# Design

## Specification and application structure

After the completion of the background research, it was necessary to form a specification so that a formal list of user requirements can be shown, which concisely displays what the system should do. These requirements are split into two categories: functional and non-functional, and they were [listed on the GitHub repository](https://github.com/outerme/1808827-FYP#specification) for the project:

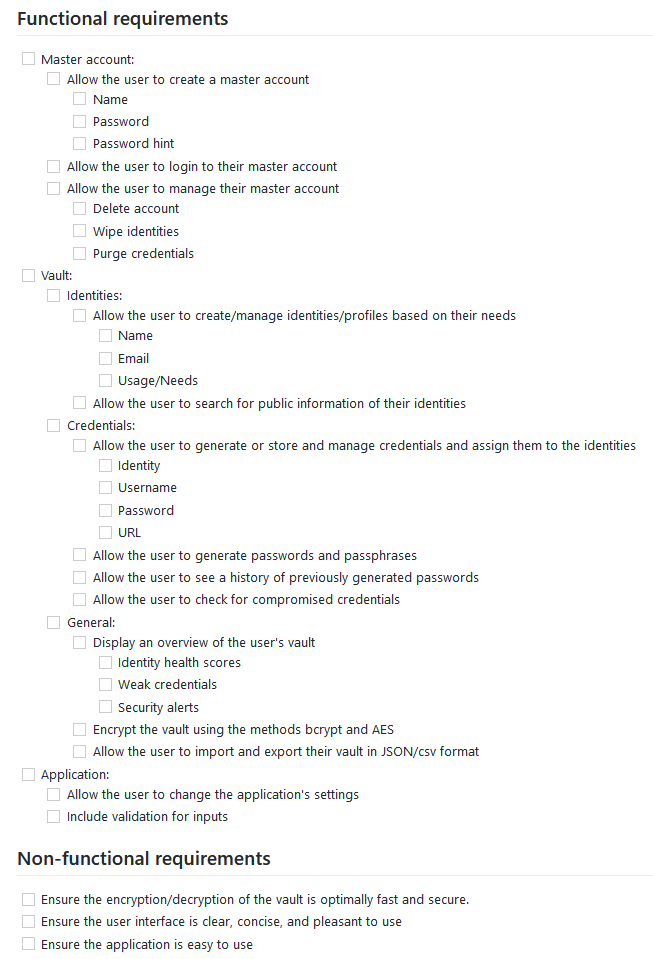


Figure 4‑1 Functional and non-functional requirements

These requirements were an extension of the already formed requirements from the background research described in the sections 0 and 2.10 through the unification and improvements of some of the key features of these password managers, and they were also later refined to conform to the interfaces which were designed as an early prototype and vice versa; these designs were discussed in section 4.2. This assists in development as it clearly defines what the system should have well and it’s beneficial for tracking which requirements were completed and which to be focused next.

With the specification created, it became easier to understand the structure of the application, and so to illustrate this, [a flowchart was created showing an overview](https://raw.githubusercontent.com/outerme/1808827-FYP/main/Designs/Interfaces/vIDsafe%20overall%20flowchart.png) without the use of specific inputs or outputs:

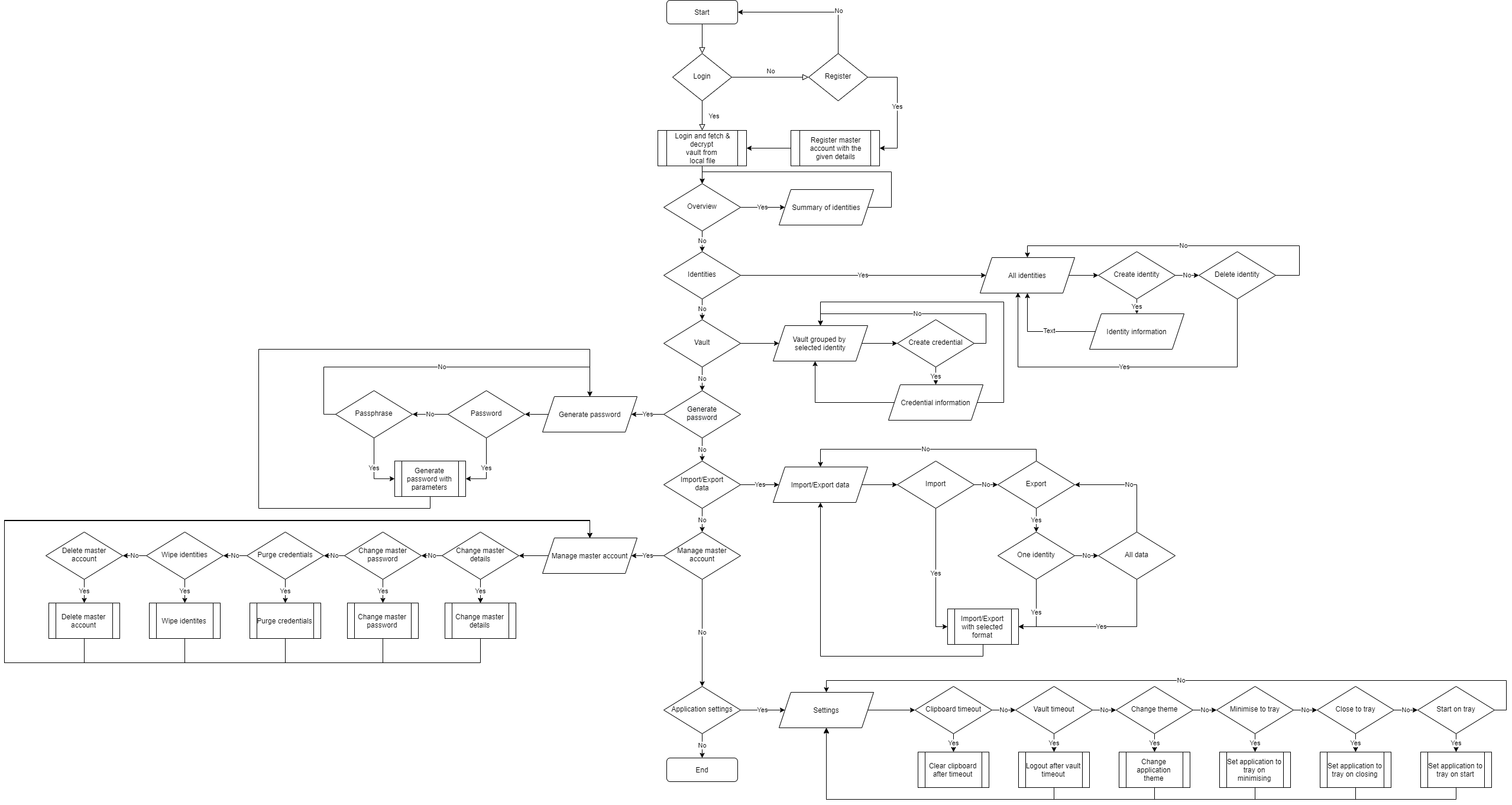


Figure 4‑2 Overall flowchart

This flowchart demonstrates the flow of the application where it displays the many tasks the user can do systems through different choices and routes, this denotes the navigation between each section of the application, and thus making it easier to create the interface designs based on this considering it also insinuates which sections should be prioritised more than the others through the number of tasks in the respective sections as well as its dependencies, which must be focused first.

