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

Having outlined in the previous chapter how the app should be designed, this chapter focuses on how the author implemented this design, discussing the tools used, the architecture of the app, the front end and the back end of the application.

**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**

Graphical user interface, text, application, email

Description automatically generatedGitHub was used for version control. The decision was made to GitHub to adhere to software engineering best practices. Alternative version control systems such as BitBucket and TortoiseSVN were considered, however, due to existing familiarity, GitHub was ultimately decided. A screenshot of the author’s private GitHub repository is provided below.

**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

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

Having established important design considerations in chapter 3 of this report, the below screenshots illustrate the final user interfaces for the application with rationale as to why they were designed this way.

**Login**

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

Description automatically generated

The login page is very basic, providing only the information needed for users to sign in, preventing unnecessary clutter, contributing to a clean and tidy display. Although the required inputs are arguably considered universally understood in this day and age, placeholder inputs are provided to aid the user in the event of any confusion.

<Add link to reset password? Serious error is if user forgets password…>

**Homepage**

2 screenshots here

Graphical user interface, application

Description automatically generated

As discussed in the design chapter of this report, the requirements analysis highlighted the importance for users to have the ability to review emission logs, to limit the likelihood of serious errors such as forgetting to log an emission or having duplicate logs. Additionally, insights from the literature review emphasised how the best rated carbon footprint calculators provide a breakdown for users of their emissions, offering education and removing ambiguity.

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

Description automatically generated

Further insight from the requirements analysis of the design chapter revealed how grouping similar items together can reduce the complexity for users in their quest to check for duplicate or missing logs.

A picture containing text, first-aid kit, clipart

Description automatically generated**Log Emission Button**

The log emission button has been implemented to remain visible on screen as the user moves around the homepage. The design chapter highlighted the importance of speeding up frequent tasks, and being able to click the log emission button from anywhere on the home screen achieves this need to speed up this frequent task of logging emissions.

Logo, company name

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated**Log Food Emission**

Clicking save will lead to the confirmation alert illustrated below.

**Log Transport Emission**

Text

Description automatically generatedGraphical 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

Simple, clear, tidy headings of “Yesterday’s Score” and “Individual Leaderboard” are displayed on screen to remove any ambiguity for the user. If they get distracted from their phone and then come back to this screen, it is clear which leaderboard they are looking at.

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

A picture containing chart

Description automatically generated

Graphical user interface, text, application

Description automatically generated**View Team Leaderboard**

Just like the individual leaderboard, the team leaderboard contains clear headings and highlights the logged in user’s team’s position in the leader, increasing speed and reducing complexity for the user when locating their position.

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, lacking motivation or a challenge against their peers, 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 the frequent tasks of viewing your emission logs, individual and team leaderboards and individual history of scores are only 1 click away for users, increasing speed and ease of use. Additionally, similarly to the leaderboard, the active tab in the navigation bar changes colour to green to reduce cognitive load on the user as to what screen they are on.

Graphical user interface, text, application

Description automatically generated

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

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

As illustrated in the design chapter, the field of human computer interaction emphasises the importance of reducing the likelihood of errors as opposed to dealing with realised errors. Analysing the hierarchical task analysis of a user logging their emissions in the design chapter revealed the importance of limiting the likelihood of entering incorrect emission log data, leading to the implementation of this confirmation button.

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

As discussed when analysing the scenario and hierarchical task analysis of a user entering emission logs in the design chapter, complex tasks involved converting one unit of measurement into another and selecting car size. As such, to reduce these complexities, a selection of different units of measurement are provided, catering for users who are used to different units of measurement, and the average car size is provided, catered for users who do not know the size of their car, thus reducing complexity and cognitive load placed on users.

## **Database (Back End)**

The database used for implementing this gamified, social mobile app was Firebase Cloud Firestore, a noSQL schema which consists of collections and documents. As discussed in section xxx, this database was chosen for the increased flexibility offered by the schema.

The design of the implemented database was motivated by industry standard database best practices, of avoiding redundancy, and following the noSQL golden rule of making collections large and documents small. To follow this noSQL rule, a user’s emission logs, for example, should be stored as a separate collection with a reference link to the corresponding user, as opposed to as an array inside of a user document. This is because the number of emission logs per user is infinite, with each new log expanding the user document, eventually exceeding the 2MB capacity limit imposed by Firebase. Instead, by storing emission logs as a separate collection, this error was avoided.

The content below contains screenshots and explanations to the different aspects of how the database was implemented.

**Storing authenticated user accounts on Firebase**

Table

Description automatically generated

Each authenticated user has an email and password login associated with their account to login, as well as having a unique user id or “User UID” to identify and refer to the associated user in collections and documents.

Text

Description automatically generated

If the entered credentials of an email and password do not match any authenticated user accounts, the user is alerted of invalid login credentials and will not proceed to the homepage. Upon supplying valid credentials, users are navigated to the homepage, and the unique ID to identify this specific user is carried forward.

**Collections**

Graphical user interface, application

Description automatically generated

The collections implemented in the database for this application were for the emission logs, scores, teams and users. The scores collection could arguably be redundant, since these scores can be calculated from the emission log documents, however, the time taken to recalculate every single score proved too costly, justifying including scores as a separate collection. Further rationale is provided in the score document section xxx further on in this report. Contrastingly, the author considered having a leaderboard collection which contained the associated scores to be displayed on this leaderboard, however, this was decided to be redundant because the scores are already stored in the database, and the time taken to retrieve them for display in the leaderboard is negligible compared to the time of calculating every score from all emission logs.

**User document**

Graphical user interface, text, application, email

Description automatically generated

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.

**Team document**

Graphical user interface, text, application, email

Description automatically generated

The decision was made by the author to include the associated user IDs for the team members as an array inside of the team document because the team size is 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**

Text

Description automatically generatedThe 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

Graphical user interface, text, application

Description automatically generated

The addLogToDatabase function is triggered when the user confirms their log details after clicking the save button on either the log transport or log screen pages.

**Transport emission log document**

Graphical user interface, text, application, email

Description automatically generated

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

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## **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.