**2700-3300 words, 700 below**

Password Manager

(Title WIP)

# Introduction

This report presents the COMP1004 coursework. The password manager created for this coursework focuses on data security.

Alongside the coursework, the Software Development Lifecycle (SDLC) is also discussed. An agile driven approach has been used for this project.

# Use of the Software Development Lifecycle (SDLC)

(Name WIP and section pending)

This project was planned over a week, with other password managers (such as the chrome password manager) being investigated.

Once planning concluded, the rough period of development was established, with sprints being assigned tentative tasks.  
A screenshot of a computer

Description automatically generated

In total, 10 sprints were planned, though the exact tasks assigned to these were significantly adjusted over the duration of the coursework, a result of tasks not being completed in the sprint period and new ones being discovered during development.

## Sprints

Microsoft Planner was used for sprint planning, the task board containing the sprints being arranged into two columns: To Do, and In Progress.

Below are the contents of the individual sprints:

### Sprint 1

20/11/23 – 03/12/23

Contents:

* Research password managers (complete)
* Research encryption algorithms (complete)
* Draft to end of design (incomplete)

Though parts of the introduction and SDLC section were complete, they were not finished in their entirety, even at a draft level.

### Sprint 2

04/12/23 – 17/12/23

Contents:

* Design page (complete)
* Build proof of concept page (incomplete)
* Begin code development (incomplete)

Minimal progress was made this week, though the page design was drafted, this can be found in the Page Design section.

### Sprint 3

18/12/23 – 07/01/24 (Abnormal duration to cover the Christmas break)

Contents:

* Make page draft (complete)
* Add Sign In functionality (incomplete)

Though not explicitly planned, initial coding was started this sprint, though this was minimal and the webpage was in no way functional.

### Sprint 4

08/01/24 – 21/01/24

Contents:

* Add Sign In functionality (incomplete)
* Get initial Loading and Saving functional (incomplete)
* Password Encryption / Decryption added (incomplete)
* Make video (complete)

As a result of a focus on the video hand in, none of the other sprint tasks were completed. This said, progress had been made on the Sign In functionality.

### Sprint 5

22/01/24 – 04/02/24

Contents:

* Add Sign In functionality (incomplete)
* Get initial Loading and Saving functional (incomplete)
* Add Show/Hide functionality (incomplete)
* Add Copy functionality (incomplete)
* Password Encryption / Decryption added (incomplete)
* Add initial Sign In functionality (incomplete)

As a result of an overestimation of efficiency and difficulties with JavaScript file management, no tasks were completed during this sprint. This said, further progress was made towards the loading and saving.

### Sprint 6

05/02/24 – 18/02/24

Contents:

* Add initial Sign In functionality (incomplete)
* Get initial Loading and Saving functional (incomplete)

As with Sprint 5, limited progress was made because of difficulties with JavaScript file management.

### Sprint 7

19/02/24 – 03/03/24

Contents:

* Add initial Sign In functionality (complete)
* Get initial Loading and Saving functional (incomplete)

Sign in was finally completed, and half of the second task -the loading- was also complete. As the export / save function had not been added yet, it could not be considered complete.

### Sprint 8

04/03/24 – 17/03/24

Contents:

* Add password copying (complete)
* Add the Show / Hide function (complete)

Both tasks for sprint 8 were completed, though these were smaller. Progress was made on unplanned tasks, but not sufficient to mark as complete.

### Sprint 9

18/03/24 – 31/03/24

Contents:

* Add credential display functionality (complete)
* Allow for the creation of credential display elements (complete)
* Allow for the creation of credentials (complete)
* Allow for credential updates to be saved (complete)
* Implement Encryption / Decryption / Hashing (incomplete)
* Test (incomplete)

Significant progress was made in sprint 9, leaving two remaining tasks program-side. The password manager was now fully functional but lacked security.

### Sprint 10

01/04/24 – 14/04/24

Contents:

* Hashing added (complete)
* Encryption / Decryption added (incomplete)
* UML diagrams created (complete)

Half-way through the sprint it was realised that time for the portfolio writeup was running out, so my priorities switched to completing it. This means that the encryption and decryption functionalities will not be implemented before this portfolio is submitted.  
The GitHub page was partially rolled back to ensure that the provided program was functional, even if missing the WIP (and in fact broken) encryption and decryption features.

