PASSWORD PROGRAM: PROJECT REPORT

Kyle Christie
[B00415210]

Written 11/04/2022
Finalised 20/04/2022

Lecturer: Joanna Olszewska
Algorithms & Collections

UNIVERSITY OF THE
WEST *of* SCOTLAND
UWS

Overview

The aim of my project was to provide computer users with a secure, reliable and simple password manager program that also allows users to find out whether their data has been breached. The core principles behind this software was to improve login security and awareness of data security while making an accessible and free product. I have since implemented and tested a software solution based on the previous design brief.

Table of Contents

Table of Figures

1. Development Lifecycle

Throughout the development of my project, I utilised the 'Rapid Application Development' methodology. The reasoning behind this being practicality as well as facilitating a linear process in which I don't waste time revisiting previous steps. I could also focus on a prototype which I quickly develop, test and then finalise. I kept track of my progress via Trello creating 4 boards which represent each stage of RAD.
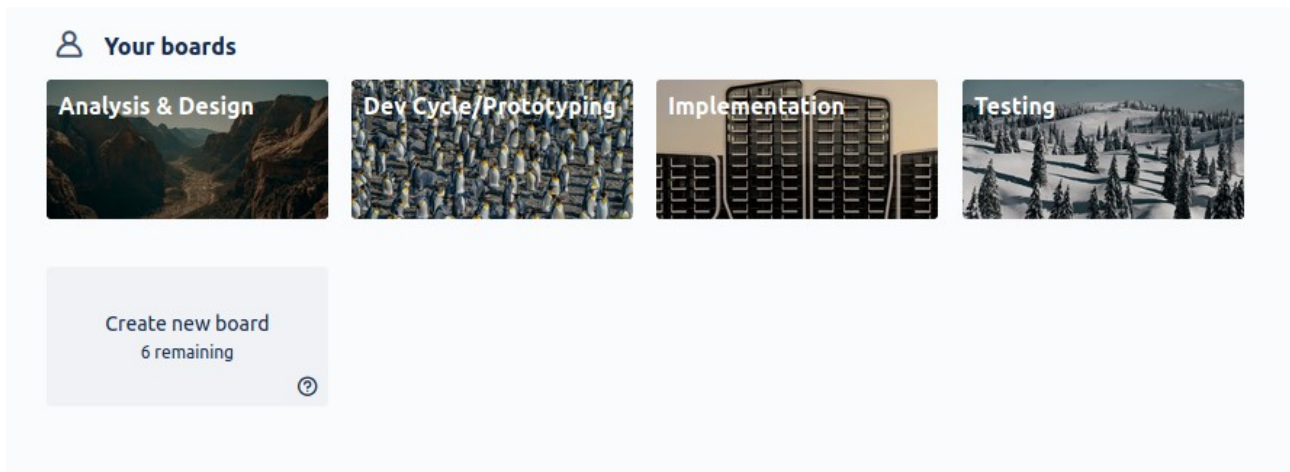
Figure 1: An overview of my Trello boards (17/4/22)

