



Mimir: A Password Manager Project

By Edward William Bennett

Supervisor: Atta Badii

Abstract

Single paragraph, talks about title and objectives

Up to 250 words – keep short

Guidance on abstract writing: An abstract is a summary of a report in a single paragraph up to a maximum of 250 words. An abstract should be self-contained, and it should not refer to sections, figures, tables, equations, or references. An abstract typically consists of sentences describing the following four parts: (1) introduction (background and purpose of the project), (2) methods, (3) results and analysis, and (4) conclusions. The distribution of these four parts of the abstract should reflect the relative proportion of these parts in the report itself. An abstract starts with a few sentences describing the project's general field, comprehensive background and context, the main purpose of the project; and the problem statement. A few sentences describe the methods, experiments, and implementation of the project. A few sentences describe the main results achieved and their significance. The final part of the abstract describes the conclusions and the implications of the results to the relevant field.

Abstract

Methodology

Solution approach

- Explain MoSCoW

Implementation

- Encrypted backend
- Explain CRUD
- Authorisation
- Authentication
- Go into more detail as to how or why certain things were chosen – mention research

- Diagram

Results

- Fill out testing form
- Make questionnaire
- Fill out questionnaire

Evaluation

- Talk about results

Conclusion & future work

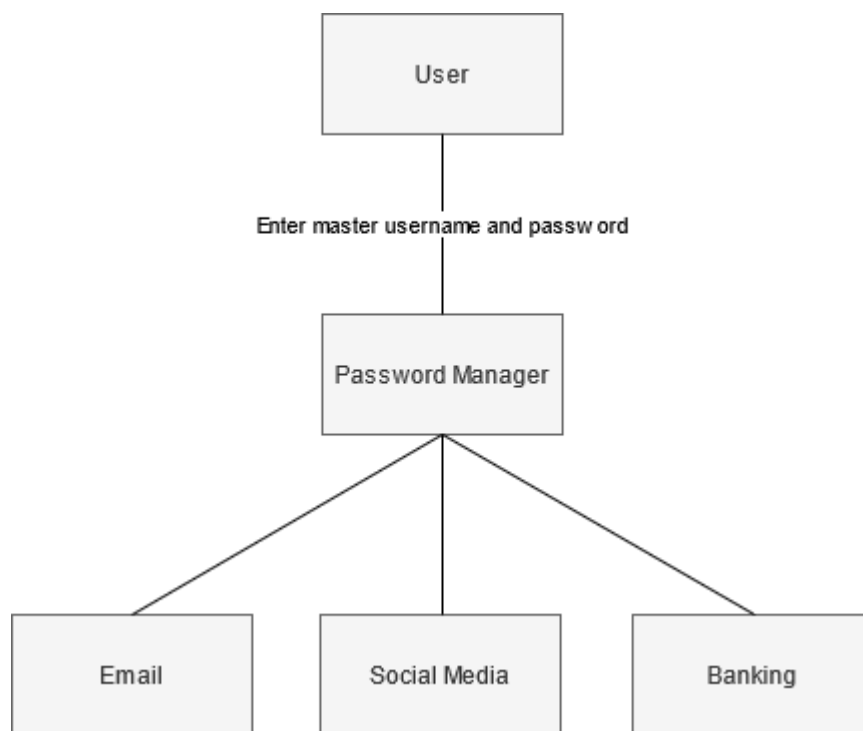
Table of Contents

Abstract.....	2
Introduction	1
Research.....	3
Existing Products.....	Error! Bookmark not defined.
Keeper	7
Bitwarden.....	13
Dashlane.....	18
LastPass	22
SplashID Safe	28
Literature Review.....	Error! Bookmark not defined.
Methodology.....	5
Requirements Engineering.....	34
Solution Approach	Error! Bookmark not defined.
Implementation	33
Results.....	55
Testing.....	55
User Feedback.....	56
Discussion and Analysis.....	57
Conclusion.....	58
Lessons Learned/Future work	59
Acknowledgements.....	60
Bibliography	61
Academic Sources	Error! Bookmark not defined.
Web Articles.....	Error! Bookmark not defined.
Libraries.....	Error! Bookmark not defined.

Introduction

A growing problem with online security is the tendency for users to choose passwords which are insecure due to length, lack of character variety, popularity, or reuse. This can open users up to having accounts hijacked, leading to stolen personal information, impersonation, or the simple inconvenience of having to make new accounts and change passwords.

At its most fundamental, a password manager is a database that allows users to store usernames and passwords for their online accounts. Users would log in to their “master” account using a username or email address and one strong “master” password. This reduces the cognitive burden of remembering multiple accounts to just one.



As users can access all other account details from this master account using a unique password for each online profile becomes much more feasible. Each unique password can now be much stronger, with every password being dozens of characters long. A common tendency is for a user to reuse either the same password for every online account, or variations of the same password along a theme. A password manager often includes facilities for generating new passwords, used to ensure that users can instead create a password that consists of random characters, rendering the password much more resistant to dictionary attacks or guesswork.

This project aims to create a simple but functional password manager available as a Web application, allowing a user to improve the security of their online presence using stronger and more unique passwords. In order for such a tool to be effective, it must be used. With this in mind, usability has been an additional focus, with the project also aiming to offer a service as simple and convenient as not using a password manager.

To achieve these aims the project must allow the user to create a master account and use it to store, retrieve, and manage their external account credentials. The project must also include a password generator tool to allow the user to create stronger passwords than they would be able to create independently. To fulfil the second aim of convenience, the project must be considered user friendly, with positive user feedback indicating ease of use.

This report is divided into several chapters. The first chapter is a problem articulation and literature review which will provide more context to the reason for password managers, and an examination of the literature around them. The second chapter, the methodology, will describe the process initially outlined to plan this project and the thought process of planning the research and evaluation. The third chapter is a systemic review of existing password manager projects. The fourth chapter, the implementation, will guide the reader through the process of conceiving and constructing a password manager. In the evaluation, the fifth chapter, this report will present the results of testing and a user feedback questionnaire to the end product. Finally, in the conclusion these results will be discussed, the project will be evaluated, and suggestions will be made as to where the project could move in the future.

Background

Online security has grown increasingly important as more and more of our day-to-day lives takes place online. This importance is magnified further as more significant actions are performed online, such as digital contract signing and online banking.

Poor password habits such as failing to frequently change passwords and choosing low entropy, easy to remember passwords are prevalent among many users (Florencio & Herley, 2007). A 2005 study found that end users have a “rather dismal record of enacting the basic hygiene behaviors”, failing to change passwords frequently or select strong passwords. Indeed, the study found that 62.5% of respondents declared they “never used numbers or punctuation marks” and 48.5% said they “had not changed their passwords in the last six months”. While the study found that training, awareness, and knowledge of monitoring passwords on average resulted in better password hygiene, the study also found that these improvements “seemed to associate with a greater likelihood of writing down one's password” (Stanton et al. 2005).

An additional pervasive issue with password security is password reuse – the tendency for users to use the same password for multiple online accounts. In 2014 a study of leaked passwords across 11 web sites found an estimated 43-51% of users reused the same password across multiple web sites (Das et al. 2014). Furthermore, the study identified that many users use the same basic password across multiple web sites which was transformed using minor variations. While this could be considered an improvement, the researchers were able to create a password-guessing algorithm which was able to guess 30% of transformed passwords within 100 attempts – a vast increase in vulnerability compared to 14% found for a standard password-guessing algorithm without cross-site knowledge. This indicates that for many users a data breach of one service can lead to compromised accounts elsewhere.

Password managers can thus be considered a more secure alternative to the average user's password habits, allowing for longer passwords which are unique to each online service. Indeed, members of the United States Computer Emergency Readiness Team asserted that password managers are “one of the best ways to keep track of each unique password or passphrase”, and that “Although moving to a password manager may take a little effort, in

the long run it is a safe and convenient method of keeping track of your passwords and guarding your online information” (Huth, Orlando and Pesante, 2012).

One concern with password managers is that if the master account is hacked, the user loses all their online accounts. Hashing is a one-way cryptographic function which is used to convert a password into a different format using a cryptographic salt. Rehashing the user’s submitted password for comparison with the stored hash is still useful for authentication, but much less useful to a malicious actor as they cannot reverse the hashing to find the original password to use elsewhere. For an attacker to obtain the original password from a hash, they must hash password guesses until they find a match. Some password hash functions such as Bcrypt employ key-stretching methods to increase the amount of time taken for a guess to be calculated (Blocki, J. and Datta, A., 2016). One example of key-stretching is rehashing the password multiple times. One method of attacking a hash, lookup tables, can be thwarted using a second input called a salt (Provos, N. and Mazieres, D., 1999). The salt is random data which is added to the password before it is hashed. As an attacker does not know the value of the salt, this adds additional difficulty in trying to guess the password from a hash. Bcrypt also uses salting.

With regards to stored password records, these must be encrypted reversibly so they can be sent back to the user in a usable way. If the user were sent back a password hash the client would be unable to reverse this, and they would not be able to use their password to login. Passwords should none-the-less be stored in as secure a manner as possible, using a private key known only to the server to encrypt them. The Advanced Encryption Standard (AES) is one such encryption algorithm which was selected by the US National Institute of Standards and Technology as the Advanced Encryption Standard from 15 candidate algorithms (Atasu, Breveglieri and Macchetti, 2004).

Methodology

Research

To research features and design choices for this project the first step was to examine existing products and identify features of these products which were both successfully and poorly implemented. Five existing password managers were identified: Keeper, Bitwarden, Dashlane, LastPass, and SplashID Safe. For each of these products an account was created with the same five example password records in two different folders to demonstrate how the manager might look under normal use.

Each password manager has been evaluated based on the common components users will interact with most: the “vault” page where the user’s password records are listed; the password records where an individual record is shown in greater detail; the password generator tool which allows the user to generate new passwords; the account settings page which can be used to update any account details and set personal preferences; and any miscellaneous features such as a security evaluator or two-factor authentication. Evaluating each of these components for existing products should give a strong foundation as to the features required of a password manager and could give an indication in how to construct them. The aesthetics and usability of each manager has also been reviewed throughout to obtain an idea as to how the interface of this project should be designed to maximise user friendliness and familiarity.

User Feedback

When evaluating usability, Whitney Quesenbery suggests in “Dimensions of Usability: Defining the Conversation, Driving the Process” the “5E” dimensions of usability: effective, efficient, engaging, error tolerant, and easy to learn. These dimensions are described below.

Effective describes the completeness and accuracy with which users achieve their goals. If a program is usually used correctly without errors, it is usually effective.

Efficient determines the speed with which work can be done. If a user feels that a task takes “too long” or “too many clicks” the user feels the software is not efficient.

Engaging describes how pleasant, satisfying, or interesting and interface is to use.

Error tolerant describes how well the product prevents errors, and helps the user recover from any errors that do occur.

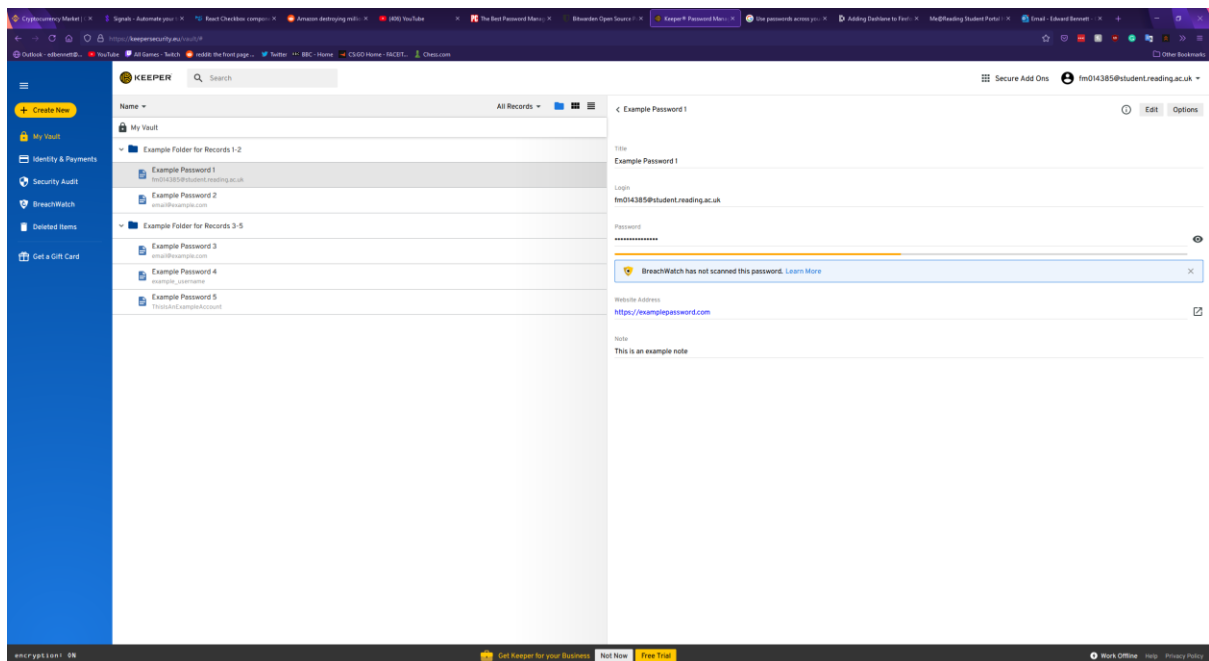
Easy to learn describes how well the product supports initial orientation, and deeper learning. If a user feels an interface is immediately obvious to learn – or relearn - to use, the user feels the software is easy to learn.

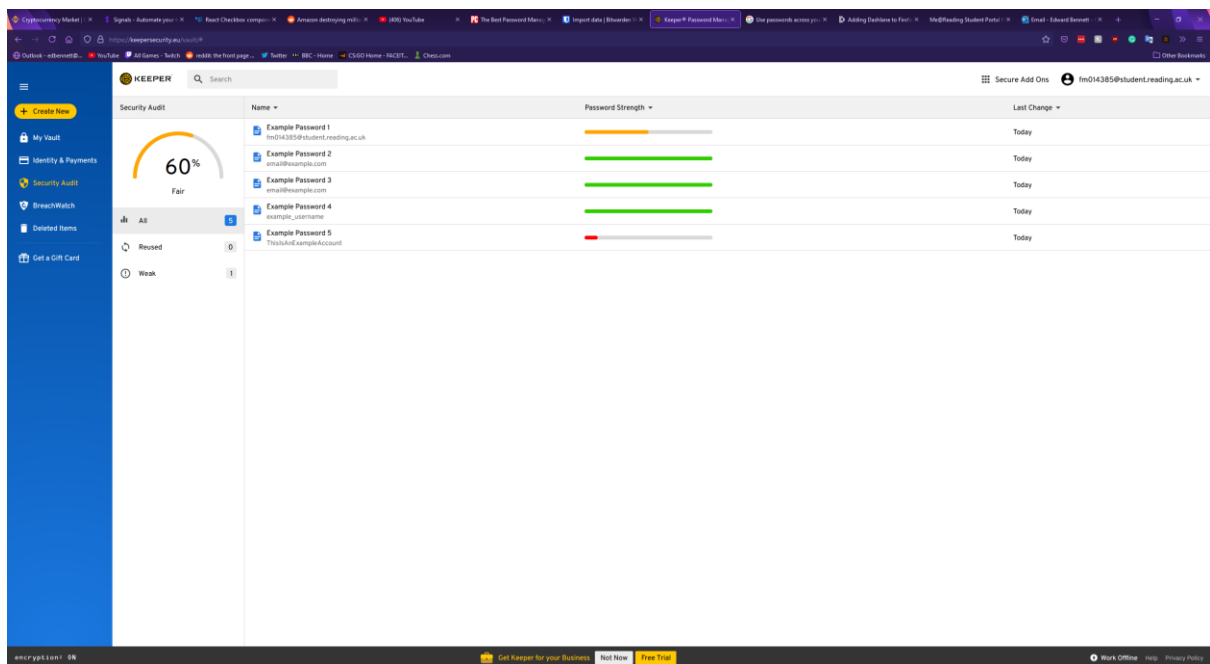
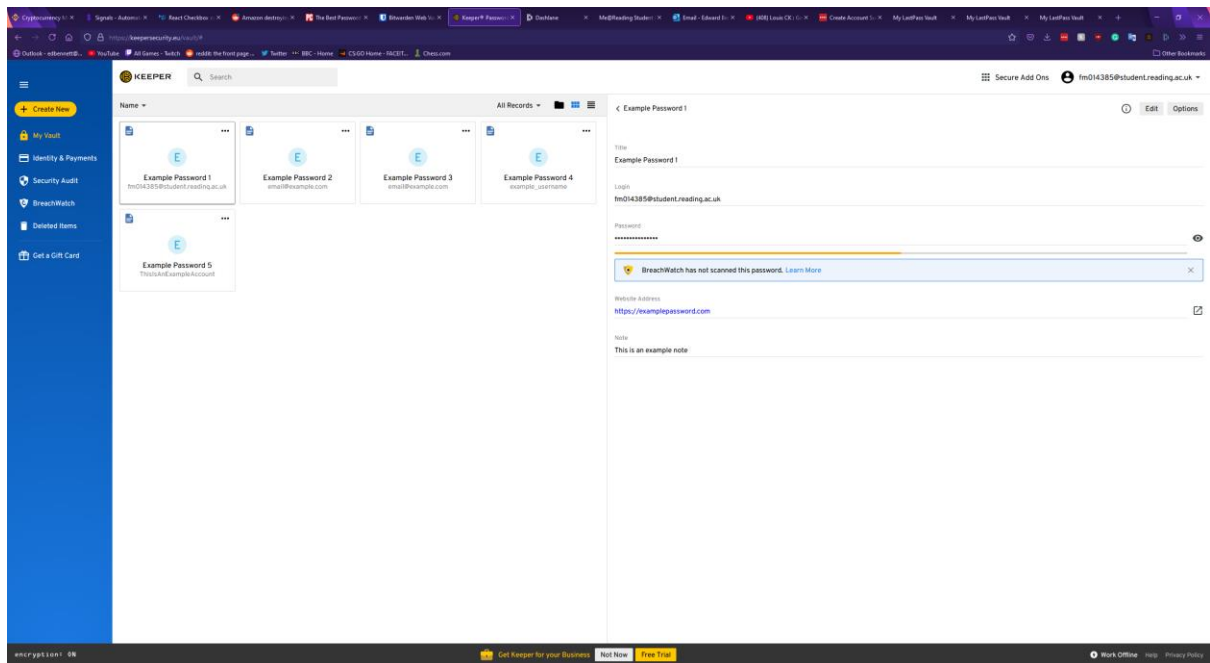
Whitney further suggests creating first-person statements that express a usability requirement for each dimension (Quesenbery, 2003). With this in mind, a user feedback questionnaire based on statements which target each dimension using Likert scales has been designed.

