Plastic Waste Awareness Mobile Application

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree Bachelor of Science, School of Engineering

Elias Haddad Spring, 2021

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature _____Elias Haddad_____ Date __05/12/2021_____

Approved _____Daniel Graham_____ Date __05/12/2021___
,Department of Choose department

Earth Ambassadeurs Mobile Application

ELIAS HADDAD, University of Virginia, Virginia
DANIEL G. GRAHAM, University of Virginia, Virginia
ANDREA CLAYTON, Caribbean Maritime University, Jamaica

The University of Virginia has teamed up with Caribbean Maritime University to address plastic pollution in a more fun and interactive manner. Daniel Graham PhD, Andrea Clayton PhD, and Elias Haddad, worked together to create an educational cross platform mobile application that educates the general population about plastic waste. The mission of the application is to not only provide information but also engage the user to integrate plastic cautious activities into their daily lives.

CCS Concepts: • **Mobile application development → Cross platform Applications**; *React Native.*

Additional Key Words and Phrases: Databases, NoSQL, navigation, rendering, JSON

ACM Reference Format:
Elias Haddad, Daniel G. Graham, and Andrea Clayton. 2021. Earth Ambassadeurs Mobile Application. 1, 1 (May 2021), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This plastic education mobile application is a cross platform application, meaning it can be used on IOS and Android mobile devices. This is possible by using React Native as our development framework. React Native is a JavaScript based framework, utilizing NodeJS. This proved to be more useful and convenient to use in contrast to native frameworks or languages such as swift for IOS, and Java for Android, for multiple reasons. First and the most compelling is that with our time frame of a single semester, the convenience of developing a single code base and ending with two different platform capabilities is the best way to maximize potential users. Since this application would be required for primary level students, it must be available for both IOS and Android to provide all students with the opportunity to use the application. The second reason is that having one code base is easier to manage and maintain; for every feature added in IOS, it must be added in Android, giving way to more possible software faults and failures.

Along with React Native, we use Google's Firebase database framework to store and retrieve data. This NoSQL database is the best option to use along React Native because of ease of use and the large amount of documentation and community support regarding the two technologies in conjunction.

Authors' addresses: Elias Haddad, University of Virginia, Charlottesville, Virginia; Daniel G. Graham, University of Virginia, Charlottesville, Virginia; Andrea Clayton, Caribbean Maritime University, Kingston, Jamaica.

XXXX-XXXX/2021/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Originally, the application was designed for primary level students, ages 7-10 years old; however, through out the development process, we began to see that the activities and content of the application would also be useful to the general public so even though the activities didn't change, the intent and language of the content changed to apply to the general public.

Through out this paper I will describe the functionality of each component used in this application and how they work together. I break up the description of the application into two halves, the first half being the applications functionality before a user has been authenticated and in contrast, the second half is the applications functionality after a user has been authenticated.

2 MOTIVATION

Plastic use has see a surge over the past 50 years or so as they are convenient to use and affordable. However, these luxuries come with a large down side, mainly that plastic can not decompose, meaning it will stay in its form over its lifetime. With that said, even though it can be difficult to determine the amount of plastic enters the ocean, researchers in "Plastic waste inputs from land into the ocean" estimated that about 8 million metric tons of plastic waste entered the ocean in 2010[Jambeck et al. 2015]. These plastics disrupt the nature ecosystem and endanger wildlife.

Jamaica relies heavy on it's sea and fishery markets so wildlife being affected by this waste can also affect the economy and markets of Jamaica. Plastic on the seashores are also particularly dangerous since the plastics begin to break apart and become micro-plastics, which are too small to clean up but large enough for wildlife to eat. Therefore, plastic waste awareness and clean plastic habits was needed to be introduced to the community of Jamaica, to both future and current generations. We, Daniel Graham PhD, Andrea Clayton PhD, and Elias Haddad, worked together to develop an educational application to address these very needs.

3 RELATED WORKS

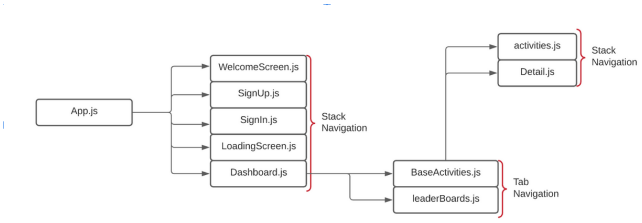
Considering the severity of the issues plastic waste brings upon us, plastic educational applications are not an uncommon product. The Plastic Soup Foundation created "My Little Plastic Footprint", which is available on the Google Playstore and Apple's App Store. This application helps users reduce their plastic footprint by allowing them monitor their plastic use and then use alternative.

