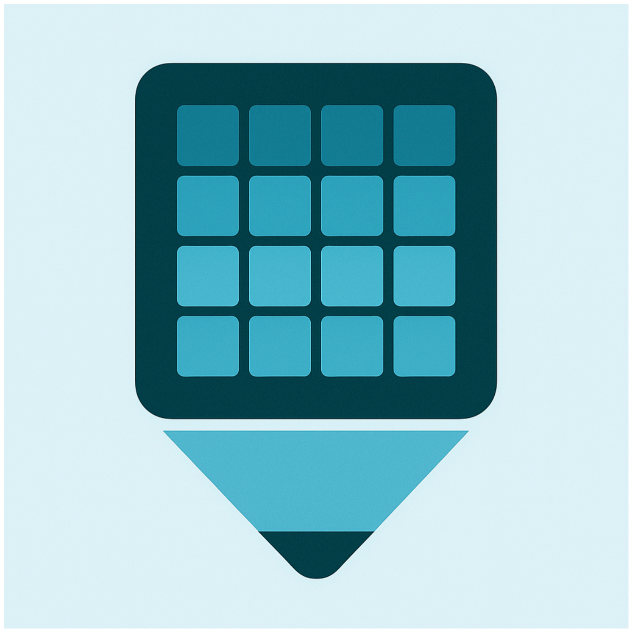
FINAL PROJECT IN COMPUTER SCIENCE

CALCULOCKED: FINAL REPORT

Shaquille Muhammad Uddin

https://github.com/CalcShaq/CalculockedPublic/



Abstract

As more people begin to face an overload in password while both users and companies begin to face data breaches, it becomes harder for the average person to keep their online accounts safe. This is especially prominent in individuals who tend to be older and face issues in regards to their memory. Password managers already existing tend to neglect user accessibility, they rely on complex methods or confusing cluttered interfaces that distances itself from demographics that are truly in need of password managers.

My project is Calculocked, a React Native mobile application. This is a vault application disguised as a basic calculator which is then able to securely save and present user login credentials. By utilising a hidden gimmick (triple tapping the equals "=" button) and a security Question and Answer prompt, Calculocked specialises in discrete yet accessible management of user data. The application uses Firebase for real-time cloud storage. Unit and user testing among both technical and non-technical users confirmed the app’s ease of use, proving my project to be effective and a desired product. My report will explain in detail the technicalities of the application alongside critically evaluating limitations of my release of this project as well as discussing future improvements such as biometric integration and additional accessibility features.

Keywords

Including why they relate to Calculocked

1. **Accessibility** - The focus on usability for my target audience
2. **Authentication** - Question and Answer safety mechanism
3. **Security** - Safety of a user’s data (their private accounts)
4. **Usability** - Design changes are highlighted based on user feedback

Introduction

In the last decade or so, society has transformed itself into a technological era where maintaining digital profiles and accounts (emails, work accounts, social media profiles, banking details) have become key to our daily lives. The transition to this digital era has led to a rapid increase of various username and password combinations, which may come with other security requirements such as Two Factor Authentication. While those who have grown up within this technological era have already grown up to understand these sophisticated technical needs, there is also a large population who are struggling to keep up and adapt to these changes. This population mentioned are usually older adults and individuals that may have cognitive challenges. Calculocked aims to cater to this audience by providing them an accessible and intuitive option for credential storage on their phones.

[1] The Office of National Statistics (2020) alarmingly states that over 325,000 mobile phone thefts occurred in England and Wales in that one year. Significant portion of that population not only lose a piece of hardware that is now essential in this day and age, but the risk of their sensitive information including accounts and passwords being exposed is increased. This means not only do we need our security tools to be stronger, but smarter tools too that can be used by all people regardless of their tech proficiency.

The Calculocked application is a response to cases and challenges such as these, designed to be the smarter tool: a password manager disguised as a calculator. This app protects user credentials using hidden gimmicks and security questions. The goal is to create a usable app that prioritises security, rather than overloading an application with many overwhelming features and incomprehensible settings. After signing in, the user is shown a calculator page which can be passed by pressing the equals sign three times before being presented the main password vault page after answering the security question correctly.

This project takes inspiration from a personal anecdote involving my mother, a woman who fits the target audience description who happened to lose access to important accounts after a phone theft. Her experience brings attention to the fact that people who are in need of data protection are typically less equipped and less understanding of how to protect themselves digitally as they do not have the right guidance or tools to do so. Calculocked attempts to empathise with users such as my mother, where design choices are made from simpler designs to help users not be overloaded with various options and accessibility is evaluated through user testing to create a simple and subtle system that remains effective throughout use.

This project report presents an overview of Calculocked's design, development, evaluation and future plans. Sections following this will explain the foundations and how it relates to cyber-security but also the interaction between people and the computers. I will explain with detail the technicalities and coding in this app which was made using React Native in TypeScript and Firebase Databases, and refinement using an iterative testing process known as the Action Research Cycle (ARC). Through this, it's possible to discuss its overall usability and limitations to which we can evaluate its potential in the future if this app was to continue being developed.

Background (Literature Review)

Password Management

Users are continuously expected to maintain unique and complex passwords for each of their accounts to reduce the risks of any data breaches including hacking and phishing. This expectation can be challenging for many users. [2] Lukas Grigas, a Cybersecurity Content

Writer, states “Our recent research revealed that today, the average user faces the challenge of maintaining about 100 passwords” and “The study also found that 30% of respondents found making such an amount of passwords is so stressful that it compares to the stress and anxiety of retiring.” These statements reiterate the fact that password management is crucial and does need a simpler yet safe option to put users at ease, to reduce their stress and anxiety.

Password management tools have emerged in cybersecurity as an asset to users to prevent stress and anxiety, including NordPass and 1Password, providing storage for user credentials which are often protected by master passwords. What these tools fail to understand is that not all users are unable to familiarise themselves with these apps and browser extensions as these are designed for users that are already experienced with digital interfaces and understand the huge importance of security. Their services does not take into account the older generation who aren’t as confident in technology as the current generation are and it does not cater towards those who have memory related issues. This is a barrier of entry for these groups of people and my application aims to target these users who are already struggling.

Existing Literature

In relation to digital security and accessibility

[3] Karen Renaud & Lizzie Coles-Kemp (2020) discuss in their article “Accessible and Inclusive Cyber Security: A Nuanced and Complex Challenge” how there is a critical need for inclusive cyber security as individuals facing disability issues and others from different groups face challenges from this. They argue that current security often exclude those populations due to barriers including: “cognitively unimpaired (e.g. can create and retain passwords), have the necessary resources (e.g. time, appropriate technology and internet access in a distraction-free environment), and have the required dexterity to interact with the security system (e.g. can use the mouse and keyboard with ease)”, which was mentioned when mentioning security systems such as CAPTCHA as they believe it only targets fully-abled users. There is emphasis in their work about the importance of accessibility integration in security designs and usability, suggesting future work and research should be significantly more inclusive and provide newer creative alternative authentication methods to cater towards a less-abled demographic.

[4] Mukta Kulkarni (2019) focuses on digital accessibility in his writing: “Digital accessibility: Challenges and opportunities”. He highlights challenges and opportunities in ensuring accessibility to less-abled people in technology. Kulkarni outlines barriers such as both institutional and technological limitations and mentions the importance of alignment between digital accessibility with business goals. He notes “accessibility solutions are not for some special group of people, but that such solutions benefit everyone”, implying that businesses and companies do not lose when being inclusive to all, including the less-abled. He mentions Microsoft’s Xbox Kinect (released in 2009) on how it “is being used to help the hearing impaired connect”, thus proving product development can be inclusive with their accessibility whilst providing education and entertainment for users of all ages.

Findings such as these form a foundation for the design of Calculocked. Simplicity within the app by minimising cognitive load and reducing complexities visually aids the user. The design of the app focuses on working with the user, not training the user to do better.

Other Security Interfaces

The main security tools found in the Google Play Store focus on multi-factor/two-factor authentication whilst there is only a small field of apps and research that explores the idea of security through disguised interfaces.

An example to note is [5] “Calculator - Photo Vault” app by FishingNet, which uses a calculator to hide the user’s photo vault. To access the hidden fault, users must input their pin to access it to make the application look normal to unauthorised users. “Vaulty” by Squid Tooth LCC offers a similar service where they also hide media with a disguise.

However, these apps face limitations which my app wants to tackle. The apps mentioned depend on users remembering a passcode or pattern lock, which can cause memorability issues for audiences with cognitive impairments. In addition to this, those applications focus on media privacy rather than notes/texts and user log-in credentials.

As mentioned before, Calculocked tackles this issue by simplifying the unlock mechanism with a triple tap gimmick and replacing passcodes with a personal security question. Rather than complicated authentications and excessive passwords for a password manager -

which is counterproductive to add - this creative system aligns more with a user's personal experience/history.

