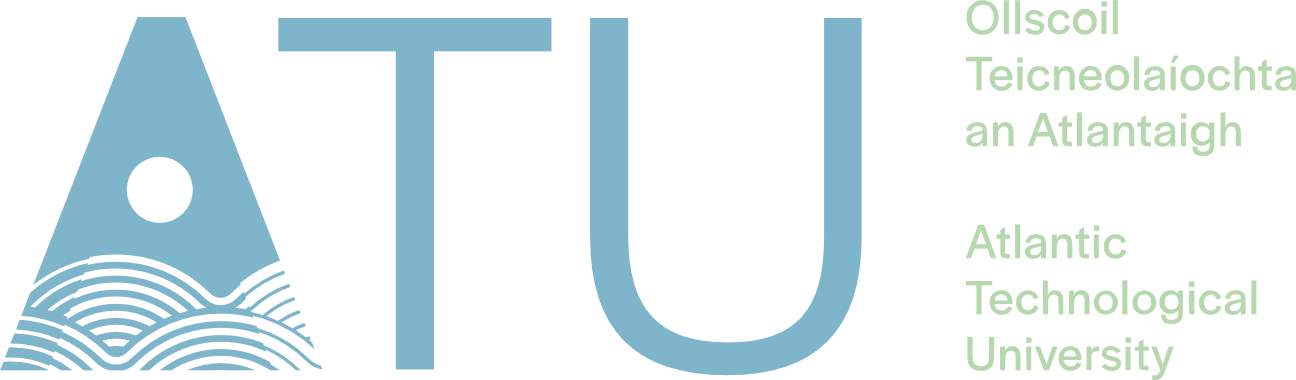
****

**“The Breathalyser Connection”**

**Project Engineering**

**(Year 4)**

**Lorcan Stakem**

**BEng (Hons) in Software & Electronic Engineering**

Atlantic Technological University

2022/2023

**Declaration**

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Atlantic Technological University.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

**Acknowledgements**

Firstly, I would like to thank my supervisor Paul Lennon for his invaluable guidance and support throughout the entire project year. His insights and feedback have been invaluable and have contributed to the quality of the project.

Together with Brian O'Shea, Michelle Lynch, and Niall O'Keeffe, I would like to thank them for their guidance and feedback on my project work throughout the course of the year.

I would also like to thank Atlantic Technological University for providing the necessary resources and facilities for conducting research and analysis which made this project possible.

Finally, I would like to express my gratitude to my family, friends and classmates for their unwavering support and encouragement throughout this academic year.

**Table of Contents**

[1 Summary 6](#_Toc133674318)

[2 Project Goals 7](#_Toc133674319)

[3 Poster 8](#_Toc133674320)

[4 Introduction 9](#_Toc133674321)

[4.1 Driving and Alcohol 9](#_Toc133674322)

[4.2 Measuring BAC (Breath Alcohol Concentration) 9](#_Toc133674323)

[4.3 State of the Law 10](#_Toc133674324)

[5 Tools and Technology 11](#_Toc133674325)

[5.1 React Native 11](#_Toc133674326)

[5.2 React Navigation 12](#_Toc133674327)

[5.3 Expo 12](#_Toc133674328)

[5.4 Node.js 12](#_Toc133674329)

[5.5 Arduino IDE 13](#_Toc133674330)

[5.6 Firebase 13](#_Toc133674331)

[5.7 Redux 13](#_Toc133674332)

[6 Hardware Overview 14](#_Toc133674333)

[6.1 MQ3 Sensor 14](#_Toc133674334)

[6.2 LCD Display 15](#_Toc133674335)

[6.3 ESP32 15](#_Toc133674336)

[6.4 LED’S 16](#_Toc133674337)

[6.5 Piezo Buzzer 16](#_Toc133674338)

[6.6 Push Button 17](#_Toc133674339)

[7 Project Architecture 18](#_Toc133674340)

[8 Project Logo 19](#_Toc133674341)

[9 Project Plan 20](#_Toc133674342)

[10 Project Code 22](#_Toc133674343)

[10.1 Software: 26](#_Toc133674344)

[11 Ethics 44](#_Toc133674345)

[12 Conclusion 45](#_Toc133674346)

[13 Code 46](#_Toc133674347)

[14 References 47](#_Toc133674348)

# Summary

The Breathalyser Connection project was designed to address the issue of drink driving by developing an alcohol breathalyser system. The primary objective of this project was to create a system that allows individuals to quickly and effortlessly take their breath samples to determine their blood alcohol content (BAC) and ensure that they do not exceed the legal limit for driving.

The project is a combination of software and hardware, and it employs an MQ3 sensor to detect alcohol in the user's breath and calculate their BAC based on the ratio of alcohol in the blood to alcohol in the breath. The BAC results are displayed in a numerical format, providing a clear indication of the individual's current BAC level.

In Ireland, breathalysers are a common tool used by law enforcement agencies to prevent drink driving. However, the Breathalyser Connection focuses more on personalised usage. The project aims to empower individuals to take control of their own drinking habits by providing them with a breathalyser device that connects to their smartphone. The device allows users to monitor their blood alcohol content (BAC) levels in real-time, providing them with a clear indication of when they are over the legal limit.

Alcohol consumption in Ireland is a significant issue and in fact, Ireland has one of the highest rates of alcohol-related road accidents in Europe. The project aims to reduce the number of drink-driving incidents on Irish roads by enabling people to monitor their own drinking.

The Breathalyser Connection represents an innovative approach to tackling a longstanding problem. By providing users with the tools they need to make informed decisions about their drinking, it has the potential to make a real difference in reducing the number of alcohol-related accidents on Irish roads.

# Project Goals

The goal of the project is to design and develop a reliable and accurate device for measuring the blood alcohol content (BAC) of individuals. The system should be user-friendly and easy to operate, providing accurate BAC readings within a matter of seconds. The project aims to address the issue of drink driving, which is a major cause of road accidents, injuries, and fatalities in Ireland.

Specifically, the project goals are:

* To design a cost-effective breathalyser device that can be used by individuals to measure BAC levels accurately and quickly.
* To develop a user-friendly interface that allows individuals to easily operate the breathalyser system and obtain accurate BAC readings.
* To conduct thorough testing of the breathalyser system to verify its accuracy and reliability under various environmental conditions, such as temperature and humidity.
* To explore the potential for integrating the breathalyser system with mobile applications to enhance its functionality and usefulness.

The successful completion of this project will result in a dependable, accurate, and easy-to-use breathalyser system that can aid in decreasing drink driving occurrences and enhance road safety.

# Poster

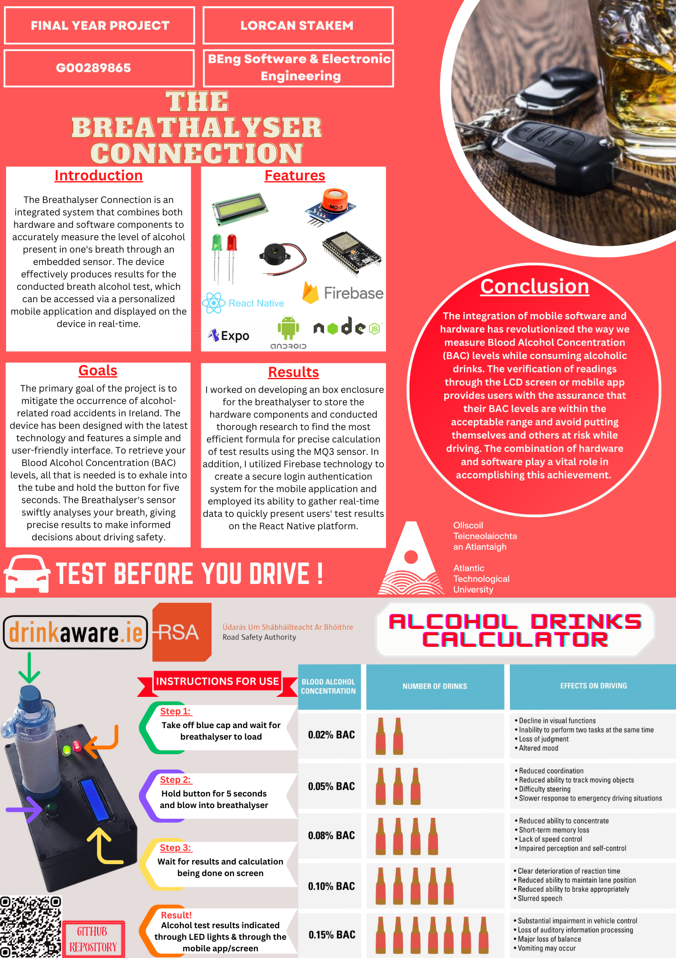


Figure 2.1 – Poster

# Introduction

## Driving and Alcohol

In many cultures, drinking alcohol has a long history and is frequently linked to celebrations, traditions, or noteworthy occasions. Many people choose to drink alcohol in pleasant company or by themselves. The stimulating and beneficial effects of alcohol use may outweigh the negative ones if it is consumed in moderation.