An ERD was not required to be made considering the system isn’t intended to contain a database, but something to conclude from the overall flowchart is that the system would need to be created using object-oriented programming, and so a [class diagram was created](https://raw.githubusercontent.com/outerme/1808827-FYP/main/Designs/Diagrams/vIDsafe%20class%20diagram.png) to show the main entities and their attributes and functionalities that will be used.

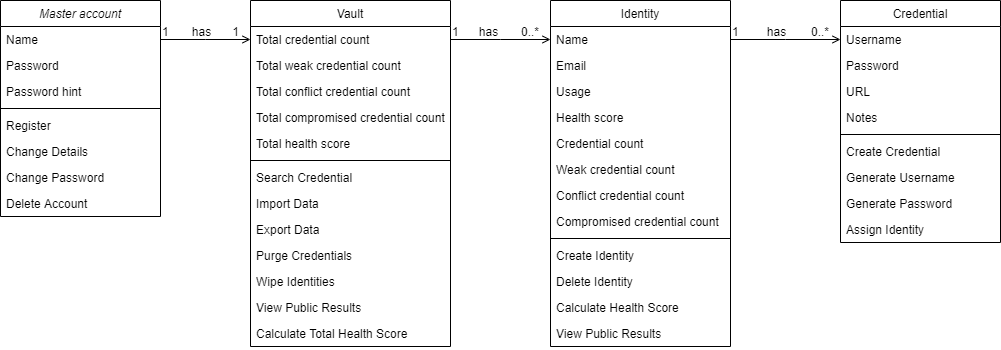


Figure 4‑3 Class diagram

Four main entities were identified where a master account may have a vault which contains many identities that contain many credentials. The diagram shows these relationships which introduces coupling, however, the methods provided by these entities ensure there’s high cohesion as they attempt to adhere to the principle of single responsibility.

## Front end design

With the specification and the structure of the system being created, it became clear as to what needs to be in the system and how it should flow from one section to another, and so to proceed, it was key that the system must be visualised in a way that will display a concept of how the final product may look like. For this, [user interface designs were created](https://github.com/outerme/1808827-FYP/tree/main/Designs/Interfaces) with the requirements and the flow in mind, although it’s also important to note that this may not be the penultimate design of the system as it may also not fulfil any future additions to the requirements.