While not in order, each stage is represented. Within each board, I utilised my own Kanban style of task/issue tracking with a to-do list, in-progress list and complete list for tasks. In certain boards, I might track specific issues or write notes that might be useful as can be seen in my prototyping board which I have a specific list of 'Known bugs' with a coloured category representing current status or severity.
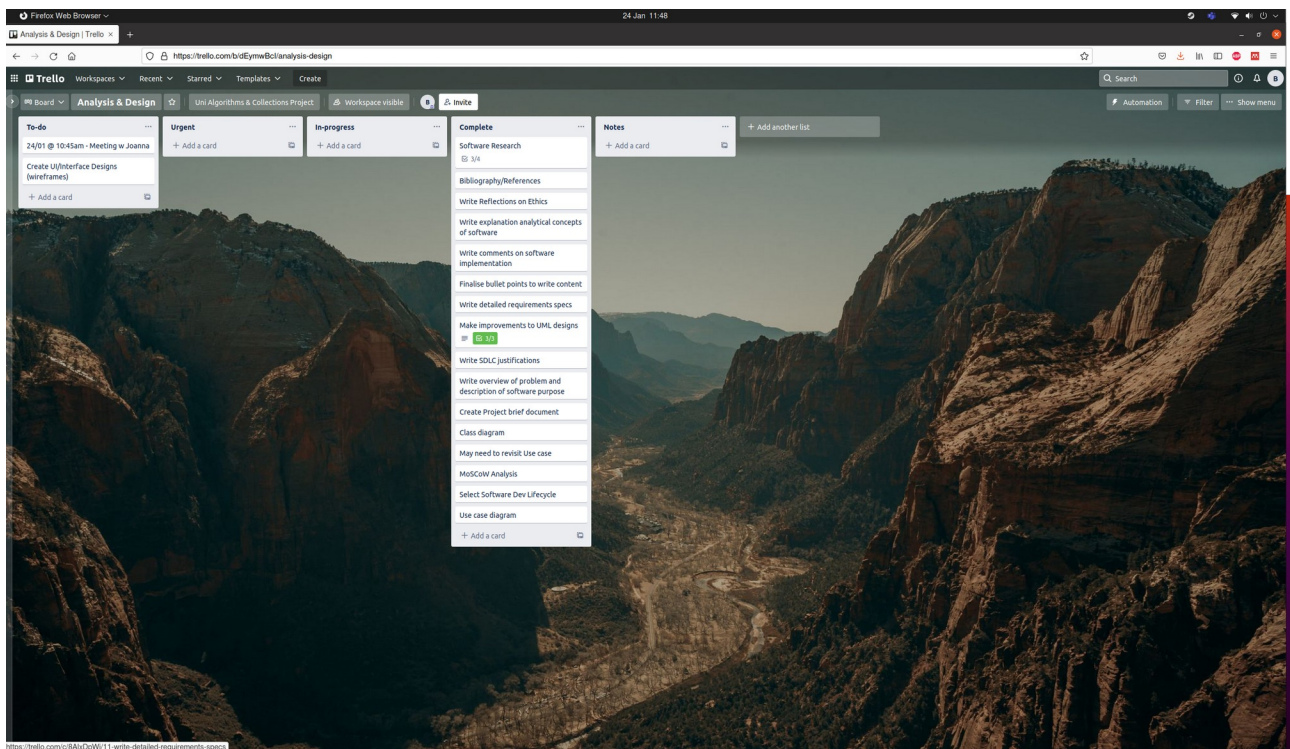
Figure 2: Trello: Analysis & Design Board (24/1/2022)
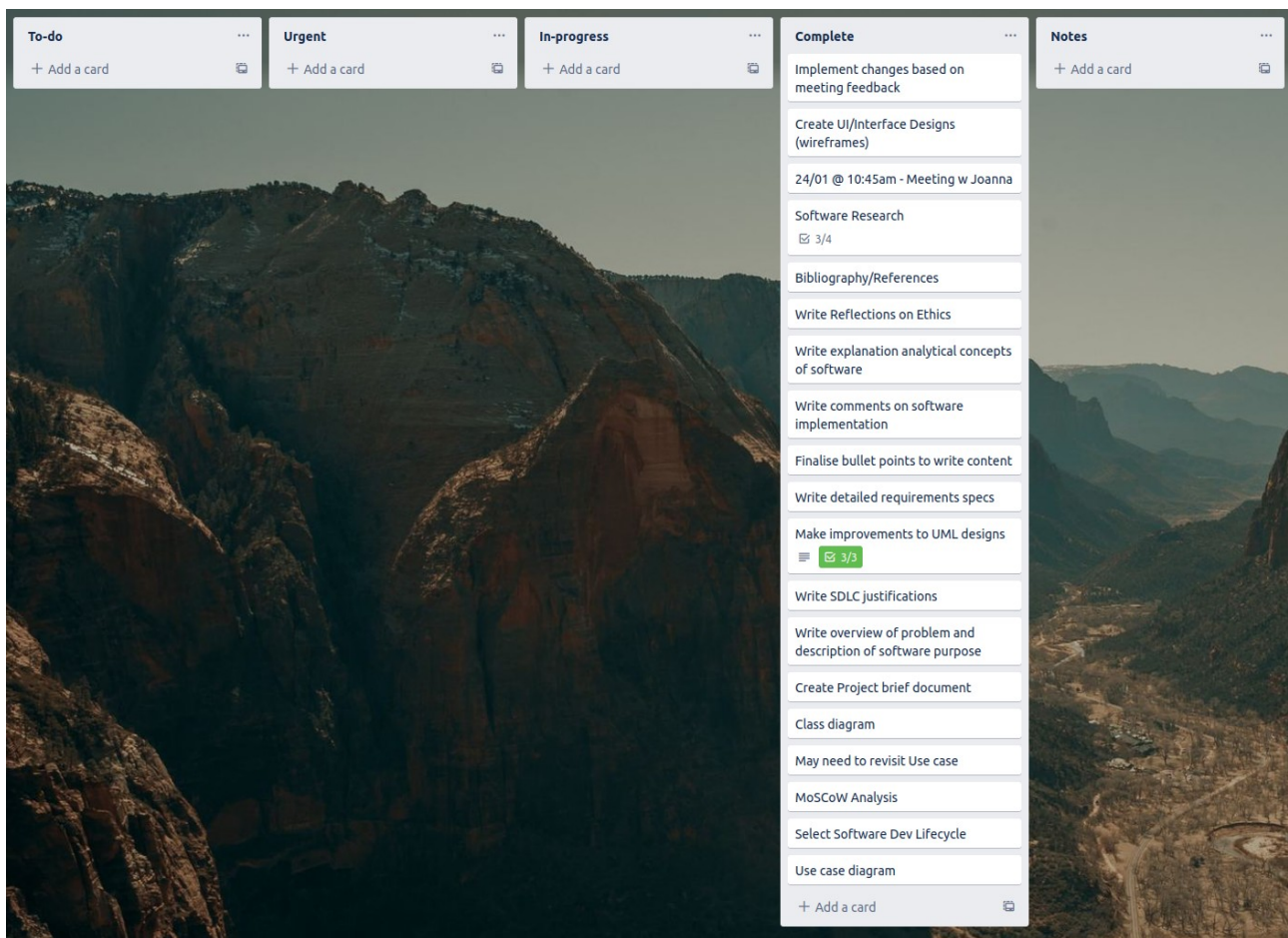
Figure 3: Trello: Analysis & Design Board (17/4/2022)
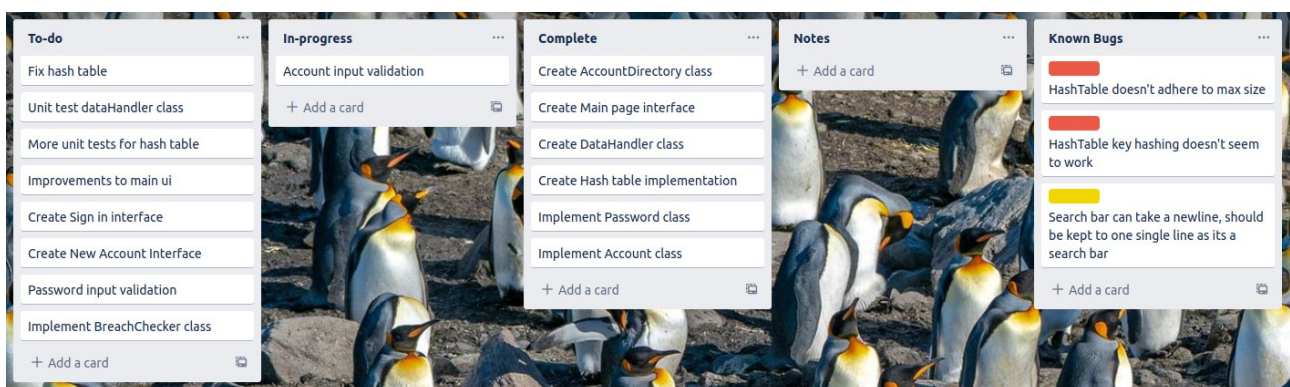
Figure 4: Trello: Dev Cycle/Protyping Board (9/3/2022)
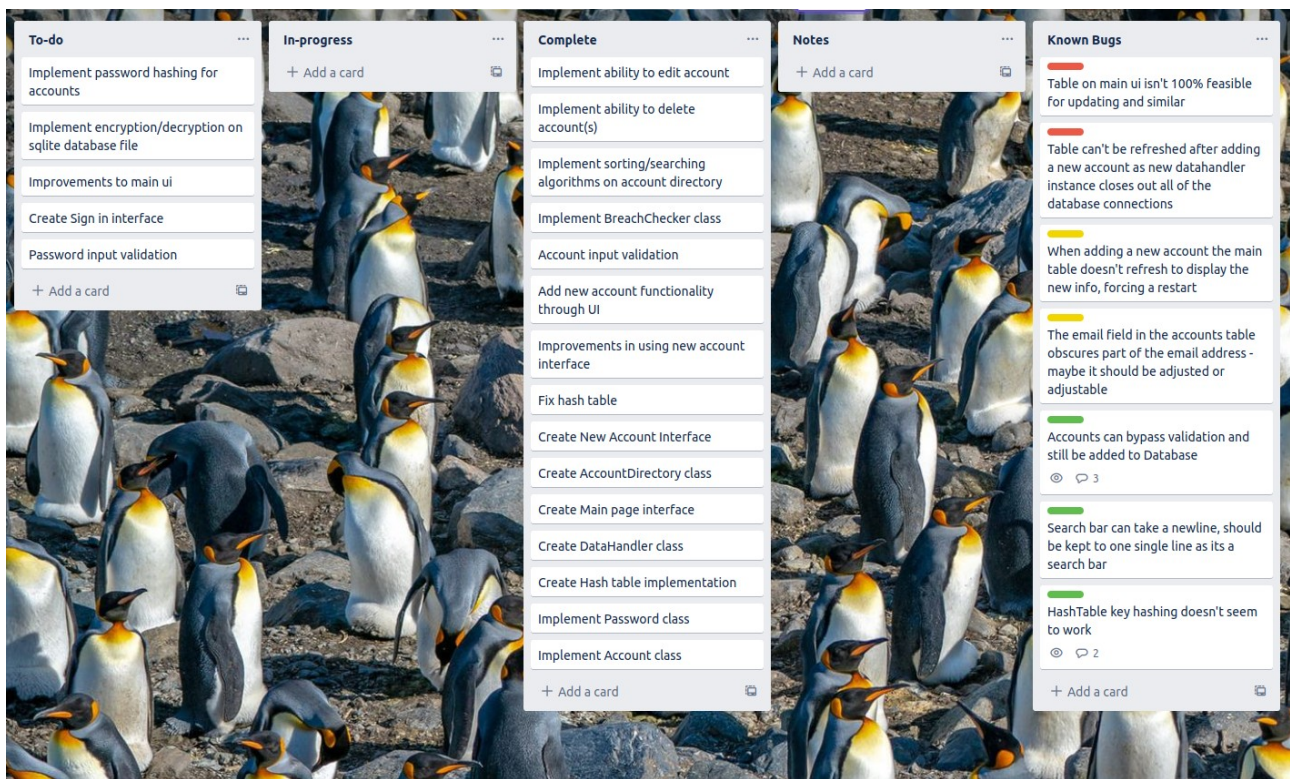
Figure 5: Trello: Dev Cycle/Protoyping Board (17/4/2022)

