Software Development Final Project

Driving Companion App

Technical Manual

Student: Chi Ieong Ng C00223421
Supervisor: Hisain Elshaafi
Submission Date: 29/04/2022

Table of content

1. Introduction

This Technical Manual is part of my final year project in ITCarlow Software Development Year 4. I decided to develop a mobile app called “Driving companion app”. The app will score and give feedback to drivers by monitoring their driving behavior.

Driving Companion App is a mobile phone application whose purpose is to analyze the user's driving behavior, obtain data from the monitoring process, and give scores and feedback to improve the user's driving behavior.

This document is the technical manual for the "Driving Companion App", which describes the technology used by this application, including its construction, all its functions and how to implement it with code. It is how to calculate driving scores and get driving data from mobile phones through code, speed, acceleration, deceleration, etc., for technical information.

2. Technology Description

To develop the Android application of this mobile phone, Flutter will be used as the development framework. The programming language is Dart, the database uses the Firebase cloud service, and the real-time geographic location and map rely on the Google API service. The following is the introduction of various technologies：

2.1 Flutter

Flutter is an open-source mobile application software development kit developed by Google. Its engine is mainly written in C++ and provides low-level rendering support using Google’s Skia graphics library. Android and IOS also provide Platform-specific SDKs. It can use one programming language (Dart) to create cross-platform applications with Android and iOS and supported Web and desktop applications.

Flutter is a widget-based technology. It can apply object-oriented programming to any element and easily modify or customize widgets.

Flutter has numerous advantages over its competitors. These advantages are inherent in programming languages and development toolsets, allowing Flutter to solve problems that other languages cannot.

- One codebase for all platforms
- Widgets offer countless possibilities
- Rich libraries
- Fast testing with hot reload

Now Flutter is widely used to create apps for Alibaba, Yandex, Airbnb, Uber, eBay and other leading companies. Many excellent social media apps, service booking apps, productivity measurement apps, utility apps, product distribution apps and healthcare apps, all built with Flutter.

Flutter is a widget-based technology. It can apply object-oriented programming to any element and easily modify or customize widgets.

Flutter has numerous advantages over its competitors. These advantages are inherent in programming languages and development toolsets, which allow Flutter to solve issues that other languages cannot cope with.

- One codebase for all platforms
- Widgets offer countless possibilities
- Rich libraries
- Fast testing with hot reload

Now Flutter is widely used to create apps for Alibaba, Yandex, Airbnb, Uber, eBay and other leading companies. Many excellent social media apps, service booking apps, productivity measurement apps, utility apps, product distribution apps and healthcare apps, all built with Flutter.

2.2. Dart

Dart is a programming language developed by Google. It is an Object-oriented language similar to JavaScript. It can compile to either native code or javascript, and its grammatical style is close to the C language. It is a programming language very suitable for web and mobile application development. Dart can be executed on a native virtual machine, converted Dart code to JavaScript, and then performed directly on the Javascript engine. Dart can quickly use the Library provided by Google, and users can also offer their self-written Library for other developers or other projects to use.

2.3. Google Maps API / Roads API

Google Maps API is a Maps programming API provided by Google for developers. It allows developers to embed Google Maps data into web apps by using Javascript. Mainly used to get location, navigation, and map service. Roads API is part of Google Maps API. It maps GPS coordinates to the road and determines speed limits and road segments. It is available via a simple HTTPS interface to expose Snap to roads, Nearest roads, and Speed limits services. And it is compatible with Java and Python. Its libraries make development more accessible by providing simple, native implementations of everyday tasks, such as authentication, request throttling and automatic retry. However, It is not entirely free to use. Google will charge it after the primary free usage is exceeded.

2.4. Firebase

Firebase is a back-end cloud platform (BaaS: Backend-as-a-Service) that provides real-time database service. It is a NoSQL cloud database and supports Android, iOS and Web apps. It can save data in JSON format and synchronize to the online client in real-time. And even if the user is offline, the data is still available.

3. Compatible

This section describes the compatibility of the development environment and the version of each software and tool when testing. It can be used as a reference for standard requirements.

