Final Year Project:

Clever Carpooling

Technical Manual

16th May 2020

Authors:

Shaun Carey *16450454*

Nigel Guven *14493422*

Supervisor:

Ray Walshe

Table of Contents

1. Introduction

1.1 Overview

Clever Carpooling is a ridesharing planner which aims to make transport options more widely available for the public. The application primarily focuses on the personal automobile market such as cars and motorbikes but also may be extendible to private vans, buses and more. Users may register as a driver or a passenger and be able to post a route of their journey for a certain date and time. Other users may be able to view that post and then contact the owner of that post, with an option to pick up or request a lift. The users can communicate within the application messaging system where they can send text messages, images, and PDF documents. There are also public forums which any user can set up with ease so that multiple users can organise a carpool.

1.2 Motivation and Research

The application is designed for the Android mobile device market which occupies 90% of handheld devices as of 2020. There was also a motivation to build the application on iOS but circumstances outside of our control which relate to the Covid-19 pandemic of 2020 forbade us from accessing tools that we required on campus and for which we had no access externally. Our understanding was that there are not enough transport options for people in Ireland and the usage of Uber is restricted. Other applications such as MyTaxi can be expensive and buses and trains are efficient but not adequate in supply or are restricted to certain routes.

We noticed that personal vehicles are nearly empty for people travelling to and from work and college. There is usually one person in each car at any one time. Our aim is to reduce the number of cars on the road by giving users the tools to fill up their cars with passengers and they may charge for ridesharing at their own will. A further effect from the usage of our application and a personal motivation is to lessen the carbon footprint of the daily users of our application.

1.3 Future Work

As of the 8th of May 2020, we have completed all functionality set out by our manifesto in the Functional Specification which can also be viewed in the same repository as this document. Further plans may be extended to this application given the ability to bring this application to market. The Google Play Store is the most likely place to platform this application. Currently, the team is using Firebase Blaze plan for extensive application features but may reconsider this given financial circumstances.

Future features may include the implementation of a payment system for users to donate as much money as they would like to a driver, and this would require a revision of Ethics and a secure system for payments. There is also the greater possibility of adding weather features to the application so that users may know of what the weather will be like for a certain day.

1.4 Glossary

Firebase – Firebase is a mobile and web application platform with storage, development, mobile machine learning and Function-as-a-Service features.

Firebase Blaze – Blaze is an upgraded Software-as-a-Service version of Firebase which provides access to the latest features of Firebase technology.

API Level – API Level is the integer value that identifies the framework AAPI offered by a version of the Android Platform.

Logcat – Logcat is a command line tool which dumps a log of system messages, including stack traces. Logs can be viewed from within Android Studio.

ADB – Android Debug Bridge is a versatile command-line tool which lets you communicate with a device and provides access to a UNIX shell.

IDE – An integrated development environment is a software application which provides facilities for development of software. Android Studio is an IDE for building mobile application for Android OS.

Volley – Volley is a HTTP library developed by Google to transmit data over networks. It makes API calls and networking simpler to implement.

JSoup – A Java library purpose-built for interacting with HTML. It provides code for cleaning HTML to retrieve the raw text data inside tags.

Glide – Glide is an image loader library for Android developed by Bumptech which provides image loading and caching.

JUnit – JUnit is a testing framework for the Java programming language. It comes installed on Android Studio and is useful for creating Integration and Unit tests.

Mockito – Mockito is an open source testing framework for creating unit tests. Mockito differs from JUnit in that certain variables can be provided in the test cases.

Recyclerview – Recyclerviews are flexible lists for displaying and sorting large datasets.

Firestore – Cloud Firestore is a flexible, scalable database for development using Firebase and Google Cloud Platform.

Realtime Database – Realtime Database is a NoSQL database and utilises JSON to store objects.

ML Kit – ML Kit is a mobile SDK which provides various machine learning functions for Firebase.

OCR – Optical Character Recognition is the recognition of text from images.

Node.js – Node.js is an open-source runtime environment which executes Javascript code based on executed events.

Activity – An activity is a single unit containing a set of actions and a user interface with which a user may interact with.

Adapter – An adapter object acts as abridge between an AdapterView and the underlying data for that view.

Fragment – A fragment represents a distinct behaviour or a subset of a user interface and are combinable, and recyclable which makes them more beneficial than activities.

Levenshtein Distance – The Levenshtein Distance is an algorithm for measuring the difference between two sequences of characters. It is measured by calculating the insertions, deletions, and substitutions between two sequences.

2. Installation and Prerequisites

2.1 System Specifications

Developed on *Huawei PRA-LX1*, *LG-X Venture*, *Nexus 5X*, *Nexus 6* for full coverage of screen sizes and Android OS versions. Issues encountered with this application on a phone not relating to one aforementioned may be reported to the development team at the following email: support@clevercarpooling.firebaseapp.com.

Mobile Platform Support: Android Mobile Devices only.

Android OS Minimum API/SDK Version: API 21/Android 4.4. KitKat

Android OS Maximum API/SDK Version: API 29/Android 10

Supports 94.1% of current Android Devices as of 10th May 2020. Phones which utilise an operating system earlier than API level 21 may experience unusual activity within the application. You can check the API level of your mobile device by clicking on Settings > About Phone > Android Version.

Android Code Version: 1.0

External Requirements:

This application makes use of third-party software which may result in unusual application behaviour if updated outside of the application:

- Firebase {Auth, Analytics, Firestore, ML-Vision, Storage, Database, Messaging, } v17.4.1
- FirebaseUI v6.2.1
- Volley Library v1.0.19
- JSoup v1.11.3
- Google {Directions API, Geocoding API, Token Service, Maps API, Cloud Vision } v17.0.0
- Google Play Services, Location v17.0.0
- Hdodenhof v3.0.1
- Square Open-Source {Picasso, Retrofit, ConverterGSON} v2.71828
- Glide v4.11.0
- Junit v4.13
- Mockito v1.10.19

2.2 Installation Instructions

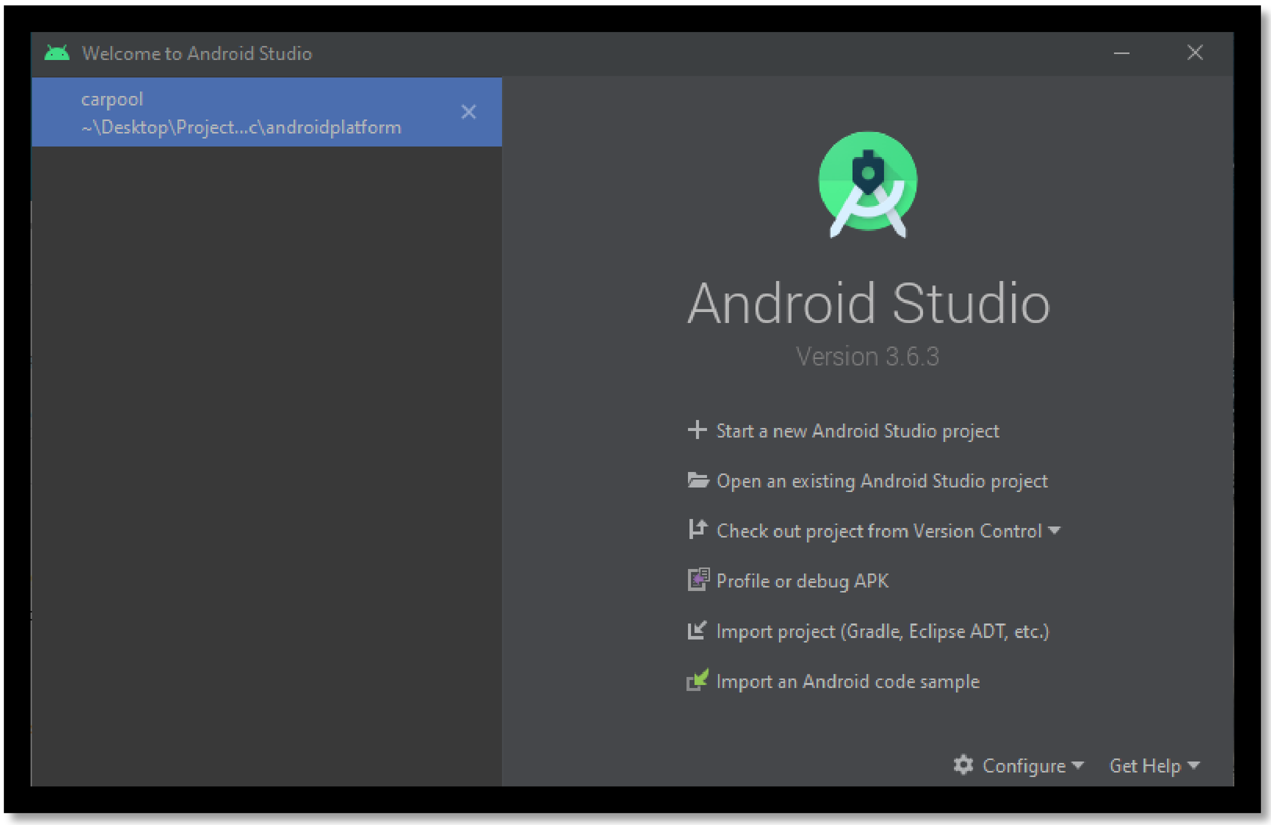
These installation instructions are described for those who wish to view the source code and provide further development features to the application. For those who want to know how to use the application, please refer to the User Manual contained within the same repository branch.

