

Stay Awake, Drive Safe

Conor Shields

C16312523

Submitted in partial fulfillment of the requirements for the degree of:

B.Sc. in Business Computing

Technological University Dublin

Year 4

Supervised by – Neil O'Connor

This is an original work. All References and assistance are acknowledged.

Signed: _____

Date:

01/05/20

Contents

Acknowledgments.....	4
Abstract	5
Project Introduction:.....	6
Requirements	7
Business Case of Stay Awake, Drive Safe.....	7
Requirements Analysis	7
Business Actors	7
Use Case Diagram	8
Objectives of Stay Awake, Drive Safe	9
Technologies Used	10
Application	10
Android Studio	10
Firebase	10
Firebase ML Face Detector	11
Firebase Authentication	11
Cloud Firestore	11
Google Maps API & Google Places API.....	12
Additional Technologies Used	12
Material Design.....	12
Glide API	12
JSON.....	12
Data Analysis	13
Python	13
Matplotlib	13
Version Control	14
Git and GitHub.....	14
Database Design.....	15
Cloud Firestore	15
Querying Database.....	22
Shared Preferences.....	23
Implementation.....	24
Implementing Real Time Facial Analysis	24

Data Analysis.....	26
Warning Implementation	29
Application Implementation	30
User Login	30
Registration	31
Dashboard	31
Track A Journey.....	32
Emergency Contact Store.....	35
Nearby Coffee Shops Feature	36
.....	36
Past Journeys.....	37
Future of the Project.....	38
Conclusion	39
References	43

Acknowledgments

I would like to take this opportunity to say a special thanks to everyone who helped me get to this stage of both college and this project.

Firstly, I would like to thank my project supervisor, Neil who has been a huge help throughout this process. I really appreciate all the help along the way and positive feedback that led me in the right direction on this project and has really helped me develop my planning and time management skills.

Also, I would like to thank Thoa for her feedback as a second reader and for her role as lecturer of Mobile Application Development which helped me greatly in the completion of this project.

Finally, I would like to thank my friends and family for keeping me going and encouraging me through this process. Especially my parents who have been very patient and encouraging since I have had to move back home full time due to the ongoing Covid 19 situation.

Abstract

Stay Awake, Drive Safe is an Android application that attempts to help reducing the risk of road traffic accidents that happen as a result of driver fatigue.

Through the use of Real Time Face Detection and Data Analysis, the application collects and analyses facial data and offers warnings when signs of fatigue are shown. Signs of fatigue are uncovered through the monitoring of a driver's blinking patterns. Once signs of fatigue are uncovered the user will be given an alert from the application and these warnings will increase if the driver continues showing signs of tiredness.

This application was designed and developed through the use of analysing data collected throughout the duration of the application development.

The aim of Stay Awake, Drive Sage is to provide drivers with a fatigue monitoring system without the need for additional hardware installations in their vehicle.

Project Introduction:

Name: **Stay Awake, Drive Safe**

Growing up in Ireland road traffic deaths are a reality of life. Everybody has been, or knows someone who has been, affected by road traffic accidents, my application sets out to reduce one factor which is a large contributor to road traffic accidents, driver fatigue.

The aim of this project is to reduce the amount of fatigue related road traffic accidents on our roads, according to data from the Road Safety Authority (RSA) is estimated that driver fatigue is a contributing factor to 1 in 5 driver deaths. Furthermore, tiredness related collisions are three times more likely to be fatal or result in serious injury.

I came about the idea for this project when I began learning to drive myself, I noticed a lot of advertising and warnings from the RSA about driver fatigue and wondered if there was a way of combating this. I also discussed with my father who is in the haulage industry about similar applications that come built in to newer, high end lorries. These systems monitor a driver's journey and can communicate with the driver's company and also have driver warning systems, this gave me the idea of developing an application on mobile so that these safety features can be available for everyone.

I also learned about the Allianz Safe Driver app use to track their policyholders driving habits in return for a reduction in insurance premium, this app does not offer any preventative measures for fatigue, but I believe my application would work with a similar business model.

As I began working on the development of this project, I quickly became aware it would be split into two sections, the Android Application section and the Data Analysis section which I did using Python.

Requirements

Business Case of Stay Awake, Drive Safe

The business case of Stay Awake, Drive Safe is a fairly straightforward one, the application offers a tracking and warning systems to drivers who may be at risk of experiencing driver fatigue, the hope is that the application will be able to contribute towards cutting down on road traffic accidents and in turn cutting down on road traffic deaths. I believe the perfect business case for this application would be working with an insurance company in order for policy holders to download and use the application in return for a reduction in their premium fees.

An example of an application that inspired me to follow this model is the Allianz, Safe Driver application. This application is downloaded by policy holders in return for benefits such as a discount on a premium or an increase in no claims bonuses. I believe this sort of business implementation would be perfect for the Stay Awake, Drive Safe application as it would result in a symbiotic relationship in which both sides the users and the company will benefit. The users of the application will see reductions and discounts in their premiums in return for access to their data which can in turn be used by the company to improve the accuracy of the application.

Requirements Analysis

In the requirements analysis phase of this application I tested a couple of similar applications already on the market. I found the market to be quite limited at the moment and the ones that are on there were basic and not very user friendly. I also found in my research that most of the competitors that claim to offer similar features require the installation of hardware into the vehicle.

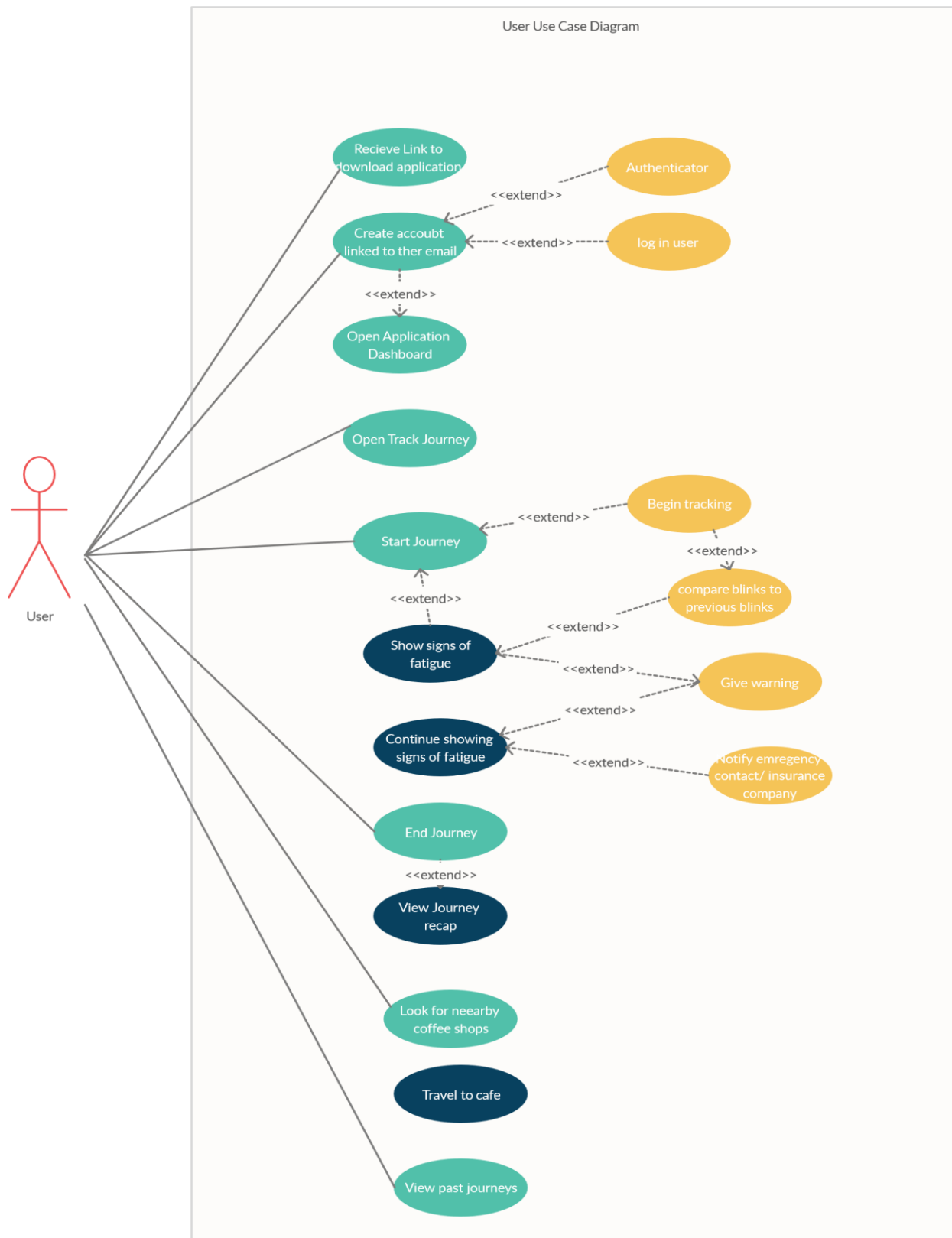
My goal was to create an application that would work just using a user's mobile phone and no additional installations.

Business Actors

In my application there is only one type of business actor, the user.

Everyone who registers is assigned the role of user. All users have access to the same features within the application. In order to register and become a user an account must be created using an email.

Use Case Diagram



Objectives of Stay Awake, Drive Safe

The primary objective of Stay Awake, Drive Safe is the monitoring of a person driving and the implementation of a warning system that will help to make drivers more aware of their fatigue levels and allow them to counteract this.

I decided to make this a phone application because when researching similar fatigue warning systems, I found most of them requiring a hardware installation into the car, such as a Raspberry Pi or similar technology which would interact with. Stay Awake, Drive Safe offers a quick out of the box solution to this issue with the only piece of hardware needed is a stand in the car for your phone so that it can view your face. Another limitation I found of apps I tested which claimed to offer a fatigue warning system was the simplicity of them. Most I tested just offered a warning whenever the person's eyes were closed, my app attempts to add nuance to this process. Instead of just notifying when a user's eyes were closed, I set out to monitor the drivers face and spot abnormalities through the use of data analysis.

