020].

Slashdot Media, 2020. *KeePass*. [Online]  
Available at: <https://sourceforge.net/projects/keepass/files/KeePass%202.x/2.46/KeePass-2.46.zip/download>  
[Accessed 31 October 2020].

Sommerville, I., 2016. Software Engineering. In: 10th, ed. *Software Engineering*. Harlow: Pearson, pp. 45-54.

STechies, 2018. *Check if a List, Array, Set, Tuple, String or Dictionary is Empty in Python?*. [Online]  
Available at: <https://www.stechies.com/check-list-array-set-tuple-string-dictionary-empty-python/>  
[Accessed 11 October 2020].

Striver, 2018. *set copy() in python*. [Online]  
Available at: <https://www.geeksforgeeks.org/set-copy-python/>  
[Accessed 3 October 2020].

Tejashwi5, 2019. *Read JSON file using Python*. [Online]  
Available at: <https://www.geeksforgeeks.org/read-json-file-using-python/>  
[Accessed 11 October 2020].

The Qt Company, 2020. *QClipboard*. [Online]  
Available at: <https://doc.qt.io/qtforpython/PySide2/QtGui/QClipboard.html>  
[Accessed 5 October 2020].

The Qt Company, 2020. *QTableWidget*. [Online]  
Available at:  
<https://doc.qt.io/qtforpython/PySide2/QtWidgets/QTableWidget.html#PySide2.QtWidgets.PySide2.QtWidgets.QTableWidget.currentItem>  
[Accessed 4 October 2020].

thefourtheye, 2015. *Using Python to find drive letter (Windows)*. [Online]  
Available at: <https://stackoverflow.com/questions/29026051/using-python-to-find-drive-letter-windows>  
[Accessed 13 September 2020].

Trend Micro Incorporated, 2018. *Data Breaches 101: How They Happen, What Gets Stolen, and Where It All Goes*. [Online]  
Available at: <https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/data-breach-101>  
[Accessed 16 October 2020].

user1006989, 2013. *Detecting enter on a QLineEdit or QPushButton*. [Online]  
Available at: <https://stackoverflow.com/questions/15561608/detecting-enter-on-a-qlineedit-or-qpushbutton>  
[Accessed 3 October 2020].

user2555451, 2014. *How can JSON data with null value be converted to a dictionary*. [Online]  
Available at: <https://stackoverflow.com/questions/27140090/how-can-json-data-with-null-value-be-converted-to-a-dictionary>  
[Accessed 13 October 2020].

user2555451, 2015. *Special Characters in string literals*. [Online]  
Available at: <https://stackoverflow.com/questions/28056843/special-characters-in-string-literals>  
[Accessed 27 September 2020].

user559633, 2015. *How to get Desktop location?*. [Online]  
Available at: <https://stackoverflow.com/questions/34275782/how-to-get-desktop-location>  
[Accessed 13 September 2020].

Varun, 2018. *Python: check if key exists in dictionary (6 Ways)*. [Online]  
Available at: <https://thispointer.com/python-how-to-check-if-a-key-exists-in-dictionary/>  
[Accessed 11 October 2020].

Vollebregt, B., 2019. *Python Encryption and Decryption with PyCryptodome*. [Online]  
Available at: <https://nitratine.net/blog/post/python-encryption-and-decryption-with-pycryptodome/>  
[Accessed 13 September 2020].

Wagenseil, P., 2020. *LastPass, 1Password and other password managers can be hacked: What to do now*. [Online]  
Available at: <https://www.tomsguide.com/news/password-manager-hacks>  
[Accessed 16 October 2020].