Similarly, PlasticScore, Inc. has created an mobile application called "PlasticScore Zero Waste Dining" which is available on the Google Playstore and Apple's App Store. This application is a restaurant plastic waste rating application. Users dine out and answer questions provided by the application afterwards about the types and amounts of plastic used; then the application generates a plastic rating for that restaurant. This is useful because it allows the user to

be part of the experience, making the application more interactive and fun.

4 APPLICATION ARCHITECTURE OVERVIEW

The application will begin by rendering App.js which creates a stack navigation, allowing us to navigation between the files declared in that stack navigation. These files or components of the stack are WelcomeScreen.js, LoadingScreen.js, SignIn.js, SignUp.js and Dashboard.js. Once the user has been authorized, they will be redirected to Dashboard.js, which creates a tab navigation for the BaseActivities.js and leaderBoards.js components. The BaseActivities.js will create a stack navigation with Detail.js and activities.js components. This layout allows us to give the user a seamless experience through navigating between screens. The Firebase databasse gets initialized in App.js so this allows us to preform CRUD (create, retrieve, update and delete) commands through out all the files by simply referencing Firebase.



5 FIREBASE DATABASE

Firebase is a database platform that provides many services such as storage, hosting, functions and more. We will discuss Firebase's Authentication and Firestore services in this section.

5.1 Authentication

Firebase Authentication is a useful alternative to building your own authentication system since Google has developed and tested it, making it simple to incorporate into your application. It is also useful because their are different methods of authentication (i.e. Facebook login, Google login, etc.) that can be used by a switch of a button and using their sample user interface. When a user logs in, Firebase Authentication sends information about the user to the device they are using; this is referred to as a call-back[Firebase 2021a]. Part of this information is a unique user ID (UID), this UID informs the application the access capabilities of the user; in our case, everything in the second half of the application, which is describe in section 6. Firebase Authentication also manages the users session so the user will maintain logged in even after they restart the application for example[Firebase 2021a].

5.2 Firestore

Firebase's Cloud Firestore is a NoSQL database that allows us to create a collection of data which takes on a document format. This document format resembles a JSON format (discussed in section 7). As the Firestore documentation mentions, it is a horizontally scaling database; which simply means that documents have the ability to be collections of other documents therefore we will have a nested collection[Firebase 2021b]. This can continue and take on a horizontal scaling schema.

Cloud Firestore's flexibility allows the developer to structure their data in multiple ways to best serve their application. The first structure is a (1) nested document, which means to organize lots of data into an array or similar data structure, and then store that array into a field of a document[Firebase 2021b]. This can save a lot of reading from the database but you might end up with more data than you need, slowing down your application. The second option is (2) sub collections; which allows the developer to create a collection within another, allowing us continue to grow our sub collections while the root document does not grow[Firebase 2021b]. However, it can be difficult to delete sub collections. The third option is the one we chose for this application; which is called (3) root-level collections. This structure creates a collection for every data set; so every set of documents will be similar to one another and all will be organized to their collection[Firebase 2021b]. This allows use to read documents very easily since all we have to specify is the root collection. Also, since part of the purpose of the application is to have instructors review the answers to the activities the students complete, having a root-level collection structure will make it simple for non-technical individuals to look through the Firestore console.

5.3 firebaseMethods.js

Before we discuss the first half of the application, it is important to understand the Firebase methods that are used throughout the application. This file holds all methods that interact with the Firebase database. Specifically it means we are either adding data to a collection or retrieving data from a collection. I created a method that adds data to a collection for each one of the activities. Note: Firebase is a NoSQL database that holds a set of data inside something called a collection. The data is represented in a document (you can think of a document as a row in a table and a collection as the table itself). In the example below we are adding data to the users collection. This method takes in arguments that are values for firstName,lastName, and institute and then creates a document in the collection with these values in them. This is done by using the db.collection(collectionName).doc(docName).set(Values); where db is the database instance.

6 FIRST HALF

The user, before getting authenticated, which means they are logged out or have not created an account yet, will be redirected to the welcome screen, rendered by WelcomeScreen.js. This unauthenticated state will be referred to as the first half of the application. The application will run App.js first then loadingScreen.js since that is the default component in the stack navigation created in App.js. The loadingScreen.js file will determine if the user is logged in using Firebase Authentication. Since we are assuming the user is not authorized, it will redirect the user to the welcome screen, and that will allow the user to sign in or sign up, rendered by the SignIn.js file and SignUp.js file respectively. Once authorized, the user will be redirected to the Dashboard.js file render, which will approach the second half of the application described in section 9.