1. Prerequisites:
   a. Android Studio is installed on the user's development environment.
   b. ADB is installed on the user's device.

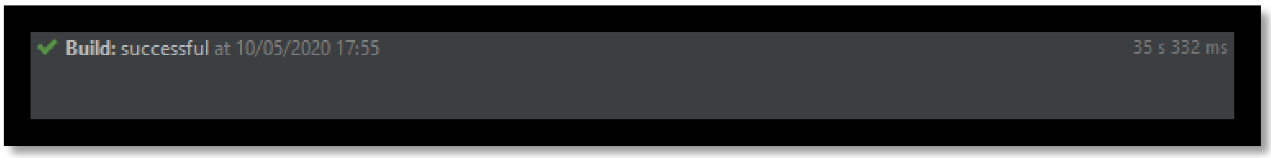
2. Download the Project folder from its repository located on Gitlab at the following link: https://gitlab.computing.dcu.ie/guvenn2/2020-ca400-nguven-scarey

3. Find the downloaded project in the local file system *(by default, the Downloads folder on Windows, Mac, or Linux operating systems).*

4. From the Android Studio home interface, click "*Open an existing Android Studio project*". Open the application root app folder in Android Studio IDE. This root app folder can be found by looking for the Android Studio icon in the folder.


Welcome to Android Studio
carpool
~\Desktop\Project...c\androidplatform
Android Studio
Version 3.6.3
Start a new Android Studio project
Open an existing Android Studio project
Check out project from Version Control
Profile or debug APK
Import project (Gradle, Eclipse ADT, etc.)
Import an Android code sample
Configure
Get Help

5. Wait for the program to compile the gradle files and build the system.



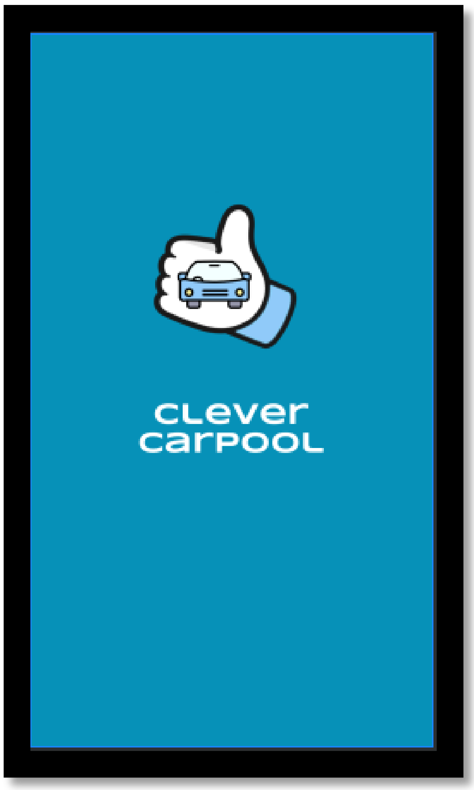
```
✓ Build: successful at 10/05/2020 17:55                    35 s 332 ms
```

6. Once the green tick symbol has been displayed, a run option should become available at the top of the screen.

7. If no virtual device has been downloaded or the ADB has not been configured, please download a virtual phone, or connect a physical device by USB or Wi-Fi. Click the Run button to the right of the Emulator drop-down menu and the application should run.

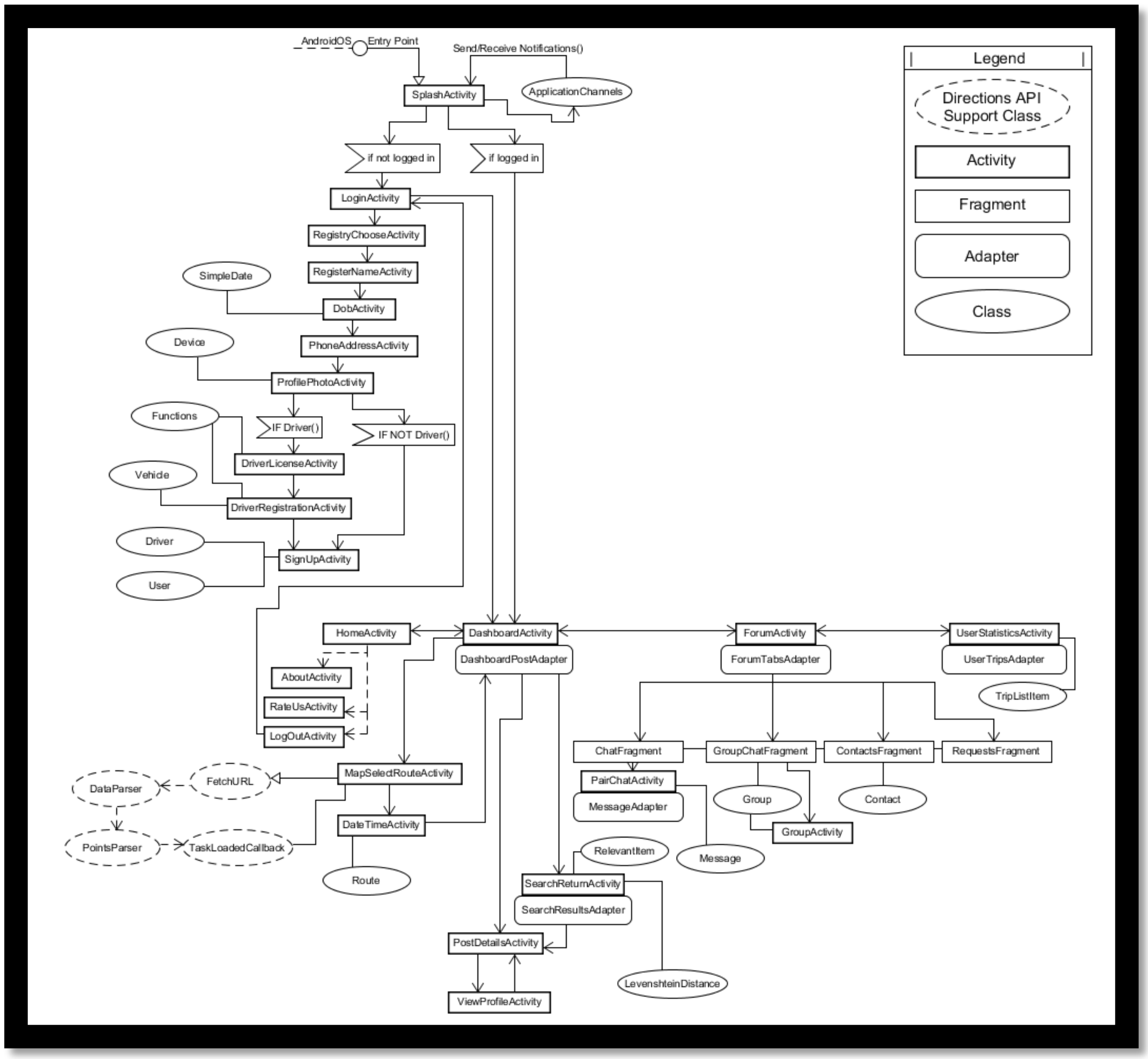
8. The application splash screen should display as below indicating that the phone has been connected to Android Studio with the application installed. The application will run automatically but if this is not the case, please refer to errors that may display in Logcat.

9. You are now ready to provide debugging for the application. The phone will output many different data points to Logcat which you help with building the application. You may wish to select a filter in Logcat so that only application specific logs are displayed to the screen.