## Appendix A – main.py (Bhatti, 2020)

```
import sys

import os

import random

import csv

import json

from platform import system

from string import ascii_uppercase, ascii_lowercase, digits, punctuation

from startPage import Ui_startPage

from genPassPage import Ui_passwordGen

from allAccountsPage import Ui_allAccounts

from AddAccountPage import Ui_addAccount

from viewAccountPage import Ui_viewAccount

from changePassPage import Ui_changePass

from importAccountsPage import Ui_importAccounts

from exportAccountsPage import Ui_exportPage

from PyQt5 import QtWidgets, QtCore, QtGui

from Crypto.Random import get_random_bytes

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad


# global variables to store paths to the vault and key file
global KEYPATH, VAULTPATH, VIEWEDITEM


class MainWindow(QtWidgets.QWidget):

    def __init__(self, *args, **kwargs):

        super().__init__(*args, **kwargs)

        self.ui = Ui_startPage()

        self.ui.setupUi(self) # initializes start page

        self.ui.startButton.clicked.connect(self.createVaultFiles)

        self.ui.selectKeyFile.clicked.connect(self.getKeyFile)
```

```

self.ui.selectVaultFile.clicked.connect(self.getVaultFile)

self.ui.openButton.clicked.connect(self.openVaultFiles)

# button variables which execute a specific function

def createVaultFiles(self):
    key = get_random_bytes(32) # 32 bytes is 256 bits
    data = ".encode('utf-8') # basic data for file to encrypt
    desktopPath = getPathToDesktop() # gets path to desktop
    keyFile = open(desktopPath + "\\key.bin", "wb")
    keyFile.write(key) # writes encryption key to file
    keyFile.close
    cipher = AES.new(key, AES.MODE_CBC)
    ciphered_data = cipher.encrypt(pad(data, AES.block_size))
    vaultFile = open(desktopPath + "\\vault.bin", "wb") # creates vault file
    vaultFile.write(cipher.iv)
    vaultFile.write(ciphered_data)
    vaultFile.close()

    Alert("Process Completed", QtWidgets.QMessageBox.Information, "Created vault.bin and
key.bin")

    # Alert function to reuse the code to generate a QMessageBox

def getKeyFile(self):
    file = QtWidgets.QFileDialog.getOpenFileName(
        self, 'Open file', "", "All Files (*)") # lets user choose files from explorer
    url = QtCore.QUrl.fromLocalFile(file[0]) # gets path to file and stores it as an object
    self.ui.keyFileLabel.setText(url.fileName()) # adjusts file name in gui
    self.ui.keyFileLabel.adjustSize() # adjusts size of text wrapper for file name in gui
    self.keyPath = file[0] # makes keyPath accessible in all of MainWindow class

def getVaultFile(self):
    file = QtWidgets.QFileDialog.getOpenFileName(
        self, 'Open file', "", "All Files (*)") # lets user choose files from explorer
    url = QtCore.QUrl.fromLocalFile(file[0]) # gets path to file and stores it as an object

```

```

self.ui.vaultFileLabel.setText(url.fileName()) # adjusts file name in gui
self.ui.vaultFileLabel.adjustSize() # adjusts size of text wrapper for file name in gui
self.vaultPath = file[0] # makes vaultPath accessible in all of MainWindow class

def openVaultFiles(self):
    keyFile = self.ui.keyFileLabel.text()
    vaultFile = self.ui.vaultFileLabel.text()
    if (keyFile == "Key File") or (vaultFile == "Vault File"):
        # checks that a Key File or Vault file have been selected
        Alert("Error", QtWidgets.QMessageBox.Critical,
            "Either one or no files were selected. Please select files to open the vault")
        # Alert function to display error QMessageBox
    else:
        # exception handling
        try:
            key, iv, data = getData(self.keyPath, self.vaultPath)
            # display new window for generating password or viewing accounts
            self.newWindow = generatePasswordWin()
            self.newWindow.show() # show new window
            self.hide() # close old window
        except (ValueError, FileNotFoundError) as e:
            Alert("Error", QtWidgets.QMessageBox.Critical, "Incorrect files selected")
            # Alert function to show error message

class generatePasswordWin(QtWidgets.QWidget):
    # displays generate password window when vault is open
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.ui = Ui_passwordGen()
        self.ui.setupUi(self)
        self.ui.genBtn.clicked.connect(self.genPassword)