However, even small amounts of alcohol may impair perception and reaction times, making it impossible to safely operate a vehicle. Serious legal repercussions may occur if a person is detected by the Garda Síochána operating a vehicle while having a blood alcohol level that is above the legal limit.

To ensure accuracy and precision, the Garda Síochána must employ advanced measuring techniques and tools. Additionally, the public is intrigued by the idea of having access to devices that are reasonably priced, simple to use, reliable, and can provide an estimate of their blood alcohol concentration by measuring their breath alcohol concentration.

## Measuring BAC (Breath Alcohol Concentration)

Breath alcohol concentration, also known as BAC, refers to the percentage of alcohol present in a person's bloodstream that is used to determine the amount of alcohol in their breath. It is often used to determine the level of a person's impairment in their capacity to drive. Body weight, gender, alcohol tolerance, amount, and type of alcohol drank are some of the variables that affect blood alcohol content (BAC).

A breathalyser is used to calculate BAC; it operates by detecting the presence of alcohol in the user's breath. The breathalyser calculates a BAC reading by measuring the amount of alcohol in the air that is inhaled from the lungs. The legal limit for blood alcohol content varies by country, however it is 0.05% in Ireland.

Alcohol has a dose-dependent effect on the body, which means that the more alcohol someone drinks, the worse they will feel. A person may have minor impairment at lower blood alcohol concentrations, such as slower reaction times, poor judgment, and diminished self-control. A person may have increasingly severe impairment as their blood alcohol content (BAC) rises, such as slurred speech, loss of balance, and poor coordination. A person may experience alcohol poisoning at extremely high BAC levels, which can be fatal. Alcohol is metabolised by the body on average at a rate of 0.015% BAC every hour. Thus, if someone had a BAC of 0.05%, it would take around five and a half hours for it to drop to 0% after quitting drinking.

## State of the Law

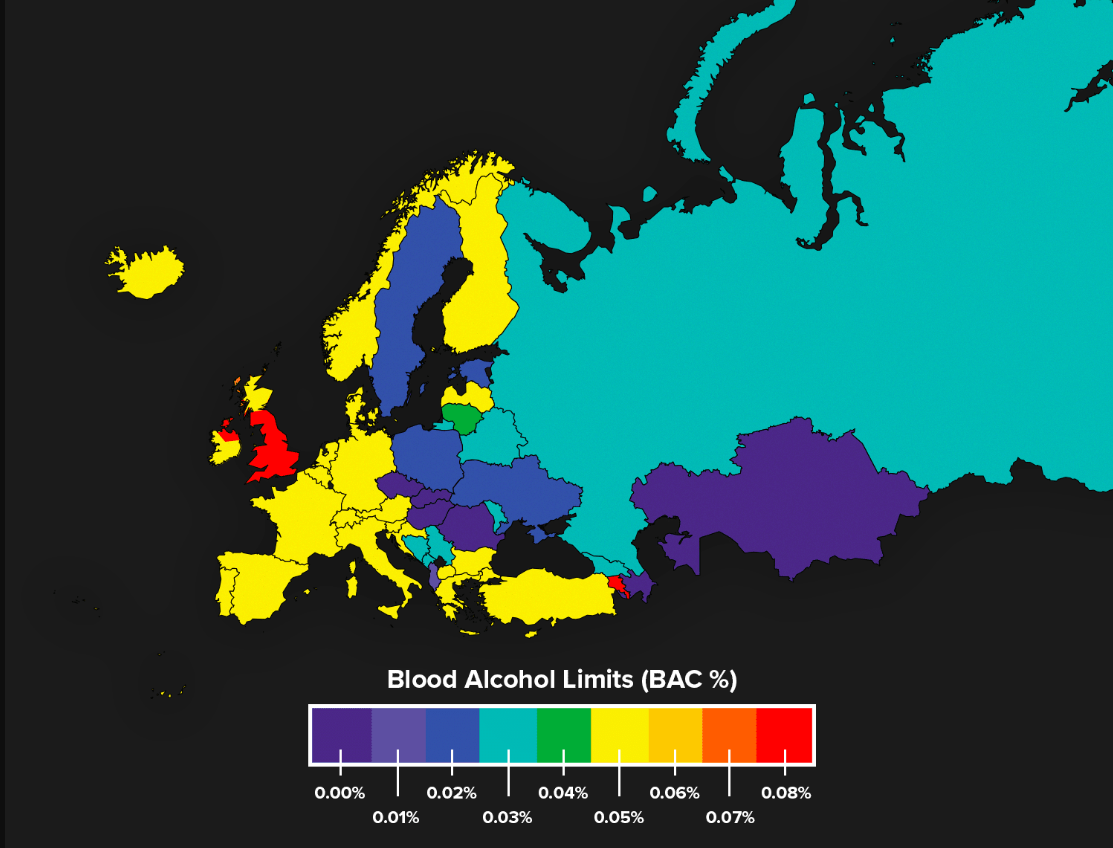


Figure 4.4 - Drink driving limits for European countries

# Tools and Technology

To ensure the successful completion of the project, I employed a variety of techniques and methods. One of the key strategies I employed was the integration of essential features into the mobile application software. This involved identifying the core functionalities that were required for the application to meet its intended purpose and incorporating them into the development process from the outset. By doing so, I was able to ensure that the app was designed with its end-users in mind, and that it provided a seamless and user-friendly experience.

In addition to integrating essential features, I also implemented a professional workflow that allowed me to work in an orderly and efficient manner throughout the project's lifecycle. This involved breaking down the project into smaller, more manageable tasks, and setting achievable goals and deadlines for each stage of the development process. By following a structured approach, I was able to stay on track and make steady progress towards the final deliverable.

To ensure that the project was executed with the most appropriate tools and technologies, I conducted extensive research into the available options. This involved reviewing various frameworks, libraries, and programming languages, and selecting the ones that were best suited to the project's requirements. By carefully evaluating the available options, I was able to identify the tools that would enable me to build a robust, scalable, and user-friendly application.

## React Native

React Native is a framework that enables the development of mobile applications using the React JavaScript library. It is an open-source framework, providing developers with the ability to build and design mobile applications with the use of the popular React library.

React Native allows to write mobile applications for both iOS and Android platforms, using a single codebase. It uses the same building blocks as React, such as components and props, making it easy for web developers to transition into mobile app development.

React Native includes a wide range of pre-built components and libraries, which can be easily integrated into applications, helping developers save time and effort. It also supports hot reloading, which allows developers to see changes in real-time without having to restart the application. Some of the most notable features of React Native include its ability to provide a native-like user experience, high performance, and support for third-party plugins and libraries.

## React Navigation

React Navigation is a navigation library for React Native that makes managing application navigation flow and switching between screens simple. It is built entirely in JavaScript, which means it can be used with any React Native project without requiring any native code changes.

React Navigation offers several navigation patterns such as Stack navigation which allows users to move back and forth between screens.

Furthermore, React Navigation offers a collection of customisable components for creating navigation UI components including headers, buttons, and tabs. A consistent user experience is offered across all screens via these components, which can be customised to match the appearance and feel of the program.

## Expo

Throughout this project, both the Expo Go application and Expo SDK components were heavily utilised. Expo is a collection of tools and services based on React Native that facilitates application testing and gives users access to mobile hardware. Normally, a macOS system is required to generate React Native projects using native iOS code, but Expo makes it possible to evaluate and debug on a Windows machine, which was crucial to the development and testing processing of the project.

## Node.js

Node.js is a cross-platform, open-source JavaScript runtime environment that enables the execution of JavaScript code on the server-side. Its purpose is to provide an asynchronous event-driven architecture that allows for the creation of scalable network applications. React Native, on the other hand, is a JavaScript framework used for building mobile applications. Given that React Native was employed for the front-end of the application, working with Node.js for the back-end development was a natural fit, as it allowed for a unified codebase written in JavaScript that runs seamlessly on both the client and server-side.

## Arduino IDE

The software program known as the Arduino IDE (Integrated Development Environment) is used to develop, compile, and upload code to Arduino microcontrollers.

The Arduino IDE comes with a text editor for writing code, a compiler for turning the code into machine-readable form, and an uploader for sending the compiled code to the Arduino board. Additionally, it includes several pre-written libraries that make it simple to add functionality to your project without having to start from scratch.

The open-source aspect of the Arduino IDE, which allows users to edit the code and contribute to the software's development, is a major benefit. The IDE also supports a wide variety of boards, including the ESP32 which was used for this project.