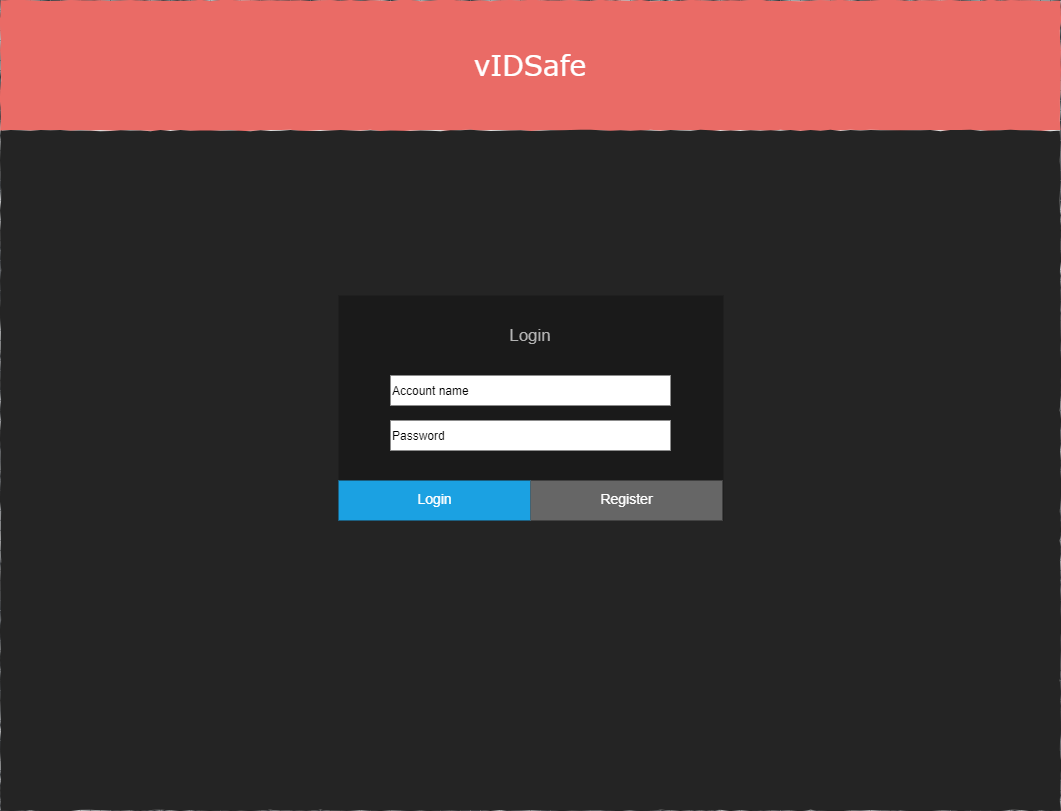


Figure 4‑4 Login screen

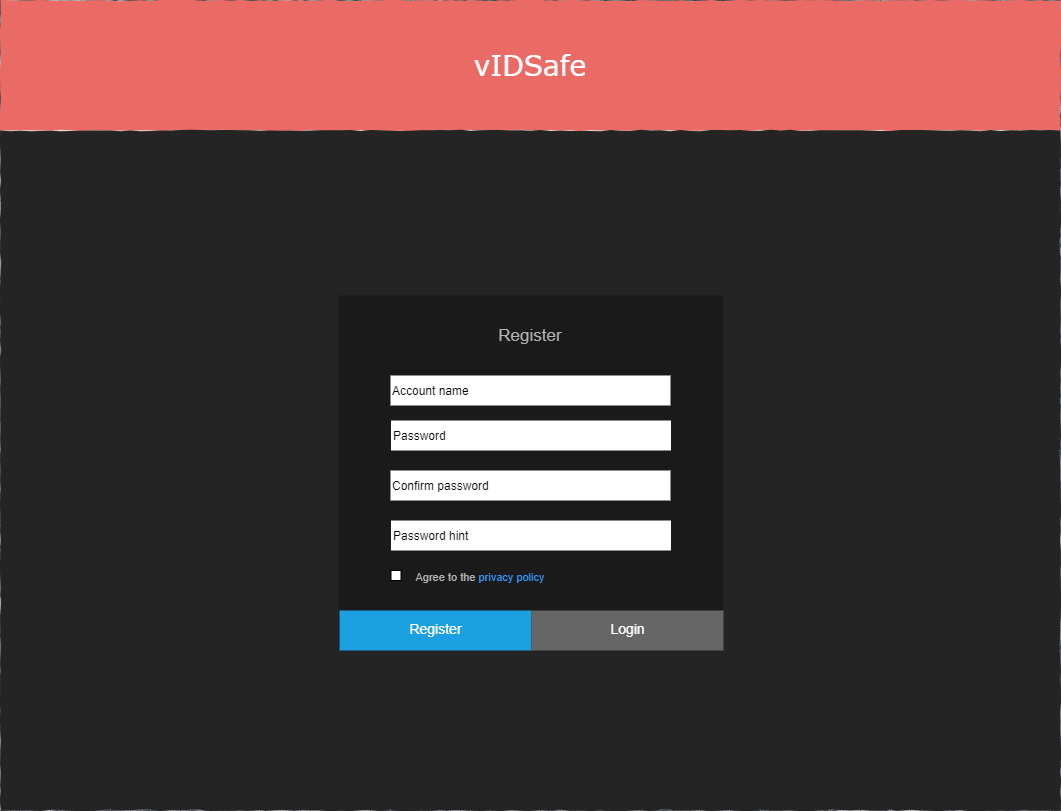


Figure 4‑5 Register screen

To begin, a screen each for the [login](https://github.com/outerme/1808827-FYP/blob/main/Designs/Interfaces/vIDsafe%20interfaces-Login.png) and the [register](https://github.com/outerme/1808827-FYP/blob/main/Designs/Interfaces/vIDsafe%20interfaces-Register.png) were created, as shown from Figure 4‑4 and Figure 4‑5 respectively in which the user is given the option to register a master account using a name of their choice, password, and a password hint, for which they can use to login, this covers most of the master account requirements.

Next, screens were created to show the details of the vault, this was through an overview of all identities, creation of an identity, and the storage of credentials, supported by the generation of the credentials as well as importing and exporting data.

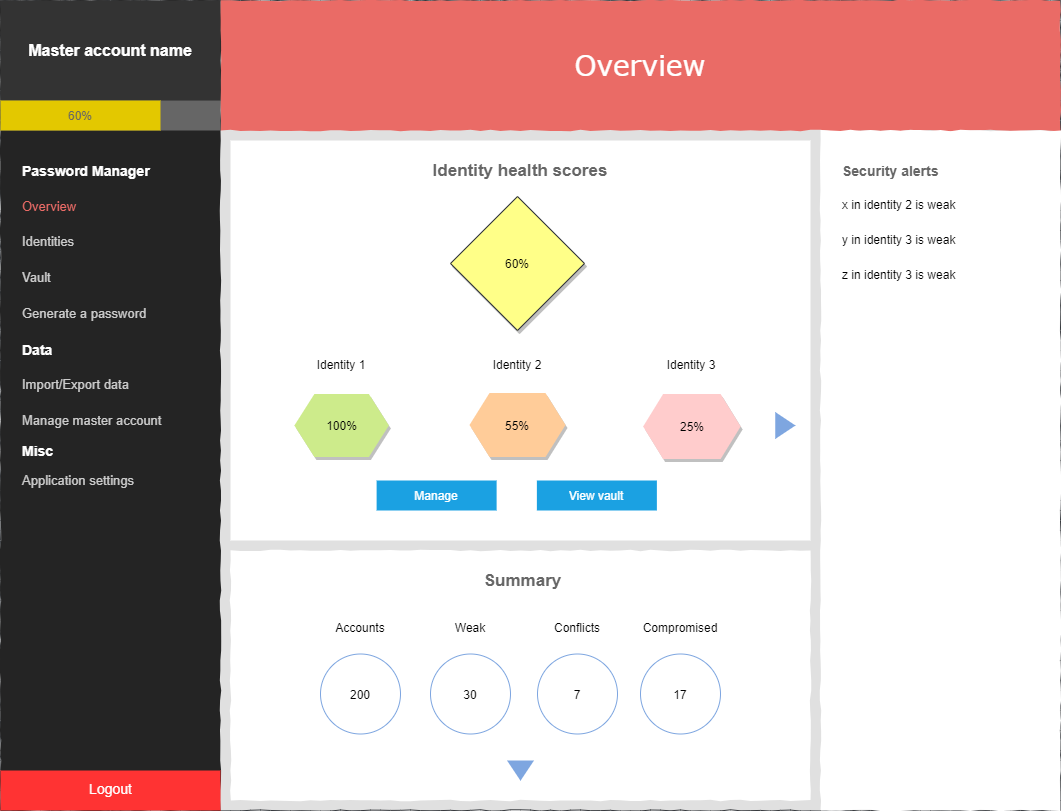


Figure 4‑6 Overview screen

The [overview](https://github.com/outerme/1808827-FYP/blob/main/Designs/Interfaces/vIDsafe%20interfaces-Overview.png) is the simplest out of all as it’s mostly displayed in shapes as shown from Figure 4‑6, which is due to statistics gained from the created identities, this is done to ensure that the user can get a clear understanding of their vault and the weaknesses of their identities.

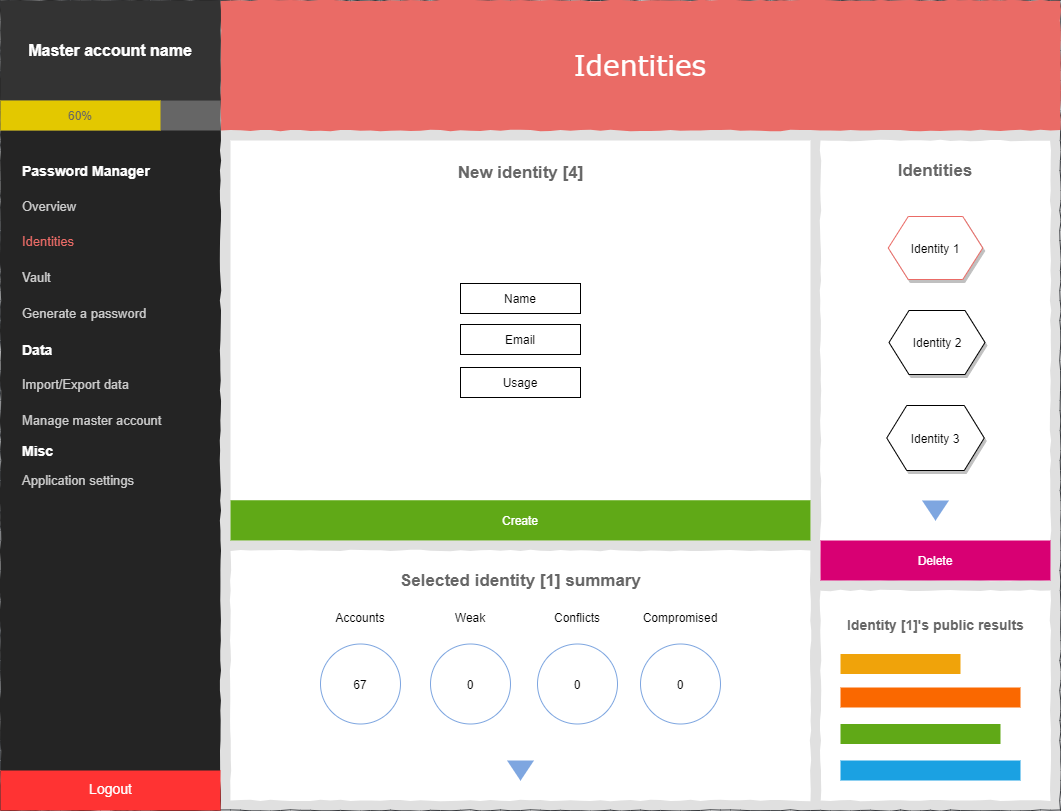


Figure 4‑7 Identities screen

The [creation of such identities](https://github.com/outerme/1808827-FYP/blob/main/Designs/Interfaces/vIDsafe%20interfaces-Identities.png) is shown on Figure 4‑7 which also lets the user see their public information based on the selected identity; these identities contain credentials associated to them, which are stored in the user’s vault.

