National College of Ireland

Bshc (Honours) in Computing

Cybersecurity

2022/2023

Luke Ferrie

19476134

X19476134@student.ncirl.ie

LockUp

Technical Report

Contents

Executive Summary

The key reasoning behind this report is to provide a full documentation of the project by discussing several key elements in detail. The sections discussed relate to the reasoning behind choosing this project, what the project aims to achieve as well as the technologies used. It enhances the readers knowledge of the project by discussing several use cases and provides in depth illustrations of each piece of functionality. Lastly it discusses four external requirements as well as displaying several screenshots from the application. (This section is purposely shortened to anticipate the full completion of the report after the Mid-Point Presentation).

1.0 Introduction

1.1. Background

Undertaking the project of building a Password Manager application in Java has many advantages and benefits. Password Manager applications are designed to help users keep their passwords secure and organized. They allow users to store all of their login information in one place, which can help prevent them from forgetting passwords or using the same password for multiple accounts. In this piece, I intend to discuss the advantages of having a Password Manager, the advantages of using Java, and some real-

world statistics backing up the need for a local password manager and my desire to build one.

One of the primary advantages of using a Password Manager is the increased security it provides. With a Password Manager, users can generate strong, unique passwords for each of their accounts. Strong passwords that are difficult to guess or crack can help protect users from identity theft, fraud, and other cyber threats.

Password Managers also make it easy to manage multiple accounts across multiple platforms. Rather than having to remember multiple passwords for various accounts, users can simply store all of their login information in one place. This can save time and help prevent frustration.

Java is a popular programming language that is widely used in the development of applications. One of the primary advantages of using Java is that it is platform-independent, meaning that it can run on any platform without the need for modification. This makes Java ideal for developing applications that need to run on multiple platforms.

Java is also known for its robust security features. The language is designed to be secure and has many built-in features that help prevent common security vulnerabilities, such as buffer overflows and race conditions. Additionally, Java has a robust ecosystem of libraries and frameworks that can help developers build secure, scalable applications.

According to a recent survey conducted by the Pew Research Center, 64% of Americans have experienced a major data breach. This includes instances where their personal information, such as passwords, has been stolen or compromised. This underscores the importance of using strong, unique passwords for each account and keeping them secure.

Using a Password Manager can help prevent data breaches by providing users with a secure and convenient way to manage their passwords. In fact, a recent study by Dashlane found that the average internet user has 90 online accounts. Trying to remember unique, strong passwords for each of these accounts can be challenging, and many people resort to using the same password for multiple accounts or using weak, easy-to-guess passwords. This can put their personal information at risk.

In conclusion, building a Password Manager application in Java is a worthwhile project that can provide many benefits to users. Password Managers can help increase security, save time, and prevent frustration. Java is an excellent choice for developing a Password Manager application due to its platform-independence, robust security features, and strong ecosystem. Real-world statistics back up the need for a local password manager, as many people have experienced data breaches and struggle to manage multiple accounts with unique, strong passwords.

1.2. Aims

The primary aim of this project is to provide users with a secure and reliable solution for managing their passwords and usernames without having to rely on third-party

websites. The current high-profile wave of mass data leaks highlights the importance of taking measures to protect one's confidential information online. With so many people relying on online services, it is essential to have a secure way to store and manage login credentials.

This application is designed to store users' passwords and usernames locally using file handling, providing them with a sense of security and control over their personal information. By keeping their login credentials stored on their own devices, users can reduce the risk of their information being exposed in a data breach. This is particularly important given the increasing number of companies experiencing data breaches that compromise the personal and financial information of their customers.

In summary, the aim of this project is to empower users to take control of their online security by providing them with a secure and reliable solution for managing their passwords and usernames locally. By doing so, users can reduce their reliance on third-party websites and have greater peace of mind knowing their personal information is secure.

1.3. Technology

To achieve the project goals, Java has been chosen as the primary programming language due to its versatility and ability to simplify the research process. After conducting thorough research, Java has been identified as the optimal choice for this project, as certain features required for the development of the application may not be available in other languages.

Java will be utilized for each component of the program, including the security aspects of file processing and writing to the user's device. The use of Java's GUI development platform will facilitate the creation of an intuitive and user-friendly application, while also enabling the implementation of robust back-end functionality.

For coding the application, the VS Code text editor has been selected due to its wide range of features, including syntax highlighting, embedded Git features, debugging tools, and code snippets. This editor has been chosen based on its ease of use and familiarity, enabling the development of high-quality code and efficient testing.

In summary, the project will utilize Java as the primary programming language, in combination with a powerful text editor, to facilitate the development of a secure and user-friendly password manager application.

1.4. Structure

The justification for my undertaking the project, the goals that my project seeks to realise, as well as the technology I will be employing and how I will use it in relation to building the project, have previously been covered. The system itself will be discussed in more detail in the paper, along with the user criteria that must be met before using the application. The functional requirements are a list of things that the system must perform in order to offer the service that is intended. There will be a number of use case illustrations showing this functionality in detail. We'll talk about the needs for data,

users, environments, and usability. The algorithms utilised throughout the programme will be mentioned in the system's design and architecture. The implementation of these algorithms will then be discussed, and a selection of screenshots of the most crucial screens the user sees will follow. Testing, including testing tools, plans, and testing results, will be thoroughly discussed. The report will conclude with a review of the project's benefits and drawbacks, as well as a quick discussion of what I would change about the application if I had more time to continue working on it. There will also be a list of references and any further appendices.

2.0 System

2.1. Requirements

The average user should be able to launch a desktop application as well as complete a registration and login form. Lastly, they should be able to make use of the given functionality within the application while being able to copy and paste text to and from the application.

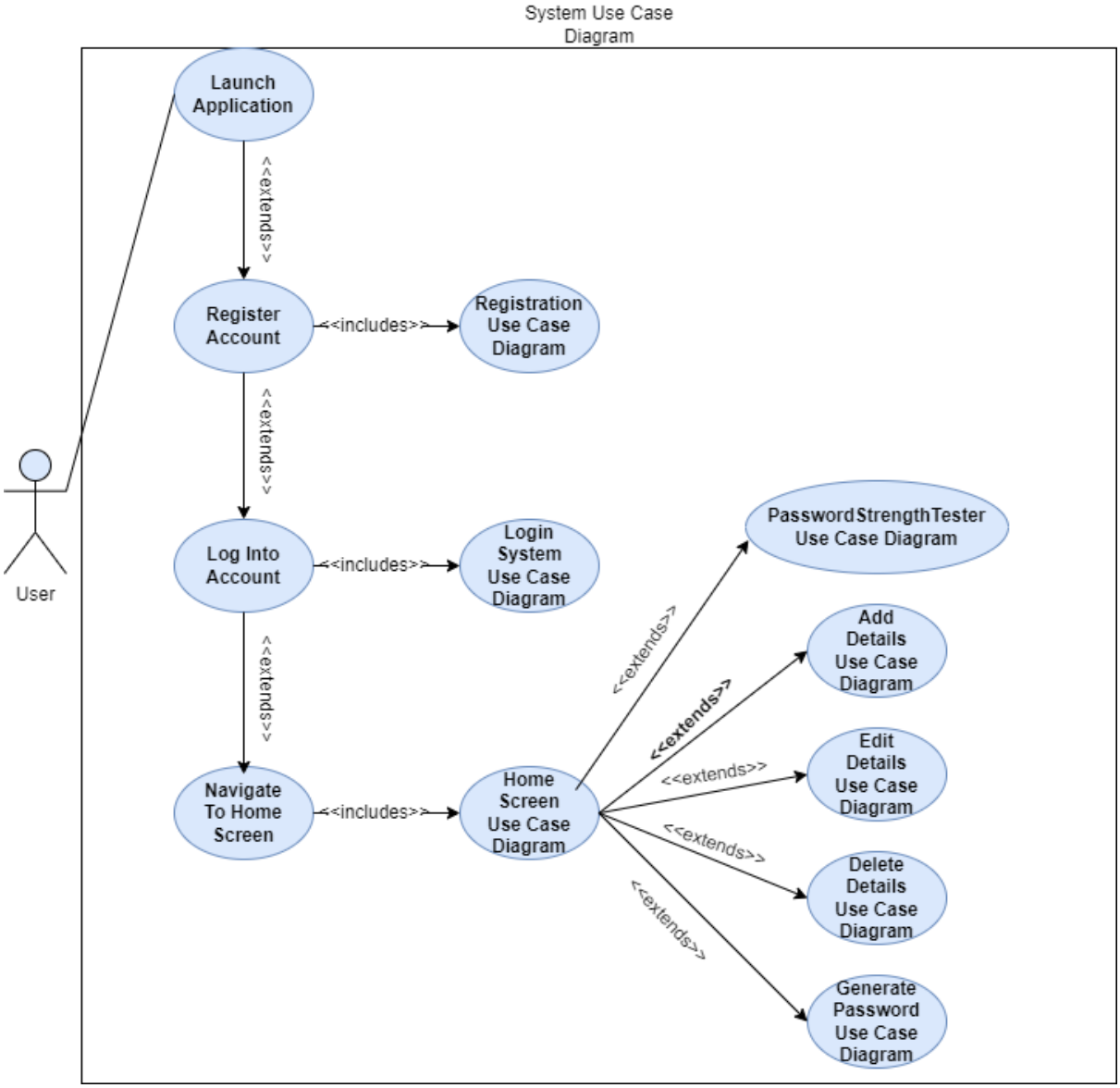
Providing the user can successfully do all mentioned above their experience should proceed to be error free and they can make full use of the application.

2.1.1. Functional Requirements

Functionality Rank 1: Registration System – Allows the user to register an account within the application by pressing the registration button and entering their master email and password as well as a pin.
Functionality Rank 2: Login System – This functionality becomes available after a user has registered an account; they can use this to login to the application.
Functionality Rank 3: Add Details – Once logging into the application and arriving at the home screen, a user must now enter some details into the appropriate text fields and add them thus making use of this functionality.
Functionality Rank 4: Edit Details – Having added some details the user can now edit any given set they wish by highlighting them and clicking the “Edit Details” button.
Functionality Rank 5: Delete Details – Now with some details added a user may wish to delete some old details or some they may not use anymore, by highlighting any given set of details and clicking the “Delete Details” button the details are removed from the application and purged locally.
Functionality Rank 6: Generate Password – If the user wishes to, they can click the “Generate Password” button and a randomly generated password will be displayed to them.
Functionality Rank 7: Password Strength Tester – In order to use the password strength tester, a user must first enter a password of their choosing and then click the test button. The result will then be presented, notifying the user of the password's level of strength and offering suggestions for how to make it stronger.
Functionality Rank 8: Home Screen – The home screen will act as a platform for functionality 3-7 housing all the buttons and the backend functionality.

2.1.1.1 System Use Case Diagram

Below will be a use case diagram displaying the functionality from the systems view.