When these signs are uncovered a warning is sent to the driver, this warning takes the place of an aural sound clip letting them know that they are showing signs of fatigue. The application also has an emergency contact store where the user selects one of their contacts to act as their delegated emergency contact, If the signs of fatigue continue and increase the app will send an SMS to this contact, giving them a warning and requesting that they check in on the user.

Throughout the designing and implementation of this application I always kept the RSA's "Stop. Sip. Sleep" campaign in my head and used it as a guide on what features to add to the application. I wanted to develop an application that would allow drivers to follow these guidelines in one place.

Stop: To help encourage drivers to stop I have developed the warning system which after continued warnings to the driver will advise them to pull over and pause their journey.

Sip: Once the driver has stopped their car, they can use the application to find nearby coffee shops or petrol stations where they can purchase a beverage. Once they select a coffee shop it will open google maps and direct them to the shop.

Sleep: The application will prompt the user to stop and have a nap if the warnings continue.

Technologies Used

I utilised numerous technologies in the design and implementation of this application. The primary technologies used are listed below and explained. The main programming language I used was Java and used Python for of the data analysis tasks.

Application

Android Studio



The Stay Awake, Drive Safe application was built solely in Android Studio using the Android SDK. Android Studio is a platform built off the IntelliJ IDEA and is the official IDE for Android Development. Android Studio offers a substantial suite of tools for application development, with built in Testing tools and Frameworks, a Gradle-based build system, an emulator and extensive Lint tools.

I carried out the testing of my application on my OnePlus 5T phone which offers much easier access to build in Android features such as using the camera and sending SMS messages. Developing on my own phone rather than on an emulator also allowed me to carry out real world testing of the application in a driving environment which would not be possible through the use of an emulator.

I chose to use Android studio with Java as it was a framework and development model which I was quite familiar with both from my own personal experience as well as using it for The Mobile Application module in the first semester of Fourth Year.

Firebase

Firebase is a Google Owned mobile and web application development tool, that offers numerous features to help build applications fast, without managing infrastructure. I choose to use a few Firebase technologies in my application as they work well in harmony as it is a well-integrated single platform where the different products can share data and insights between each other.

The Firebase platforms which I chose to integrate were the Firebase Machine Learning Kits face detector, Firebase Authentication for allowing users to register and login and Cloud Firestore Database.

Firestore ML Face Detector



Firestore ML is a mobile SDK that brings Google's Machine Learning to mobile applications in a ready-to-use package. The ML kit provides numerous APIs such as Text Recognition, Image Labelling and Face Detection.

Face Detection and monitoring is a huge part of my application. I decided to go with the ML face Detector after trailing multiple APIs such as OpenCV. I went with the Firestore ML face detector as it. I went with the Firestore ML face detector as I felt it worked best on Android and as I was already using Firestore in other aspects, it felt like a natural fit. Although this product is still in Beta, I found it quite a challenge as there are not a lot of resources online, so I was reliant on the official documentation.

Firestore Authentication



Firestore Authentication offers an easy registration and sign-in with any platform. It provides an end-to-end identity solution supporting many types of sign in options such as email and password, Google sign in and Facebook log in.

Firestore Authentication offers comprehensive security features and advanced encryption of passwords meaning that not even the admin of the database can view a user's password.

Firestore Cloud Firestore



Cloud Firestore is a NoSQL database stored in the cloud. It allows for structuring and querying of data in the way in which a user chooses. Firestore allows for the building of truly serverless applications, with strong security and is easily linked up with Firestore Authenticator.

I decided to go with Cloud Firestore after quite a bit of testing and trailing of other technologies. Originally, I had planned to use Firestore's Realtime Database and tested using SQLite and MongoDB for my application. I decided on Firestore as I felt it was most fit for my purpose. My application relies

on continuous storing of quite a large amount of data on the users' face and Firestore was the most appropriate tool for this and communicated seamlessly with the other Firebase products already implemented in the application.

Google Maps API & Google Places API



Google Maps API allows for built in map features in the application. I used this to show the beginning and end of a user's journey on a Journey Recap Screen

The Google Places API is used to get detailed information about places. I used this API to display to user's nearby coffee shops and petrol stations.

Additional Technologies Used

Material Design

Material Design is a comprehensive library for visual interaction design in Android applications. I used components from material design throughout the application such as CardView Layouts.

Glide API

Glide is a fast and efficient open source media management and image loading framework for Android that wraps media decoding, memory and disk caching, and resource pooling into a simple and easy to use interface.

JSON

JSON (JavaScript Object Notation) is an open-standard file format that uses key value pairs to structure its data to make it easy to retrieve and parse for use. I used JSON to parse data received from the Google Places API call

Data Analysis

I carried out tasks for the data analysis section of the application externally from the Android application, I decided to do this in a Python script which I used to generate graphs.

Python



Python is an interpreted, high-level, general-purpose programming language. I decided to use Python for my graphing section as I feel it is the most fit for the purpose of Data Analysis.

I am quite comfortable using Python from the third-year module Dynamic Programming Language as well as using it to create scripts in

my own time.

Matplotlib



Matplotlib is a plotting library for Python. It provides an API for the creation of numerous types of charts and plots.

I used this library with NumPy to create graphs in order to carry out data analysis for the project.

Version Control

Git and GitHub



I used GIT for local version control throughout the project and stored these on GitHub. Git and GitHub are very powerful tools which I found extremely useful in the development process to store my changes and develop new feature branches

Database Design

Cloud Firestore

Cloud Firestore is a flexible and highly scalable NoSQL database that is part of the Firebase ecosystem which is made by Google. The Cloud Firestore keeps all data in sync across client apps through the use of real time listeners and offers offline support, so you can build responsive apps that work regardless of network latency or internet connectivity. Cloud Firestore also works seamlessly with other Firebase products such as Firebase Authentication and Firebase Cloud Functions.

Cloud Firestore is a NoSQL database so instead of using tables and rows like relational databases, Cloud Firestore uses a series of Collections and Documents in order to store and structure the database. In Firestore you store data in documents that contain fields mapping to values. Documents are then stored in Collections, which are used to organise data and build queries. Documents support the storing of any data type from simple strings to nested collections. Cloud Firestore also has quite a flexible, efficient querying system, that can range from a shallow query such as getting all the documents in a database to deep nested querying such as finding a specific data item from a sub-collection. These queries can also be sorted, filtered and can have limits applied to them. As well as this Cloud Firestore offers a series of Realtime Listeners which can trigger events when a change happens in a database.

Cloud Firestore also offers offline support which I identified as a key feature for my application as the app is designed to be used by drivers going on long journeys who could be travelling in and out of connectivity. Cloud Firestore caches data that your app is actively using, so the app can write, read, listen to, and query data even if the device is offline. When the device comes back online, Cloud Firestore synchronizes any local changes back to Cloud Firestore.

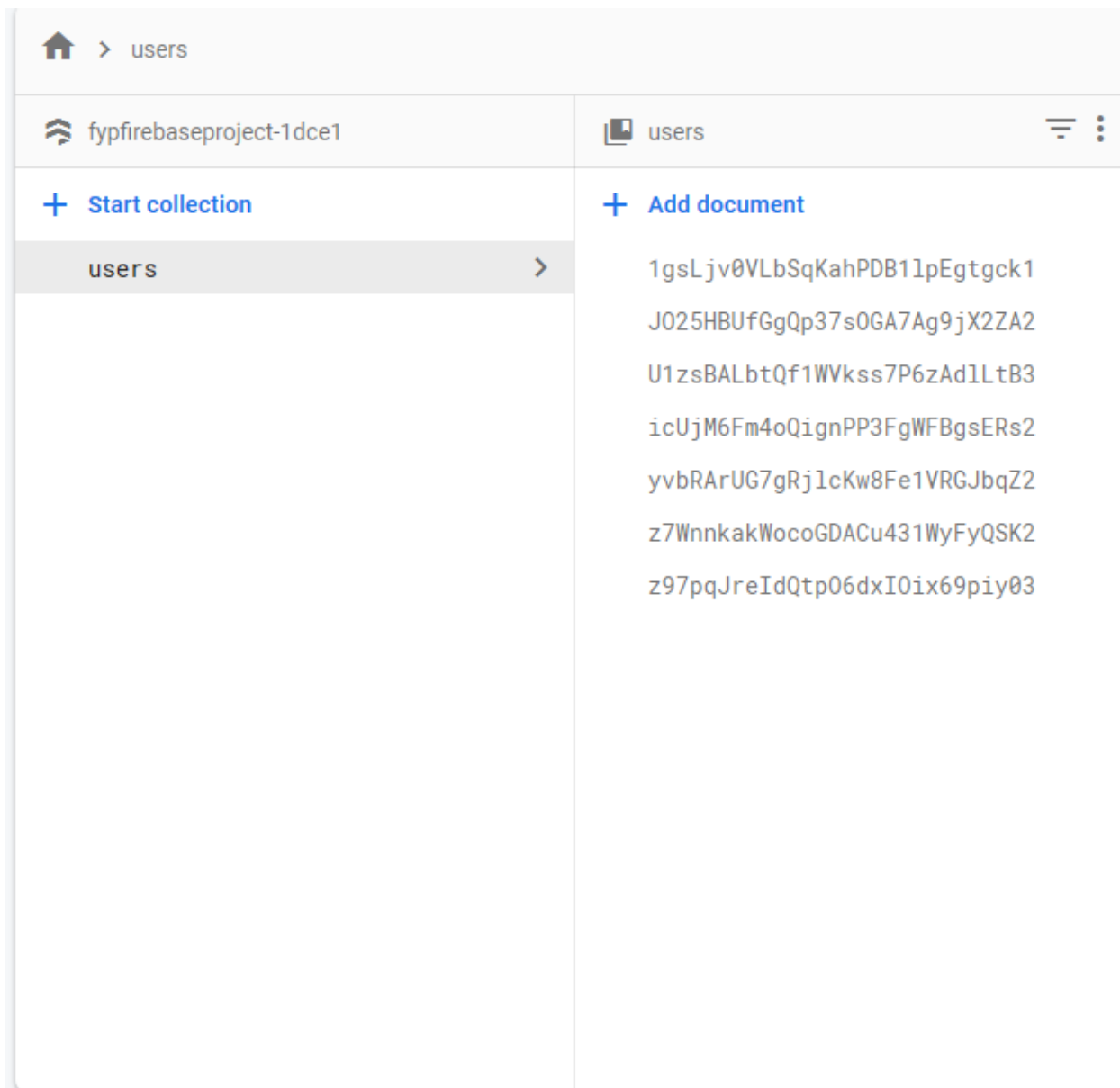
Scalability was another key factor as to why I chose Cloud Firestore as my Database for Stay Awake, Drive Safe. For the analysis I needed to collect large amounts of Data, this meant choosing a database with a strong infrastructure that is able to handle massive amounts of data being written to and read from.