3. System Architecture

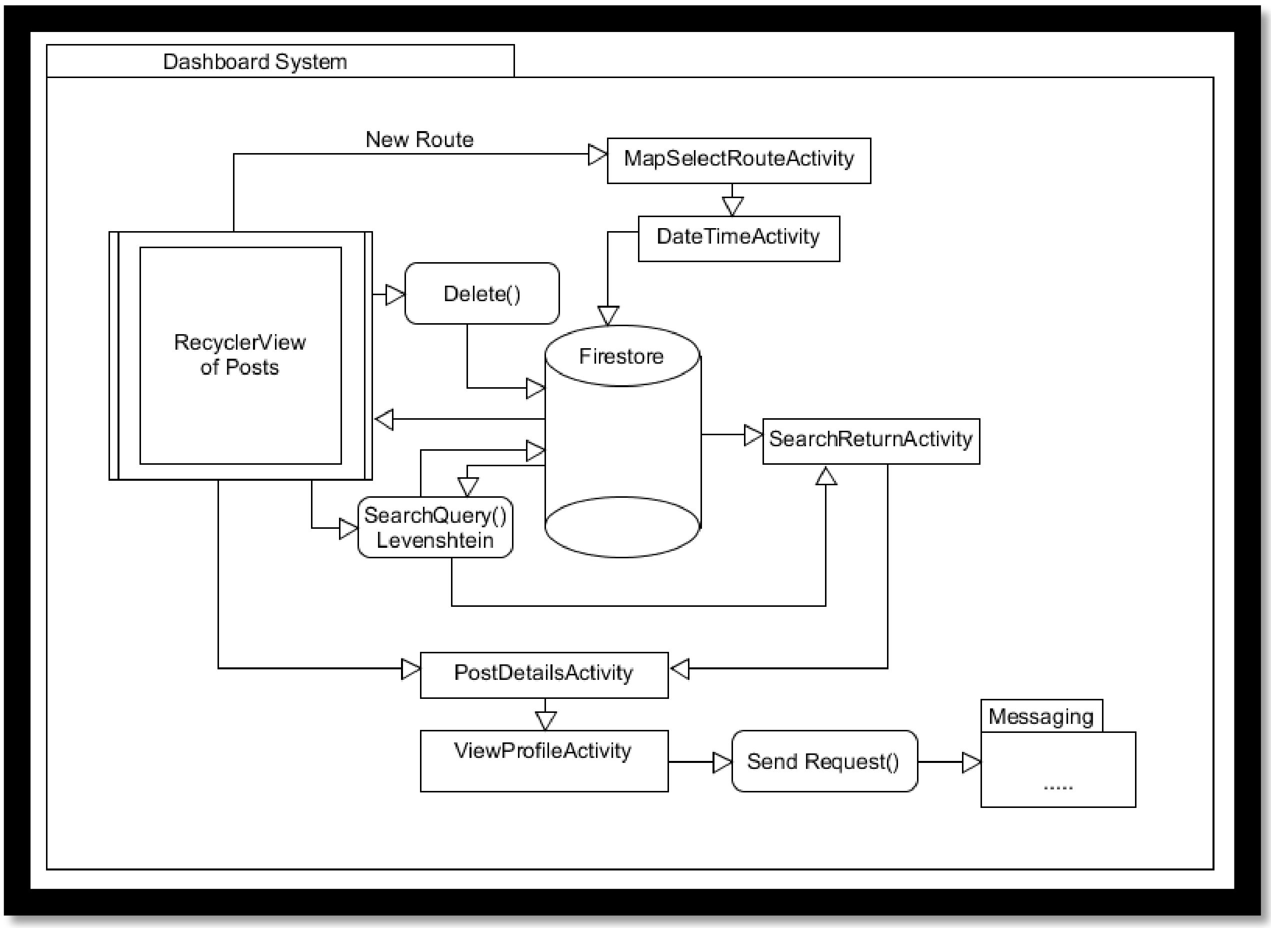
3.1 High Level Design



The High-Level Design encompasses an overview of the entire system with each of its components. It is biased towards a view of the flow of activities and their dependencies. A legend is viewable in the top right to indicate what each type of object is in the architecture.

These are analysed in depth in the following diagrams of this chapter.

3.2 Dashboard Architecture

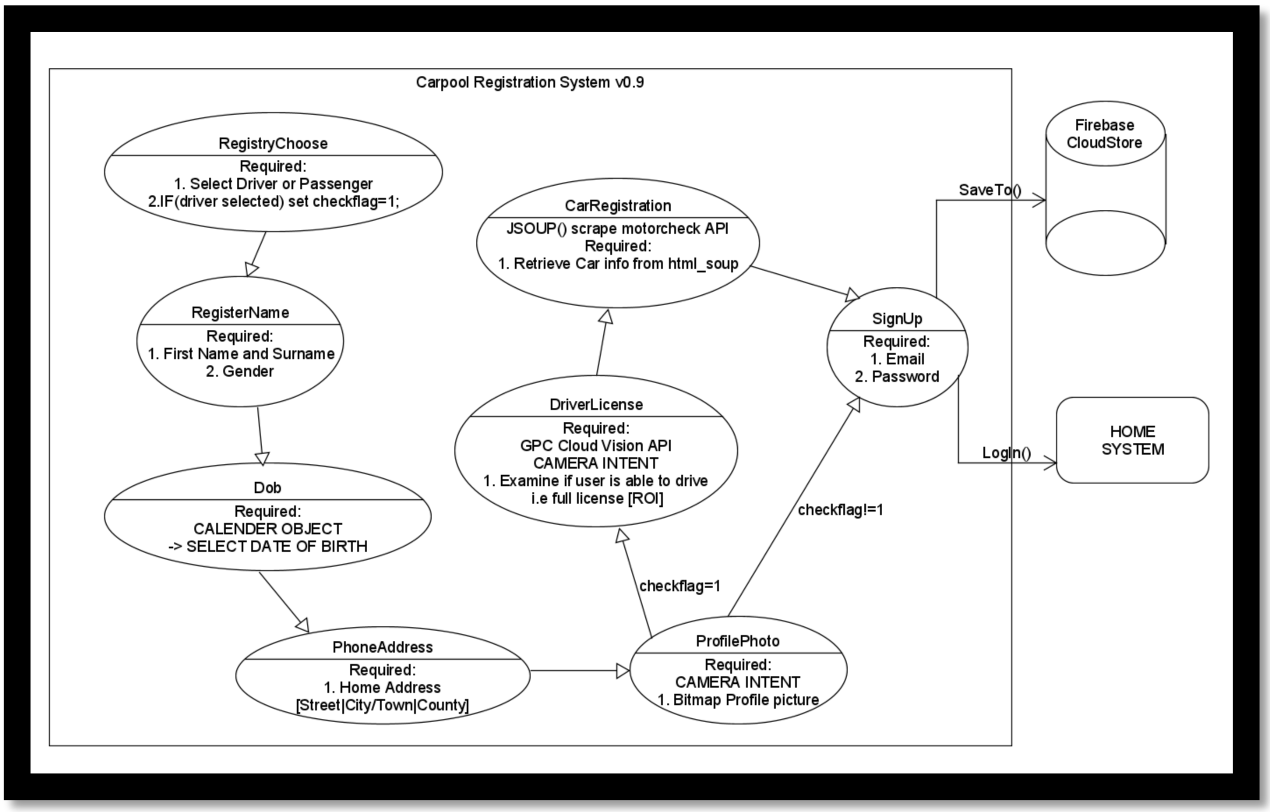


The dashboard is the central system of the application. It holds every post which users may create. Users can view all posts, including their own in the main recycler view. The user can delete their own posts if they so wish. Firestore holds the posts and the posts are loaded at runtime. A sub-deletion method removes posts which are out of date. The user clicks on a floating button indicating to add a new post. The user may select the origin and destination on a Google Map. The Directions API is inferred to create a polyline route in JSON with accurate distance and duration of the route.

Following on from this, the user selects a date and time which indicates what day and time the user is leaving for their destination. From this, the route is stored on Firestore and is viewable on the main list of posts. Users may enter a search query which is provided by a Levenshtein distance score for relevance. This retrieves relevant route items.

Posts are clickable which reveals the full details about a route and its owner. The user can view the owner's profile from here and then send a request to contact them. The owner of a post receives notifications from here for when their post is viewed, and someone sends them a request. The user can also cancel a request from the owner's profile if they so choose.