Mobile Design Accessibility

Designing applications for less tech proficient users such as the elderly requires better notices to accessibility. [7] W3C's Web Content Accessibility Guidelines (WCAG) suggest:

- **High Contrast**: Taken from WCAG 1.4.3
- **Readable Text**: Taken from WCAG 1.4.4
- **Simpler Navigation**: Taken from WCAG 4.1.2
- **Touch Area/Button Size Increased**: Taken from WCAG 2.5.5

Calculocked attempts to incorporate these elements into its UI and UX. Buttons are clearly labelled and colours contrast each other mainly using dark gray and different shades of blue. The usability of the app is also linear:

1. Sign-in
2. Calculator
3. Security: Question and Answer
4. Password Vault

From the password vault users can lock the app to send them back to before the security question or sign-out so they are back to the sign-in beginning. The linear design will minimise user frustration and anxiety.

To add, interactions such as error messages and confirmation prompts are handled using simple language rather than raw errors. Instead of the common "Error 404" which many might not understand, the application informs the user with what the alert is in plain English.

Table of Comparison: Analysis

This is a table on how Calculocked compares itself to competitors in the current market.

Table 1: Application Comparisons

Feature / Application	**1Password**	**NordPass**	**Calculator - Photo Vault**	**Vaulty**	**Calculocked**
Biometric Authentication	Yes	Yes	Yes	Yes	No (Future Improvement)
Media Storage	No	No	Yes	Yes	No (Not relevant)
Credential Storage	Yes	Yes	No	No	Yes
Interface Disguise	No	No	Yes	Yes	Yes
Password/Passcode Free	No	No	No	No	Yes
Accessibility Focus	No	No	No	No	Yes

Table 1 Conclusion

From this table it is evident that Calculocked focuses on accessibility with these features whilst focusing on credential management, taking a combination of what these apps offer to become its own unique project. While it may lack features such as biometric authentication, it will be noted for testing and future development if continued.

Methods

System's Architecture

This section will talk in detail about the technicalities of the application, including its structure, design process and development methods. The approach throughout the development of the application was an iterative design, implementation and testing cycle known as the Action Research Cycle (ARC). This method was used for continuous improvements after planning, observing and reflecting from user feedback to continuously remind itself of the aims of the project.

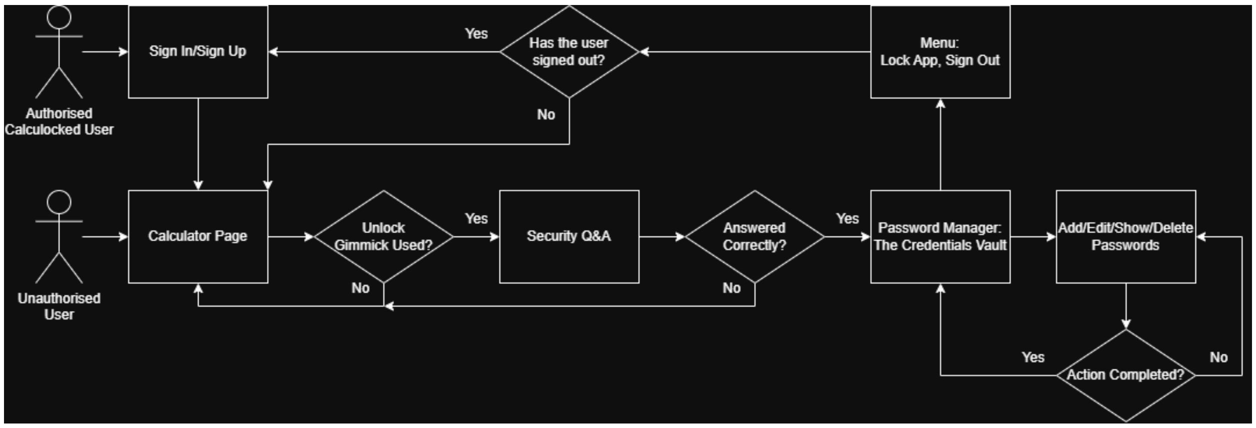
Environments and Tools

- **Language - TypeScript**: A language built on top of JavaScript. Its extra features help prevent mistakes in code by providing type validation, allowing developers to avoid and catch common type errors easily to better maintain code. This makes for a more agile development cycle
    - **Static Typing**: Define the type of data a variable should hold (e.g. numbers or text) to prevent errors.
    - **Error Checking**: Catches errors before running, JavaScript only shows errors at runtime (when the program is being used).
    - **Classes and Interfaces**: Organisational features for clearer code.
    - **JavaScript Features**: Typescript supports all features found within JavaScript but adds extra features for both clarity and safety.
- **Framework - React Native**: A web development framework made to function on native hardware. React Native was chosen to help create a cross platform using a single codebase (collection of code). Using React Native enables developments into other platforms such as iOS, not just solely focusing on Android.
- **Environment for Development**: Visual Studio Code (VSCode), and Firebase for back end data collection.
- **Environment for Testing - Expo Go**: A popular environment when developing apps using React Native. Download the application and scan the QR code on the Android device. Android SDK, provides emulator to test native code
- **Backend - Firebase**: Authentication and real-time data storage.
    - Firebase Authentication
    - Firebase Realtime Database

Overview of Application Structure

Figure 1

Figure 1 showcases a use-case diagram, differentiating the use between an authorised user and unauthorised user after initial sign-up and sign-in. The diagram shows the linear flow of the application use and choices made from authorised and unauthorised users. It should be known that unauthorized users are likely to continue being looped at the gimmick phase, and on a rare occasion, loop at the security Q&A being answered incorrectly.



Files Used

- Screens (Folder)
    - LogUser (Folder)
        - login.tsx
        - signup.tsx
        - welcome.tsx
    - Main (Folder)
        - main.tsx
        - settings.tsx
    - Modals (Folder)
        - DeleteConfirmationModal.tsx
        - ProfileModal.tsx
    - Security (Folder)
        - Calculator.tsx
        - question.tsx
        - SetQna.tsx
- Services (Folder)
    - firebaseConfig.js
- Utils (Folder)
    - Classes (Folder)
        - Profile.ts
    - Types (Folder)

        - types.ts
    - Database.tsx
    - Navigation.tsx

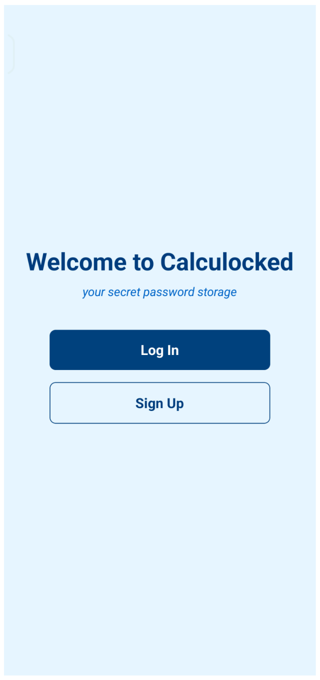
The next sections will attempt to explain each main file with snippets of code in a chronological order when using the app to aid with visualising and understanding the linear flow of the application. All comments have been removed as the snippets will be explained.

All snippets of code will be Calculocked before v.1.0.0., however the screenshots provided for each page will be Calculocked near its v.1.0.0 update.

Welcome Page

screens\LogUser\welcome.tsx

```
const WelcomeScreen = () => {
  const navigation = useNavigation<WelcomeScreenNavigationProp>();
  return (
    <View style={styles.container}>
      <Text style={styles.heading}>Welcome to Calculocked</Text>
      <Text style={styles.subheading}>your secret password storage</Text>

      <TouchableOpacity
        style={styles.button}
        onPress={() => navigation.navigate('Login')}
      >
        <Text style={styles.buttonText}>Log In</Text>
      </TouchableOpacity>

      <TouchableOpacity
        style={[styles.button, styles.secondaryButton]}
        onPress={() => navigation.navigate('SignUp')}
      >
        <Text style={[styles.buttonText, styles.secondaryButtonText]}>Sign
Up</Text>
      </TouchableOpacity>
    </View>
  );
};
```



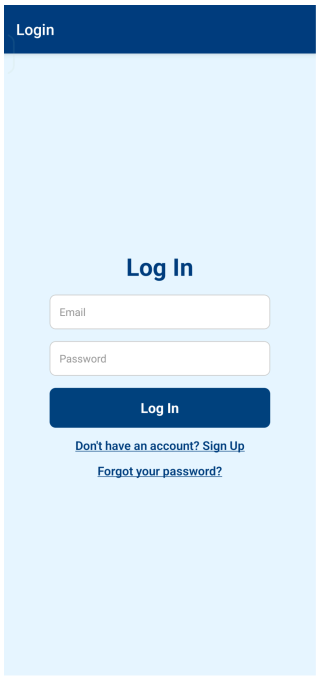
This is your starting default landing screen with clear labels to show what the app is and your two choices to make: Login and Signup. The first page is straightforward and welcoming for all users and does not instantly overload the user with advertisements and extra features. "TouchableOpacity" in this case is used to create two buttons with clear labels and colours to differentiate itself from other titles and texts.