2.1.2.1 Requirement 1: Account Registration

2.1.2.2 Description & Priority

Since the user cannot proceed with logging in and accessing the application itself without first creating an account, this feature can be regarded as the most crucial and having the highest priority. The use case illustrates the user's step-by-step registration process. First, the user must run the application from their desktop. Next, they must click the "Register and account" button. Finally, they must enter their personal information into the corresponding text fields. After doing so, they

are likely to click the "Submit Details" button, which, if their information meets the filtering requirements, will take them to the login page and use file handling to write their information to their local device.

2.1.2.3 Use Case 1.0 (Registration System)

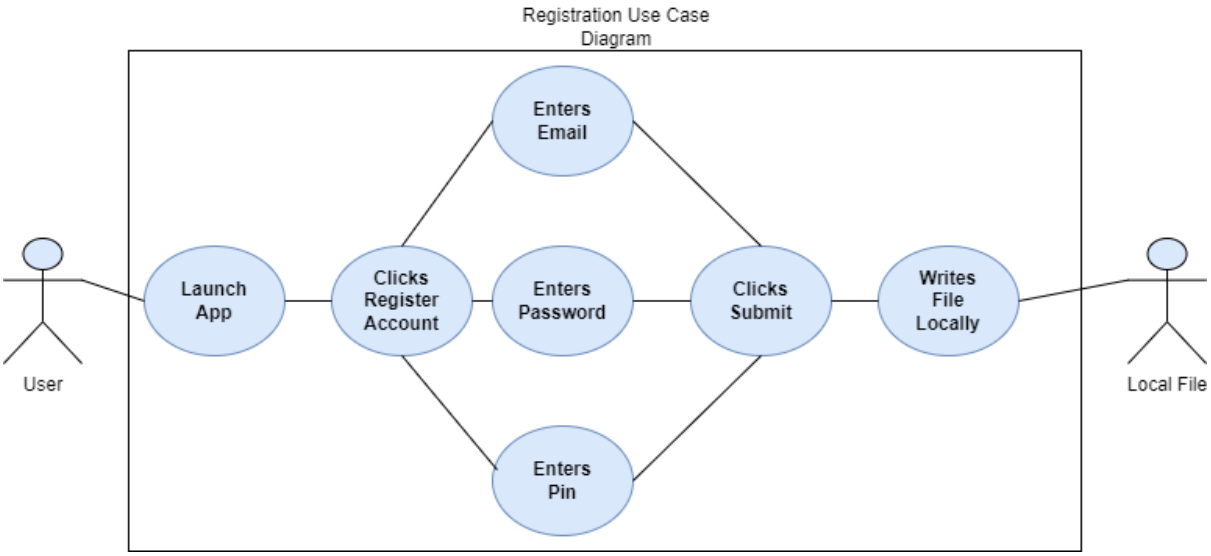
Scope

The scope of this use case allows the user to register an account for the application.

Description

This use case describes the process of registering an account by navigating to the registration screen, entering in personal details and submitting them.

Use Case Diagram



Flow Description

Precondition

The system has been launched and the user has navigated to the registration screen.

Activation

This use case starts when the user has reached the registration screen.

Main flow

1. The system identifies the user.
2. The user chooses to register and account and clicks the “Register Account” button.
3. The system redirects the user to the appropriate screen.
4. The user enters their details into the required text fields.
5. The users submit their details using the “Submit Details” button.
6. The users’ details are saved locally using file handling.

Alternate flow

A1: User navigates to incorrect screen

1. The system identifies the user.
2. The user realised they have navigated to the incorrect screen.

3. The user opts to click the “Home Screen” button and is return to the Home Screen.

Exceptional flow

E1: Invalid details entered

1. The system identifies the user.
2. The user has entered invalid details, being they inputted text which does not match the details requirements.
3. The user corrects their details.
4. The user clicks “Submit Details” button.
5. The users’ details are successfully saved locally.

Termination

The system redirects the user to the Login Screen upon successfully registering their account.

Post condition

The system goes into a wait state while the user decides to either log in or register another new account.

List further functional requirements here, using the same structure as for Requirement1.

2.1.3.1 Requirement 2: Account Login

2.1.3.2 Description & Priority

The user must log into the programme after creating an account and being taken to the login screen. After registering their details in the registration screen, they must enter them in the corresponding text boxes on the login screen. After entering these details, users are asked to click the "Login" button. This causes the details to be checked against those that have already been registered, and if a match is made, the user will be signed in and taken to the "Home Screen" of the application. As the second stage in using the application, this requirement comes after priority list registration.

2.1.3.3 Use Case 1.0 (Login System)

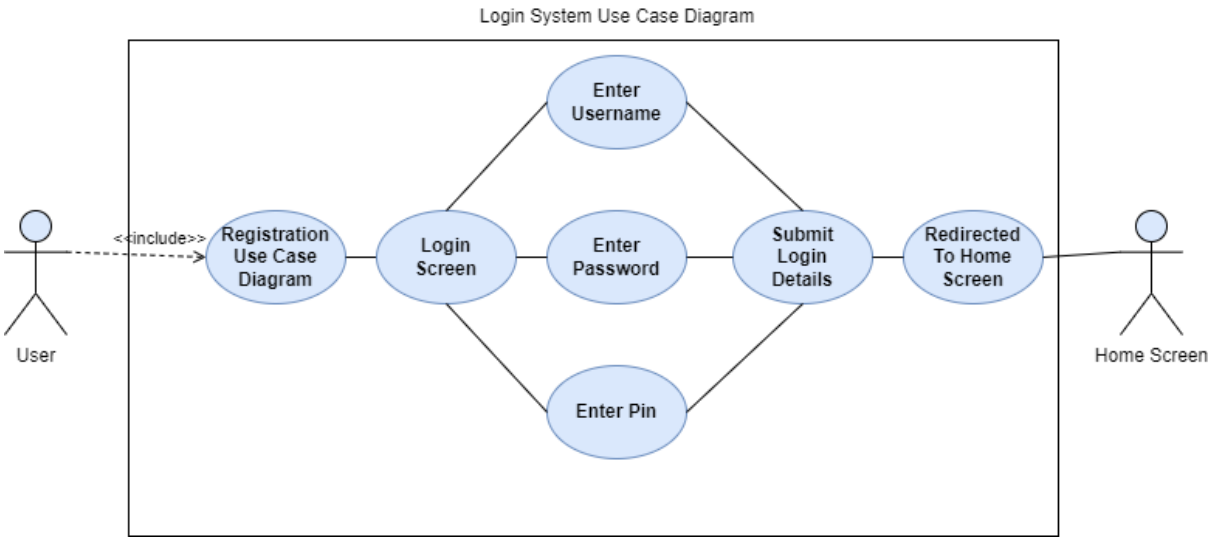
Scope

The scope of this use case allows the user to login to their account upon registering or if they already have an account to access the applications “Home Screen”.

Description

This use case illustrates the process of a user logging into an account. Having navigated to the login screen they, entering in personal details they have previously saved and clicking “Login”.

Use Case Diagram



Flow Description

Precondition

The system has been launched, the user has registered an account, they have then navigated to the login screen.

Activation

This use case starts when the user has reached the login screen with a registered account.

Main flow

1. The system identifies the user.
2. The user has arrived at the login screen.
3. The user enters their details into the required text fields.
4. The users submit their details using the “Login” button.
5. The user is successfully logged into the application and is redirects to the Home Screen.

Alternate flow

A1: User does not have an account

1. User clicks “Register Account” button.
2. User is redirected to the registration screen.
3. The user registers an account and is redirected back to the login screen.
4. The user now proceeds with the use case.

Exceptional flow

E1: Invalid details entered

1. The user enters details which are not recognized being either an invalid password, username or pin with those that are saved.
2. Error message is displayed.
3. User enters in corrected details.
4. User is successfully logged into the application and is redirected to the home screen.

E2: Details entered are not saved.

1. The user enters details which have not been saved before.

2. The user is prompted to register an account.
3. Upon registering the user is redirected to the login page once more.
4. The user logs in with their new details and is redirected to the Home Screen.

Termination

The system redirects the user to the Home Screen upon successfully logging into their account.

Post condition

The system goes into a wait state while the user decides what to do next.

2.1.4.1 Requirement 3: Adding Details

2.1.4.2 Description & Priority

The user will now want to enter a set of information, including the password, username, URL, and service, having successfully reached the Home Screen. They must then click the "Add Details" buttons to store and display the details in a text box after doing so.

2.1.4.3 Use Case 1.0 (Add Details)

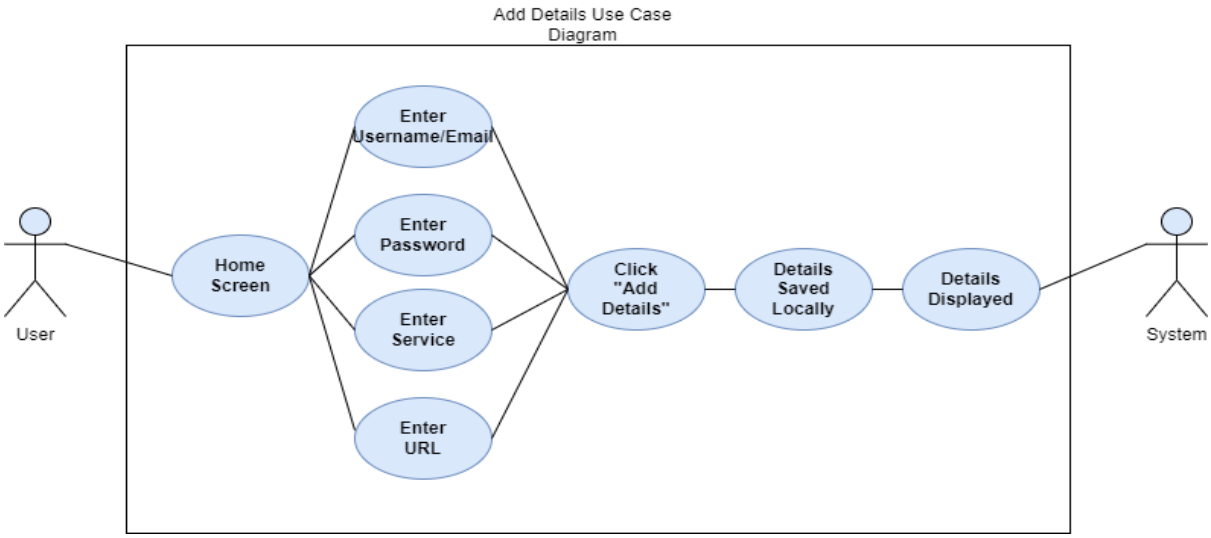
Scope

The scope of this use case allows the user to add a set of details and have them saved and displayed in the assigned text box.

Description

This use case illustrates the process of a user adding a set of details and saving them using the “Add Details” button functionality.