```

```

self.ui.saveBtn.clicked.connect(self.savePassword)

self.ui.viewAccountsTab.clicked.connect(self.openAccountsPage)

def genPassword(self):
    passwordOptions = ""

    if self.ui.lowerCaseCheck.isChecked() or self.ui.upperCaseCheck.isChecked() or
self.ui.numbersCheck.isChecked() or self.ui.specialCharsCheck.isChecked():

        if self.ui.lowerCaseCheck.isChecked():
            passwordOptions += ascii_lowercase

        if self.ui.upperCaseCheck.isChecked():
            passwordOptions += ascii_uppercase

        if self.ui.numbersCheck.isChecked():
            passwordOptions += digits

        if self.ui.specialCharsCheck.isChecked():
            passwordOptions += punctuation.replace(',', '')

        lengths = [i for i in range(8, 17)]
        passLength = random.choice(lengths)
        password = ""

        for i in range(0, passLength):
            password += random.choice(passwordOptions)

        self.ui.generatedPassLabel.setText(password)
        self.ui.nameOfAccountEdit.setEnabled(True)
        self.ui.usernameEdit.setEnabled(True)
        self.ui.saveBtn.setEnabled(True)
    else:
        Alert("Error", QtWidgets.QMessageBox.Critical, "No options to generate password from")

def savePassword(self):
    if (self.ui.nameOfAccountEdit.text() == (None or "")) or (self.ui.usernameEdit.text() == (None
or "")):
        Alert("Error", QtWidgets.QMessageBox.Critical,
            "Account name or Username has been left empty")
    else: # displays any error message if the user input fields are empty or incorrectly entered

```



```

    if (self.ui.nameOfAccountEdit.text()[0] == " ") or (self.ui.nameOfAccountEdit.text()[-1] == "
"):
        Alert("Error", QtWidgets.QMessageBox.Critical,
              "Please remove spaces from the beginning or end of Account name")
    elif " " in self.ui.usernameEdit.text():
        Alert("Error", QtWidgets.QMessageBox.Critical,
              "Please remove spaces from Username")
    elif ("," in self.ui.nameOfAccountEdit.text()) or ("," in self.ui.usernameEdit.text()):
        Alert("Error", QtWidgets.QMessageBox.Critical,
              "Please remove commas from name of account or username")
    else:
        nameOfAccount = self.ui.nameOfAccountEdit.text()
        username = self.ui.usernameEdit.text()
        password = self.ui.generatedPassLabel.text()
        writeData(nameOfAccount, username, password)
        Alert("Process Completed", QtWidgets.QMessageBox.Information, "Account saved")
        # reset check boxes after saving accounts
        self.ui.lowerCaseCheck.setChecked(False)
        self.ui.upperCaseCheck.setChecked(False)
        self.ui.numbersCheck.setChecked(False)
        self.ui.specialCharsCheck.setChecked(False)
        # the code below resets that generatedPassLabel, nameOfAccount input and username input
        after saving
        self.ui.generatedPassLabel.setText("")
        self.ui.nameOfAccountEdit.setText("")
        self.ui.usernameEdit.setText("")
        self.ui.nameOfAccountEdit.setEnabled(False)
        self.ui.usernameEdit.setEnabled(False)

def openAccountsPage(self): # opens window to view all accounts
    self.newWindow = allAccountsWin()
    self.newWindow.show() # show new window
    self.hide() # close old window

```

```

class allAccountsWin(QtWidgets.QWidget): # view all accounts window

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.ui = Ui_allAccounts()
        self.ui.setupUi(self)

        # button which links to generate password window
        self.ui.genPassTab.clicked.connect(lambda: self.changeWindow(generatePasswordWin()))
        self.loadAccounts()
        self.ui.accountsTable.itemClicked.connect(self.viewItem)
        self.ui.addAccountBtn.clicked.connect(lambda: self.changeWindow(addAccountWin()))
        self.ui.searchBox.returnPressed.connect(self.searchAccounts)
        self.ui.importBtn.clicked.connect(lambda: self.changeWindow(importWin()))
        self.ui.exportBtn.clicked.connect(lambda: self.changeWindow(exportWin()))

    def changeWindow(self, classToAccess): # takes new window argument
        self.newWindow = classToAccess
        self.newWindow.show() # show new window
        self.hide() # close old window

    def loadAccounts(self): # added feature to read accounts from file
        global KEYPATH, VAULTPATH
        self.searchedAccounts = {}
        self.ui.accountsTable.setEditTriggers(QtWidgets.QTableWidget.NoEditTriggers)
        key, iv, data = getData(KEYPATH, VAULTPATH)
        data = data.decode('utf-8')
        self.count = 1 # count for resetting all accounts view
        if data != "":
            row = data.split("\n")
            self.accounts = {}
            i = 0