Log-in Screen

screens\LogUser\login.tsx

This screen manages authentication of the user using signInWithEmailAndPassword

```
  const handleLogin = async () => {
    try {
      await auth().signInWithEmailAndPassword(email, password);
      Alert.alert('Success', 'Logged in successfully!');
      navigation.replace('Calculator');
    } catch (error) {
      const errorMessage = error instanceof Error ? error.message :
String(error);
      Alert.alert('Error', errorMessage);
    }
```



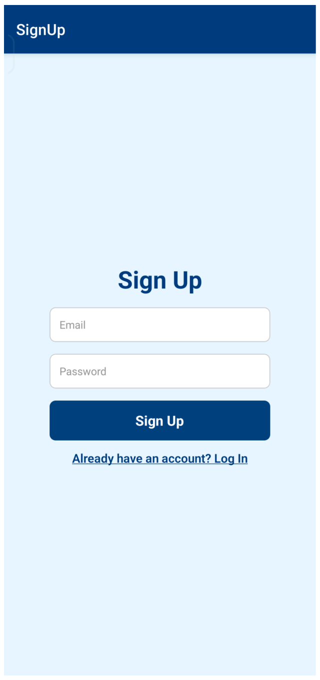
This snippet of code has error handling, being able to differentiate between native and unknown errors using “instanceof” to ensure users understand the error and can forward it to the developer if needed. The users can understand the error through the alert given with the error message attached. A successful login navigates the user to the calculator page whilst alerting the user they have been successful in their log-in. Alerting the user for each crucial decision is a good design choice as the user must stay aware of what is happening throughout their app usage. Since these alerts are easy to dismiss, the user can quickly tap away any alerts once they’re comfortable with using the app on a regular basis. If not, it is not a problem at all and the alerts will always help the user.

Sign-up Screen

screens\LogUser\signup.tsx

```
  const handleSignUp = async () => {
    try {
      await auth().createUserWithEmailAndPassword(email, password);
      Alert.alert('Success', 'Account created successfully!');
      navigation.replace('Calculator');
    } catch (error) {
      const errorMessage = error instanceof Error ? error.message :
String(error);
```

```
      Alert.alert('Error', errorMessage);
    }
  };
```



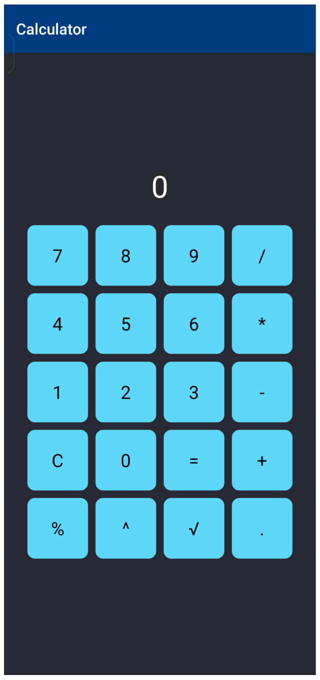
This sign-up screen guides the user, trying to create a new account using “await auth().createUserWithEmailAndPassword(email, password);”. This line tries to create a new account using the inputted email and password. If successful, the user is alerted with a success alert and then is navigated to the Calculator page. If unsuccessful it will be caught by “catch (error)”. The error message will then pop up through the alert and inform the user something went wrong. UX is primarily focused on as the user constantly receives feedback through the sign-up experience and navigation is kept direct from quickly moving to sign-up to the calculator page. It’s short and snappy as it needs to be so the main app can be used instantly.

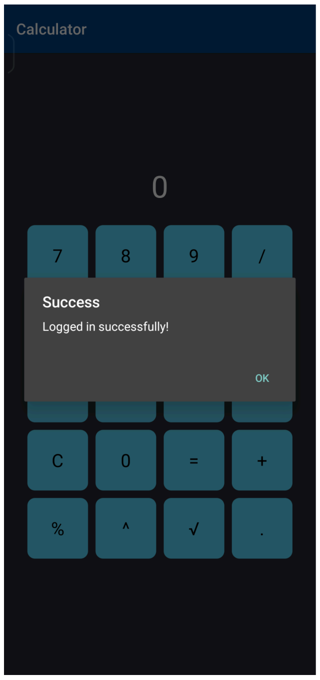
Calculator Page

screens\security\Calculator.tsx

Tutorial Alert

```
 <Modal
        visible={showModal}
        transparent={true}
        animationType="slide"
        onRequestClose={() => setShowModal(false)}
      >
        <View style={styles.modalContainer}>
          <View style={styles.modalContent}>
            <Text style={styles.modalText}>
              Welcome to the Calculator! this is a clever masquerade for
the real function of this app.
              {"\n"}{"\n"} "to learn more tap the equal sign 3 times in a
row.
              {"\n"}{"\n"} Enjoy!
            </Text>
            <Button title="Got it!" onPress={() => setShowModal(false)} />
          </View>
        </View>
      </Modal>
    </View>
```





Success

Logged in successfully!

OK

When entering the calculator page for the first time, the user is notified with a short modalText. A modal is like a pop-up window that appears on the screen to gain the user's attention. This window notifies the user to triple tap the equals sign to proceed to the next part of the application.

Triple Tap Equals Gimmick

```
    if (value === "=") {
      if (lastPress === value) {
        setPressCount(pressCount + 1);
      } else {
        setPressCount(1);
      }

      setLastPress(value);

      if (pressCount + 1 === 3) {
        if (hasQna === "true") {
          navigation.replace("Question");
        } else {
          navigation.replace("Profile");
        }
      }
```

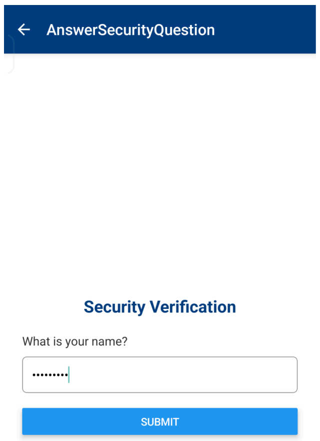
This snippet of code starts with checking if the = sign has been pressed, if it has been pressed then it continues by tracking it and checking if the = button is being continuously pressed. If it is pressed again continuously, it increases the “pressCount” by “+1”. If something else is pressed after = then presscount is set back to 1 through “else { setPressCount(1); }” This is to avoid the gimmick being caught and the app can continue functioning as a normal calculator disguise. The count press is always tracking until it counts to 3 taps, through “if (pressCount + 1 === 3)”, once 3 taps have been triggered it will check for the security Q&A setup and forward the user to the Question and Answer security page. This is the subtle security feature, a blended disguise using user behaviours to access control over the application.

Security Question & Answer

screens\security\question.tsx

Submitting Answers

```
  const handleSubmit = () => {
    if (!userAnswer.trim()) {
      Alert.alert('Error', 'Answer cannot be empty.');
      return;
    }

    if (userAnswer.trim().toLowerCase() ===
storedAnswer?.trim().toLowerCase()) {
      Alert.alert('Success', 'Your answer is correct.');
      navigation.replace('Profile');
    } else {
      Alert.alert('Incorrect', 'That answer is incorrect. Please try
again.');
    }
  };
```



This snippet starts with the handlesubmit function which is used when the user submits an answer within their input box for the Q&A. Upon submission, the first check is on whether the answer box is empty. If so, there will be an error notifying the user that the input box cannot be empty. If not, it will continue to check if the answer is correct by comparing the answer to the correct stored answer. If correct, the user will be notified that the entry was successful and they will be navigated to the main page. If incorrect, the user will be shown they are incorrect and they will have to try again. I have used trimming for answering the Q&A to be forgiving to my user demographic in the validation for the answer. For my user-base, it’s possible to answer the question with unnecessary caps and extra spaces - a common behaviour with the older generation. Removing this can stop answers such as "ADAM" and "adam" from being incorrect when the correct answer is "Adam". This remains as the final security point before entering the vault. When the answer is correct, they are navigated to the 'Profile' screen defined component in main.

Finding Q&A

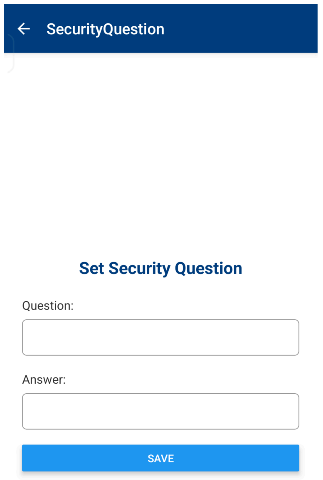
```
  useEffect(() => {
    const loadQna = async () => {
      try {
        const question = await AsyncStorage.getItem('question');
        const answer = await AsyncStorage.getItem('answer');

        if (!question || !answer) {
          Alert.alert('Error', 'No security question found. Please set one
first.');
          navigation.replace('SetQna');
          return;
        }

        setStoredQuestion(question);
        setStoredAnswer(answer);
      } catch (err) {
        Alert.alert('Error', 'Failed to load security data.');
      }
    };

    loadQna();
  }, [navigation]);
```

Before the user answers the security Q&A, this snippet of code redirects the user to the Q&A set-up page assuming there are errors or a Q&A has not been set up.

