­­Testing and Evaluation

## General and unit testing

The effects of this application are mostly shown in the long-term in the cases of data loss through exposures and more, therefore its success isn’t easily quantifiable and thus ensuring that the application still works as expected is still good, and hence why testing is also included as part of the evaluation.

Throughout the development of this application, there was consistent testing carried out, since there were some methods which where complex and at times long, it was important that immediate testing must be done before proceeding to write incorrect code, one of ways this was done was through the old-fashioned way of printing variable values and debugging code coverage paths, for which the inbuilt method “WriteLine” was used. This is due to the fact that it’s very easy and fast and it helped debug all logical errors in the application as opposed to test-driven development which would be considerably slower.

While most of the testing in development wasn’t formal, white-box testing was carried out as there were still many unit tests implemented after the completion of the application, this was to ensure that the vital parts of it were functioning correctly, hence why there were assertions used. These unit tests were all in a separate project under the same solution, however, since key classes were being tested, for organisation, the tests for each class were separated into different testing classes, as shown below:

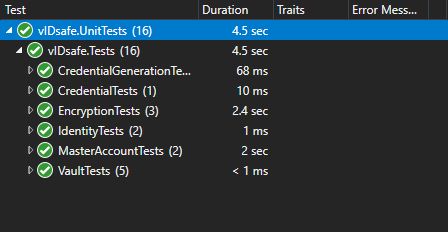


Figure 6‑1 Tested entities

As shown, all of the tests had passed, and it’s also worth noting that performance testing wasn’t really a requirement considering that the test duration already alludes the efficiency of the tested methods. There was also continuous integration implemented using GitHub Actions where these tests are re-run upon committing to the repository, ensuring that there aren’t any errors caused by the changes as shown in Figure 6‑2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test method** | **Test description** | **Expected result** | **Actual Result** | **Duration** |
| TryImportVaultTest | Try creating, delete, and import the vault in CSV, JSON, encrypted | 1 total credential | 1 total credential | 968.2ms |
| TryExportVaultTest | Try creating and export the vault as CSV, JSON, encrypted | True/Exported | True/Exported | 942ms |
| CalculateOverallHealthScoreTest | Check health score of a vault with an unsafe and a safe credential in two separate identities | 50 | 50 | < 1ms |
| TryChangeIdentityEmailTest | Try change an identity’s email | True/Changed | True/Changed | < 1ms |
| DeleteIdentityTest | Try delete an identity in a vault | True/Deleted | True/Deleted | < 1ms |
| DeleteAllCredentialsTest | Try delete all credentials in a vault | 0 total credentials | 0 total credentials | < 1ms |
| DeleteAllIdentitiesTest | Try delete all identities in a vault | 0 total identities | 0 total identities | < 1ms |
| CalculateHealthScoreTest | Check health score of an identity with an unsafe and a safe credential | 50 | 50 | 1ms |
| DeleteAllCredentialsTest | Try delete all credentials in an identity | 0 total credentials | 0 total credentials | < 1ms |
| CalculateStatus | Calculate the status of a credential with a duplicate username | Conflicted status | Conflicted status | 8ms |
| GenerateUsernameTest | Try generating a username | 15-character username | 15-character username | < 1ms |
| GeneratePasswordTest | Try generating a password and a passphrase | 15-character password and 5-word passphrase | 15-character password and 5-word passphrase | < 1ms |
| CheckStrengthTest | Check strength of a password | Strength higher than 30 | Strength higher than 30 | 78ms |
| DeriveKeyTest | Try deriving key from a secret | Hashed password | Hashed password | 761.6ms |
| AesEncryptTest | Try encrypting the vault using AES | Encrypted text | Encrypted text | 764ms |
| AesDecryptTest | Try decrypting a vault using AES | Decrypted text | Decrypted text | 787.2ms |

Table 6‑1 Test results

As seen from Table 6‑1, all tests that were carried out gave the expected result, this meant that the internal operations work as expected. These tests were simply testing the key public methods of the main classes, it wasn’t necessary to do testing of the private methods as these public methods invokes them and thus encapsulating their functionality in a way, this is further elaborated in section 5.7.