# Design

## Project Vision

This password manager will be for individuals to help store their passwords in a secure way. The application will store passwords in encrypted files, and these will only be decrypted when a successful sign-in occurs, preventing unauthorised access to passwords.

## Background

Password managers are important for the secure, everyday use of computers. Provided they are competent in securing their accounts, a person will have several passwords, most of which are not memorable. A user can either store their passwords physically (which is less portable, and may be vulnerable to theft), in an unsafe plaintext file, or in a secure password manager.

Password managers are effectively databases which store usernames and passwords for different sites in a controlled form. A strong manager protected behind a strong access password means the user only needs to remember one password (Li, 2014).

A password manager must be secure, otherwise it can introduce a significant vulnerability by potentially exposing all of a users’ passwords. Many managers have security flaws, with 4 out of 5 studied password managers in a study being vulnerable to attacks which reveal credentials (Li, 2014).

Regardless of where password manager data is stored: portable, cloud or local, there are dangers. This is because USB Memory Sticks and Smartphones can be stolen, and cloud storage can be accessed without permission by attackers. Though storing locally is more secure, requiring an attacker to either gain access to the computer, it is still not acceptable. Because of these threats, the password manager must use a secure format. (Gasti, 2012).

## User Stories and Use Case Scenarios

Below are the User Stories identified for this project. These are shown in a Use Case Diagram and decomposed later in this section.

* As a user, I want to have easy access to my passwords so that I can spend less time signing in and won’t lose them.
* As a user, I want my log in information to remain secure, so it is unlikely for my accounts to be breached.
* As a user, I want secure information (such as my passwords) to be behind a sign in, so malicious actors cannot access them even if they gain access to my computer.
* As a user, I want to be able to reset my login password, so if I forget it, I can still access my other passwords.
* As a user, I want the application to be easily accessible, preferably without requiring any prerequisites, so it is easier to install and use.  
  Note: As this is a web-based project, the JavaScript is visible to anyone using inspect element. For security reasons it is necessary to find a way of hiding the encryption method. The best way of achieving this is through an API.
* As a user, I want to be able to add new passwords for when I create new accounts.

Below is the Use Case Diagram for these stories:  
A diagram of a sign in

Description automatically generated

Decomposed Sign In process:

|  |  |
| --- | --- |
| Usage: | Sign In |
| Description: | The user attempts to sign in to the password manager. |
| Precondition: | The user has a stored account and key, credentials are provided |
| Post-condition: | The user is signed in and credentials are displayed. |
| Error Situations: | There is no stored account / key. The stored account does not match the provided credentials. |
| Error State: | Waits for the stored account / key to be provided and for the user to try again. Error displayed stating incorrect credentials have been provided. |
| Actors: | User |
| Triggers: | User needs to gain access to the password manager. |
| Standard Process: | 1. User enters their username and password into the respective textboxes. 2. User presses the Key button and provides a key (JSON) file. 3. User presses the Sign In button and provides the credential storing (JSON) file. 4. System verifies the username and password are identical to the stored ones. 5. System displays credentials, decrypted using the key file. |
| Alternative Process | 3’. User does not provide a file. 4’. System waits for the user to press the button again and provide a file.  4’’. The username, password or both do not match.  5’’. System displays an error message.  4’’’. System detects that the credential storing file is incorrectly formatted.  5’’’. System displays an error message.  5’’’’. System fails to decrypt the credential storing file as the key file is wrong or incorrectly formatted. 6’’’’. System displays an error message. |

The other processes would also have been decomposed; however, these do not have alternative processes, and regardless of user input will always run the same.

## Architecture

(Architecture pending – the architecture contains UML class diagrams)

A sequence diagram for the Sign-In process:  
A Sequence Diagram for the Sign In process of the Password Manager.
The five lifelines are Webpage, signIn, extractDetails, the Crypto API, and createContainer.