Use Case Diagram



Flow Description

Precondition

The system has been launched, the user has logged in and been redirected to the Home Screen. They have decided to add a set of details and have begun filling out the required text fields.

Activation

This use case starts when the user has reached the Home Screen.

Main flow

1. The system identifies the user.
2. The user has arrived at the Home Screen.
3. The user enters the details they want to save into the required text fields.
4. The users submit their details using the “Add Details” button.
5. The user’s details are written locally and saved.
6. The user’s details are displayed in the text box.

Alternate flow

A1: User decides not to add details

1. User doesn’t enter any details into the text fields.

Exceptional flow

E1: User enters invalid details

1. The user enters details which are not recognized as they don’t meet the requirements behind each text fields.
2. An error message is displayed prompting the user to correct their error.
3. The user corrects their error and clicks the “Add Details button”.
4. The details are successfully added, saved and displayed.

Termination

The system adds the users details and waits for the next input.

Post condition

The system goes into a wait state while the user decides what to do next.

2.1.5.1 Requirement 4: Editing Details

2.1.5.2 Description & Priority

Having successfully added some details to the password manager, the user now wishes to edit the details as they may have changed their username, email or password associated with any given service. The user simply highlights what details they wish to edit and clicks the edit details button. They are prompted to enter in the new set of details and click the “Add Details” button. The details are then updated.

2.1.5.3 Use Case 1.0 (Edit Details)

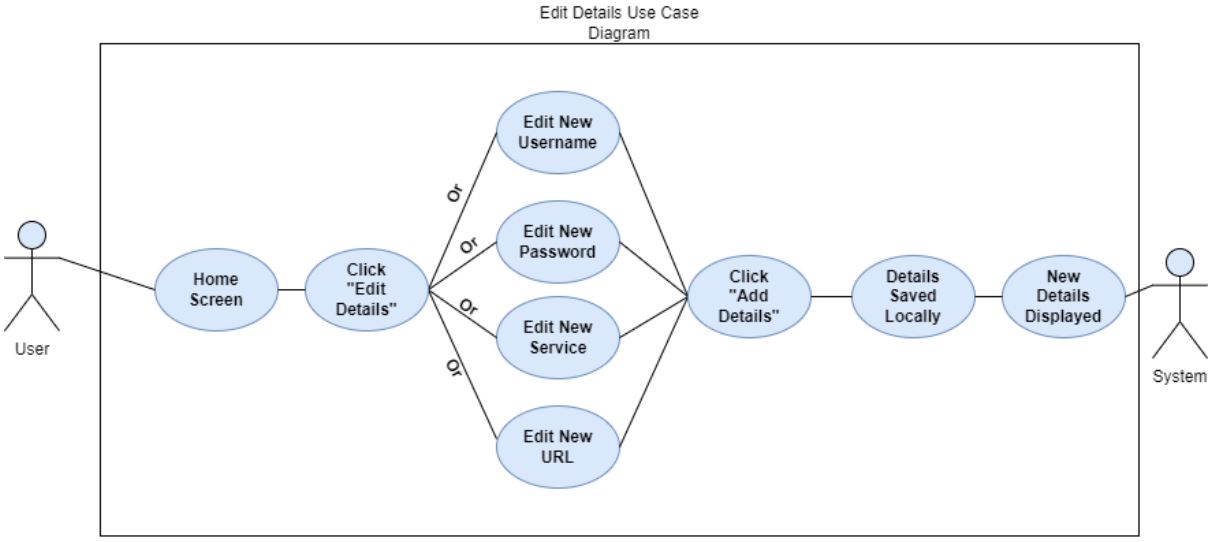
Scope

The scope of this use case allows the user to edit a set of details and have them saved and displayed in the assigned text box.

Description

This use case illustrates the process of a user editing a set of details and saving them using the “Edit Details” and “Add Details” button functionality

Use Case Diagram



Flow Description

Precondition

The system has been launched, the user has logged in and been redirected to the Home Screen. They have decided to a set of details and have begun filling out the required text fields.

Activation

This use case starts when the user has reached the Home Screen.

Main flow

1. The system identifies the user.
2. The user has arrived at the Home Screen.
3. The user clicks the “Edit Details” button
4. The appropriate text fields become available.
5. The user changes the details they wish to.
6. The user clicks the “Add Details” button.
7. The new details are saved and displayed.

Alternate flow

A1: User decides not to edit details:

1. User doesn’t enter any new details into the text fields.
2. The system enters a waiting state.

Exceptional flow

E1: User enters invalid details:

1. The user enters details which are not recognized as they don’t meet the requirements behind each text fields.
2. An error message is displayed prompting the user to correct their error.
3. The user corrects their error and clicks the “Add Details button”.
4. The details are successfully added, saved and displayed.

Termination

The system edits and adds the users details and waits for the next input.

Post condition

The system goes into a wait state while the user decides what to do next.

2.1.6.1 Requirement 5: Deleting Details

2.1.6.2 Description & Priority

Having successfully added some details to the password manager, the user now wishes to delete the details as they may no longer have a use for their username, email or password associated with any given service. The user simply highlights what details they wish to delete and clicks the “Delete Details” button. The details are then removed from the application and are purged from the local save file. The details which remain are then each moved upwards in the display box.

2.1.6.3 Use Case 1.0 (Delete Details)

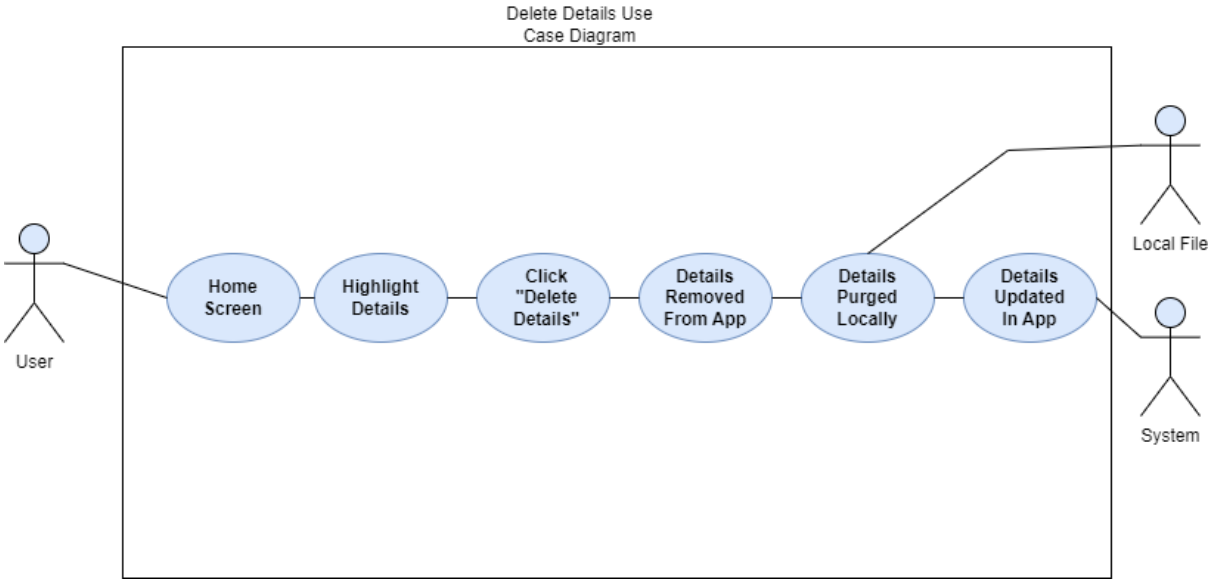
Scope

The scope of this use case allows the user to delete a set of details and have the text box update with the remaining details.

Description

This use case illustrates the process of a user deleting a set of details and making use of the “Delete Details” button functionality

Use Case Diagram



Flow Description

Precondition

The system has been launched; the user has previously saved some details in the application.

Activation

This use case starts when the user has reached the Home Screen and has added some details previously.

Main flow

1. The system identifies the user.
2. The user has arrived at the Home Screen.
3. The user highlights previously added details.
4. The user clicks the “Delete Details” button.
5. The details are removed from the application.

6. The details are purged locally.
7. The current standing details are updated in the text box.

Alternate flow

A1 : User decides not to delete details:

1. The user highlights a set of details, prior to deleting them they realise they wish to keep them and decide not to delete them.

Exceptional Flow

In the case of this use case diagram there is no exceptional flow

Termination

The system deletes the users details and waits for the next input.

Post condition

The system goes into a wait state while the user decides what to do next.

2.1.7.1 Requirement 6: Generating Password

2.1.7.2 Description & Priority

While on the home screen the user can press the “Generate Password” button. This will generate a random password based on Googles “How to create a strong password” algorithm. Upon pressing the button, a password will be generated and displayed to the user with the option to press a button to copy the password to their clipboard. They can use this password externally when signing up any service or if they wish to add it to the password manager first then sign up to their chosen service this also works.

2.1.7.3 Use Case 1.0 (Generate Password)

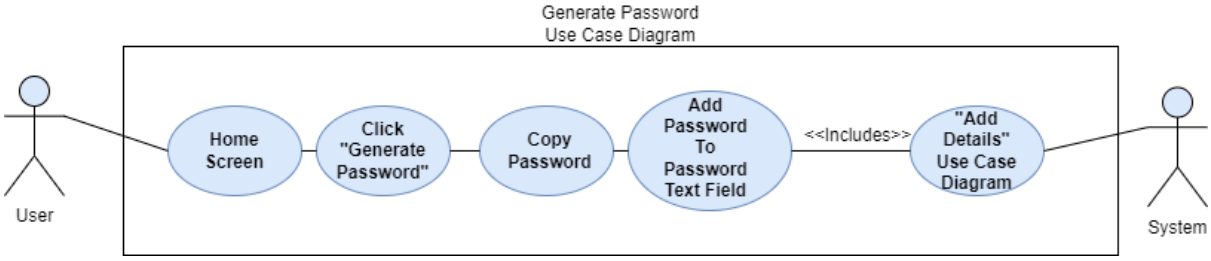
Scope

The scope of this use case allows the user to generate a password in line with Googles “How to create a strong password” algorithm.

Description

This use case illustrates the process of a user generating a password using the “Generate Password” button functionality

Use Case Diagram



Flow Description

Precondition

The system has been launched; the user has located the home screen.

Activation

This use case starts when the user has reached the Home Screen and has clicked the “Generate Password” button.

Main flow

1. The system identifies the user.
2. The user has arrived at the Home Screen.
3. The user clicks the “Generate Password” button.
4. A new password is generated and displayed to the user.

Alternate flow

A1: User decides to copy the password and add it to an external site.

1. The user generates a password.
2. The user clicks the copy icon.
3. The password is copied to their clipboard.
4. They add this password to the registration form of their given service.

A2: User decides not to use generated password and wishes to generate a new one.

1. The user generates a password.
2. The user chooses not to use this password.
3. The user generates a new password.
4. The user makes use of this password.