```

```

for value in row:
    if value != "":
        self.accounts[i] = value.split(',')
        i += 1

self.ui.accountsTable.setRowCount(0) # removes all data in table before making table
for n, key in enumerate(sorted(self.accounts.keys())): # displays code in table in window
    self.ui.accountsTable.insertRow(n)
    newitem = QtWidgets.QTableWidgetItem(self.accounts[key][0])
    viewLabel = QtWidgets.QTableWidgetItem("View")
    viewLabel.setTextAlignment(QtCore.Qt.AlignCenter)
    self.ui.accountsTable.setItem(n, 0, newitem)
    self.ui.accountsTable.setItem(n, 1, viewLabel)
    viewLabel.setBackground(QtGui.QColor(210, 210, 210))
    viewLabel.setFlags(viewLabel.flags() ^ QtCore.Qt.ItemIsEditable)
else: # else disables table
    self.ui.accountsTable.setEnabled(False)
    self.ui.searchBox.setEnabled(False)

def viewItem(self):
    global VIEWEDITEM
    if (self.ui.accountsTable.currentItem().text() == "View") and
    (self.ui.accountsTable.currentColumn() == 1):
        row = self.ui.accountsTable.currentRow()
        if not(self.searchedAccounts): # checks if searchedAccounts is empty
            VIEWEDITEM = self.accounts[row]
        else:
            for n, key in enumerate(sorted(self.searchedAccounts.keys())):
                if row == n:
                    VIEWEDITEM = self.accounts[key]
            self.changeWindow(viewAccountWin())

def searchAccounts(self):
    term = self.ui.searchBox.text()

```

```

if term != (None or ""):
    self.searchedAccounts = self.accounts.copy() # copy sets values to new variable to edit
    self.count -= 1 # decreases count for table to reset when nothing in searchBox
    self.ui.accountsTable.setRowCount(0) # deletes tables contents
    for n, key in enumerate(sorted(self.accounts.keys())): # displays code in table in window
        if not(term.lower() in self.accounts[key][0].lower()):
            self.searchedAccounts.pop(key) # removes values not in search
    # code below works just like in loadAccounts but with search terms
    for n, key in enumerate(sorted(self.searchedAccounts.keys())):
        self.ui.accountsTable.insertRow(n)
        newitem = QtWidgets.QTableWidgetItem(self.searchedAccounts[key][0])
        viewLabel = QtWidgets.QTableWidgetItem("View")
        viewLabel.setTextAlignment(QtCore.Qt.AlignCenter)
        self.ui.accountsTable.setItem(n, 0, newitem)
        self.ui.accountsTable.setItem(n, 1, viewLabel)
        viewLabel.setBackground(QtGui.QColor(210, 210, 210))
        viewLabel.setFlags(viewLabel.flags() ^ QtCore.Qt.ItemIsEditable)
    else: # if search box is empty
        if self.count <= 0: # comparison to make sure you only run loadAccounts after a search
            self.searchedAccounts = {}
            self.loadAccounts()

```

```

class addAccountWin(QtWidgets.QWidget):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.ui = Ui_addAccount()
        self.ui.setupUi(self)
        self.ui.cancelBtn.clicked.connect(self.goBack)
        self.ui.saveBtn.clicked.connect(self.saveAccount)

    def goBack(self):

```

```

self.newWindow = allAccountsWin()

self.newWindow.show()

self.hide()

def saveAccount(self):
    if (self.ui.nameOfAccountEdit.text() == (None or "")) or (self.ui.usernameEdit.text() == (None
or "")) or (self.ui.passwordEdit.text() == (None or "")):
        Alert("Error", QtWidgets.QMessageBox.Critical,
            "Account name, Username or the Password field has been left empty")
    else: # displays any error message if the user input fields are empty or incorrectly entered
        if (self.ui.nameOfAccountEdit.text()[0] == " ") or (self.ui.nameOfAccountEdit.text()[-1] == "
"):
            Alert("Error", QtWidgets.QMessageBox.Critical,
                "Please remove spaces from the beginning or end of Account name")
        elif (" " in self.ui.usernameEdit.text()) or (" " in self.ui.passwordEdit.text()):
            Alert("Error", QtWidgets.QMessageBox.Critical,
                "Please remove spaces from Username or Password")
        elif ("," in self.ui.nameOfAccountEdit.text()) or ("," in self.ui.usernameEdit.text()) or ("," in
self.ui.passwordEdit.text()):
            Alert("Error", QtWidgets.QMessageBox.Critical,
                "Please remove commas from Name of account, Username or Password")
    else:
        nameOfAccount = self.ui.nameOfAccountEdit.text()
        username = self.ui.usernameEdit.text()
        password = self.ui.passwordEdit.text()
        writeData(nameOfAccount, username, password)
        Alert("Process Completed", QtWidgets.QMessageBox.Information, "Account saved")
        self.goBack()

class viewAccountWin(QtWidgets.QWidget):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