Overall the reasons which I chose Cloud Firestore for my application were for the key capabilities it offers, these are:

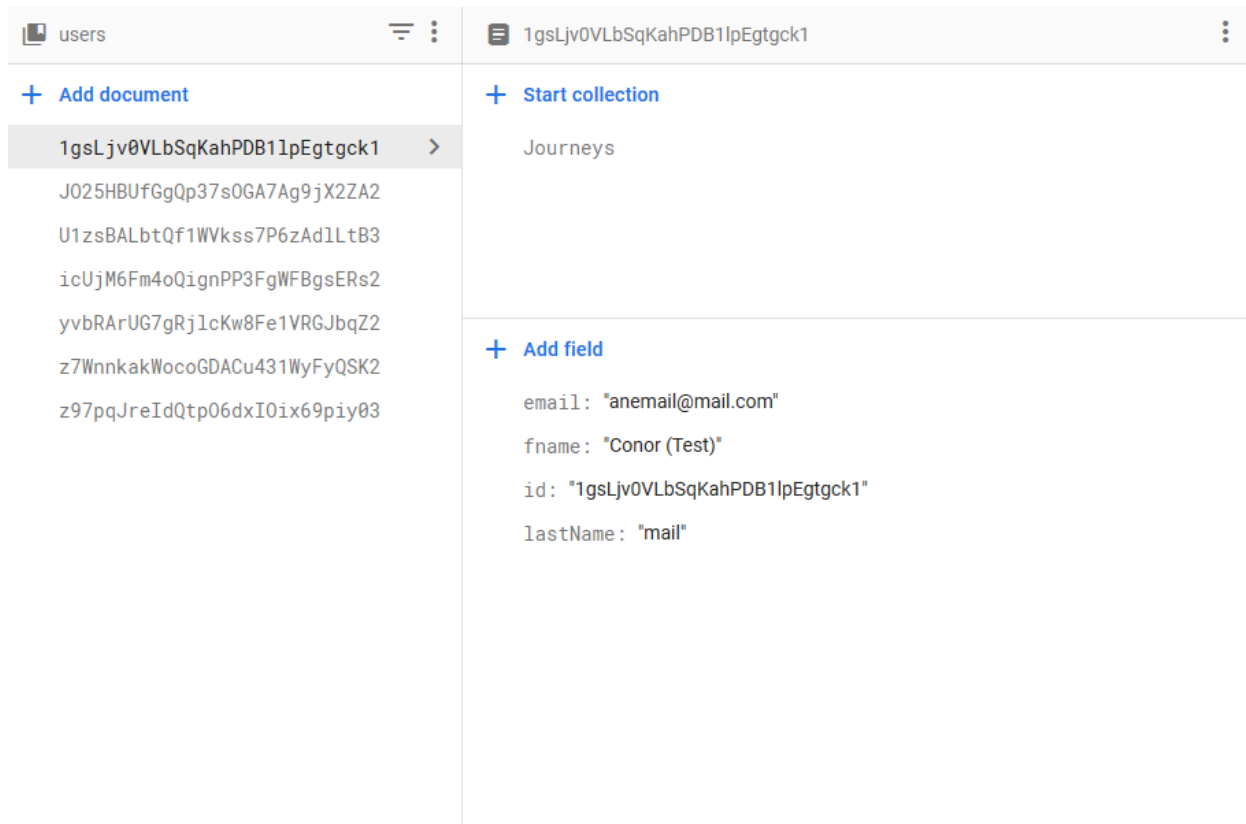
Flexibility on how you wish to store and organise data.

- Expressive data querying
- Realtime updates
- Offline support
- Scalability

The Firebase Cloud Firestore is stored in the Google cloud and is interactable through the Firebase console through a web browser, this console is linked to a Google account.

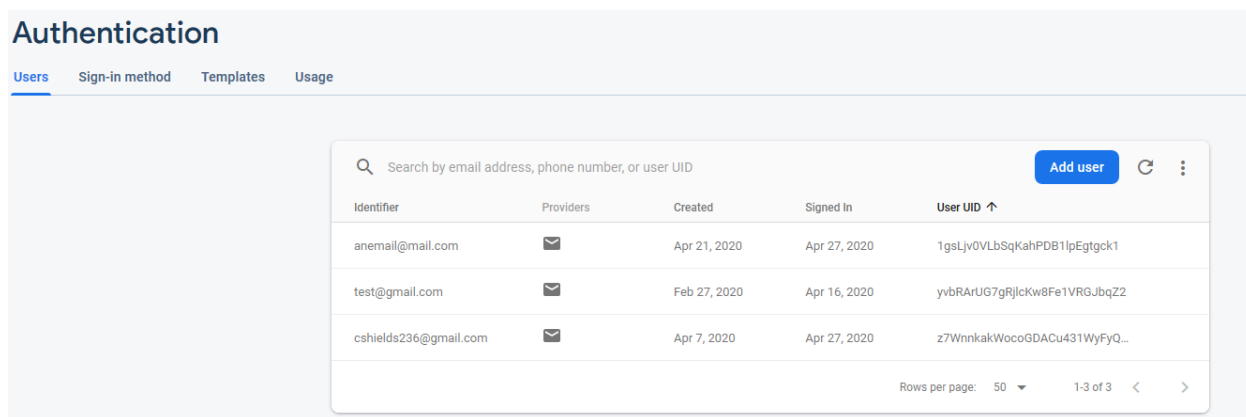


Above we see the main page of the database, on the left is the collection users and, on the right, the specific user document which contains a user object.



The ID for the user object is not auto generated but instead linked to Firebase Authentication. We see this below with the account “anemail@mail.com”, the User ID in the Cloud Firestore is linked to the User UID generated from Firebase Authentication.

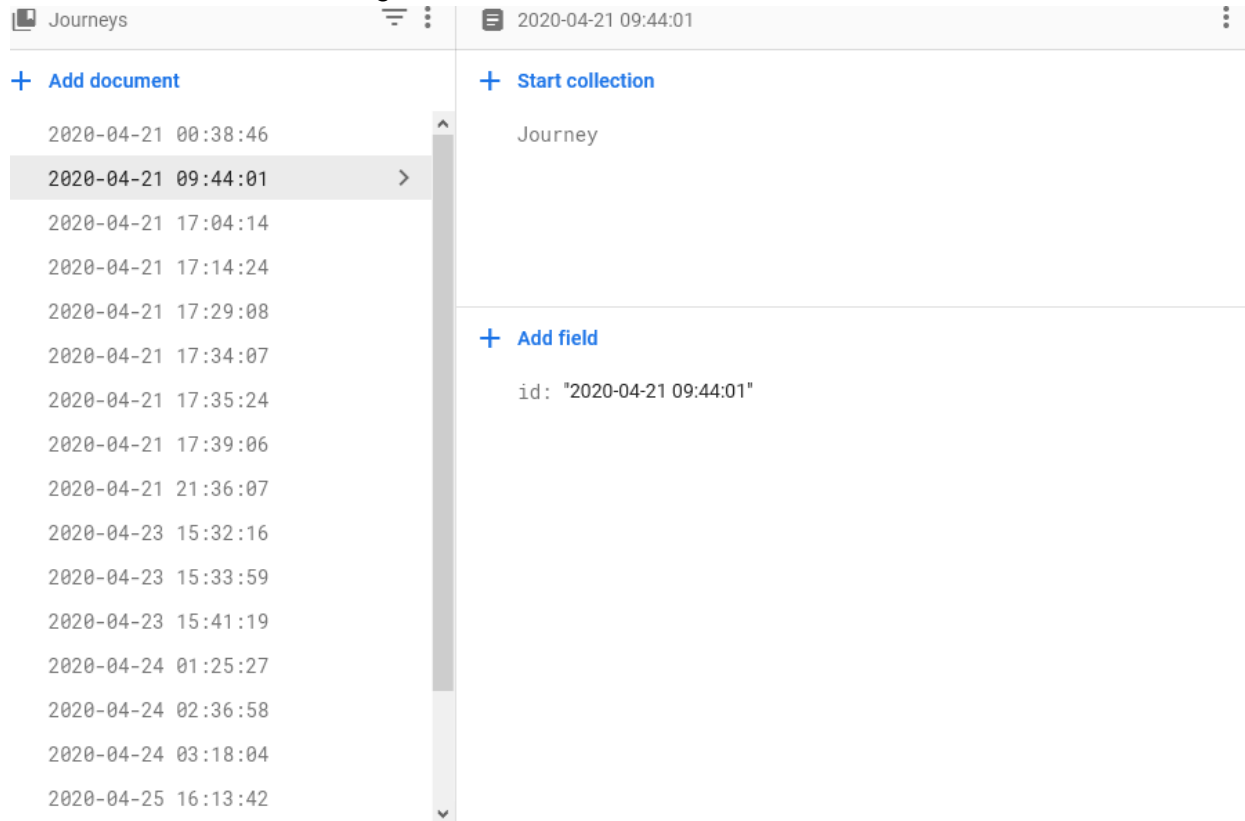
When a user logs into the account for the first time a new document is added to the database.



The User object contains an email address, first name, last name and an ID. I felt there was no need for a username as the email must be unique and that is what is used to register and sign in. The user class also contains a collection Journeys.