Exceptional Flow

In the case of this use case diagram there is no exceptional flow.

Termination

The system presents the user with their newly generated password.

Post condition

The system goes into a wait state while the user decides what to do next.

2.1.8.1 Requirement 7: Password Strength Tester

2.1.8.2 Description & Priority

A user must first enter a password of their choice and then click the test button in order to utilise the password strength tester. The outcome will then be displayed, informing the user of the strength of the password and providing advice on how to make it more robust.

2.1.8.3 Use Case 1.0 (Password Strength Tester)

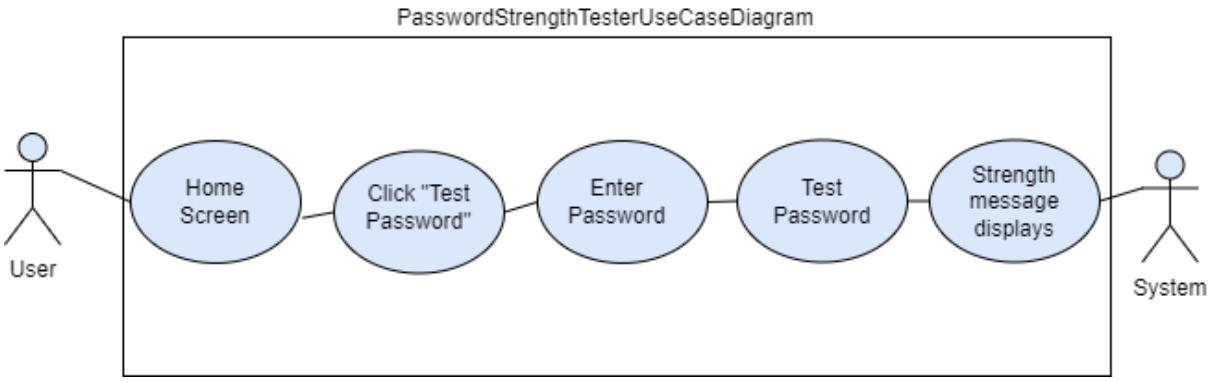
Scope

The scope of this use case allows the user to make use of the password strength tester feature.

Description

This use case illustrates the process of a user navigating the password strength tester.

Use Case Diagram



Flow Description

Precondition

The system has been launched; the user has located the home screen.

Activation

This use case starts when the user has clicked the “Password Strength Tester” button.

Main flow

1. The user has clicked the “Password Strength Tester” button.
2. The user enters their given password they wish to test.
3. The user clicks the “Test” button.
4. The strength tester returns either a message informing the user their password is already strong, or it will return a message letting them know it is a weak password along with some ways to increase its strength.

Alternate flow

In the case of this use case diagram there is no alternate flow.

Exceptional Flow

In the case of this use case diagram there is no exceptional flow.

Termination

The system presents the user with either a positive or negative message.

Post condition

The system goes into a wait state while the user decides what to do next.

2.1.9.1 Requirement 8: Home Screen

2.1.9.2 Description & Priority

Upon arriving at the home screen, the user can choose to make use of any of the given functionality. The home screen acts as a platform for the functionality which has been previously discussed therefore has a lessened functionality rank.

2.1.9.3 Use Case 1.0 (Home Screen)

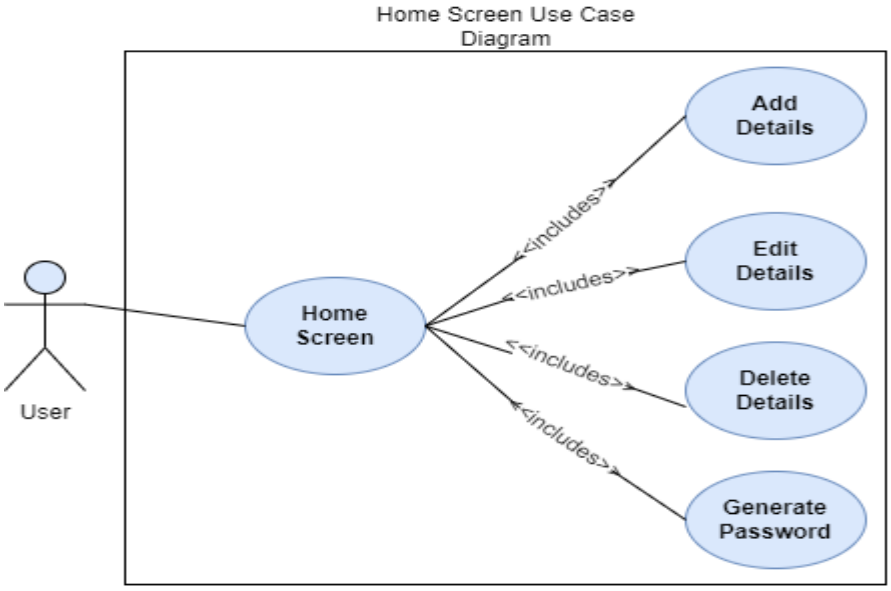
Scope

The scope of this use case allows the user to navigate the home screen.

Description

This use case illustrates the process of a user navigating the home screen and making use of any of the given functionality

Use Case Diagram



Flow Description

Precondition

The system has been launched; the user has located the home screen.

Activation

This use case starts when the user has reached the Home Screen.

Main flow

1. The system identifies the user.
2. The user has arrived at the Home Screen
3. The user clicks any of the given functionality.
4. The appropriate functionality launches.

Alternate flow

In the case of this use case diagram there is no alternate flow.

Exceptional Flow

In the case of this use case diagram there is no exceptional flow.

Termination

The system presents the user with their newly generated password.

Post condition

The system goes into a wait state while the user decides what to do next.

2.1.3 Data Requirements

The programme will employ file handling to save and preserve the users' login information as well as any information they supply to the programme.

2.1.4 User Requirements

The user should anticipate receiving a straightforward and user-friendly password management solution from this application. They should have little trouble using the functionality and picking up on the application quite fast. The only requirements are that the user log in each time and follow the instructions on the registration form.

2.1.5 Environmental Requirements

Other than suitable file handling software installed locally on each device, there are no environmental requirements. The application won't rely on internet databases or servers and will just use the device's maximum average CPU capacity.

2.1.6 Usability Requirements

I want to make sure that everyone who wants to try out and maybe utilise this programme in their daily life can do so. The application is intended for users who have many accounts, some who, like myself, don't trust internet storage, or users who simply want to boost online security. This will be accomplished through its functionality and user-friendly design, which will make it simple to use and understand. All features will have clear labels and be created to enhance user experience. Performance problems should only be those related to the CPU and processing power of the user's device. Utilizing frequent unit testing should guarantee that error management is an easy operation.

2.2 Design & Architecture

The design varies depending on the screen the user is viewing, however the general idea in relation to the design of the application is such that there are several text fields for the user to enter their login details as well as a button to navigate to the registration screen in the event the user does not have an account. Upon logging in, there are several buttons associated with the functionality previously mentioned as well as a text box displaying all saved details.

The password manager application is designed with a modular architecture that separates the functionality into several classes. Each of these classes is designed to handle specific functionality of the application.

First off, the code for the login form and all of its related functionality is handled by the "Login.java" file. This class contains the code necessary to validate the user's login information and grant them access to the application's primary features.

Second, the signup form and its associated functions are handled by the "signup.java" class. It contains the code needed to validate user inputs, save new user information, and enable access to the application's core functions following registration.

Thirdly, the "passwordstrength.java" file contains all the code for the functioning of the password strength test feature. This class is in charge of creating the user's random passwords and evaluating the security of the passwords they input.

The "mainscreen.java" file is responsible for containing all of the code for the home screen's features. The code for adding, modifying, and removing login information and passwords is contained in this class. It also grants access to additional services like password generating and password strength testing.

The "main.java" file brings everything together to create a working application. The user interface for the application is provided by this class, which also initialises all the other classes. It regulates the data flow between various components of the application.

The system architecture of the password manager application is made more organised and manageable by categorising the functionality into these classes. Future updates and modifications are made simpler as a result.

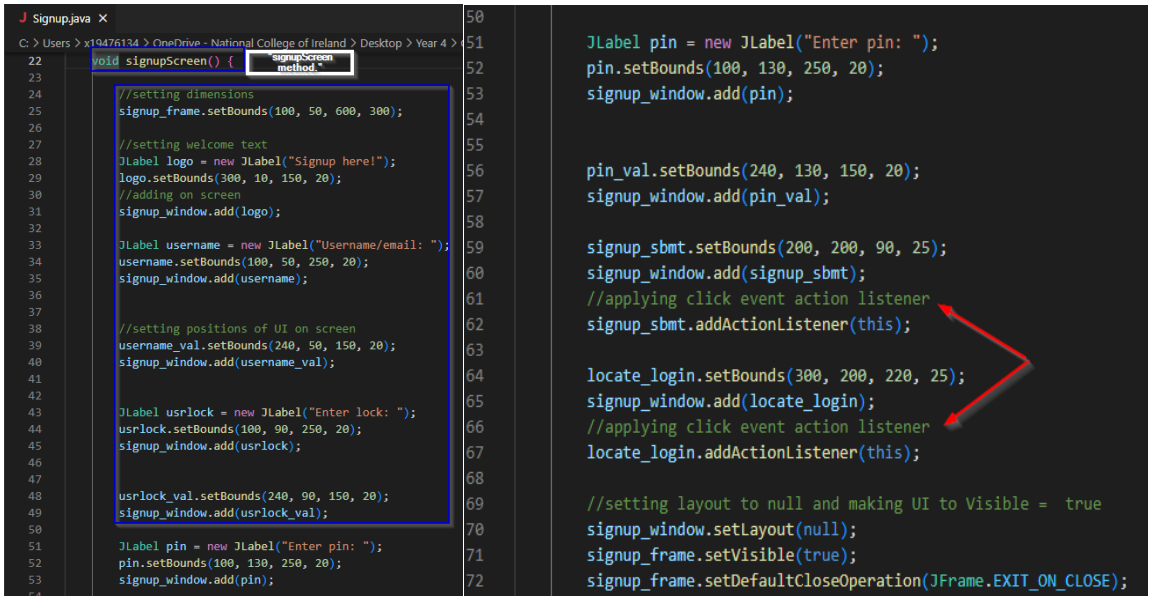
2.3 Implementation

We will go through each class and highlight the essential functionality as well as offer some code snippets in this section. We will also explain the important algorithms, classes, and functions that have been used in the code.

Since the "signup.java" file technically represents the first piece of functionality a new user would need to access, we'll start there.

Signup is a class that implements the ActionListener interface. This means it has a method called actionPerformed() that is called when an event occurs (such as a button click).