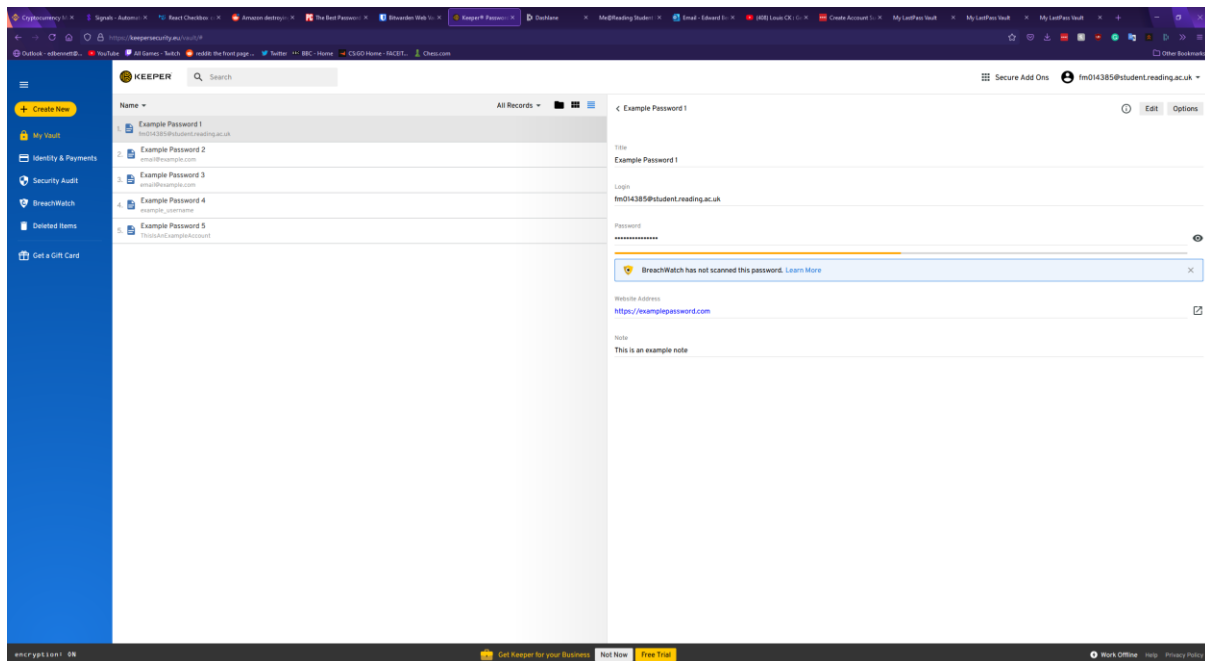
Systemic Review of Existing Password Managers

Keeper

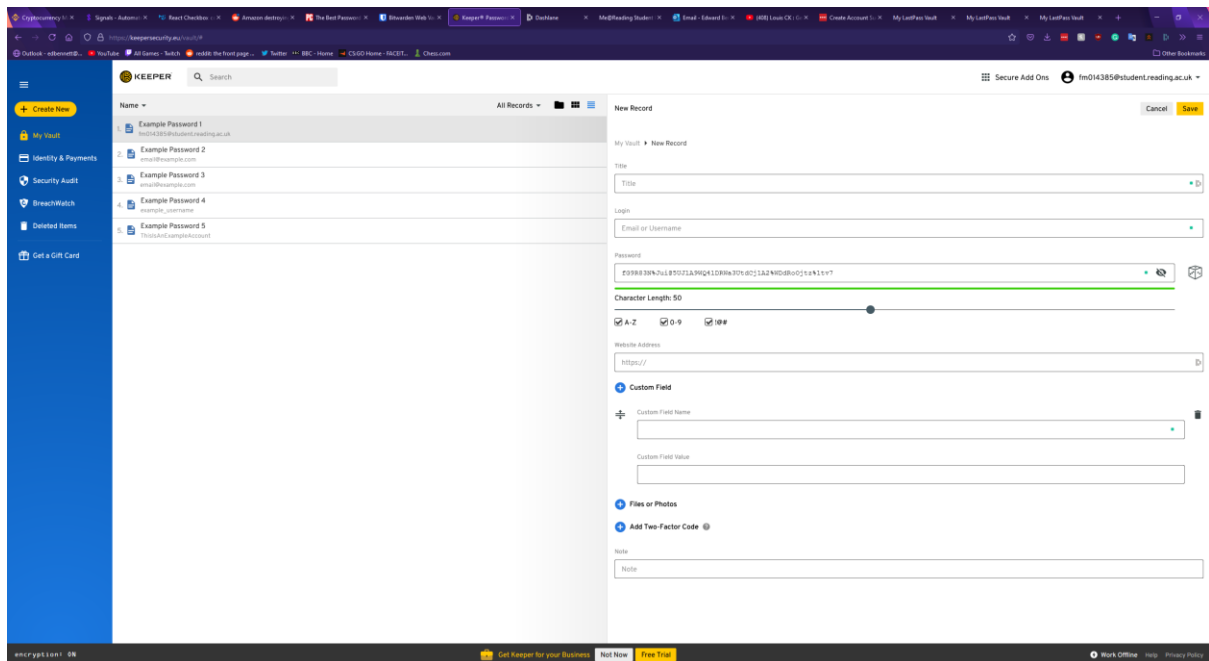
Keeper is a password manager with a clean and simple modern design with little wasted space. Keeper has a simple intuitive interface which clearly communicates to the user how to add a new password, view existing passwords, and organise their database. Most of the real estate of the page is dedicated to password records and clicking on a password opens a window to the right showing all the information about the record. The passwords in the main window are organised alphabetically, inside drop-down lists of folders if the passwords are assigned to a folder. Three different views for the records are also available, a list view of passwords and folders, a card view of passwords, and a list view of all passwords without folders. These three views are shown below with the first example password selected.





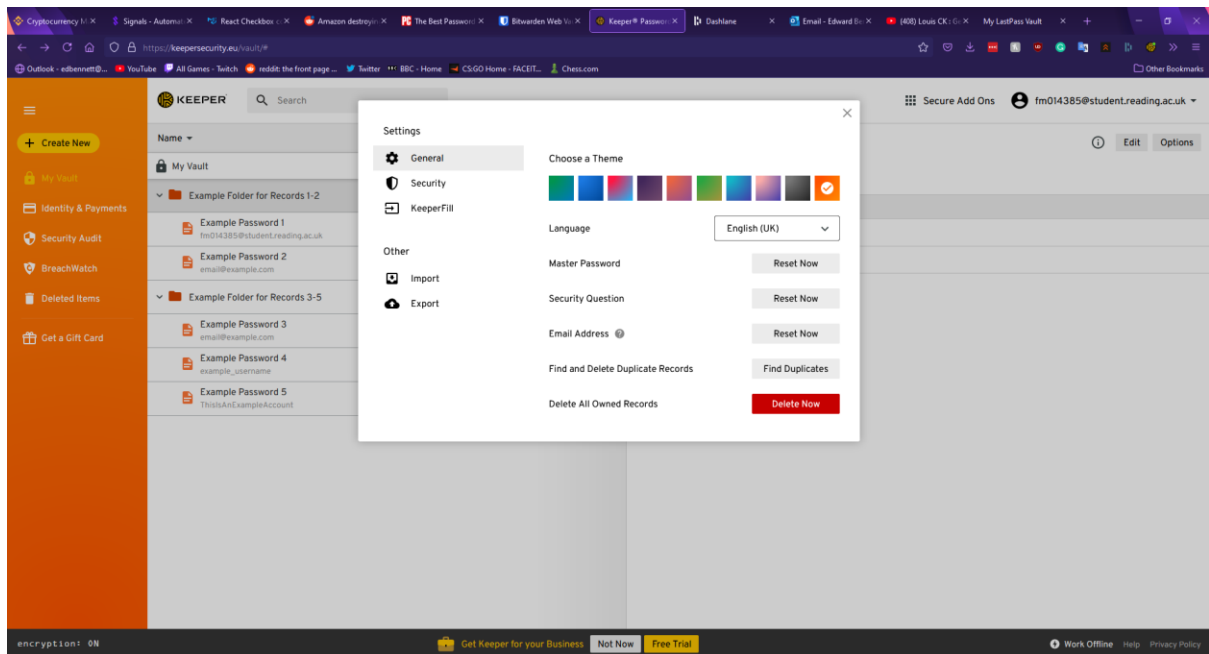


Records in Keeper are displayed simply, with each record possessing a title, a login value for a username or password, a field for the password itself, the website address, and a notes section. Each of these sections are optional, except for the title which must be filled in. In addition to these standard fields, Keeper allows custom fields, with the user able to give the field a name and a value, file or photo attachments, or a two factor authentication code. Once a record has been saved to the vault clicking on a field copies the field's value to the clipboard, allowing for more convenient copy and pasting of a username or password elsewhere, however this does mean altering a record can only be done after clicking a small edit button in the top right corner of the record. This could make editing a record frustrating for a user, and as the button is grey instead of highlighted a new user could struggle to find this feature. A second button labelled "Options" is next to the edit button, and this offers a shortcut to open the corresponding website for the record, a premium feature to roll-back the version of the record, sharing the record, adding the record to a list of favourites, assigning a colour to the record, duplicating the record, or deleting the record. Offering a colour coding system for records is a unique feature that could allow for quick identification of a password at a glance. The option to mark select passwords as favourites could also offer a convenient timesaving measure for a user, however placing the function to delete a record in such an obscure position could cause irritation for users as such a key feature is not immediately apparent. The window for creating a new record is shown below.

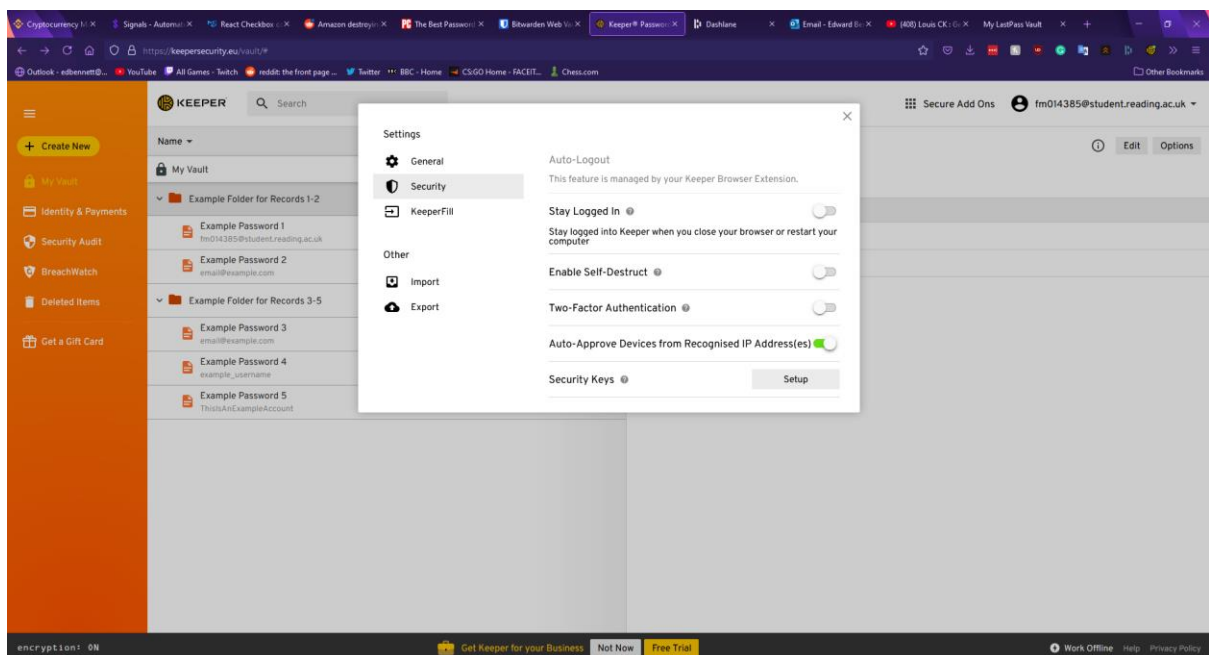


Keeper also includes a convenient password generator built into the new record window. The generation of a password is depicted by a dice icon, and once clicked a 20-character password is immediately generated and the generation settings expand if the user would like to tweak these settings. The settings for password generation include capital letters, numbers, and symbols, however lowercase letters are enabled by default and cannot be turned off. The length of the password can be customised between 8 and 100 characters in length. The above screenshot also shows the password generator with the settings expanded and a generated 50-character password with all characters enabled.

The settings and account page of Keeper is simple, with multiple tabs offering separation of concerns. The first tab shows general settings, and this allows a user to pick a theme. A feature for colour control could offer greater accessibility for users with conditions such as colour blindness or dyslexia, however the implementation here appears to mainly target aesthetics, as it only changes the colour of the sidebar and some icons. The general options page also includes options to reset the account's master password, security question, and email address. An organisational feature to automatically find and delete any duplicate records is included here and could be valuable for any users with large and poorly managed databases. Finally, a feature to delete all records is included at the bottom, highlighted in red to indicate that it is a dangerous option a user should not click out of curiosity. The general settings page is shown below.

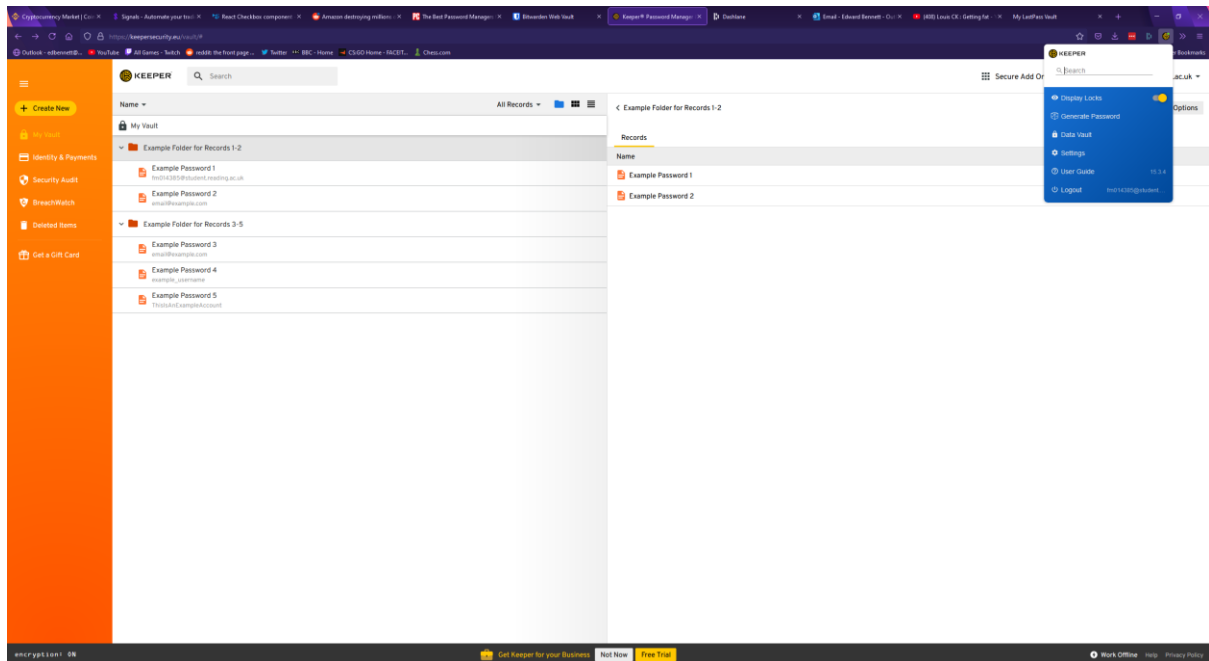


The second tab in the options menu is the security tab, and this section offers the user toggleable options to keep the user permanently logged in, a “self-destruct” option to enable automatic deletion of all records if five incorrect attempts to log in have been made in a row, an option to enable two-factor authentication for the account (both mobile app based and physical key based), and automatically approve devices from recognised IP addresses. This window of the settings menu is shown below.

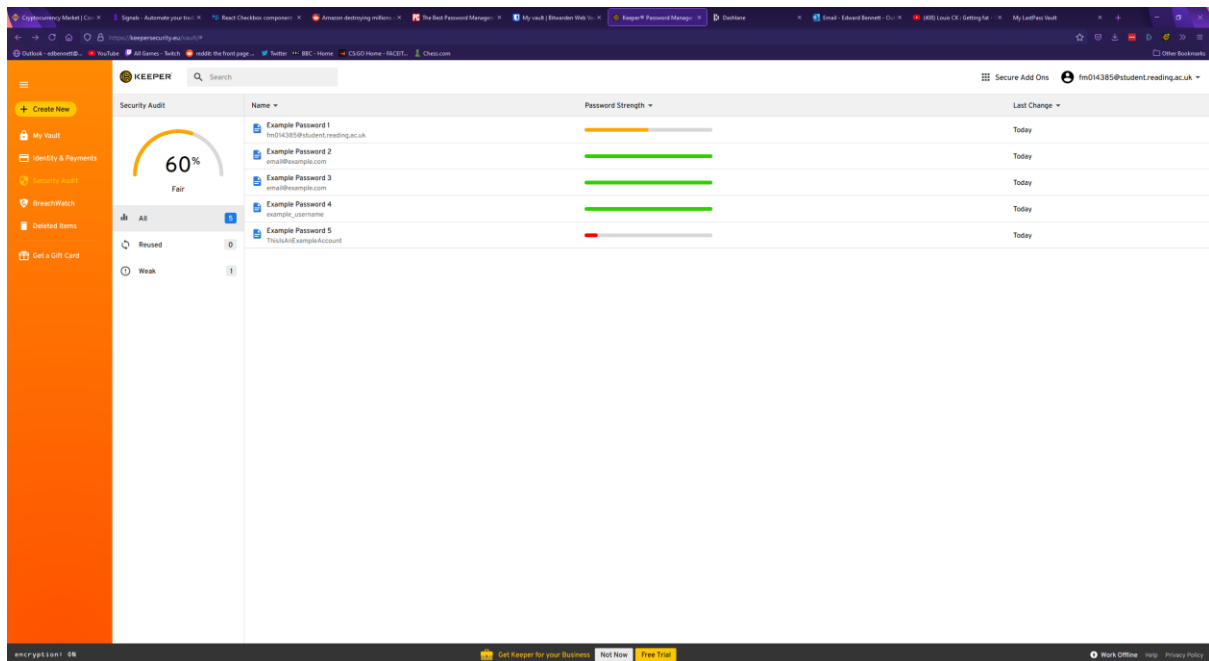


The final sections of the settings menu offer links to download the Keeper browser extension and import or export records as a file. The Keeper browser extension allows for

the quick searching of password records, generation of new passwords, and shortcuts into the main vault or the settings page. When logging into a website with the extension installed an icon appears next to the username and password fields with the Keeper logo. Upon clicking this icon, the extension offers to automatically fill the fields with stored records, or if no corresponding records are found the extension offers to create a new record and generate a password. This extension is shown below.

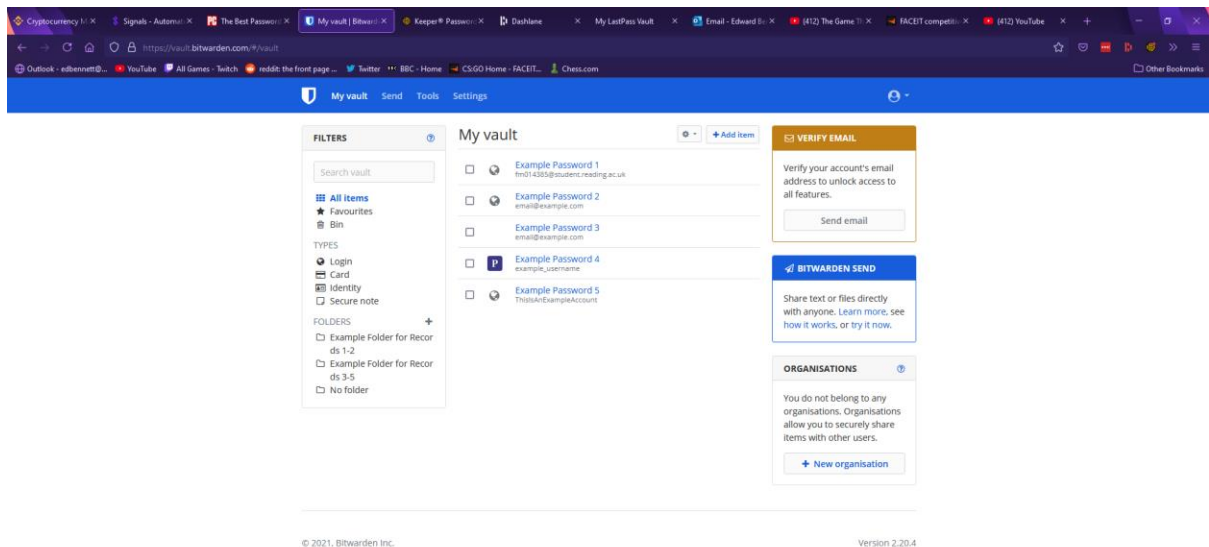


One other feature Keeper offers is a security rating page. This page rates the security of every password in the account with clear, simple breakdowns of each record, however the metric Keeper uses to determine the strength of a password, and specific reasons why a password is weak are not visible. The security audit page is shown below.