1gsLjv0VLbSqKahPDB1lpEgtgck1		Journeys	
+ Start collection		+ Add document	
Journeys >		2020-04-21 00:38:46	
		2020-04-21 09:44:01	
		2020-04-21 17:04:14	
		2020-04-21 17:14:24	
		2020-04-21 17:29:08	
		2020-04-21 17:34:07	
		2020-04-21 17:35:24	
		2020-04-21 17:39:06	
		2020-04-21 21:36:07	
		2020-04-23 15:32:16	
		2020-04-23 15:33:59	
		2020-04-23 15:41:19	
		2020-04-24 01:25:27	
		2020-04-24 02:36:58	
		2020-04-24 03:18:04	
		2020-04-25 16:13:42	

Here we see the Journey collection. I decided to use the beginning time of the journey as the idea for ease of retrieval and ordering.



Here we see a specific Journey object. I went through many iterations of this Database design before eventually settling on this. Here the journey is broken down into multiple document objects. There were a few reasons as to why I came to this solution.

Writing to Database Implementation

The first way I had it designed was writing to the database every time I processed an image; this began to overload the database and lead to crashing and severe slowdown to the application as to get an accurate data sample I needed to be processing multiple face information objects a second.

Next I added a delay and added one second delay to the facial analysis, this got rid of the slowing down and crashing of the application but the loss of graduality meant that I was not able to graph a blink.

The third iteration was saving all the face data in an ArrayList in the Java class and then writing to the database once the "End Journey" Button is pressed. I felt this had sorted my issues as it was adding the information to the Database in one go and allowed for it to be all stored in the one document. But after further testing with trailing the application for longer journeys I found that once the ArrayList reached a certain size the application would run into memory problems and crash.

So, I finally ended up with this solution where the face information objects are stored in an ArrayList in the Java class and then written to the database once the ArrayList hits a certain size. This solution has

eradicated all previously mentioned issues but means each journey must be stored in multiple documents, this has added a slight bit of complexity to queries but I believe is the correct solution for the applications usability and scalability.

2020-04-21 09:44:01

+ Start collection

Journey

+ Add field

id: "2020-04-21 09:44:01"

Journey

+ Add document

```

"2020-04-21 09:44:04"
TBVZydYAGm1iaWHuhfJs
"2020-04-21 09:44:08"
7nVmFSZq8hUZEQe0Y3mk
"2020-04-21 09:44:11"
DS9bWuV6vv6Dqa38Vi6R
"2020-04-21 09:44:14"
vB2wkoydDvhEgw43SYj6
"2020-04-21 09:44:18"
0IDeZ6y3myKn1fGt05E3
"2020-04-21 09:44:21"
B1SzB1CCWxhk7amT27tS
"2020-04-21 09:44:24"
zK0B0h4J19Ippd01FIsl
"2020-04-21 09:44:27"
Cs20jsM13C7rh9pAQ61Q
"2020-04-21 09:44:31"
DpH8MQygyRLaCkEpDJGg
"2020-04-21 09:44:34"
01kpL6ZfjGnz1sfhMVcF
"2020-04-21 09:44:37"
CRHoNxv7oXRBmRqjKtM1
"2020-04-21 09:44:41"
TR60uYhbks7LkrxHS3D
"2020-04-21 09:44:44"
GY1A9I7jiw1XTuq2PY4N
"2020-04-21 09:44:47"
4bIh1ioU3INEKb18PCbV
"2020-04-21 09:44:50"
4g8urxYtZHggqA5RJ181
"2020-04-21 09:44:54"
VQ7M1P5rRWp1hkNLFaT0

```

Here we see the individual Journey class which has a nested collection of journeyInformations which hold the facial data given by the Face Detector. The number is the collection ID, blink is a number value to signify the amount of blinks at that point of the journey, leftEye and rightEye signify the probability that the eye is open with 1 being completely open as according to the Face Detector, it also contains a references to the email of the user and the specific time at which it was recorded.

The screenshot displays a MongoDB document in a web interface. The document is titled "Journey" and has a key "TBVZydYAGmliaWHuhfJs". It contains a nested array named "journeyInformations" with two elements, indexed 0 and 1.

Document Structure:

```

{
  "TBVZydYAGmliaWHuhfJs": {
    "2020-04-21 09:44:04": "TBVZydYAGmliaWHuhfJs",
    "2020-04-21 09:44:08": "7nVmFSZq8hUZEQe0Y3mk",
    "2020-04-21 09:44:11": "DS9bWuV6vv6Dqa38Vi6R",
    "2020-04-21 09:44:14": "vB2wkoydDvhEgw43SYj6",
    "2020-04-21 09:44:18": "0IDEzGy3myKn1fGtGSE3",
    "2020-04-21 09:44:21": "BIS2B1CCWxhk7amT27tS",
    "2020-04-21 09:44:24": "zK0B0h4Jl9Ipdd01FIsl",
    "2020-04-21 09:44:27": "Cs20jsM13C7rh9pAQ6lQ",
    "2020-04-21 09:44:31": "DpH8MQgyRLaCkEpDJGg",
    "2020-04-21 09:44:34": "0lkl6ZfjGnz1sfhMVcF",
    "2020-04-21 09:44:37": "CRHoNxv7oXRBmRqjKtMl",
    "2020-04-21 09:44:41": "TR60uYhbks7LkrxHS3D",
    "2020-04-21 09:44:44": "GYiA9I7jiw1XTuq2PY4N",
    "2020-04-21 09:44:47": "4bIhIioU3INEKb18PCbV",
    "2020-04-21 09:44:50": "4g8urxYtZHggA5RJl8l",
    "2020-04-21 09:44:54": "VQ7MlP5rWp1hkNLfaT0"
  }
}

```

journeyInformations Array:

- Index 0:**
 - blink: 0
 - blinking: false
 - leftEye: 0.9858568906784058
 - name: "anemail@mail.com"
 - rightEye: 0.9915161728858948
 - time: "2020-04-21 09:44:01"
- Index 1:**
 - blink: 0
 - blinking: false
 - leftEye: 0.9287917613983154
 - name: "anemail@mail.com"
 - rightEye: 0.9884875416755676
 - time: "2020-04-21 09:44:01"

Querying Database

Writing To

Querying NoSQL databases is quite a bit different from querying an SQL database which I would have been a lot more familiar with before embarking on this project.

```
// [START create_user_with_email]
mAuth.createUserWithEmailAndPassword(email, pw)
    .addOnCompleteListener(activity, new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()) {

                Intent i = new Intent( packageContext RegisterActivity.this, DashboardActivity.class);
                startActivity(i);

                // Get reference to database
                FirebaseFirestore db = FirebaseFirestore.getInstance();

                Log.d( tag: "", msg: "createUserWithEmail:success");
                FirebaseUser u = mAuth.getCurrentUser();
                // User Details gotten from textfields above
                User ul = new User();
                ul.setId(u.getId());
                ul.setFname(fname);
                ul.setLastName(lname);
                ul.setEmail(email);
                // Add a new document with ID from the Authenticator
                db.collection( collectionPath: "users").document(u.getId())
                    .set(ul).addOnSuccessListener((OnSuccessListener) (aVoid) -> {
                        Log.d(TAG, msg: "DocumentSnapshot added with ID: ");
                    }).addOnFailureListener((e) -> {
                        Log.v(TAG, msg: "Error adding document", e);
                    });

                Log.d(TAG, msg: "onComplete: " + "created");

            } else {
                Log.v( tag: "", msg: "createUserWithEmail:failure", task.getException());
                // Show user error message
                Toast.makeText( context: RegisterActivity.this, text: "Authentication failed." + task.getException().getMessage(),
                    Toast.LENGTH_LONG).show();
            }
        }
    });
}
```

Here we see a snippet of code that creates a user Document in the database if the Firebase Authentication method to create a new account using an email and password is a success. As you can see you must set the collection path and then add the data. This code then interacts with the Firebase Cloud Firestore database and can be tracked through the use of Event Listeners, like I used above with `onSuccessListener` to log a success to the console and in the `onFailureListener`.

Reading From

Reading from the database is quite similar to writing to it. Here we are getting all the journeys for the current logged in user. You once again get the current user from the Authenticator, get an instance of the database and define your chosen database path. In this snippet we are getting all the Journey documents from the Journey collection and sorting them by the object variable “time” in descending order.

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
FirebaseFirestore db = FirebaseFirestore.getInstance();
CollectionReference ref = db.collection( collectionPath: "users").document(user.getId()).collection( collectionPath: "Journeys");