Moreover, these tests were repeated five times to form an average, each of these tests took at max 1 second to run, this completes the non-functional requirement of “optimally fast” encryption/decryption as the password hashing test (DeriveKeyTest) only took 761.6ms, while encrypting/decrypting the vault took 764ms and 787.2ms, respectively. The reason why this is important is that most of the time-consuming tasks such as encrypting/decrypting or importing/exporting a vault are done asynchronously, which prevents the UI from being blocked while they’re being processed, and so with these durations being this low, the user won’t notice any blocking even if it wasn’t being done asynchronously.

When talking about the import (TryImportVaultTest) and export (TryExportVaultTest) testing in particular; these two tests weren’t done exactly as planned as all 3 formats were tested at once. Furthermore, it was initially supposed to have credentials generated through an online site and placed in CSV and JSON files for testing, although it wasn’t necessary to do this as there were methods implemented to generate identities and credentials, which makes the testing process much easier and faster.

Below shows the aforementioned implemented GitHub Actions continuous integration:

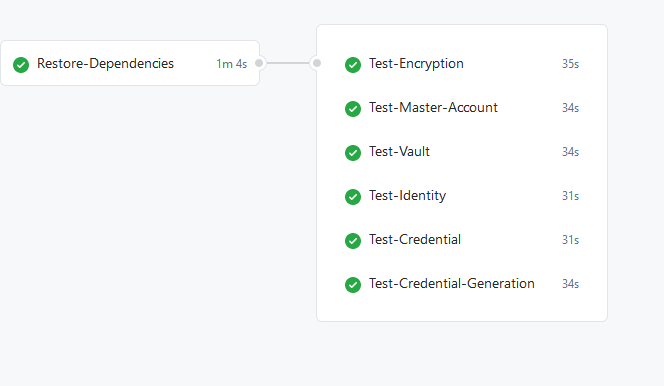


Figure 6‑2 GitHub actions automated testing

The test duration times in GitHub actions may differ due to virtual machine assignment as well as checkout times and hardware differences, although it allows uploading artifacts, which in this case is the test build in the “Restore-Dependencies” job, which must successfully pass to run the other tests.

## Code quality metrics

Since the project is open source, it was important to ensure that the quality of the code is good enough for maintenance, this was elaborated in section 5.7, however, considering that the development of this application was done through object-oriented programming, it was also necessary to use certain metrics in addition for the code quality as a lot of functionalities were split by methods which were separated by different classes for different uses.

For this, a GitHub application known as “CodeFactor” was used, this made things simpler as on each commit, it would display a grade based on the quality of the code on all but the excluded files and folders, which in this case was “A”, which is the second-best grade. This grade is determined by a few metrics, where one of them includes “Duplication”, and where for this project, there were only 2 instances of exact duplicate code as shown in Figure 8‑9.

This is very minimal, and they are related to the “form” classes, and these instances aren’t related to any objects instantiated, although disregarding this, when looking at the rest of the classes, all of them are given an “A” grade which is optimal.

If the “form” classes and the external classes are ignored, the class with the highest complexity value is the “Porting” class with a value of 31, which is still considered fine. The way this value is determined is using a few other metrics, namely complexity per method (C/M) and lines of code per method (LOC/M); higher the value, the “worse” the code is.

Below shows an overview of these metrics for the classes created for this system:

|  |  |  |
| --- | --- | --- |
| **Statistic** | **Complexity per method** | **Lines of code per method** |
| Average | 2.134 | 13.1595 |
| Maximum | 4.33 | 27 |
| Minimum | 1 | 1 |
| Median | 1.81 | 12.925 |

Table 6‑2 Overview of metrics

As shown from the above table, on an average, there’s roughly 13 lines of code, with a method having as much as 27 lines of code, the average complexity per method on the other hand is 2.134, with a maximum of 4.33, from observing other projects that use CodeFactor, it’s apparent that a value below 5 is decent although a value around 2 is ideal.

Some of these values are slightly surprising as they seem excessive, although they are still optimal as determined by the analyser CodeFactor use for C#, StyleCop, and thus the code quality can be considered good enough for a final product.