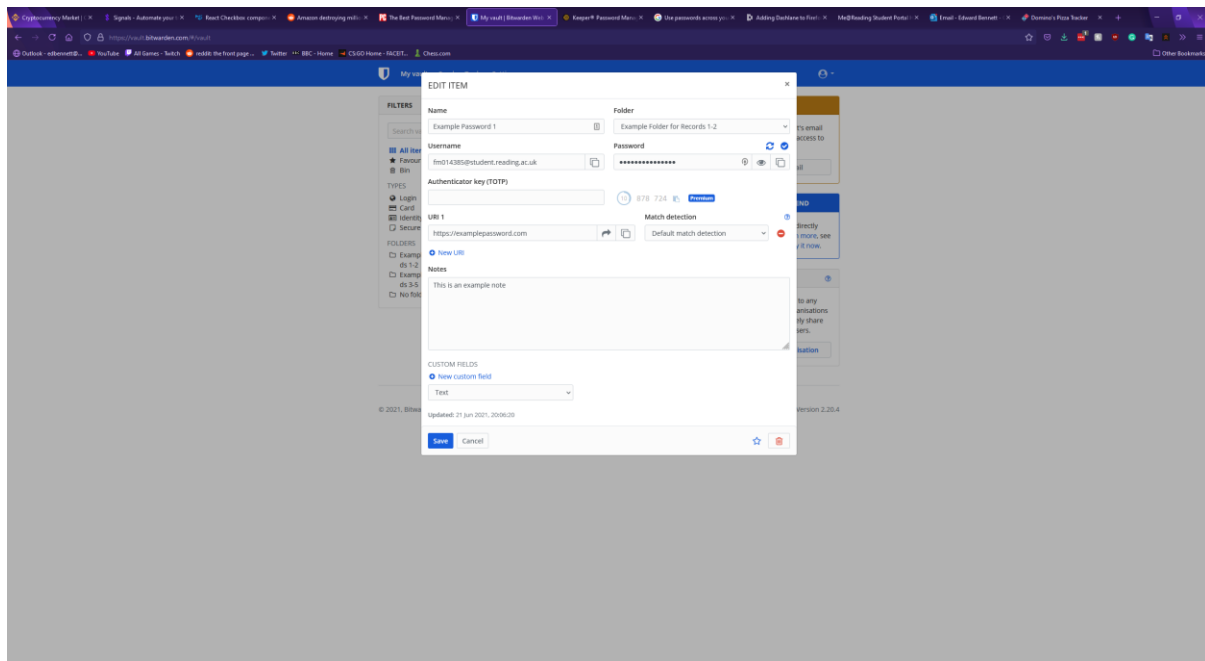


Bitwarden

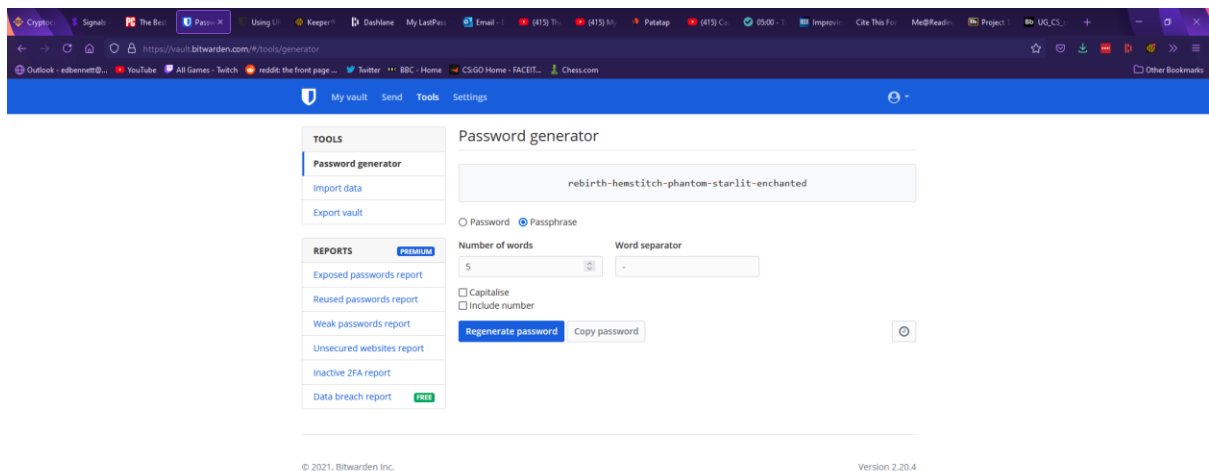
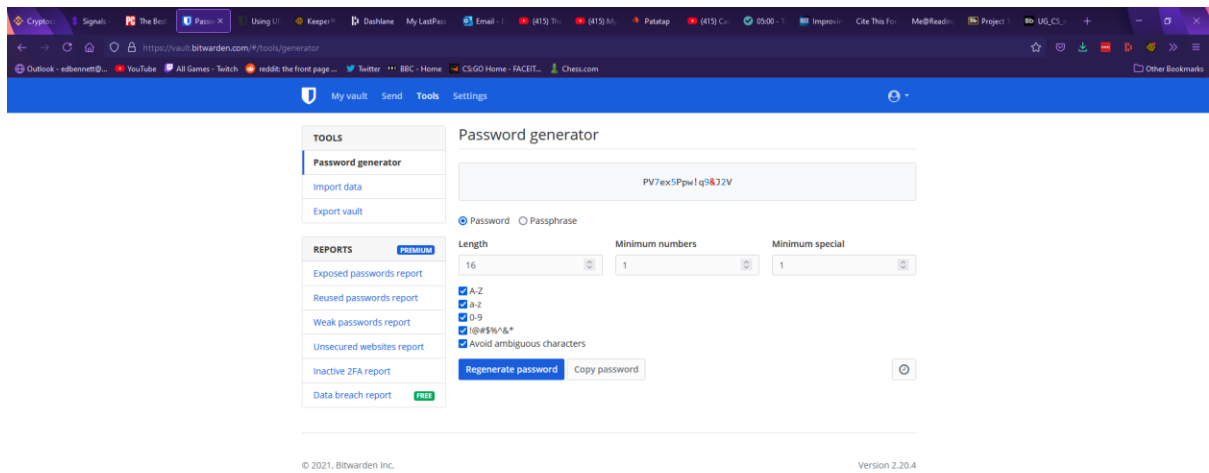
Bitwarden is a free open-source password manager. Bitwarden's vault page has much more wasted space than Keeper's, with records being listed in a centre column. Various filters and folders are shown in a panel to the left, with no obvious indication on the password records themselves which folders they belong to. To open a folder the folder must be clicked on the left panel, which narrows down the list of records to only those records which are assigned to the selected folder, making Bitwarden's implementation of a folder system more similar to a filter system. Keeper's folder system appears superior in this regard, as multiple folders can be open and viewed simultaneously. Bitwarden's main vault page is shown below.



To view a record, one must click on the record in the centre column to expand information about the record. The record window in Bitwarden is much simpler to edit, with each field simply being a text box which can be typed in. Any changes are discarded unless a highlighted save button is clicked at the bottom. Bitwarden supports most of the standard fields, with fields for a record name, a username, password, and a notes section. Much like Keeper, Bitwarden supports custom fields, with the user able to choose from a text field, a hidden field, or a Boolean field. The records page is more upfront than Keeper's, with options being immediately visible as links and buttons rather than hidden behind drop-down menus, potentially making the records page easier to navigate for a less experienced user. Bitwarden also supports a field for authentication keys, much like Keeper. Instead of URLs, Bitwarden uses a uniform resource identifier (URI) system, supporting multiple identifiers such as URLs, IP addresses, or mobile app package IDs. This allows for a greater range of identifiers to be used but may be confusing for a less knowledgeable user. A password record in Bitwarden is shown below.



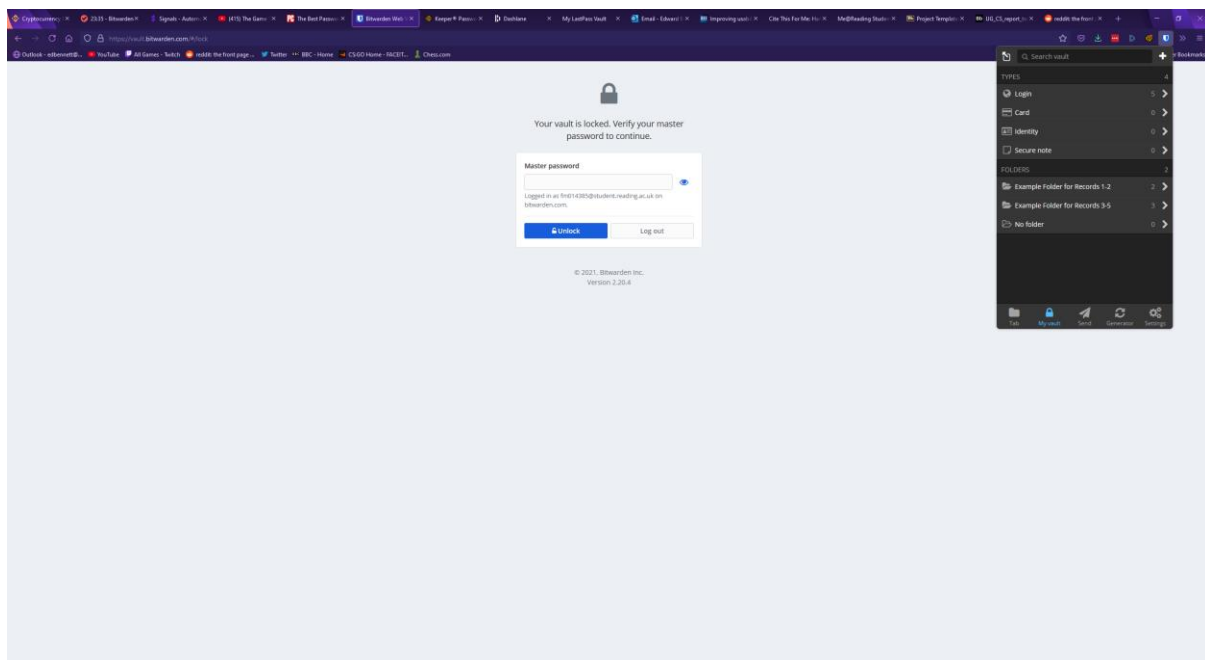
With regards to password generation Bitwarden's record page lacks behind Keeper in terms of customisation. Just above the field for a password a button offers to generate a new password, however there are no settings for this generation. A randomised password including lowercase letters, uppercase letters, numbers, and symbols is immediately generated and filled in the password field. In addition to the records page, however, Bitwarden has a tools page which contains a more detailed password generator, offering a checklist of character types to include along with a customisable length, a minimum number of numbers and symbols, and an option to avoid ambiguous characters. Bitwarden also includes the option to generate a passphrase instead, randomly selecting a set number of words and putting them together. Nielsen et al believe "One way to deal with the failings of password based authentication systems is to increase the length without necessarily increasing the complexity of recalling the password". Passphrases are then suggested as "Most people find it easier to remember passphrases, consisting of real sentences, than passwords created from random characters" (Improving usability of passphrase authentication, 2014). This suggests that Bitwarden's passphrase generator could be more effective for password security than a standard password generator. The more detailed password generator is shown below, followed by the passphrase generator.



Under the remainder of the tools page Bitwarden offers features to import and export password data, and premium features to analyse whether stored passwords are weak or have been exposed. Under the settings page Bitwarden first offers account settings, with options to change the account name and password hint at any time, with the master password and the account email only being changeable once the current master password has been entered to confirm. Settings are also offered to change the encryption of the account, such as changing the number of key derivation function iterations. The second section of the settings page refers to customisation options and allows the user to set an

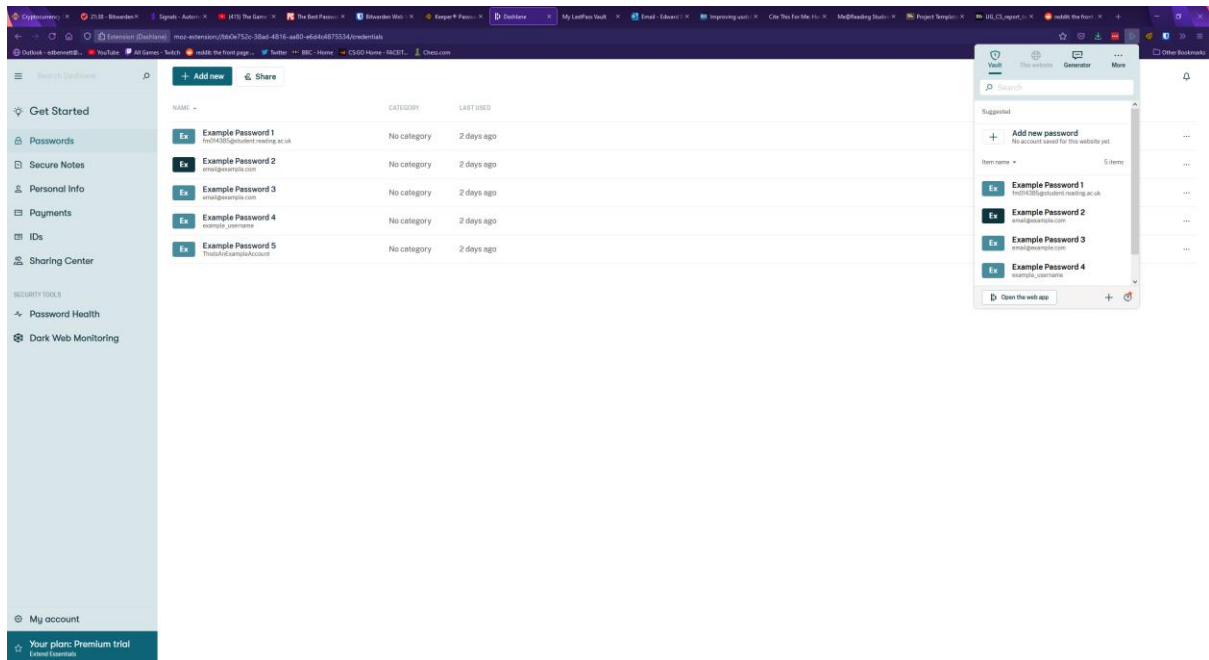
interval after which their account will be locked and will require the master password to be re-entered. This page includes a few other miscellaneous settings such as setting an avatar for the account. In a third “organisations” tab Bitwarden allows the user to add groups they can share passwords with. The final tabs of the settings page offer billing information, emergency contacts, and two-factor authentication. Bitwarden’s two-factor authentication supports various mobile apps such as Google Authenticator and physical security keys such as Yubikey.

Bitwarden also offers a browser extension, which shows any logins for the website open in the current tab, along with tabs to search the entire vault, generate a new password, and change settings for the extension. While the Keeper extension placed an icon next to any login forms, Bitwarden’s extension can autofill a login form by right clicking on a username or password field and selecting autofill. An option is also given to generate a password and copy it to the clipboard; however, the extension does not automatically fill the field. This implementation is less obvious than Keeper’s icon, and many users may not find this feature. Upon logging in to a new account Bitwarden does not have a record for, a banner appears at the top of the tab offering to automatically add the account to the user’s vault. Bitwarden’s extension is shown below.

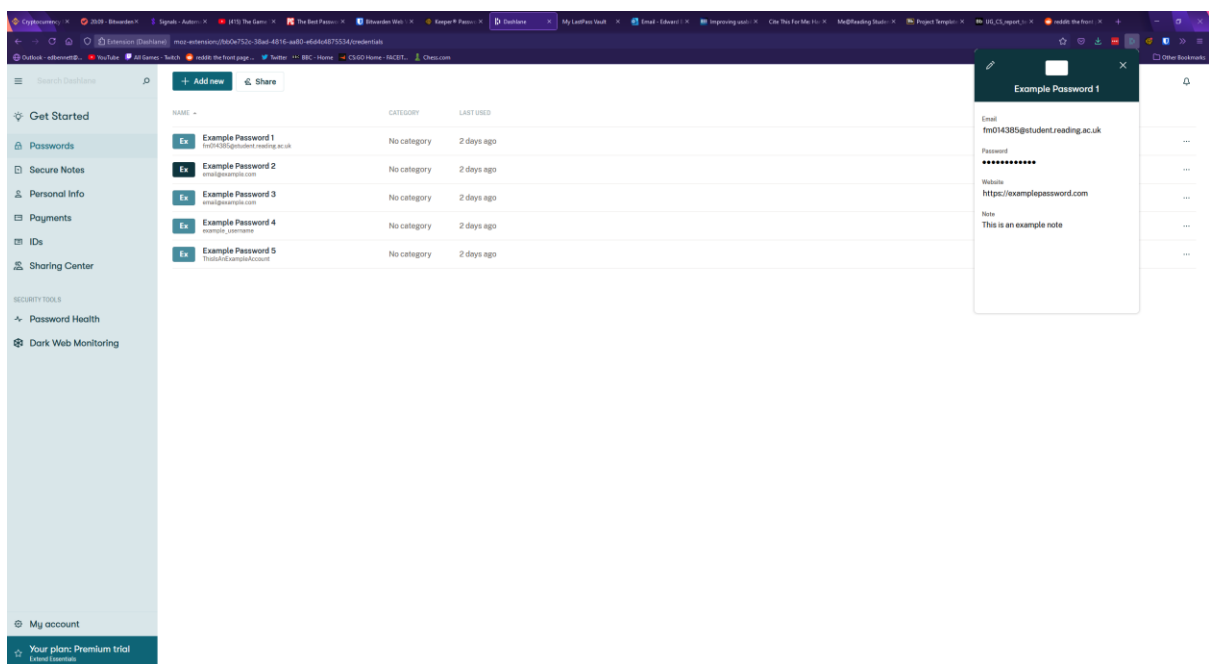


Dashlane

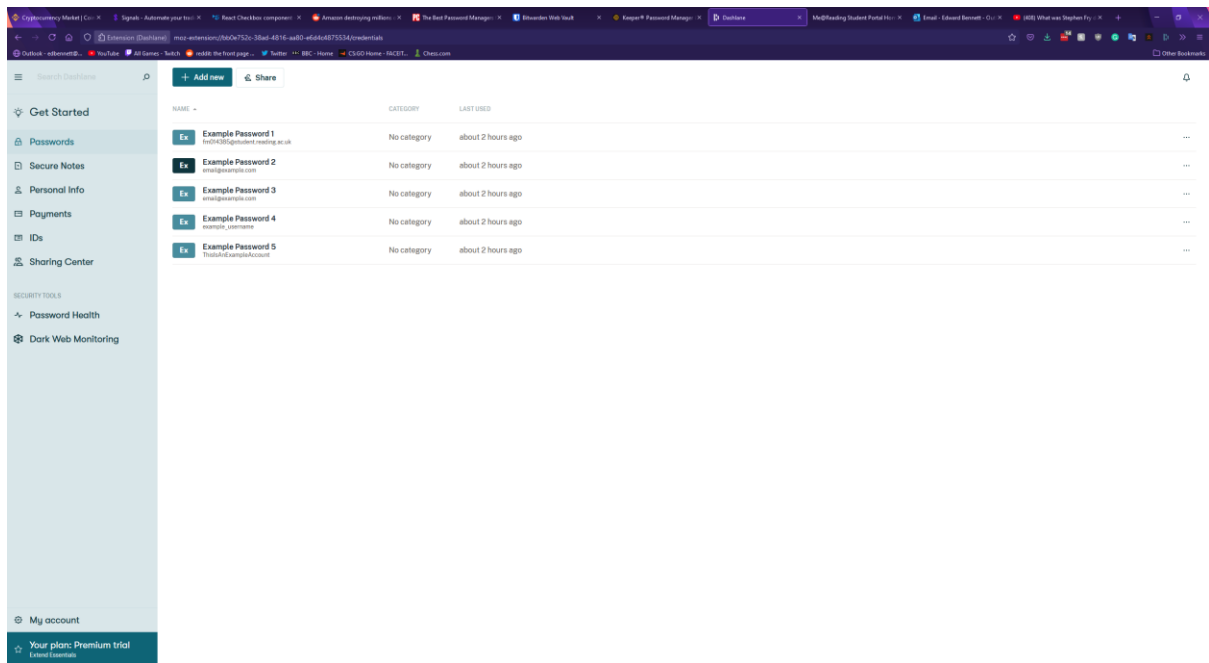
Dashlane is a password manager that functions through a browser extension. Rather than being a website which can be accessed from any device, Dashlane requires the user to have installed the extension to use it. Upon clicking the icon for the extension, it expands similarly to the Keeper extension.



The user's vault can be accessed and searched from this extension and clicking on a password will open that password record, shown below.



The screenshot displays the 1Password application interface. On the left, a sidebar contains navigation options: 'Get Started', 'Passwords', 'Secure Notes', 'Personal Info', 'Payments', 'IDs', 'Sharing Center', 'SECURITY TOOLS', 'Password Health', 'Dark Web Monitoring', 'My account', and 'Your plan: Premium trial'. The main area shows a list of passwords with columns for NAME, CATEGORY, and LAST USED. Five example passwords are listed, each with a category of 'No category' and a last used date of '2 days ago'. An 'Add new' button is visible at the top left of the list. On the right, a 'Generate' overlay is open, showing a generated password 'DR#kTockQn3Gv66s' and options to copy, adjust length, and select character types (Letters, Uppercase, Lowercase, Symbols, Numbers). The overlay also includes a 'Save as default' button and a 'Open the web app' link.