Setting Security Question & Answer

screens\security\SetQna.tsx

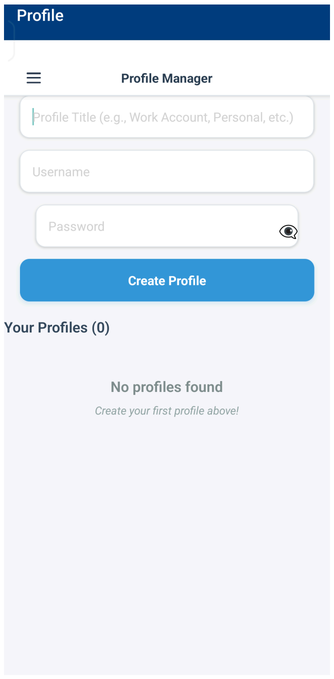
```
  const handleSave = async () => {
    if (question.trim() === '' || answer.trim() === '') {
      Alert.alert('Error', 'Both the question and answer fields are
required.');
      return;
    }

    try {
      await AsyncStorage.setItem('question', question.trim());
      await AsyncStorage.setItem('answer', answer.trim());
      await AsyncStorage.setItem('hasQna', 'true');

      Alert.alert('Success', 'Security question and answer saved.', [
        {
          text: 'OK',
          onPress: () => navigation.replace('QuestionScreen'),
        },
      ]);
    } catch (error) {
      Alert.alert('Error', 'Failed to save your security question.');
    }
  };
```



“handleSave” will be the function used to save the Q&A. The function firstly checks whether the question and answer frields have been filled. If empty, the user will be alerted that they require to fill these fields. The user cannot proceed through the application until this is completed. When both question and answer fields are filled, it saves their inputs with “AsyncStorage” whilst trimming the question and answer. Trimming, as explained before, allows the user to answer their question with unnecessary caps and extra spaces to be forgiving to my user demographic for any unintentional mistakes. AsyncStorage then uses “setItem” to save “hasQna” as ‘true’, meaning the Q&A set up has been completed.

Password Vault - The Main Application

screens\main\main.tsx



Profile

Profile Manager

Profile Title (e.g., Work Account, Personal, etc.)

Username

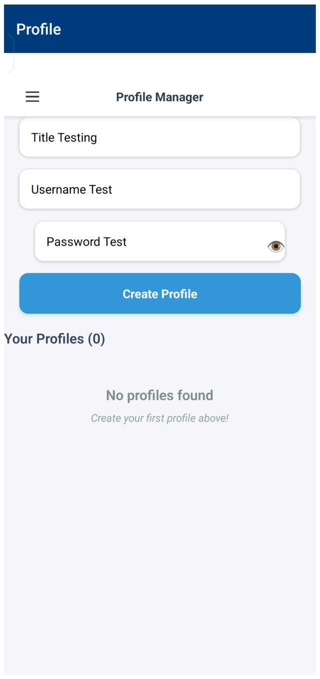
Password

Create Profile

Your Profiles (0)

No profiles found

Create your first profile above!



Profile

Profile Manager

Title Testing

Username Test

Password Test

Create Profile

Your Profiles (0)

No profiles found

Create your first profile above!

Variables

```
const ProfileScreen = () => {
  const [profiles, setProfiles] = useState<Profile[]>([]);
  const [loading, setLoading] = useState(true);
  const [title, setTitle] = useState('');
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');
```

These are variables which store the necessary profile information.

- Profiles stores the array of profiles
- Loading is a boolean value (true or false values) to whether the data is still loading

- Title, Username, Password: Holds values for the new profile the user wishes to create then save.

Profile Fetching

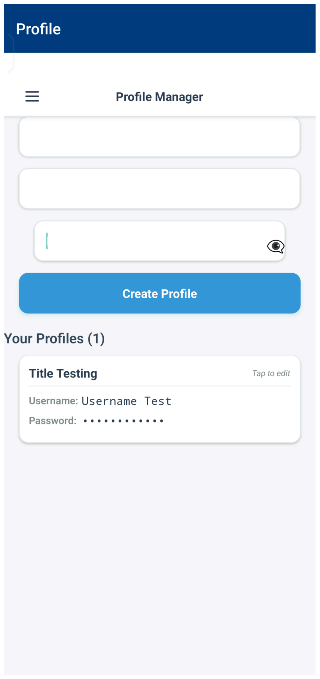
```
useEffect(() => {
  const unsubscribe = firestore()
    .collection('profiles')
    .onSnapshot(
      snapshot => {
        const fetchedProfiles: Profile[] = snapshot.docs.map(doc => ({
          id: doc.id,
          ...(doc.data() as Omit<Profile, 'id'>),
        }));
        setProfiles(fetchedProfiles);
        setLoading(false);
      },
      error => {
        console.error('Firestore onSnapshot error:', error);
        Alert.alert('Error', 'Failed to fetch profiles.');
        setLoading(false);
      }
    );
  return () => unsubscribe();
}, []);
```

Inside useEffect, the Firestore cloud Database is being used to get the list of profiles. The profiles will be in real-time through onSnapshot so whenever data is changed within the database, it will be automatically updated in the application too. Once the profiles are fetched, the profiles are stored in “profiles” using “setProfiles(fetchedProfiles)”. Loading is set to false when the data is ready to be shown. In the case of errors, the user will be alerted with an error and loading will also stop. When the screen is closed, “return () => unsubscribe();” is used to stop the app from listening to the cloud database. This can stop any unwanted updates or leaks.

Creating Profiles

```
const createProfile = async () => {
  if (!title.trim() || !username.trim() || !password.trim()) {
    Alert.alert('Error', 'All fields are required!');
```

```
      return;
    }

    try {
      await firestore().collection('profiles').add({ title, username,
password });
      Alert.alert('Success', 'Profile created successfully!');
      setTitle('');
      setUsername('');
      setPassword('');
    } catch (error) {
      console.error('Create profile error:', error);
      Alert.alert('Error', 'Failed to create profile.');
    }
  };
```



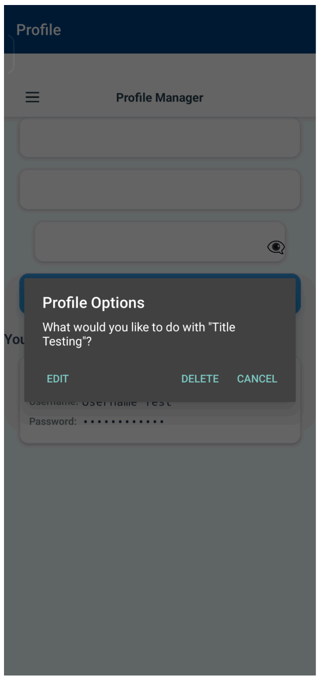
Your Profiles (1)

Title Testing — Tap to edit
Username: Username Test
Password: •••••••••••••

The function “createProfile” is used to save the user credentials into Firestore. It starts by checking if all fields are filled in. If not, the user will be alerted with an error message. If the fields are filled in, the new profile will be saved into Firestore and it clears the input fields ready for the next input + save when it’s reused. Assuming there is an error, the user will be alerted that the profile creation failed.

Deleting + Editing Options

```
const handleProfilePress = (profile: Profile) => {
  Alert.alert(
    'Profile Options',
    'Choose an action',
    [
      {
        text: 'Edit',
        onPress: () => {
          promptEditProfile(profile);
        },
      },
      {
        text: 'Delete',
        onPress: () => {
          deleteProfile(profile.id);
        },
      },
      { text: 'Cancel', style: 'cancel' },
    ],
    { cancelable: true }
  );
};
```



Profile Options

What would you like to do with "Title Testing"?

The “handleProfilePress” function starts with presenting the user with options on either editing or deleting the app. Depending on selected choice, the user can either delete profiles through “deleteProfile” or edit profiles through “promptEditProfile”.

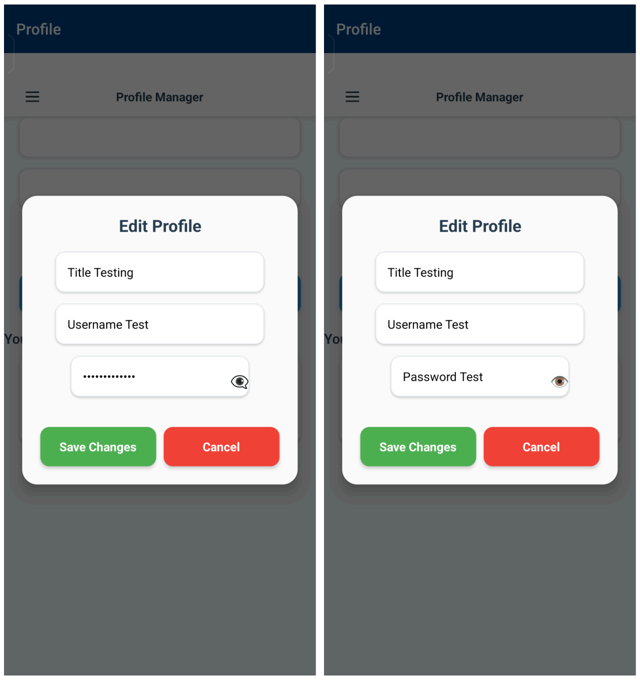
Deleting Profiles

```
const deleteProfile = async (id: string) => {
  try {
    await firestore().collection('profiles').doc(id).delete();
    Alert.alert('Success', 'Profile deleted successfully!');
  } catch (error) {
    console.error('Delete profile error:', error);
    Alert.alert('Error', 'Failed to delete profile.');
  }
};
```

The deleteProfile function deletes the profiles from Firestore based on its saved id. Whether deleting was successful or not, there will be a user alert notifying the user on the result. If there is an error, the console logs this.