3.3 Login Subsystem

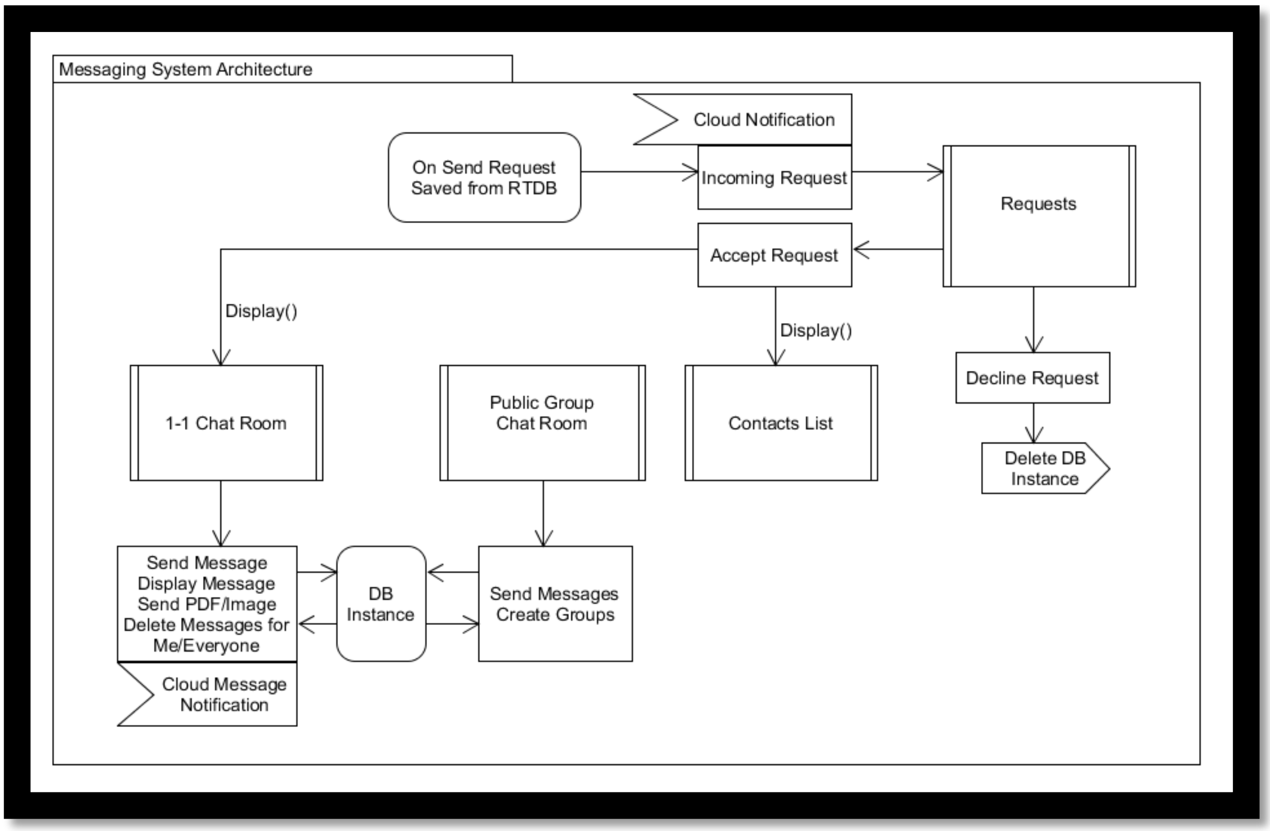


The Login system provides the ability for users to create accounts. There are many checks for formatting and verification of age, licence, and vehicle registration. Firstly, a user selects to register as a driver or passenger. Naturally, drivers have more control involving setting up routes and cancelling routes. If the user is not over the age of 18, then the application will not register for that instance, and the user will be sent back to the main login screen. The user may select a Republic of Ireland or Northern Ireland mobile number alongside their address. Following the taking of a user selfie for a profile picture, based on selection of driver or passenger, different circumstances follow.

For a driver, they must then take a picture of their driver's licence, valid for the Republic of Ireland. If they take a picture of an invalid or learner permit licence, then the application will not allow the user to continue. A longest common substring algorithm combined with Firebase ML Kit for Object Character Recognition (OCR) detects a valid licence.

Following a valid licence, the user must enter their car registration which is combined with the Motor Check API and is web-scraped to retrieve car details. Both drivers and passenger finally add their emails and password which are checked for validity. Upon completion of successful registration, the user is brought to the Dashboard screen with their details saved to Firestore and Realtime Database.

3.4 Messaging System Architecture



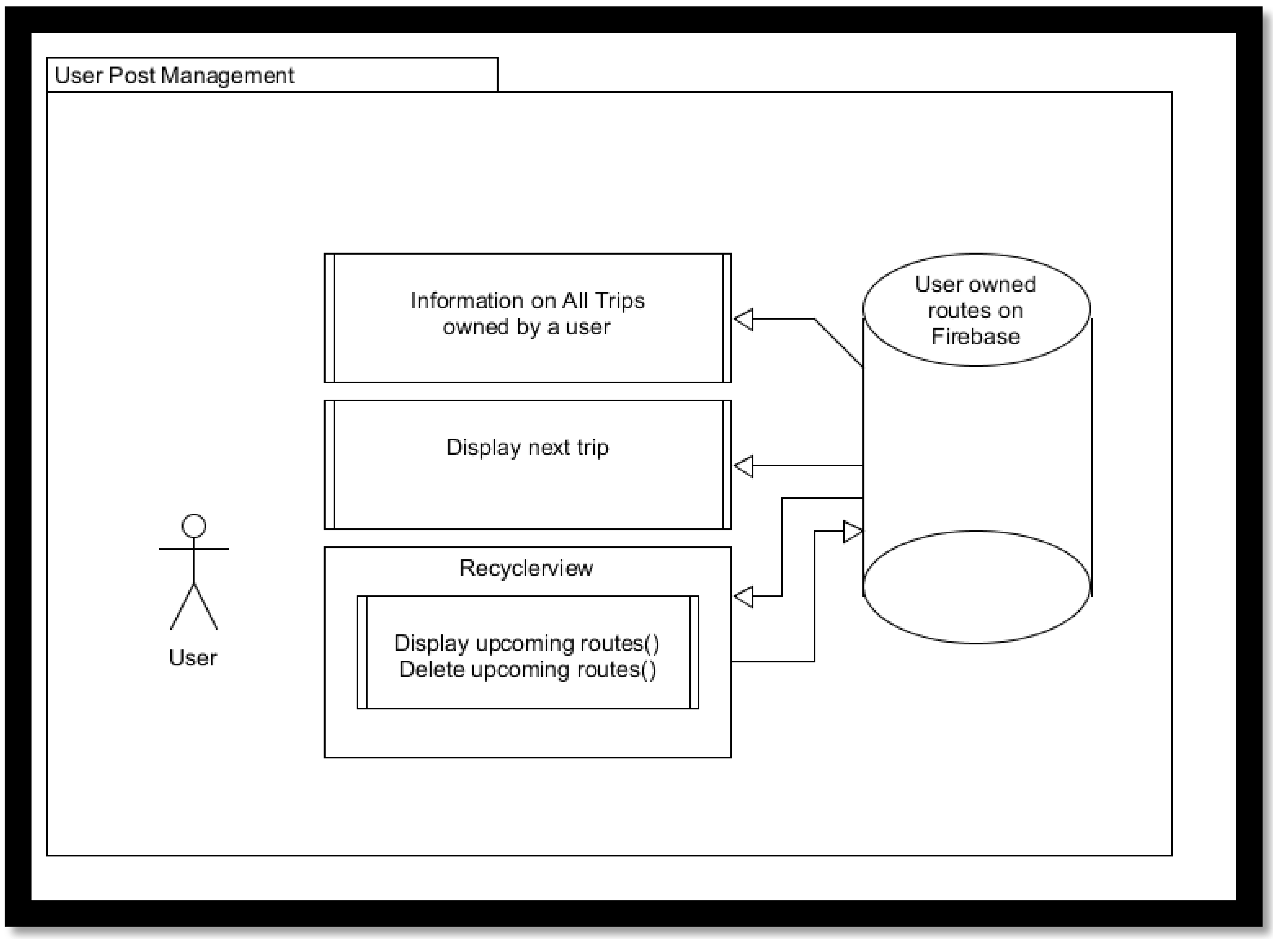
Messaging System Architecture
Cloud Notification
On Send Request Saved from RTDB
Incoming Request
Requests
Accept Request
Display()
Display()
1-1 Chat Room
Public Group Chat Room
Contacts List
Decline Request
Delete DB Instance
Send Message Display Message Send PDF/Image Delete Messages for Me/Everyone
DB Instance
Send Messages Create Groups
Cloud Message Notification

Users may view requests from others inside the request fragment. They may accept or decline the request and depending on their choice, a contact is saved on the database instance or is deleted. Node.js notifications are received when a request is sent from another user. A user who has been accepted by the current user will be shown in the list of contacts as well as on the 1-1 chat fragment.

For group chats, any user may create a group with a title and all users in the application may write in that chat room. Only connected people may use the private 1-1 chat interface. Users can also see if their contacts are online or offline. If a user is not currently using the application, but is not logged out, then a timestamp of their last usage of the application is displayed. 1-1 chat rooms have more functionality than public chat rooms.

Private users can delete messages for themselves and for everyone as well as send images and documents. When a user sends a message of type text, image or PDF, the receiver will get a notification from Firebase with the user who sent that specific message.

3.5 User Statistics Subsystem


User Post Management
Information on All Trips owned by a user
Display next trip
Recyclerview
Display upcoming routes()
Delete upcoming routes()
User
User owned routes on Firebase

Users can manage their trips from the User Trip section of the home system. Information regarding upcoming journeys are displayed to the user including the total time and distance they will be travelling according to the active posts they have on the dashboard. Their next trip is shown in detail and further routes are displayed and removable. This section is necessary to make it easier for users to manage their own posts.