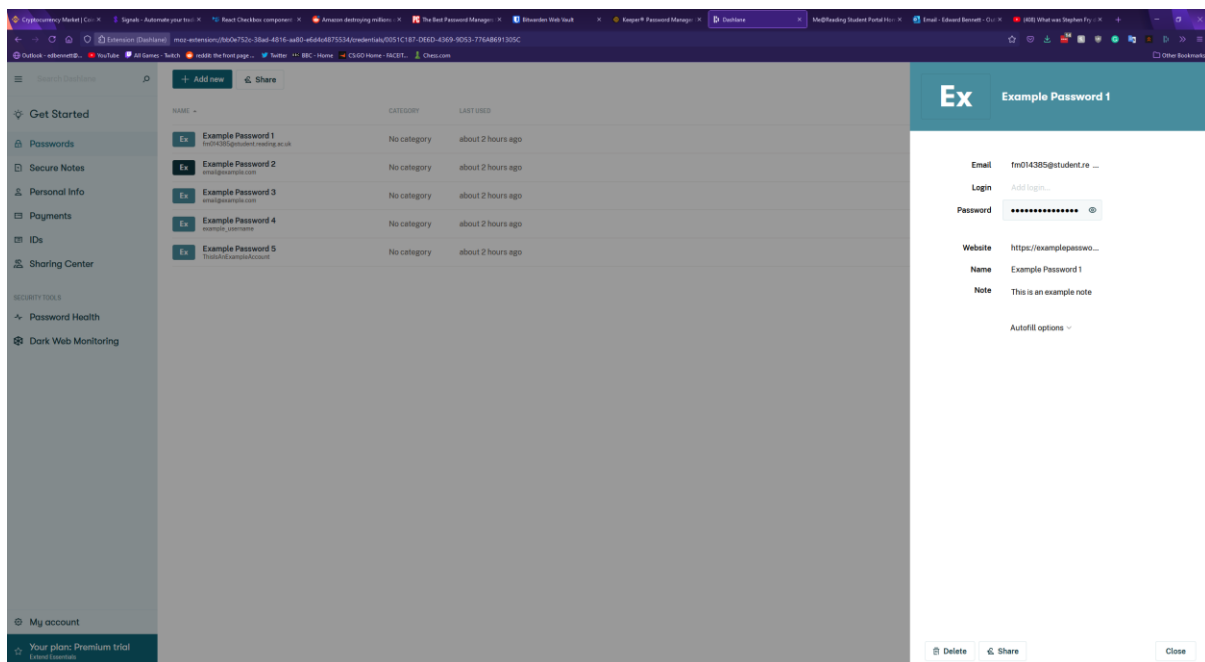
Clicking on a password record in the vault expands the record in a new panel to the right.

This panel shows all the fields of the password along with buttons at the bottom to share or delete the record. Autofill options for use with the browser extension are also available.

Interestingly, Dashlane includes separate fields for an email and a login, whereas both Bitwarden and Keeper elected to treat the email of a record as a type of username. Each field is a text box and can be edited simply by selecting the field and starting to type.

Dashlane includes no folder system, and passwords must be found simply by either scrolling through the list of all records or by searching for them. Dashlane also does not include

custom records. The Dashlane vault with an expanded record is shown below.

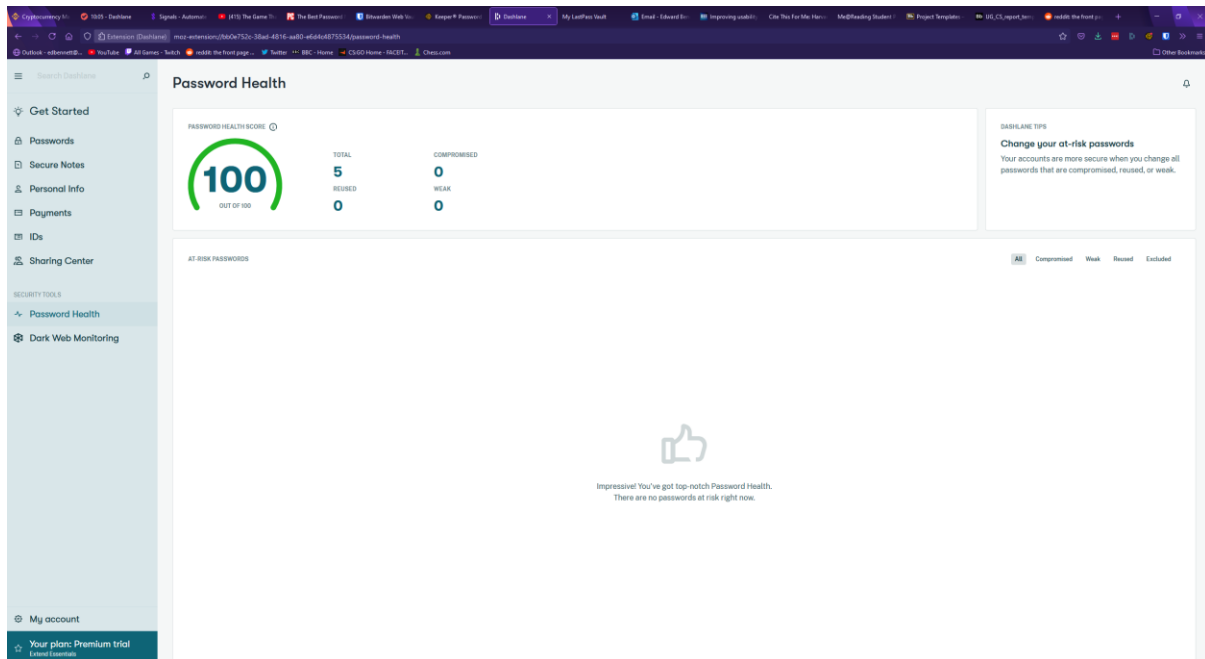


Creating a new password in Dashlane is started by clicking the green button at the top of the vault. This expands a blank record panel to the right, with empty fields. While Keeper and Bitwarden required the record to be given a name, Dashlane only requires that one field has a value, which could potentially make finding the record again later difficult. Password generation in Dashlane takes place solely in the extension, with no generation offered in the web app.

Dashlane's settings page is accessed from the "My account" button at the bottom of the sidebar which expands another panel to the right. From here users can view and update their account information, manage any devices connected to the account, import and export password data, and set up two-factor authentication for the account. Dashlane's two-factor authentication only supports biometrics and does not appear to support mobile or physical security key authenticators.

In terms of other features Dashlane supports the storage of personal information, such as emails, credit card and banking information, and IDs. Dashlane also offers a "Dark Web Monitoring" feature, promising a tool which "scans the dark web for leaked or stolen personal information". If a tracked email address is found the user will be notified so they can act. Finally, Dashlane has a "Password Health" page, much like Keeper's security audit page. This page shows a password health score out of 100, with any at-risk passwords

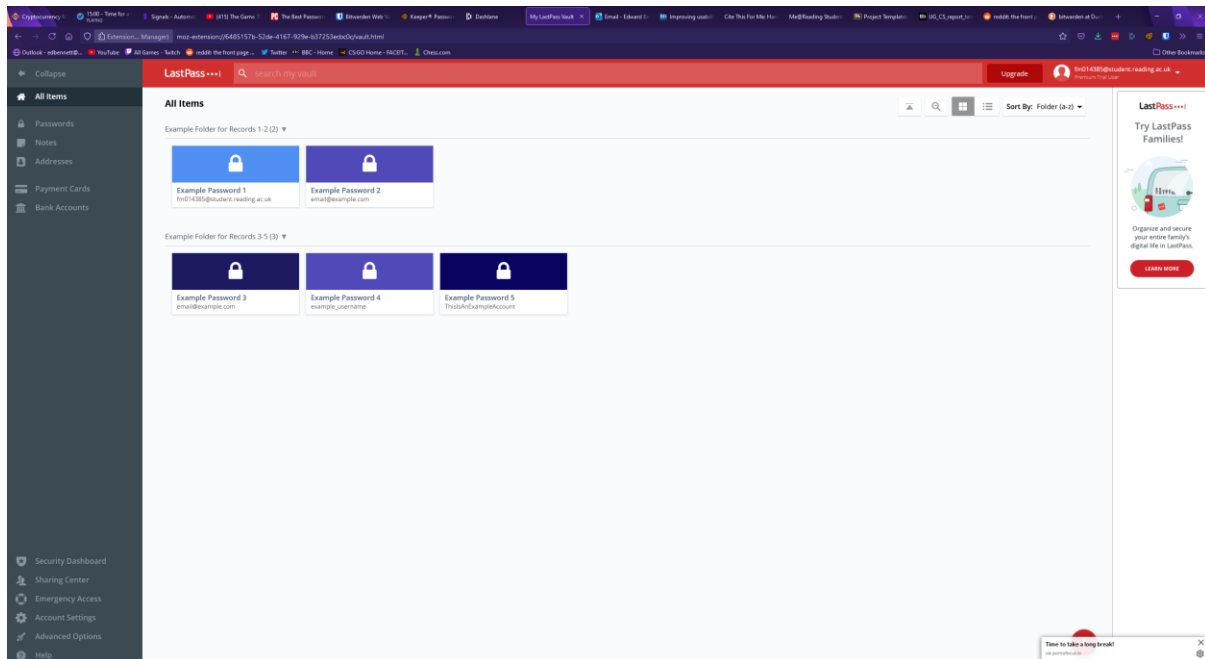
displayed below. Interestingly, Dashlane gave differing results from Keeper's security audit - though as neither service explains the metrics used to measure password security it is difficult to see why. Dashlane's password health page is shown below.



LastPass

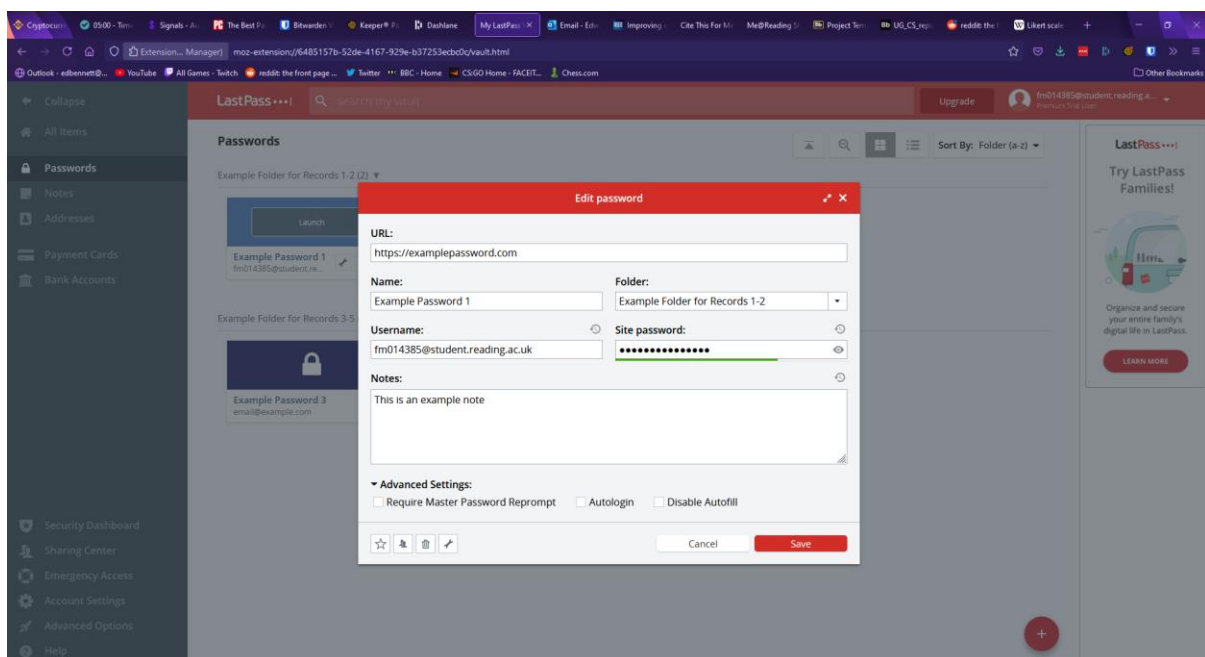
LastPass is another popular password manager which operates primarily through a web app rather than an extension. LastPass' vault page displays records primarily as cards, with various buttons to change the layout offered above the list of records. These views include a more compact view, and a list view. Records are listed in folders which can be collapsed, much like Keeper's vault design. LastPass has little wasted space, with the vault taking up the full size of the window, two sidebars, and a top navigation bar with a search bar. The top search bar narrows down the list of passwords displayed in the vault, rather than opening a drop-down menu with the search results, and next to the search bar is a button showing the user's avatar (if one has been set) and email address. When clicked a drop-down menu is expanded which offers account settings, a forum-based support centre, and a log-out button. The support centre could be incredibly valuable for a new user initially getting to grips with the software, offering a dedicated place to learn from others and ask any questions about LastPass. The right sidebar contains little content valuable to the user, only containing an advert for LastPass' other services, while the left sidebar is used for

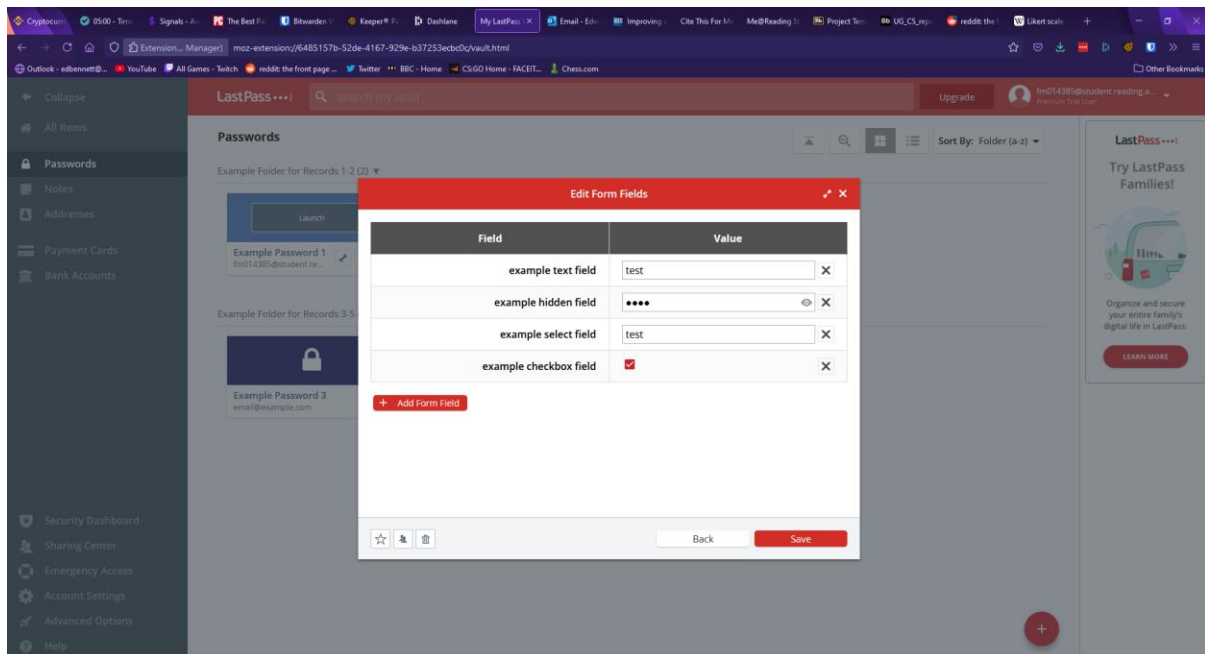
navigation, containing a list to different categories in the vault such as passwords, addresses, and payment cards. These categories are clearly labelled along with icons, which should make navigating the website as painless as possible for users. The home vault page for LastPass is shown below.



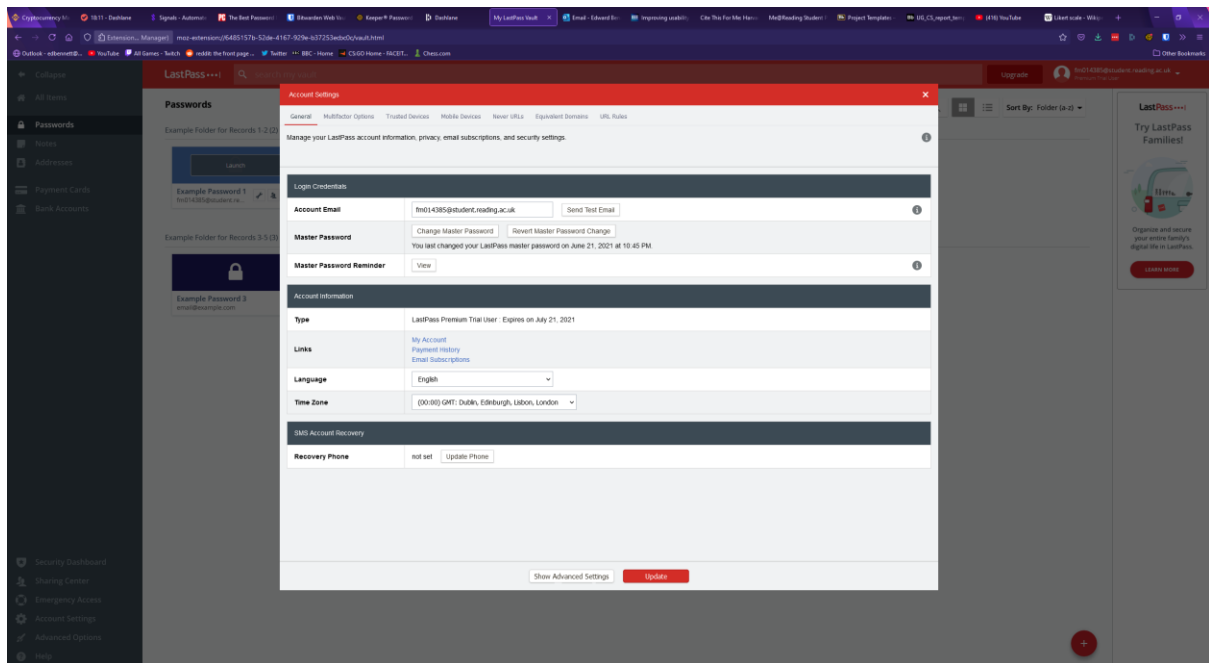
When a record in the vault is hovered over, buttons are displayed over the card. Three smaller buttons can be clicked to edit, share, or delete the record, while most of the card is dedicated to a launch button which acts as a shortcut to immediately open the record's corresponding website. This could potentially become a source of frustration for a user, as if they click the centre of the card for a record expecting to see an expanded record they would instead be redirected to a different website. Clicking on the record anywhere other than these four buttons selects the record and would allow the user to select multiple records for deleting or moving to a different folder. Clicking the edit button opens an expanded record view like the previously examined password managers. LastPass has all the standard fields, each being a textbox, with values that can be changed by simply selecting the field and typing, unlike Keeper. LastPass has a drop-down menu towards the bottom of the record window for advanced settings where the user can toggle whether their master password will be required to view the record again, and whether this webpage will be automatically logged in to. LastPass includes buttons for viewing the version history of some fields, allowing the user to view previously viewed usernames, passwords, and notes for the

record. This could be extremely useful for users who mistakenly change values in the wrong record, and now find themselves unable to reverse the changes they have made. Buttons to cancel any changes, save any changes, edit custom fields, delete the record, share the record, and mark the record as a favourite are placed at the bottom of the record window, with most of them identified by iconography. Custom fields are strangely placed behind one of these buttons, instead of being appended to the main record as they are in Keeper and Bitwarden. The placement of this feature could make it difficult for users to find or quickly refer to later. The image below shows a LastPass record followed by example custom fields assigned to the record.