Editing Profile

```
const promptEditProfile = (profile: Profile) => {
  Alert.prompt(
    'Edit Profile',
    'Enter updated title, username, password (comma separated)',
    [
      {
        text: 'Cancel',
        style: 'cancel',
      },
      {
        text: 'OK',
        onPress: input => {
          if (typeof input === 'string') {
            const [newTitle, newUsername, newPassword] = input
              .split(',')
              .map(s => s.trim());
            if (newTitle && newUsername && newPassword) {
              updateProfile(profile.id, {
                title: newTitle,
                username: newUsername,
                password: newPassword,
              });
            } else {
              Alert.alert('Error', 'All fields are required!');
            }
          }
        },
      },
    ],
    'plain-text',
    `${profile.title},${profile.username},${profile.password}`
  );
};
```



“promptEditProfile” shows a prompt providing the user with new input boxes to update their existing profile details. This includes all fields including title, username and profile: “const [newTitle, newUsername, newPassword]”. Once new inputs have been detected through “if (newTitle && newUsername && newPassword)”, the profile will update using “updateProfile(profile.id,” after pressing “OK”. If there are any errors, the user will be notified with an alert.

Updating Profile

```
  const updateProfile = async (id: string, updatedData: Partial<Profile>)
=> {
    try {
      await firestore().collection('profiles').doc(id).update(updatedData);
      Alert.alert('Success', 'Profile updated successfully!');
    } catch (error) {
      console.error('Update profile error:', error);
      Alert.alert('Error', 'Failed to update profile.');
    }
  };
```

The updateProfile function allows profile editing by updating details in Firestore, searching for the profile and then saving using its id. Similar to profile deletion; regardless of whether the updating was successful or not, there will be a user alert notifying the user of the result. If there is an error, the console logs this.

Profiles: Creation and List

```
return (
    <View style={styles.container}>
      <Text style={styles.heading}>Create Profile</Text>
      <TextInput
        style={styles.input}
        placeholder="Title"
        value={title}
        onChangeText={setTitle}
      />
      <TextInput
        style={styles.input}
        placeholder="Username"
        value={username}
        onChangeText={setUsername}
      />
      <TextInput
        style={styles.input}
        placeholder="Password"
        value={password}
        onChangeText={setPassword}
        secureTextEntry
      />
      <Button title="Create Profile" onPress={createProfile} />
      {loading ? (
        <ActivityIndicator style={{ marginTop: 20 }} size="large"
color="#000" />
      ) : (
        <ScrollView style={styles.scrollView}>
          {profiles.map(profile => (
            <TouchableOpacity
              key={profile.id}
              onPress={() => handleProfilePress(profile)}
              style={styles.profileButton}
            >
              <Text style={styles.profileText}>{profile.title}</Text>
            </TouchableOpacity>
          ))}
        </ScrollView>
      )}
    </View>
  );
};
```

“<View>” is the main container that holds everything on the screen. “<TextInput>” allows the user to type in their desired profile title, username and password, using “placeholder” to indicate where each input should be placed.

Settings

screens\main\settings.tsx

Resetting Security Q&A

```
  const handleSetupSecurityQuestion = () => {
    navigation.navigate("SetQna");
  };
```

```
      <Button
        title="Change Security Question"
        onPress={handleSetupSecurityQuestion}
        color="#007BFF"
      />
```

The function “handleSetupSecurityQuestion” navigates the user back to SetQna to restart the Q&A setup process. This can be activated by pressing the “Change Security Question” button through “onPress”.

Sign Out

Delete Profile Modal

screens\modals\DeleteConfirmationModal.tsx

```
interface DeleteConfirmationModalProps {
  visible: boolean;
  onClose: () => void;
  onConfirm: () => void;
}

const DeleteConfirmationModal: React.FC<DeleteConfirmationModalProps> = ({
  visible,
  onClose,
  onConfirm,
}) => {
  return (
    <Modal transparent visible={visible} animationType="fade"
onRequestClose={onClose}>
      <View style={styles.overlay}>
        <View style={styles.modalContainer}>
          <Text>Are you sure you want to delete this profile?</Text>
          <Button onPress={onConfirm} title="Yes" />
          <Button onPress={onClose} title="No" />
        </View>
      </View>
    </Modal>
  );
};
```

DeleteConfirmationModal is a functional component that shows a modal window asking the user if they want to delete something. “Visible” is a boolean value, setting to true and false, on whether it should be visible or hidden. The void value “onClose” closes the modal after the no choice is made. The void value “onConfirm” closes the modal after the yes choice is made. Void values do not return values back, they are used for actions such as closing and confirming. Creating the modal separately as its own creates modularity in this application, allowing code to be reusable and easier to manage.

Profile Modal

screens\modals\ProfileModal

```
const ProfileModal: React.FC<ProfileModalProps> = ({
  visible,
  onClose,
  onSubmit,
  profile,
}) => {
  const [title, setTitle] = React.useState(profile?.title || "");
  const [username, setUsername] = React.useState(profile?.username || "");
  const [password, setPassword] = React.useState(profile?.password || "");

  const handleSubmit = () => {
    const newProfile = new Profile(profile?.id || "", title, username,
password);
    onSubmit(newProfile);
    onClose();
  };
```

The functional component “ProfileModal” shows a pop-up window to edit or view profiles, using the properties “visible”, “onClose”, “onSubmit”, and “profile”. “title” , “username”, and “password” variables hold values of the profiles that are being created or edited at the moment. “React.useState(profile?.title || "")” ensures profiles that are not empty will use the “title” variable to set its value, and if it is empty then it will set the value to an empty string. “handleSubmit” function submits the form after the “Update Profile” or “Create Profile” button is pressed. It either creates a new profile object with the added values or updates the values of the existing edited profile. It then calls “onSubmit(newProfile)” which forwards the updated profile to where the form is being used so it can be saved and shown correctly in the app’s display for profiles. Finally, “onClose()” is called to close the modal after profile submission. This modal allows users to easily edit profile details as a modal without navigating away from the current screen. Using the onSubmit and onClose functions allow it to be reusable in other parts of the application that handles submitting and closing modals too, e.g. delete profile modal. Overall these modals are user friendly as they provide clear and simple user interfaces for users to update their profile without being overloaded by multiple unnecessary features.

Services

services\firebaseConfig.js

```
import { initializeApp } from '@react-native-firebase/app';
import firestore from '@react-native-firebase/firestore';

const app = initializeApp();

export { firestore };
```

This snippet of code sets up Firestore database services for the data management and exports for the use of the application.

- **Imports**: Imports the functions to set up Firebase within the application
- **Initialising**: Connecting and starting up Firebase for the application
- **Export**: Makes Firestore services available to use in other files so that other parts of the application can interact with the database.

Profile Object

utils\classes\Profile.ts

```
class Profile {
  id: string;
  title: string;
  username: string;
  password: string;
  constructor(id: string, title: string, username: string, password:
string) {
    this.id = id;
    this.title = title;
    this.username = username;
    this.password = password;
  }
}
export default Profile;
```

The class titled “Profile” is used to create objects with properties including “id”, “title”, “username”, “password” all as strings - sequences of characters inc. letters, numbers and symbols. Exporting “Profile” allows the class to be accessed by other files within the app so it can be used. Separating this makes it easier to maintain and manage credentials between screens in the application.

Defining Data Types

utils\types\types.ts

```
export type RootStackParamList = {
  Welcome: undefined;
  Login: undefined;
  SignUp: undefined;
  Calculator: undefined;
  Question: undefined;
  Security: undefined;
  ProfileScreen: undefined
};
```

“RootStackPeramList” is used to define names of the screens and what parameters are expected. All of them are undefined as they do not require extra data such as ids. An example of if they need ids would be “ProfileScreen: {id: string}”. This is used to prevent errors as TypeScript catches errors when compiling. This overall improves the quality of the code and reduces bugs.

App Navigation

utils\Navigation.ts

```
        {/* XYZ Screen */}
        <Stack.Screen
          name="XYZ"
          component={XYZScreen}
          options={{
            title: 'XYZ',
            headerStyle: { backgroundColor: '#004080' },
            headerTintColor: '#fff',
          }}
        />
```

Nearly all of these screens follow this exact structure. “<Stack.Screen>” is a navigation pattern in React Native where screens are on top of eachother. Proceeding to a new screen moves from one screen to the next screen below it. Pressing back will take you back to the screen before it.

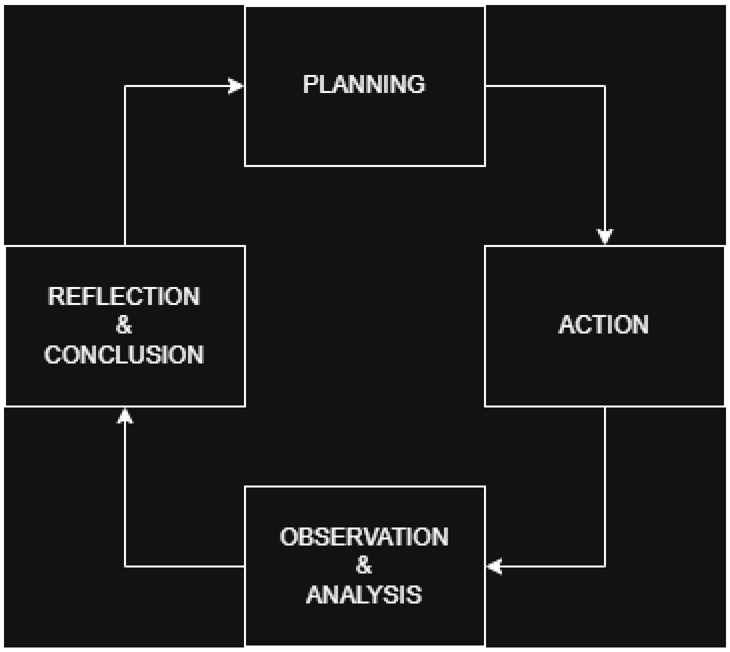
Example

1. Start on home screen
2. Moved to the profile screen, it is now stacked on top of the home screen.
3. Pressing back takes you to the home screen that was removed from the stack, removing the profile screen.