4. Testing

4.1 Procedure

Testing is critical to the delivery of a high-quality product which is in coordination with the goals of completeness, correctness, and consistency. A rigid testing strategy is key to delivering successful software and the Testing Manual, which is contained in the same repository, is proof that we have established our project with testing as a fundamental part of the CA400 Final Year project.

Our strategy involves testing in four critical stages. We want to perform unit and integration testing, so the functionality of our project is correct and reliable and provides redundancy measures. A key aspect relating to our project's development came in inspiration from "Software Engineering" by Ian Sommerville specifically the topic of Dependability and Security whereby a system should be specifically available, secure, and reliable. This book was provided as a referential document to our CA4004 module.

We also want to provide performance testing and ad hoc testing. Performance testing relates to how well the application operates with the underlying system. Ad Hoc testing has been performed throughout the duration of the development stage and is mainly unplanned work which refers to bugs which creep up during development and are fixed when necessary rather than being provided with further analysis.

JUnit4 is to be used to provide the tools for testing on Android Studio for unit, instrumentation, and integration tests. These automated test cases are written separately to the main application under debugging. Manual cases are handled for critical features and are a proponent of ad hoc testing.

Performance testing is monitored by analysing the resource constraints on the underlying system. We are using 3rd party tools handled by 3C Task Manager to monitor resource usage at certain points of the application in the hopes that we can refactor functions to optimise application activities. API calls and read/write operation are expensive so we must examine how we can measure the intensity and reduce the laborious functions.

The Testing Manual provides extensive documentation on how we carried out testing and our strategy towards delivering robust software. We include an in-depth analysis in four areas of testing and our evaluation of our testing strategy. Please refer to the Testing Manual for further details on this aspect. Further below, there is a chapter on the problems we faced and how we were able to circumvent and solve those problems.

5. Problems and Resolutions

This chapter describes some of the major problems we stumbled upon during the development stage of our application. Through careful analysis and research, we were able to solve these problems using various algorithms and functions, which were developed in-house or are credited by third parties.

Licence Validation

Description: The Activity containing the functionality for a user to take a picture of their driver's licence required us to provide a system to read a valid Irish driving licence from an image. Furthermore, we required a systematic approach to analyse the text and force a decision on whether a licence was valid or not. We specifically wanted to compare a valid licence against a learner's permit which both contain different text in English and Irish. This made our job slightly easier as the system could clearly distinguish between the two texts.

Resolution: Our first task was to develop a way of reading text from an image. We found that Firebase had a machine learning system called ML Kit which provides various AI functions for Android projects. We went a step further and subscribed to the Firebase Blaze plan which extends support for the latest versions of ML Kit as well as the entire anthology of Firebase development libraries. We were able to take a picture of various images containing text, including our own licences and learner permits with which we could perform testing upon. We utilised Logcat to see exactly what was being read from a FirebaseVisionImage object. This object returns an OCR array of text blocks.

Our next step was to decide how the system could analyse the blocks of text and decide if a valid licence may be contained within. We utilised a longest common substring algorithm and by our algorithm providing the sum of all common substring totals against two templates texts, one for the driver's licence and the other for the learner's permit. Based on two confidence values which were the sums of the two LCS algorithms, the system could decide if a licence was valid or not. The confidence value also must be over a specified range, or the user will be forced to retake their picture.

Car Registration Retrieval at Runtime

Description: We planned to use JSoup to scrape an API call result from the Irish motor check API web page. This was a workaround to circumvent having to pay for the usage of this API. After having it working and retrieving data from car registrations, we ran into a problem of how we would call this at runtime. The user must be provided with their car details on the screen after inputting their car registration.

Resolution: We fixed this problem by inserting a timer so that when a user has stopped typing, a 3 second delay occurs, which allows the retrieval of the web scraped data. This is then displayed to the user, and if it is valid it is displayed to the user with all their vehicle details.

Multiple Directions API calls at Runtime

Description: In the Post Details Activity, the current user may view another user's post which includes directions on a Google map and more text-based data such as date, time origin and destination of the route in particular. However, we wanted to provide further interesting data to the user by means of stating, for example the post owner is a passenger requesting a lift for a certain journey and the current viewer is a driver, the driver receives information saying the extra time and distance required to pick up that passenger. This is the same case for a passenger viewing a driver's post.

Resolution: We came up with an innovative but labour-intensive function which requires 4 API calls for the current user location and 3 Directions API calls. One directions API call displays the route associated with the post on Google Maps. The other two are called by the Volley Library to retrieve JSON data for time and duration for the origin to the user and the user to the destination. These are displayed to the user on the screen in an example for a driver viewing a passenger's post like:

"Picking this person up will add 15 km(s) to the journey and 30 minutes in extra time."

This provides valuable information for the likes of the driver which may influence their decision to connect with that other person and later pick them up. Despite the expensive functionality, we were able to get it to work and it performed without causing any issues.

User 'Last Seen' Feature

Description: We implemented a last seen feature for our messaging system so that users could be informed when another contact was using the application. However, this feature was originally contained to the Messaging system and not available outside of this. When we extended this feature to every activity apart from Login activity for obvious reasons, we found that the Last seen would display that the user was offline and then quickly become online. Also, the decision was made to extend this feature to the whole application since another user may be using other features of the application but would be logged out.

Resolution: We had to first understand how the Android Activity lifecycle works. We began to improve the functionality of this feature so that for a user who was logged out, the application would display 'Offline' and save this instance to Firebase Realtime Database. For a user who has the application running in the background or is still logged in, a last seen followed by a timestamp would display in the messaging system and this was also saved to the Realtime Database. A user displays 'Active Now' given that they are inside the home system of the application. We have our functions in the OnStart(), OnPause() and OnDestroy() methods of each activity.

Effective Search Algorithm

Description: We wanted to implement an effective search algorithm which could find the most relevant posts according to user-provided query in the Dashboard Activity Search Bar. Our original algorithm was simple and essentially checked to see if a query string matched any string related to posts. However, this was not effective as there was no measure for relevance and some queries did not provide accurate results.

Resolution: We fixed this problem by including a Levenshtein Distance algorithm which would provide a similarity score between the query and post strings. The Levenshtein Distance algorithm operates by analysing substitutions, deletions, and insertions of characters between two strings and returns a score between 0 and 1. We then displayed these in a new activity recycler view ranked by relevance. This made user search queries more effective. We also made sure that if a query provided no results, then the application tells the user that the search query returned no results rather than sending the user to another activity with an empty list.

Group Messaging

Description: Group Messages do not act the same way as peer-to-peer messages. Since they are open forums, and any users may send messages within a single group. Group messages are stored under the same database instance while peer-to-peer chat messages are contained within two instances under the two users who are members of that chat room. The main problem with how we would display the group messages was how we would display them

Resolution: We found an innovative solution by displaying messages at runtime and creating dynamic layout views for each message. Designs for each message are created dynamically.

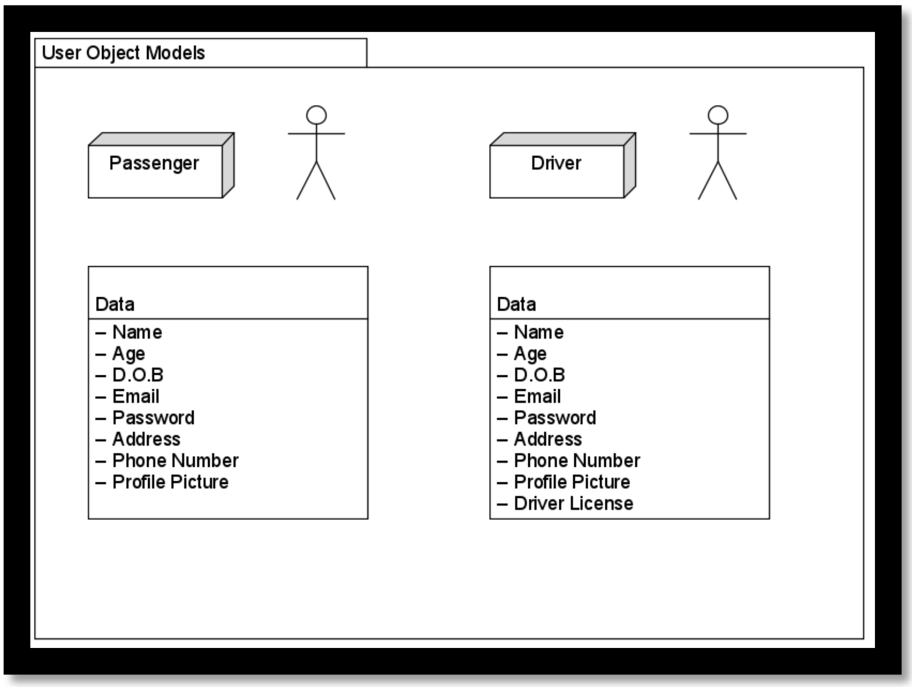
Notifications Operations

Description: We planned to send notifications between users using the Android SDK with specific notifications for requests, messages, and post viewings. Our problem was that there were not enough tutorials with notifications using the Android SDK, so it was hard to develop our own working example.