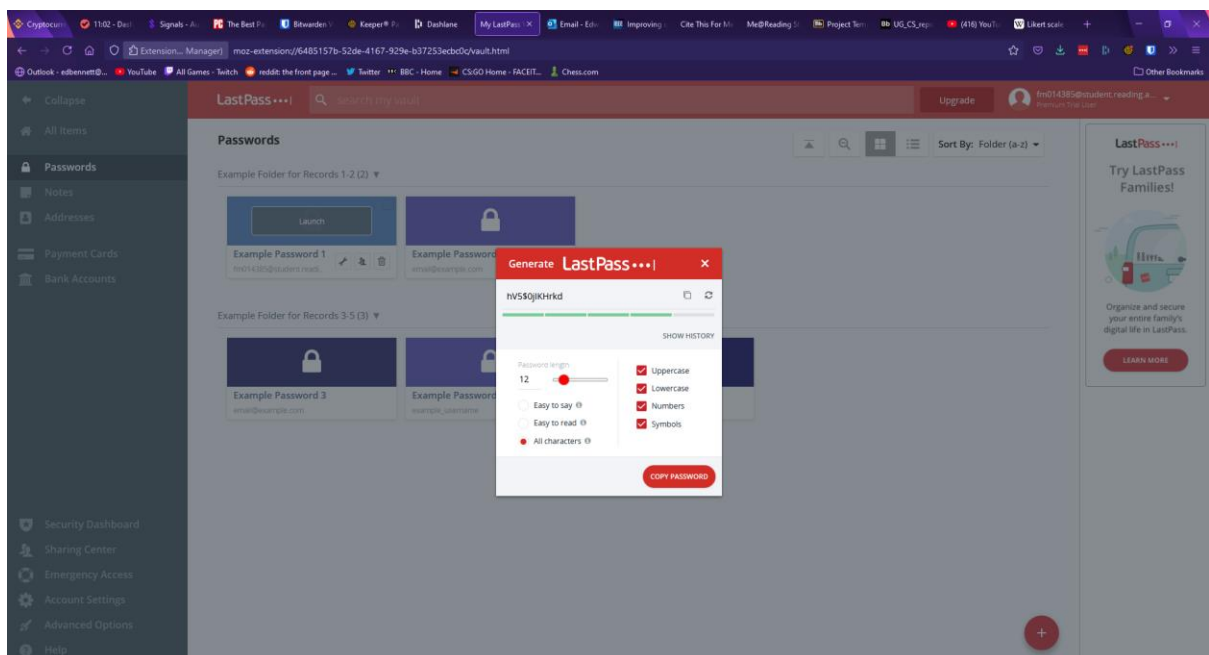




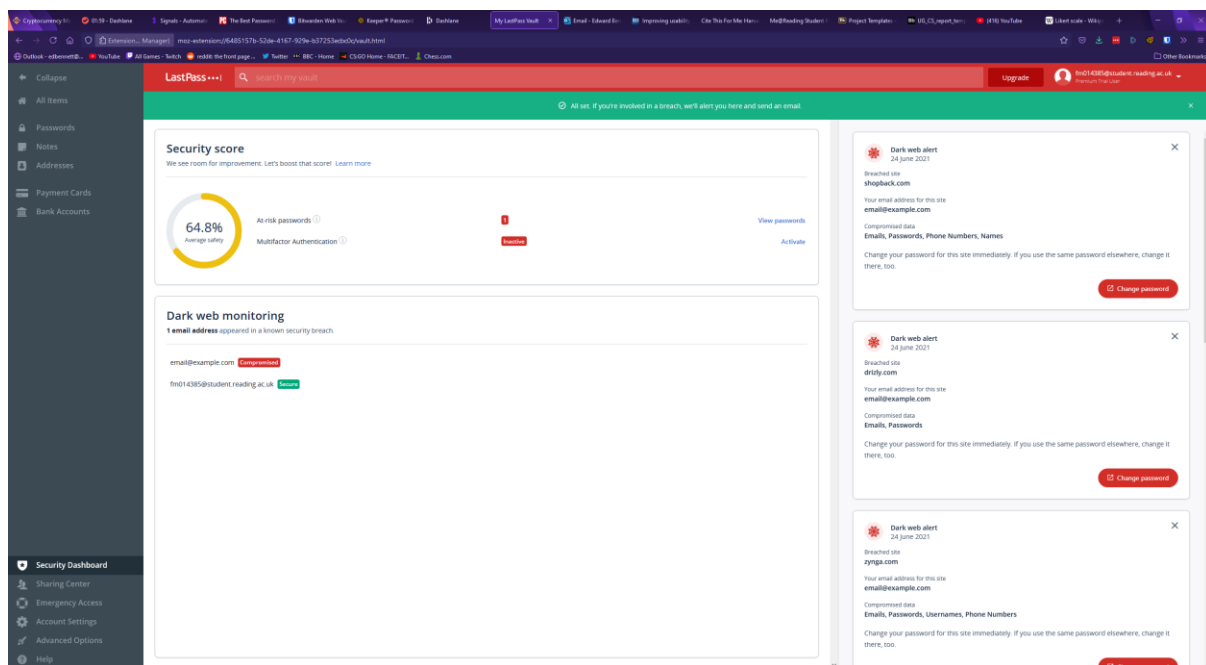
The settings page of LastPass can be accessed either from the drop-down menu on the navbar under the username or through a button on the sidebar and appears much more detailed than the other password managers'. The general page allows the user to change account information such as email, master password, and master password reminder, along with settings other account information such as a preferred language and time zone. The second tab of the settings window is for multi-factor authentication, and it lets the user set up authentication with a range of apps or physical keys. The third and fourth tabs allow the user to manage devices which have access to the account, and revoke this access, while the remaining tabs govern autofill settings, such as blocking the LastPass browser extension from filling certain websites. LastPass has a second, advanced options settings window which allows the user to perform other actions such as importing or exporting password data, upgrading their subscription, view favourite records, or generate a new password. The main settings page is shown below.



When it comes to password generation, LastPass has a panel which allows the user to select various settings. The default password length is 12 characters, with checkboxes offered to include uppercase, lowercase, numbers, and symbols. The user can customise the length of the password with either a slider or a text field, and radio buttons are offered to make the password easier to say or read by avoiding numbers, symbols, or ambiguous characters. Buttons are then displayed to allow the user to regenerate the password or copy the password to the clipboard for use. The password generator is shown below.

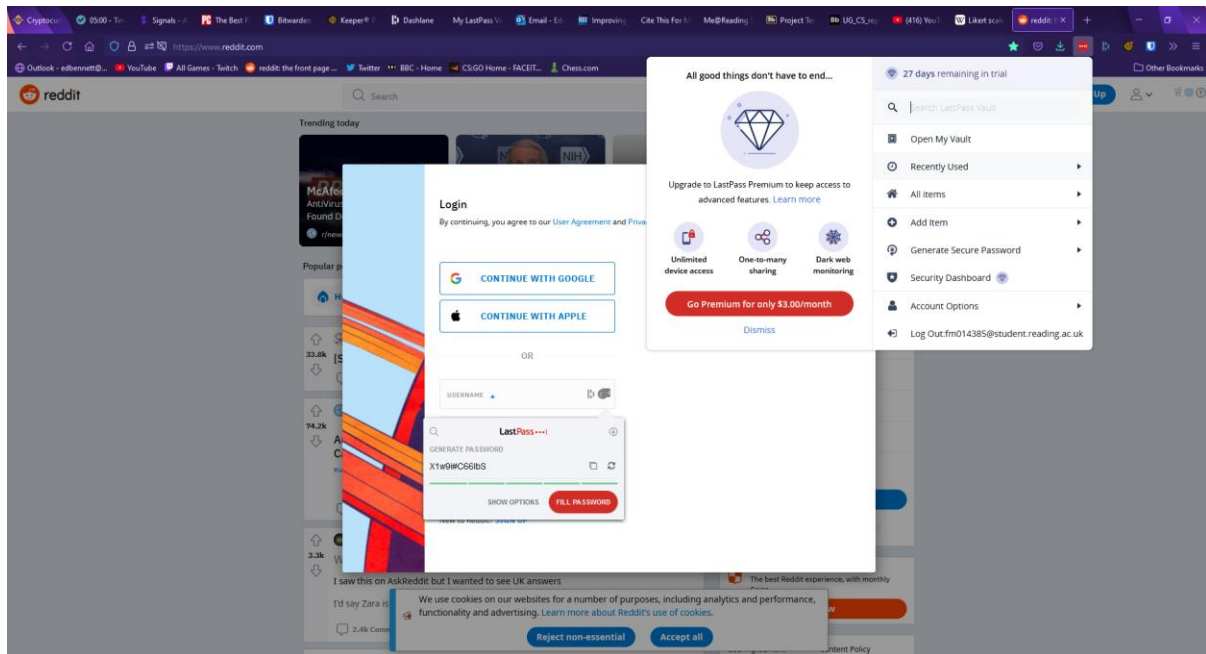


In terms of other features LastPass includes storage of private items other than passwords, including debit and credit cards, bank account information, and addresses. LastPass also includes a security dashboard, which fills a similar role to the security audit page in Keeper or Dashlane's password health page. The security dashboard shows an overall security score which rates the average safety of all passwords, along with showing any passwords determined to be at risk. For this account LastPass also suggested turning on multi-factor authentication to improve the account's security. Much akin to Dashlane, LastPass offers a dark web monitoring feature, which promises to proactively alert users "if sites from your vault are breached. Monitor these addresses. All day, every day". Upon opting into the service, the master account email address and the email address of any other records are automatically searched and identified if they appear in any known security breaches. Alerts are then shown on the right-hand side for any compromised email address, describing the security breach and the specific data that was leaked so that the user can change compromised information to protect themselves. The security dashboard is shown below.



Like every password manager seen thus far, LastPass offers a browser extension. This extension can be used to autofill login forms for websites with existing records. In a manner very similar to Keeper and Dashlane an icon is placed at the end of username or password fields. If LastPass finds any corresponding records it will display them in a list for the user to select for auto-filling. If LastPass detects no such records, it will offer to generate a new

password for a new record. Upon manually logging into a website for the first time the extension will pop up with a message offering to automatically add a record for the website with the username and password used. Clicking on the icon for the extension in the top right of the browser opens a menu with various options, including a search bar to search for records within the smaller extension window, a shortcut to open the user's vault, and password generation within the extension. The main menu of the extension is shown below.



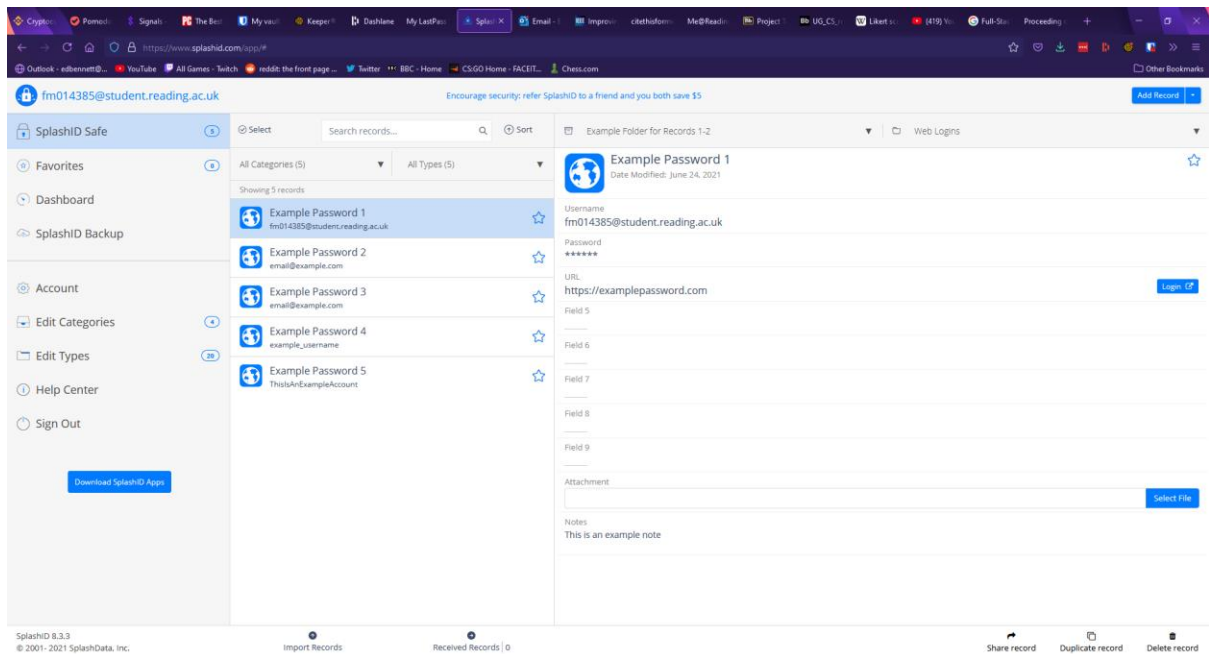
SplashID Safe

SplashID is the final password manager this report will examine. SplashID's vault page is divided into three segments, with a top and a bottom navigation bar. The top navigation bar consists of text displaying the master account email address along with a button to add a new record or category, while the bottom navigation bar contains labelled buttons to import, share, duplicate, or delete records. The first of the three main panels in SplashID's vault design is a sidebar used for navigation, with links leading to the user's main vault, a list of favourite records, and premium dashboard and backup features. The second panel shows a list of password records along with a search bar and various buttons to change how records are filtered and sorted. Each record displayed in the list is marked with either an empty or solid star to indicate whether the record has been marked as a favourite record, and the star can be clicked to toggle this state. The final, right-hand panel displays a password record in greater detail, showing the record's fields along with information about

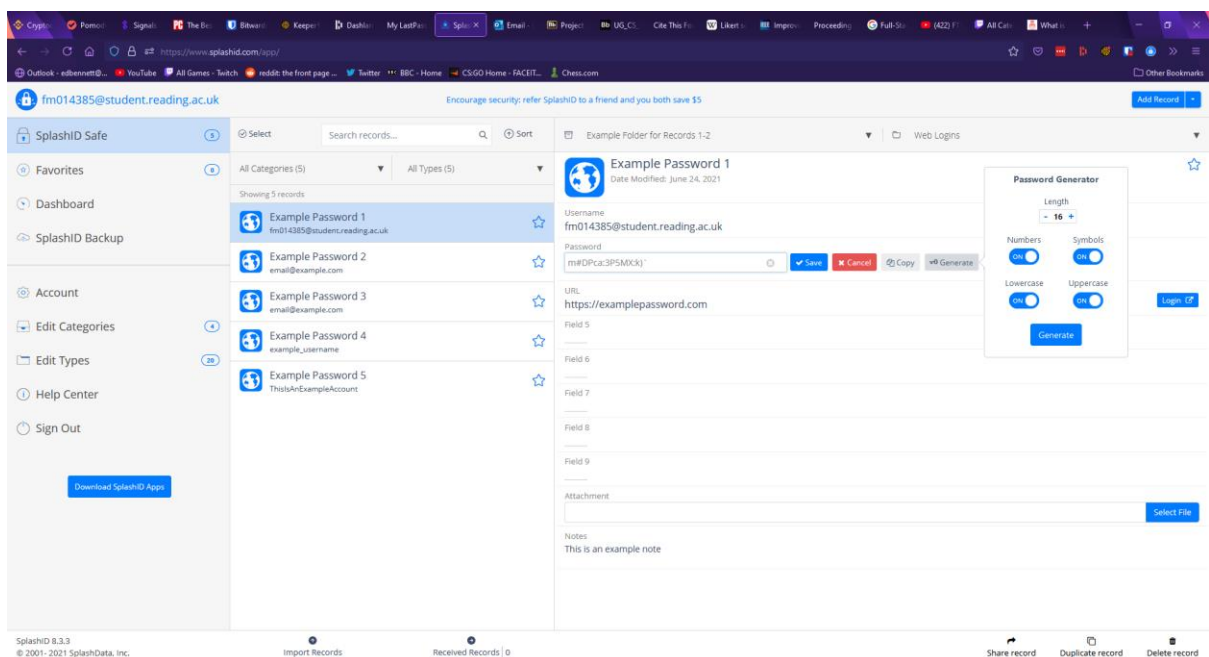
the record such as the date it was last modified and its type and category. Clicking on the type or category expands a drop-down list which allows the user to set a different type or category for the record. These types and categories act as SplashID's substitute for a folder system, with both acting as customisable assignable tags and folders used to identify records. Default suggestions for categories include business and personal, while built-in options for types include email addresses, bank accounts, and credit cards.

Perhaps confusingly, SplashID displays records with five empty fields with placeholder names. While the titles of these fields can be changed, they cannot be removed, and new custom fields cannot be added. The values of these fields can only be text values, meaning SplashID does not support the same diversity of custom field types as Keeper, LastPass, and Bitwarden. SplashID does support attachments, allowing seemingly any file type to be added to a record. SplashID also sports a login button next to the URL field of a record which can be clicked to open the corresponding URL and automatically log the user in.

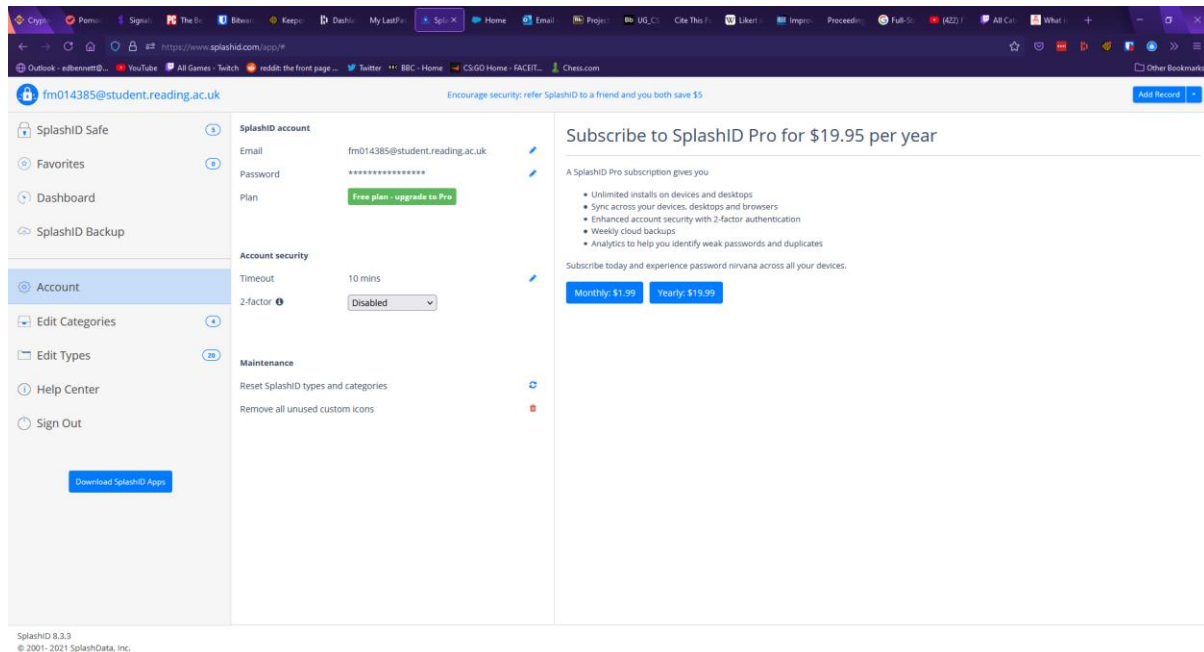
SplashID uses very little in the way of colour, keeping much of the page in shades of white and grey, while highlighting important icons and actions in blue. While appearing plain, this keeps contrast high, making the page clear and easy to read for visually impaired users, and the use of blue as the sole accent colour will help prevent confusion. The layout of the page, however, is clustered and busy which could leave users feeling claustrophobic as little space or padding is placed between items. This could be alleviated somewhat if the user were able to resize the various panels, however SplashID has not included this as an option. The main vault page of SplashID is shown below.



Password generation in SplashID is performed by clicking on a field within a password record. The field becomes an editable textbox and buttons appear to save changes, discard changes, copy the contents to the clipboard, generate a new password, and mask a field. The length of a generated password must be four characters or above with a default length of 16. Toggles are available to include lowercase, uppercase, numbers, and symbols. Clicking a highlighted “Generate” button immediately fills the selected field with the output. The SplashID password generator is shown below.



The account and settings page in SplashID is relatively bare, containing options for the user to change their master account email and password, upgrade their plan, change the length of time before the user is automatically logged out, enable two-factor authentication, and reset all types and categories. The two-factor authentication supported by SplashID is only offered through SMS text messaging or email authentication. Mobile apps and physical security keys are not supported. The settings page is shown below.



Similarly, to LastPass, SplashID also includes a link to a “Help Center”, offering the user a frequently asked questions page, and the option to submit a support form to receive help or advice from the SplashID team, although there is no forum section. This could be extremely valuable for any users who are easily confused or less experienced with password managers and being able to contact a support team member directly could especially help a user obtain an accurate and relevant answer rather than a forum system which relies more heavily on community members.

Like all previously examined managers, SplashID also offers a browser extension. This extension displays a searchable list of all records along with a tab to only show favourite records or records the extension believes are relevant to the currently open tab. The extension does not support password generation or the creation of a new record, these functions must be performed from the main web app which can be accessed through the SplashID logo, which acts as a shortcut. The browser extension is shown below.

Screenshot of the SplashID Pro web interface. The browser address bar shows <https://www.splashid.com/app/#>. The user is logged in as **fm014385@student.reading.ac.uk**. The interface includes a sidebar with navigation options: SplashID Safe, Favorites, Dashboard, SplashID Backup, Account (selected), Edit Categories, Edit Types, Help Center, and Sign Out. The main content area displays account details for **fm014385@student.reading.ac.uk**, including a password field, a plan section with a **Free plan - upgrade to Pro** button, and account security settings (Timeout: 10 mins, 2-factor: Disabled). A subscription promotion for **SplashID Pro for \$19.95 per month** is visible, listing benefits like unlimited installs, sync across devices, enhanced security, weekly backups, and analytics. A **Showing 5 records** dropdown menu is open, displaying a list of example passwords (Example Password 1 to 5) with **Login ID** and **Star** icons. The footer indicates **SplashID 8.3.3** and **© 2001-2021 SplashData, Inc.**

Solution Approach

The chosen solution for this project was to create a Javascript web app using the MERN software stack consisting of MongoDB for the database, Express for routing and HTTP requests, React as the client-side frontend framework, and Node as the server runtime. NestJS, a Javascript framework was selected to be used in the construction of the backend to provide advantages such as “a level of abstraction above these common Node.js frameworks” along with “[exposing] their APIs directly to the developer” (Mysliwiec, n.d.). To integrate MongoDB with NestJS, a MongoDB object modelling tool for Node called Mongoose was chosen. Mongoose focuses on schema-based solutions to model data, each of which maps to a MongoDB collection (Mongoose v5.12.15, n.d.). To handle user authentication, a library called Passport was chosen as NestJS supports Password with a dedicated module. Passport can be used to authenticate users by defining strategies which determine when authentication succeeds or fails (Hanson, n.d.), and for this project users will be authenticated via JSON Web Tokens, an open industry standard used to securely transmit data using digitally signed tokens (auth0.com, n.d.). Much of the backend will be designed as a RESTful API, a type of API which meets 6 criteria: that client and server concerns are separated; that the API is stateless, meaning each request from the client to the server must contain all necessary information without using stored context on the server; that data is labelled as cacheable or non-cacheable so that data can be reused for equivalent requests; that there is a uniform interface between components; that the system is hierarchically layered; and that code is available on demand, extending client functionality by allowing the downloading and executing of code as scripts (Fielding, 2000). A significant portion of the project will be programmed using TypeScript, a language which compiles to Javascript.