## Firebase

Google Firebase is a platform designed for the development of mobile and web applications. The platform offers a diverse set of tools and services that developers can employ to create applications of high quality. It is a backend-as-a-service (BaaS) platform, which means that it allows developers to focus on the frontend of their applications without worrying about the server-side infrastructure.

Firebase offers a wide range of features that can be used individually or together to build robust applications. Firebase's authentication service provides easy-to-use authentication and authorisation solutions for mobile and web applications. This service supports a range of authentication methods such as email/password, phone number, and social login.

## Redux

Redux is a library designed to manage the state of a React Native application, which plays a critical role in achieving an efficient and maintainable codebase. It does so by providing a centralised and predictable approach to state management that helps simplify the code and enhances its maintainability. Redux stores the application state in a global store and leverages actions, which are plain JavaScript objects, to describe changes to the state. These actions are dispatched to the store, and reducers, which are pure functions, promptly respond by updating the state. By using these features, developers can achieve a streamlined and scalable approach to state management in their React Native applications.

# Hardware Overview

The fundamental objective of an alcohol breathalyser is to gauge the concentration of alcohol in an individual's breath, which underscores the significance of a suitable hardware configuration for the device's precision. This configuration is precisely customised to align with the specific requirements of the project and encompasses various essential components like a MQ3 sensor, microcontroller, LCD display, power supply, and calibration tools. By seamlessly integrating these components, an entirely operational alcohol breathalyser can be developed and be adjusted to cater to the user's demands.

## MQ3 Sensor

An MQ3 sensor, also known as a gas sensor, is used in an alcohol breathalyser system to measure the concentration of alcohol in a person's breath. The MQ3 sensor works on the principle of the change in electrical conductivity of the sensing material when exposed to the alcohol vapours. As a person exhales into the breathalyser, the sensor detects the presence of alcohol in the breath, and the resistance of the sensor changes in response to the level of alcohol detected.

The MQ3 sensor is an essential component of an alcohol breathalyser device, as it provides a quantitative measure of the concentration of alcohol in the breath, which is then used to calculate the blood alcohol content (BAC) of the individual. The BAC is calculated by analysing the resistance change in the sensor and converting it into a corresponding alcohol concentration using a pre-calibrated equation.

Overall, the MQ3 sensor's implementation in an alcohol breathalyser system enables accurate and reliable BAC measurements, which is critical for the safety of individuals and the prevention of alcohol-related accidents.

## LCD Display

The use of an LCD display is crucial in an alcohol breathalyser system to provide an immediate and clear output of the BAC measurement results. Typically, the LCD display shows the results in percentage or parts per million format, and the results are displayed briefly for the user to read and record. This feature offers several advantages, such as aiding in decision-making, especially in safety-critical situations. In order to promote personal safety and prompt action when needed, it is crucial for the BAC results to be easily visible and understandable on the LCD display.

## ESP32

The ESP32 is a microcontroller that finds versatile applications in various IoT settings, including alcohol breathalyser systems. In the context of the project, the ESP32 operates as the central processing unit, responsible for computing and executing the necessary functions required for measuring the user's BAC.

One of the key benefits of the ESP32 is its built-in Wi-Fi and Bluetooth capability, which facilitates seamless wireless communication with other devices, such as smartphones, for data analysis and transfer.

The usage of an ESP32 in an alcohol breathalyser system offers multiple advantages, such as high processing power, wireless connectivity, and data storage capabilities. These features allow the ESP32 to perform complex computations, communicate the results wirelessly to other devices, and store the measurements for future analysis, making it an optimal choice for alcohol breathalyser systems.

## LED’S

In alcohol breathalyser systems, Light Emitting Diodes (LEDs) function as indicators that visually inform the user of their Blood Alcohol Content (BAC) level. The LEDs are available in two colours, green and red, with green indicating a BAC level that falls within the legal limit, and red indicating a level that surpasses it.

When a user breathes into the MQ3 sensor, their breath is analysed, and the BAC level is determined. Based on this value, the LED lights up either green or red. If the BAC level is below the legal limit, the green LED illuminates, signifying that the user has passed the test. Conversely, if the BAC level is over the legal limit, the red LED turns on, indicating that the user has failed the test.

Alcohol breathalyser systems use Light Emitting Diodes (LEDs) as a means of displaying results due to their various benefits, such as being cost-effective, durable, and energy efficient. LEDs also have a high visibility, making them suitable for use in dim environments. Alcohol breathalyser systems use LED technology to indicate BAC levels promptly and clearly, thereby promoting responsible drinking habits and safety.

## Piezo Buzzer

A Piezo Buzzer is used as an auditory indicator to alert the user that they have exceeded the legal BAC limit. The Piezo Buzzer is an electronic device that produces sound when an electric current is passed through it, and it is commonly used in applications that require a loud and clear audible signal.

When the user exhales into the breathalyser, the device measures the BAC level, and if it exceeds the legal limit, the Piezo Buzzer is activated, producing a loud and distinct sound. This sound serves as a warning to the user that they should not operate a vehicle or perform other activities that may require coordination and attention until they are sober.

The use of a Piezo Buzzer in an alcohol breathalyser system offers several advantages, including low power consumption, compact size, and a high sound output. The sound produced by the Piezo Buzzer is easy to hear and can be heard even in noisy environments. This makes it an ideal choice for alcohol breathalyser systems, as it provides an effective warning to the user that they should not operate a vehicle or engage in other activities that may put themselves or others at risk.

## Push Button

In an alcohol breathalyser system, a push button serves as the trigger to begin the BAC testing process. The user must hold down the button for approximately five seconds to allow the device to calibrate and prepare for use.

After loading the system, the user must exhale into the MQ3 sensor and this measures the concentration of alcohol in the user’s breath and sends this information to the microcontroller for analysis.

The use of a push button in an alcohol breathalyser system offers several advantages. First, it provides a simple and convenient way for the user to initiate the testing process. Second, it helps prevent accidental or unintended testing, ensuring precise and reliable results. Lastly, it helps conserve power by allowing the system to remain in standby mode until needed.

# Project Architecture

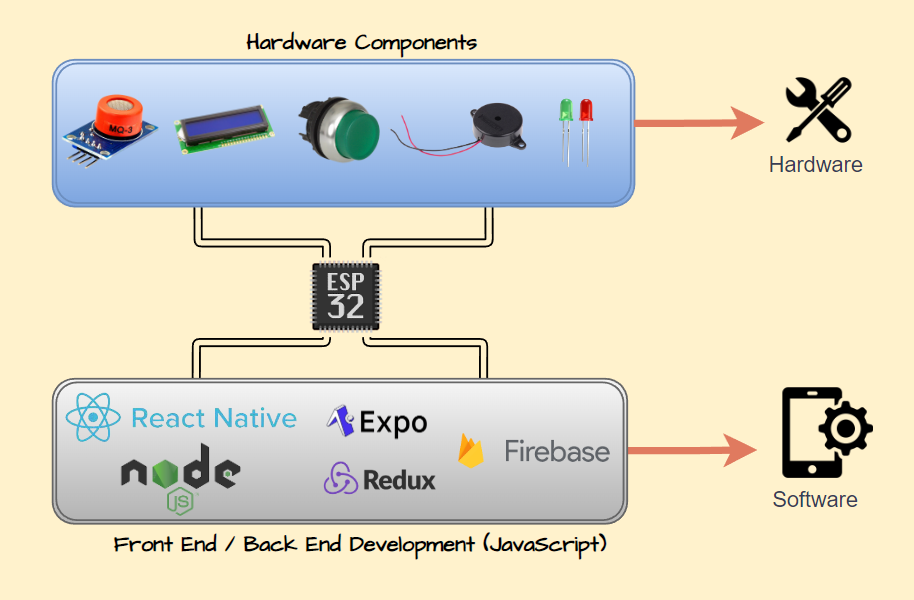


Figure 7.1 - Architecture Diagram

# Project Logo

****

Figure 8.1 – Project Logo

# Project Plan

In the context of my project planning and management, I set up the advanced functionalities of the JIRA project management software, which enabled me to streamline my workflow and optimise my productivity. Additionally, I employed a detailed and organised approach to logging my project tasks, employing comprehensive project logs to precisely track progress and set reminders for upcoming deadlines. By doing so, I was able to effectively manage my workload and ensure timely completion of project deliverables.