## User feedback

Once the project was completed and the testing was carried out, it was appropriate to gather some user feedback regarding the system through different perspectives, hence why there was a research conducted with 6 participants where each briefly use the system and give their opinion through a questionnaire. However, before data was even collected, it was taken into consideration that this research would not determine the success of the application for the same reason described in section 6.1, although it’s still very useful in identifying visual issues, hence why only 6 people were recruited.

The questionnaire consisted of 8 questions in total where 5 of them were functional and 3 of them were non-functional. All of these questions can fall into both qualitative as well as non-qualitative data as the functional questions consists of the answers “Yes” and “No” where the “No” option gives the user a field to give an open answer, whereas with non-functional questions the user can select from 1 to 5 with an additional option “Other” to do the same. It is formatted this way so that in each question, the user is given the opportunity to voice their opinion in detail as opposed to the predefined choices.

Below shows the functional questions:

|  |  |
| --- | --- |
| **Question number** | **Question** |
| Q1 | Does the system provide a clear overview of your identities? |
| Q2 | Does the system help identify weakness in your identities? |
| Q3 | Does the system provide a way of online compartmentalisation? |
| Q4 | Does the system allow the management of all your data? |
| Q5 | Does the system assist you in minimising data loss in the future? |

Table 6‑3 Functional questions

Below shows the non-functional questions:

|  |  |
| --- | --- |
| **Question number** | **Question** |
| Q6 | Rate the interface |
| Q7 | Rate the simplicity of the system |
| Q8 | Rate the ease of use of the system |

Table 6‑4 Non-functional questions

Below shows the results for these questions:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Participant** | **Q1** | **Q2** | **Q3** | **Q4** | **Q5** | **Q6** | **Q7** | **Q8** |
| 1 | Yes | Yes | Yes | Yes | Yes | Other | 4 | 4 |
| 2 | Yes | Yes | Yes | Yes | Yes | 4 | 3 | 3 |
| 3 | Yes | No | Yes | Yes | Yes | 2 | 3 | 4 |
| 4 | Yes | Yes | Yes | Yes | Yes | 5 | 4 | 5 |
| 5 | Yes | Yes | Yes | Yes | Yes | 5 | 5 | 5 |
| 6 | No | Yes | Yes | Yes | Yes | 2 | 3 | 3 |

Table 6‑5 Questionnaire results

Below shows the open answers for these questions:

|  |  |  |
| --- | --- | --- |
| **Participant** | **Question** | **Answer** |
| 1 | Q6 | “For signing up there’s no labels indicating what is required” |
| 2 | Q2 | “what the colour codes mean are clear, I'm just unsure of where they actually are” |
| 6 | Q1 | “It’s hard to navigate in the application.” |

Table 6‑6 Questionnaire open answers

As shown from the results, for the functional questions (1 to 5), most answers were “Yes”, which affirms the functional requirements regarding the identities and compartmentalisation although there were some issues regarding the interface pointed out in both functional and non-functional questions, this is elaborated in the pie chart shown below for question 6:

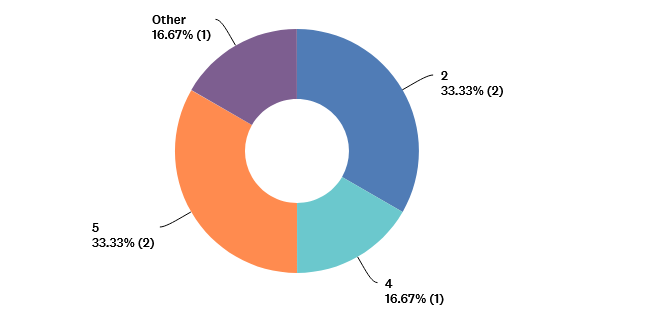


Figure 6‑3 Rating the interface

This question in particular has an even but contrasting mix of answers where there two participants had rated the interface a “2” and one have provided an “other” answer, although the rest have rated “4” or “5” and it’s indecisive as to whether not the interface is easy to use.

