Criterion C: Development

Table of Contents

[Programming languages, frameworks and libraries used 1](#_Toc3423759)

[Documents 2](#_Toc3423760)

[Starting Screen 2](#_Toc3423761)

[Register and sign in screens 4](#_Toc3423762)

[Firebase and user authentication 9](#_Toc3423763)

[Making changes to the password list 12](#_Toc3423764)

[Adding an entry 12](#_Toc3423765)

[Removing an entry 13](#_Toc3423766)

[Editing an entry 14](#_Toc3423767)

[Encryption 15](#_Toc3423768)

[Hashing 16](#_Toc3423769)

[Decrypting 16](#_Toc3423770)

[Encrypting 17](#_Toc3423771)

[References 17](#_Toc3423772)

# Programming languages, frameworks and libraries used

* JavaScript
* HTML
* CSS
* Bootstrap (Bootstrap, 2019)
* JQuery (The JQuery Foundation, 2019)
* Firebase (Google Developers, 2019)
* AES encryption-JS (Ricmoo, 2018)
* Js-scrypt (Garnock-Jones, 2016)
* FontAwesome (Fonticons, Inc., 2019)
* clipboard.js (Rocha, 2019)

# Documents

The Solution is split between three main documents. The HTML file contains the layout and structure of the Solution. The CSS file contains the style of the solution, and essentially makes the Solution look good. The JavaScript file handles all the responsive elements of the Solution, and makes the Solution work.

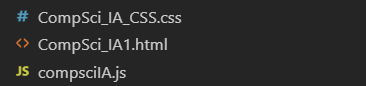


Figure 1: Documents

In addition to these documents, several other documents from outside libraries were used. These are: Bootstrap, JQuery, FontAwesome, clipboard.js, AES encryption-JS, and Js-scrypt. Furthermore, Firebase was also utilized. These are all linked at the top of the HTML file.



Figure 2: Links at beginning of HTML

# Starting Screen

The first screen that the client sees when opening the application was designed using HTML and CSS. The figures below show this screen:

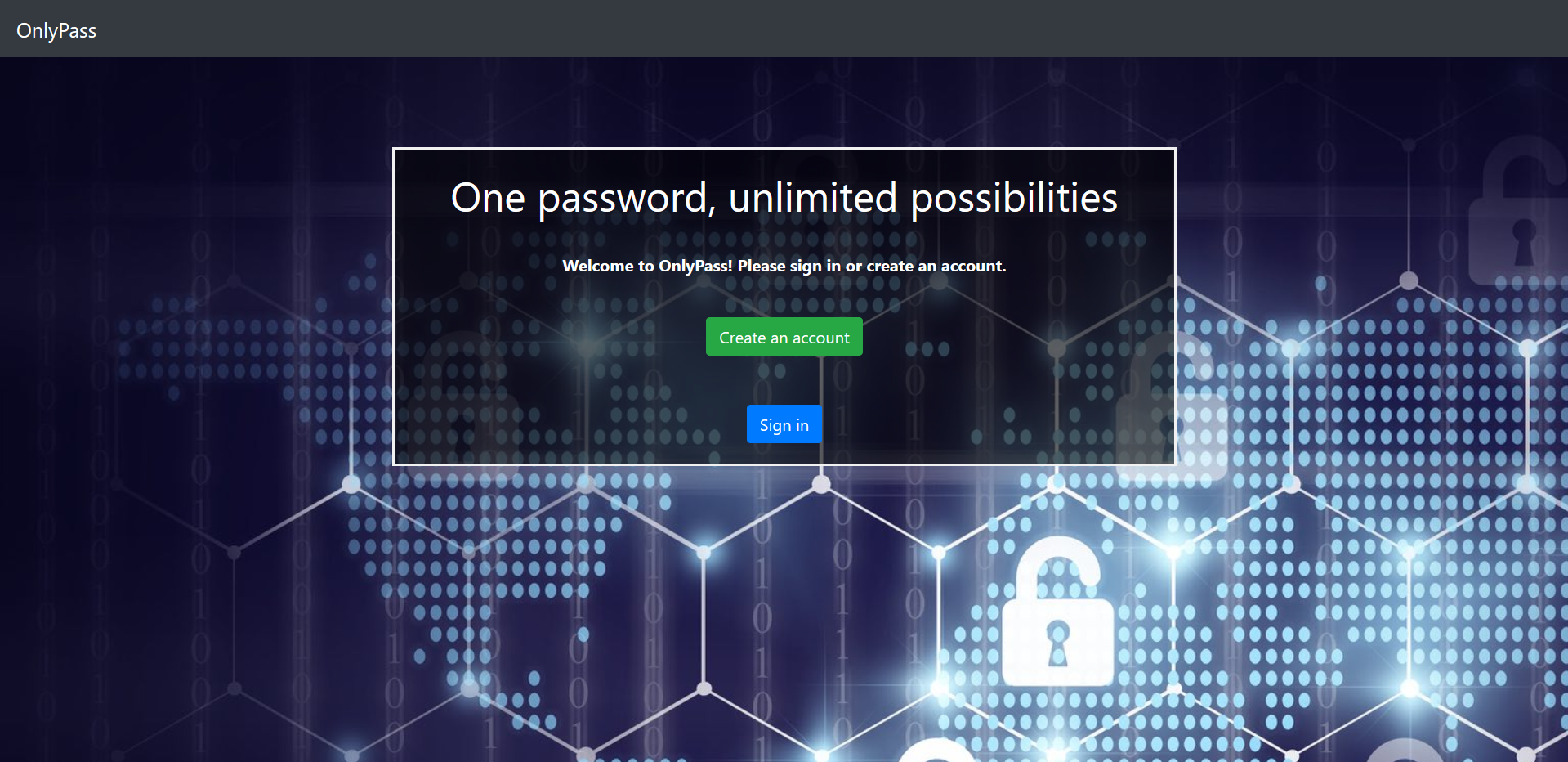


Figure 3: Starting Screen (Shutterstock, Inc., 2018)

The header at the top of this screen, where OnlyPass is displayed, was made using HTML and Bootstrap classes. The <nav> element was used, along with the classes navbar, navbar-expand-sm, bg-dark, and navbar-dark. Navbar-expand-sm is used to make the navbar mobile responsive. The other classes are used to make the navbar look the way it does.