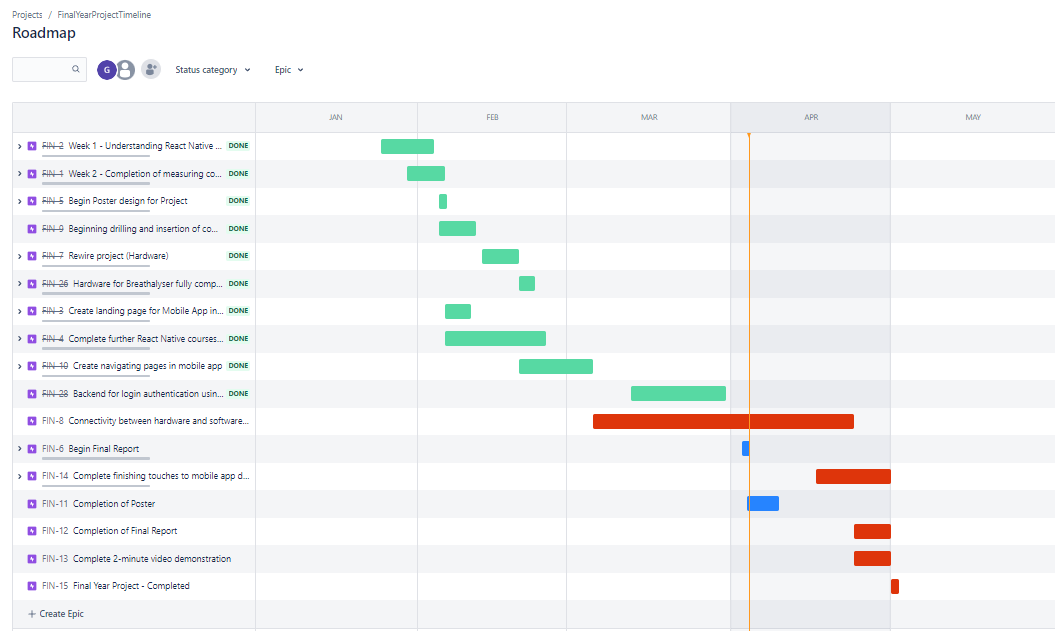


Figure 9.1 - Jira Project Roadmap

Throughout the entirety of my final year project, I found myself heavily reliant on the robust and versatile project management capabilities that OneNote offered. More specifically, I capitalised on the utility of OneNote's detailed project logs to meticulously document and monitor the progression of my work, including significant milestones and achievements that arose along the way.

Consistently, on a weekly basis, I dedicated a sizeable portion of my time towards updating and refining my project logs to reflect the status and developments of my project, thereby providing me with a precise and accurate representation of my progress and alignment with my overall project goals and objectives. By adhering to this diligent and organised approach to project management, I was able to maintain a clear and well-informed perspective on my project's trajectory and avoid any substantial setbacks or delays that would have impeded my progress.

My strategic and effective use of OneNote's project logs played an essential role in facilitating my successful navigation of the complex and versatile landscape of my final year project, culminating in an exceptional level of achievement, and surpassing of expectations.

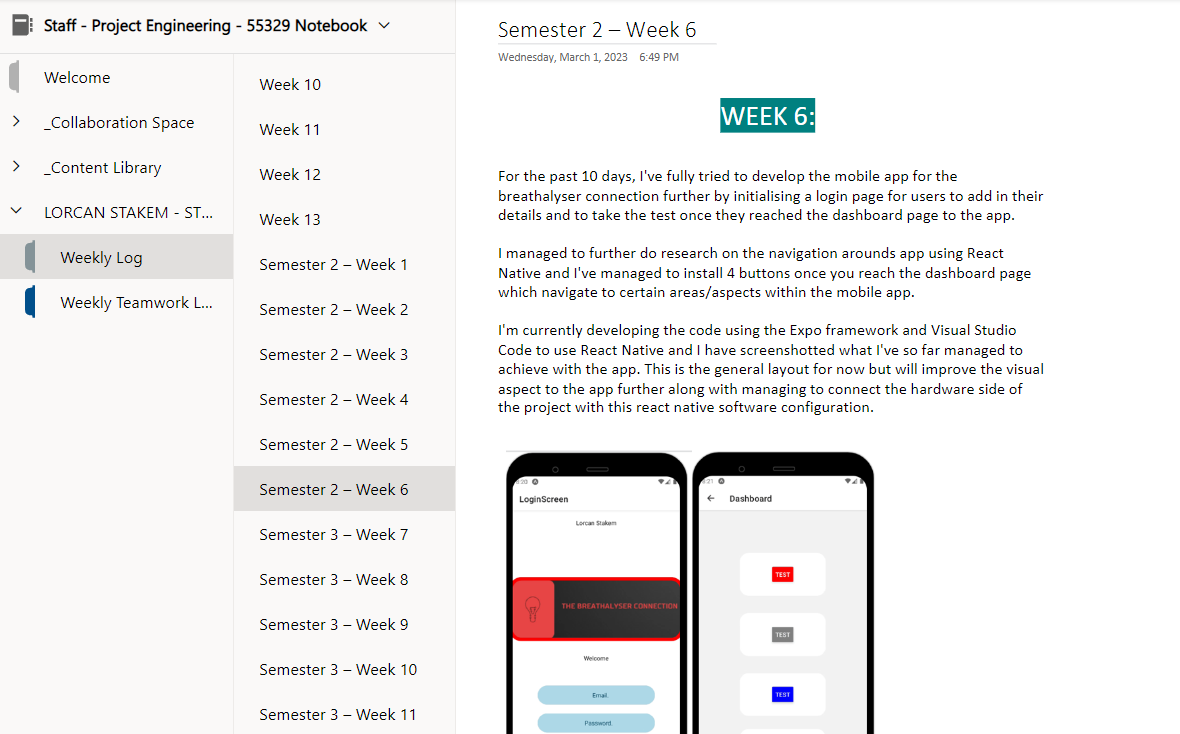


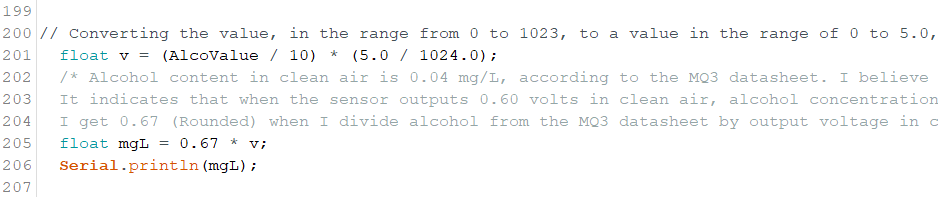
Figure 9.2 – OneNote Project Logs

# Project Code

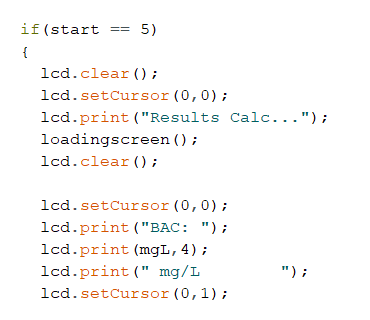
To start this project, I conducted extensive research on various topics related to both the software and hardware components. After further analysis, I made the decision to divide the project into two distinct parts. During the first semester, I focused on configuring the hardware and integrating all the necessary components needed for the project using the Arduino Software (IDE). Subsequently, in the second semester, I devoted my attention towards developing the mobile application using React Native which is based in JavaScript.

To develop the hardware configuration for the breathalyser system, I initiated the process by programming with Arduino IDE and working with the C++ programming language. This enabled me to design a program that established a communication channel between the Arduino board and the different device components.

Accurately determining the blood alcohol concentration (BAC) readings from the MQ3 sensor was a crucial aspect of integrating it into the breathalyser system as it is the primary component for detecting the amount of alcohol present in a person's breath. The configuration of the pins on the ESP32 to interpret the MQ3 sensor's analog output was the first step taken. The sensor can detect the presence of alcohol in a person's breath, and outputs an analog signal proportional to the concentration of alcohol present.

The concentration of alcohol in the air is calculated by converting the analog voltage value read from the MQ3 sensor to a corresponding alcohol concentration value. First, the value reads from the sensor from the range of 0 to 1023 and is converted to a voltage value in the range of 0 to 5.0 volts using a formula that multiplies the sensor value by 5.0 and divides it by 1024. This gives a voltage value that is proportional to the concentration of alcohol in the air.

Next, a conversion factor of 0.67, which is derived from the MQ3 datasheet, is used to calculate the alcohol concentration. According to the MQ3 datasheet, the output voltage of the sensor in clean air is 0.60 volts, which corresponds to an alcohol concentration of 0.04 milligrams per litre. Dividing 0.04 by 0.60 yields 0.67. This value is then multiplied by the voltage value obtained from the sensor to get the alcohol concentration in milligrams per litre (mg/L).



To trigger a Blood Alcohol Content (BAC) reading, the user is required to exert pressure on a push button for a duration of five seconds while simultaneously exhaling into the MQ3 sensor. Following the release of the push button, the system automatically conducts a reading and exhibits the outcome on the Liquid Crystal Display (LCD) screen.