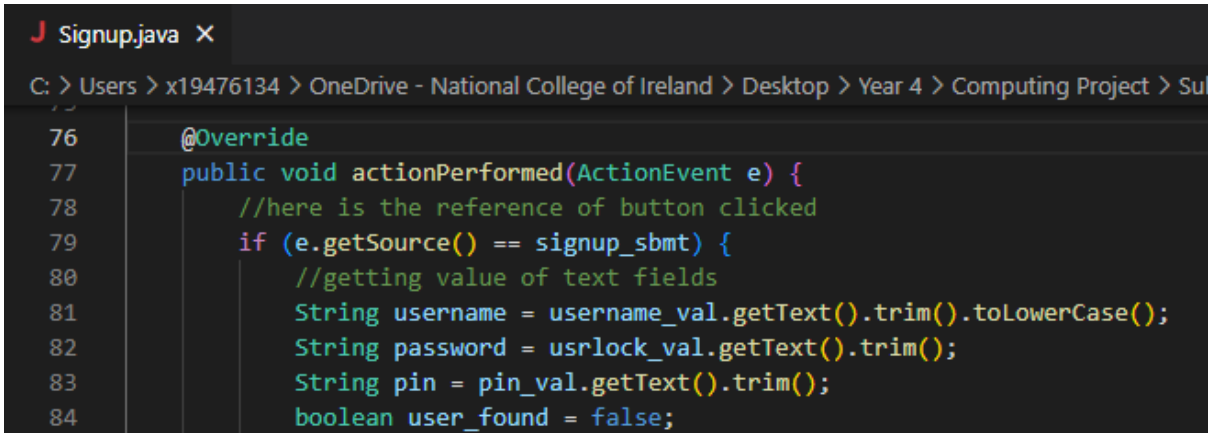
The Signup class has a method called signupScreen() that creates a window (using JFrame and Container objects) with several UI elements such as text fields, buttons, and labels. Below you can see a snippet of some code setting the dimensions and the welcome text the user sees when they open the signup form.



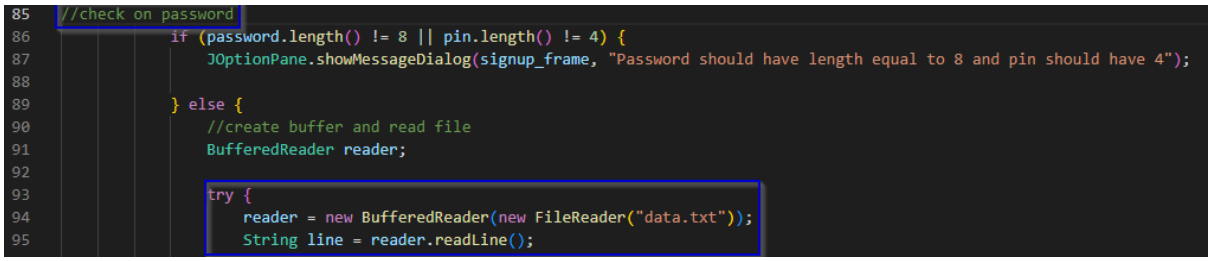
The actionPerformed() method is where most of the logic takes place. It first checks if the "Signup" button was clicked, and if so, it retrieves the values of the text fields for the username, password, and PIN. It then checks if the password and PIN meet certain criteria (length of 8 and 4, respectively). If they do not meet the criteria, an error message is displayed. If they do meet the criteria, the data.txt file is read to see if the user already exists. If the user exists, an error message is displayed. If the user does not

exist, their information is written to the data.txt file, and a success message is displayed. The Signup window is then closed and the Login window is opened.

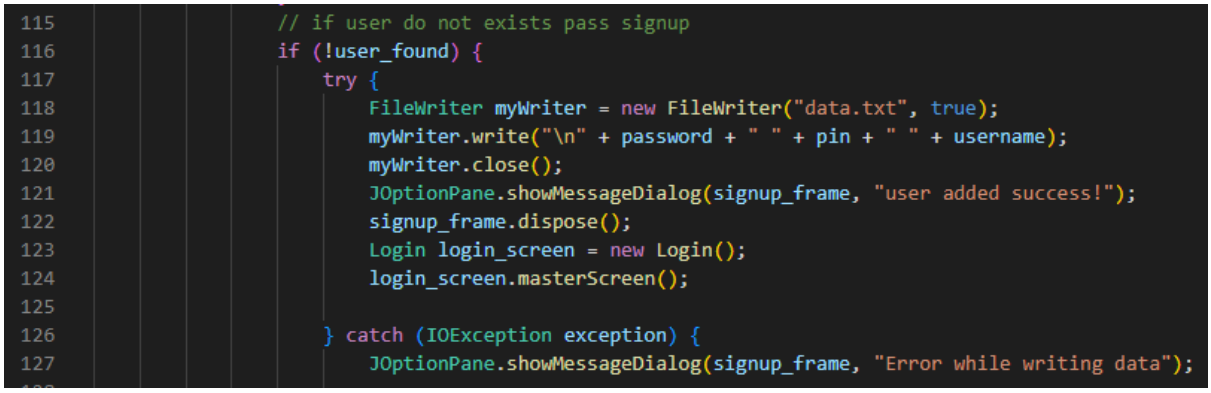
actionPerformed() method snippet:



The reader = new BufferedReader(new FileReader("data.txt")); - is another interesting piece of code, this line creates a BufferedReader object to read from the data.txt file. This method is used when checking the users password meets the criteria set.



Onto the code myWriter.write("\n" + password + " " + pin + " " + username); - This line writes the user's information (password, PIN, and username) to the data.txt file, separated by spaces. The \n character adds a new line before the new user's information.



The final snippet id like to highlight from this class is as follows **login_screen.masterScreen();** and **login.masterScreen();** - These lines create a new

Login object and call its **masterScreen()** method, which opens the login window and destroys the signup form once the user has registered.

```
133         } else if (e.getSource() == locate_login) {
134             //destroying signup frame loading login frame
135             signup_frame.dispose();
136             Login login = new Login();
137             login.masterScreen();
138         }
```

Now we'll begin discussing the "login.java" file. This file defines the Login class which like the "signup.java" file implements the ActionListener interface. It provides the GUI for the login screen and checks the user's credentials by reading them from the file handling file named "data.txt". For this file we'll break up, list and briefly describe each class and function.

Classes:

1) JFrame class: This class is used to create a window frame that can be used to display the GUI components.
2) Container class: This class is used to get access to the screen by creating a container for the GUI components.
3) JTextField class: This class is used to create text fields for the user to enter their username, pin, and lock.
4) JLabel class: This class is used to create labels for the text fields.
5) JButton class: This class is used to create buttons for the user to click.
6) BufferedReader class: This class is used to read data from a file.
7) MainScreen class: This class is used to load the main screen after the user has logged in successfully.

Functions:

1) readLine() function: This function is used to read a line of text from the file.
2) substring() function: This function is used to extract a substring from a string.
3) equals() function: This function is used to compare two strings for equality.
4) JOptionPane.showMessageDialog() function: This function is used to display a message dialog box.
5) setLayout(null) function: This function is used to set the layout of the GUI to null so that the components can be positioned using absolute coordinates.
6) addActionListener() function: This function is used to attach an ActionListener to the buttons.
7) dispose() function: This function is used to destroy the current window.

Now we can view and discuss some code snippets from this file.

1) "master_sbmt.addActionListener(this)": This code snippet attaches an ActionListener to the "unlock" button. When the button is clicked, the actionPerformed() function is called.

```
66          master_sbmt.setBounds(200, 200, 90, 25);
67          login_window.add(master_sbmt);
68          //applying on click action listener on the button
69          master_sbmt.addActionListener(this);
```

2) “if (e.getSource() == locate_signup)”: This code snippet checks if the "No account? signup" button is clicked. If yes, the current window is destroyed, and the Signup screen is loaded.

```
86      @Override
87      public void actionPerformed(ActionEvent e) {
88          //here e is the reference of button clicked
89          if (e.getSource() == locate_signup) {
90              //destroying current screen and creating object of Signup class
91              login_frame.dispose();
92              Signup signup_screen = new Signup();
93              signup_screen.signupScreen();
94          } else if (e.getSource() == master_sbmt) {
95              //getting data from text fields
96              String username = username_val.getText().trim().toLowerCase();
97              String password = usrlock_val.getText().trim();
98              String pin = pin_val.getText().trim();
99              boolean user_found = false;
```

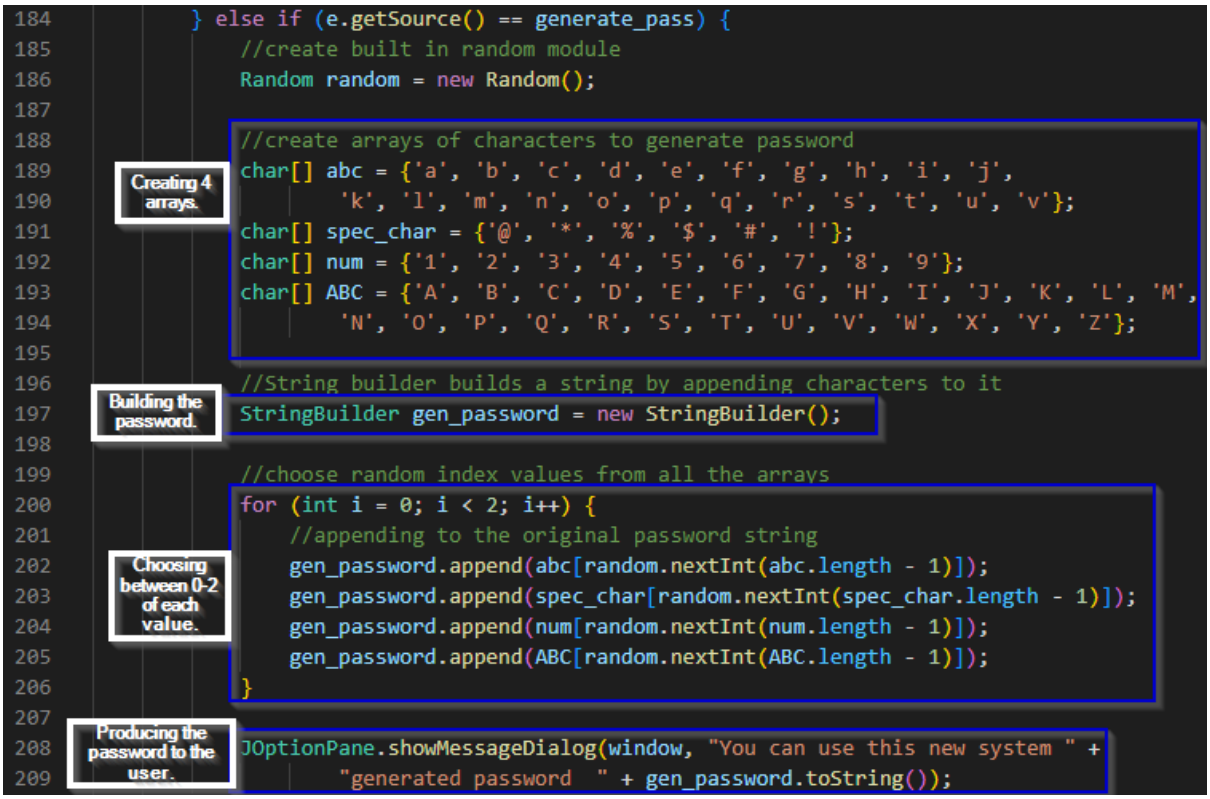
3) “if (line.substring(0, 8).equals(password) && line.substring(9, 13).equals(pin) && line.substring(14).equals(username))”: This code snippet checks if the username, pin, and lock entered by the user match the credentials stored in the file. If yes, the MainScreen is loaded. If no, an error message is displayed.

```
112             //iterating through the file
113             while (line != null) {
114
115                 if (line.length() > 0)
116                     if (line.substring(0, 8).equals(password) && line.substring(9, 13).equals(pin) && line.substring(14).equals(username)
117
118                         user_found = true;
119                         break;
120                     }
121                 // read next line
122                 line = reader.readLine();
123             }
124
125             reader.close();
126         } catch (IOException readException) {
127             JOptionPane.showMessageDialog(login_frame, "Error while reading");
128         }
129         if (user_found) {
130             //destroying current window and initiating Main screen on login
131             login_frame.dispose();
132             MainScreen mainScreen = new MainScreen();
133             mainScreen.loadMainScreen();
134         } else {
135             JOptionPane.showMessageDialog(login_frame, "User not found");
136         }
```

