# **Implementation (2,500 words)**

With the previous chapter discussing how the app should be designed, this chapter focuses on how the author implemented this meaningful design.

**Tools Used**

First and foremost, React Native was the development language of choice for the app. Additional tools such as XCode and Android Studio, the Integrated Development Environments (IDEs) for iOS and Android respectively, were needed to create emulators to test the app on these platforms. Additionally, GitHub was the tool used for version control to ensure any progress was not lost and that had the author ran into any unforeseen circumstances where there was some issue with the app, he could simply continue from a previously working version of the code.

**React Native**

React Native was the development language of choice. Chose this over Swift and Java/Kotlin because React Native is cross-platform whereas Swift and Java/Kotlin are native, only working on either iOS or Android respectively. Flutter is an alternative cross-platform development language which was considered, but ultimately the author favoured implementing through React Native due to extensive online documentation and transferable skills whereby React Native is very similar to React, which is a development language used for building websites.

**Firebase Cloud Firestore**

Firebase Cloud Firestore was chosen to facilitate database functionality, saving user progress, scores and data as they used the app. Firestore Realtime Database was also considered for implementation, however, having analysed the specific use case for this application and the type of and frequency of the kind of database calls which would be used for this application, the more flexible noSQL Firebase Cloud Firestore schema was a far greater match than the strict schema of the SQL Firestore Realtime Database. Further discussion and illustration to this point will be justified through the emission logs screenshots in section 1.2 of this chapter on the database.

**XCode**

To-do

**Android Studio**

To-do

**GitHub**

To-do

Attach screenshot of GitHub

**VSCode**

VSCode was used as the integrated development environment (IDE) of choice by the author due to industry popularity with this IDE, a vast collection of available extensions such as prettier used for code formatting, and because of the author’s familiarity and experience with this IDE.

**Architecture**

* React Native – Front-end
* Firebase Cloud Firestore – back-end
* Figma – prototyping
* VSCode – Integrated Development Environment (IDE)

Add a diagram here and explain it in above paragraphs

## **User Interface (Front End)**

Insert screenshots below for each screen on a newer iPhone emulator.

**Login**

Graphical user interface, text, application, chat or text message

Description automatically generated

**Homepage**

2 screenshots here

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Similar items are grouped together on the homepage, where food logs appear together and transport logs appear together, making it easier for users to find and explore logs, reducing complexity for users.

**Log Emission Button**

A picture containing text, first-aid kit, clipart

Description automatically generated

The log emission button is easily accessible from anywhere on the home screen.

Graphical user interface

Description automatically generated**Log Food Emission**

Clicking save will lead to the confirmation alert illustrated below.

**Log Transport Emission**

Graphical user interface, text

Description automatically generated

Clicking save will lead to the confirmation alert illustrated below.

**View Individual Leaderboard**

Graphical user interface, text, application

Description automatically generated

The logged in user’s score is highlighted in green to speed up and reduce the complexity of locating their position in the leaderboard.

Graphical user interface, text, application

Description automatically generated**View Team Leaderboard**

Again, the logged in user’s team’s score is highlighted in green to speed up and reduce the complexity of locating their position in the leaderboard. If a user is constantly winning or losing in the individual leaderboard, their teammate’s (only one teammate) score may alter the balance, providing a more competitive landscape to compete in.

**View Individual History of Scores**

A picture containing graphical user interface

Description automatically generated

This screen sustains motivation, whereby users who are either winning or losing all of the time, can find competition against themselves, targeting the competence aspect of self-determination theory where they can strive to master their performance.

**Navigation Bar to Speed Up Frequent Tasks**

Graphical user interface, text, application, chat or text message

Description automatically generated

The navigation bar means these frequents tasks are only 1 click away for users, increasing speed and ease of use.

Graphical user interface, text, application, chat or text message

Description automatically generated**Confirmation Button to Reduce Likelihood of Errors**

The field of human computer interaction emphasises the importance of reducing the likelihood of errors as opposed to dealing with realised errors.

Graphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, application

Description automatically generated**Reducing Complexity Through Variety of Units of Measurement**

Graphical user interface, text, application, chat or text message

Description automatically generated

Users who are used to different units of measurement or who do not know the size of their car are catered for, thus reducing complexity and cognitive load placed on users.

## **Database (Back End)**

**Storing authenticated user accounts on Firebase**

Table

Description automatically generated

Design of database was to follow the rules of avoiding redundancy, and following the noSQL golden rule of make collections large and documents small, whereby instead of storing, for example, a user’s emission logs as an array inside of a user document, the emission logs were stored as a separate collection with a reference link to the corresponding user. This is because the number of emission logs per user is infinite, so with each new log, the user document would have kept expanding, eventually exceeding the 2MB capacity limit imposed by Firebase. Instead, by storing emission logs as a separate collection, this error was avoided.

**Collections**

Graphical user interface, application

Description automatically generated

**User document**

Note, the user documents do not contain a “uniqueID” field because this value is stored as the unique ID for the document itself, which is connected to the unique user ID assigned to each authenticated user as per the screenshot above. For example, in the attached screenshot below, Jack’s uniqueID is 80ng7PdDcIeGOd7a58YQj6tVKnX2.

Graphical user interface, text, application, email

Description automatically generated

**Team document**

Graphical user interface, text, application, email

Description automatically generated

The decision was made by the author to include the associated userIDs for the team members as an array inside of the team document because the team size fixed to a size of 2. As such, the size of this array, and therefore the size of each team document, will not grow infinitely, exceeding the 2MB capacity limit for Firebase documents.

**Emission Logs**

The 2 types of emission logs for the gamified social mobile app are food and transport logs. The example food and transport document screenshots attached below highlight the benefit of using the noSQL database schema of Cloud Firestore where documents in the same collection can have different fields, versus using the relational database management system style of Firebase Realtime Database where all documents in a collection must follow a strict criteria of having the same fields. As screenshots x and y below illustrate, a food emission log document has fields such as “portionSize” and “portionUnit”, whereas a transport emission log document has fields such as “distanceTravelled” and “unitOfDistance”. Had a relational database such as Firebase Realtime Database been used, separate collections for food and transport logs would need to be created, adding unnecessary complexity to the database design.

**Food emission log document**

Graphical user interface, text, application, email

Description automatically generated

**Transport emission log document**

Graphical user interface, text, application, email

Description automatically generated

**Score document**

The score documents could arguably be considered redundant, since they can be calculated from the emission logs of each user. However, the author decided to store them in the database for speed performance of the application. Once the scores have been generated, by storing them in the database, these same scores never have to be regenerated. This is particularly important for performance for users viewing the history of their scores, where every score would need to be recalculated, and viewing the individual and team leaderboards, where every single users score would need to be regenerated. Given the use case and expected database calls to be made while using the application, this solution proved optimal.

Graphical user interface, text, application, email

Description automatically generated

## **Code Examples**

## **Implementation Issues (May have these…)**

Originally, the author had intended to use Firebase’s Cloud functions to automatically generate user scores at the end of each day for the previous day without having to run the app. Unfortunately, Firebase’s Cloud functions have changed and are no longer provided for free. This unexpected challenge required the author to be creative, harnessing his problem solving skills acquired throughout his time in college. The solution the author came up with was once the user logs in, check the database to see if a score exists for the previous day and if not generate the scores for each user. All scores will always be generated together, at the same time, thus if a score document does exist then all scores for the previous day have already been created and the code can exit this function without re-fetching the scores.

The novelty of learning React Native was challenging for the author, however this challenge was to be expected and as such the author had accommodated for this, allowing slippage in his project plan.

At times, depending on where the author was working from, the expo simulator would not load or take an absurdly long time to load. Such occurrences happened whilst on campus using the college WIFI or commuting to campus on Dublin Bus using the Dublin Bus WIFI. The author overcame these obstacles by turning on his personal hotspot and connecting to this connection, and planning his workload accordingly. For example, when the author was on campus or on Dublin Bus where the internet may have been inadequate for expo to run, he would utilise this time to focus on the offline workload such as the documentation for this report.