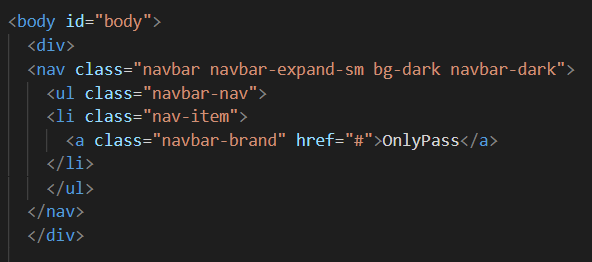


Figure 4: Navbar HTML

The rest of the HTML code that deals with the starting screen is shown in figure 5, and the corresponding CSS in figure 6:

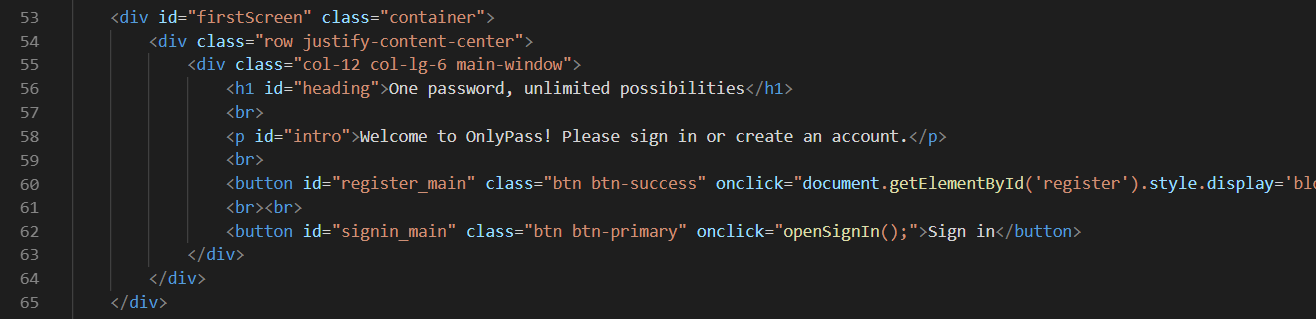


Figure 5: Starting screen HTML

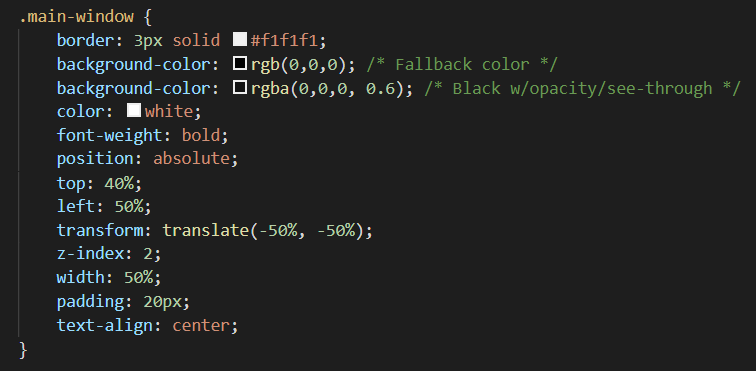


Figure 6: Starting screen CSS

The relevant HTML is given the class main-window. In the CSS, this class is defined to have a white border, a black background but without 100% opacity, and to be centered, among other things. The other classes are from Bootstrap, used to make the application mobile-responsive. Trivially, some text is displayed, and there are two buttons, which when clicked open the register or sign in interfaces.

# Register and sign in screens

The register and the sign in interfaces are modals, which are displayed on top of the starting page. They were created by consulting w3schools (Refsnes Data, 2019).

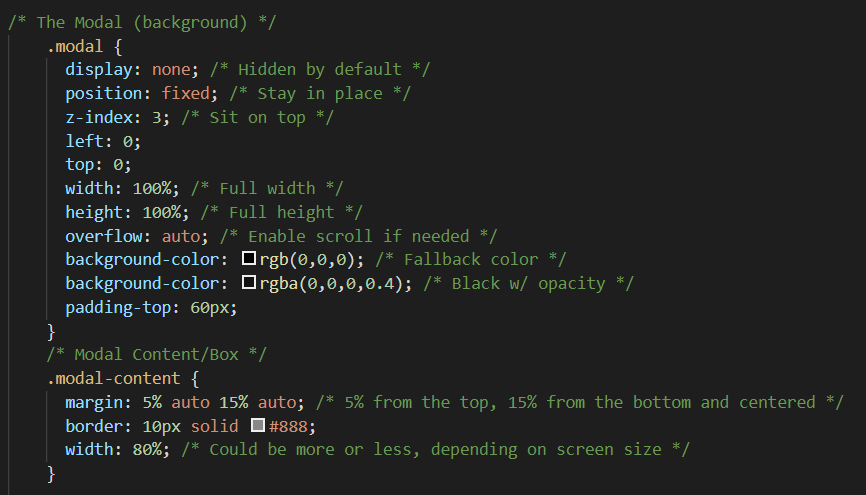


Figure 7: Modals CSS

The modals are implemented by having all HTML elements with the modal class hidden by default, and by having a higher z-index than all other elements, which results in modals being on top of everything else, once they are unhidden. The modals appear once the relevant aforementioned buttons are clicked; the second line in Figure 8 shows what code is used to make them appear. The second part of openSignIn() in Figure 8 is an event listener, and listens for the enter key being pressed (13 is the keycode for enter). If the enter key is pressed while the sign-in modal is open, the program attempts to sign the user in by running the signIn() function, which will be discussed in greater detail later. The variable signInOpen (overall line 166) is set used so that pressing the enter key will only trigger the signIn() function if the sign in screen is open. The variable is of type Boolean, and is true if the sign in screen is open, and false otherwise.