Onto the “MainScreen.java” file, this program uses the swing and action listener libraries for the graphical user interface. The MainScreen class implements the ActionListener interface to handle the actions of the buttons. The loadMainScreen

method sets the dimensions of the main window, adds the UI elements to the window, and reads the data from the "services.txt" file to display it in the text area. The actionPerformed method handles the actions of the buttons, such as adding a new service, editing a service, deleting a service, generating a password, checking password strength, logging out, or exiting the program.

We can take a look at the code snippet below taken from this file.




```
184         } else if (e.getSource() == generate_pass) {
185             //create built in random module
186             Random random = new Random();
187
188             //create arrays of characters to generate password
189             char[] abc = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
190                     'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v'};
191             char[] spec_char = {'@', '*', '%', '$', '#', '!'};
192             char[] num = {'1', '2', '3', '4', '5', '6', '7', '8', '9'};
193             char[] ABC = {'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M',
194                     'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z'};
195
196             //String builder builds a string by appending characters to it
197             StringBuilder gen_password = new StringBuilder();
198
199             //choose random index values from all the arrays
200             for (int i = 0; i < 2; i++) {
201                 //appending to the original password string
202                 gen_password.append(abc[random.nextInt(abc.length - 1)]);
203                 gen_password.append(spec_char[random.nextInt(spec_char.length - 1)]);
204                 gen_password.append(num[random.nextInt(num.length - 1)]);
205                 gen_password.append(ABC[random.nextInt(ABC.length - 1)]);
206             }
207
208             JOptionPane.showMessageDialog(window, "You can use this new system " +
209                     "generated password  " + gen_password.toString());
```

In this code snippet we can seen where we generate a random password by creating arrays of characters that will be used to generate the password. The arrays are defined as follows:

1) abc: an array containing lowercase letters from a to v.
2) spec_char: an array containing some special characters, such as @, *, %, $, #, and !.
3) num: an array containing numbers from 1 to 9.
4) ABC: an array containing uppercase letters from A to Z.

A loop is then used to generate the password. It runs twice since we want a password that is eight characters long (two characters from each of the four arrays). Inside the loop, the append() method of the StringBuilder class is used to add random characters from each of the four arrays to the password string. The nextInt() method of the Random class is used to generate a random index within the length of each array, and the character at that index is appended to the password string. Finally, a message dialog box is displayed using the

JOptionPane.showMessageDialog() method, which shows the generated password to the user.

The "passwordstength.java" file is the last one; as implied by the name, it contains the code for the password strength tester. This code creates the ActionListener interface-implementing class PasswordStrength. When an action, such as a button click, takes place, the ActionListener interface's method actionPerformed is called. The main window for the password strength checker, a JFrame named strength_check_frame, two JButton objects, and a JTextField object are among the instance variables defined by the class.

The JTextField and JButton objects are added, and their limits and actions are set up in the checkStrength() method, which also creates the GUI for the password strength checker window. The actionPerformed method is invoked when the check button is clicked. By iterating through the password's characters and setting flags for the presence of lowercase, uppercase, and special characters, this technique determines the password's strength. A message dialogue stating that the password strength is strong is presented if the password contains all three types of characters. If not, a message dialogue is displayed advising the user to add special characters, lowercase letter combinations, and uppercase letter combinations to their password in order to make it more secure.

As seen below in this code when any component registered for action events triggers an action event, the overriding function actionPerformed() defined in this code is called. In this instance, the JButton components check and main_page have been registered for action events, and this function defines the corresponding actions for each of these components.

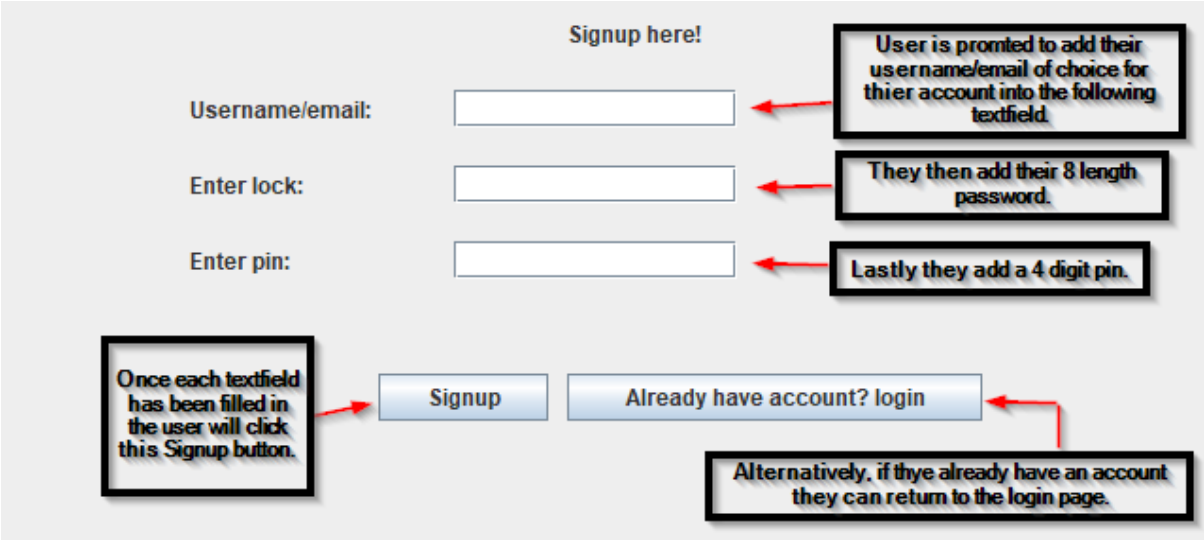
When the check button is pressed, the procedure initially uses the password text field's getText() method to retrieve the user's supplied password. Then, using the isLowerCase(), isUpperCase(), and isLetterOrDigit() methods of the Character class, iteratively verifies each character in the password string to see if it is a lowercase letter, an uppercase letter, or a special character.

As it examines each character, it checks for lowercase, uppercase, and special characters, and if any of those are present, it sets the boolean variables hasLowercase, hasUppercase, and hasSpecialChar to true. It checks to see if all three variables are true after going through every character in the password. If they do, a message dialogue box stating that the password is strong is displayed. If not, a notification dialogue box appears advising the user to add special characters, lowercase letters, and capital letters to strengthen the password.

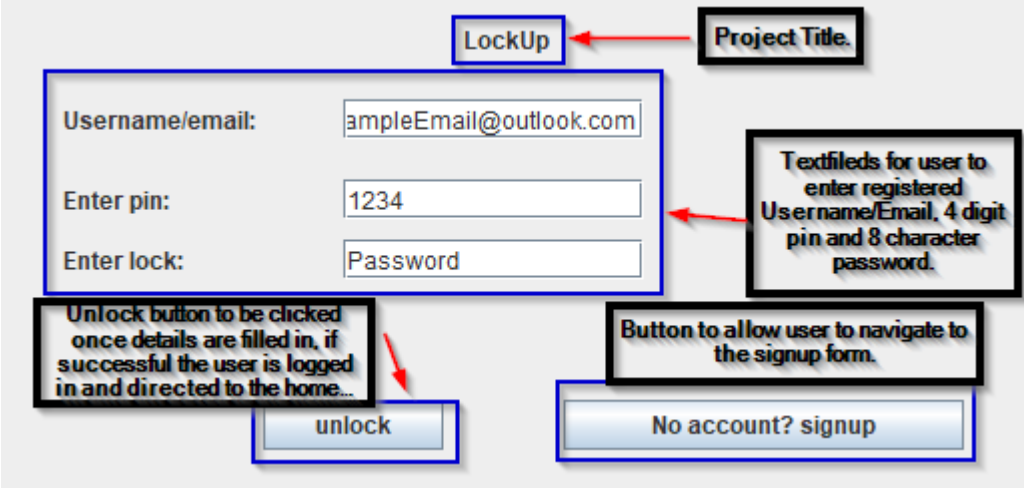
```java
@Override
public void actionPerformed(ActionEvent e) {
    if(e.getSource() == check){
        String pass = password.getText();
        boolean hasLowercase = false;
        boolean hasUppercase = false;
        boolean hasSpecialChar = false;

        for (int i = 0; i < pass.length(); i++) {
            char c = pass.charAt(i);
            if (Character.isLowerCase(c)) {
                hasLowercase = true;
            } else if (Character.isUpperCase(c)) {
                hasUppercase = true;
            } else if (!Character.isLetterOrDigit(c)) {
                hasSpecialChar = true;
            }


        }
        if (hasLowercase && hasUppercase && hasSpecialChar) {
            JOptionPane.showMessageDialog(strength_check_frame, "password strength is strong");
        } else {
            JOptionPane.showMessageDialog(strength_check_frame, "password strength is " +
                    "not strong i.e include special characters, lower case ,  upper case letter combination to" +
                    " make it strong");
        }
```

2.4 Graphical User Interface (GUI)

Registration Screen: The user is encouraged to create an account by providing a username or email address, a password, and a four-digit pin, as seen in the figure below. There is a "Signup" button that, when clicked, directs the user to the Login screen after a successful registration. In the event that a user accidentally clicked on the registration form or if they already have an account, they have the option to go straight to the login screen.