Process for Design & Methods for Development

Calculocked development was inspired by the Action Research Cycle. ARC uses iterative improvement from its process: planning, action, observation, reflection. This approach matches the project goals the best as it would allow full focus on the target demographic and understanding their weaknesses to then build upon that. Unlike other methodologies, ARC allows the application to improve in response to personal insights and user feedback after use.

ARC Diagram



Step 1: Planning

Choices and priorities are made clear through research and general assumptions. An example would be the early concepts of the application where calculator disguises and behaviour was the only security to unlock the application.

Step 2: Action

This phase is to work on the application and implement the features planned previously. This phase may also spark new ideas and have you going back to step 1 to add new material to later implement them.

Step 3: Observation & Analysis

This phase involves testing for functionality then collecting qualitative feedback from users testing the app. Continuing from the previous example, this would be to pay attention to how users interacted with the calculator feature and then how they managed credentials. From this we observe if they faced any problems and what they liked or disliked about the process.

Step 4: Reflection and Conclusion

After user and unit testing is complete, both positive and negative feedback is collected. To finalise the example, from user feedback it was apparent that users wanted more security to which we implemented the Q&A feature. This phase should be open to complete criticism so step 1 can be effective again and the cycle continues.

Conclusion

This iterative loop worked well throughout the project development, allowing Calculocked to continuously improve and align with the expectations of all users, especially in the target audience. Due to this, the app ensured it was continually reminded and focused on its target goal to provide a safe and accessible cybersecurity tool.

Because of these iterative improvements, results were easily obtained throughout for both functional performance and responses from users.

Results

This section of this report discusses the functionality and usability outcomes received from user and unit testing phases of the project.

Unit Testing

Feature	Description	Expectation	Result
Sign-Up	Account creation for sign in	Account now saved and session begins	Pass
Log-in	Sign in using correct credentials	Log-in successful, user sent to Calculator screen	Pass
Unlock Gimmick	Triple press equals sign	Access Q&A prompt	Pass
Q&A Creation	Create Q&A as as security layer	Save Q&A and access password manager	Pass
Security Q&A	Retrieves question,	Correct: access vault	Pass

Prompt	enter correct answer to access vault	Incorrect: Try again	
Add Profiles	Create profiles with title, username and password	Added to database, new profiles are presented on screen	Pass
Edit/Delete Profiles	Tap to edit or delete saved credentials	Changed/removed information from database.	Pass
Lock App	App is sent away from the password manager page	App state is reset to Calculator	Pass
Settings	Display option buttons	Display option buttons	Pass
Sign Out	User is logged out	Sent back to the welcome page, user is no longer signed in	Pass

User Testing: Procedure

User testing involved a small group of 5 participants varying with technical proficiency - including those within my target audience, excluding myself. In most stages of testing, Calculocked used Expo Go to launch and test the application. This was set-up for the tested users before starting.

For their first ever test, I made sure to not tell the users about the calculator gimmick to see if they can try and find the unlock mechanism, to which everyone thought would be typing a series of numbers as a passcode.

Each participant was given a set of instructions to follow during testing:

1. Create a new account
2. If already created, sign in
3. Pass the calculator gimmick
4. Set question and answer security layer
5. If already set, answer security question and answer prompt

6. Manage credentials: save/edit/delete profiles

Their process and voices were recorded during the whole process from start to finish, along with a recorded interview.

Post-Testing Interviews

Interviews taken after testing provided valuable qualitative data to further improve the application, but also took notes of the accomplishments of the application. Users were asked questions including:

- What was the easiest part of using the application?
- What was the hardest part of using the application?
- Would you recommend this to others just like you?
- What would you change if you could improve this application?

All participants agreed the easiest part of the application was the triple tap gimmick on equals sign, although they understood that the calculator functioned as any other calculator from their very first test.

Although feedback proved everything was relatively easy, the majority of the answers were setting up the Q&A mainly due to being unsure of what to make of their Q&A. It was a matter of people being picky. From this it was easy to tell people would rather be given options as well as have the freedom to create options for themselves. Having choice is what people truly value when it comes to personalised security.

4 out of 5 participants said they would recommend this application to others. 1 participant, within the younger audience, said they would not recommend this as they tend to already remember all their passwords that fit most criteria that websites ask for (capital letters, symbols, etc.)

A change all users recommended is biometrics. Some suggested it as a layer of security on-top of the Q&A whilst others suggested it as an alternative layer. I was informed it would be good as a second layer by the younger audience of security to push away those who may know the answer to the security question and answer such as family or friends if they were to snoop on a personal device. My target demographic suggested it be an alternative layer as they find it easier to scan their thumb-print rather than answering the Q&A. In future developments, it may be noted that biometrics can be chosen as an extra layer, an alternative or just toggled off if necessary.

Quoted Feedback

Participants were ordered in terms of age, 1 being youngest and 5 being oldest.

Participant 1

Good feedback: "People don't usually expect to find account details in calculator apps."

Improvement: "I feel like a lot of close friends would pass my question and answer."

Participant 2

Good feedback: "I appreciated the alerts, it was surprisingly guiding"

Improvement: "If this was on app stores, wouldn't everyone know what Calculocked looks like?"

Participant 3

Good feedback: "The colour choice on the app is soothing."

Improvement: "Is the app only going to be a calculator screen? Or will there be more at later dates?"

Participant 4

Good feedback: "I can finally stop asking others to help with my accounts"

Improvement: "Is there any way I can share this account with those I trust?"

Participant 5

Good feedback: "I don't need to write my passwords in my notes app anymore"

Improvement: "Is there any way to automatically add accounts like Google does?"

Quotes Feedback - Review

Generally, the younger audience focused a lot more on the security side on their main quoted feedback. They wished for more security layers and customisation. To make Calculocked appeal to all audiences rather than just a target demographic, the application will have to implement new ways to add optional security layers without bombarding the target demographic.

The older audience wanted to rely a bit more on easier saving. Participant 4 wishes to have the details shared, perhaps an idea is to have a master-account/family-shared section where users are linked together so they can share details such as a Netflix account or a Spotify account but also have a section for personal accounts such as social media accounts which won't be shared between users. The risk with this is overcomplicating the app.

Statistics for Task Completion

All results shown in Table 2 are the tests with the recent build of Calculocked, proving participants of varying backgrounds are able to complete their tasks independently by the end of the testing session.

Note: All their relevant information such as their emails were given to them already.

Table 2: Task Completion Stats

Task Completed	Completion Rate	Average Time Taken	Notes/Extras
Account Creation	5/5	~43 seconds	Some users were slower, due to typing email and passwords
Calculator triple tap gimmick	5/5	~5 seconds	Straightforward
Setting Q&A	5/5	~36 seconds	Some questions were longer than others
Answering security question	5/5	~11 seconds	All answers were correct, one anomaly with a typo.

Adding new profile	5/5	~29 seconds	N/A.
Editing existing profiles	5/5	~ 17 seconds	Varied between the number of details changed. Had to explain that profile must be clicked on to edit it
Deleting profiles	5/5	~7 seconds	Straightforward

Iterative Updates

Through iteration over the long period of app development, several improvements were made to maintain user trust, performance and clarity. This includes:

- Encryption
- Alerts
- Clearer interface
- Colour scheme

Results Summary

Through the stages of the Action Research Cycle, the project has basically reached completion as Calculocked v1.0.0. This current application not only serves as proof that cybersecurity can appeal to all users and demographics when users who are less tech-able are targeted, but it can be kept simple yet effective. This can be used as a strong foundation for other project ideas and research involved in accessible cyber-security for all.

Discussion

This section of the report offers analysis of my results from the previous section, keeping in the contexts of cybersecurity and mobile app design. Correctly assessing whether Calculocked fulfils its purpose and addresses the problems it wishes to overcome is the goal.

The Problem: Was it addressed?

The problem Calculocked wished to solve is very specific yet remains relevant: “how can those with lesser digital fluency store and access private log-in information safely without being overwhelmed by design complexity?”
The answer does not rely on advanced ideas such as cryptic messages and AI implementation, rather it was found in simplicity through familiarity and seclusion.

Password managers traditionally work around the idea that increasing the number of features results in better security. This may be true in other technical areas however it does not apply when usability and accessibility is the issue. It is common for users to avoid complex systems, often writing their passwords in physical or digital notepads/paper that are obviously less safe. These unsafe regular practices cancel out the benefits that current developed software provides for users.