6.1 App.js

This file is the main file of the application, meaning when the application starts up and runs; it will start running the App.js file and then running everything else that is specified in the App.js file. In our App.js file we create a stack navigation that provides our application a manner of transition between screens. In our case the screens consist of the welcome, sign up, sign in, loading, and the dashboard screen. Since App.js is the first file our application will run, it is a good place to specify the database we are connecting to, which in our case is a firebase database. This is done by creating a json object specifying the firebase credentials and then using the firebase.initializeApp(..) function to connect to the database.

```
<NavigationContainer>
  <Stack.Navigator>
  <Stack.Screen name={'Loading'} component={LoadingScreen} options={{ headerShown: false }}/>
  <Stack.Screen name='Home' component={WelcomeScreen} options={{ headerShown: false }}/>
  <Stack.Screen name='Sign Up' component={SignUp} options={{ headerShown: false }}/>
  <Stack.Screen name='Sign In' component={SignIn} options={{ headerShown: false }}/>
  <Stack.Screen name={'Dashboard'} component={Dashboard} options={{ headerShown: false }} />
  </Stack.Navigator>
</NavigationContainer>
```

6.2 Loading Screen

The first component of the stack specified in App.js is the loading screen, this means we run the LoadingScreen.js file which does a simple check to determine whether or not this user exists in the database and if they do, then load the Dashboard screen and if not, then load the Welcome screen (which is referred to as 'Home' in our stack). This can be seen here:

```
firebase.auth().onAuthStateChanged((user) => {
  if (user) {
    navigation.replace('Dashboard');
  } else {
    navigation.replace('Home');
  }
});
```

6.3 Welcome Screen

If the user is not logged in or does not have an account yet, the application will render the WelcomeScreen.js file, this file gives the user an option for signing in or signing up. Depending on which button is pressed the application will either navigate to the SignIn.js or SignUp.js file. Again this navigation is using the stack navigation defined in App.js. This file also displays a background of a woman holding a globe; if this is desired to be changed, it is recommended to create an image large enough to fill the screen and use that as the background. Instead of inserting an image directly because then all box components(CSS components) will be adjusted and will require more effort to fix. This can be seen here:

```
style={styles.background}
source={require('../assets/landingPage.png')}>
<View style={styles.titleContainer}>
  <Text style={styles.title}>myEarth</Text>
</View>
<View style={styles.buttonContainer}>
  <TouchableOpacity style={styles.button} onPress={() => navigation.navigate('Sign Up')} >
    <Text style={styles.buttonText}>Sign Up</Text>
  </TouchableOpacity>
  <Text style={styles.inlineText}>Already have an account?</Text>
  <TouchableOpacity style={styles.button} onPress={() => navigation.navigate('Sign In')}>
    <Text style={styles.buttonText}>Sign In</Text>
  </TouchableOpacity>
</View>
```

6.4 Sign In Screen

When a user signs in, they enter their email address and password. On the backend, we store these values into state variables that we pass into the signIn(...) method in firebaseMethods.js that uses a firebase function to sign in the user given these parameters. The function call and function itself are shown below:

In SignIn.js: `signIn(email, password);`

In firebaseMethods.js:

```
export async function signIn(email, password) {
  try {
   await firebase
      .auth()
      .signInWithEmailAndPassword(email, password);
  } catch (err) {
    Alert.alert("signin error", err.message);
  }
}
```

6.5 Sign Up Screen

When a new user signs up for this application, they will be prompted to enter their name, email, institute and password to create the account. On the backend, these values will be stored in state variables that are passed into the registration(...) method in firebaseMethods.js to create the user and add them to the users collection.

6.6 Dashboard Screen

Once the user has signed in, they will be redirected to the dashboard screen of the app, via stack navigation. Once there, we have created a tab navigation that creates two tabs, one for activities and another for a scoreboard. A tab navigation is used to navigate screens with the use of tab touches. The activities tab calls and renders the BaseActivities.js file and the scoreboard tab once pressed calls and renders the leaderBoards.js file.