```

```

self.ui = Ui_viewAccount()
self.ui.setupUi(self)
self.ui.backBtn.clicked.connect(lambda: self.changeWindow(allAccountsWin()))
self.ui.nameOfAccountLbl.setText(VIEWEDITEM[0])
self.ui.nameOfAccountLbl.adjustSize()
self.ui.usernameLbl.setText(VIEWEDITEM[1])
self.ui.usernameLbl.adjustSize()
self.ui.passwordLbl.setText(VIEWEDITEM[2])
self.ui.passwordLbl.adjustSize()
self.ui.copyUserBtn.clicked.connect(self.copyUsername)
self.ui.copyPassBtn.clicked.connect(self.copyPassword)
self.ui.changePassBtn.clicked.connect(lambda: self.changeWindow(changePassWin()))
self.ui.deleteBtn.clicked.connect(self.deleteAccount)

def changeWindow(self, classToAccess):
    self.newWindow = classToAccess
    self.newWindow.show()
    self.hide()

def copyUsername(self):
    cb = QtGui.QGuiApplication.clipboard()
    cb.setText(self.ui.usernameLbl.text(), mode=cb.Clipboard)
    Alert("Confirmed", QtWidgets.QMessageBox.Information,
          "Username copied to clipboard")

def copyPassword(self):
    cb = QtGui.QGuiApplication.clipboard()
    cb.setText(self.ui.passwordLbl.text(), mode=cb.Clipboard)
    Alert("Confirmed", QtWidgets.QMessageBox.Information,
          "Password copied to clipboard")

def deleteAccount(self):

```

```

message = QtWidgets.QMessageBox()
message.setWindowTitle("Warning")
message.setIcon(QtWidgets.QMessageBox.Warning)
message.setText("Are you sure you want to delete the account?")
message.setStandardButtons(QtWidgets.QMessageBox.Yes | QtWidgets.QMessageBox.Cancel)
message.setDefaultButton(QtWidgets.QMessageBox.Cancel)
message.buttonClicked.connect(self.confirmDelete)
message.exec_()

```

```

def confirmDelete(self, clickedBtn):

```

```

    if clickedBtn.text() == "&Yes":

```

```

        key, iv, data = getData(KEYPATH, VAULTPATH)

```

```

        data = data.decode('utf-8')

```

```

        row = data.split('\n')

```

```

        accounts = []

```

```

        for value in row:

```

```

            if value != "":

```

```

                # stores accounts as nested lists seperated by value

```

```

                accounts.append(value.split(','))

```

```

        for account in accounts:

```

```

            if account == VIEWEDITEM:

```

```

                index = accounts.index(account)

```

```

                accounts.pop(index)

```

```

                # when this code was a for loop in range len(accounts) sometimes it would give

```

```

                # a random error when lots of accounts were added and then someone attempts to delete
an account

```

```

                # although the code is now longer, this fixes the index error issue

```

```

                updateAccounts(accounts) # calls updateAccounts

```

```

                self.changeWindow(allAccountsWin())

```

```

class changePassWin(QtWidgets.QWidget):

```

```

    def __init__(self, *args, **kwargs):