As the program runs, it continuously checks the alcohol concentration value that was calculated earlier when exhaling into the MQ3 sensor. If the value is greater than 0.5 milligrams per litre (mg/L), which is above the legal limit for driving in the Republic of Ireland, the program will trigger an alarm to warn the user not to drive. A message indicating failure will be displayed on the LCD screen, while the system will generate an output message for the Serial Monitor.

The alarm will be activated by turning on the piezo buzzer and emitting a tone at a frequency of 1000 hertz for a duration of one second. Additionally, the red LED will be turned on for 10 seconds to further signal a warning. After 10 seconds, the buzzer will be turned off, the red LED will be turned off, and the LCD screen will be cleared.

If the alcohol concentration value is less than or equal to 0.5 mg/L, the program will indicate that the user has passed the BAC test. The LCD screen will display a passed message and the system will output a message to the Serial Monitor saying "Normal". The green LED will be turned on for 10 seconds to indicate that the test was successful. After 10 seconds, the green LED will be turned off and the LCD screen will be cleared.

After either the "FAILED" or "PASSED" message is displayed, the program will wait for 5 seconds before resetting the ESP32 using the resetFunc function. This will cause the program to return to the start of the setup() function and start the breathalyser test again.

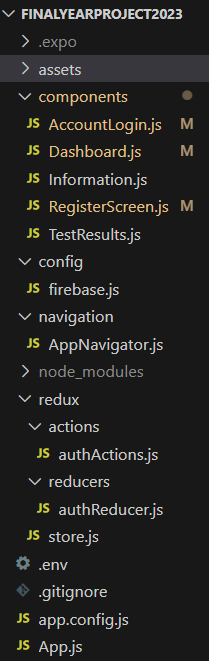
After finalising the code configuration, I took on the responsibility of designing an enclosure box that would seamlessly accommodate and integrate all the hardware components together. I precisely focused on the details of the project by separately reorganising and rewiring the different components within the enclosure. To achieve the most efficient outcome, I ensured each component was measured individually before installation. This allowed me to determine its exact location within the enclosure and drill the necessary holes for precise alignment. Upon completion of the rewiring of the project, I integrated an inhaler container onto the enclosure. The MQ3 sensor can detect the test sample with high accuracy and precision as the container offers a secure and enclosed space for it.



Completed hardware setup of the Breathalyser Connection

## 10.1 Software:

After successfully configuring the hardware, I proceeded to the software components of my project which I worked on during the second semester. This section will provide an elaboration of the approach taken towards the front-end code, which was divided into various primary sections on the Visual Studio Code platform. The App.js file of the application has been configured to enable the expenditure of the Redux store and AppNavigator component for smooth navigation throughout the entire application. In the navigation folder, in turn, houses the AppNavigator.js file, which serves as the primary homepage for the user upon accessing the mobile application, providing the essential framework for a mobile application complete with user authentication and navigation capabilities. The app's navigation structure is composed in components, whereas the responsibility of initialising Firebase and establishing the app's connection to Firebase services is assigned to the config folder. In the redux folder, there is two separate primary folders called actions and reducers which are related to authentication functionality in a web application using Firebase as a backend. Finally, the store.js file exports a Redux store that includes middleware for handling asynchronous actions.

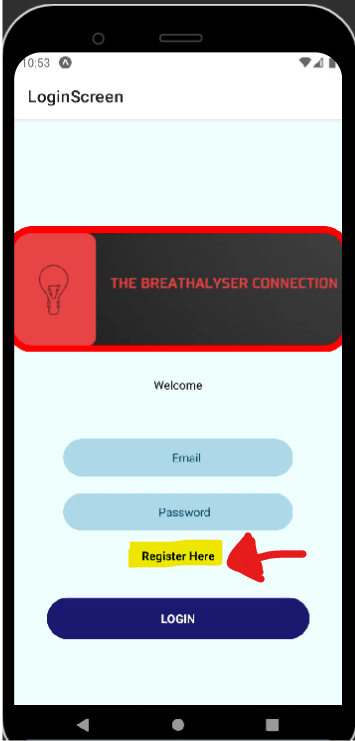


Within the AppNavigator.js file, it can be observed that every screen is contained within a Stack.Navigator component.



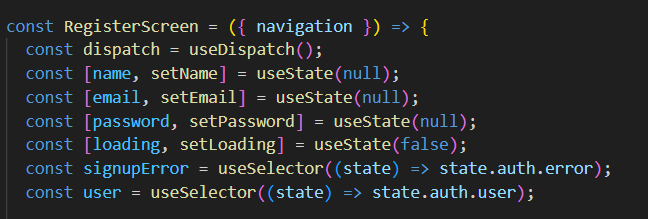
The application comprises multiple screens that the user can navigate. The initial screen encountered upon launching the app is the login screen, which can be accessed by providing registration details and subsequently logging in. Once successfully logged in, the user gains access to a plethora of navigation pages within the application.

To gain access to the mobile application, it is necessary for the user to initiate the registration process by pressing the designated “Register Here” button. This action will redirect the user to the Register Screen, where they can input their personal information and create a unique account. Once the registration is complete, the user will be able to log in and engage the full range of features provided by the mobile application.

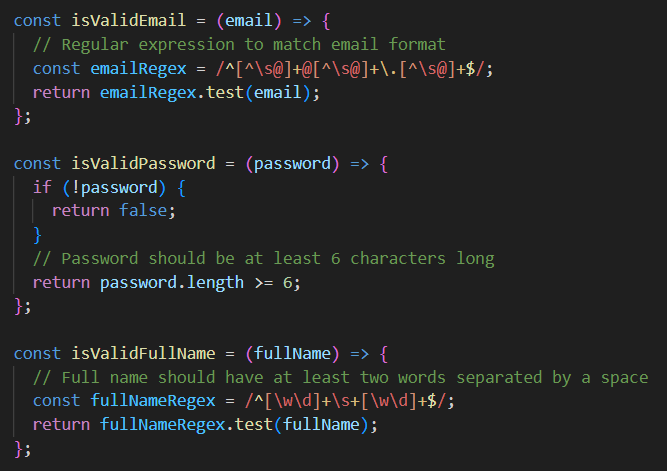
 Graphical user interface, text, application

Description automatically generated

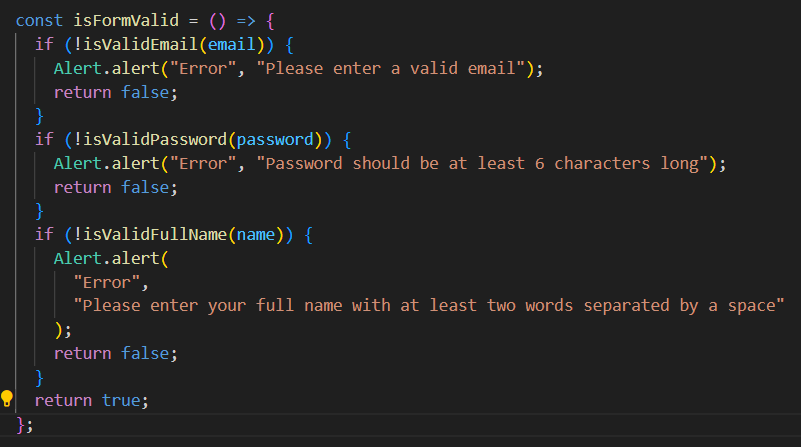
The Register Screen component is implemented using the arrow function syntax and receives a navigation prop as its input. Within the component, several constants and state variables are declared using the useState hook. Furthermore, the useSelector hook is used to fetch the authentication error and user data from the Redux store.



Three functions are defined to validate email, password, and full name inputs. The first function, isValidEmail, uses a regular expression to verify if the input email string conforms to the standard email format. The second function, isValidPassword, checks if the input password is at least six characters long, returning false if the password is empty or undefined. The third function, isValidFullName, uses a regular expression to check if the input fullName has at least two words separated by a single space.