A sequence diagram for the Saving process:  
A sequence diagram for the Saving process of the Password Manager.
The three lifelines are Webpage, updateFile, and the Crypto API.

## Page Design

Below are the initial page designs, these were adjusted during implementation.

A black text on a white background

Description automatically generated

A sign in box with red text

Description automatically generated

The sign-in box was to be displayed on top of the main password manager page, which would not have any information present, and might have been blurred.

A screenshot of a computer

Description automatically generated  
This is an initial prototype page.  
When functional, no passwords will be shown until the user signs in, at which time they will be decrypted and the containers filled.  
The contents of Website 1 can be shown and hidden.  
Show Password sets the password box from the “password” type to “text”, making it visible and able to be directly copied.  
Copy Password copies the contents of the password box to clipboard without showing it.

The implemented page design is below:  
A screenshot of a computer

Description automatically generated  
A screenshot of a login box

Description automatically generated

The only notable changes with the implemented design are that there is now a “New Entry” container, and “Save Changes” button. The former is used to create new credential entries, while the latter saves all entries to file.

# Implementation

File format:

|  |
| --- |
| {  "username": "19226408612613073252229515524293121112222592141381781015236",  "password": "22616935841013821195321921029912002372027015711419412114",  "site1":{"username":"abcd","password":"password1"},  "site2":{"username":"def","password":"password2"},  "Google":{"username":"Max","password":"123"},  "Plymouth University":{"username":"Max.Lowther","password":"123"}  } |

As can be seen, the username and password values associated with sign-in are hashed. Due to space constraints, the values have been shortened.

This file is downloaded when either the “Save Changes” or “Sign Out” buttons are pressed, ensuring it remains up to date. Due to the limitations of the system, however, the file which the values were loaded from cannot be updated, meaning an updated copy is created on every save.

The site names ("site2", for example), alongside username and password values were intended to be encrypted, but this feature was not implemented before the Portfolio was completed.  
Had encryption been implemented, there would be a second JSON file, Key, storing the output of the crypto.subtle.exportKey function, this would have allowed for the input to be decrypted.

## Poster

A screenshot of a computer screen

Description automatically generated

## Reflection

500 words, currently 50%

(Pending, discusses issues of learning new technologies, planning and accounting for modules and deliverables)

This project is limited in terms of what can be used. Ideally, an encryption algorithm would be in C#, C or C++, unfortunately it is required that the scripting side of the project is exclusively done in JavaScript. This means that the application is less secure than alternatives, and many open-source encryption / hashing algorithms cannot be used.

As the program required a number of HTML element creations, namely the creation of the Credential containers, and these elements needed to be uniquely identified, the function for creating containers is incredibly large, almost certainly being slower than using HTML templates to create non-uniquely identified elements. The only potential improvement that could be made would be to use templates, and then modify IDs, though this has not been implemented.

Web security measures prevented the manipulation of the saved credentials in files. This resulted in the need to have the user select the credentials when signing in. Though it could be argued that this is beneficial, as it allows for multiple users to use the app on the same device, it requires the user to do more to sign in. As files seemingly cannot be updated, a new file must be downloaded whenever the file is saved, occurring when the “Save File” and “Sign Out” buttons are pressed. If multiple edits are made and saved, this can result in file duplication, and if a user saves before signing out, the program will end up giving them two identical files.

# GitHub Repository

<https://github.com/CentralC0re/COMP1004>

# References

All references here. Standard format:  
*Title, Section (page, chapter, etc), Source: URL (Access Date)*

The Emperor’s New Password Manager: Security Analysis of Web-based Password Managers (Zhiwei Li et al), Page 3, Section 2.1, Source: <https://www.usenix.org/system/files/conference/usenixsecurity14/sec14-paper-li-zhiwei.pdf> (Accessed 28/11/23)

On The Security of Password Manager Database Formats (Paolo Gasti and Kasper B. Rasmussen), Section 1, Source: <https://ora.ox.ac.uk/objects/uuid:926086ea-180b-4f11-a599-2522a80837f4/download_file?file_format=application%2Fpdf&safe_filename=pwvault.pdf&type_of_work=Conference+item> (Accessed 28/11/23)