Figure 8: openSignIn function

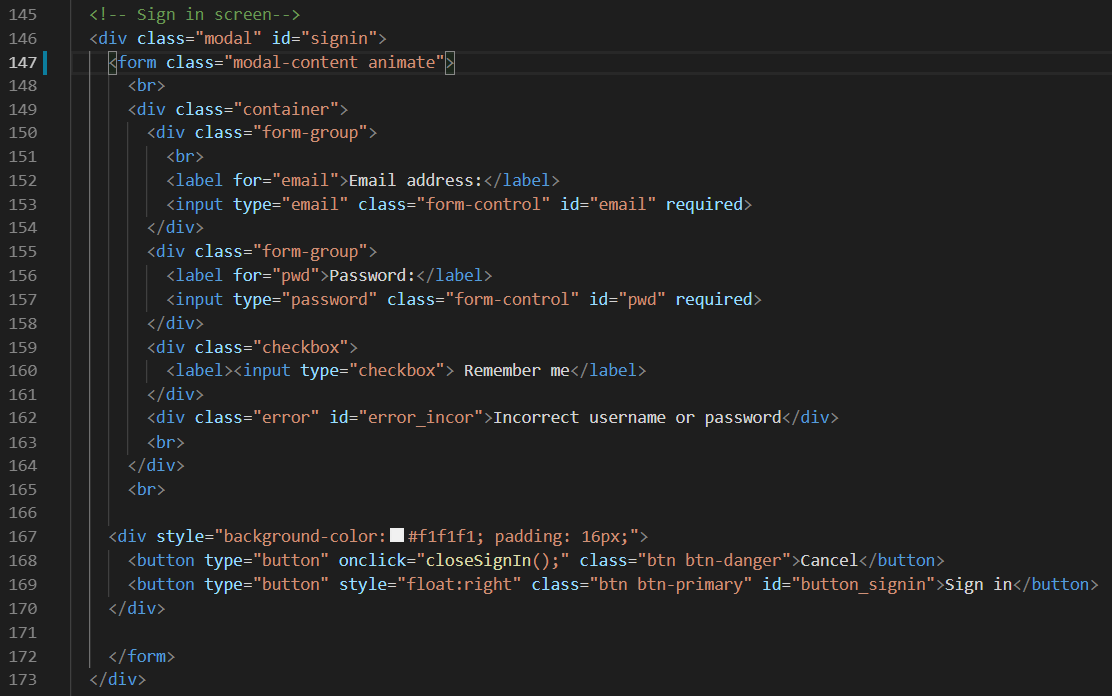


Figure 9: Sign in screen HTML

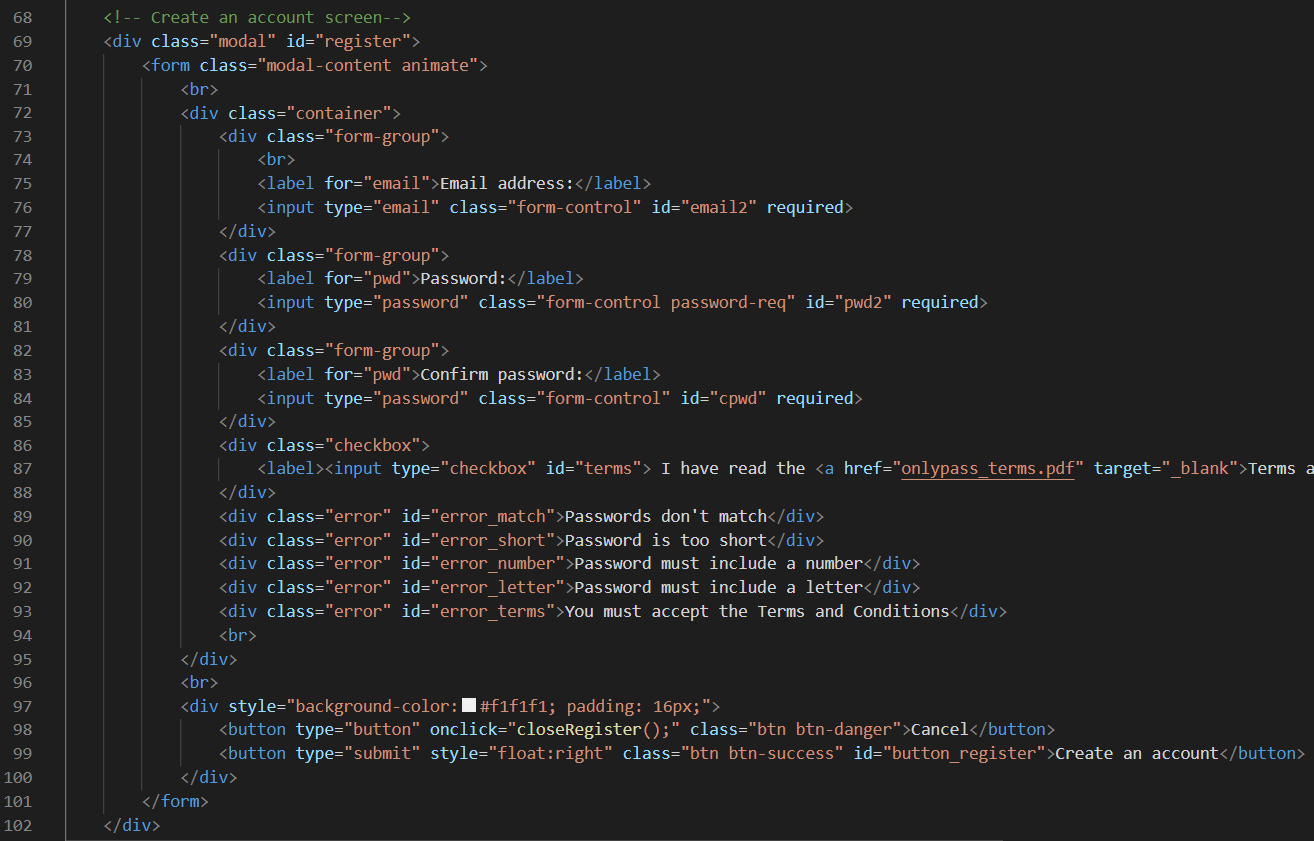


Figure 10: Register screen HTML

Figures 9 and 10 show the HTML of the sign in and register screens, respectively. Both contain a form, which the user fills out. Both also contain errors, which are hidden by default by giving them the class error, which is defined in CSS to cause elements to be hidden by default. Furthermore, the class also causes elements to appear in red, to draw the user’s attention to the error. Both screens contain two buttons, one to close the modal and one to sign in or register. The button clicks are handled directly (using onclick) for the cancel buttons, and Figure 11 shows the event listeners which handle the sign in and register buttons. Since the application is a single-page application, these are used to prevent the form from being submitted, thus preventing the page from being reloaded.