The web-client will then consist of sign-up and login pages which direct the user to a main dashboard. This dashboard will act as a central recognisable page for the user, off which many components and most of the features will be usable.

Requirements Engineering

To create my list of requirements the MoSCoW prioritisation method was used. This method is used to identify the features that a project can include, from the most important features the project must have, to the least important features the project could have. The MoSCoW technique prioritises requirements based on the following criteria, “must have”, “should have”, “could have”, and “won’t have this time”. Must have describes requirements that must be included in the final product. Should have defines requirements which are high priority and should be included if there is time. Could have requirements are “desirable or nice to have”, but are the first to be dropped if the project will not be finished in the timeframe. Finally, “Won’t have this time” describes features a stakeholder would desire, but a consensus is reached that the requirement will not be implemented at this time (Ahmad et al., 2017).

In the list below various features have been identified as features the project must have, should have, could have, or features the project will not have. These features have been identified based on the review of existing password managers, along with the literature review.

Mandatory features (Must have):

- Users are able to store usernames and corresponding passwords
- Users are able to retrieve usernames and corresponding passwords
- Users are able to delete entries
- Users are able to search for specific entries
- Users are able to create an account
- Users are able to log into an existing account with their username and password
- Users are able to update their account details
- A password generator tool

Desirable features (Should have):

- A folder system can be used to organise records
- A master password hint can be used to remind the user of a forgotten password

- The user's password is stored as a hash
- The passwords on the database are encrypted

Optional features (Could have):

- If a user makes an unsuccessful attempt to log in a certain number of times in a row, they are locked out of their account for a certain length of time
- Verification email upon sign-up

Excluded features (Will not have):

- Security evaluator
- Two-factor authentication
- Biometric authentication
- Physical security key authentication (such as Yubikey)

Implementation

The construction of this project began with the backend, particularly the MongoDB database. Once the fundamentals and essential files had been set up using NestJS various schemas were designed with Mongoose to encapsulate the functionality that would be required of the database as identified during requirements engineering. Schemas are used with Mongoose to define a model, an instance of which is a document – a term MongoDB uses for records which consist of field-and-value pairs (MongoDB.com, n.d.). The user and password schemas consisted of the properties a user and a password record would need to be functional and can be seen below.

```

1  import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
2  import { Document } from 'mongoose';
3  import { Role } from 'src/roles';
4  import { enumValues } from 'src/utils';
5
6  export type UserDocument = User & Document;
7
8  @Schema()
9  export class User {
10     @Prop({ required: true })
11     email: string;
12
13     @Prop({ required: true })
14     username: string;
15
16     @Prop({ required: true, enum: enumValues(Role), default: Role.User })
17     role: Role;
18
19     @Prop({ required: true })
20     passwordHash: string;
21
22     @Prop()
23     passwordHint: string;
24
25     @Prop({ required: true, default: false })
26     hasVerifiedEmail: boolean;
27
28     @Prop()
29     secretQuestions: { question: string; answer: string }[];
30
31     @Prop({ required: true, default: 0 })
32     failedAttempts: number;
33
34     @Prop()
35     lastAttemptDate: Date;
36 }
37
38 export const UserSchema = SchemaFactory.createForClass(User);
39 |

```

```

1  import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
2  import { Document } from 'mongoose';
3
4  export type PasswordDocument = Password & Document;
5
6  @Schema()
7  export class Password {
8    @Prop({ type: 'ObjectId' })
9    userID: string;
10
11    @Prop()
12    password: string;
13
14    @Prop()
15    name: string;
16
17    @Prop()
18    URL: string;
19
20    @Prop()
21    username: string;
22
23    @Prop()
24    notes: string;
25  }
26
27  export const PasswordSchema = SchemaFactory.createForClass(Password);
28

```

With schemas for the database designed, the next step was to begin work on the endpoints. When a client communicates with the backend, the requests made by the client will reach a certain endpoint, and the server must perform certain actions depending on the nature of the request and the endpoint it was sent to. In NestJS, endpoints are built using controllers, providers, and modules. Controllers are responsible for handling incoming requests and returning responses to the client (Mysliwiec, n.d.). In essence, the controller receives the requests, determines the correct service needed, and makes the correct service calls in the providers before returning any responses or HTTP status codes to the client.

The service, on the other hand, acts as a provider for the complex requests, carrying out the request and performing the necessary work. Controllers and services were made for user and password endpoints, with the user endpoint being used to deal with CRUD operations for users and the password endpoint being used to deal with CRUD operations for an

individual user's passwords. As an example, the first two decorators from the user controller are shown below.

```
17 @Controller('api/users')
18 @ApiTags('User')
19 export class UserController {
20     constructor(private readonly userService: UserService) {}
21
22     @Get('')
23     @Roles(Role.Admin)
24     @UseGuards(AuthGuard(JWT_STRATEGY), RolesGuard)
25     async getAllUsers(@Res() response: Response): Promise<void> {
26         const result = await this.userService.getAllUsers();
27         response.status(HttpStatus.OK).send(result);
28     }
29
30     @Post('')
31     @Roles(Role.Admin)
32     @UseGuards(AuthGuard(JWT_STRATEGY), RolesGuard)
33     async createUser(
34         @Body() body: UserCrudDto,
35         @Res() response: Response,
36     ): Promise<void> {
37         response
38             .status(HttpStatus.CREATED)
39             .send(await this.userService.createUser(body));
40     }
```

Both of these decorators handle requests made to the /api/users endpoint, however the first decorator deals with HTTP GET requests and the second deals with HTTP POST requests. Both consist of an asynchronous function which calls a certain function in the user service. While the GET decorator does not need to pass any additional information to the service function, the POST decorator passes the body of the response. The two code snippets below show the corresponding functions in the user service. The getAllUsers function simply finds every user and returns them, while the createUser function performs a little more business logic. The createUser function uses the body of the request to create a new userModel, which is used to map the request to the format of a user which can be saved to the MongoDB database.

```

17  /**
18   * Get all users in the database, only executable by an admin
19   */
20  async getAllUsers(): Promise<UserDto[]> {
21    const result = await this.userModel.find().exec();
22    return result.map(fromUser);
23  }

```

```

43  /**
44   * Create a new user and save it to the database
45   * @param {UserCrudDto} createUserDto A DTO of the new user to be saved
46   */
47  async createUser(createUserDto: UserCrudDto): Promise<string> {
48    const createdUser = new this.userModel(createUserDto);
49    const result = await createdUser.save();
50    return result._id;
51  }

```

It should be noted that in the `createUser` function the body of the POST request is used as a `UserCrudDto`. Data transfer objects (DTOs) are objects which define how data will be sent over a network (Mysliwiec, n.d.) and were created to describe the properties of an object sent over the network. Three DTOs were outlined for users and passwords, each following the same pattern. The `UserBaseDto` contains only properties used by both of the other DTOs, which for the user is a username and an email. The `UserCrudDto` contains properties needed to perform CRUD operations and extends `UserBaseDto` but also includes password and passwordHint properties. Finally, the `UserDto` extends `UserBaseDto`, however, it additionally includes required id and role properties which will contain a unique ID and a role - both of which will be assigned. The `UserDto` also contains a function which can be used to map from a `UserDocument` - defined by the user schema - to a `UserDto`. The password DTOs follow suit, with the `PasswordBaseDto` containing reused properties, the `PasswordCrudDto` containing additional properties which can be used to perform CRUD operations, and the `PasswordDto` containing an id property and a function to map from a `PasswordDocument` to a `PasswordDto`. The user DTOs can be seen in the three code snippets below.

```

1  export interface UserBaseDto {
2    username?: string;
3    email?: string;
4  }
5

```

```

1  import { UserBaseDto } from './user-base.dto';
2
3  export interface UserCrudDto extends UserBaseDto {
4      password?: string;
5      passwordHint?: string;
6  }
7

```

```

1  import { Role } from 'src/roles';
2  import { UserDocument } from 'src/schemas/user.schema';
3  import { UserBaseDto } from './user-base.dto';
4
5  export interface UserDto extends UserBaseDto {
6      id: string;
7      role: Role;
8  }
9
10 export function fromUser(user: UserDocument): UserDto {
11     return {
12         id: user._id,
13         username: user.username,
14         email: user.email,
15         role: user.role,
16     };
17 }
18

```

With the endpoints constructed, the next step was to handle authentication and authorisation. Authentication involves ensuring that a user is who they claim to be, and authorisation involves ensuring that a user can only perform actions they should be allowed to perform.

Authentication:

Ensuring users are who they say they are

Auth strategy calls validate method in auth service

Auth service compares user against password hash

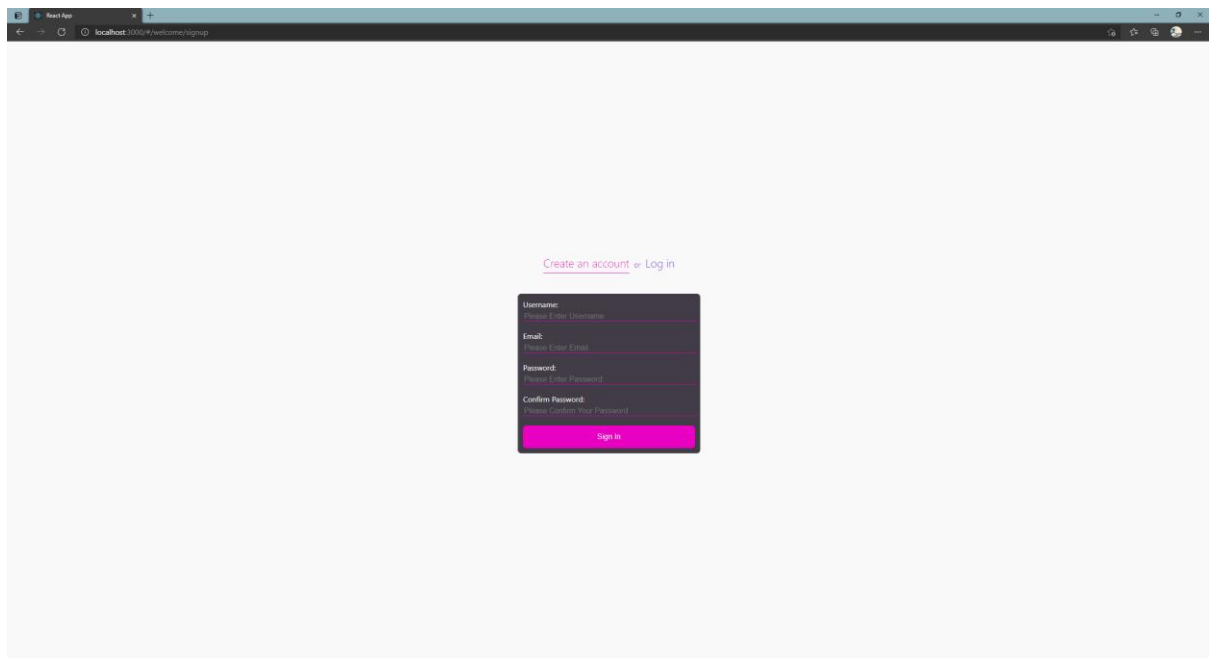
JWT tokens assigned to user once they are authenticated

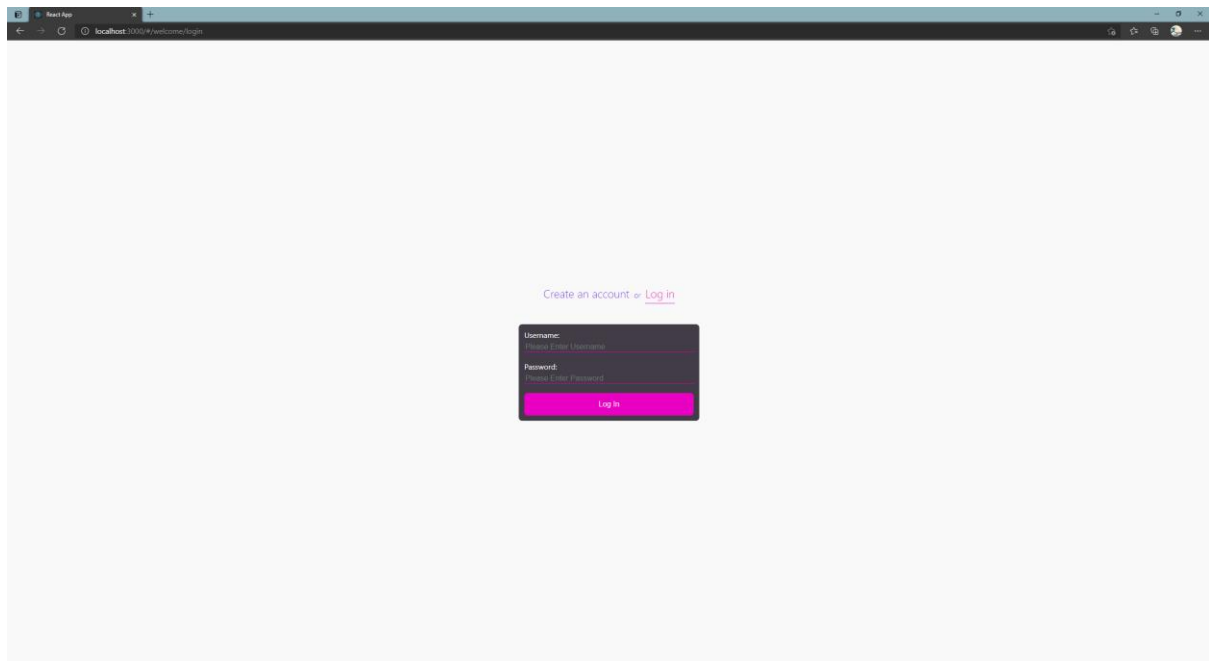
Authorisation:

Ensuring users can only do what they should be allowed to do

Roles guard

With the backend mostly complete, work shifted to focus mainly on the frontend of the app. Due to the use of the MERN software stack, the frontend was mainly produced through React, JSX, HTML, and CSS. To begin construction and to gain familiarity with frontend development, sign-up and login forms were first created. These forms were created using HTML and CSS, and represent initial experimentation with regards to colours, fonts, and other styles. The forms were initially created on a dark card background on a larger white background. Above the forms was a title which also acted as a React router NavLink component, allowing the user to swap between the sign-up and login forms, with the currently selected form being highlighted in a different colour. The forms themselves consisted of wide labelled textboxes above an equally wide highlighted submit button. The text fields were labelled and contained placeholder text to avoid any potential confusion. The first iteration of these forms can be seen below.





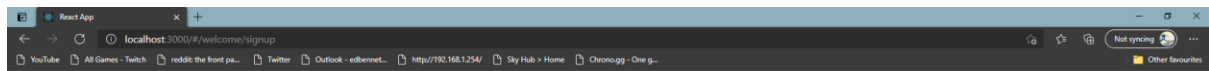
The second iteration of these forms was designed using Material-UI, a component based React library which is an implementation of Google's Material Design. Material Design is a design system created by Google to allow for easier implementation of cohesive, accessible, project-wide theming. While the developer can set some global themes such as fonts and colours, Material-UI aids accessibility and ease of use by calculating whether text or icons on a surface should be black or white to maximise contrast and legibility.

This second iteration of the forms consisted of the same NavLink title with the forms inserted below as components. The sign-up and login components themselves were built using Material-UI's TextField and Button components. Each field was clearly labelled and contained initial placeholder text to make its purpose clear to the user, and required fields are marked with an asterisk. Input validation was then completed, for which a React library called Formik was used, "the world's most popular open source form library for React" (Palmer, n.d.). Formik was used in conjunction with Yup, a Javascript schema builder which Formik documentation recommends. Once a validation schema was constructed with Yup, Formik could then be used to perform input validation and handle submission to the backend. A code snippet from the sign-up component is included below to exemplify this use of Yup and Formik. In the snippet below, every field except the password hint is required; the email field is validated to check the input's format meets that of an email

address; the password field must contain at least 8 characters; and the second password field must match the first.

```
119 const signUp = withFormik({
120   mapPropsToValues: ({ username, email, password, confirmPassword, passwordHint }) => {
121     return {
122       username: username || "",
123       email: email || "",
124       password: password || "",
125       confirmPassword: confirmPassword || "",
126       passwordHint: passwordHint || "",
127     };
128   },
129
130   validationSchema: Yup.object().shape({
131     username: Yup.string().required(" "),
132     email: Yup.string().email("Invalid email address").required(" "),
133     password: Yup.string().min(8, "Password must contain at least 8 characters").required(" "),
134     confirmPassword: Yup.string()
135       .required(" ")
136       .oneOf([Yup.ref("password")], "Passwords do not match"),
137   }),
138
139   handleSubmit: (values, { setSubmitting }) => {
140     setTimeout(() => {
141       // submit to the server
142       setSubmitting(false);
143     }, 1000);
144   },
145 })(form);
146
147 export default signUp;
148
```

Material-UI's TextFields come with properties to mark whether an input causes an error, and this was used in conjunction with Formik so when a field has been touched without leaving a valid input the field changes colour to red to indicate to the user that there is a problem with their form values. Where applicable, helper text also appears under the field to elaborate on the error for the user. This was used under the email and password forms to indicate that the user's email address was invalid, or their password was too short. The second iteration of the sign-up and login forms are shown in the two screenshots below.