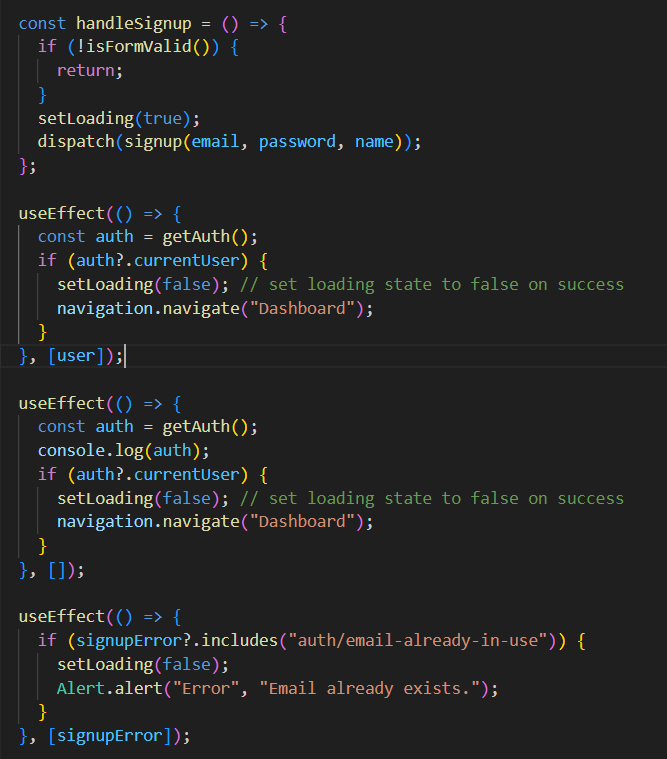


As a function, isFormValid is responsible for validating the email, password, and full name fields that is entered. It checks if these fields are valid or not. If any of them are invalid, it displays an error message using the Alert.alert method and returns false. However, if all of them are valid, the function will return true.

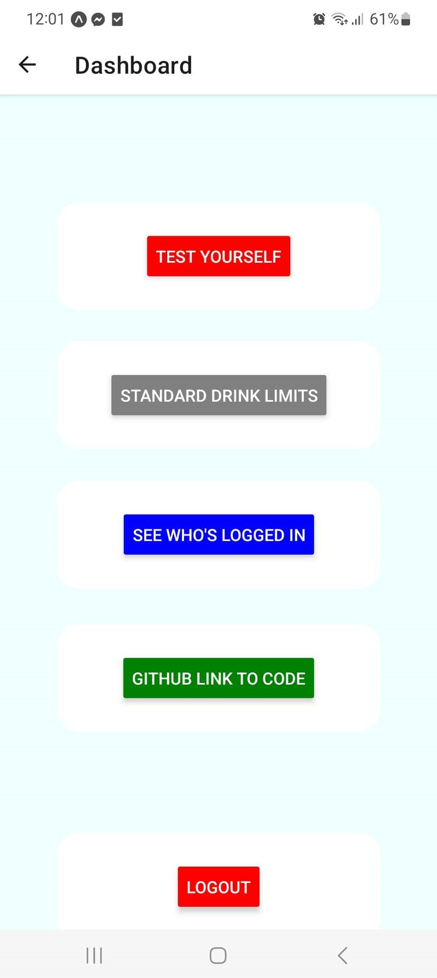


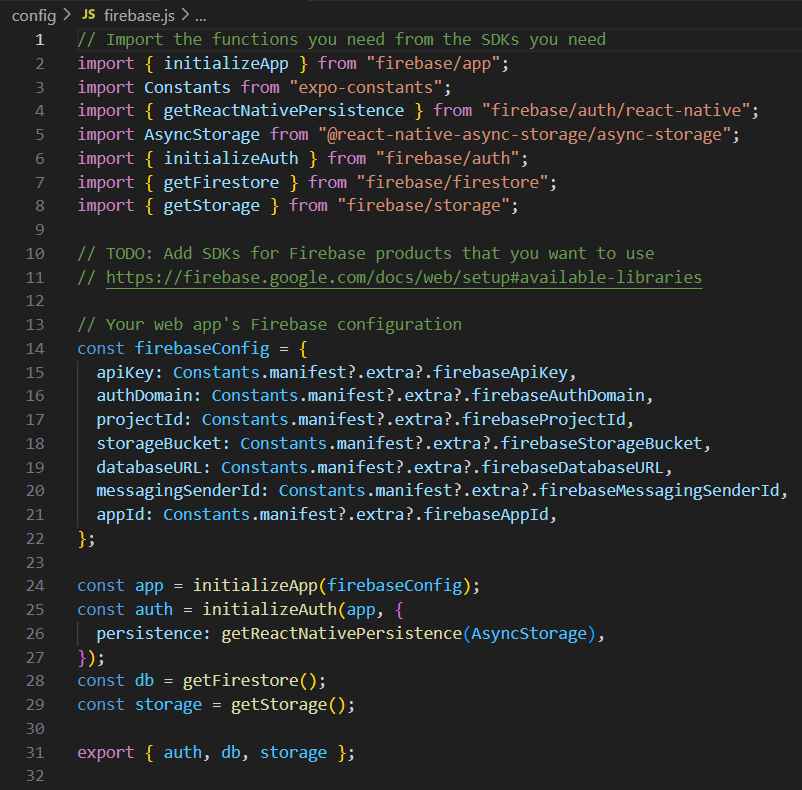
The handleSignup function is called when a user clicks on the signup button. It checks if the form is valid, and if so, sets the loading state to true and calls the signup function using the user's email, password, and name.

Three useEffect hooks are used. The first is triggered by a change in the user state, sets the loading state to false, and navigates to the Dashboard screen if the auth object has a currentUser property. The second runs once when the component mounts, checks for a currentUser property, and navigates to the Dashboard screen if present. The third useEffect hook monitors changes to the signupError state variable and, if it contains the "auth/email-already-in-use" string, sets the loading state to false and displays an error message.



The Dashboard component is presented to the user upon successful authentication, verified through Firebase authentication. The Dashboard component is responsible for rendering a view that displays a total of five buttons, each with its unique title, colour, and onPress function that directs the user to a separate screen within the app. Three of the buttons navigate to different areas of the app, while the fourth button leverages the Expo Web Browser module to open a link to the app's code repository on GitHub. Finally, the fifth button initiates a logout process by invoking the handleLogout function, which in turn triggers the dispatch of a logout action to the Redux store.

A

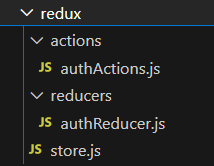


This code sets up the Firebase SDK in the React Native app by importing functions from the Firebase SDKs. These functions are used to initialise the Firebase app, authenticate users, and store data in the database and files in the storage.

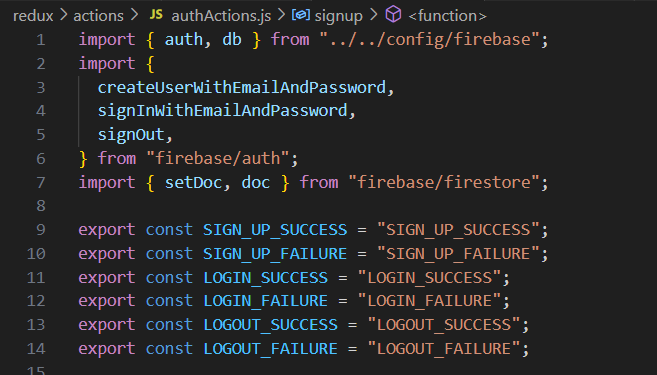
The Firebase app is initialised with the initializeApp function using the firebaseConfig object, which contains the necessary configuration information such as API key, authentication domain, project ID, storage bucket, database URL, messaging sender ID, and app ID.

The initializeAuth function is then used to set up Firebase authentication with the app object, and the React Native persistence is set to AsyncStorage.

Lastly, the getFirestore and getStorage functions are called to retrieve the Firestore database and Firebase storage object respectively, which can be used to interact with the app's database and storage.



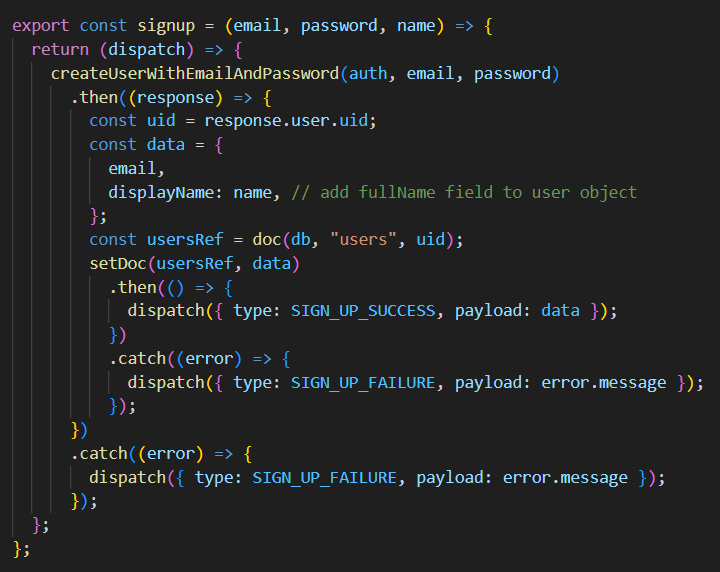
The above screenshot shows the implementation of user authentication functionality using Firebase, Redux, and Thunk middleware. There are two folders - actions and reducers, and a store.js file.



Within the actions folder, you will find the authActions.js file, which is responsible for outlining the various actions that can be executed within the application. The file comprises of three key functions: signup, login, and logout, which uses Firebase's authentication mechanisms to facilitate user signup, login, and logout functionalities.