2. Requirements

After figuring out my base requirements, I carried out MOSCOW analysis to categorise each requirement into what is most and least necessary.
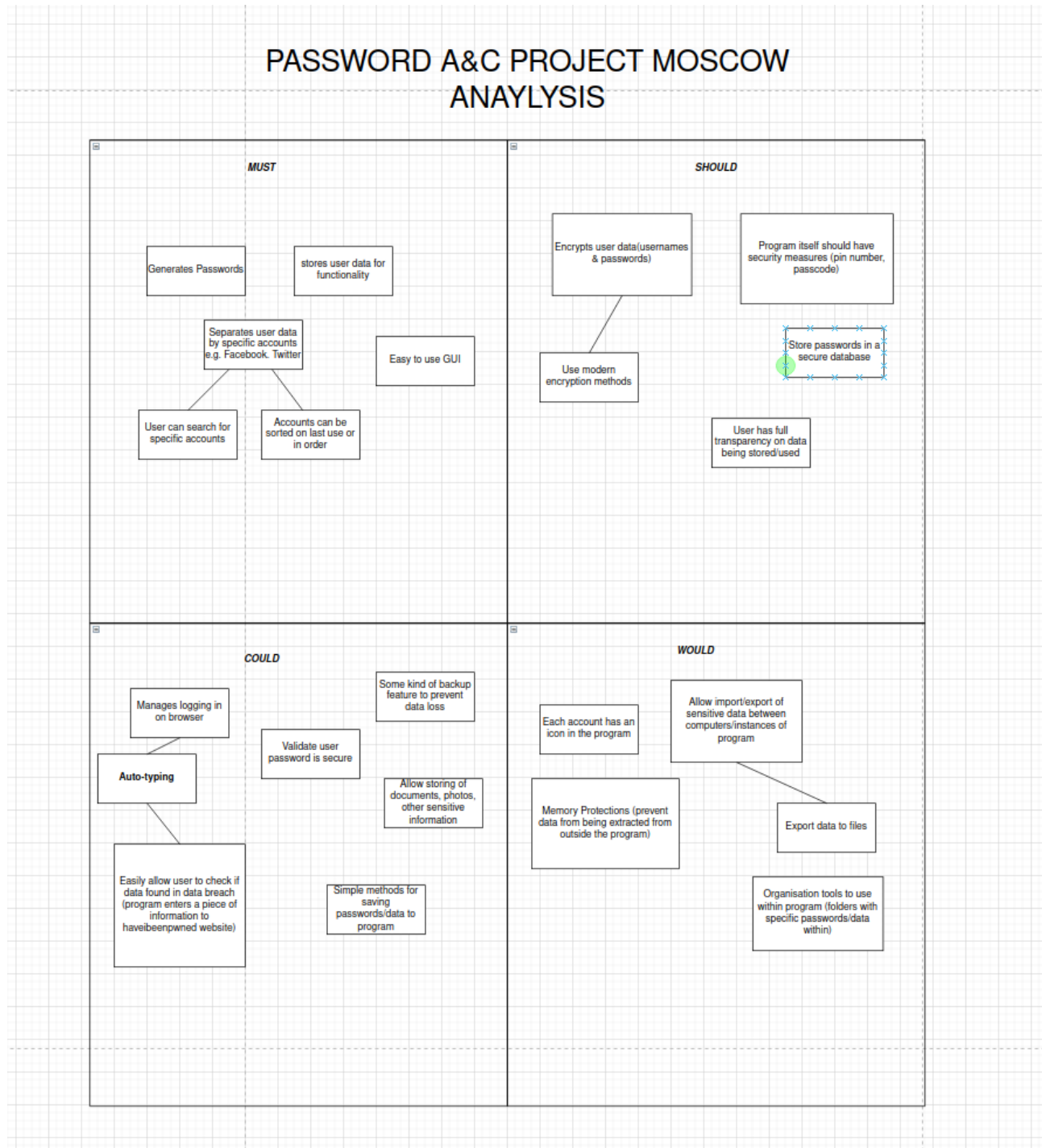
Figure 6: Initial MOSCOW analysis for Password Program