//Getting all journeys from database
ref.orderBy( field: "time", Query.Direction.DESENDING).get().addOnCompleteListener((task) -> {
    if (task.isSuccessful()) {
        for (QueryDocumentSnapshot doc : task.getResult()) {
```

Shared Preferences

As well as using Cloud Firestore for storage in my application, I also used shared preferences. Shared Preferences is a way to store and retrieve small amounts of data within the application using key value pairs. I used shared preferences for storing the users emergency contact details as well as storing and retrieving the users most recent location for the coffee suggestion part of the activity.

Implementation

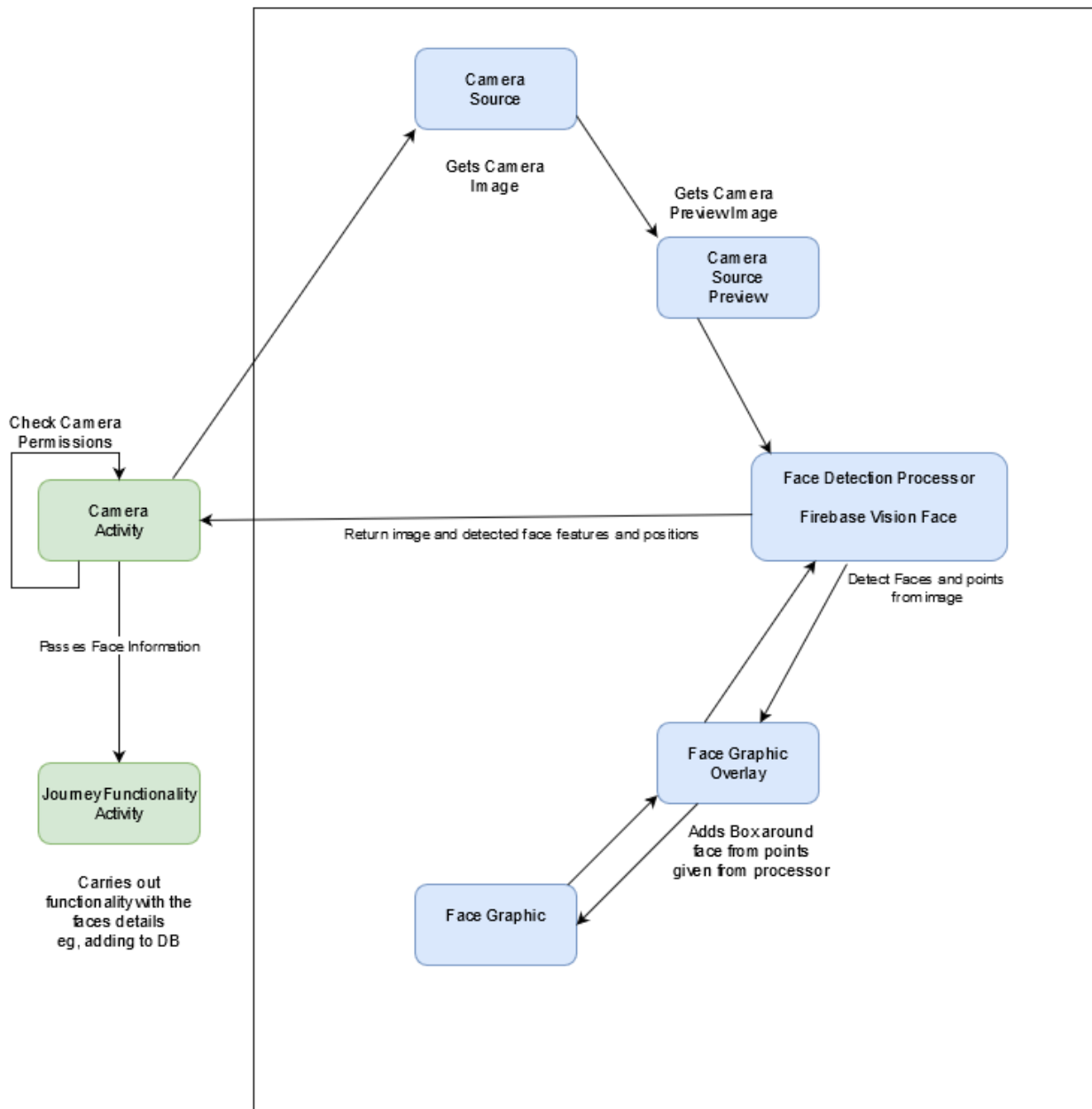
While the application is the part that the user sees it is only a small portion of the work which I put into the development of this project. The application was built using data and constantly iterated upon using the data collected. The creation of the app was broken down into multiple steps.

Implementing Real Time Facial Analysis

In the beginning I developed a straightforward application using the Firebase ML Face Detector in order to collect data in order to analyse the data for the designing of the application.

This first part of the application building process was not easy as I had never worked with using a camera or camera kits before. At first, I had developed an application which just took pictures, analyses these and returned the data. This application was obviously not fit for purpose as in order to collect enough data to find patterns I would need to be monitoring and processing faces in Real Time, the implementation of this system was quite a learning curve for me.

Face Detection System



We can see above the solution I ended up implementing. Firstly, we have the Camera Activity class, this class first checks permission to use the devices camera, if this is granted it creates a Camera Source. It is fine to just use the Camera Source if you are analysing still images, for my application I needed to be analysing continuously, therefore I needed to implement the camera source preview class.

The Camera Source Preview allows us to access and add overlays to the preview of the device's camera, this allows the implementation of the real time facial detection. The camera

source preview feeds image data in the form of a bitmap to the Face Detection Processor. This face detection processor processes these image bitmaps and returns a FirebaseVision Image. These Images are then passed through the chosen Firebase MLkit Machine Learning algorithm, in my case the face detector algorithm. After the image is passed through this it returns a FireabaseVisionFace object, this object is composed of numerous attributes in relation to pictures facial features, it is from this we get the raw facial data such as eyes open probability.

This data is then fed to a Face Graphic Overlay class, which extends the Graphic Overlay class. This Face Graphic Overlay class takes the points from the FirebaseVisionFace object and draws a bounding box around the face and at the points of the two eyes with the number for the probability that they are open.

The face Object is also passed back to the Camera Activity which calls on the Journey Functionality Activity where the main functionality of the Android application takes place.

Data Analysis

After collecting this data, I began working on the data analysis end of the application which involved graphing the collected data in order to define what a blink is. The library which I am using to analyse the face data only provides a figure which indicates the probability that a user's eye is open, there is no method which returns whether a person is blinking. In order to design a method in the application to define a blink, I carried out exploratory analysis on the data. From this exploratory analysis I came to the conclusion that any figure below 20% open would be classified as a blink in the application. This figure is not perfect as I am working with the limitations of the face detection library and in that case some blinks might get missed in the running of the application.

I trailed on many different thresholds until I concluded that 20% was the most accurate threshold in order to find a blink. After trailing the program with the threshold comprising different values ranging from 10% to 40%.

After carrying out these different tests I came to the conclusion that the lower end of the spectrum was not picking up blinks as they were happening as it was not quick enough, despite the fact it could be processing multiple images per second.

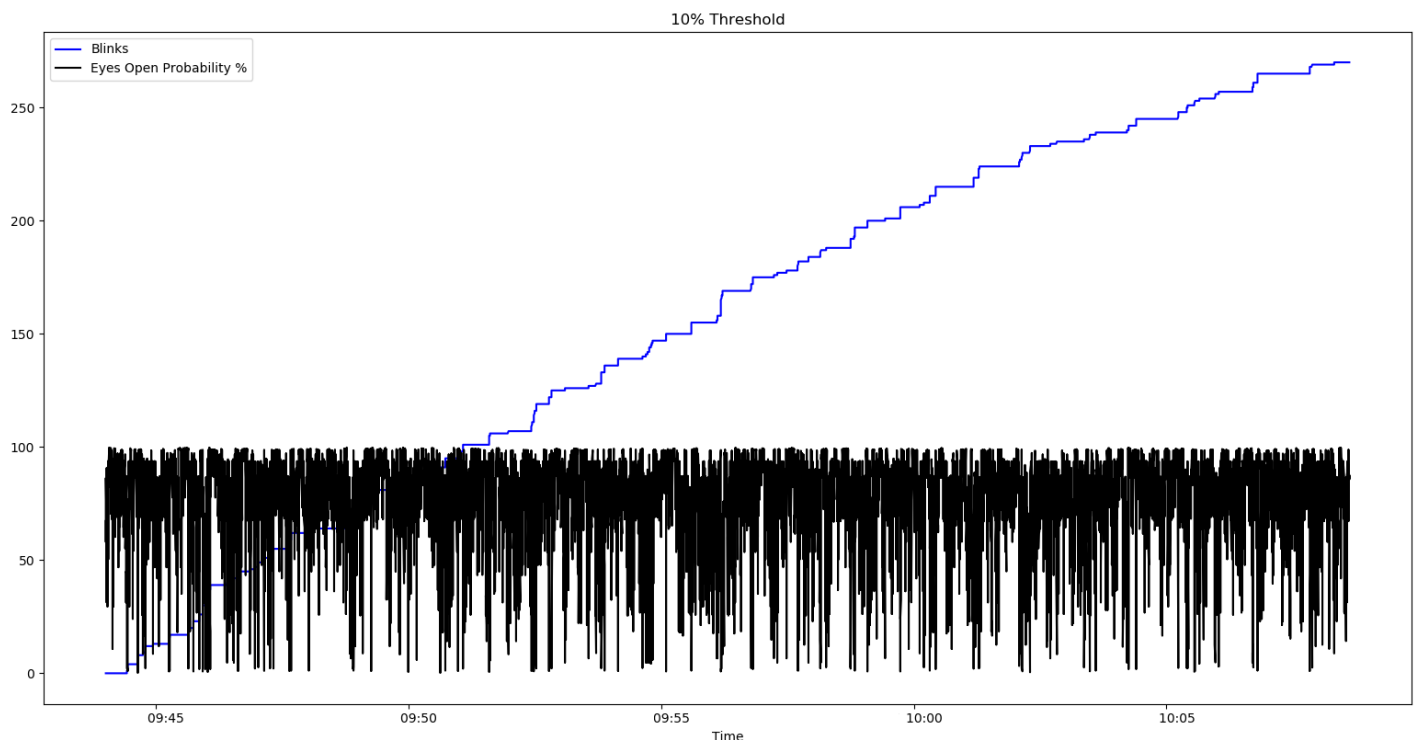
Whereas on the other end of the scale, when it gets up to a 40% threshold, we can see far more blinks being recorded than actually occurring, this is particularly the case I believe in brighter conditions where I found squints being interpreted as blinks which painted an

unrealistic picture of what was happening and skewed the data showing far more blinks than reality.