```

```

super().__init__(*args, **kwargs)

self.ui = Ui_changePass()

self.ui.setupUi(self)

self.ui.nameOfAccountLbl.setText(VIEWEDITEM[0])

self.ui.usernameLbl.setText(VIEWEDITEM[1])

self.ui.cancelBtn.clicked.connect(self.goBack)

self.ui.changePassBtn.clicked.connect(self.changePassword)


def goBack(self):

    self.newWindow = viewAccountWin()

    self.newWindow.show()

    self.hide()


def changePassword(self):

    if (self.ui.passwordEdit.text() == (None or "")) or (self.ui.confirmPassEdit.text() == (None or
    "")):

        Alert("Error", QtWidgets.QMessageBox.Critical,

            "One or Both of the password fields are empty")

    else:

        if self.ui.passwordEdit.text() != self.ui.confirmPassEdit.text():

            Alert("Error", QtWidgets.QMessageBox.Critical, "Passwords dont match")

        elif (" " in self.ui.passwordEdit.text()) or (" " in self.ui.confirmPassEdit.text()):

            Alert("Error", QtWidgets.QMessageBox.Critical, "Remove spaces from password fields")

        elif ("," in self.ui.passwordEdit.text()) or ("," in self.ui.confirmPassEdit.text()):

            Alert("Error", QtWidgets.QMessageBox.Critical, "Remove commas from password fields")

        else:

            key, iv, data = getData(KEYPATH, VAULTPATH)

            data = data.decode('utf-8')

            row = data.split('\n')

            accounts = []

            for value in row:

                if value != "":

                    # stores accounts as nested lists seperated by value

```



```

        accounts.append(value.split(','))

    for i in range(len(accounts)):
        if accounts[i] == VIEWEDITEM:
            VIEWEDITEM[2] = self.ui.passwordEdit.text() # updates the item being viewed
            accounts[i] = VIEWEDITEM # updates the item in the accounts nested list
            updateAccounts(accounts) # calls updateAccounts
            Alert("Confirmed", QtWidgets.QMessageBox.Information, "Password Changed")
            self.goBack() # go to view account page after password is changed successfully

class importWin(QtWidgets.QWidget):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.ui = Ui_importAccounts()
        self.ui.setupUi(self)
        self.ui.cancelBtn.clicked.connect(self.goBack)
        self.ui.selectFileBtn.clicked.connect(self.getFile)
        self.ui.importBtn.clicked.connect(self.importData)

    def goBack(self):
        self.newWindow = allAccountsWin()
        self.newWindow.show() # show new window
        self.hide()

    def getFile(self):
        file = QtWidgets.QFileDialog.getOpenFileName(
            self, 'Open file', "", "All Files (*)") # lets user choose files from explorer
        url = QtCore.QUrl.fromLocalFile(file[0]) # gets path to file and stores it as an object
        self.ui.fileLbl.setText(url.fileName()) # adjusts file name in gui
        self.ui.fileLbl.adjustSize() # adjusts size of text wrapper for file name in gui
        self.Path = file[0] # makes path accessible in importWin

```

```

def importData(self):
    if self.ui.fileLbl.text() == "Select file to import from":
        # checks that a Key File or Vault file have been selected
        Alert("Error", QtWidgets.QMessageBox.Critical,
              "No file was selected. Please select a file to import from")
        # Alert function to display error QMessageBox
    else:
        accounts = []
        if self.ui.fileLbl.text().lower().endswith(".csv"):
            with open(self.Path, 'r') as csvFile:
                reader = csv.DictReader(csvFile, delimiter=',')
                for row in reader:
                    if ('name' in row) and ('username' in row) and ('password' in row): # lastpass format
                        if (row['username'] != "") and (row['password'] != "") and (row['name'] != ""):
                            values = [row['name'], row['username'], row['password']]
                            accounts.append(values)
                        elif ('name' in row) and ('login_username' in row) and ('login_password' in row): #
bitwarden format
                            if (row['name'] != "") and (row['login_username'] != "") and (row['login_password']
!= ""):
                                values = [row['name'], row['login_username'], row['login_password']]
                                accounts.append(values)
                    if len(accounts) < 1:
                        Alert("Error", QtWidgets.QMessageBox.Critical,
                              "CSV file format not supported or no data to import was found")
                else:
                    for item in accounts:
                        writeData(item[0], item[1], item[2])
                        Alert("Confirmed", QtWidgets.QMessageBox.Information,
                              "Imported accounts from .CSV")
                    self.goBack()
            elif self.ui.fileLbl.text().lower().endswith(".json"):
                with open(self.Path) as jsonFile:

```

```

data = json.load(jsonFile)
if 'items' in data:
    for item in data['items']: # checks for bitwarden format
        if 'login' in item:
            if ('username' in item['login']) and ('password' in item['login']):
                if (item['login']['username'] is not None) and (item['login']['password'] is not
None):

                    values = [item['name'], item['login']
                             ['username'], item['login']['password']]
                    accounts.append(values)
            else:
                Alert("Error", QtWidgets.QMessageBox.Critical,
                     "JSON file format not supported")
        if len(accounts) < 1:
            Alert("Error", QtWidgets.QMessageBox.Critical,
                 "JSON file has no data to import")
        else:
            for item in accounts:
                writeData(item[0], item[1], item[2])
            Alert("Confirmed", QtWidgets.QMessageBox.Information,
                 "Imported accounts from .JSON")
            self.goBack()
        else:
            Alert("Error", QtWidgets.QMessageBox.Critical, "File type not supported")

```

```

class exportWin(QtWidgets.QWidget):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.ui = Ui_exportPage()
        self.ui.setupUi(self)
        self.ui.cancelBtn.clicked.connect(self.goBack)
        self.ui.csvBtn.clicked.connect(self.exportCSV)

```

```

self.ui.jsonBtn.clicked.connect(self.exportJSON)

def goBack(self):
    self.newWindow = allAccountsWin()
    self.newWindow.show()
    self.hide()

def exportCSV(self):
    key, iv, data = getData(KEYPATH, VAULTPATH)
    data = data.decode('utf-8')
    path = getPathToDesktop()
    path += "\\Accounts.csv"
    if data != "":
        row = data.split('\n')
        accounts = []
        for value in row:
            if value != "":
                terms = value.split(',')
                temp = { }
                temp["name"], temp["username"], temp["password"] = terms[0], terms[1], terms[2]
                accounts.append(temp)

        with open(path, 'w') as file: # writes to csv file in lastpass format as lastpass' format is widely
supported
            columns = ['url', 'username', 'password', 'extra', 'name', 'grouping', 'fav']
            writer = csv.DictWriter(file, fieldnames=columns, lineterminator='\n')
            writer.writeheader()
            writer.writerows(accounts)

            Alert("Confirmed", QtWidgets.QMessageBox.Information, "CSV file successfully created")
    else:
        Alert("Error", QtWidgets.QMessageBox.Critical, "No accounts to export")
    self.goBack()

def exportJSON(self):

```

```

key, iv, data = getData(KEYPATH, VAULTPATH)
data = data.decode('utf-8')
path = getPathToDesktop()
path += "\\Accounts.json"
if data != "":
    row = data.split("\n")
    accounts = { }
    account = []
    for value in row:
        # json uses None for null and False for false when writing to a json
        if value != "":
            terms = value.split(',')
            loginValues = { }
            uris = [{"match": None, "uri": "http://"}]
            loginValues['uris'], loginValues['username'], loginValues['password'], loginValues['totp']
= uris, terms[1], terms[2], None
            temp = { }
            temp['id'], temp['organizationId'], temp['folderId'], temp['type'], temp['name'],
temp['notes'], temp[
                'favorite'], temp['login'], temp['collectionIds'] = "", None, None, 1, terms[0], None,
False, loginValues, False
            account.append(temp)
    accounts['items'] = account
    with open(path, 'w') as file:    # writes to csv in lastpass format
        json.dump(accounts, file, indent=4)
    Alert("Confirmed", QtWidgets.QMessageBox.Information, "JSON file successfully created")
else:
    Alert("Error", QtWidgets.QMessageBox.Critical, "No accounts to export")
self.goBack()

def getPathToDesktop():
    # path to desktop is different on windows and unix systems as on windows the drive the desktop is
on can be changed

```

```

if system() == 'Windows':
    desktopPath = os.environ["HOMEPATH"] + "\\Desktop" # finds path to desktop
    for driveLetter in ascii_uppercase: # find drive desktop folder is on
        if os.path.exists("{0}:{1}".format(driveLetter, desktopPath)):
            desktopPath = "{0}:{1}".format(driveLetter, desktopPath)
    else:
        desktopPath = os.path.join(os.path.join(os.path.expanduser('~')), 'Desktop')
return desktopPath

```

```

def Alert(title, icon, text):
    # creates QMessageBox based on arguments in function
    message = QtWidgets.QMessageBox()
    message.setWindowTitle(title)
    message.setIcon(icon)
    message.setText(text)
    message.exec_()