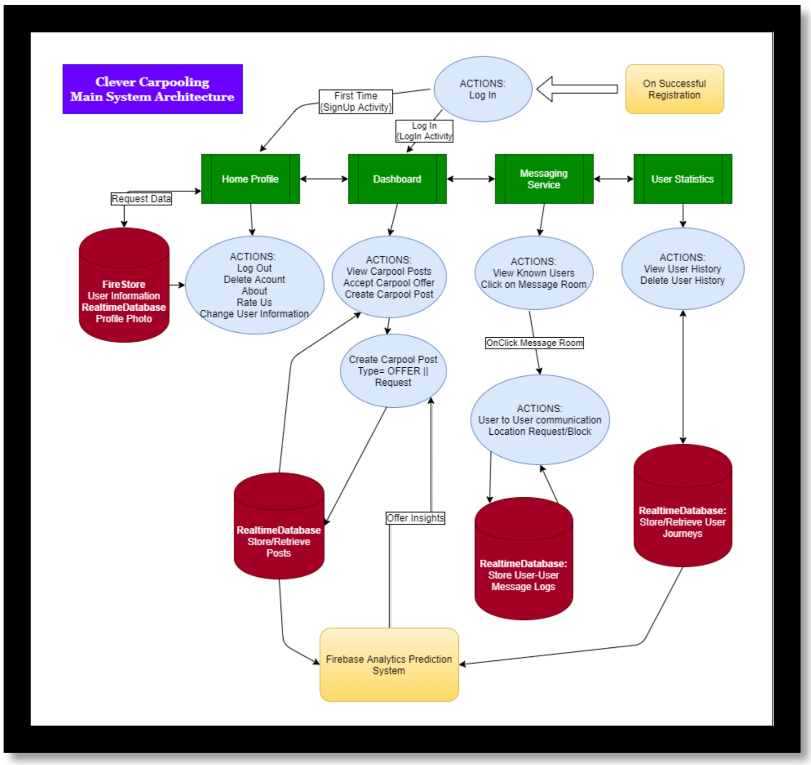
Resolution: We found tutorials regarding the usage of Node.js and Firebase Cloud Functions to send notifications between users. According to the device token with which the user has the application currently installed on, and given one of the three types of notifications, a notification is sent to that user's phone. We uploaded Node.js functions to Firebase and these reacted to various database events. A notifications reference on the Realtime Database is created when a user performs an action. The Firebase Cloud Function then reacts to that event reference and sends a notification.

6. Appendices

Driver and Passenger Object Models



Home System Architecture


Clever Carpooling
Main System Architecture
ACTIONS: Log In
On Successful Registration
First Time {SignUp Activity}
Log In {LogIn Activity}
Home Profile
Dashboard
Messaging Service
User Statistics
Request Data
FireStore User Information RealtimeDatabase Profile Photo
ACTIONS: Log Out Delete Acount About Rate Us Change User Information
ACTIONS: View Carpool Posts Accept Carpool Offer Create Carpool Post
ACTIONS: View Known Users Click on Message Room
ACTIONS: View User History Delete User History
Create Carpool Post Type= OFFER || Request
OnClick Message Room
ACTIONS: User to User communication Location Request/Block
RealtimeDatabase Store/Retrieve Posts
Offer Insights
RealtimeDatabase: Store User-User Message Logs
RealtimeDatabase: Store/Retrieve User Journeys
Firebase Analytics Prediction System

Driver Licence Firebase OCR Functions

```java
private void detectTextFromImage()
{
    FirebaseVisionImage = FirebaseVisionImage.fromBitmap(mbitmap);
    FirebaseVisionTextRecognizer =
FirebaseVision.getInstance().getCloudTextRecognizer();

firebaseVisionTextRecognizer.processImage(firebaseVisionImage).addOnSuccessListene
r(new OnSuccessListener<FirebaseVisionText>()
    {
        @Override
        public void onSuccess(FirebaseVisionText firebaseVisionText)
        {
            readTextFromImage(firebaseVisionText);
        }
    }).addOnFailureListener(new OnFailureListener()
    {
        @Override
        public void onFailure(@NonNull Exception e)
        {
            makeText(DriverLicenseActivity.this, "Error:" + e.getMessage(),
Toast.LENGTH_SHORT).show();
        }
    });
}

private void readTextFromImage(FirebaseVisionText firebaseVisionText)
{
    String image_ocr = "";
    List<FirebaseVisionText.TextBlock> mblocklist =
firebaseVisionText.getTextBlocks();
    if(mblocklist.size()==0)
    {
        makeText(DriverLicenseActivity.this, "A valid license has not been found
in the image." , Toast.LENGTH_SHORT).show();
    }
    else
    {

        for(FirebaseVisionText.TextBlock block:
firebaseVisionText.getTextBlocks())
        {
            image_ocr += block.getText();
            if(!containsDigit(image_ocr))
            {
                listOfBlocks.add(image_ocr);
                Log.d(TAG, "DEBUG: " + image_ocr);
            }
        }
    }
}
```

Driver Registration JSoup HTML Scraper

```
if (editable.length() >= 3)
{

    timer = new Timer();
    timer.schedule(new TimerTask()
    {
        @SuppressWarnings("SingleStatementInBlock")
        @Override
        public void run()
        {
            try
            {
                final String mcar_registration =
mCarReg.getText().toString().trim().replace("-", "");
                if (TextUtils.isEmpty(mcar_registration)) {
                    mCarReg.setError("Valid car registration is required.");
                }
                else if(mcar_registration.matches("^.*[^a-zA-Z0-9 ].*$"))
                {
                    mCarReg.setError("Valid car registration is required.");
                }

                doc = Jsoup.connect(url + mcar_registration).get();
                String html_soup = doc.select("div[class=col-md-6 align-items-
center]").toString();
                html_soup = Jsoup.clean(html_soup, Whitelist.none());
                Log.d(TAG, "+++ DEBUG +++ " + html_soup);

                if(html_soup.equals(""))
                {
                    Looper.prepare();

                    Toast.makeText(DriverRegistrationActivity.this, "Failed to
retrieve car registration details.", Toast.LENGTH_SHORT).show();
                }
                else {
                    html_soup = html_soup.replaceAll(",","\n");
                    html_soup = html_soup.replaceAll("Body:", "\n");
                    html_soup = html_soup.replaceAll("Engine CC:", "\n");
                    html_soup = html_soup.replaceAll("Fuel:", "\n");
                    html_soup = html_soup.replaceAll("Colour:", "\n");
                    Log.d(TAG, "+++ DEBUG +++ " + html_soup);
                    String[] vehicleData = html_soup.split("\n", 7);
                    String [] tmpArr = vehicleData[5].split("W",10);
                    mvehicle = new
Vehicle(vehicleData[0].trim(),mcar_registration.trim(),vehicleData[1].trim(),tmpAr
r[0].trim(),vehicleData[3].trim(),vehicleData[4].trim(),vehicleData[2].trim());
                    Log.d(TAG, "+++ DEBUG CAR TYPE = " + vehicleData[0]);
                    Log.d(TAG, "+++ DEBUG MCARREG = " + mcar_registration);
                    Log.d(TAG, "+++ DEBUG CAR YEAR = " + vehicleData[1]);
                    Log.d(TAG, "+++ DEBUG CAR COLOR = " + vehicleData[5]);
                    Log.d(TAG, "+++ DEBUG CAR ENGINE = " + vehicleData[3]);
                    Log.d(TAG, "+++ DEBUG CAR FUEL = " + vehicleData[4]);
                    Log.d(TAG, "+++ DEBUG CAR BODY = " + vehicleData[2]);
                 runOnUiThread(new Runnable()
                 {
                        @Override
                        public void run()
                        {
                            mCarReg.setVisibility(View.INVISIBLE);
                            retypeBtn.setVisibility(View.VISIBLE);
                            mtext1.append(mvehicle.getCar_type());
                            mtext2.append(mvehicle.getCar_year());
                            mtext3.append(mvehicle.getCar_color());
                            mtext4.append(mvehicle.getCar_engine_cc());
                            mtext5.append(mvehicle.getCar_fuel_type());
                            mtext6.append(mvehicle.getCar_body());

                        }
                    });
                }
            }
            catch (IOException e)
            {
                e.printStackTrace();
                Looper.prepare();
                Toast.makeText(DriverRegistrationActivity.this, "This requires an
internet connection to process.", Toast.LENGTH_LONG).show();

            }
            catch (NullPointerException f)
            {
                f.printStackTrace();
                Looper.prepare();
                Toast.makeText(DriverRegistrationActivity.this, "NullPointer: This
app requires an active Internet connection. Please enable Wifi in Settings. ",
Toast.LENGTH_LONG).show();
            }

        }

    }, DELAY);
```