Login Screen: The user is greeted by the name of the application—"LockUp"—on the Login page, as shown in the image below. They are then asked to enter their account information from the registration form into the designated textfields before clicking the "unlock" button. If their login information is accurate, they will successfully connect into the application and be taken to the home screen, which will be displayed next. A notification showing "User not found" will appear if the system is unable to recognise their information.



Home Screen: The home screen, the host of the primary functions, is seen in the following image. As can be seen, once the user starts adding information, there is enough of room for it to be stored and shown. There are 4 text boxes available for the user to contribute information to. The first area, "service," asks the user to specify which service the information they are saving is related to, such as Netflix or Facebook. "service url" lets the user paste the website's or service's URL so they can quickly return to the one where they have an account, and "username" will be used to capture the user's username or email that they used to sign up for the specific service, Lastly, the "password" text area, like the username text field, will accept the password that the user entered when they created their account.

When all of these details have been added, the user can click the "Add Details" button, and if their entry fits the requirements, a success message will show up. Once entered, the information will be written to a local file created on the user's device and shown in the text box for the user to examine and scroll through.

The next option is the "Edit Details" button, which, when selected, enables the user to choose the details they want to edit and prompts them to input those changes in the relevant text area.

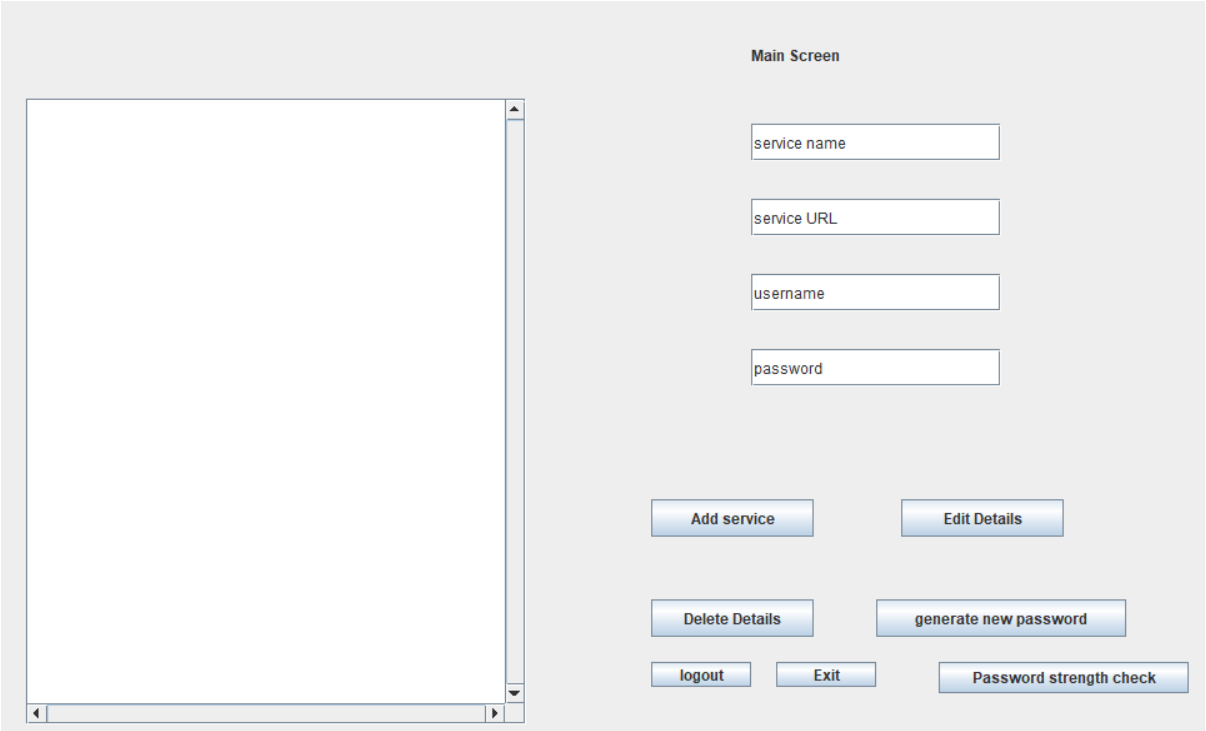
There is also a delete option, which, like the edit option, enables the user to choose a specific set of information and then press the delete key. This will erase the information from the application as well as the "data.txt" file that is used to store user information.

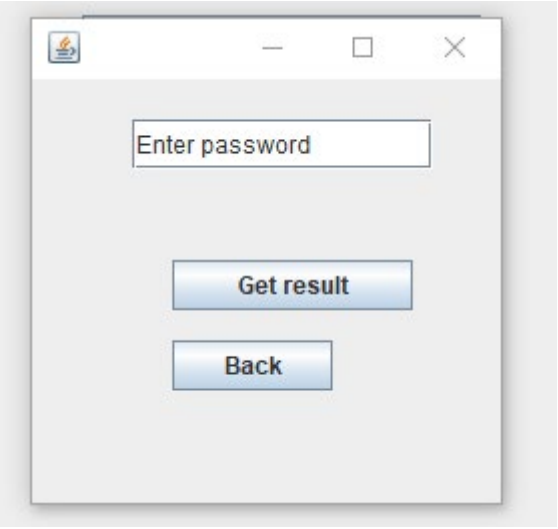
We also have a password-generate button. As was already said, when this button is pressed, a sequence of operations are carried out to create a brand-new password for the user that combines upper- and lowercase letters, digits, and special characters.

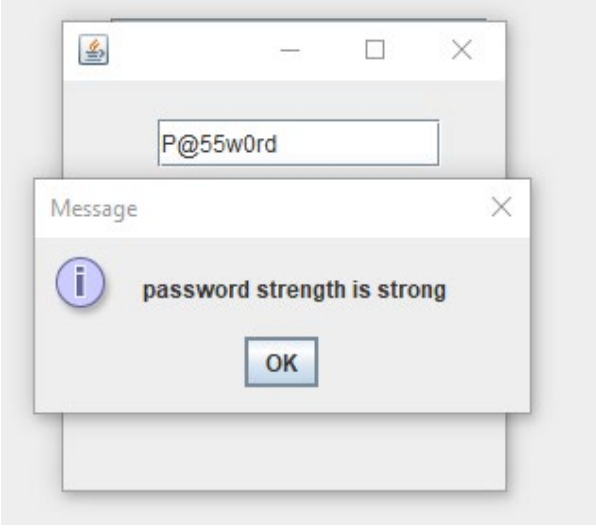
The password strength check button also has a gui that asks the user to enter the password they want to test. Based on the criteria specified in the code, a JOptionPane function runs and displays to the user whether their password is strong or weak and offers suggestions for how to make it stronger if it is weak.

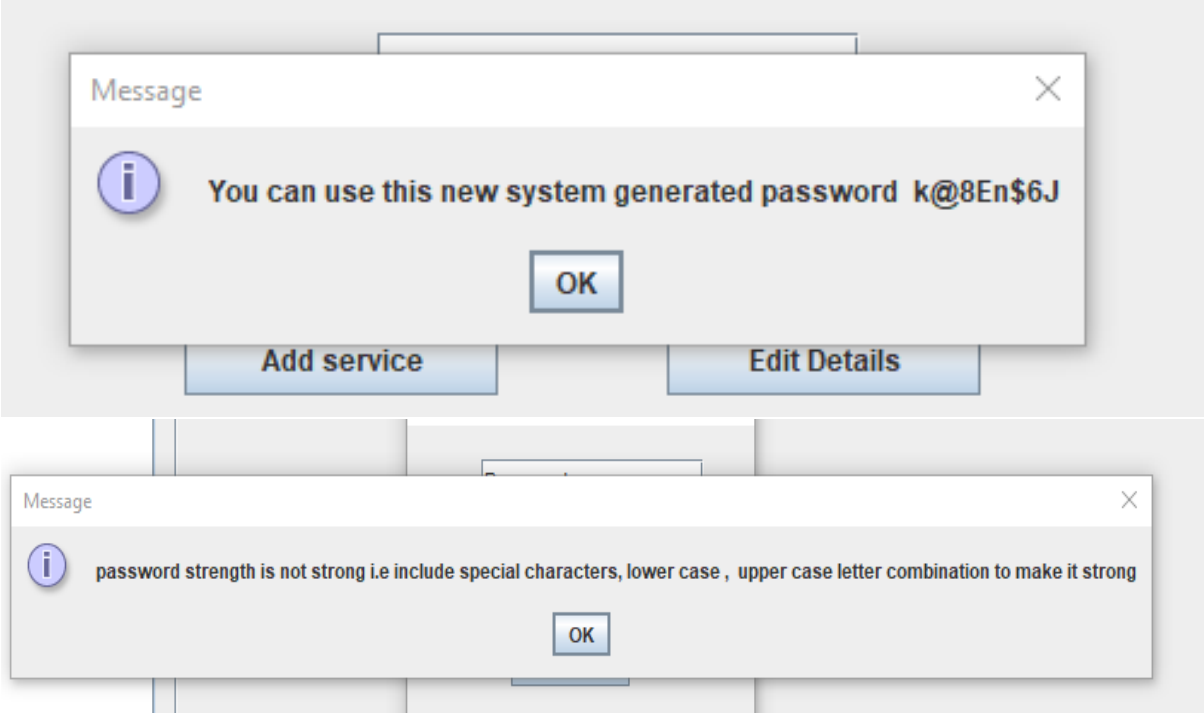
Finally, there are two straightforward exit and logout buttons. The exit button forces the application to close while the logout button logs the user out and takes them back to the login page.

Below you can see screenshots of the home screen, the password generator and the password strength tester.









2.5 Testing

Working on this project, I knew I would have to deal with testing at some point. I came to the conclusion that unit testing, particularly black box testing, would be the most effective kind of testing that could be applied to the programme while it was being developed. Below are the details of the black box testing I ran on the application, including the plan and results. Due to their size, I've included them as openable files. The first file contains the testing strategy, which should be read before continuing. The following nine files contain the results of the nine tests I conducted.

Testing%20Plan%20Documentation.doc
Test%20Case%20No.1%20User%20Regis
Test%20Case%20No.2%20User%20Login
Test%20Case%20No.3%20File%20Handli
Test%20Case%20No.4%20Adding%20De
Test%20Case%20No.5%20Editing%20De
Test%20Case%20No.6%20Deleting%20D
Test%20Case%20No.7%20Password%20
Test%20Case%20No.8%20Password%20
Test%20Case%20No.9%20User%20Interf

2.6 Evaluation

When evaluating my application, I choose to consider the following factors, performance, security, usability, compatibility and scalability. Each factor was evaluated individually, and the results again will be attached as openable files in order to save space.

Performance: When evaluating the applications performance, I was attempting to find out how fast and effectively does the password manager work? Is there any lag or delay in password generation or retrieval. To do this we'll be trialling each piece of functionality while having the application as the only software running on the test

device, then we will open several other types of software and complete the same tests again and measure the time it takes for the password manager to execute each feature The results of this evaluation will be below.