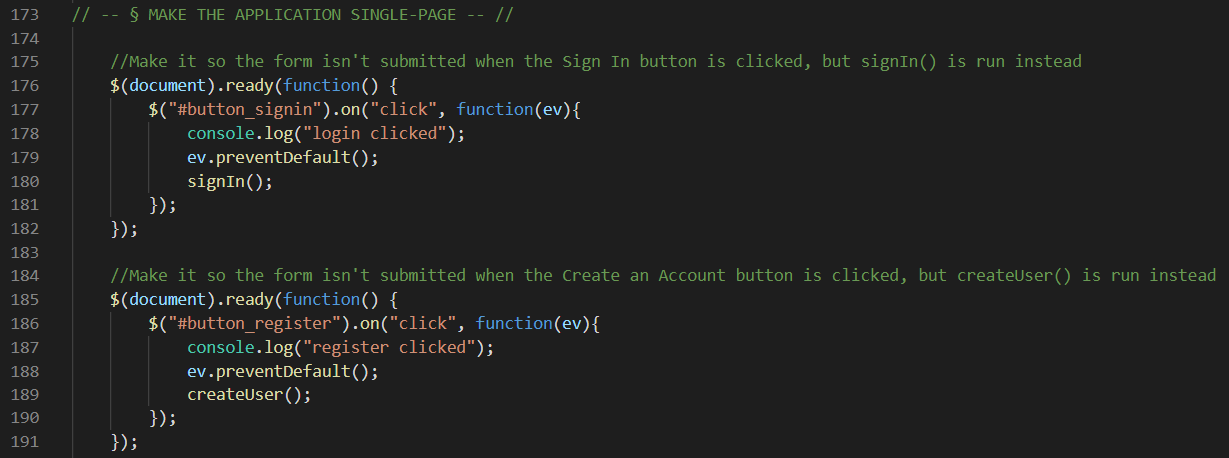


Figure 11: Event listeners for sign in and register buttons

Once the createUser() function is triggered, first all errors currently displayed will be hidden, so that there is a clean slate. Next, it will be checked whether the password and confirm password entries match. Moreover, the password will be checked against several criteria, namely whether it is sufficiently long and whether it contains both letters and numbers. Finally, it will be checked whether the user has accepted the Terms and Conditions. Should any of these checks fail, the relevant error message will be displayed and the function will stop execution. Otherwise, the function will attempt to create a user.

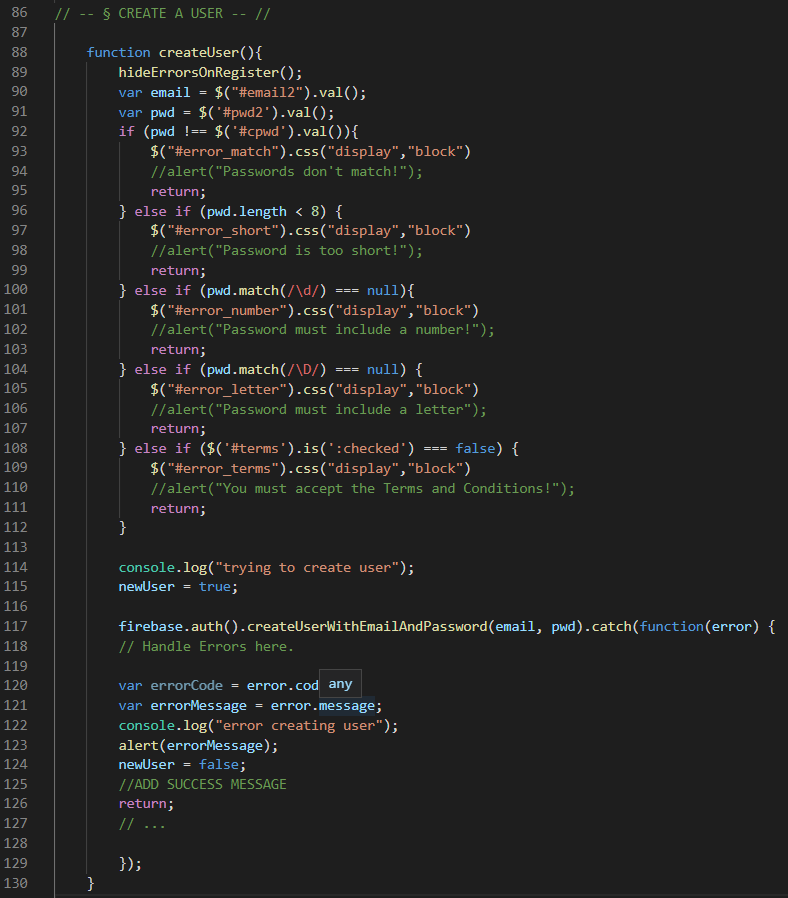


Figure 12: createUser function

User authentication is done via Firebase. This framework makes it possible to use predefined functions (e.g. line 80) to create user accounts and authenticate users. Firebase also serves the purpose of storing data.

# Firebase and user authentication

Firebase first has to be initialized, which is shown in the figure below. A link to the relevant Firebase project is established, and a connection with the online database is created. The fact that the database is online makes it easier to sync passwords across devices. The potential drawback of unsecure storage is removed by only sending already encrypted data to Firebase. The encryption key is never sent.