A different approach is taken by Calculocked. It simplifies tools by hiding its main features through a camouflage (the calculator). This idea comes from camouflage computing, where an app hides its true purpose to reduce effort yet improve privacy in an easier way.

This project shows that innovative design isn’t about adding complexity and overloading the application with features, but removing obstacles that may be deemed unnecessary. It focuses on easy to understand actions that align with everyone’s experiences, whether past or current. Pressing a calculator button for all users is something done countless times. The “triple tap to unlock” gimmick is not just an action but a familiar practice all users can achieve.

Innovative Design: Disguise

Camouflaged computing through vault-styled applications to hide sensitive and personal data such as media and notes is not a new concept but it hasn’t been applied to password management the way Calculocked has done it. Other vault applications tend to hide media with their own camouflage and primarily focus on younger audiences that understand the value of privacy.

Participants in the user testing phases often described the app as “safe”, “an ease of mind”, and “less intimidating”. These words and phrases are rarely used when looking into tools for digital security, however these descriptions hold importance when promoting its use and easing anxiety for people.

Taking all this into account, Calculocked fits itself in this niche where it prioritizes cybersecurity and technology to assist a less-abled audience. Looking at previous studies and research mentioned, Calculocked attempts to contribute within those areas that emphasises in better accessibility designs in security software. Systems should be inclusive to use as well as secure.

Evaluation: The Security Model

Security apps should not be lacking with only a basic. It should focus on protection and making sure the user is comfortable with using the service. In Calculocked's initial implementation, the sensitive credentials were saved in Firebase's Realtime Database as plaintext. This was a huge security risk as unauthorised access to the backend would be a breach of privacy to those users, regardless of the protection in the frontend.

Development after this has fixed this by encrypting the credentials in the backend. This means even if Firebase faces security issues, stored data remains unharmed without the decryption logic and key. This improvement fits with the principle of end-to-end encryption and significantly increases the user trust on the application.

Regardless of the problems faced with encryption, biometric authentication remains unimplemented. The released model relies on a behavioural gimmick and security question, however this does have a key weakness such as unauthorised access having familiarity with the user to bypass the Q&A.

Integration of biometric options such as fingerprint or facial scan as an alternative or secondary method to unlock the password manager would not just improve the security but enhance usability for users with extremely limited memory. This would make Calculocked with current industry's best practices without compromising its focused design to remain accessible to all.

Accessibility: Does my project match the criteria?

The foundation of the application relies on its accessibility. Every screen and interaction between the user and the application is designed with needs to meet the target audience: the elderly and less-abled physically or mentally. Looking at theWeb Content Accessibility Guidelines, the application attempts to match its criteria.

1. **Visuals**: Contrasting colours, larger buttons
2. **Mental Accessibility**: Minimalist design, screen not overloaded with features, clear labels

The success the application achieved in testing shows that criteria such as these were implemented well. However, some accessibility features could be included.

- Text to speech software
- Adjustable text sizes and fonts
- Custom colourblind themes
- Light/dark mode?

These features are not yet developed but remain noted for future work.

Ethics: Is the design empathetic?

Older users can feel anxiety when using apps with sensitive data especially with rising cases of hacking, phishing and general scamming which targets that demographic. Calculocked aims to focus on emotional safety. The hiding functionality is like hiding the anxiety. The app does not ever scarily alert the user.

The app remains ethical as it allows users to have full control over their personal data without breaching their privacy. The app is simple to not rely on external guidance. All testing was conducted with full transparency meaning users were aware of the app's potential flaws and strengths and consented to testing and post-testing interviews. The production of the application always continued to follow ethical practices throughout.

Comparisons: Current Solutions vs My Solution

Mentioned in previous sections, Calculocked and vault/password tools such as LastPass, 1Password, Calculator Photo Vault, and Vaulty were compared, notably in Table 1. These comparisons were primarily focused on differentiating their features, but a deeper understanding into my project reveals Calculocked is not competing in the same category as these other alternatives. Rather than identifying as just a full-service password manager, it presents itself as a lightweight and accessible vault app for users who may not require or want complex features.

In regards to this, it is more appropriate for Calculocked to be compared to assistive tools rather than multi-functional tools and systems. By accepting this more focused ideal, Calculocked has been able to refine all its features and reliability without straying from its main goal.

Limitations: What is lacking in this release?

- **Simple Unlock Mechanism**: If this app was to be publicly released and gains success, the unlock gimmick would be widely known, making it ineffective.
    - *Possible Solution*: Customised gestures + customised front-page + biometrics
- **No iOS Testing**: Limitations on cross-platform testing means there could be iOs-specific bugs
    - *Solution*: Test on iOS users
- **No Offline Mode**: Credential access and saving depends on cloud availability
    - *Possible Solution*: Create versions implementing local caching/storage, which then updates once the device is online again.

Research: The possible future within cybersecurity

The development of Calculocked expands on current industry and academic research.

1. **Disguise as security**: How can techniques such as vaults which hide information be used in other types of applications?
    a. These techniques can be added to various applications such as banking, health or social media and communication tools. Such important apps can

often have overwhelming security upon app entry so implementing optional simpler ideas can improve the usability whilst trying to maximise its security.

2. **Credential management for the memory-challenged**: What are methods that make it easier to remember passwords whilst keeping them secure and effective for those with impairments?
    a. Approaches that are useful for individuals with impairments include methods with simple verification such as biometrics or memorable yet unique passphrases. This allows the maintenance of secure access without the struggle to remember complex passwords and passcodes.
3. **Privacy digitally for more vulnerable populations**: How does culture and emotions affect the trust between humans and the solutions technology provides?
    a. Individual experiences and culture in regards to breach or acceptance of privacy does show differences in trust with what technology provides.
    b. It is a matter of teaching the importance of privacy and teaching all regardless of culture that digital privacy *is* an extension of basic privacy and should be viewed as a normal human right.
4. **Designing cybersecurity tools to become more inclusive**: What are some of the best ways to create security applications for those who have limited skills and experience with technology?
    a. Creating inclusive interfaces in security should become a common practice. Navigation should be clear, visuals should be simpler and apparently whilst large paragraphs/texts should be kept to a minimum. These changes ensure those with technology limitations can understand and manage their security easily without becoming confused or intimidated by long complicated features or instructions.

Limitations and Future Work

Software, especially in early-stage development, faces challenges including scope, time, skill, available technologies and work-force. Working on Calculocked as a solo project has shown both limitations of the app but also brought new ideas for future work based on personal ideas and user feedback.

Dependency on Firebase

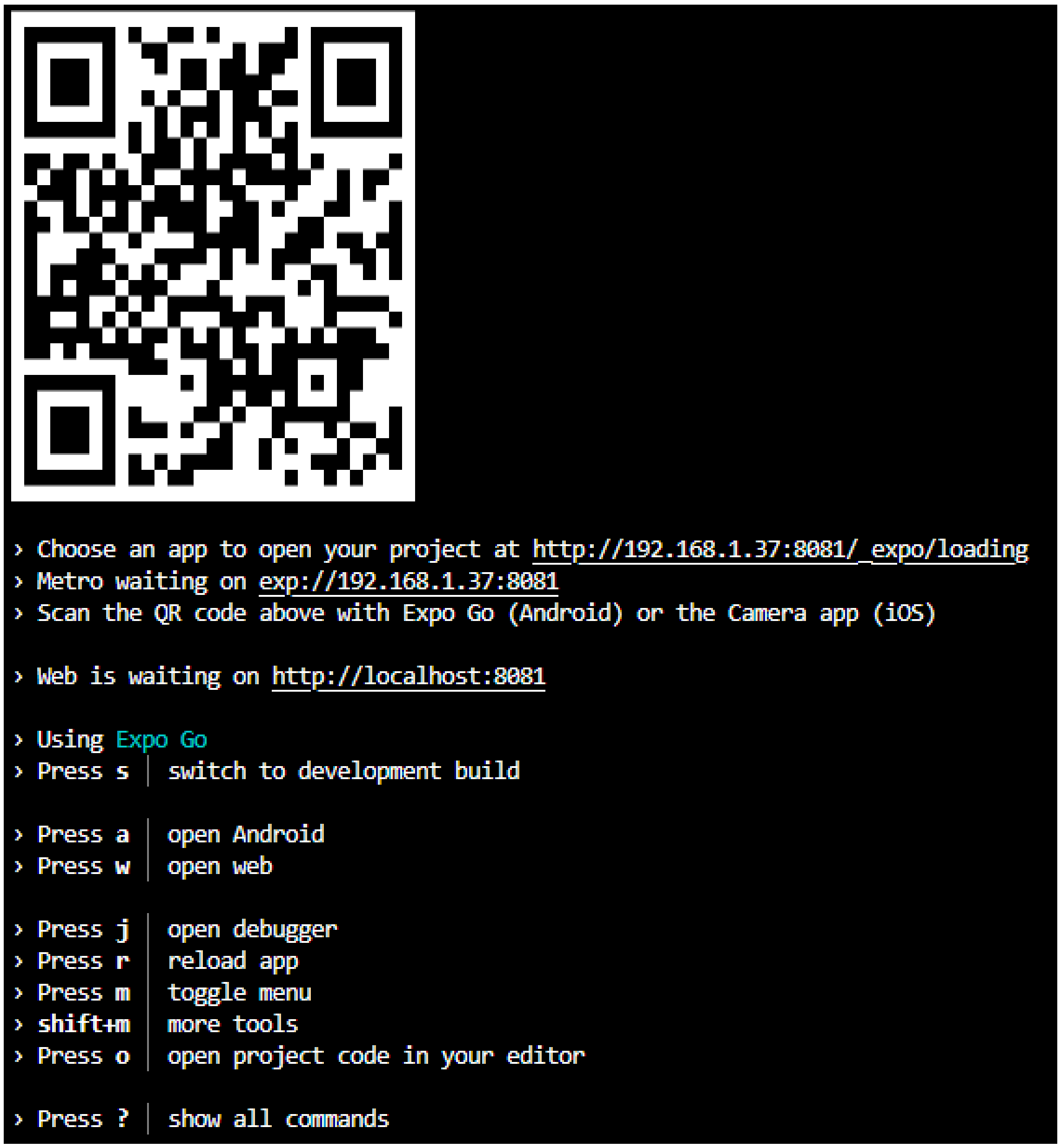
Calculocked currently depends on Firebase for its authentication and data storage. This is a significant drawback to the application as solely depending on a third party service can lead to many possible problems:

- **Pricing changes**: Google services aren't shy to charge for subscriptions and tiers.
- **Discontinued or Deprecated**: Google services can always be deprecated such as Google Cloud Print. Firebase Dynamic Links service will be shut down on August 25, 2025.
- **GDPR**: Users want full control of their data or ensure it follows GDPR regulations.

While Firebase does provide convenience and long-term scalability, an application of higher production level would ideally not be solely dependent on a singular backend, being available to other options.

Limited Platform Availability

Currently, the application that has been developed has only been tested on Android devices using Expo Go only. As a result of this, the user experience for iOS remains untested. This is a huge limitation as it may develop device specific issues such as buttons not working correctly or fonts displaying incorrectly. Implementing biometric features may also arise as iOS and Android will have different biometric technologies and it will behave differently which will affect future developments for this application.



```
› Choose an app to open your project at http://192.168.1.37:8081/_expo/loading
› Metro waiting on exp://192.168.1.37:8081
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Web is waiting on http://localhost:8081

› Using Expo Go
› Press s │ switch to development build

› Press a │ open Android
› Press w │ open web

› Press j │ open debugger
› Press r │ reload app
› Press m │ toggle menu
› shift+m │ more tools
› Press o │ open project code in your editor

› Press ? │ show all commands
```

Lack of Biometrics

Biometric features such as fingerprints and facial recognition are standard security updates on nearly all mobile devices now. Calulocked only relies on behavioural components and a simple Q&A as a layer of security for access to private information. Whilst accessibility is improved, it does sacrifice a layer of security that users may expect.

To counter this, in later versions it is possible to add biometric features after the triple tap gimmick. After passing the calculator page, the user can then use biometrics to unlock the vault instead of the security question. The security Q&A can be a back-up incase there are problems with the biometric bypass, such as shaking during the fingerprint scanning or wearing gloves/masks. This does implement a better security layer without taking away the accessibility aim that Calculocked wishes to achieve.

Unlock Gimmick

Currently the app uses triple pressing “=” to unlock the vault. Although it is a creative gesture, if the app was to be released publicly it's probable that unauthorised users can recognise the app design and have a go at cracking the Q&A.

To counter this, it’s possible that future designs can create “skins”, where users can create their own calculator design or even stray from a calculator front-page and display other productivity apps such as calendars and clocks which can also be customised. Being unique in these front pages makes it less likely for people to realise it’s a Calculocked application.

Offline Functionality

Calculocked relies on internet connections to access the user's credentials through Firebase. This can simply be changed by storing an encrypted cache of the credentials locally, then syncing the data by adding/editing/removing the data accordingly when the device is online again. Alerting the user is offline when the vault is unlocked is essential whilst explaining the risks of data not being synced to make them aware that changes may not happen if there are any errors within the code.

Tutorials

Upon testing, it was evident that a better beginner friendly tutorial is needed rather than a text box alert. When testing the application it was apparent that I had to explain how to use the application for the user to understand. If this app was to be launched, it’s possible it won’t appeal to my target audience as they may not understand the full app’s function since a lot of instructions may have to be described in the installation description rather than upon entering the application. On top of this, a help button can be added in the burger menu with a full guide on app usage, FAQs and general digital safety advice.

Personalised and Enhanced Accessibility Features

To serve the target audience better, alongside appealing to other audiences, the app can include the following:

- **Font Sizes**: Slider in settings to change font size to either increase or decrease
- **Dark Mode**: Buttons to change calculator and password manager from white background to black background, or to match system default.
- **Color Blind Palettes**: Option to change secondary colours to colours for different types of color blindness (deutan,protan, tritan)
- **Haptic Feedback**: Sensory feedback (buzzing) for confirmation on important actions such as saving and deleting profiles, or even locking the application.

These features will make the application not just usable, but encourage users facing disabilities or sensory issues to have greater independence.

Conclusion

As digital dependency increases, responsibility for managing sensitive information such as credentials for accounts have solely been put on individuals without being provided tools to help with this - especially those who are of an older generation, those who suffer from disabilities or those who are simply unfamiliar with security practices to a good standard. Mainstream password managers to prioritise functionality but they lack the ease of use and accessibility for the demographic I have mentioned. The exclusion of this group is the very group that will benefit from applications such as password managers.

Calculocked was developed as a direct response to this gap which started from a personal anecdote. Built using React Native and Firebase as its backend, the application is a sufficient tool that utilises different ways of security to cater to its audience: behavioral authentication (triple tap gimmick) and knowledge-based authentication (security question and answer prompt). This unique design preserves the privacy of the user without any overbearing complex authentication methods.

Throughout its development, Calculocked always attempted to adhere to its principles of maintaining user-centered design and accessibility through iterative refinements. The application uses familiar features such as a calculator (which can be found as a default application on all devices), and simple design with coloured and correctly labeled buttons and alerts for contrast with a simple linear flow from page to page to be more inclusive. From applying Action Research Cycle (ARC) methodology, feedback from users were a direct influence for improvement. This includes visual cues, error prompts, and refinements in the layout of the app. Collected feedback from usability testing confirmed the concept: not only did users understand and appreciate the approach of the app, but also reported safety and empowerment whilst using Calculocked.

Viewing the app from a technical standpoint, Calculocked proves it can handle authentication, storage of credentials and real-time vault operations (add/edit/deleting credentials). While the project does show its limitations - as do all projects on initial release - such as offline functionality and better tutorials further its safety and accessibility, it does not take away from its main goal to be an accessible password management application that is approachable, secure, and reassuring for my target demographic yet inclusive to those outside of that audience.

Beyond the functionality of Calculocked, the app begins to open new conversations in the cybersecurity industry:

1. What does it mean for user interfaces to be secure, not just technically but in terms of empathy - human trust?
2. Can behavioural and camouflaged designs be effective tools in this current age of technology and security?
3. How can tools be designed for users who have been excluded from societies which are less reliant on tech and are beginning to transition to a more reliant technical society?

Questions such as these only show Calculocked does not only wish to solely be an app, but a starting change to the security field where security should be inclusive to all regardless of technological proficiency and tools should be built to respect that whilst preserving their online safety.

The app in its current position has proven to be lightweight, accessible and innovative. If further developed through encryption, integration with biometrics, and further supportive features such as tutorials, Calculocked has infinite potential and can become a widely trusted tool especially for those who struggle and fall behind in cybersecurity knowledge.

As society continues to further its digital complexity at a rapid pace, tools such as Calculocked become more of a necessity rather than an option. Digital security should be extended to all audiences, not just those who consider themselves to be “tech-savvy”. My project Calculocked represents this message, it is a step towards realising digital security for all is the ideal.

Bibliography: Sources/Citations

1. Office for National Statistics. (2020, October 19). Mobile phone theft in the UK per annum. Office for National Statistics. https://www.ons.gov.uk/aboutus/transparencyandgovernance/freedomofinformationfoi/mobilephonetheftintheukperannum
2. Lukas Grigas, (2021, July 7). NordPass is more than a cybersecurity tool. NordPass. https://nordpass.com/blog/cybersecurity-peace-of-mind/
3. Renaud, K., Coles-Kemp, L. Accessible and Inclusive Cyber Security: A Nuanced and Complex Challenge. SN COMPUT. SCI. 3, 346 (2022). https://doi.org/10.1007/s42979-022-01239-1
4. Mukta Kulkarni, Digital accessibility: Challenges and opportunities, IIMB Management Review, Volume 31, Issue 1, 2019, Pages 91-98, ISSN 0970-3896, https://doi.org/10.1016/j.iimb.2018.05.009
5. Calculator - Photo Vault by FishingNet: https://play.google.com/store/apps/details?id=com.hld.anzenbokusucal&pcampaignid=web_share
6. Vaulty by Squid Tooth LLC: https://play.google.com/store/apps/details?id=com.theronrogers.vaultyfree&pcampaignid=web_share & https://vaultyapp.com
7. Web Content Accessibility Guidelines (WCAG) 2.0. W3C Recommendation (Quick Reference) (2008, December 11).: https://www.w3.org/TR/WCAG20/

(Used Code Blocks extension on Google Docs for code formatting)