The code imports the Firebase configuration and employs it to enable user account creation and login using email and password authentication. The signup function is an asynchronous action creator that registers a new user and updates their display name in the Firestore database. Similarly, the login function is also an asynchronous action creator that verifies user authentication by validating their email and password. Lastly, the logout function is another asynchronous action creator that logs out the currently signed-in user.

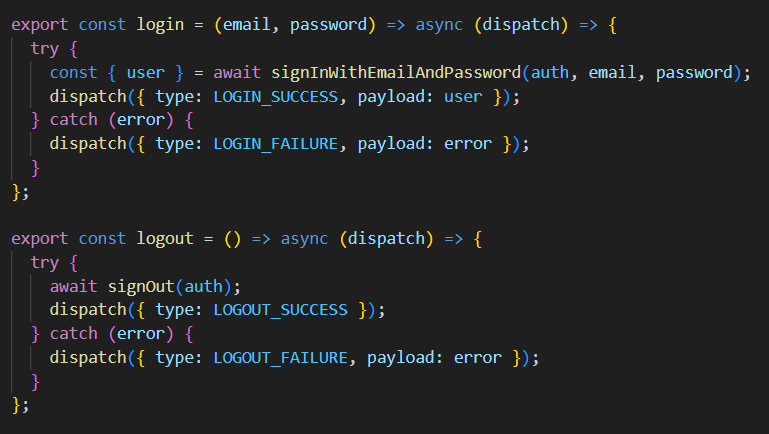
Specific action types, namely SIGN\_UP\_SUCCESS, SIGN\_UP\_FAILURE, LOGIN\_SUCCESS, LOGIN\_FAILURE, LOGOUT\_SUCCESS, and LOGOUT\_FAILURE, are dispatched upon successful or failed authentication events. These action types are defined as constants.



This signup function is designed to create a new user account and update the user's display name in the Firestore database. The function expects three arguments: email, password, and name. When called, it returns a function that accepts dispatch as an argument.

Inside the returned function, it uses the Firebase createUserWithEmailAndPassword method to create a new user account. Upon successful account creation, the user's uid is retrieved and a new data object is created with the user's email and display name. The setDoc method is then used to set this data in the Firestore database under the user’s collection, using the user's uid as the document ID.

Finally, the function dispatches either the SIGN\_UP\_SUCCESS or SIGN\_UP\_FAILURE action types, depending on whether the Firebase operations were successful or not.



The login function is an action creator that authenticates a user's email and password. It returns an asynchronous function taking dispatch as an argument. Using Firebase's signInWithEmailAndPassword method, the function authenticates the user's credentials. Upon successful authentication, it dispatches LOGIN\_SUCCESS action type with the user object as the payload. On the other hand, if the authentication fails, the function dispatches LOGIN\_FAILURE action type with the error object as the payload.

The logout function, on the other hand, logs out a user. It returns an asynchronous function taking dispatch as an argument. The function uses Firebase's signOut method to log out the user. If log out is successful, the function dispatches LOGOUT\_SUCCESS action type. However, if log out fails, the function dispatches LOGOUT\_FAILURE action type with the error object as the payload.

The reducers folder contains the authReducer.js file which manages the state for the user authentication and oversees the action types dispatched by the signup, login, and logout functions.

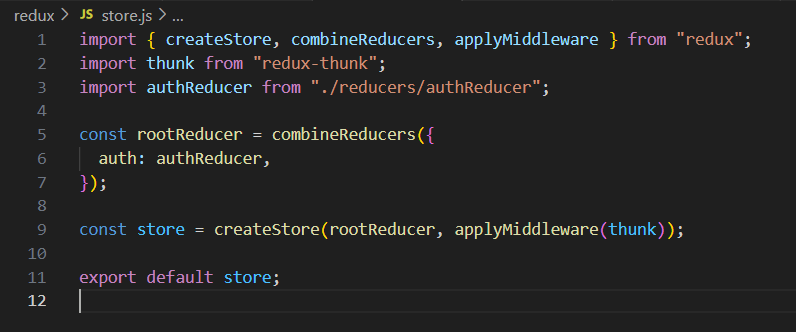


The authReducer function takes two arguments: the current state and an action object, which describes what changes should be made to the state. The code initialises the initial state of the authentication reducer using an object named initialState, which has two properties - user and error. The user property is set to null to indicate that no user is logged in initially, and the error property is set to null to indicate that there are no errors at the start.

If the action type is "SIGN\_UP\_SUCCESS" or "LOGIN\_SUCCESS", the authReducer returns a new state object that copies the previous state using the spread operator (...state) and overwrites the user property with the payload property of the action object. It also sets the error property to null. If the action type is "LOGOUT\_SUCCESS", the authReducer returns a new state object that copies the previous state using the spread operator and sets both the user and error properties to null.

If the action type is "SIGN\_UP\_FAILURE", "LOGIN\_FAILURE", or "LOGOUT\_FAILURE", the authReducer returns a new state object that copies the previous state using the spread operator and overwrites the error property with the payload property of the action object.

In case an action type is not recognised, the present state object is unchanged and returned. The authReducer module is exported as a default and is used in other sections of the application.

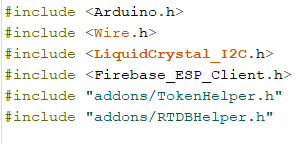


The code above sets up a Redux store for the application by importing necessary functions from the Redux library such as createStore, combineReducers, and applyMiddleware along with the thunk middleware for handling asynchronous actions. The authReducer function defined in the ./reducers/authReducer.js file and specifies how the application state should change in response to different actions dispatched to the store.

The combineReducers function combines all reducers into a single reducer function for the store, where authReducer is the only reducer combined.

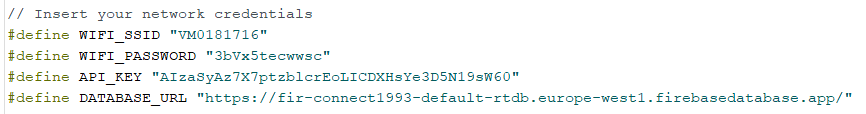
The createStore function creates a new Redux store, initialising it with rootReducer and the thunk middleware and the store is exported for other parts of the application to access its functionality to manage the application's state.

Firebase's real-time database feature is created for precise calibration of breathalyser test outcomes and their incorporation into a database, which captures and presents real-time data feeds of user-generated test results on the TestResults.js navigation page of the mobile application. This approach results in the development of an effective and user-friendly solution for the seamless monitoring and analysis of breathalyser test results.



The code shown in the screenshot above includes several libraries that are necessary for implementing various functionalities in the project. The "Arduino.h" library is a fundamental requirement for compiling any Arduino sketch, while the "WiFi.h" library is used to establish connections to Wi-Fi networks. Additionally, the "LiquidCrystal\_I2C.h" library is included to enable the control of LCD displays over I2C.

Moreover, the "Firebase\_ESP\_Client.h" library is used to interface with the Firebase Realtime Database, which allows for the storage and retrieval of data in real-time. This library is further enhanced by two addons, namely "TokenHelper.h" and "RTDBHelper.h," which provide additional functionalities to enable secure authentication and efficient data handling in the Firebase Realtime Database.



To enable the device to connect to the internet, the code encodes the Wi-Fi network name and password. The device requires these credentials, specifically the network name or SSID and password, to successfully connect to the Wi-Fi network. To maintain these values consistently, they have been declared as constants, with WIFI\_SSID representing the network name and WIFI\_PASSWORD representing the password.

Furthermore, as the application necessitates the use of a Firebase Realtime Database to store and access data, an API key is employed to verify the device's access to Firebase services. The API key functions as a unique identifier to authorise the device to use Firebase services and is stored within the code as a constant variable as API\_KEY.

Finally, the web address of the Firebase Realtime Database, where the device stores and retrieves data, is specified. This URL is declared as a constant variable called DATABASE\_URL and serves as a connection point to the specified database location. Providing this URL within the code enables the device to establish a connection with the Firebase Realtime Database.

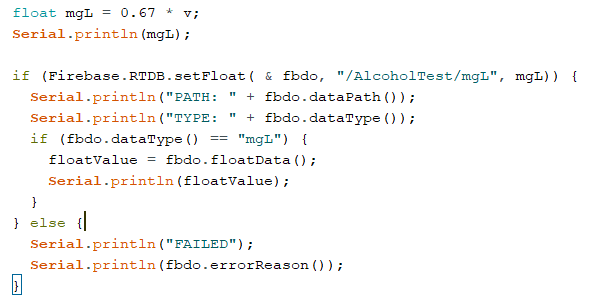


This code initiates the process of connecting the device to a Wi-Fi network and establishing authentication to access Firebase Realtime Database services. The code starts by calling the WiFi.begin function and providing the Wi-Fi network name and password as arguments.