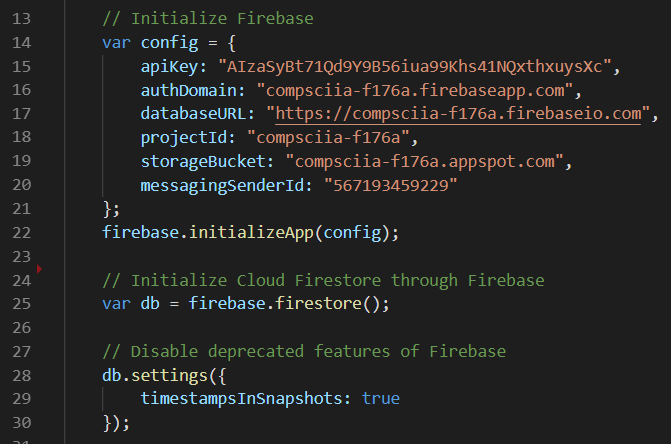


Figure 13: Initialization of Firebase

Once a new user account is created, the user has to sign in. This is done through the sign in screen as previously discussed, and the signIn() function is run.



Figure 14: signIn() function

A potential error message is first hidden, and then another predefined Firebase function is run. If there is no matching email and password pair, an error is displayed. Otherwise, the user is authenticated. There is an event listener that listens for a change of the authentication state, shown in Figures 15 and 16 below:



Figure 15: Authentication listener (part 1)

Once the authentication listener detects that a user has signed in, all elements of the starting screen are hidden, and the main interface of the program is displayed.



Figure 16: Authentication listener (part 2)

The program then obtains the encrypted password list from Firebase, and runs the function to decrypt and parse it (see decryption section). Next, the listPasswords() function is run, in which a loop displays all the different passwords for different websites in an orderly table (Figures 17, 18).