However, when looking at the results for question 8 regarding the ease of use of the system, it shows slightly different results:

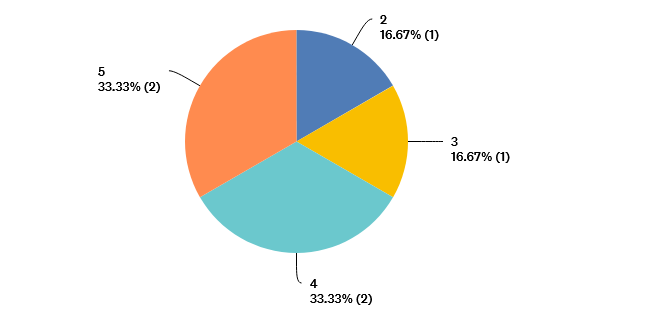


Figure 6‑4 Rating the ease of use of the system

This is important is because while the majority had rated the ease of use of the system as “4” or “5” which supports the previous question that the interface is easy to use, this is further supported by question 7 results as shown below:

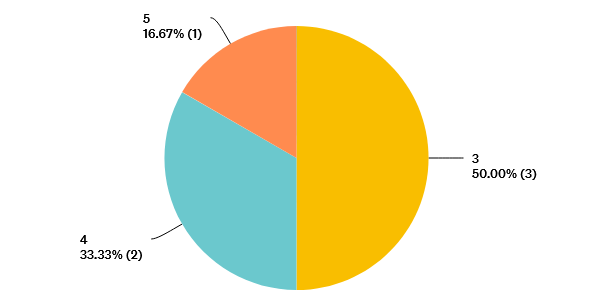


Figure 6‑5 Rating the simplicity of the system

While it’s a similar question, the results showed that all participants have rated the simplicity of the system as “3” and above, which discards the previous lower ratings as this question is focused specifically on the simplicity of the system. This affirms and completes the non-functional requirements that the application is easy to use as well as it’s conciseness and clarity.

While this research provided positive results, an issue can be pointed out regarding the selection of the participants as there was a slight selection bias as these participants were handpicked and thus some results may be deemed to be inaccurate, however, they were all computer science students and had all briefly used the system before giving their feedback which disregards this issue.

## Comparison to competitors

While there was unit testing completed and user feedback gathered, to finalise that the system provides a solution to the discovered issues, it must be compared to similar applications, which in this case are password managers.

Below shows a table of comparisons for some of the core features provided by each; these features were originally discussed and noted as part of the background research in section 2.9:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Password manager** | **Open source** | **Pleasant UI** | **Portable** | **Generate passphrases** | **Focus on identities** |
| vIDsafe | Yes | Yes | Yes | Yes | Yes |
| Bitwarden | Yes | No | Yes | Yes | No |
| DashLane | No | Yes | Yes | No | No |
| LastPass | No | No | Yes | No | No |
| LessPass | Yes | Yes | No | No | No |

Table 6‑7 Feature comparison between password managers

Ensuring that the application is open source is the first requirement as it proves that there aren’t any underlying untrustworthy features embedded into it, although 2 of these password managers do not follow this which already makes this solution better in terms of privacy. This is mainly because the code is also well documented and customisable making it easier to make changes by anyone, which is described in section 5.7, but when observing the source code of applications such as Bitwarden and LessPass, there’s simply just long methods with lack in-code comments while also being harder to navigate and find a specific functionality.

The decision that some of these password managers do not contain a pleasant UI was made through personal opinion although some have described Bitwarden’s UI to be too simple while LastPass’s UI to be overengineered and unintuitive (NicoCrash, 2021) whereas for vIDsafe, as shown in 6.3 through user feedback, this wasn’t the case where users have shared mostly positive views on the application’s UI.

LessPass is the only password manager on the list that disallows the portability of the vault, this is already a flaw as this would mean that the user’s credentials would forever be in the vault unless the user manually save each one by one. The other password managers allow porting, although the issue with vIDsafe is that while it can export as CSV, JSON or in encrypted format, it can’t be easily passed onto another password manager, although a developer can easily define the functionality for this as described in section 5.7.