This result came from a long period of trailing and testing. This end of the project took up the bulk of my time as it was all new to me, I had never carried out any sort of data analysis before starting this progress. While getting and graphing the results was interesting, I struggled to know what to do once I had the data. I eventually came to the culmination of the data analysis; in that I am confident that the application is catching enough blinks for us to analyse and find abnormalities.

I felt it was very important to use data to back the decisions I made in the designing and implementation of this application. Although not perfect, I endeavored to research and have a reason for every piece of functionality in the application.

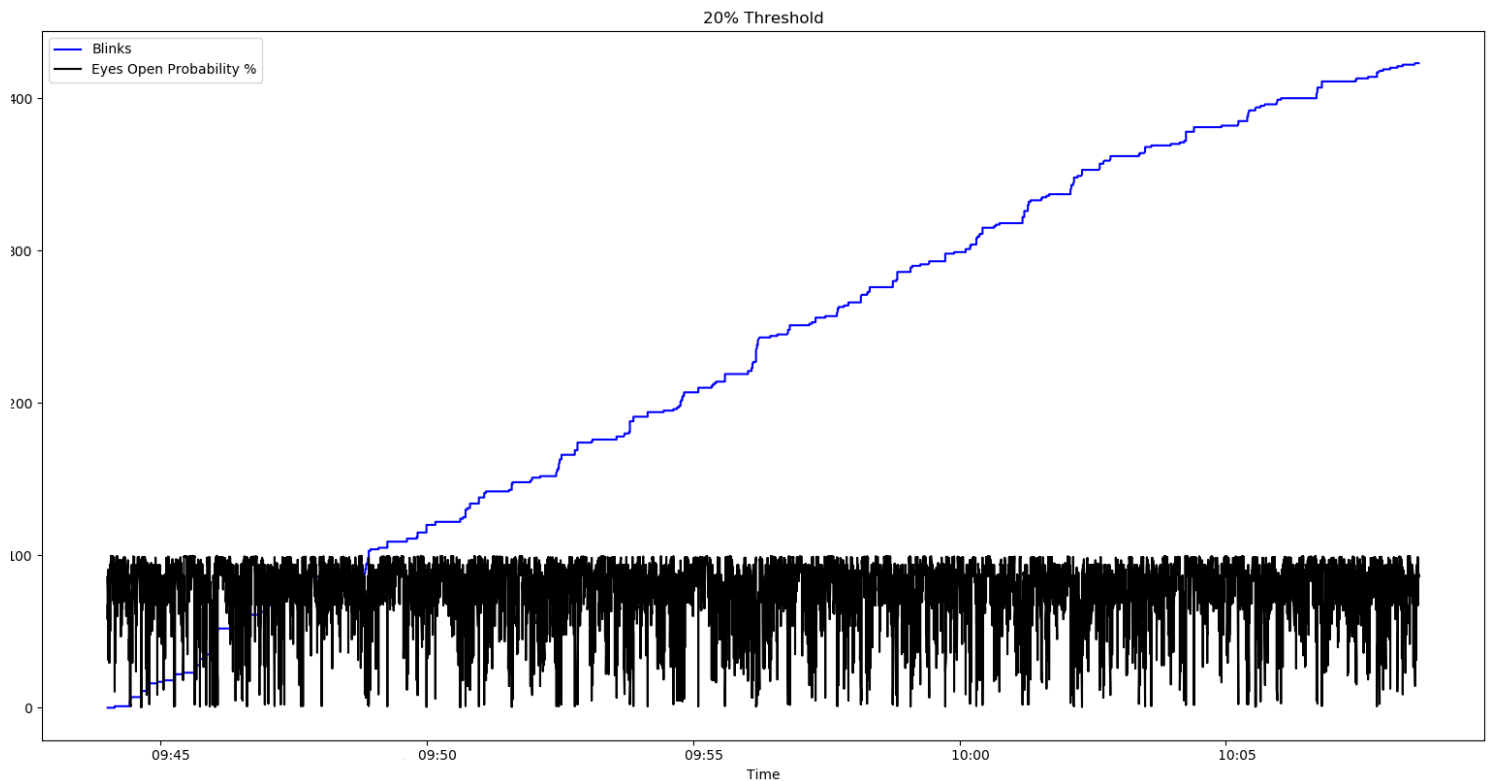
I base the majority of my data analysis on one near hour long journey which I recorded. The plan was to have many more journeys to analyse but sadly because of the lockdown sanctions I was unable to drive long distances that were needed to gain more accurate data. Below I will show some snippets



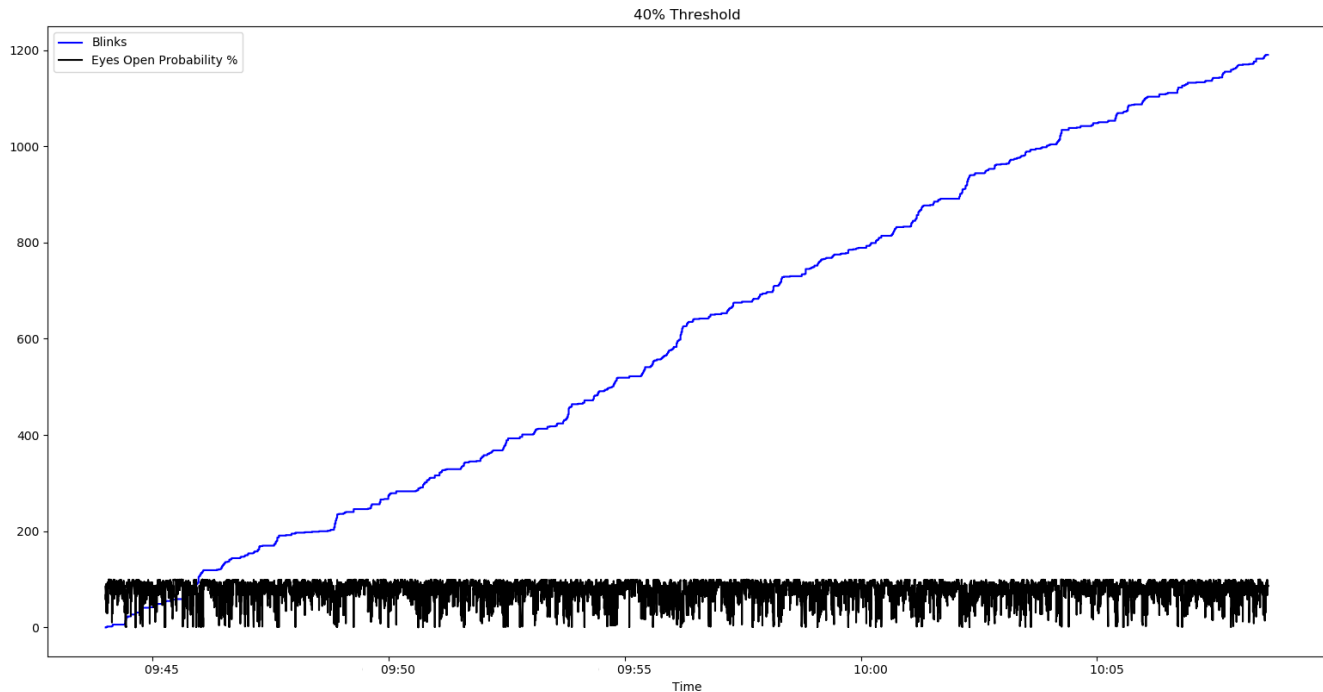
from this journey.

Here we see a graph of a 20-minute snippet of the journey recording blinking with a blinking threshold, where any eyes recorded with below a 10% probability of being open were counted as a blink. From analysing this data by watching back a screen recording of the journey I found that this threshold was

missing out on blinks. I concluded that the 10% probability was too low as some blinks were happening so fast that the library was never registering that the users' eyes were fully closed.



The 20% threshold was the most accurate I found. Here the blinks corresponded very closely with the video of the journey with only a negligible amount being missed. I concluded that 20% was the most accurate and developed my application using that number in the implementation of the blink tracker.



Above we see the 40% threshold. I found this to be far too sensitive and resulting in a far higher number of blinks than actually occurred. From this classification it estimates that the user is averaging nearly 50 blinks a minute. I believe this is down to a few factors such as light squints being interpreted as blinks and any loss of vision on the eye being counted as a blink.

Overall, I found this data analysis to be hugely beneficial in the creation of the application. It allowed me to get to know the data and understand the intricacies the application. It was my first time trailing any sort of data analysis aspects in application and it is something that I would like to continue with and massively improve my skills in. I believe implementing a data led development method into my application has helped improve the accuracy of the applications monitoring system as I was not just developing of assumptions and for the most part tried to prove why an implementation was necessary.

Warning Implementation

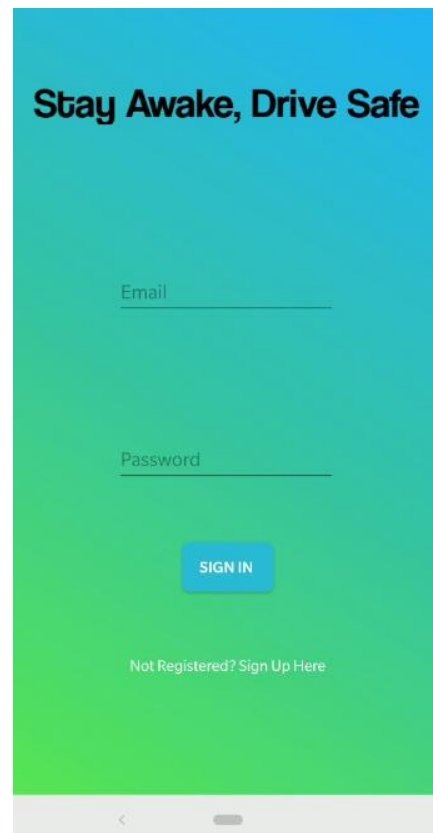
For the implementation of the warning system I compare the number of blinks from the current minute to the average number of previous blinks, if this number is more than 20% higher. I was originally sending a warning any time the current number was higher than the average but found this to be an over simplified and not fit for purpose solution.

Application Implementation

User Login

Application users will open on the login screen where they are prompted to log in using their email and password or create a new account with the register feature.

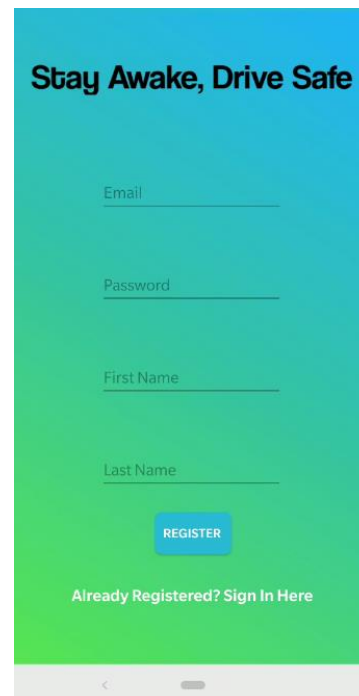
The login and authentication are powered by Firebase Authenticator and create a new session which can be used to retrieve user data using the `getCurrentUser()` method.



Registration

If users are not registered, they will be prompted to register using an email and password

Registering automatically logs the user into the application and creates a document for them in the database.



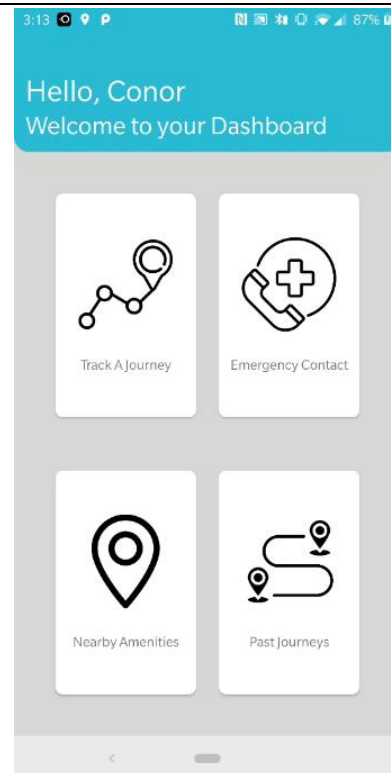
The registration screen features a blue header with the text "Stay Awake, Drive Safe". Below the header are four input fields: "Email", "Password", "First Name", and "Last Name". A blue "REGISTER" button is positioned below the input fields. At the bottom, there is a link that says "Already Registered? Sign In Here". The background of the form is a gradient from blue at the top to green at the bottom.

Dashboard

If a user's login or registration is successful, they are brought to this screen.

It will be customized to each individual user displaying their name.

This dashboard shows them all the main features of the application that they may wish to use



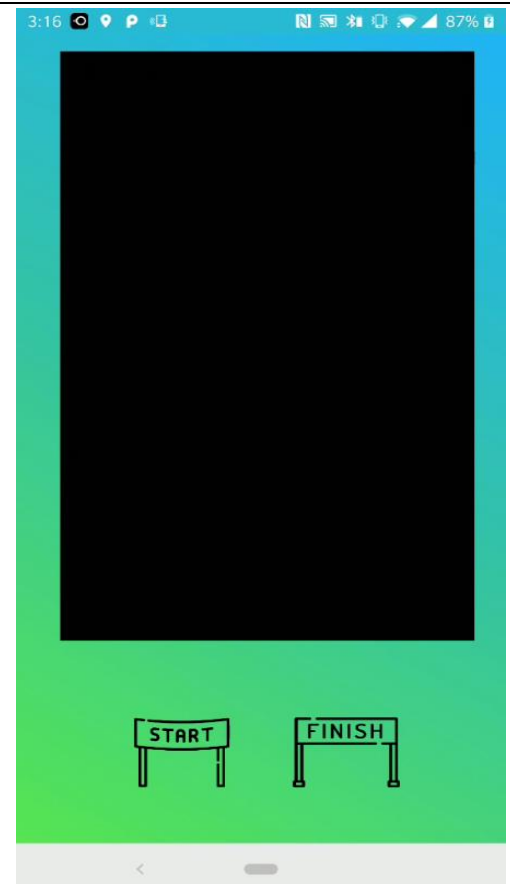
Track A Journey

This is the core functionality of the application.

I implemented a simple UI to help with the user experience. There are just two simple options Start and Finish.

When start is pressed a new Camera Source Preview is created and displayed and the face tracking begins.

The finish button cannot be pressed before a journey has begun and the start button can only be pressed once.



Here we see the raw output from the library displayed on screen. The face detector kit will return a number between 0 and 1 which signifies the confidence the algorithm has that an eye is open.