Figure 17: listPasswords() function

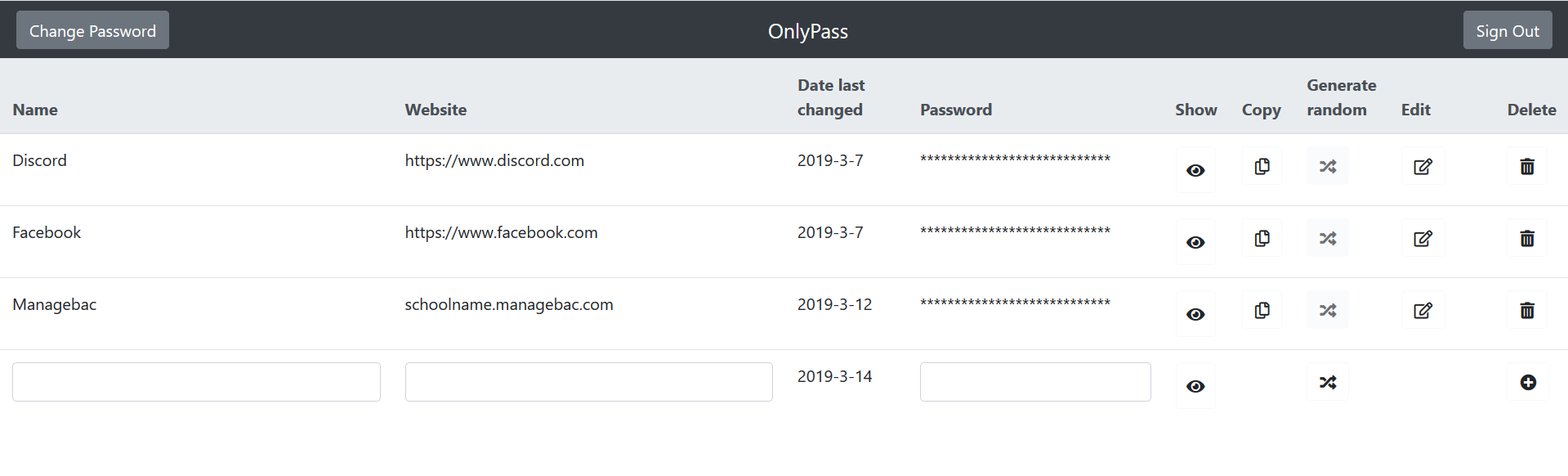


Figure 18: Password list screen

# Making changes to the password list

## Adding an entry

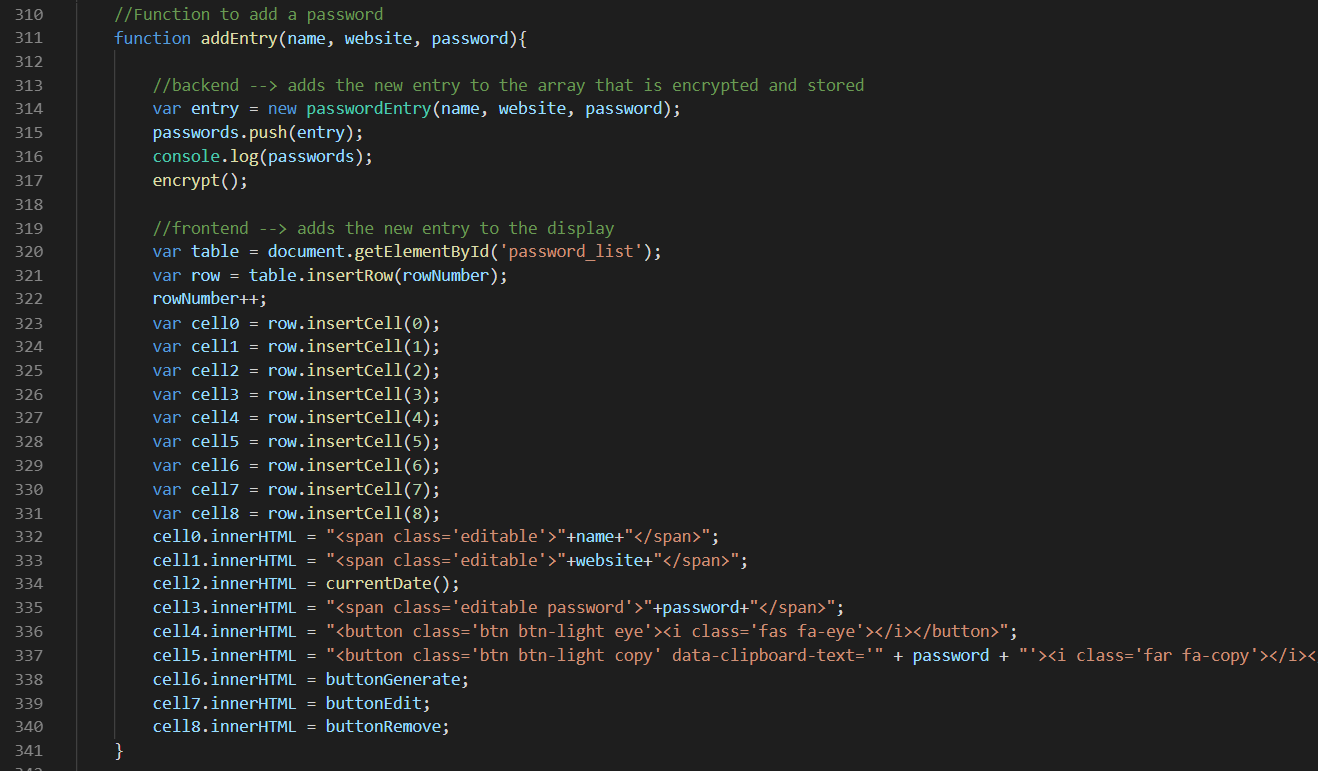


Figure 19: addEntry() function

To add a new password, the addEntry() function is run. This creates a new instance of the passwordEntry class and pushes it to the passwords array, then calls the encrypt() function to save changes. Next, the new password is displayed at the bottom of the table.

## Removing an entry

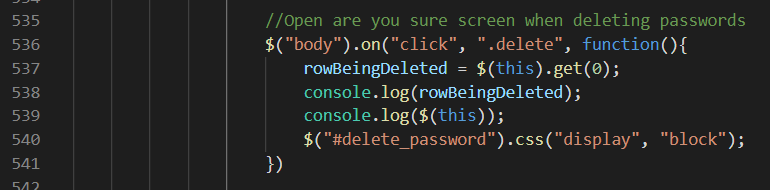


Figure 20: Deleting a password

Once the user clicks the delete icon, a modal opens, prompting a confirmation.

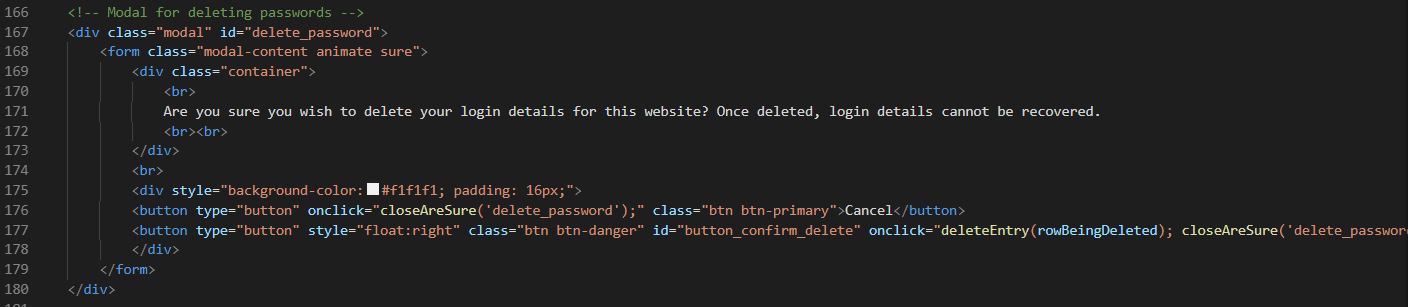


Figure 21: Modal for deleting passwords HTML

Should this be given, deleteEntry() is run.