```

```

def getData(pathToKey, pathToVault): # allows me to access Paths throughout document
    global KEYPATH, VAULTPATH
    KEYPATH, VAULTPATH = pathToKey, pathToVault
    readVaultFile = open(VAULTPATH, 'rb') # Open the file to read bytes
    iv = readVaultFile.read(16) # Read the iv out - this is 16 bytes long
    ciphered_data = readVaultFile.read() # Read the rest of the data
    readVaultFile.close()
    readKeyFile = open(KEYPATH, 'rb')
    key = readKeyFile.read()
    readKeyFile.close()
    cipher = AES.new(key, AES.MODE_CBC, iv=iv) # Setup cipher
    # Decrypt and then up-pad the result
    data = unpad(cipher.decrypt(ciphered_data), AES.block_size)

```

```
return key, iv, data
```

```
def writeData(nameOfAccount, username, password): # writes name of account, username and  
password to vaultFile
```

```
    global KEYPATH, VAULTPATH  
    key, iv, data = getData(KEYPATH, VAULTPATH)  
    data += ("{}},{ }\n".format(nameOfAccount, username, password)).encode('utf-8')  
    cipher = AES.new(key, AES.MODE_CBC, iv=iv)  
    ciphered_data = cipher.encrypt(pad(data, AES.block_size))  
    vaultFile = open(VAULTPATH, "wb") # creates vault file  
    vaultFile.write(cipher.iv)  
    vaultFile.write(ciphered_data)  
    vaultFile.close()
```

```
def updateAccounts(data):
```

```
    global KEYPATH, VAULTPATH  
    key, iv, oldData = getData(KEYPATH, VAULTPATH)  
    accounts = []  
    for value in data:  
        row = ','.join(value)  
        accounts.append(row)  
    newData = b"  
    for line in accounts:  
        newData += ("{} \n".format(line)).encode('utf-8')  
    cipher = AES.new(key, AES.MODE_CBC, iv=iv)  
    ciphered_data = cipher.encrypt(pad(newData, AES.block_size))  
    vaultFile = open(VAULTPATH, "wb") # creates vault file  
    vaultFile.write(cipher.iv)  
    vaultFile.write(ciphered_data)  
    vaultFile.close()
```

```
if __name__ == "__main__":  
    # displays when starting application  
    app = QtWidgets.QApplication(sys.argv)  
    startPage = MainWindow()  
    startPage.show()  
    sys.exit(app.exec_())
```



## Appendix B – Project Proposal

<b>Student name:</b> Kamran Bhatti
<b>Project title:</b> Offline password vault
<p><b>Problem definition (3-5 lines)</b></p> <p>Online password managers are likely to be compromised in contrast to individual users as they store users' data on centralised servers so if the company was hacked, you and thousands of other users would have their data compromised and some of these companies may store the details of each of these accounts in plain text making these accounts more vulnerable. This project solves this by making password generation and storage offline in encrypted or non plain text files.</p>
<p><b>Project description and objectives (2-3 paragraphs)</b></p> <p>The project should mimic the style of online password managers but without storing any of a user's information online thus all account information would be stored locally in an encrypted file. As this will work as an offline application the program will only work for one user per system the software is installed on. Furthermore, the program should try to have a smaller storage footprint than alternatives like Keepass.</p> <p>The program needs to be able to:</p> <ul style="list-style-type: none"> <li>- Generate unique passwords (and user can choose features for the password to have like special characters, numbers, upper/lower-case letters)</li> <li>- Let users save passwords with a name for the account and username for that account</li> <li>- Allow users to view and search through all of their accounts</li> <li>- Allow users to copy their username and password associated with a specific account</li> <li>- Encrypt the file used to store users account details</li> <li>- Decrypt upon entering or locating the password or decryption key for the vault</li> <li>- Allow users to remove accounts from their vault</li> <li>- Import account details from online password managers like LastPass</li> <li>- Export account details into a readable format for any other type of password manager</li> <li>- Provide dialogue boxes for errors or user intervention</li> </ul>
<p><b>Required HW/SW</b></p> <p>I will need to use python to create the logic and graphical interface of the application and I may need to use a USB or external drive in order to store the encrypted file to isolate the account data from the host system. I may need to use a website for distributing the software such as on GitHub.</p>

Table 6