Levenshtein Algorithm for String Similarity

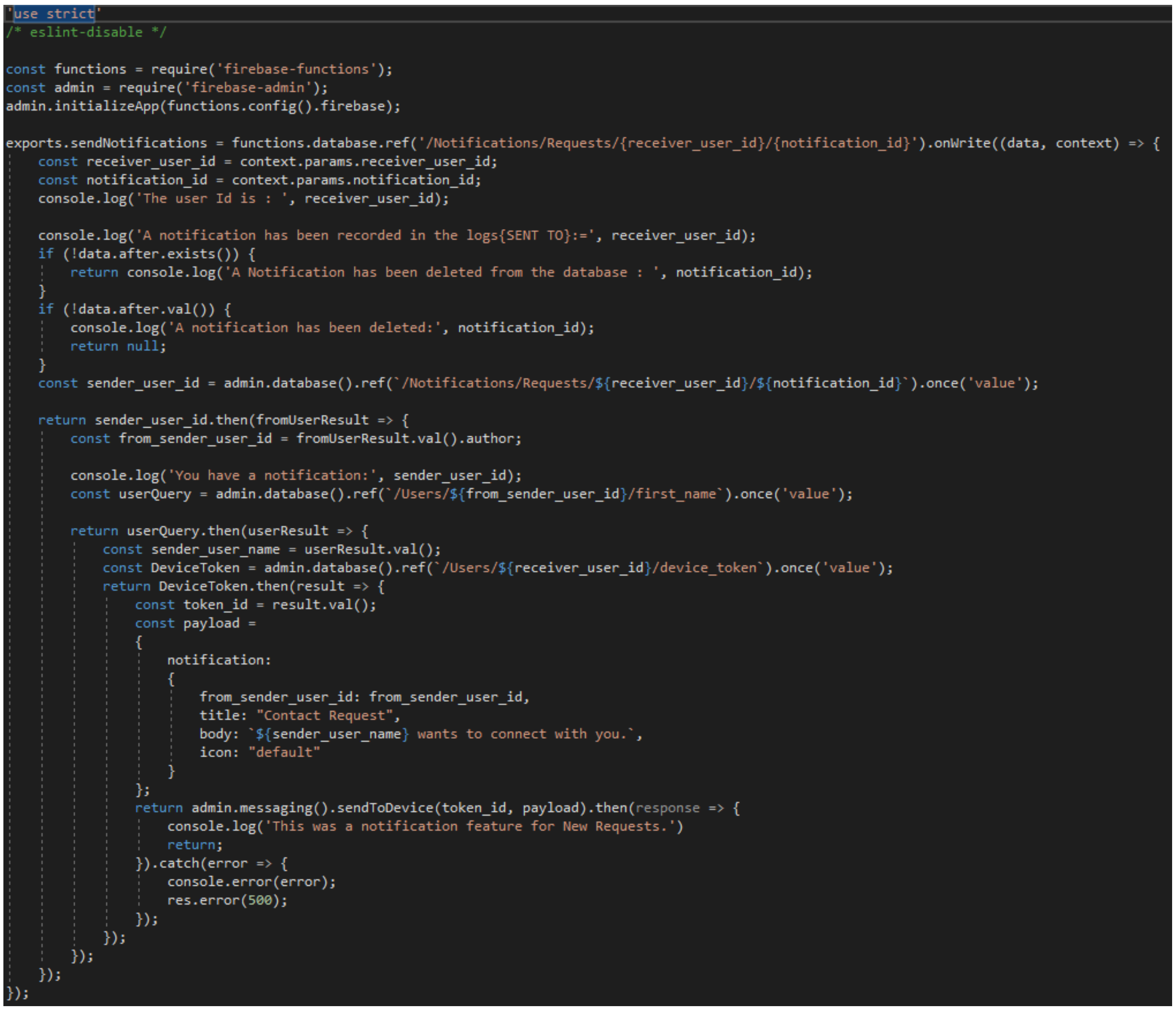
```
private static int computeDistance(String s1, String s2)
{
    s1 = s1.toLowerCase();
    s2 = s2.toLowerCase();

    int[] costs = new int[s2.length() + 1];
    for (int i = 0; i <= s1.length(); i++)
    {
        int lastValue = i;
        for (int j = 0; j <= s2.length(); j++)
        {
            if (i == 0)
            {
                costs[j] = j;
            }
            else
            {
                if (j > 0)
                {
                    int newValue = costs[j - 1];
                    if (s1.charAt(i - 1) != s2.charAt(j - 1))
                    {
                        newValue = Math.min(Math.min(newValue, lastValue),
costs[j]) + 1;
                    }
                    costs[j - 1] = lastValue;
                    lastValue = newValue;
                }
            }
        }
        if (i > 0)
        {
            costs[s2.length()] = lastValue;
        }
    }
    return costs[s2.length()];
}

public static double returnDistance(String s1, String s2)
{
    double relevance_score;
    int editDistance;
    if (s1.length() < s2.length())
    {
        String swap = s1;
        s1 = s2;
        s2 = swap;
    }
    int bigLen = s1.length();
    editDistance = computeDistance(s1, s2);
    if (bigLen == 0) {
        relevance_score = 1.0;
    } else {
        relevance_score = (bigLen - editDistance) / (double) bigLen;
    }
    return relevance_score;
}
```

Node.JS Firebase Cloud Function

This cloud function sends a request to a user from another with the name displayed.



```
'use strict'
/* eslint-disable */

const functions = require('firebase-functions');
const admin = require('firebase-admin');
admin.initializeApp(functions.config().firebase);

exports.sendNotifications = functions.database.ref('/Notifications/Requests/{receiver_user_id}/{notification_id}').onWrite((data, context) => {
    const receiver_user_id = context.params.receiver_user_id;
    const notification_id = context.params.notification_id;
    console.log('The user Id is : ', receiver_user_id);

    console.log('A notification has been recorded in the logs{SENT TO}:=', receiver_user_id);
    if (!data.after.exists()) {
        return console.log('A Notification has been deleted from the database : ', notification_id);
    }
    if (!data.after.val()) {
        console.log('A notification has been deleted:', notification_id);
        return null;
    }
    const sender_user_id = admin.database().ref(`/Notifications/Requests/${receiver_user_id}/${notification_id}`).once('value');

    return sender_user_id.then(fromUserResult => {
        const from_sender_user_id = fromUserResult.val().author;

        console.log('You have a notification:', sender_user_id);
        const userQuery = admin.database().ref(`/Users/${from_sender_user_id}/first_name`).once('value');

        return userQuery.then(userResult => {
            const sender_user_name = userResult.val();
            const DeviceToken = admin.database().ref(`/Users/${receiver_user_id}/device_token`).once('value');
            return DeviceToken.then(result => {
                const token_id = result.val();
                const payload =
                {
                    notification:
                    {
                        from_sender_user_id: from_sender_user_id,
                        title: "Contact Request",
                        body: `${sender_user_name} wants to connect with you.`,
                        icon: "default"
                    }
                };
                return admin.messaging().sendToDevice(token_id, payload).then(response => {
                    console.log('This was a notification feature for New Requests.')
                    return;
                }).catch(error => {
                    console.error(error);
                    res.error(500);
                });
            });
        });
    });
});
```