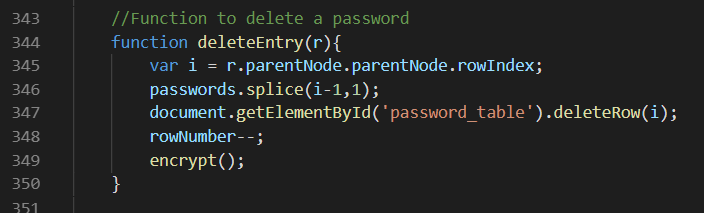


Figure 22: deleteEntry() function

## Editing an entry

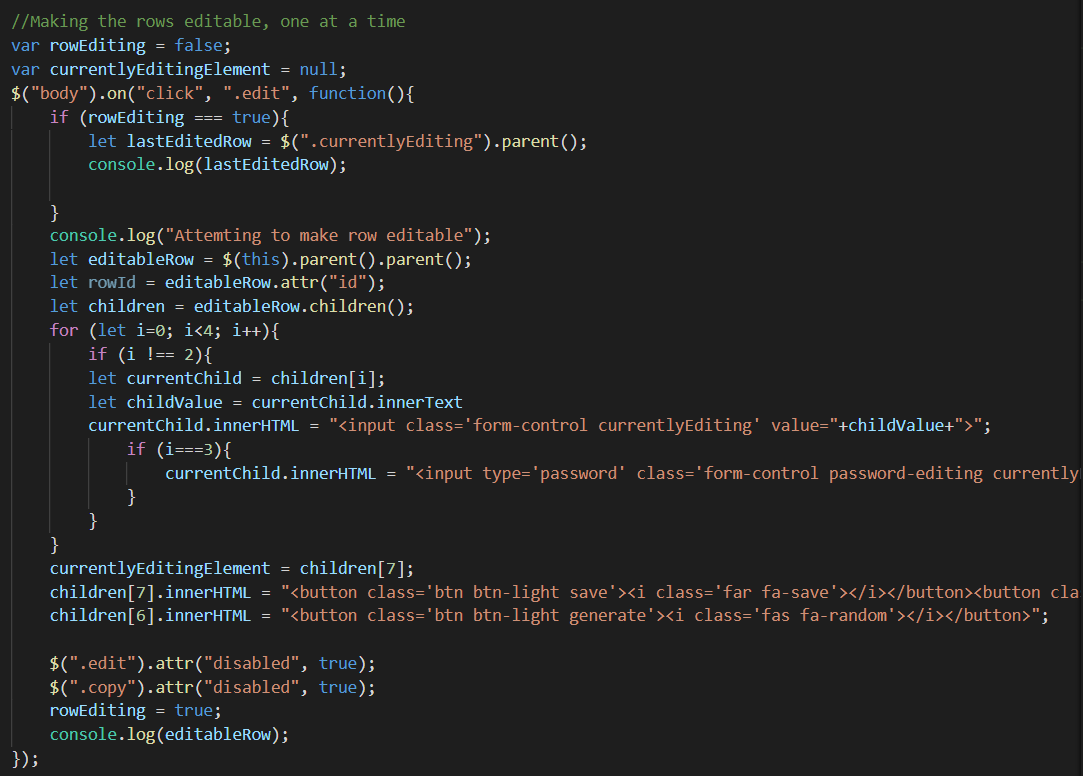


Figure 23: Editing an entry

Once the edit icon is clicked, the relevant row is obtained and elements are cycled through in a loop, where their innerHTML is changed to turn them into input fields. Save and undo buttons replace the edit button. Figures 24 and 25 show how changes are saved or undone.

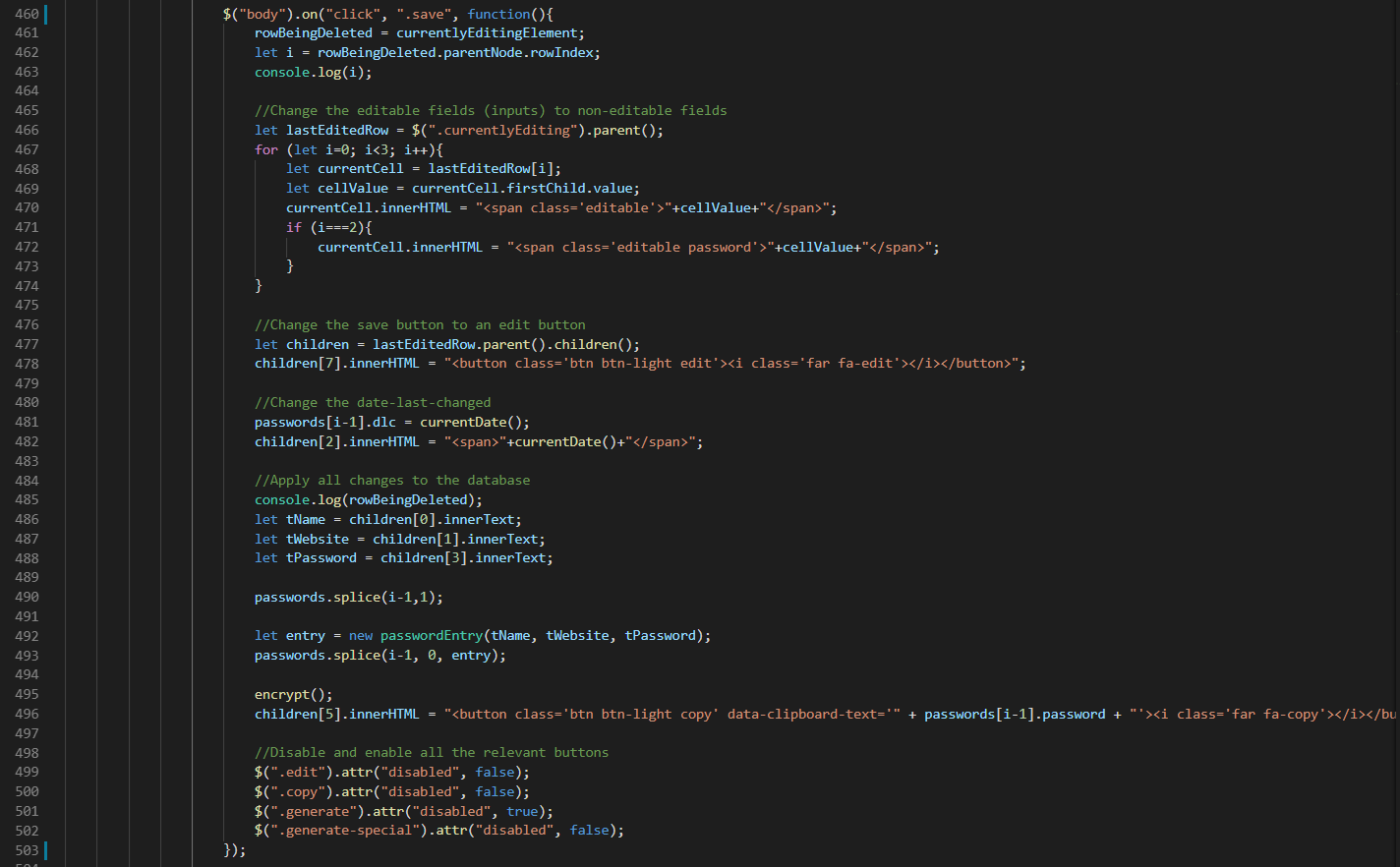


Figure 24: Save changes

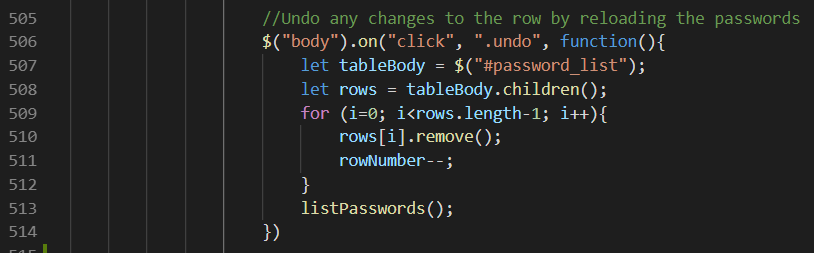


Figure 25: Undo changes

# Encryption

AES-256 encryption is used. This is one of the most secure forms of encryption available and is therefore used as the client specifically wished for the encryption to be robust and as secure as possible.