Performance%20Eva
luation%20Results.c

Security: This password manager appears to make use of file handling in a reasonably secure manner. A user-generated master password is used by the programme to encrypt the user's password data before it is stored locally in a file. This makes it more difficult for a possible attacker to obtain the user's password information on the device because it is not saved there in plain text. The possibility that an attacker may succeed in accessing the user's device and stealing the encrypted password data still exists. Additionally, the encryption may not be as successful at safeguarding the user's data if they use a weak or simple-to-guess master password.

Any programme is susceptible to malware assaults in terms of possible weaknesses, and this password manager is no different. An attacker could be able to extract the encrypted password data from the local file or intercept the master password if they are successful in infecting the user's device with malware. Another possible weakness for this password manager is shoulder surfing. An attacker may be able to get the user's password information if they can see the user's screen while they type their master password.

While I believe that this password manager has some security precautions to safeguard the user's data through the processing of files, it is still susceptible to virus assaults and shoulder surfing. Users should use secure, one-of-a-kind master passwords and take other essential security measures to safeguard their devices from potential attacks.

Usability: To assess how simple it is for users to browse and utilise the programme, the password manager's usability must be evaluated. The UI of this password manager is designed to be simple to use and intuitive. Users can quickly locate and use the application's numerous capabilities thanks to its clear and uncomplicated structure. Users may easily add, modify, and remove personal information. The generator and password strength testing functions are easy to use and offer helpful feedback on the security of the passwords being generated and saved.

A user might try a test scenario where they establish a new account and add a set of details in order to become completely acquainted with all of the password manager's features. Then, they might attempt modifying and erasing this information. Then, they may generate a new password and store it using the generator and password strength testing capabilities.

Through accessing the application over the course of development I can give a accurate evaluation that the application is simple to use, and users will have no trouble navigating through it and using all of its features thanks to its user-friendly layout. Users may

familiarise themselves with the password manager's capabilities and raise their level of familiarity with the programme by experimenting with its many features and functionalities.

Compatibility: Although the password manager programme can only be used on desktop computers right now, this does not restrict its applicability. Since many users still primarily use desktop or laptop computers for their everyday business and personal activities, a wide spectrum of users may access and benefit from the password manager. Users may access their passwords on their chosen device thanks to the application's interoperability with a variety of operating systems, including Windows, Mac OS, and Linux.

Considering this, having a mobile version of the programme that works on both Apple and Android smartphones would be excellent. The availability of a password manager on smartphones and tablets would significantly increase its user base and make it much more comfortable to use, given the rising popularity of these devices. A mobile version that smoothly integrates with the desktop application and offers consumers a full password management solution across all of their devices might be created with additional time and resources.

Scalability: I've chosen to evaluate this by discussing the following points, can the password manager manage massive user data volumes? Is it effective at storing and retrieving passwords even when the number of user accounts and passwords increases?

By posing these queries to myself, I may state Since there is no limit to how much data may be saved, the password manager programme is highly scalable. This is because file handling is being used, which is a productive approach to store and retrieve user login information. The storage and retrieval mechanism is well optimised to handle an ever-increasing volume of data since the application is built to handle many users and their login credentials.

Its capability to store and administer a huge number of user accounts without any problems is one of this application's key advantages. No matter how many accounts have signed up, the programme will always be able to access and get the data it requires thanks to the file management technology in use. The system is quite effective, so even with a huge number of users, the programme will be able to function at its best.

3 Conclusions

In this section, I will go over the benefits, drawbacks, strengths, and weaknesses of this Java-based Password Manager Desktop Application. We'll look at the advantages of the application's password management, convenience, and user-friendly interface, as well as its drawbacks, such as its reliance on Java and limited functionality. Furthermore, we will investigate the potential security issues associated with the use of file handling to store user data locally, as well as how the application's password strength tester and password generator features can improve security.

Firstly, we'll begin by discussing the advantages of this application.

- Improved security: The Password Manager Desktop Application enables users to safely save their login credentials on their device locally. The 8-character password and 4-digit pin make it challenging for other individuals to access the user's account information, increasing security.
- Convenience: The application relieves the user of the burden of remembering multiple login credentials for various services. They can easily access all of their account information from a single point.
- Password Strength Tester: The application includes a password strength tester to assist users in creating more secure passwords. It is a beneficial piece of functionality that encourages users to choose secure passwords, increasing their security both within the application and when creating new accounts for other services.
- Password Generator: The application's password generator feature generates a unique and secure password for users. This eliminates the need for a user to create a new password for themselves each time they create an account, which when uninformed of how to make a strong password could put the user at risk.
- User-Friendly Interface: The application has an easy-to-use user interface that is simple and intuitive. This feature makes it usable for users who are not technically savvy.

Next, we can move onto the disadvantages.

- Dependency: The Password Manager Desktop Application is dependent on file handling to store the user's data locally. If the file handling system fails, the user's data may be lost, leading to inconvenience and potential security issues.
- Limited Platforms: As the application is developed using Java, it may not be compatible with all platforms. It may not work on older operating systems or non-Java environments. Although, any modern device with any version of Java 7 onwards would find no issues using the application.
- Single Point of Failure: Since the application stores all the user's login credentials in one centralized location, it creates a single point of failure. If the user loses access to the application, they could potentially lose access to all their accounts. However, this would often be the case with any type of application centred around storing data locally.
- Lack of Cross-Device Functionality: The Password Manager Desktop Application is designed to work only on a single device, which may be inconvenient for users who need to access their accounts from different devices.

Thirdly, we have can discuss the strengths of the application.

- Local Storage: The application stores user data locally, which is a significant benefit for security-conscious users. Users can reduce the risk of online hacking or data breaches by storing data locally, as sensitive information is not transmitted over the internet. Furthermore, local storage gives users complete control over their data because they can choose where and how it is stored.

- Password Strength Tester: The password strength tester feature is a useful tool that assists users in creating secure passwords. The feature evaluates password strength based on factors such as length, complexity, and uniqueness. It gives users immediate feedback on the strength of their password, encouraging them to choose stronger passwords that are less vulnerable to hacking or brute force attacks. Furthermore, the password strength tester feature teaches users about best password security practises, which improves their overall online security.
- Password Generator: The password generator tool is a crucial component of the Password Manager Desktop Application since it guarantees that users establish unique and safe passwords for their accounts. Users can avoid the typical problem of choosing readily guessable passwords or repeating passwords across several accounts by utilising the password generator. The password generator generates random and complicated passwords that are difficult to decipher, improving users' overall online security.
- User-Friendly Interface: One of the Password Manager Desktop Application's primary assets is its simple and straightforward UI. Users may easily explore the application's features, update or change their stored account credentials, and generate or test passwords thanks to the application's user-friendly design. The application's UI is meant to be usable by users with varied degrees of technical competence, which improves usability and makes it a useful tool for password management.

Finally, we can discuss the limitation of the application.

- Java dependency: Because the programme was created in Java, it may not be compatible with all systems. While Java is extensively used and supported, some platforms, such as mobile devices or newer operating systems, may not support the programme or may require additional configuration. This may limit the app's potential user base, especially among those who rely on mobile devices for password management.
- Limited Security Features: While the Password Manager Desktop Application has enhanced security features such as password strength checking and local data storage, it may not be as safe as other password managers that employ more advanced encryption techniques. The programme, for example, employs file handling to store user data locally, which may be exposed to assaults such as malware or physical theft. Furthermore, the programme may lack features such as end-to-end encryption and password sharing controls seen in other password managers. While the programme is fine for personal use, users who want higher levels of protection for professional or corporate accounts should look into more complex password managers.

4 Further Development or Research

In my opinion, the Password Manager Desktop Application that I created is a fantastic starting point for an important project of strengthening online user security. There are various places I'd want to take this project if I had more time and resources to improve its functionality, security, and user-friendliness.

To begin, one possible route for this project is to provide industrial-level security measures. Although the application has certain security features, such as the ability to save user data locally using file management, it may not be as safe as other password managers that employ more modern encryption techniques. To improve user security, more complex encryption mechanisms, such as AES encryption, would need to be added to the programme. This protects users' login passwords and other critical information from unauthorised access and hacking attempts.

Second, while file management is appropriate for small-scale databases, it may not be scalable for large-scale databases. To guarantee that the programme can handle large-scale databases, a database management system, such as MySQL or MongoDB, must be integrated into the application. This would enable the application to more efficiently store and handle data, making it easier for users to maintain their login passwords and other sensitive information.

Next, one of my favourite things to accomplish with this project would be to turn it into a mobile app and make it available on the Windows and Mac app stores. This would increase the number of users who may use the programme to handle their login passwords and other sensitive information. Furthermore, turning the programme into a mobile app would allow users to access their account information from anywhere, at any time, making it more handy and accessible for consumers. Incorporating face ID as a method of login for the mobile application will also make it easier for users to access their account information while also boosting security by ensuring that only authorised users can use the application.

Furthermore, making the programme a browser plugin would be a useful feature for users. The browser extension would be accessible for all major browsers, allowing users to stay logged into the programme while browsing the web. When a user creates a new account for a service, the application might prompt them to add new login credentials to the password manager. For example, if a user opens a new Netflix account, the application may request the user via the browser extension to add these data to the password manager. If the user hits agree, the information is simply added to the user's password management account saving the user from having to do it manually thus improving the functionality of the application as well as the users experience.

Penultimately, establishing reminders in users' calendars to notify them when their passwords have been kept for more than 30 days is a significant feature that would increase user security. Users frequently fail to update their passwords, putting their accounts at danger of being stolen. The programme would remind the user to

update their passwords after a specified period of time had passed by creating reminders in the user's calendars. This could help to guarantee that user accounts are safe and not accessed by unauthorised parties.

Finally, an enhanced design and layout for the programme would increase its use. Users would find it simpler to browse the programme and handle their login passwords and other sensitive information if it had a more current and straightforward design.

In conclusion, the Password Manager Desktop Application that I created has a lot of room for future expansion. By including some, if not all, of the capabilities outlined, this programme has the potential to become a frequently used and popular password manager. This project, with the correct resources and effort, has the potential to significantly improve the security and ease of maintaining login passwords and other sensitive information.

5 References

Pew Research Center. (2017). Americans and cybersecurity. https://www.pewresearch.org/internet/2017/01/26/americans-and-cybersecurity/

Dashlane. (2020). The worst passwords of 2020. https://www.dashlane.com/download/Dashlane_Worst_Passwords_of_2020.pdf

6 Appendices

6.1 Project Proposal

https://studentncirl-my.sharepoint.com/personal/x19476134_student_ncirl_ie/Documents/Desktop/Year%204/Computing%20Project/Project%20Proposal/Computing%20Project%20Proposal%20Luke%20Ferrie%20x19476134.docx

6.2 Reflective Journals