After several more designs, I developed class diagrams denoting the final requirements.
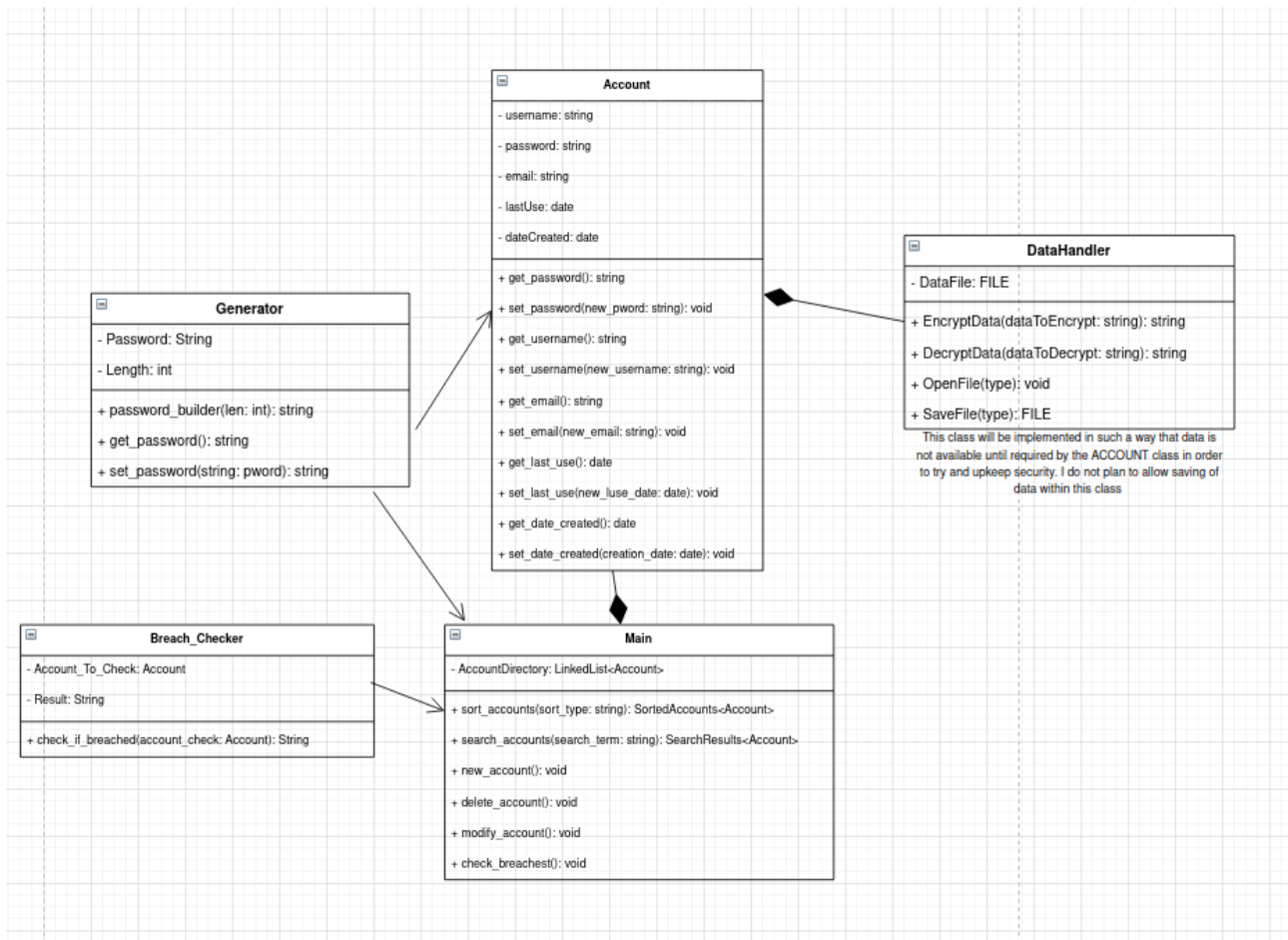
Figure 7: Initial class diagram

After several meetings with the client, I built on top of this class diagram to create the final design which would then be used throughout development – this would not be able to change due to my use of the RAD methodology.
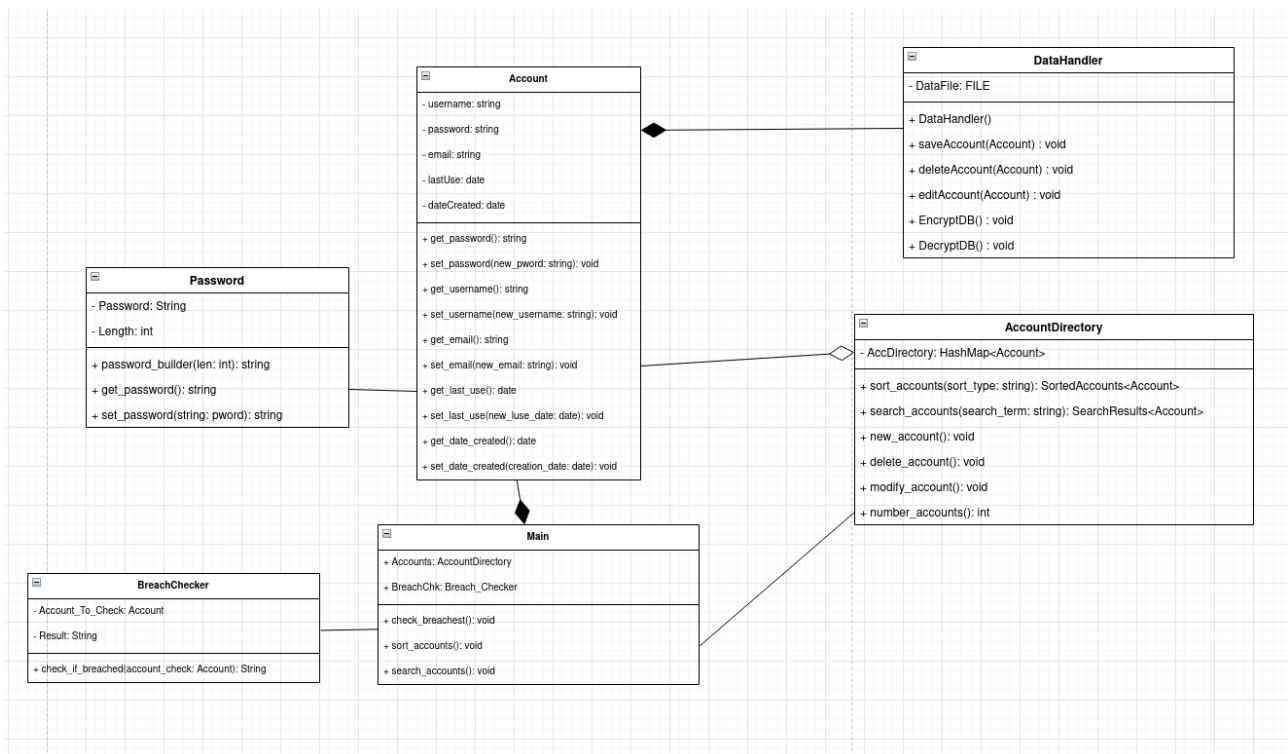
Figure 8: Finalised class diagram

In the final design, several classes are present which are necessary for the application to function. Account can be thought of as the core of the program as it allows data to be accessed and stored from a database, meeting the *strict* requirements of storing data and separating data. Password is a small component of Account and allows for the generation of secure passwords, allowing the user to better secure their accounts – another necessary requirement. AccountDirectory is built on a HashTable which allows for the storing and manipulation of specific Accounts, this was a necessary addition to efficiently manage and organise several user accounts. DataHandler allows for interaction with account storage which, in the final implementation, is an SQLite database. Finally, the class BreachChecker permits the user to check if their data was recently involved in a major data breach. This is facilitated by the use of the website 'haveibeenpwned' which tracks data breaches and alerts users when/if their data is present.

After completing my initial logic designs, I moved on to create interface designs which would then be used in the program.