Create an account or Log in

Username *

Email Address *

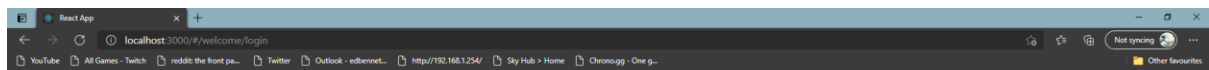
Password *

Confirm Password *

Password Hint

SIGN UP

localhost:3000/#/welcome/signup



Create an account or Log in

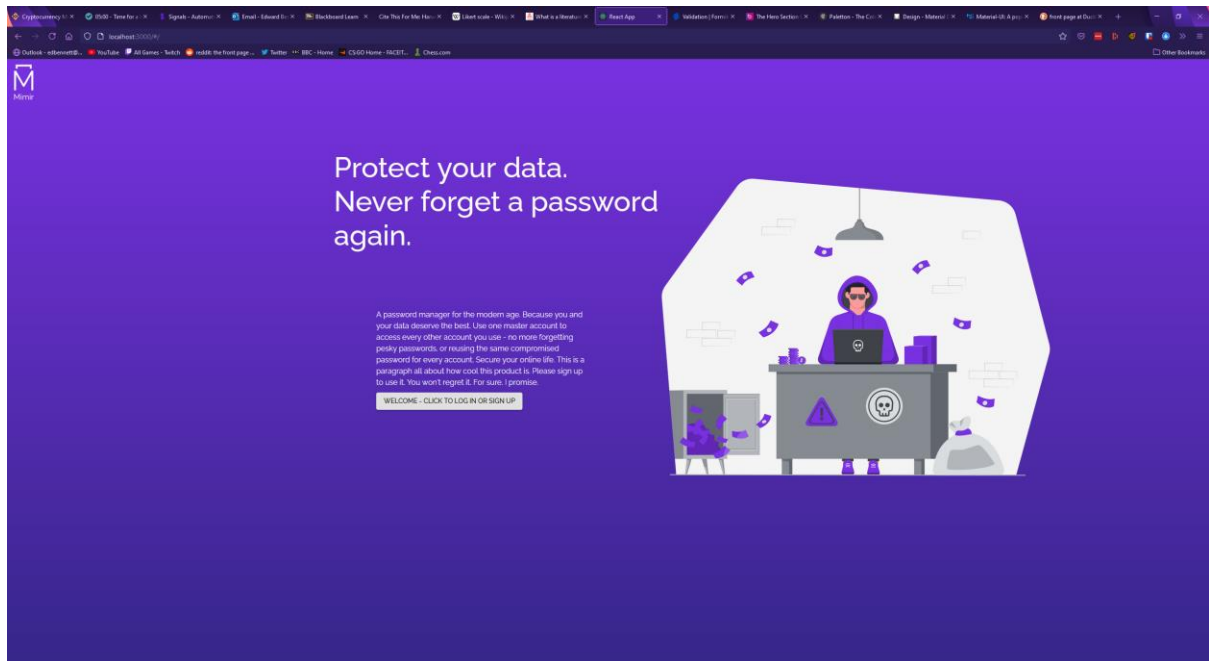
Username *

Password *

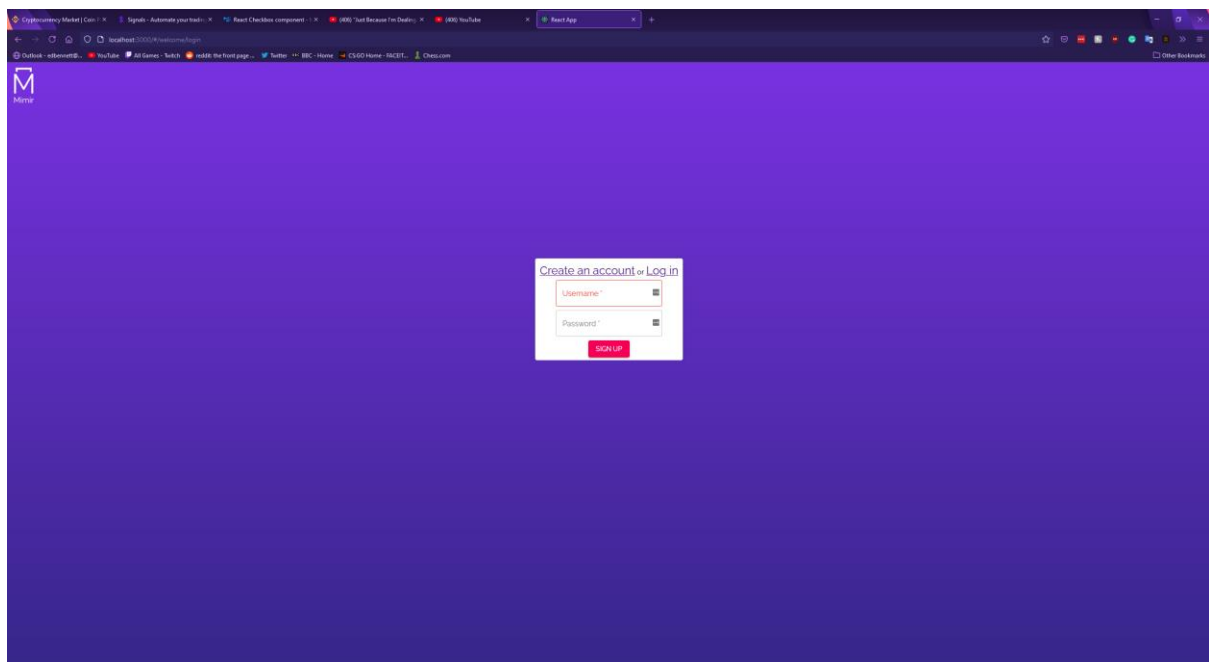
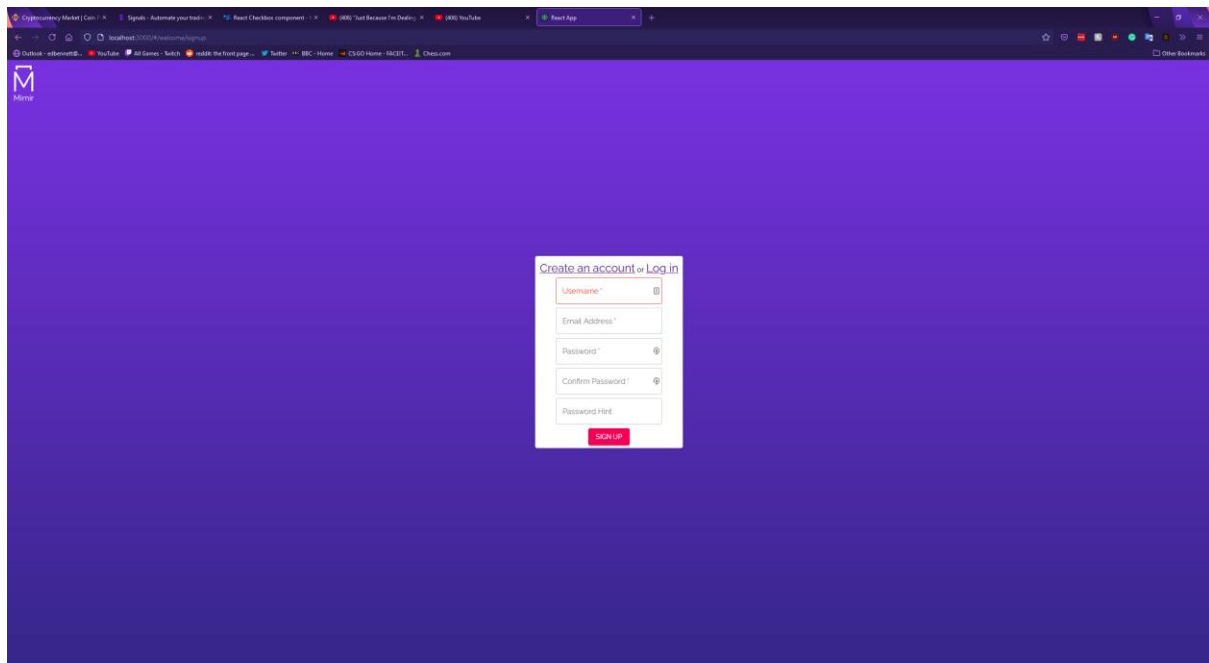
SIGN UP

With the above forms brought to a relatively functional state, work began on the front page of the website. The front page of the website is used to provide to the user some context about the password manager, a brief explanation of a password manager's purpose, and offer them a way to sign up. The front page was designed as a simple hero section, with large white title text which stands out against a strong purple gradient background. This text consists of two short sentences designed to capture the user's interest. Below this title text an image and a paragraph are placed either side of the centre of the window. The image is a stock image used to represent cyber-crime and hackers who may try to steal user's data,

while the paragraph is intended to give additional context and persuade the user to sign up using the button placed below. A simple logo was designed using the Raleway font – the same font which has been used across the front page – and placed in the top right corner. The front page is shown below.

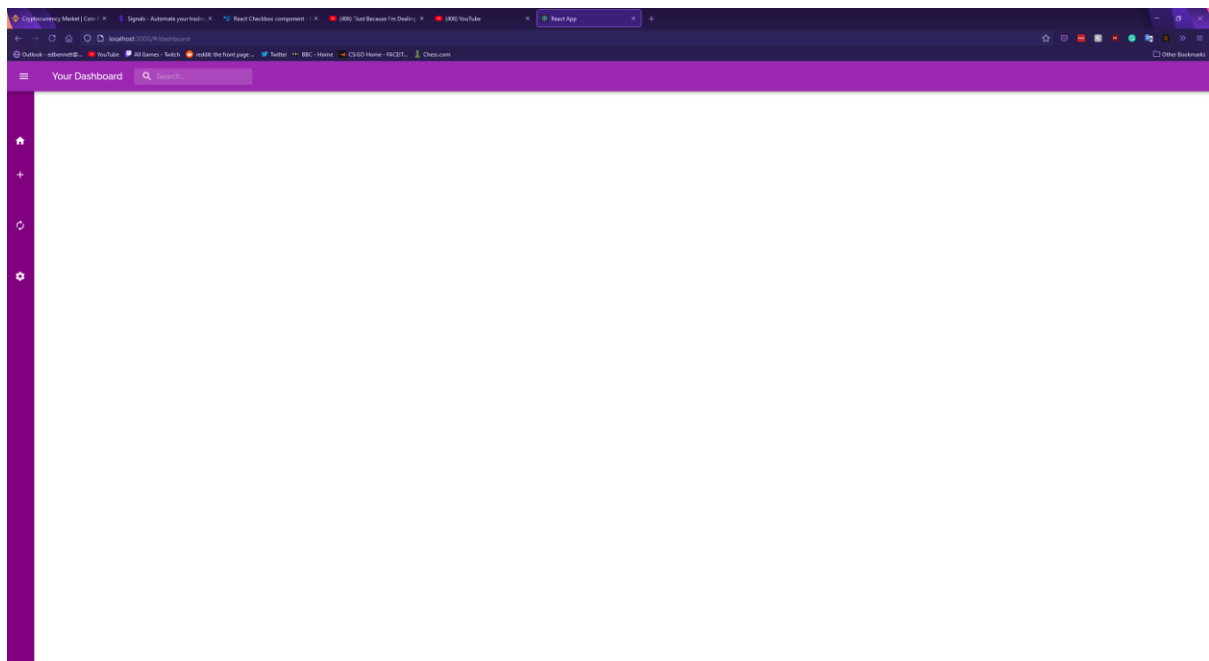
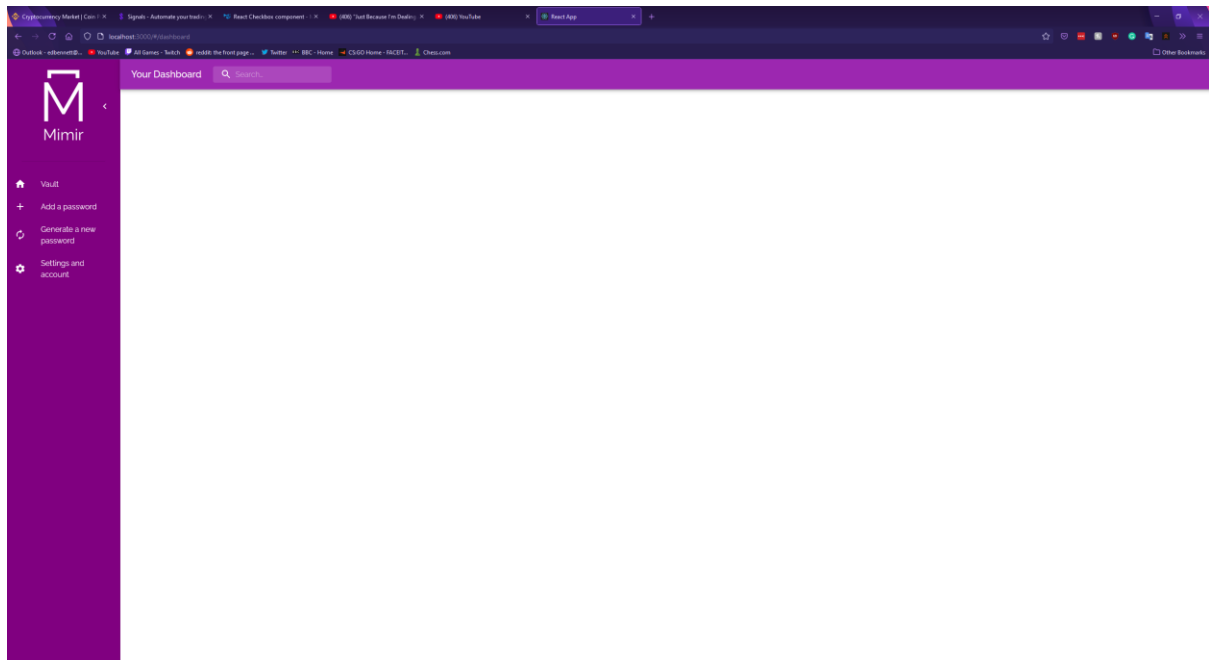


With the front page created, the next step was to update the sign-up and login forms to homogenise the two pages. The same gradient background was placed behind the form's paper background, and the project logo was placed in the same location in the top left corner, where it acts as a link back to the front page. This next iteration of the sign-up and login forms can be seen below.



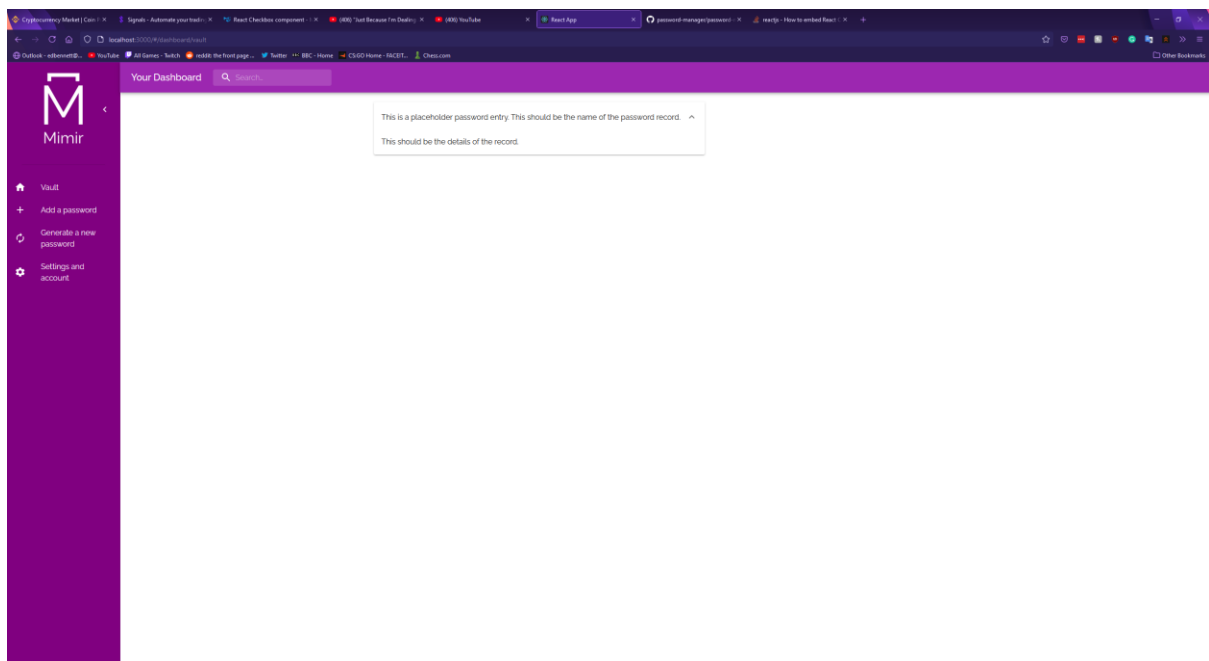
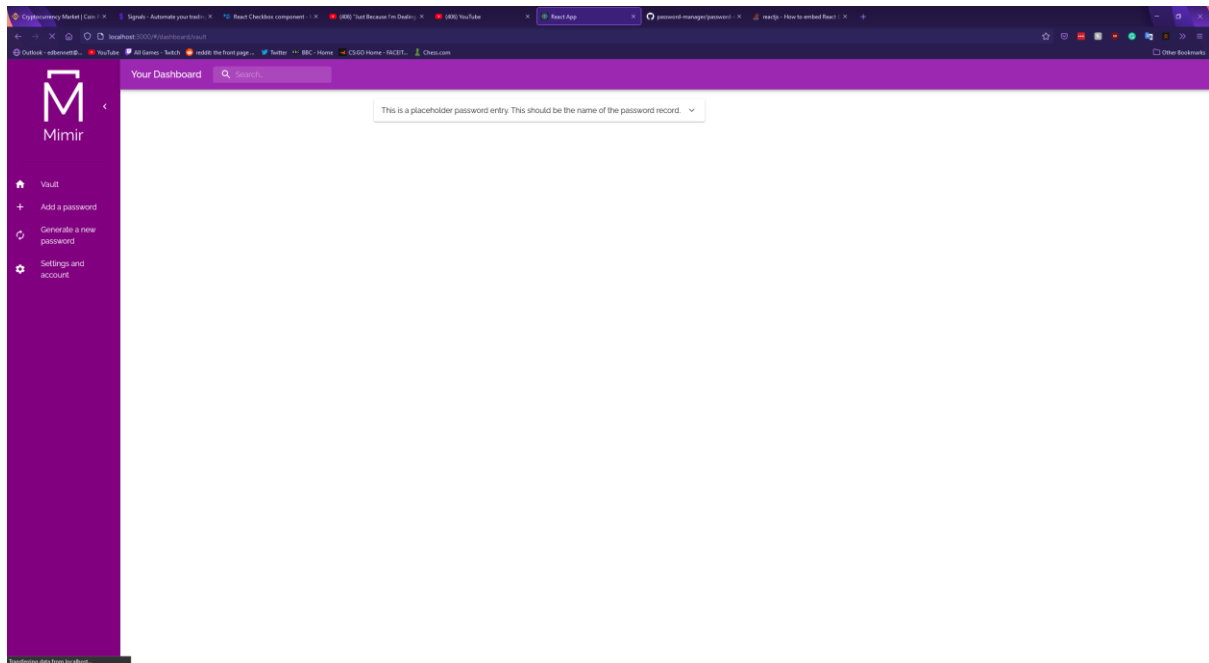
The final page of the web app was the dashboard. This would be the main user experience and would be the location the user interacts with most. Following the Material-UI documentation, the basic interface for a dashboard was designed with a large blank white space for various components to be placed. A purple AppBar was placed at the top of the window, with decorative typography and a search bar intended for searching the user's vault. A sidebar was placed on the left-hand side, containing the project logo, a button to minimise the sidebar, and a list of links to different components of the dashboard. This dashboard layout was inspired by the existing products researched, primarily Keeper,

Dashlane, and LastPass. The basic interface of the dashboard is shown below, with the sidebar at full size and minimised.

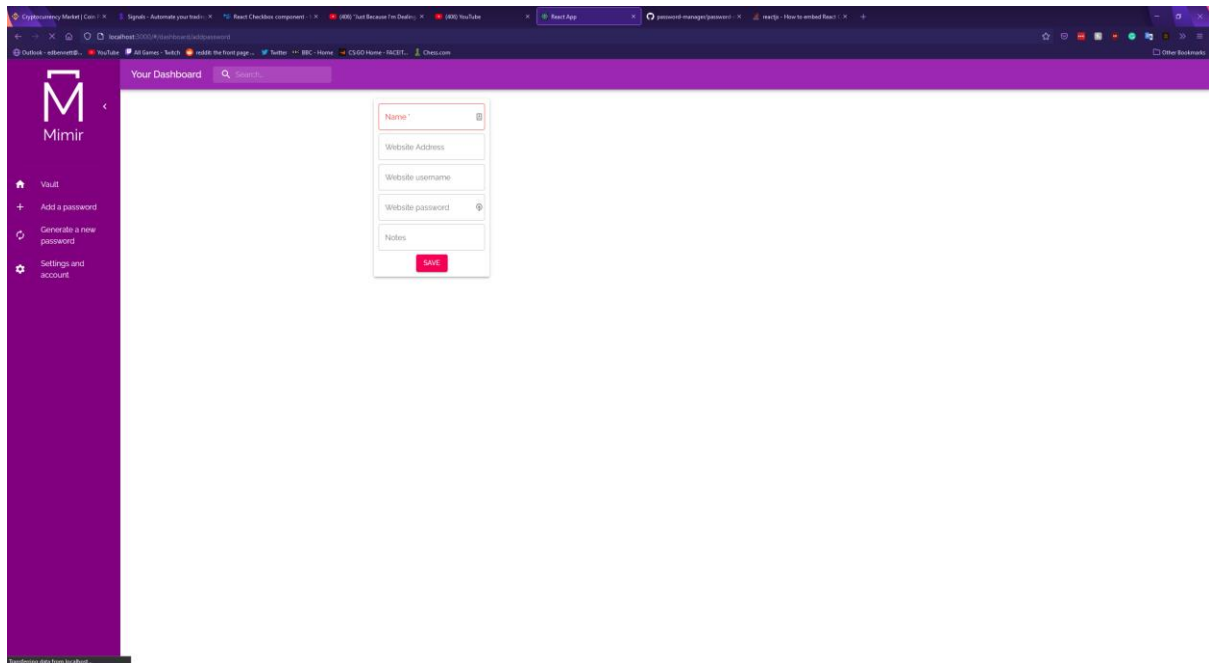


With the layout of the dashboard designed, the next step was to design the dashboard components. Following the structure of the researched existing products, I identified four components: the user's vault showing their password records, a component to create and save a new record, a component to generate passwords, and an account settings page for the user to update their account details.

The first component was the vault, and it was designed using Material-UI's Accordion component. Records would be displayed as entries in the accordion, with the record's name being used as the entry's summary. If the user were to click on an entry it would expand showing the rest of the record's fields such as the URL and password. An initial implementation of such an accordion is shown below, though this early implementation is not functional and uses hardcoded placeholder text.