Volley/FusedAPI/Directions API Calls

```
private void getCurrentLocation()
{
    final LocationRequest = new LocationRequest();
    locationRequest.setInterval(2000);
    locationRequest.setFastestInterval(1000);
    locationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

    LocationServices.getFusedLocationProviderClient(PostDetailsActivity.this)
            .requestLocationUpdates(locationRequest, new LocationCallback()
            {
                @Override
                public void onLocationResult(LocationResult locationResult)
                {
                    super.onLocationResult(locationResult);

LocationServices.getFusedLocationProviderClient(PostDetailsActivity.this).removeLo
cationUpdates(this);
                    if(locationResult != null &&
locationResult.getLocations().size() > 0)
                    {
                        int latestLocationIndex =
locationResult.getLocations().size() - 1;
                        mylatitude =
locationResult.getLocations().get(latestLocationIndex).getLatitude();
                        mylongitude =
locationResult.getLocations().get(latestLocationIndex).getLongitude();
                        Log.d("FUSED_LOCATION_API", mylatitude + " " +
mylongitude);
                        if (mylatitude != 0.0)
                        {
                            MarkerOptions options = new
MarkerOptions().position(SRC).icon(BitmapDescriptorFactory.fromResource(R.drawable
.src_mapmarker));
                            MarkerOptions options2 = new
MarkerOptions().position(DST).icon(BitmapDescriptorFactory.fromResource(R.drawable
.dst_mapmarker));
                            mMap.addMarker(options);
                            mMap.addMarker(options2);
                            coordinate = new LatLng(mylatitude, mylongitude);
                            MarkerOptions options3 = new
MarkerOptions().position(coordinate);
                            mMap.addMarker(options3);
                            Log.d("FUSED_API", "" + mylatitude);
                            CameraUpdate yourLocation =
CameraUpdateFactory.newLatLngZoom(coordinate, 7);
                            mMap.animateCamera(yourLocation);
                        }
                        String urlToUser1 =
"https://maps.googleapis.com/maps/api/directions/json?origin=" + SRC.latitude +
","+ SRC.longitude + "&destination=" + mylatitude + "," + mylongitude + "&key=" +
getString(R.string.google_maps_key);
                        Log.d("FUSEDAPI", urlToUser1);
                        JsonObjectRequest request = new
JsonObjectRequest(urlToUser1, new Response.Listener<JSONObject>()
                        {
                            @Override
                            public void onResponse(final JSONObject response1)
                            {
                                String urlToUser2 =
"https://maps.googleapis.com/maps/api/directions/json?origin=" + DST.latitude +
","+ DST.longitude + "&destination=" + mylatitude + "," + mylongitude + "&key=" +
getString(R.string.google_maps_key);
                                Log.d("FUSEDAPI", urlToUser2);
                                JsonObjectRequest request = new
JsonObjectRequest(urlToUser2, new Response.Listener<JSONObject>(){
                                    @Override
                                    public void onResponse(JSONObject response2) {
                                        try
                                        {
                                            JSONArray jRoutes1 =
response1.getJSONArray("routes");
                                            JSONArray jLegs1 = ((JSONObject)
jRoutes1.get(0)).getJSONArray("legs");
                                            JSONObject srcToUsr_distanceJSON1 =
jLegs1.getJSONObject(0).getJSONObject("distance");
                                            JSONObject srcToUsr_durationJSON1 =
jLegs1.getJSONObject(0).getJSONObject("duration");
                                            double srcToUsr_distance1 =
formatDistance(srcToUsr_distanceJSON1.getString("text"));
                                            int srcToUsr_duration1 =
formatTime(srcToUsr_durationJSON1.getString("text"));
                                            Log.d("FUSEDAPI", srcToUsr_distance1 +
" || " + srcToUsr_duration1);

                                            JSONArray jRoutes2 =
response2.getJSONArray("routes");
                                            JSONArray jLegs2 = ((JSONObject)
jRoutes2.get(0)).getJSONArray("legs");
                                            JSONObject srcToUsr_distanceJSON2 =
jLegs2.getJSONObject(0).getJSONObject("distance");
                                            JSONObject srcToUsr_durationJSON2 =
jLegs2.getJSONObject(0).getJSONObject("duration");
                                            double srcToUsr_distance2 =
formatDistance(srcToUsr_distanceJSON2.getString("text"));
                                            int srcToUsr_duration2 =
formatTime(srcToUsr_durationJSON2.getString("text"));

                                            Log.d("FUSEDAPI", srcToUsr_distance2 +
" || " + srcToUsr_duration2);
                                            Log.d("FUSEDAPI", routeDistance + " ||
" + routeDuration);

                                            double srcToDst_distance =
formatDistance(routeDistance);
                                            int srcToDst_duration =
formatTime(routeDuration);

                                            double new_distance =
(srcToUsr_distance1 + srcToUsr_distance2) - srcToDst_distance;
                                            int new_duration = (srcToUsr_duration1
+ srcToUsr_duration2) - srcToDst_duration;
                                            Log.d("TAG_A", routeDistance+"");
                                            if(isOffer)
                                            {
                                                String info_line = "Picking you up
will add " + Math.round(new_distance) + " km(s) to the journey and " +
new_duration + " minutes in extra time.";
                                                textViewInfo.setText(info_line);
                                            }
                                            else {
                                                String info_line = "Picking this
person up will add " + Math.round(new_distance) + " km(s) to the journey and " +
new_duration + " minutes in extra time.";
                                                textViewInfo.setText(info_line);
                                            }
                                        } catch (JSONException e) {

textViewInfo.setText(R.string.json_path_error);
                                            e.printStackTrace();
                                        }
                                    }
                                    }, new Response.ErrorListener()
                                    {
                                        @Override
                                        public void onErrorResponse(VolleyError
error2)
                                        {
                                            Log.d(TAG, error2.toString());
                                        }
                                    }
                                );
                                requestQueue.add(request);
                            }
                        }, new Response.ErrorListener()
                        {
                            @Override
                            public void onErrorResponse(VolleyError error1)
                            {
                                Log.d(TAG, error1.toString());
                            }
                        });
                        requestQueue.add(request);
                    }
                }
            }, Looper.getMainLooper());
}
```

Update User Status

```
public static void updateUserStatus(String state)
{
    String saveCurrentTime, saveCurrentDate;

    Calendar = Calendar.getInstance();
    @SuppressLint("SimpleDateFormat")
    SimpleDateFormat currentDate = new SimpleDateFormat("MMM dd, yyyy");
    saveCurrentDate = currentDate.format(calendar.getTime());

    @SuppressLint("SimpleDateFormat")
    SimpleDateFormat currentTime = new SimpleDateFormat("H:mm");
    saveCurrentTime = currentTime.format(calendar.getTime());

    HashMap<String, Object> onlineStateMap = new HashMap<>();

    onlineStateMap .put("time", saveCurrentTime);
    onlineStateMap .put("date", saveCurrentDate);
    onlineStateMap .put("state", state);

    String currentUserID =
Objects.requireNonNull(mFirebaseAuth.getCurrentUser()).getUid();


databaseReference.child("Users").child(currentUserID).child("user_status").updateC
hildren(onlineStateMap);
}
```

Group Messages Dynamic Layout

```
private void printMessages(DataSnapshot dataSnapshot)
{
    Iterator = dataSnapshot.getChildren().iterator();
    while(iterator.hasNext())
    {

        String chatDate = (String) ((DataSnapshot) iterator.next()).getValue();
        String chatMessage = (String) ((DataSnapshot) iterator.next()).getValue();
        String chatName = (String) ((DataSnapshot) iterator.next()).getValue();
        String chatTime = (String) ((DataSnapshot) iterator.next()).getValue();
        addLinearLayout(chatName + " :", chatMessage, chatDate + " " + chatTime);
        count++;
        hashSet.add(chatName);
        groupRef.child("post_count").setValue(count);
        groupRef.child("user_count").setValue(hashSet.size());
        mScrollView.fullScroll(ScrollView.FOCUS_DOWN);
    }
}
@SuppressWarnings("deprecation")
private void addLinearLayout(String tmp1, String tmp2, String tmp3)
{
    LinearLayout ll = new LinearLayout(this);
    TextView displayName = new TextView(this);
    TextView displayMessages = new TextView(this);
    TextView displayDateTime = new TextView(this);
    ll.setOrientation(LinearLayout.VERTICAL);
    LinearLayout.LayoutParams layout_params = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.WRAP_CONTENT,
LinearLayout.LayoutParams.WRAP_CONTENT);
    layout_params.setMargins(24, 16, 24, 8);
    ll.setLayoutParams(layout_params);
    ll.setBackground(ResourcesCompat.getDrawable(getResources(),
R.drawable.group_message_background, null));
    ll.setPadding(8, 8,8, 8);
    displayName.setText(tmp1);

    displayMessages.setText(tmp2);
    displayDateTime.setText(tmp3);

    displayName.setTypeface(null, Typeface.BOLD);
    displayMessages.setTextSize(TypedValue.COMPLEX_UNIT_SP,16);
    displayMessages.setTextColor(getResources().getColor(R.color.black));
    displayDateTime.setTextSize(TypedValue.COMPLEX_UNIT_SP,11);
    displayDateTime.setTypeface(null, Typeface.ITALIC);
    displayName.setPadding(16, 10, 24, 12);
    displayMessages.setPadding(16, 10, 24, 12);
    displayDateTime.setPadding(16, 10, 24, 12);

    ll.addView(displayName);
    ll.addView(displayMessages);
    ll.addView(displayDateTime);
    linearLayout.addView(ll);
}
```

7. References

Android - https://developer.android.com/

Firebase - https://firebase.google.com/

Google Cloud Platform - https://cloud.google.com/

CA400 - https://www.computing.dcu.ie/~pclarke/ca400/

JSoup - https://jsoup.org/

JUnit - https://junit.org/junit4/

Mockito - https://site.mockito.org/

Node.js - https://nodejs.org/en/

Hdodenhof - https://github.com/hdodenhof/CircleImageView

UMLet - https://www.umlet.com/

MotorcheckAPI - https://www.motorcheck.ie/api/

Software Engineering by Ian Sommerville - https://dinus.ac.id/repository/docs/ajar/Sommerville-Software-Engineering-10ed.pdf

3C Task Manager - https://play.google.com/store/apps/details?id=ccc71.tm&hl=en_IE

RSA - https://www.rsa.ie/

-End of Document-