Figure 9: Initial interface design

Figure 10: Improved interface design

Figure 11: Concept sign-in page

Figure 12: New Account Interface design

3. Approach

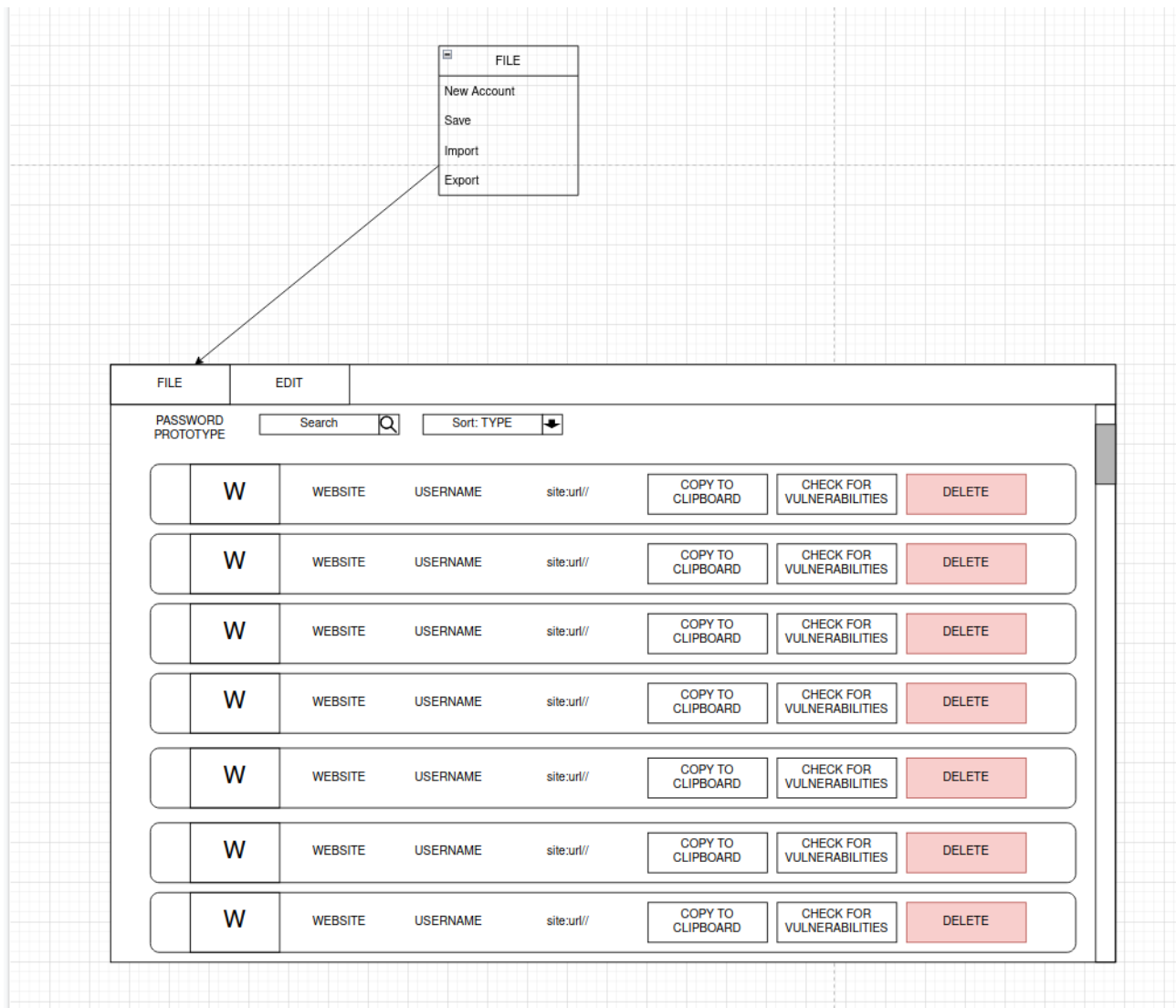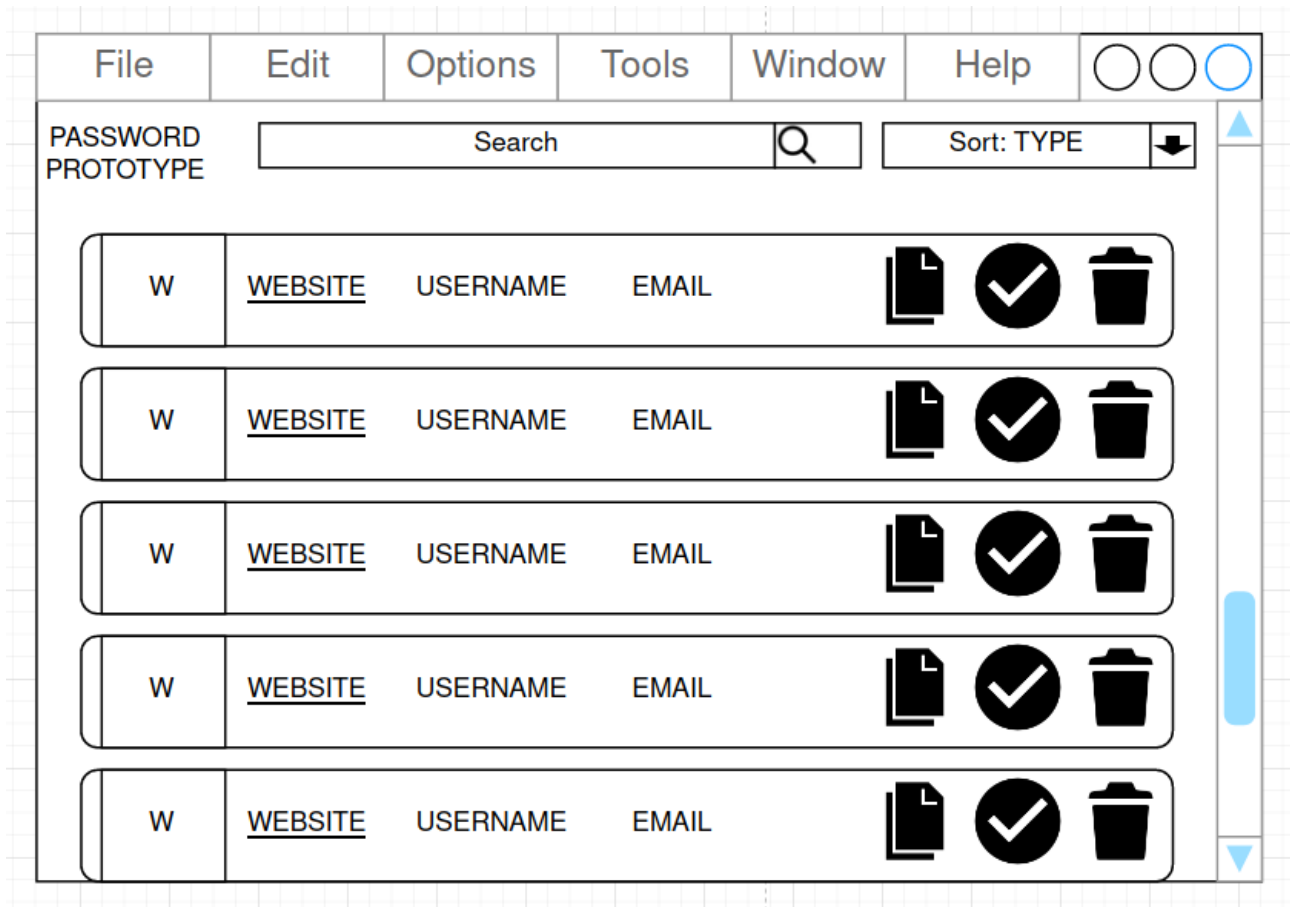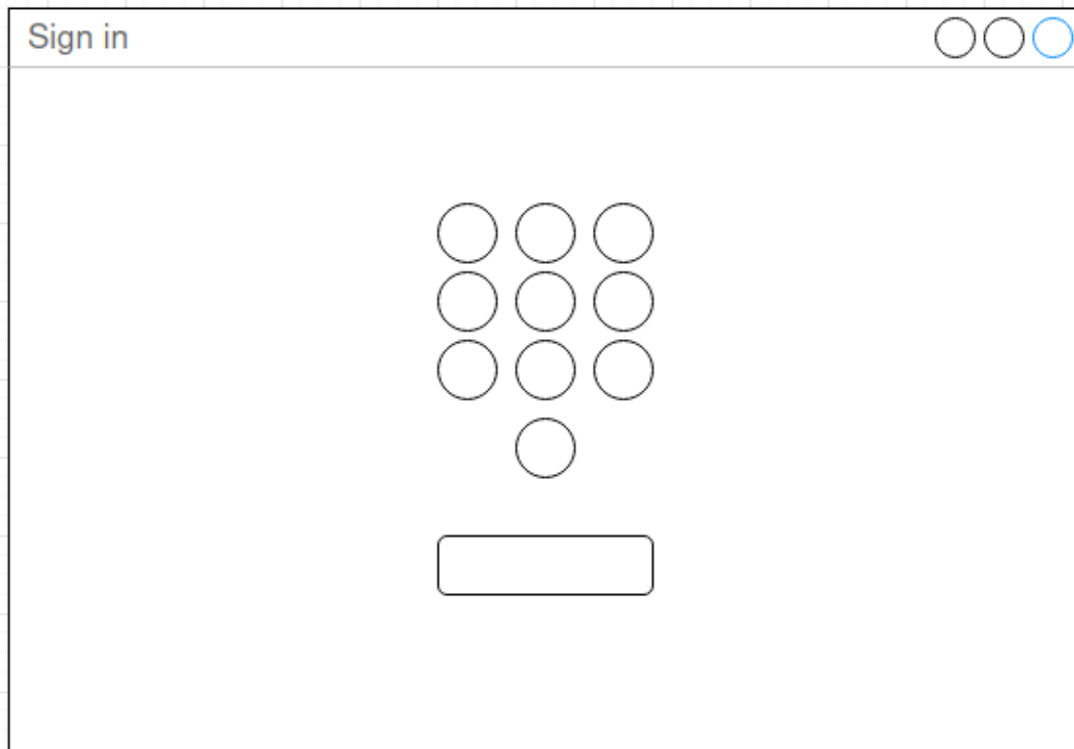During my prototyping stage, I utilised C++ and the Qt IDE. The initial reasoning was a want to gain experience in developing C++ software as well as cross-platform functionality. The program was developed on an Ubuntu Linux Workstation however the use of Qt will allow me to produce versions compatible with Windows and/or MacOS. I used various standard Qt libraries such as `QString`, `QDate`, `QException`, `QDebug` and `QFile`. While the program mostly consists of Qt C++ there is also a small amount of standard C++ spread throughout the code. As I previously discussed, I used an SQLite database in order to store user data. I planned to use the Qt Cryptographic Architecture in order to hash passwords and encrypt files but could not do so due to time constraints.