The [vault](https://github.com/outerme/1808827-FYP/blob/main/Designs/Interfaces/vIDsafe%20interfaces-Viewing%20vault.png) is a key feature of this system as it contains all the user’s credentials as this system is still a password manager, and so it’s important that a screen specifically for this is also created:

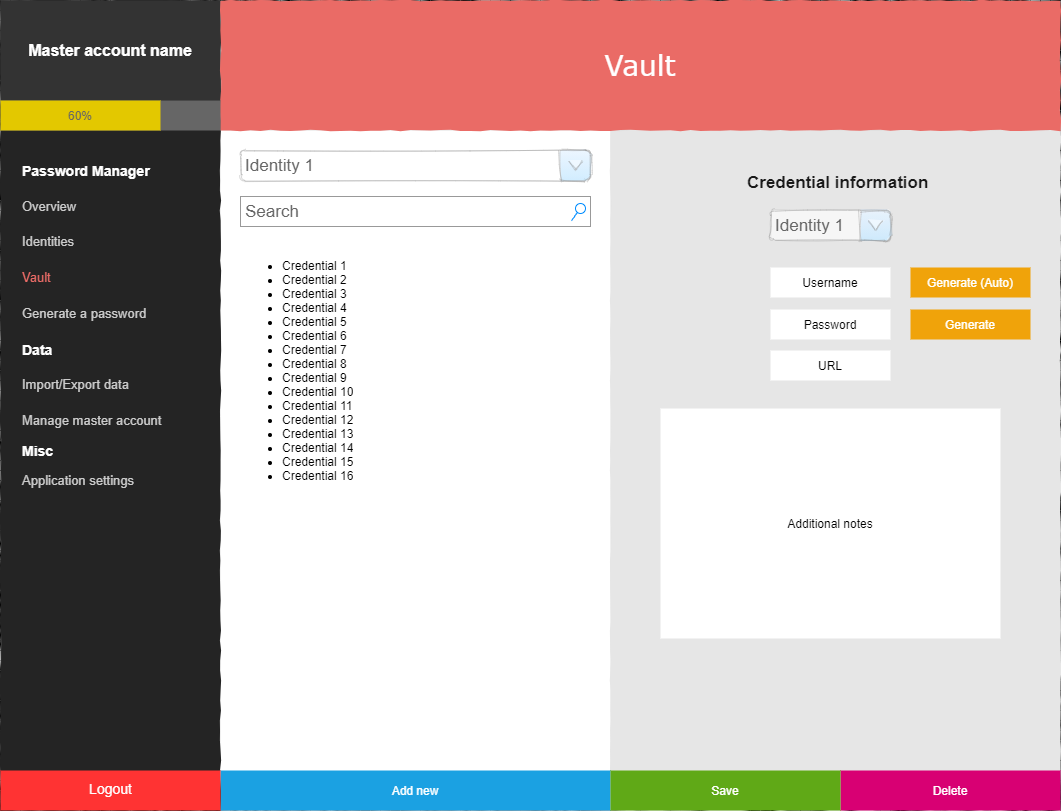


Figure 4‑8 Vault screen

As shown in Figure 4‑8, it allows the user to select an identity and to view its existing and to create/generate new credentials, where the username may be based on the identity’s name as indicated by the text “(Auto)”.

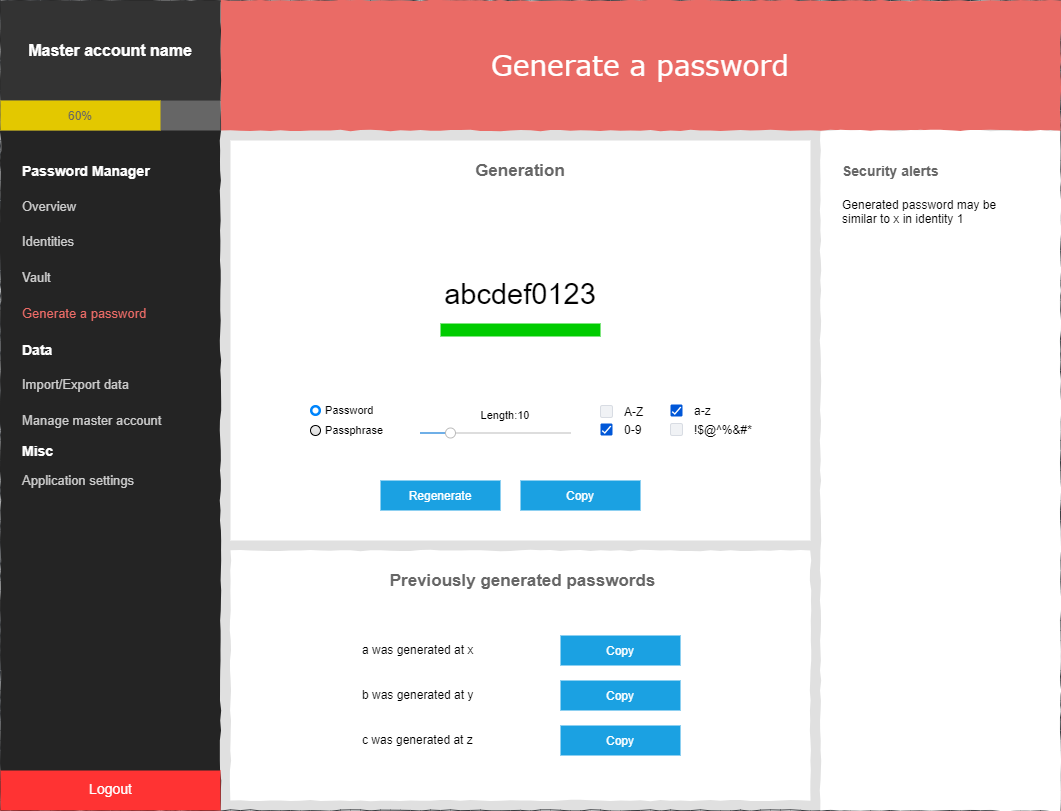


Figure 4‑9 Password generation screen

The [generation of a password](https://github.com/outerme/1808827-FYP/blob/main/Designs/Interfaces/vIDsafe%20interfaces-Password%20generation.png) is shown in Figure 4‑9 where it allows the user to generate a password or a passphrase with a bar below displaying the strength of the generated credential based on the selected parameters while also displaying the recently generated as well as the weaknesses.