In this image we can see that the eyes are fully open so return a perfect score.

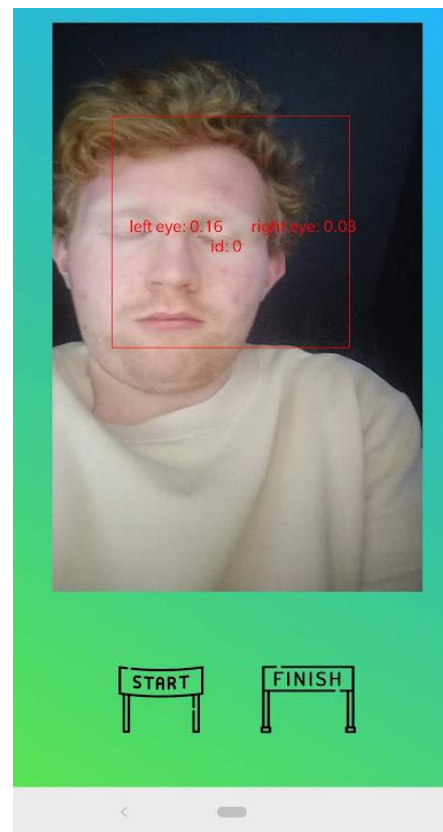
Every time a face is processed the face object is added to an ArrayList and the box is re-drawn to follow the faces movements.



Here is what is returned when the eyes are closed during a blink.

This information is also added to the same ArrayList. If the algorithm finds that the user is blinking a minute blink counter and a total blink counter is incremented.

The minute blink counter is used to compare the current minutes blinks to the average of the previous and the total blinks just keeps track of all blinks that take place over the course of a journey.



The warning system in the application takes the form of a voice to text system which reads out a warning to the user. This warning will increase over time if warnings persist. For example, if a user is warned 10 times in a single journey they will be told to consider resting and a message will be sent to the user's emergency contact asking them to check in on the driver.

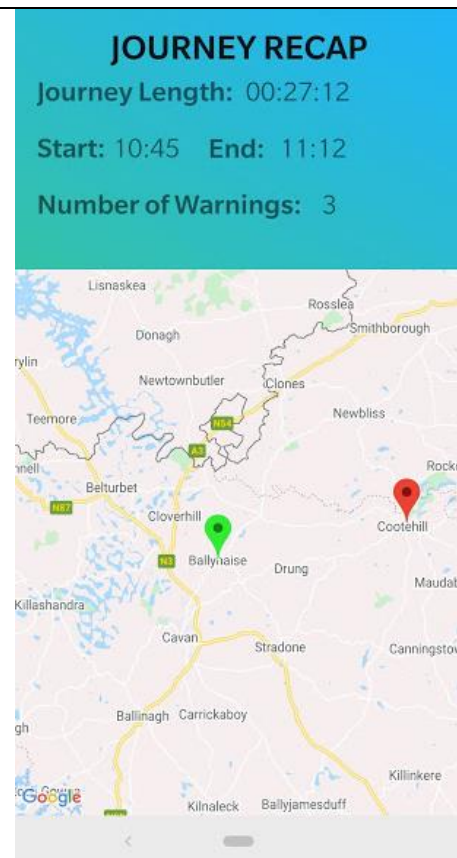
A warning is sent out to a user if their blinking rate for a minute is 20% higher than the average blinks per minute for the journey. I believe this 20% increase displays an abnormal increase in blinking. Continued increases in blinking has been proven to be a visual sign of fatigue and loss of concentration.

Once the finish journey has been pressed the users are brought to the journey recap screen.

This screen provides the user with information about the journey that they have just taken such as the time and the number of warnings which occurred throughout the journey.

Also, when the "Finish" journey button is clicked it writes the remaining details of the journey to the database and passes information to the journey recap screen using Intents.

The Google Maps fragment contains two markers, the green signifying where the journey began and the red showing the end of the journey.

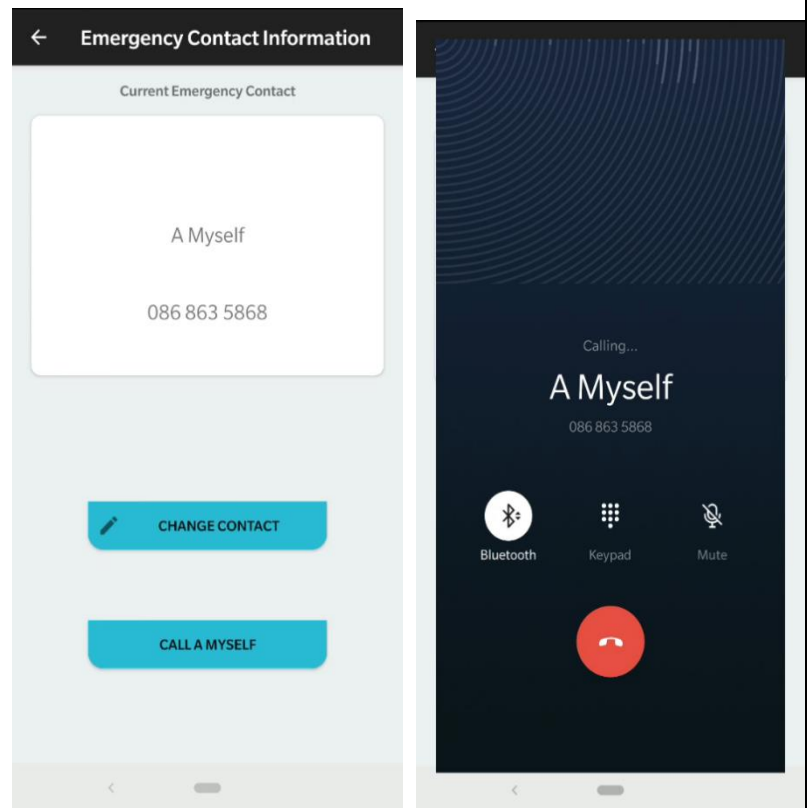


Emergency Contact Store

Here we see the Emergency contact page. Here a user can select a user a member of their contacts to be their designated emergency contact.

This is currently a prototype application and if used in the business case of working in collaboration this process could be done behind the scenes and instead of having a contact from the users phone the warning message would be sent to the insurance company.

The call button can be used as an emergency call function and allows the user to call the contact without leaving the application.



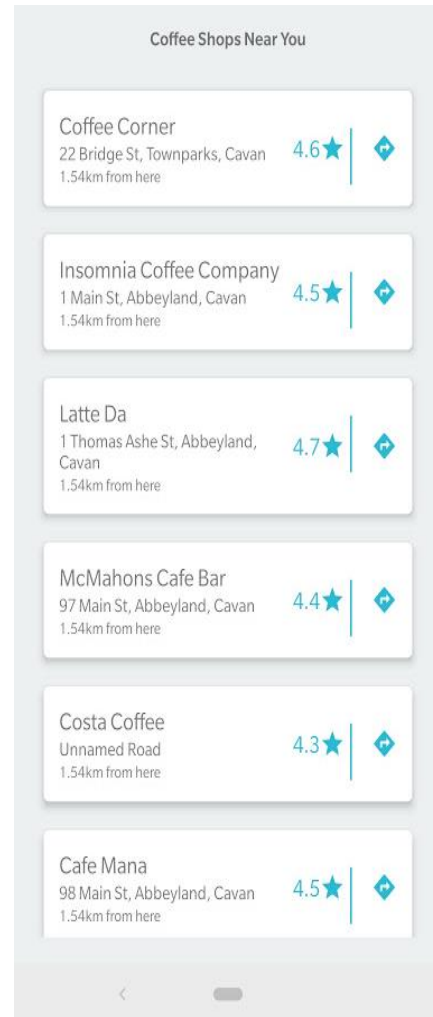
Nearby Coffee Shops Feature

Here we see the nearby coffee shops feature. This is powered by the Google places API. I input the user's current position into a JSON request with the variable "coffee shop" added.

This then returns a JSON object which then needs to be parsed, I did this by creating a places class which parse the information and stores them as String variables. After this these stored variables are used to populate the recycler view.

I feel there is a nice straight forward UI that displays all the details of the coffee shops simply.

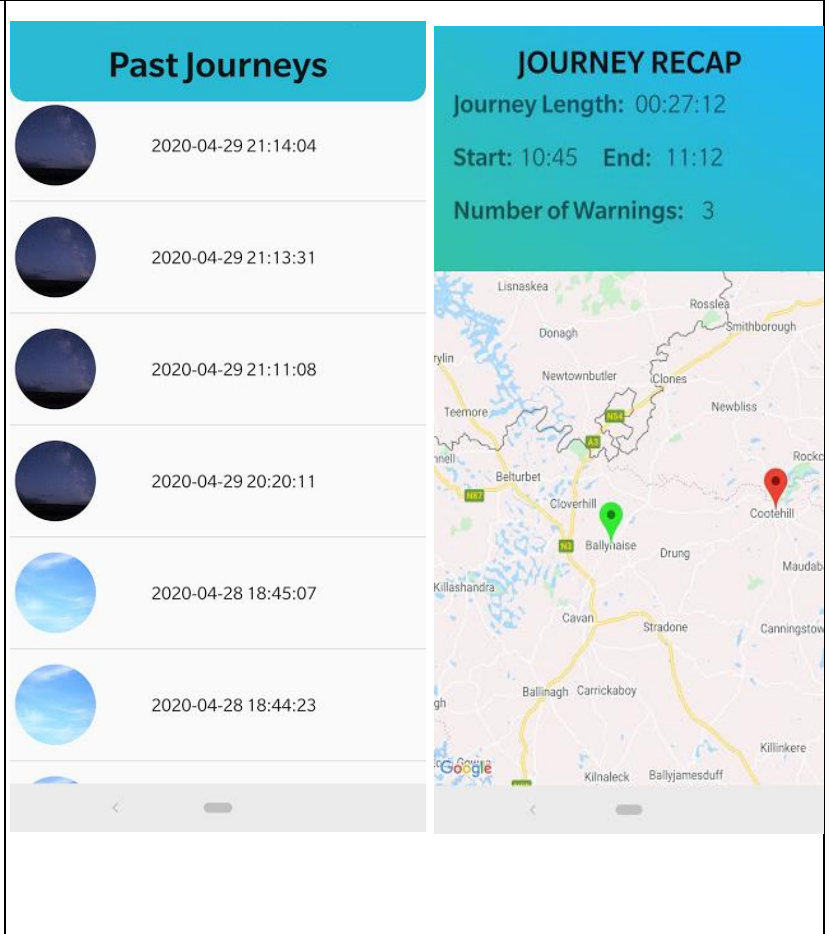
There is an onClick method for this recycler view that opens the selected coffee shop in the Google Maps app to display to the use



Past Journeys

This is the past journeys component of the application. Here all the current logged in users' journeys are retrieved from the database and placed on to this recycler view page.

Upon clicking on a specific journey, that journey's ID is then passed using intents to a page that gets populated with the journey's information.



Future of the Project

I believe there is massive scope to improve on and add functionality to this project if I was to peruse it in the future. At the moment the application features only a fairly straightforward algorithm for detection tiredness, this could be worked on and improved massively with additional data to use.

Another area which I believe could be implemented is linking the application with smart ware items such as Fitbit or Apple Watches in order to add a more in nuanced algorithmic way to detect tiredness. These products are constantly collecting information on their users such as their heart rate and the amount of hours sleep, they had the night before. These are all indicators of tiredness and as such could be taken into account by the application.

In the future I would also like to test this application with many more real-world examples with people driving. Sadly, the current Covid 19 situation hindered this aspect of the project and I had to work with the data I had already collected which was not ideal.

Conclusion

Ultimately, I am quite happy with my application. I feel it was quite a challenging topic and one in which I learnt a lot. I found working on it to be very interesting and enjoyed the process of develop and analysis and the marrying of the two. I believe it is tackling a real challenge and hope if I was to develop further could lead to