## Hashing

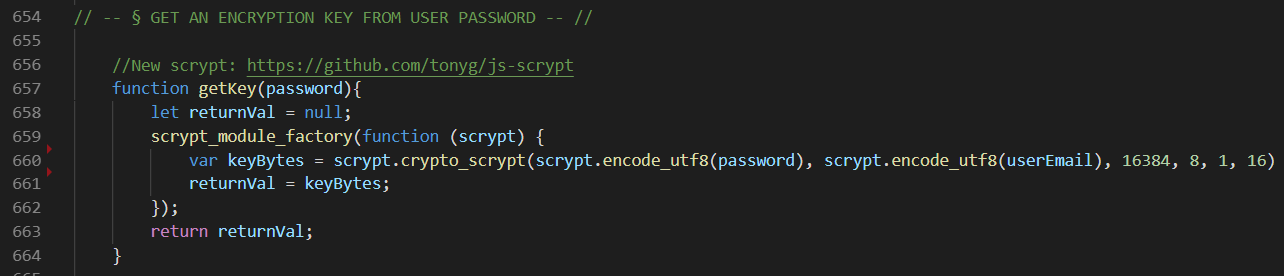


Figure 26: Hashing

The user’s password is hashed to obtain an encryption key using Js-scrypt (Garnock-Jones, 2016).

## Decrypting

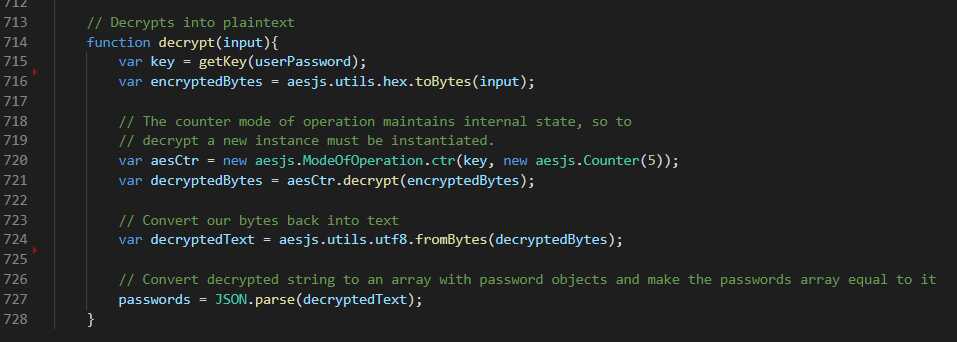


Figure 27: decrypt() function

The decryption function, run when the user logs into the application, first calls the hashing function to obtain a key, uses it to decrypt the passwords, before ultimately setting the passwords array equal to a parsed version.

## Encrypting

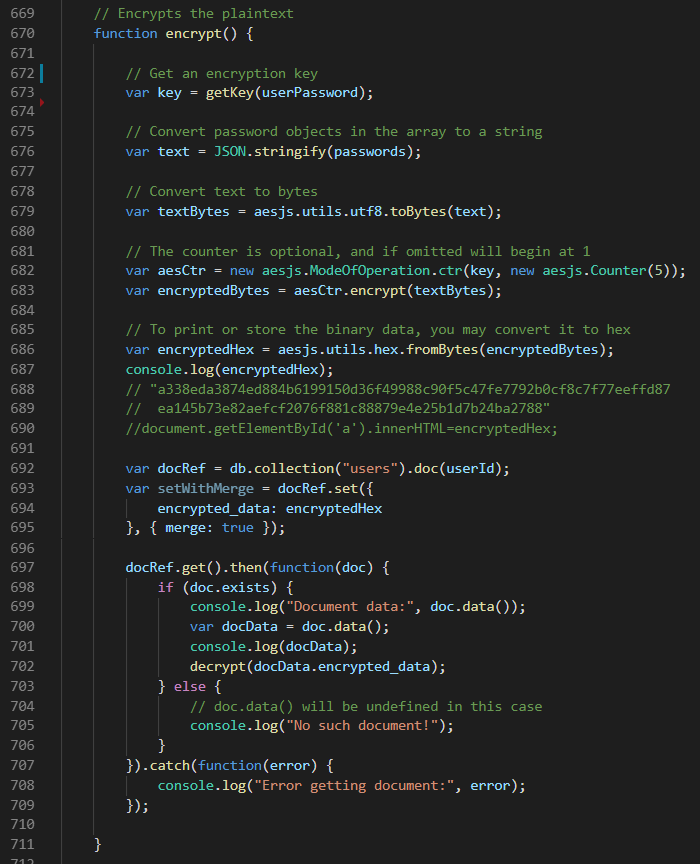


Figure 28: encrypt() function

Encrypt() likewise calls getKey(). It then converts the array of password objects to a string of bits, encrypts it, and saves it to Firebase.

*Word Count: 1223*

# References

Bootstrap. (2019). *Bootstrap*. Retrieved from https://getbootstrap.com

Fonticons, Inc. (2019). *Font Awesome*. Retrieved from https://fontawesome.com/

Garnock-Jones, T. (2016). *Js-scrypt: Pure-Javascript Emscripten-compiled scrypt routine*. Retrieved from https://github.io/tonyg/js-scrypt/

Google Developers. (2019). *Firebase*. Retrieved from https://firebase.google.com/

Refsnes Data. (2019). *How to - Login Form*. Retrieved from w3schools: https://www.w3schools.com/howto/howto\_css\_login\_form.asp

Ricmoo. (2018). *AES-JS*. Retrieved from https://github.com/ricmoo/aes-js

Rocha, Z. (2019). *clipboard.js*. Retrieved from https://clipboardjs.com/

Shutterstock, Inc. (2018). *Shutterstock*. Retrieved from https://www.shutterstock.com

The JQuery Foundation. (2019). *jQuery CDN – Latest Stable Versions*. Retrieved from https://code.jquery.com/