7 JSON DATA

JSON stands for JavaScript Object Notation and is a data formatting notation used is in JavaScript. The reason it is used in react native is because it is used to send and receive data from client and servers. I use it through out the application because it is easy to access in JavaScript. For example, if we had a JSON object representing a dog, it would look like this, dog:{breed: 'Labrador', age: 2}, and this key-value pair makes it easy to access the breed value. We

can simply print the breed value by doing console.log(dob.breed); and this will print 'Labrador'. The ability to contain multiple key value pairs in one JSON object makes it a lot more convenient to use through out the application so redundant parameter calls are avoided[Chris Mills 2021].

7.1 data.js

This file is a JSON object of other JSON objects that hold information about the activities, like title, info, and headers. As we can see below, level 1.2 is represented by this JSON object so when we pass in a data parameter from activities.js to Detail.js, we can render this information in a nice format assuming the type passed in is of type: '1.2' (more on this in the activities.js and Detail.js sections below).

```
// Level 1.2
level_one_two_obj :
{
        title: 'Your Plastic Footprint',
        header_one: 'How do you dispose?',
        header_two: 'Where do plastics end up?',
        header_three: 'How have you used plastic in the past month?',
        type: '1.2',

},
```

8 PARAMETERS

A parameter is anything that gets passed into a function and in React Native, a parameter can be passed into a render option such as onPress{parameter} for example. Parameters can also be passed into navigation routes between screens[Facebook 2021]. For example, if I create a stack navigation with component A and component B in that stack, I can create an array in the class representing component A and navigate from component A to component B, while passing that array I created from component A to component B as a parameter.

Throughout the application I use parameters with respect to navigation routes so I can gain information in a component of the navigation, where it is easier to get that information, and then send it to another component in the navigation where that information gained can be used to be rendered to the screen.

9 SECOND HALF

Once the user has been authenticated, they will enter the second half of the application. The first thing that happens is the user will be redirected to the DashBoard.js board, which creates a tab navigation for the BaseActivities.js file and leaderBoards.js file.

9.1 Base Activities File

Once the user pressed the activities tab, it will render the BaseActivities.js file. This file creates a stack navigation that allows us to navigate to activities.js (which renders the list of activities the users see). The second component of the stack is the Detail.js file. This file renders the individual activities that a user clicks on (i.e. pre-questionnaire).

9.2 Activities Screen

This file renders the list of activities available to the user. The file first imports JSON data from data.js which is then passed to the Detail.js along with a type of activity so the detail page knows what specific activity to render. The activities.js file renders all activities, so if one were to add activities, they would need to be added to this file's render. We can see an example below:

```
<TouchableOpacity
  style={styles.itemContainer}
  onPress={() => navigation.navigate('Detail',{data, type:'3'})}>
  <Text style={styles.buttonText}>Recycling Art (Trash to Treasure) </Text>
</TouchableOpacity>
```

As we can see above, this specific piece of code is rendering the level three activity "Recycling Art (Trash to Treasure)". We create a TouchableOpacity, allowing us to do something on the press of the user. In this case when the user presses the activity three button, it will navigate to the Detail.js file, by specifying "Detail" since that is the name we associated with the Detail.js file in the stack created in BaseActivities.js. Once pressed we will navigate and pass data and type: '3' along to the Detail.js file. The data is all of the titles, information and questions we have on each activity, and the type is 3 for level three, so we know of all the data from data, which part do we want to render and in this case, Detail.js will know that we want to render activity three since we passed in type: '3' .

9.3 Detail File

Once a user has pressed a specific activity from activities.js, we will render that specific activities information. The way it does this is by using condition rendering, which checks for something (type in our case) and renders depending on the result of that check. Since the majority of activities require students to enter answers to questions, there are state variables that hold these answers and then get passed into a firebaseMethods.js method for that activity. An Example can be seen below: Let's assume the user clicked on activity 1.2 ,"Your plastic footprint". The data and type of 1.2 will be passed to the Detail.js file, however to know if the user clicked on activity 1.2, we check if the type passed in as a parameter is equal to '1.2', if so we will render the level_one_two_obj data, which again is the json object holding the title, information and question headers for activity 1.2 from the data parameter, as seen in the data.js section above. Activity 1.2 has three questions so we allow the user to input answers to these questions and the value entered will be held in the answerOne, answerTwo and answerThree state variables. Once the user clicks submit, it will call the handlePress function that checks for the type again and calls the appropriate database function for that activity. In our case it's the level_one_two_db(answerOne, answerTwo, answerThree); function from firebaseMethods.js. Note: the render variable is used later on in the file where we actually render the components render holds.

```
}else if(type=='1.2'){
  render=[
  <Text style={styles.title}>{level_one_two_obj.title}</Text>,
  <Text style={styles.info}>{level_one_two_obj.info}</Text>,
  <Text style={styles.label}>{level_one_two_obj.header_one}</Text>,
  <TextInput
      style={styles.input}
      placeholder="answer one*"
      value={answerOne}
      onChangeText={(answer) => setAnswerOne(answer)}
      />,

  <Text style={styles.label}>{level_one_two_obj.header_two}</Text>,
    <TextInput
        style={styles.input}
        placeholder="answer two*"
        value={answerTwo}
        onChangeText={(answer) => setAnswerTwo(answer)}
    />,
  <Text style={styles.label}>{level_one_two_obj.header_three}</Text>,
      <TextInput
          multiline={true}
          numberOfLines={10}
          style={styles.textarea}
          placeholder="answer three*"
          value={answerThree}
          onChangeText={(answer) => setAnswerThree(answer)}
      />,
  ];
```

```
const handlePress = () => {

  if(type=='1.2'){
    level_one_two_db(answerOne, answerTwo, answerThree);
  }
```