Looking back, I feel there are many areas which could be improves improved upon. This was my first time attempting to develop a project which incorporated data analysis and I was mainly learning from reading online articles as well learning from my own mistakes, which while an effective way of learning is quite time consuming and leads to achieving little on some occasions.

I feel the application does have a place in the market, I believe that if it can lead to even one less road traffic collision then it was a worthwhile adventure.

There are limitations to the application I believe. As it is reliant on the MLkit face detector it is not a perfect system and can lead to some blinks being missed. The data being collected is a bit limited as I cannot compare driving conditions which could lead to an increase in blinking. As it is reliant on the phones camera the results may differ from phone to phone as there are hardware differences.

But overall, I feel the app that I created is fit for purpose and has allowed me to develop many skills in terms of software development, data collection and analysis and project management. I believe the work I put in this year in both this project and the modules outside will stand to me greatly as I move into the workplace.

#	Test name	Test Data	Test Conditions/ steps	Expected Result	Result
1	Check user sign in functionality	email: anemail@mail.com Password: 123456	Launch the app. Enter email. Enter password. Click 'Sign in	Login successful Dashboard screen appears.	Pass
2	Check User registration	Email: c162523@mytudublin.com Password: pass12 First name: Conor Last Name: Shields	Enter email. Enter Password. Enter First Name. Enter last name.	User saved to firebase authenticator. New user document added to DB Dashboard screen appears.	Pass
3	Dashboard should show users name in the header after login	Email: c162523@mytudublin.com Password: pass12	Log in. View Dashboard.	Dashboard should show the message "Hello, Conor Welcome to your Dashboard"	Pass
3	Track Journey activity is launched from dashboard		Press on "Track a Journey"	Track Journey activity should launch	Pass
4	Emergency Contact activity is launched from dashboard		Press on "Emergency Contact"	Emergency Contact activity should launch	Pass
5	Nearby Coffee shops activity is launched from dashboard		Press on "Nearby Amenities"	Nearby Coffee shops activity should launch	Pass
6	Track Journey activity is launched from dashboard		Press on "Track a Journey"	Track Journey activity should launch	Pass
7	Past Journeys activity is launched from dashboard		Press on Past Journeys	Past Journeys activity should launch	Pass
8	Start a journey		Press on Start Journey button in journey tracking activity	Camera should be shown on screen. Box drawn around users face with data	Pass

9	Finish a Journey.		Press on the Finish Journey button.	Camera should stop recording Journey recap activity should launch with details of the journey	Pass
10	Show signs of tiredness in journey		Current minutes blink numbers 20 > than the average so warning is given.	Text to speech should play telling user they are showing signs of fatigue	Pass
11	Continued signs of fatigue		More than 10 warnings in a single journey	Text to speech should play telling user to stop journey and SMS Sent to emergency contact	Pass
12	Marker on map in journey recap		In journey recap page the map is updated with the locations of where the journey started and finished	Two markers shown on map for start and finish	Pass
13	Journey Recap is updated with most recent details	Finish a 2-minute journey	Check that journey length is accurate. Check that start time and end time are accurate.	Journey length set to 2:00:00 Time is set to accurate times	Pass
14	Check coffee suggestion page is populated			When user clicks on the coffee suggestion option view should be populated with coffee shops near current location	Pass

15	Maps open from selecting coffee shop	Select coffee shop from list		When coffee shop is clicked maps open	Pass
16	Past Journeys populated			When past journeys button is clicked page is populated with current users' previous journeys.	Pass

References

1. Rsa.ie. 2020. *RSA.ie - Driver Fatigue*. [online] Available at: <<https://www.rsa.ie/RSA/Road-Safety/Campaigns/Current-road-safety-campaigns/Drunk-With-Tiredness/>> [Accessed 28 April 2020].
2. Firebase. 2020. *Cloud Firestore | Firebase*. [online] Available at: <<https://firebase.google.com/docs/firestore>> [Accessed 28 April 2020].
3. Firebase. 2020. *ML Kit For Firebase*. [online] Available at: <<https://firebase.google.com/docs/ml-kit>> [Accessed 29 April 2020].
4. Google Developers. 2020. *Overview | Places API | Google Developers*. [online] Available at: <<https://developers.google.com/places/web-service/intro>> [Accessed 30 April 2020].
5. Google Developers. 2020. *Overview | Maps SDK For Android | Google Developers*. [online] Available at: <<https://developers.google.com/maps/documentation/android-sdk/intro>> [Accessed 31 April 2020].
6. Android Studio Documentation [online] Available at: <<https://developer.android.com/docs>> [Accessed 1 May 2020].