Moreover, this entire vault must be portable so that the user can use their vault outside of the system by [importing/exporting](https://github.com/outerme/1808827-FYP/blob/main/Designs/Interfaces/vIDsafe%20interfaces-ImportExport%20data.png), and so there’s a screen for this as well:

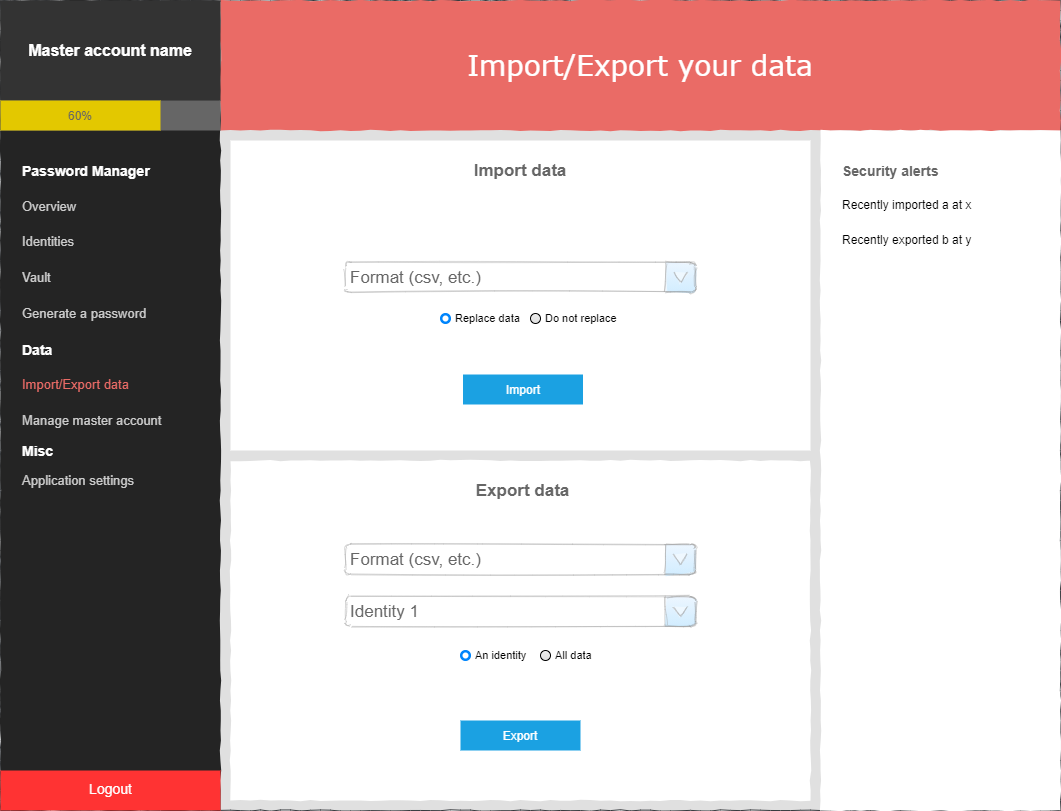


Figure 4‑10 Import/Export screen

As shown in Figure 4‑10 where the user can select their format of choice with the given parameters and import/export their vault; these then covers the rest of the requirements regarding the vault.

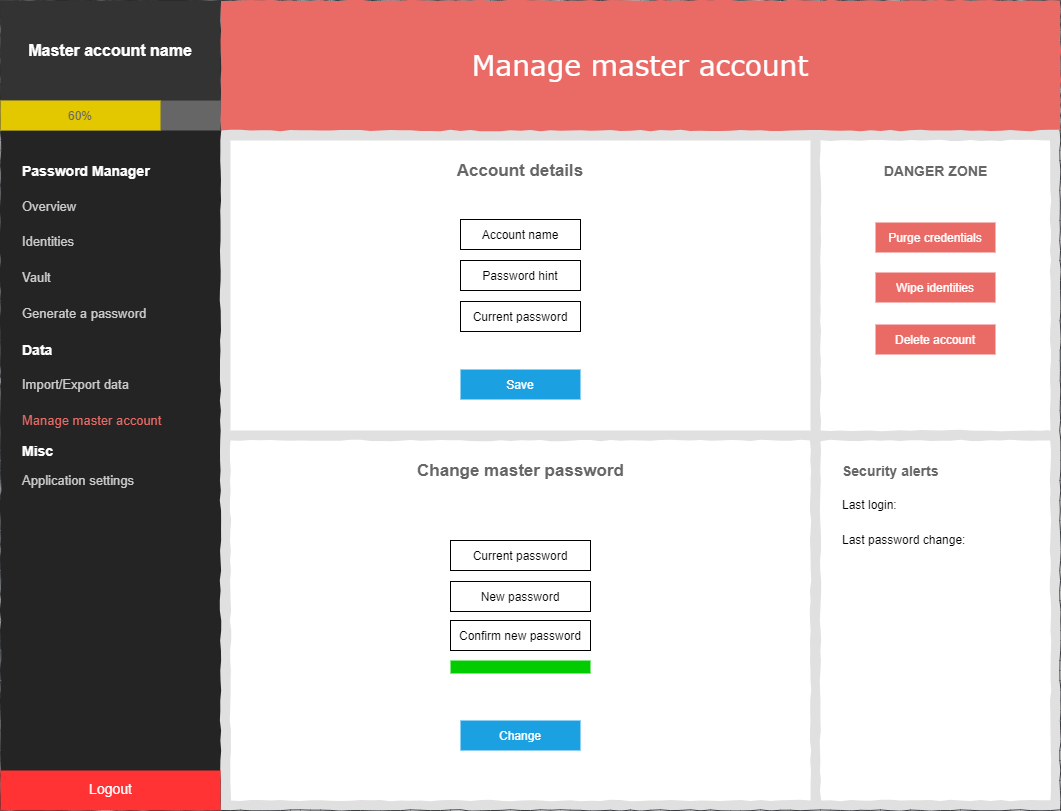


Figure 4‑11 Manage master account screen

Similarly, there’s also the screen to [manage the master account](https://github.com/outerme/1808827-FYP/blob/main/Designs/Interfaces/vIDsafe%20interfaces-Manage%20master%20account.png) as shown in Figure 4‑11 where the user is given the option to change their details as well as erase their existing data, this covers the remaining master account requirements.