9.4 Score Board Screen

This file simply takes in the array from userData.js and renders it. However, since it is an array, we cannot render it and must convert it to a list of strings instead and then we display this list in a ScrollView. This file also gives users the opportunity to log out which calls the loggingOut(); method from firebaseMethods.js.

```
const listItems = userData.map((d) => <Text style={styles.itemContainer} key={d.firstName}>{d.firstName}</Text>);
return (
  <View style={styles.container}>
      <ScrollView style={{marginTop:85, width:"100%"}}>
        <Text style={styles.titleText}>{user_institute} Leader Boards</Text>
        {listItems}
      </ScrollView>
      <TouchableOpacity style={styles.button} onPress={handlePress}>
        <Text style={styles.buttonText}>Log Out</Text>
      </TouchableOpacity>
  </View>
);
```

9.5 Logging out

The log out functionality is present in the leader board screen, at the very bottom, in a form of a button. Once pressed, the onpress attribute of React Native allows us to call the loggingout() function from firebaseMethods.js, which uses Firebase Authentication's signOut() method to end the users current session. This can be seen below:

```
export async function loggingOut() {
  try {
    await firebase.auth().signOut();
  } catch (err) {
    Alert.alert('loggingOut error', err.message);
  }
}
```

10 THE FUTURE OF THE APPLICATION

Even through the application has come a long way, it also has a bright future. Future development of application consists of first, adding an image capability feature, which would prove useful since the activities will be more dynamic if that feature is present. Secondly, I believe the application would achieve it's goal of integrating plastic safe habits if the application was more interactive. Through out the development process, Daniel Graham PhD, proposed a feature of Geo-Location with respect to plastic waste in the area. The user will be able to take a photo of the plastic waste found, and the GPS coordinates of the users location will be marked on a map with an image associated with that mark. Other users will be able to see this mark and clean up the location, in which, a new and clean picture can be uploaded to those coordinates; showing a before and after photo. This allows the users to have more fun while using the application and building plastic safe habits by actually becoming part of the movement through physical activities. Other improvements to the application can always be made, however demo's of the application along the way will allow future development to be more effective since feedback from the users can be considered.

11 CONCLUSION

Daniel Graham PhD, Andrea Clayton PhD, and Elias Haddad, developed an application that addressed the increasing problematic issue of plastic waste. Developing the application in React Native paired with a Firebase database proved to be optimal considering the time to production efficiency it provides. This application utilizes stack and tab navigation to transition between screens and also pass in parameters of data to those screens during the transition. The Firebase database allows us to authorize users and store data provided through completed activities. This application has a bright future of educating the future generations to be more plastic aware and safe.

REFERENCES

Nick Schonning Peter Bengtsson Chris Mills, Florian Scholz. 2021. Working with JSON. *Science* (2021). https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

Facebook. 2021. Props. *Science* (2021). https://reactnative.dev/docs/props

Firebase. 2021a. Firebase Authentication Documentation. *Science* (2021). https://firebase.google.com/docs/auth

Firebase. 2021b. Firebase Cloud Firestore Documentation. *Science* (2021). https://firebase.google.com/docs/firestore

Jenna R. Jambeck, Roland Geyer, Chris Wilcox, Theodore R. Siegler, Miriam Perryman, Anthony Andrady, Ramani Narayan, and Kara Lavender Law. 2015. Plastic waste inputs from land into the ocean. *Science* 347, 6223 (2015), 768–771. https://doi.org/10.1126/science.1260352 arXiv:https://science.sciencemag.org/content/347/6223/768.full.pdf