During development I utilised a standard object-oriented approach. Implementing classes as they are represented in the finalised design *[fig. 8]*. Specifically, when implementing my Hash table, I utilised a template class which allows me to use almost any value for my table key or stored data, these are commonly known as generic types. There are a handful of instances where I had to use overloading to create several different versions of a single function or attribute.

3.1. Data Structure

As previously mentioned, this program makes use of a Hash Table. This is used in the AccountDirectory class in order to store many user accounts. It was chosen for its speed as you can locate data by simply providing the key value. In this implementation, key clashes are solved by placing data in the next available space rather than chaining data together, this is known as Open Addressing. The key itself is generated via;

(*the day of account creation + the month of creation*) `MOD` *initial table size*

With 50 being the initial set size of the Hash table. These values were selected as none other would successfully create keys. I decided to utilise Open Addressing as it was a simpler solution and I felt that chaining account data together would needlessly complicate the AccountDirectory class and its underlying functionality.

3.2. Algorithms

The program makes use of an insertion sort which is performed on generated account ids. Each ID is stored in an array which is then passed to the sort function. The array is sorted and then passed to the constructor for a small interface that will then display the array.

An insertion sort works by splitting an array into a sorted area and unsorted area. Unsorted values are then placed in their sorted position within the sorted area. It is commonly likened to sorting a deck of cards with two piles.

An insertion sort begins by marking the first item as sorted and the remainder unsorted. Then the first item of the unsorted list is compared with each from the sorted list in a descending order. As

these comparisons occur, the items are swapped if that of the unsorted is found to be smaller than the current sorted item.

Otherwise, if the unsorted item is larger then it has found its sorted place. The algorithm will run until no more comparisons can be made.

3.3. Complexity Analysis

3.4. Testing and Metrics

Testing is one of the four stages of RAD, significant time has been allocated to carrying out the testing phase before moving on to finalisations in implementation. I will note that, during the early prototyping stage, I wrote a small collection of unit tests to check the functionality of important or core software components.

During testing, further issues were highlighted in that exceptions would throw but the action wasn't stopped, this can be observed in case PT2 and PT6. It was also made clear that the underlying process for adding new accounts requires several improvements as the user must restart the entire program to load new data from the database. Some test cases weren't considered due to the necessary components not being present in the software.

Plan

Case #	Description	Data	Expected Results
PT1	Create a new Account (Acceptable)	Username: markus101 Password: (generate) Email: markus101@gmail.com Date Created: (current_date) Date last used: (current_date)	Account is created
PT2	Create a new Account (Exceptional)	Username: ffgghhjjkkll Password: pass12 Email: ffgghhjjkkllzz Date Created: (blank) Date last used: (blank)	Account is not created. Exception thrown.
PT3	Delete Account (Acceptable)		Account is deleted from Database
PT4	Delete Account (Exceptional)		Account is not deleted from database. Exception thrown.
PT5	Edit Account (Acceptable)	Username: julia1131 Email: jules@hotmail.com	Account edited on Database
PT6	Edit Account (Exceptional)	Username: doesntexist10 Email:doesntexi(at)gmailemail.comom.eu	No edit committed. Exception Thrown.
PT7	Sort into Alphabetical order		Account listing sorted into Alphabetical order.
PT8	Sort into Most recently created		Account listing sorted by latest creation date
PT9	Sort into Earliest created		Account listing sorted by earliest creation date
PT10	Sort into Last used		Account listing sorted by latest last use date
PT11	Sign into online Account		Online account signed into on browser
PT12	Check for breaches on email	Email: jules@hotmail.com	Results opened on online breach website

Figure 13: Testing Plan

Results

Case #	Actual Results	Success/Failure?	Notes
PT1	Account successfully added message, Account present in table on restart	Success	Test went as expected, restarts required for new data to load
PT2	Account successfully added message, exception thrown as email was not valid, Account present in table on restart without email or date created	Failure	Exception thrown but Account was still created
PT3	Account 'test' successfully deleted	Success	
PT4	Account 'ffggghhjjkkll' not deleted	Success	
PT5	Account succesfully edited message, Account 'ffggghhjjkkll' edited with test data	Success	
PT6	Succesfully edited message, exception thrown for invalid email, edit was made to account	Failure	Clearly accounts can still be created/edited even if some fields are invalid. This should be looked into
PT12	Breach website successfully opened, pasting data displays results	Success	Case/Task is based purely on whether the user follows the message and pastes their email into the site.