Finally, there’s the screen to [application settings](https://github.com/outerme/1808827-FYP/blob/main/Designs/Interfaces/vIDsafe%20interfaces-Application%20settings.png), which is shown in Figure 4‑12, this is to be continued in the implementation as it will depend on the future additions to the requirements:

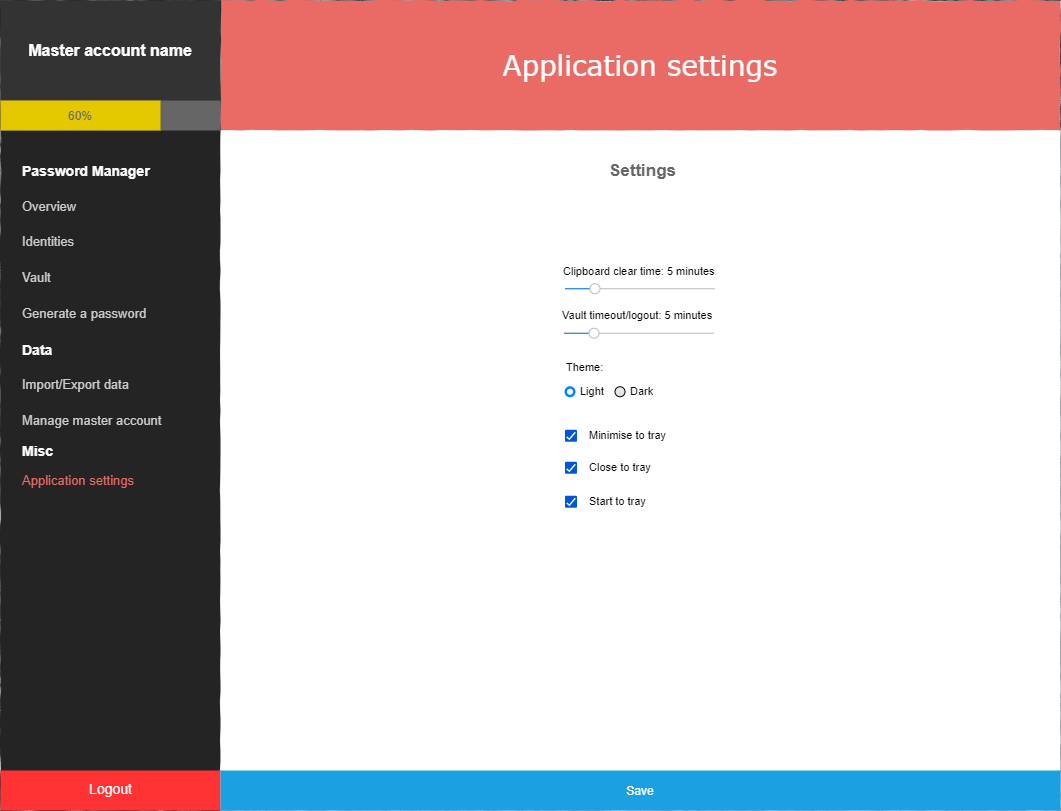


Figure 4‑12 Application settings screen

For now it includes the basic general settings of the application, although it is planned to focus more on the UI as well as notification settings for the security alerts shown in the previous figures.

­­­

A key thing to note in all these interface designs is that they are all simple, and pleasing to use, this is mainly due to the colour scheme used where the important parts are highlighted with vibrant colours, but they are also contrasted with the text on top of them which has a different colour, which assists in readability, these same colours remain throughout all screens for consistency so that the user would understand the consequences of an action just from the colour of its button (e.g. red button for dangerous actions).

Furthermore, regardless of what the requirements are, these designs also apply the design principle “workload reduction” where everything is made to be coherent using section titles as well as the minimisation of elements on the screen, this reduces the memory load as it eliminates mental calculations, estimations, comparisons, and unnecessary thinking (Association, 2005). This also involves the idea of proximity as different groups of elements are also spaced out while similar elements are grouped together using short distance and outlined sections, which is helpful as people tend to see elements as related if they are close to each other in comparison to other objects even if other features differ (Studio, 2017).

## Back end design

After the structure and the interfaces were designed, it was easy to see how the system should operate and what should be in it, although they didn’t coherently describe what the user can do and what these actions are being handled by, and so [a use case diagram was created](https://github.com/outerme/1808827-FYP/blob/main/Designs/Diagrams/vIDsafe%20use%20cases.png) as shown below:

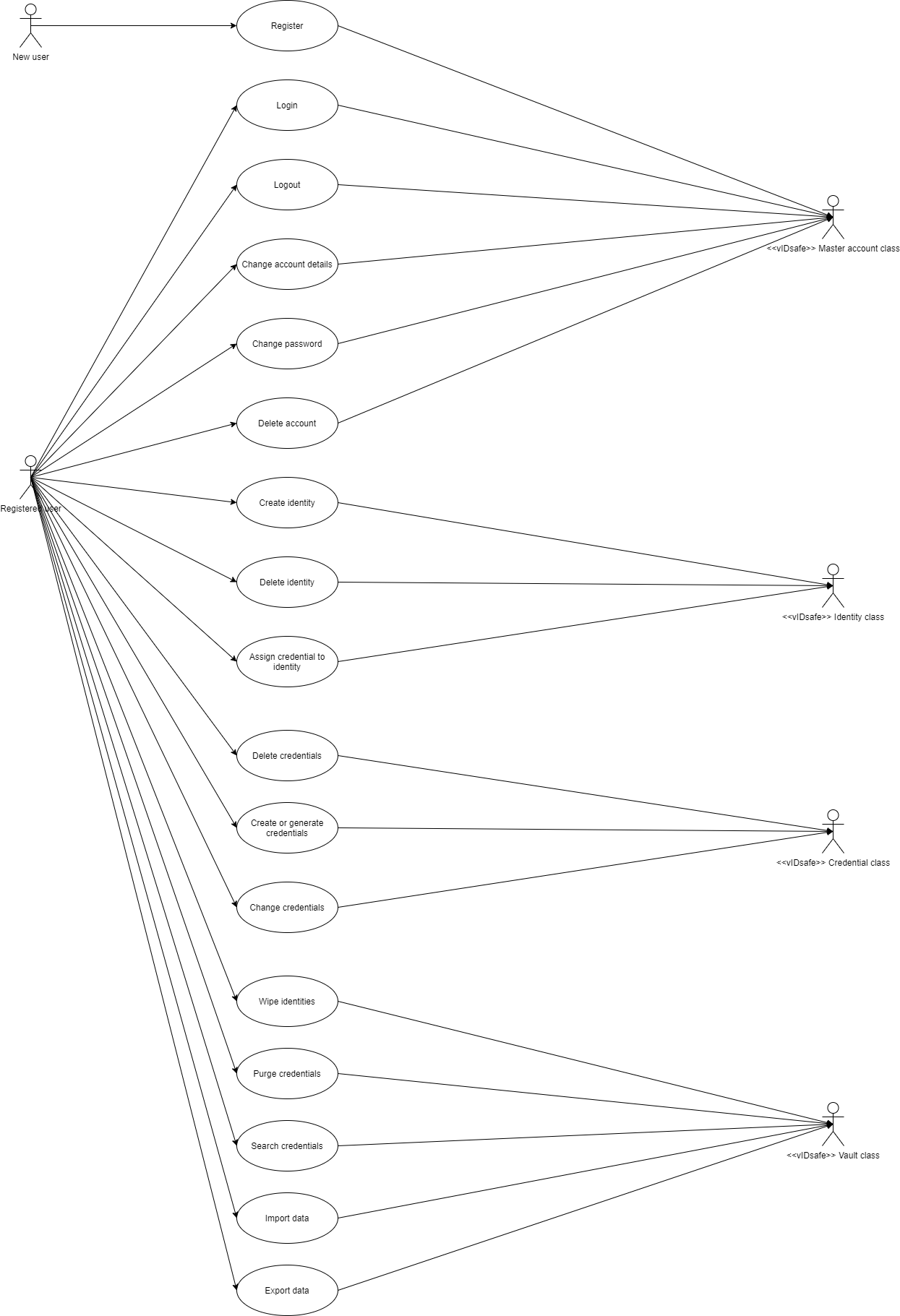


Figure 4‑13 Use cases

This diagram shows two things in particular: actors and goals. On the left side, the actors include a “New user” and a “Registered user”, whereas on the right it’s the internal classes/sections in the system that handles the actions given by the actors on the left side. These actions are otherwise known as goals or use cases as they are what the actors can do to gain what they need, from this diagram it’s concludable that there’s only one goal the new user has and that’s to register where afterwards the user has full access to the system as they become a registered user.