An overlooked feature is the ability to generate a passphrase, where Bitwarden is the only other password manager with this functionality, the importance of this is described and explored in section 2.6. While it isn’t necessary, it’s certainly a very good feature it produces memorable passwords while still being secure, vIDsafe provides this although it isn’t as customisable as Bitwarden’s.

The most important of all is the idea of associating credentials with identities as opposed to plainly storing them unorganised, this is explained in section 2.8, and it is unfortunately not followed by any of the password managers; DashLane slightly follows this idea as it provides an overall strength score for the vault and allows the creation of identities but does not associate credentials them. vIDsafe on the other hand, gives the user the ability to create credentials based on identities created by them while also providing a strength/health score for each, which makes it more privacy focused.

Lastly, while it’s not listed in the table, a useful comparison is the memory usage of the application; most desktop applications nowadays are created using Electron, a framework for JavaScript which allows a developer to customise and develop applications with pleasant UIs. Although a drawback of this is that these applications are based on the Chromium browser, which is notorious for high memory usage; this makes most of these password managers use a lot of memory without much reason, as an example, below shows the comparison between vIDsafe, a password manager created through WinForms, and Bitwarden, a password manager created in Electron:

Figure 6‑6 Comparison of memory usage between vIDsafe and Bitwarden

While vIDsafe shows to be more lightweight, the reason why only Bitwarden was used as a contender for this is because it’s the best competitor made with Electron yet with a bland UI.

In addition to these comparisons, it’s useful to visualise how vIDsafe helps in the case of a data breach in comparison to the other password managers, below shows a simple diagram showing this for the typical password manager:

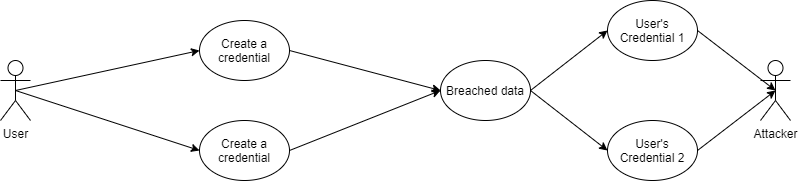


Figure 6‑7 Typical password manager data breach scenario

From this diagram, it’s shows that the average password manager allows the user to create credentials where there isn’t any precaution in the case of a data breach, where in this example the two created credentials can probably be linked back to its original user by their similar attributes such as their username, email, or password which may be the case, although such a thing is usually disregarded in a typical password manager.

This is slightly different for vIDsafe as “identities” are used:

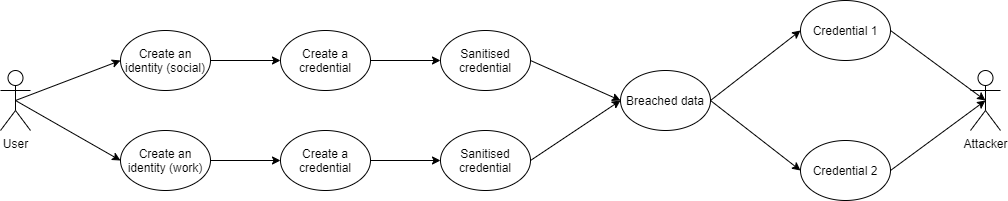


Figure 6‑8 vIDsafe data breach scenario

Using the same two credentials, they are instead separated by two identities instead, this allows vIDsafe to sanitise these credentials by checking for conflicts, breaches, and weaknesses. These two identities have two different purposes, one for socialising and other for work, where the two would have two different emails and two different identity names. Furthermore, these credentials may be generated using these identities instead of being manually created which makes it even better as there’s less identifiable data as their usernames and passwords can be randomised and knowing that they are in two separate emails they are very unlikely to be linked together.

The main difference between these two scenarios is that in the case of a data breach, vIDsafe takes precaution when storing a credential and thus making it harder for an attacker to crosslink data from different credentials through compartmentalisation through identities. While the idea is very simple, its effect is much greater in cases such as these hence why it’s a better solution in regards to privacy compared to other password managers.