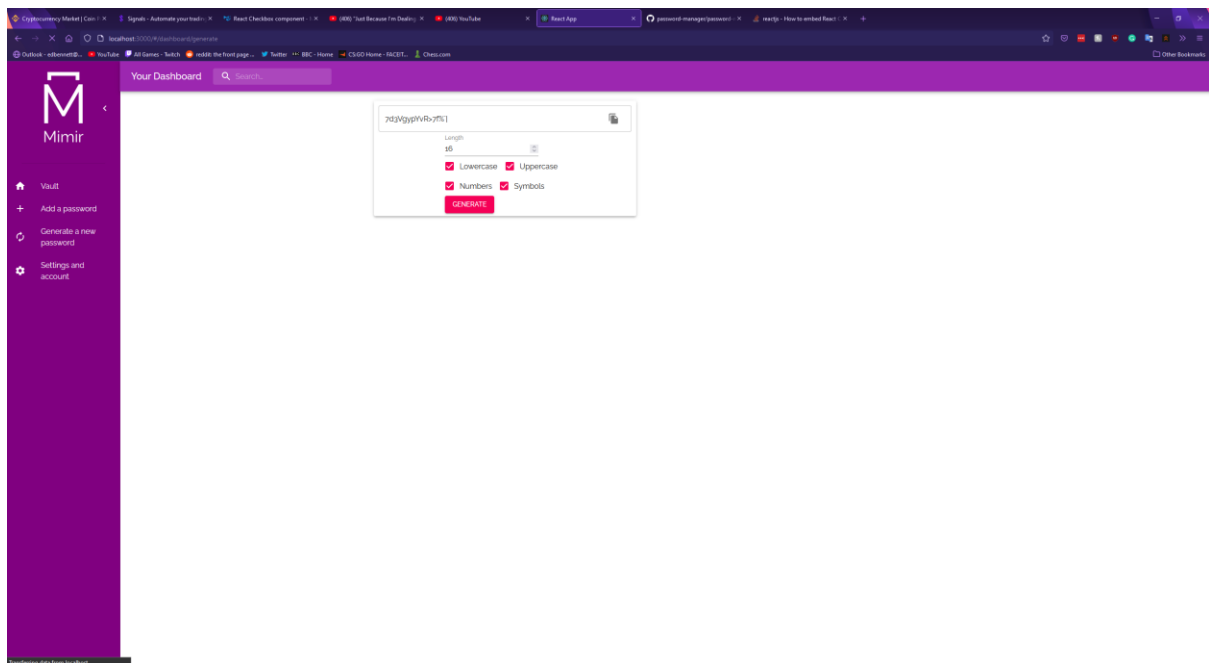
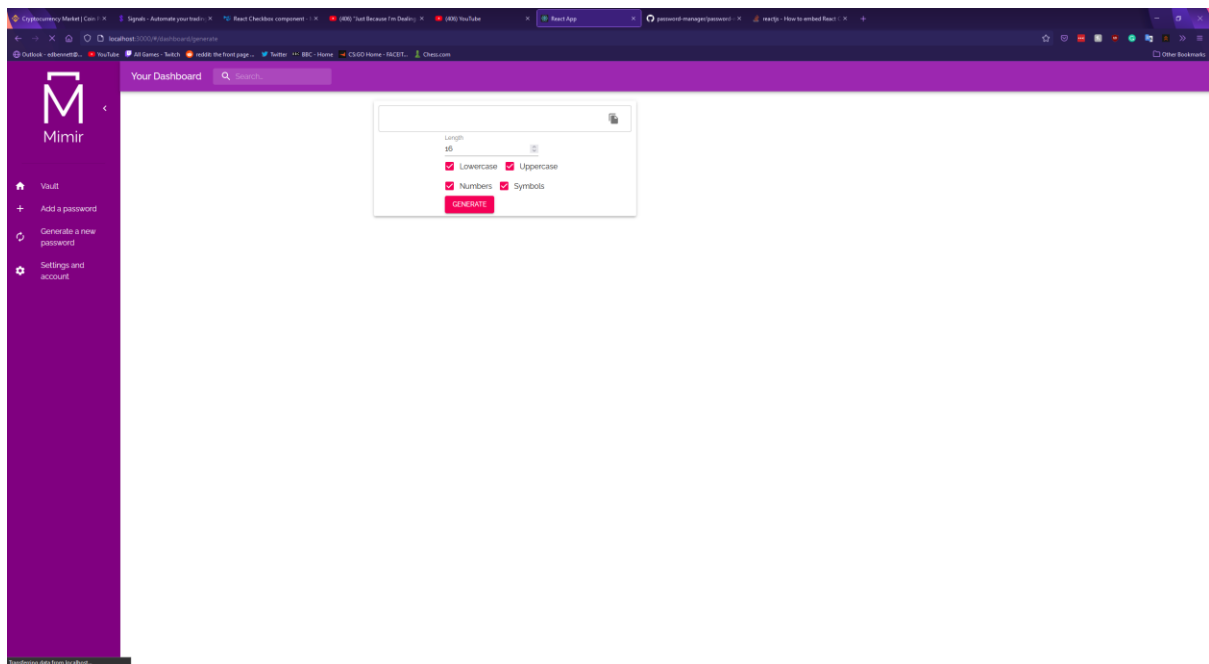


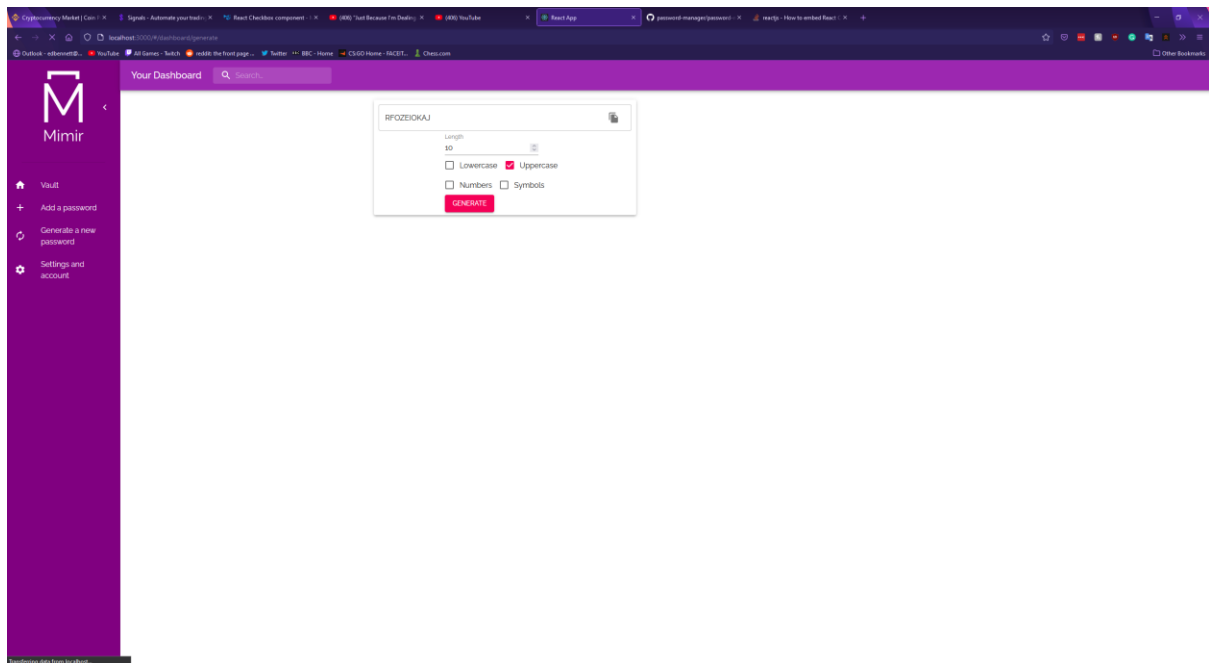
The second dashboard component, the record creation component, was designed as a new form. In keeping with the design of the sign-up and login forms, the new record form was designed as a column of TextField components followed by a highlighted save button all placed on a Material-UI Paper surface designed to give the appearance of elevation from the background and draw the user's focus. Formik was again used to validate the user's input, ensuring that the user entered a name for the record.



The next dashboard component was the password generator. The password generator was created as two TextFields, four checkboxes, and a button. The first TextField stays empty until a password has been generated, after which the generated string is placed in the field. The second TextField was set as a numerical input field with a default value of 16. This is used to obtain the length of the password to be generated. The four checkboxes are used to allow the user to select whether uppercase letters, lowercase letters, numbers, and symbols are included in the generated password. Once the user clicks the generate button a Javascript snippet runs which constructs a password using the length and character settings. An icon was placed in the right edge of the password TextField, which copies the generated password to the user's clipboard when clicked. The aim behind this was to make using this generated password slightly more convenient, rather than forcing the user to manually click and highlight the contents of the field. This feature was inspired by the existing password managers examined in the research phase of this project. Example screenshots of the

component are shown below, along with the code snippet which handles password generation.





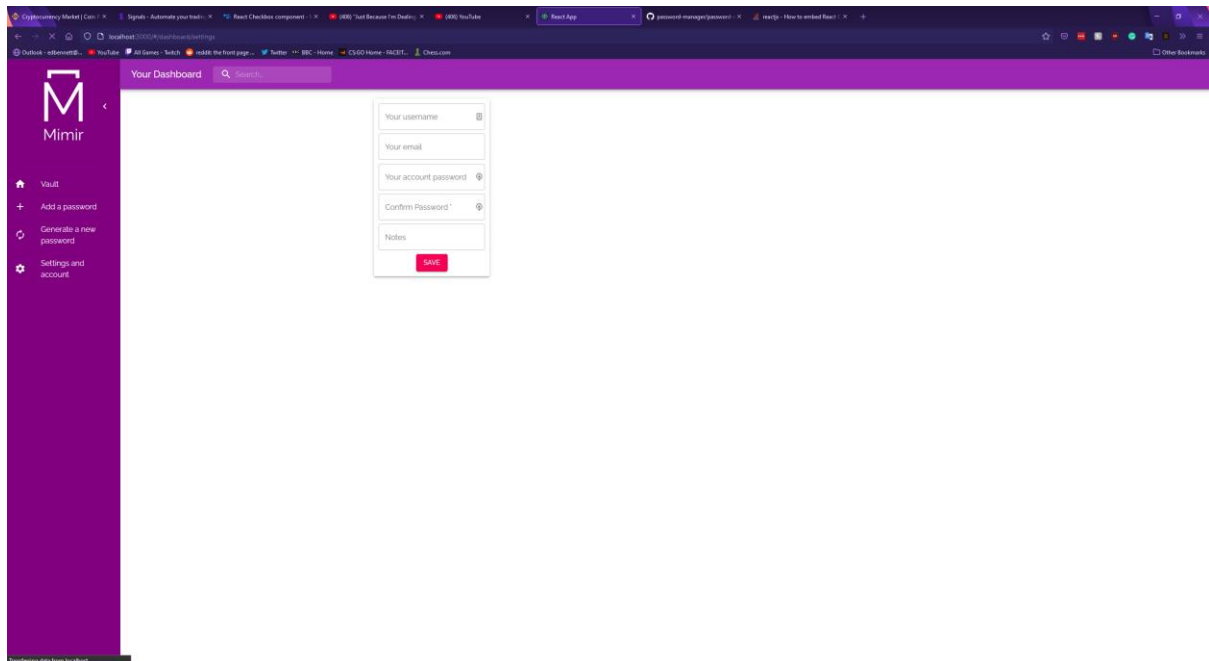
```

29   const [length, setLength] = useState("16");
30   const lower = "abcdefghijklmnopqrstuvwxyz";
31   const upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
32   const num = "1234567890";
33   const sym = "!@#$%^&*()_+~`|}{[]:;?><.,/-=";
34
35   function generate() {
36     var charset = "";
37     var result = "";
38     if (lowercase) {
39       charset += lower;
40     }
41     if (uppercase) {
42       charset += upper;
43     }
44     if (numbers) {
45       charset += num;
46     }
47     if (symbols) {
48       charset += sym;
49     }
50     for (var i = 0, n = charset.length; i < length; i++) {
51       result += charset.charAt(Math.floor(Math.random() * n));
52     }
53     setPassword(result);
54   }

```

The final dashboard component was the account settings form, which should be used to update the user's own account details. This component was constructed using a repurposed

version of the sign-up form, allowing the user to change any of the account settings they submitted when they created their account. Input validation was again performed using Formik and Yup.



Once the design was finished the frontend was connected to the backend using a library called Axios – “a simple promise based HTTP client for the browser and node.js” (Zabriskie, n.d.). Axios was used to make HTTP requests to the endpoints on the backend so data can be retrieved from the frontend and stored in the MongoDB database, and so data can be retrieved from the database to be displayed in the web client. The first endpoint connected was for the login form, using a POST request to send data to the endpoint created at <http://localhost:3100/api/auth/login>. The backend then responds with the user’s access token, which is saved into local storage before the user is redirected to their dashboard.

```
14  async function logIn() {
15      const { username, password } = values;
16      const { data, status } = await axios.post("http://localhost:3100/api/auth/login", {
17          username,
18          password,
19      });
20      if (status === 201 && data?.access_token) {
21          localStorage.setItem("access_token", data.access_token);
22          window.location.assign("/#/dashboard/");
23      }
24  }
```

This method was then repeated with the sign-up, create a record, and settings forms. The second step to connecting the client to the backend involved using HTTP GET requests to retrieve data from the database so that it could be displayed to the user.

First push signup and login and locally store token then pull records from database to show in vault

Search bar to find records

Only allow user into dashboard with a token, otherwise redirect to login

Colour palette and spacing changes

Any other tweaks

Results

The results chapter tells a reader about your findings based on the methodology you have used to solve the investigated problem. For example:

- If your project aims to develop a software/web application, the results may be the developed software/system/performance of the system, etc., obtained using a relevant methodological approach in software engineering.
- If your project aims to implement an algorithm for its analysis, the results may be the performance of the algorithm obtained using a relevant experiment design.
- If your project aims to solve some problems/research questions over a collected dataset, the results may be the findings obtained using the applied tools/algorithms/etc. Arrange your results and findings in a logical sequence.

Testing and Validation

Test description	Expected outcome	Actual outcome	Success?
Creating a new account	A new account is added to the database		
Attempting to create a new account with password fields that do not match	The user should not be able to create a new account and the request should not go through		
Attempting to create a new account with a password shorter than 8 characters	The user should not be able to create a new account and the request should not go through		
Logging in to an existing account	An existing account is successfully logged into, and the dashboard is opened		
Attempting to login with incorrect details	The user is not allowed to login		
Passwords are retrieved and displayed on the user's vault page	All password records for the logged in user are displayed on their vault page		

Adding a new record	A new record is created using the component to add a record, and the record is displayed on the user's vault page		
Changing user settings	The user's settings are successfully updated		

User Feedback

Questionnaire – Likert scale, create questions and results to evaluate project

Discussion and Analysis

Talk about results

Depending on the type of project you are doing, this chapter can be merged with “Results” Chapter as “Results and Discussion” as suggested by your supervisor. In the case of software development and the standalone applications, describe the significance of the obtained results/performance of the system.

A section - Discussion and analysis chapter evaluates and analyses the results. It interprets the obtained results.

Significance of the findings - In this chapter, you should also try to discuss the significance of the results and key findings, in order to enhance the reader’s understanding of the investigated problem

Limitations - Discuss the key limitations and potential implications or improvements of the findings.

Summary - Write a summary of this chapter.

Conclusion

Was the project successful? Why? Why not?

Typically a conclusions chapter first summarizes the investigated problem and its aims and objectives. It summaries the critical/significant/major findings/results about the aims and objectives that have been obtained by applying the key methods/implementations/experiment set-ups. A conclusions chapter draws a picture/outline of your project's central and the most signification contributions and achievements. A good conclusions summary could be approximately 300–500 words long, but this is just a recommendation. A conclusions chapter followed by an abstract is the last things you write in your project report.

Lessons Learned/Future work

What would I do differently were I to do it again

This section should refer to Chapter 4 where the author has reflected their criticality about their own solution. The future work is then sensibly proposed in this section.

Guidance on writing future work: While working on a project, you gain experience and learn the potential of your project and its future works. Discuss the future work of the project in technical terms. This has to be based on what has not been yet achieved in comparison to what you had initially planned and what you have learned from the project. Describe to a reader what future work(s) can be started from the things you have completed. This includes identifying what has not been achieved and what could be achieved. A good future work summary could be approximately 300–500 words long, but this is just a recommendation.

Acknowledgements

Is it weird to have acknowledgements for a dissertation? Fuck if I know. Thank my cat I guess

Bibliography

- Florencio, D., & Herley, C. (2007). A large-scale study of web password habits. In Proceedings of the 16th international conference on World Wide Web (pp. 657-666).
- Stanton, J. M., Stam, K. R., Mastrangelo, P., & Jolton, J. (2005). Analysis of end user security behaviors. *Computers & security*, 24(2), 124-133.
- Das, A., Bonneau, J., Caesar, M., Borisov, N., & Wang, X. (2014,). The tangled web of password reuse. In NDSS (Vol. 14, No. 2014, pp. 23-26).
- Huth, A., Orlando, M. and Pesante, L., 2012. Password security, protection, and management. *United States Computer Emergency Readiness Team*.
- Blocki, J. and Datta, A., 2016, June. CASH: A cost asymmetric secure hash algorithm for optimal password protection. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)* (pp. 371-386). IEEE.
- Provos, N. and Mazieres, D., 1999, June. A Future-Adaptable Password Scheme. In *USENIX Annual Technical Conference, FREENIX Track* (pp. 81-91).
- Atasu, K., Breveglieri, L. and Macchetti, M. (2004). Efficient AES implementations for ARM based platforms. *Proceedings of the 2004 ACM symposium on Applied computing - SAC '04*. [online] Available at: <https://dl.acm.org/doi/pdf/10.1145/967900.968073>.
- Quesenbery, W. (2003). *Dimensions of Usability: Defining the Conversation, Driving the Process*. [online] <https://www.wqusability.com/>. Whitney Interactive Design. Available at: <https://www.wqusability.com/articles/5es-upa2003.pdf>.
- Fielding, R.T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. [online] Uci.edu. Available at: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.
- IEEE, 2014. Improving usability of passphrase authentication. [online] Available at: <<https://ieeexplore.ieee.org/abstract/document/6890939>>.
- Aggarwal, S., 2018. Comparative analysis of MEAN stack and MERN stack. *International Journal of Recent Research Aspects*, 5(1), pp.127-132.

IBM Cloud Education, 2021. *mean-stack-explained*. [online] Ibm.com. Available at: <<https://www.ibm.com/cloud/learn/mean-stack-explained>>.

Ahmad, K.S., Ahmad, N., Tahir, H. and Khan, S., 2017, July. Fuzzy_MoSCoW: A fuzzy based MoSCoW method for the prioritization of software requirements. In *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICT)* (pp. 433-437). IEEE.

BAW Agency. 2021. *The Hero Section Checklist: 5 Essential Elements of a Hero Section that will Wow Visitors*. [online] Available at: <<https://baw.agency/hero-section/>>.

Elizabeth, L., 2016. *A Simple Web Developer's Color Guide — Smashing Magazine*. [online] Smashing Magazine. Available at: <<https://www.smashingmagazine.com/2016/04/web-developer-guide-color/>>.

Fireship (2021). *RESTful APIs in 100 Seconds // Build an API from Scratch with Node.js Express*. [online] www.youtube.com. Available at: <https://www.youtube.com/watch?v=-MTSQjw5DrM>.

Mysliwiec, K., n.d. *Documentation | NestJS - A progressive Node.js framework*. [online] Documentation | NestJS - A progressive Node.js framework. Available at: <<https://docs.nestjs.com>>.

Mongodb.com. (n.d.). *The MongoDB 4.2 Manual — MongoDB Manual*. [online] Available at: <https://docs.mongodb.com/manual/>.

Mongoosejs.com. n.d. *Mongoose v5.12.15*. [online] Available at: <<https://mongoosejs.com/docs/>>.

Hanson, J. (n.d.). *Passport.js*. [online] Passport.js. Available at: <https://www.passportjs.org/>.

auth0.com (n.d.). *JWT.IO*. [online] Jwt.io. Available at: <https://jwt.io/>.

Spec.openapis.org. 2020. *OpenAPI Specification v3.0.3 | Introduction, Definitions, & More*. [online] Available at: <<https://spec.openapis.org/oas/v3.0.3>>.

Reactjs.org. n.d. *Getting Started – React*. [online] Available at: <<https://reactjs.org/docs>>.

Material Design. n.d. *Material Design*. [online] Available at: <<https://material.io/>>.

Material-ui.com. n.d. *Material-UI: A popular React UI framework*. [online] Available at: <<https://material-ui.com/>>.

Palmer, J., n.d. *Formik*. [online] Formik.org. Available at: <<https://formik.org/>>.

Quense, J., n.d. *Yup*. [online] GitHub. Available at: <<https://github.com/jquense/yup>>.

Zabriskie, M., n.d. *Axios*. [online] Axios-http.com. Available at: <<https://axios-http.com/>>.