A use case diagram is used more as a summary and thus it doesn’t go into a lot of details, and so elaborate this diagram, it’s crucial to show the inputs and outputs of the use cases as it helps describe the specific steps needed to complete a goal. This is shown by sequence diagrams, although only two of them were made as they were deemed to be the only use cases that required clarity.

Below shows the [creation of an identity](https://github.com/outerme/1808827-FYP/blob/main/Designs/Diagrams/vIDsafe%20sequence%20diagrams-Creating%20an%20identity.png) in the system:

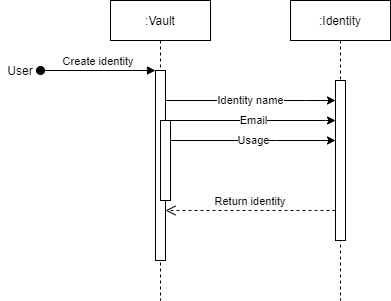


Figure 4‑14 Creating an identity

This is a very simple sequence diagram although what this denotes is that only two entities should be involved, which is the vault and the identity where the identity takes in 3 inputs and returns itself onto the vault so that the vault can be updated. Similarly, below shows the [creation of a credential](https://github.com/outerme/1808827-FYP/blob/main/Designs/Diagrams/vIDsafe%20sequence%20diagrams-Creating%20a%20credential.png):

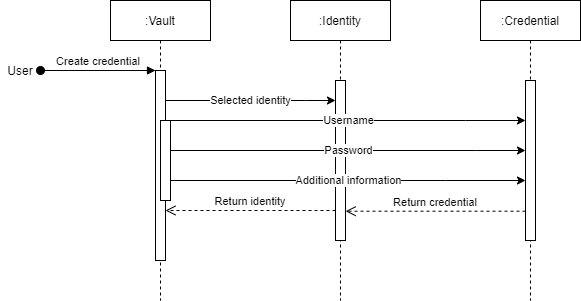


Figure 4‑15 Creating a credential

In this case, assuming an identity is already created, the user can input 3 values to create a credential in a selected identity, which then gets returned to the vault to update that existing identity. This shows that the credentials won’t be stored on its own, which will greatly help categorise and organise them so that they can be easily viewed and modified in the vault.

Regarding the back end of the system, it also includes the logic, which when observing the previous designs it doesn’t display this, and so separate flowcharts were created where the self-explanatory algorithms were omitted (as they were deemed to be time consuming) and instead some of the key and complicated algorithms were described to show how some sections of the system should operate.

Below shows the [flowchart for how the encryption should work](https://github.com/outerme/1808827-FYP/blob/main/Designs/Diagrams/vIDsafe%20backend%20diagrams-Encrypting%20credentials.png):

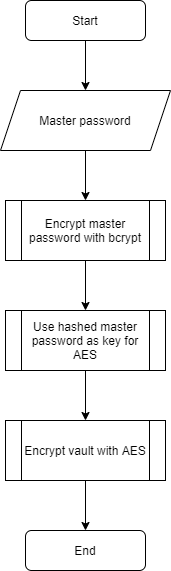


Figure 4‑16 Encrypting the vault

This may seem simple although the reason why it was crucial to make this flowchart was because two types of encryption methods are to be used in the system; bcrypt and AES, and so it’s convenient to make a flowchart describing the relation between the two where in this case it uses the master password as the input which is encrypted with bcrypt and it is used as the key for AES to encrypt the vault, this encryption will take place upon creating the master account and also making changes to the vault as well as exporting it.

This also shows that the entire vault will be encrypted and not each credential one by one, this may increase performance of the encryption overall. It wasn’t necessary to make a flowchart for the decryption of the vault as it’s very similar to the encryption as it follows the same procedure but with the inverse algorithm used for the AES decryption its library, this decryption will only be used for logging in to the vault and importing data.

Moving on from encryption/decryption, when specifically talking about the credentials, the [generation](https://github.com/outerme/1808827-FYP/blob/main/Designs/Diagrams/vIDsafe%20backend%20diagrams-Generating%20credentials.png) was slightly confusing at first, and so a flowchart was also created to make it clearer:

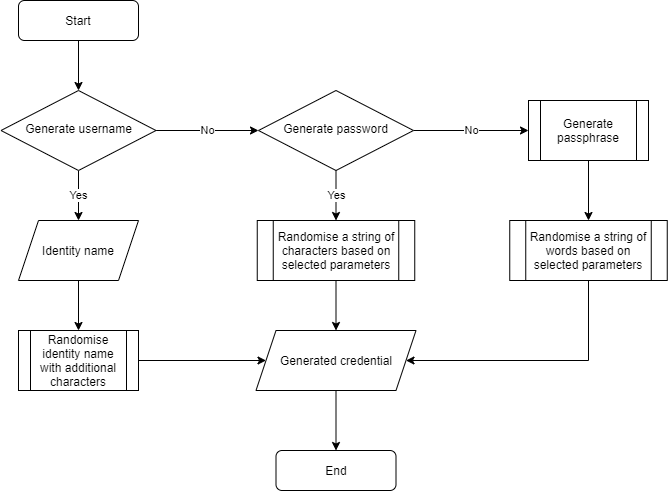


Figure 4‑17 Generating a credential

In this, it shows that the user is given the option to either generate a username or a password, where if they pick a username it’ll be based on the identity they chose, or if it’s a password, it’ll be randomised with the given parameters, but if neither, then a passphrase is similarly generated.