Figure 14: Testing Plan

| markus101 | markus101... | 20/04/2022 | 20/04/2022 | EDIT | DELETE |
| test_account... | test@mail.c... | 20/04/2022 | 20/04/2022 | EDIT | DELETE |

Figure 15: Test case PT1
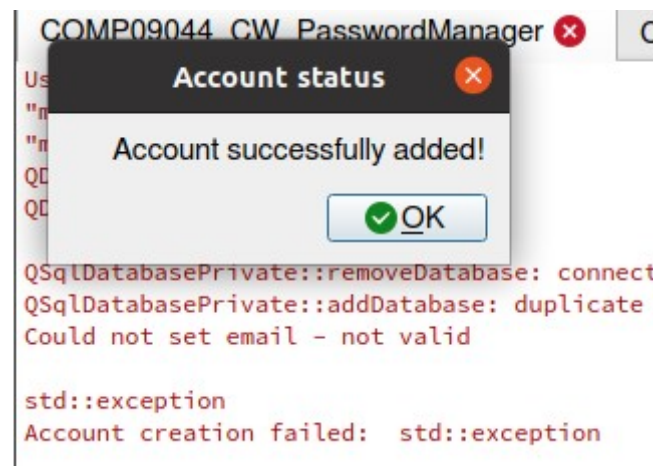
Figure 16: Test case PT2 Exception

| Username | Email | Date created | Last Used | |
| ffggghhjjkkll | | | 20/04/2022 | EDIT |

Figure 17: Test case PT2 result
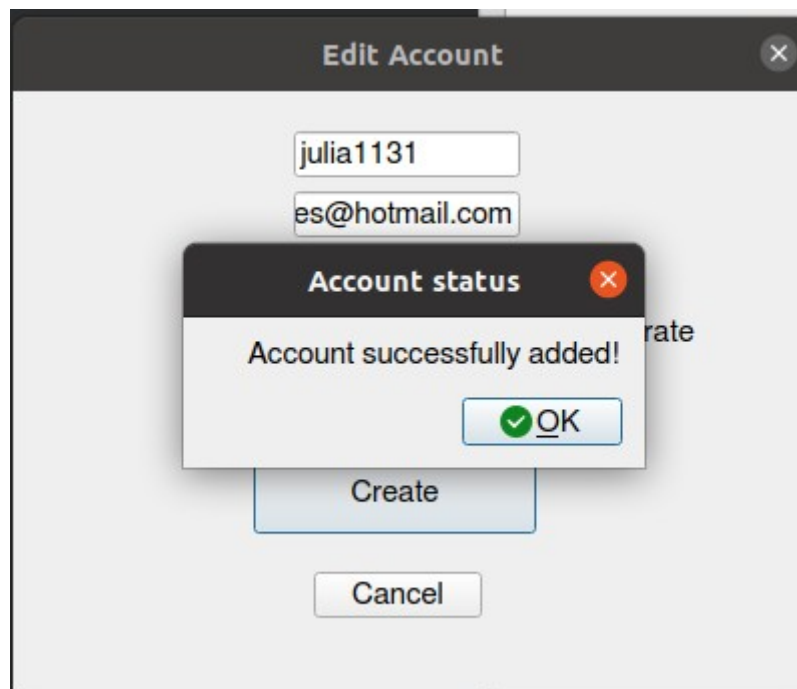
Figure 18: Test case PT5 results