To ensure a successful Wi-Fi connection, a while loop repeatedly checks the status of the connection using the WiFi.status function. This loop continues until a connection is established.

Once the device is successfully connected to Wi-Fi, the Firebase.signUp function is invoked to authenticate the device with Firebase using the API key that is defined as a constant called API\_KEY. Additionally, the Firebase Realtime Database URL is included and is specified by the constant variable DATABASE\_URL, to identify the specific database location that the device needs to access.

The Firebase.signUp function returns a Boolean value that is used to check if the authentication process was successful or not. If the authentication is successful, the signupOK variable is set to true, which indicates that the device has been authorised to access the Firebase services. However, if authentication fails, an error message on the Serial Monitor is shown and the device attempts to authenticate again.



The code above conducts various data processing and storage operations. The data processing operation involves the calculation of a value of mgL, which represents the concentration of alcohol in a person's blood and the data storage operation involves sending the calculated value of mgL to the Firebase Realtime Database,

After the mgL value is calculated, the Firebase.RTDB.setFloat function sends it to the Firebase Realtime Database. This function requires three arguments: a reference to the Firebase Data Object called fbdo, the path where the data is to be stored, and the value of mgL. In this case, the data path is specified as "/AlcoholTest/mgL".

Upon successful storage of data, the Serial Monitor displays the dataPath and dataType of the fbdo object. If the operation is successful, the mgL value is retrieved from floatData and saved in the floatValue variable. However, if the storage operation fails, an error message is displayed in the Serial Monitor using the errorReason function of the fbdo object.

Graphical user interface, text, application, email

Description automatically generated

After the mgL has been obtained and evaluated in the loop function, the BAC results are shown not only on the LCD display but also on the Real Time Database, as indicated earlier. Initiating the breathalyser test is signalled by pressing and holding down the push button for 5 seconds, which prompts the database to refresh the data on a regular basis.

The React Native component called "TestResults" communicates the result of the alcohol breathalyser test to the mobile app via the Firebase Real Time Database.



The code retrieves information from the Real-Time Database and displays it on the mobile application's screen. The displayed information comes from a specific location in the Real-Time Database, which in this case is the path "/AlcoholTest/mgL." The retrieval process is executed in the code by creating a database reference with the "ref" function and passing in the relevant path and database instance as arguments. Once the data is retrieved, it is stored in the "data" state variable using the "setData" function. The "useEffect" hook ensures that this retrieval process is only executed once during the component's mounting phase.

Diagram

Description automatically generated with low confidence

BAC test result reading from the React Native App

# Ethics

Having grown up in a society where alcohol consumption has been widespread for decades, I acknowledge that consuming excessive amounts of alcohol can be hazardous and raises the risk of accidents. To prevent drink driving and related accidents, breathalysers have been developed, which are praised for their potential to save lives. However, I also recognise that there are ethical issues that need to be considered when using breathalysers.

One of the ethical concerns with breathalysers is the invasion of privacy. Personal data such as my blood alcohol concentration levels can be collected, which may be used against me in court or by insurance companies. To ensure privacy is respected, clear guidelines for the collection and use of breathalyser data must be established.

Another significant ethical issue is the accuracy and reliability of breathalysers. False positive results can lead to unfair arrest and prosecution, while false negative results can result in dangerous situations on the road. Therefore, I believe it is crucial to ensure that breathalysers are regularly calibrated, tested, and maintained to guarantee their accuracy and reliability.

In summary, I understand that while breathalysers have the potential to save lives and prevent drink driving accidents, ethical issues such as privacy concerns, accuracy and reliability, and legal implications must be considered and addressed. By doing so, I believe breathalyser testing can be conducted in a fair and ethical manner, ultimately making Irish roads safer.

# Conclusion

The Breathalyser Connection provides in promoting responsible drinking and preventing incidents associated with drink driving. The device for the project gauges the alcohol content in an individual's breath and translates it into a BAC reading, which is then showcased on the LCD screen and transmitted to the user's smartphone. The mobile app provides further information on the effects of alcohol on the body, enabling them to make informed decisions about their alcohol consumption.

Real-time BAC readings allow users to avoid driving if they exceed the legal limit, while the app enables users to monitor their BAC levels over time and identify patterns in their drinking behaviour, leading to necessary changes. The combination of various components and features makes this project an effective and convenient solution to preventing incidents related to drink driving.

In conclusion, the Breathalyser Connection project was a success, as it met the project's initial objectives of creating a functional breathalyser prototype. The project provided valuable insights into breathalyser technology, including formula configuration for accurate BAC readings, as well as a hands-on experience with mobile development, project management tools, and professional project delivery.

However, due to other priorities and unexpected issues with the Firebase configuration, there was not enough time to develop a portable version of the breathalyser system as initially intended. This goal remains an opportunity for future development of the Breathalyser Connection project.

# Code

To access the code for the Breathalyser Connection, you may do so by scanning the bar code provided below:

Qr code

Description automatically generated

# References

|  |  |
| --- | --- |
| [1] | Bot1398More. “Arduino Breathalyzer.” *Instructables*, www.instructables.com/Arduino-Breathalyzer/. |
| [2] | “DIY Breathalyzer.” *Hackster.io*, www.hackster.io/coderscafe/diy-breathalyzer-d9541f. |
| [3] | Javaid, Aaliyan. “How to Create Animations on Arduino LCD.” *Linuxhint*, linuxhint.com/create-animations-arduino-lcd/. |
| [4] | “Drink Driving in Ireland.” *Drinkaware*, www.drinkaware.ie/drink-driving-in-ireland/. |
| [5] | “StackPath.” *Drinkaware.ie*, drinkaware.ie/wp-content/uploads/2021/01/Drinkaware-Research-Report-National-Barometer-Trends-2017-2018.pdf. |
| [6] | “Blood Alcohol Content.” *Www.utoledo.edu*, www.utoledo.edu/studentaffairs/counseling/selfhelp/substanceuse/bac.html. |
| [7] | “Build and Deploy Your First Modern React Native App | NFT Marketplace Course - Extremely Easy!” *Www.youtube.com*, www.youtube.com/watch?v=\_ivIUCSOZ78&ab\_channel=JavaScriptMastery. |
| [8] | “React Native Crash Course | Build a Complete App.” *Www.youtube.com*, www.youtube.com/watch?v=VozPNrt-LfE&ab\_channel=Academind. |
| [9] | “React Native Authentication with Firebase and Expo in 27 Minutes.” *Www.youtube.com*, www.youtube.com/watch?v=ql4J6SpLXZA&ab\_channel=MadeWithMatt. |
| [10] | “Email Authentication with Verification in React Native and Firebase App | React Native Tutorial |.” *Www.youtube.com*, www.youtube.com/watch?v=lBUjggiWQ1U&list=LL&index=2&ab\_channel=BugNinza. |
| [11] | Faruq, Yusuff. “Getting Started with React-Redux-Firebase.” *LogRocket Blog*, 10 Nov. 2021, blog.logrocket.com/getting-started-react-redux-firebase/. |
| [12] | *ESP32: Getting Started with Firebase (Realtime Database) | Random Nerd Tutorials*. 16 Sept. 2021, randomnerdtutorials.com/esp32-firebase-realtime-database/. |
| [13] | Khurana, Sukant. “A Toy IOT System for Detecting Alcohol Levels.” *Medium*, 14 Apr. 2019, medium.com/@sukantkhurana/a-toy-iot-system-for-detecting-alcohol-levels-9fff4082270f. |
| [14] | *Reactnavigation.org*, 2021, reactnavigation.org/docs/getting-started/. |
| [15] | “Redux Fundamentals, Part 3: State, Actions, and Reducers | Redux.” *Redux.js.org*, redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers. |
| [16] | “Using Firebase with React and Redux | Pluralsight.” *Www.pluralsight.com*, www.pluralsight.com/guides/using-firebase-with-react-and-redux. |
| [17] | https://www.facebook.com/verywell. “How Impairment from Alcohol Begins with the First Drink.” *Verywell Mind*, 2018, www.verywellmind.com/blood-alcohol-content-62695. |
| [18] | “How Does MQ-3 Alcohol Detector Work with Arduino and Detect Alcohol?” *Circuitdigest.com*, circuitdigest.com/microcontroller-projects/interfacing-mq3-alcohol-sensor-with-arduino. |
| [19]  [20] | “Firebase Authentication Limits.” *Firebase*, firebase.google.com/docs/auth/limits.  Carolli, Adrian. “So You Want to Build a Bluetooth App with React Native and Expo.” *Medium*, 5 Feb. 2023, blog.expo.dev/so-you-want-to-build-a-bluetooth-app-with-react-native-and-expo-6ea6a31a151d. |