Although the creation of an identity is simple, [importing](https://github.com/outerme/1808827-FYP/blob/main/Designs/Diagrams/vIDsafe%20backend%20diagrams-Importing%20data.png) one could raise issues:

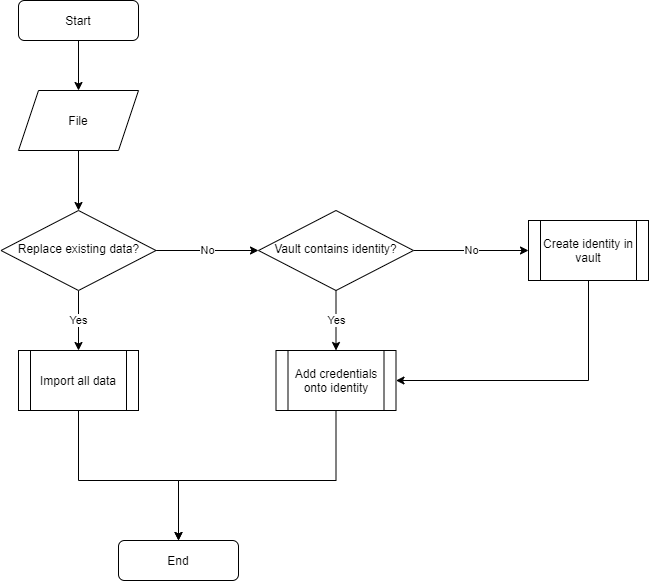


Figure 4‑18 Importing data

If the user were to just export and import the same data onto the same vault, it may create conflicts and cause irreversible errors to the identities, and so to prevent this, the user is given the option to either replace or not to replace the data, where if it’s not being replaced, it simply checks if the identity already exists which if it does it adds the credentials onto it or create a new identity for it.

Regarding the identities, a key feature of them is their [health score](https://github.com/outerme/1808827-FYP/blob/main/Designs/Diagrams/vIDsafe%20backend%20diagrams-Calculate%20health%20score.png), where all the weak credentials are gathered up and counted against the total credential count:

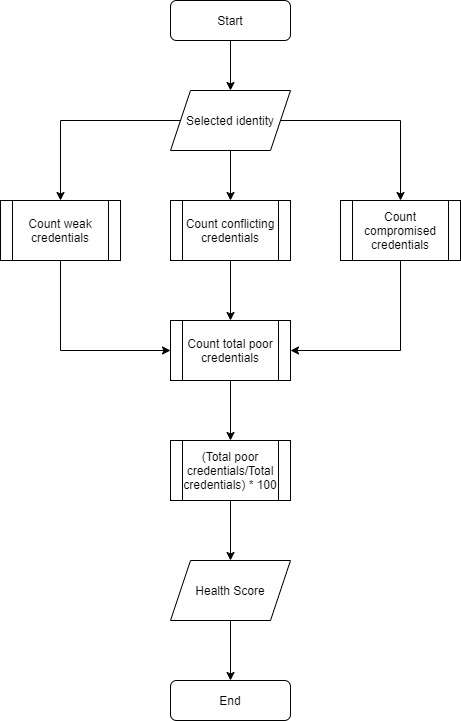


Figure 4‑19 Calculating health score

This takes three types of credentials; weak, conflicts, and compromised. These are all added up and divided by the total amount of credentials and multiplied by 100 so that a percentage is calculated depicting the overall strength for that identity.

Similarly, following shows how it checks [if credentials are breached](https://github.com/outerme/1808827-FYP/blob/main/Designs/Diagrams/vIDsafe%20backend%20diagrams-Data%20breach%20check.png):

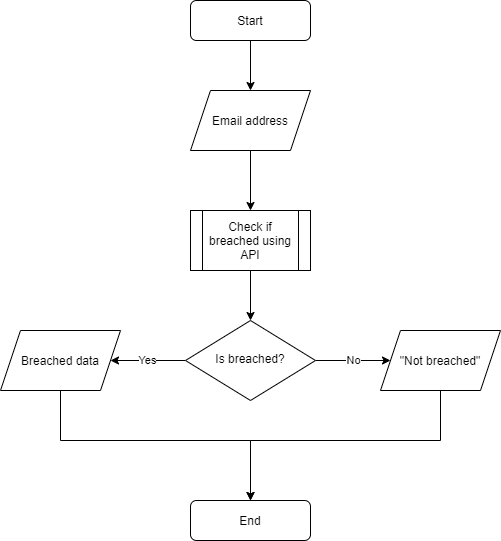


Figure 4‑20 Checking for data breaches

The method of doing this is using an email address as it’s quite effective and commonly used by data breach checking APIs such as haveibeenpwned.com, where in this instance it’ll simply return the breached data if it has been breached, or “Not breached” if it hasn’t.

Lastly, when talking about the storage of the vault, none of these designs displayed the structure of vault when it’s stored, although internally they are stored as objects using classes, externally, they must be stored somewhere locally so that they can be loaded into the system. The proposed way of doing this is by automatically creating a file of no type in a directory (chosen or auto assigned) where the name of it will be the user’s master account name where inside it’d be a long string containing the user’s identities and credentials which are encrypted using AES-256 which can only be retrieved using the user’s master account password, which will be automatically done and loaded when logging in to the vault.

Furthermore, as indicated by the specification, the user should be able to import and export their vault in the formats CSV and JSON, and so it’s only right that their structure is also specified as when the vault is exported it will be in raw data so that the user will be provided with clarity.

There are two things the user can import and export: identities, and credentials belonging to identities. Due to this, there must also be two structures made to satisfy this requirement, the following shows the structure of a CSV file of how identities will be stored as:

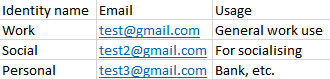


Figure 4‑21 CSV format of identities

When looking at the same structure but as a keyed JSON format converted in [convertcsv.com](https://www.convertcsv.com/csv-to-json.htm), it shows the following:



Figure 4‑22 JSON format of identities

You can see that the identity names are used as the objects which encloses the details of that identity, this not only increases readability, but it also helps ease the serialisation and the deserialization of the vault into/from objects when using libraries such as [ServiceStack.Text](https://github.com/ServiceStack/ServiceStack.Text) and [Json.NET](https://www.newtonsoft.com/json).

Although they may seem very similar, it shows a different result when looking at the CSV and JSON formats of how credentials are stored:

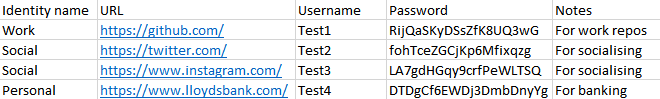


Figure 4‑23 CSV format of credentials

Here you can see there’s two instances of the identity “Social” in use, due to this, in the keyed JSON format, it shows the following:

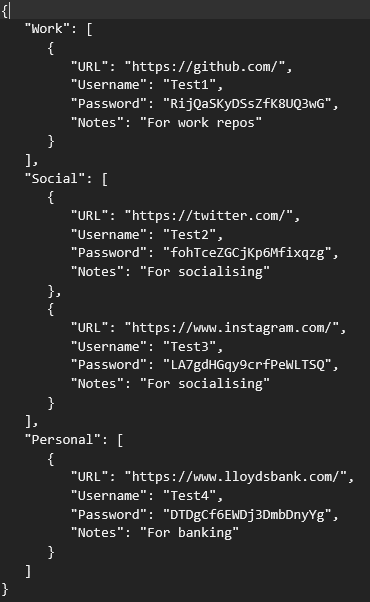


Figure 4‑24 JSON format of credentials

Unlike in Figure 4‑22, for the keys/objects, square brackets are used instead of curly brackets, this indicates that it’s a list hence why the “Social” identity contains two instances of the same attributes.

One thing to note is that if a credential doesn’t contain an identity, the user will be prompted to select an identity for it whereas if it contains an email instead and it doesn’t yet exist in the identities, an identity can be created for it.