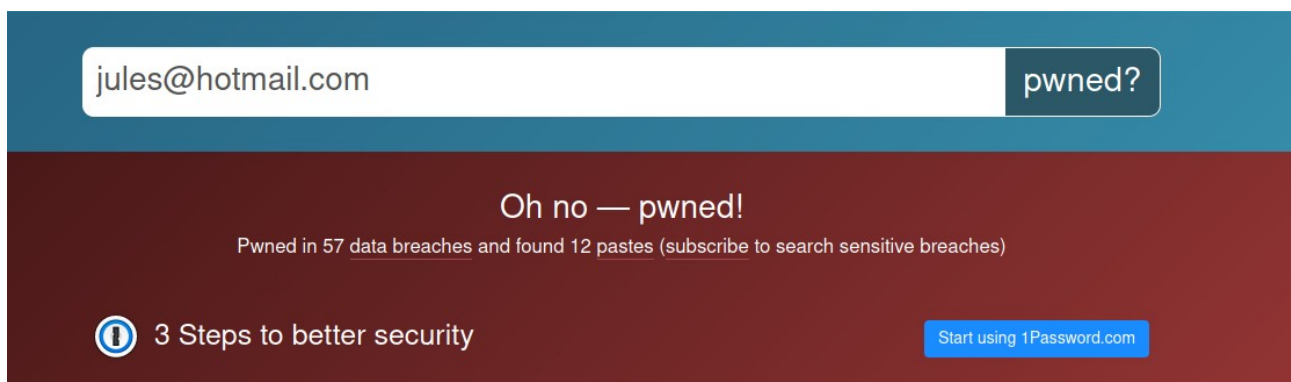*Figure 19: Test case PT6 results*

Figure 20: Test case PT12 results

4. Results

4.1. Verification & Validation

While I don't think I entirely built the product right, I do feel that I mostly built the right product. Throughout my code there are several alterations to classes which are not present in class diagrams or designs but were necessary additions. For this reason, the software doesn't entirely match its specification but that isn't without good reason. I do feel that regardless, the software continues to meet the given requirements and functions as was written in my earlier design brief. It should be noted that there are a handful of issues that persist – none of these issues cause any major problems with the software and can be regarded as minor defects. Some of these issues are a result of a changes implemented during development.
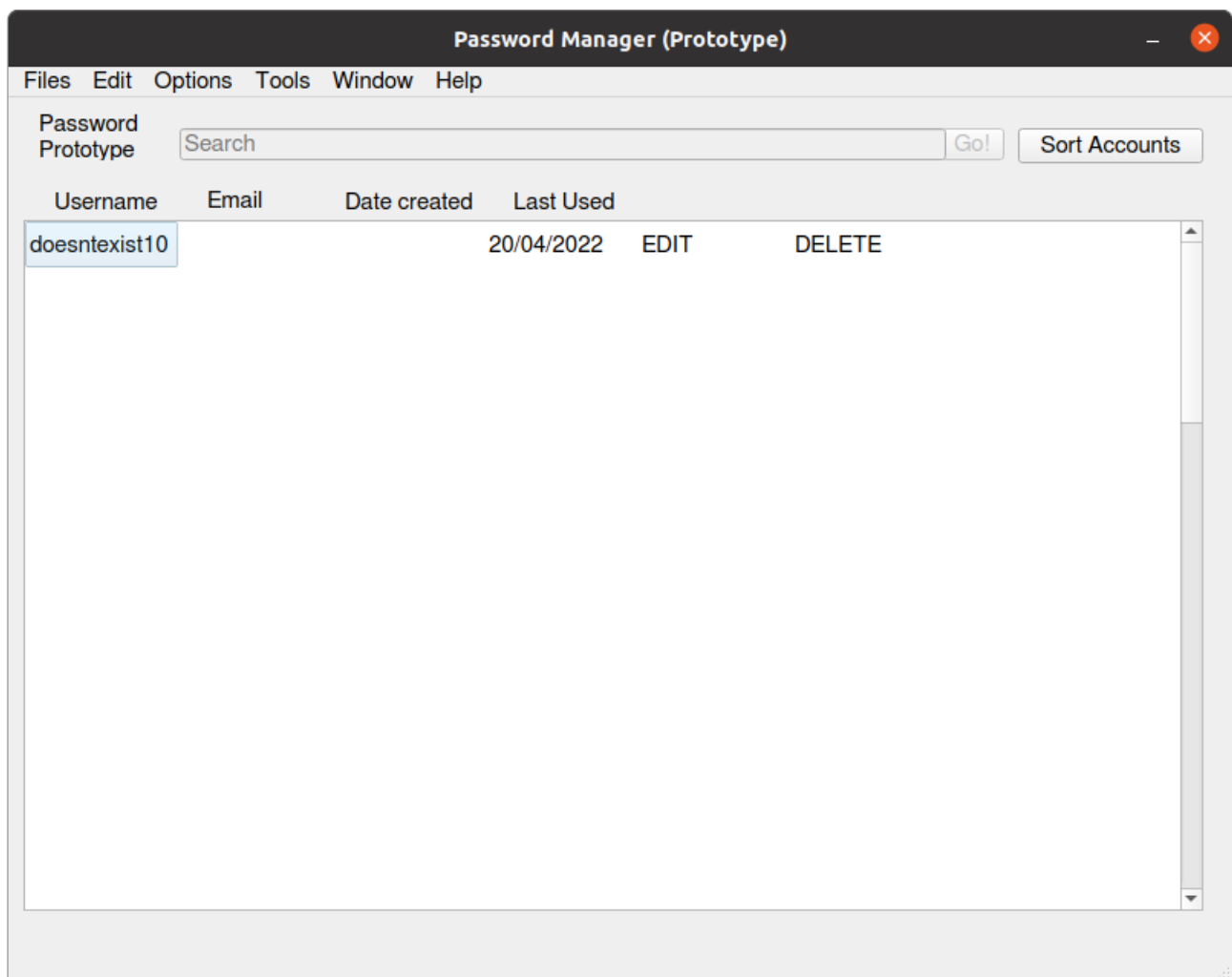

Figure 21: Main interface running on Ubuntu Linux

4.2. Reflecting on Methods

I feel that the RAD methodology did in fact prove beneficial to the development of my project as I was able to quickly develop a piece of software and did not need to revisit any steps. I will admit that the program is not exactly as was intended in my designs. This was caused by a degree of oversight on my part and limited time constraints which led me to focus only on the most vital aspects of the software. In reflection, for my next work, I'd like to allocate time to ensure I meet all requirements and do so properly as well as finding better, more efficient solutions. An example of an area which could benefit from this is the main account table. I feel that there is likely a better solution for its implementation which might reduce the amount of code and simplify its use.
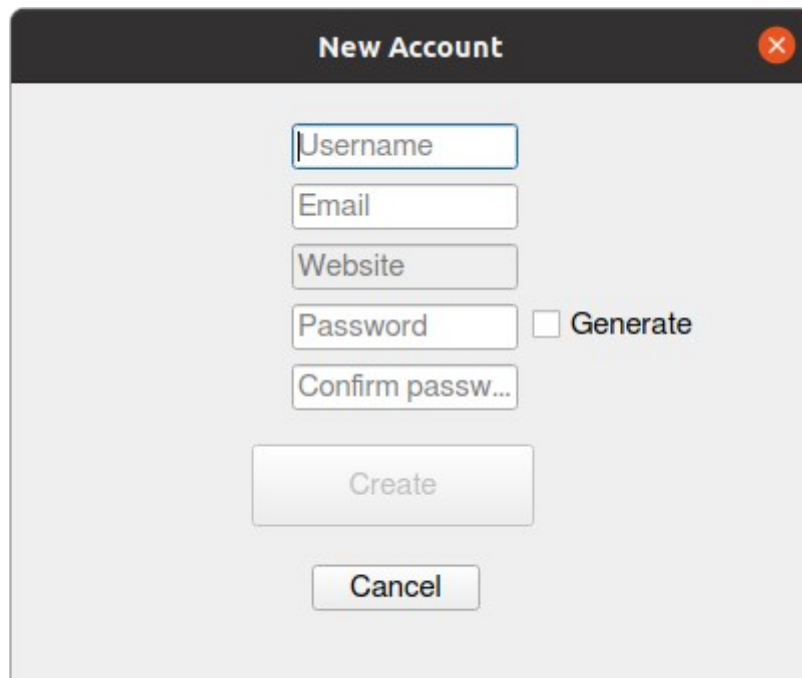

Figure 22: New account interface running on Ubuntu Linux

4.3. SWOT Analysis

Based on the content of this report and my evaluation of the software, I have created developed this SWOT analysis sheet.

Strengths	Weaknesses
Quick development via RAD methodology	Group of issues present
Base 'MUST' requirements met from MOSCOW analysis	Deviation from designs may not have been to softwares benefit
Better cross platform availability	Some oversights have also affected software outcome
Opportunities	**Threats**
Learn better methods for next project	Proposed protections for data missing
Solid base to extend and continue to develop program	Similar products from other vendors may provide better experience

Figure 23: SWOT Analysis

4.4. Ethics and Legalities

Due to time constraints, it was not possible to implement all proposed features in order to meet previous ethical considerations. However, as this is only a prototype and not intended for real-world distribution or use, it is not entirely detrimental. If this were a realistic project then I would be required to implement the missing protections. The implementation of password hashing within the SQLite database will ensure some degree of safety for user data. It can be assured that data is never transmitted to myself or a third party as previously considered.

Bibliography

cplusplus.com (2021) C++ Language: Function templates [Online] Available: https://www.cplusplus.com/doc/oldtutorial/templates/ [Accessed 2 March 2022]

Wikipedia (2022) Hash Table: Open Addressing [Online] Available: https://en.wikipedia.org/wiki/Hash_table#Open_addressing [Accessed 2 March 2022]

Wikipedia (2022) Object-oriented programming [Online] Available: https://en.wikipedia.org/wiki/Object-oriented_programming [Accessed 5 February 2022]

GeeksforGeeks (2021) Insertion Sort [Online] Available: https://www.geeksforgeeks.org/insertion-sort/ [Accessed 12 April 2022]