- OS: Android 10 or above
- Hardware: Google Pixel 4
    - Qualcomm SM8150 Snapdragon 855
    - 64GB 6GB RAM, 128GB 6GB RAM
- Framework:
    - Flutter 2.10.3
    - Dark 2.16.1
    - DevTools 2.9.2
- Libaries:
    - cupertino_icons: ^1.0.2
    - google_maps_flutter: ^2.1.1
    - location: ^4.3.0
    - flutter_polyline_points: ^1.0.0
    - geolocator: ^8.0.1
    - fl_chart: ^0.46.0
    - firebase_auth: ^3.3.6
    - flutter_signin_button: ^2.0.0
    - cloud_firestore: ^3.1.8
    - firebase_database: ^9.0.6
    - firebase_core: ^1.12.0
    - http: ^0.13.4
    - flutter_progress_hud: ^2.0.0
    - screen_loader: ^4.0.1
    - page_view_indicators: ^2.0.0
    - get: ^4.6.1
    - marker_icon: ^0.7.1
    - wakelock: ^0.6.1+2
    - flutter_session_manager: ^1.0.3

    - shared_preferences: ^2.0.13
    - percent_indicator: ^4.0.0
- Network:
    - Eircom Limited Ireland 4G network

Note: The network environment will affect the accuracy of driving controls, and some functions may be disabled once the signal to connect to the Internet is lost.

4. Database structure

The database of the "Driving Companion App" is performed through the Firebase cloud platform. It is a back-end cloud platform (BaaS: Backend-as-a-Service) that provides real-time database service. It is a NOSQL cloud database and supports Android, iOS and Web apps. It can save data in JSON format and synchronise to the online client in real-time. And even if the user is offline, the data is still available.

The main cloud data is divided into “Algorithms”, “Journeys” and “Users”. The details are as follows:

4.1 Algorithm data

It is mainly responsible for accessing the driving score scoring methods, and each scoring method has three factors that affect the driving score:

- **“Speeding”** : The percentage value of 5 levels exceeding the speed limit of the road is a List stored in int format.
- **“Accelerating”** : 5 levels of driving acceleration values, it is a List that stores values in int format.
- **”Decelerating”** : 5 levels of driving deceleration values, it is a List that stores the value in int format.

And each file is named by scoring: "Beginner", "Intermediate", and "Advanced", and their respective three-factor values are stored and read in List mode.

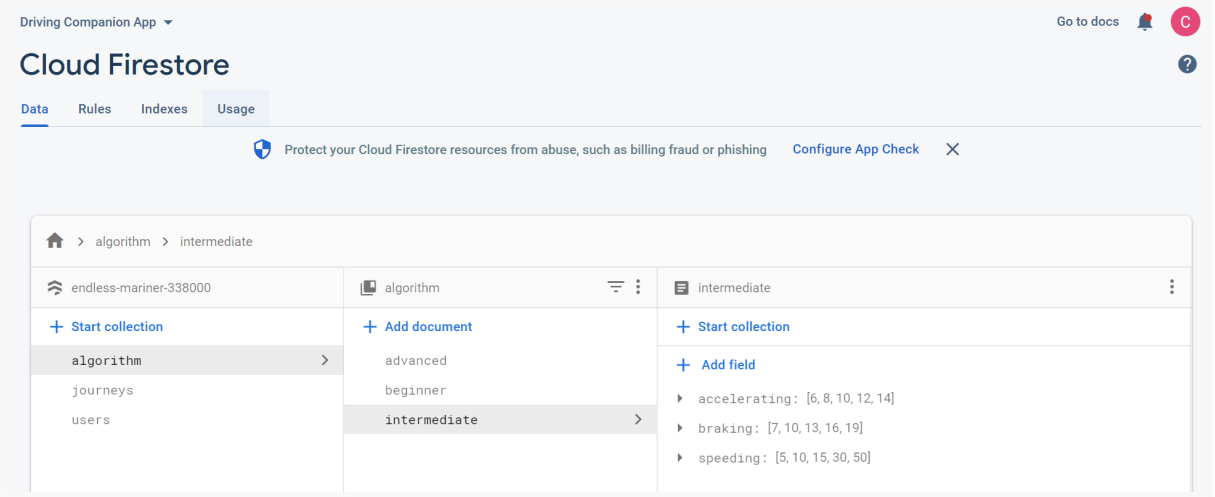


Fig 1 - Algorithm data structure

4.2 Journeys data

It is responsible for the driving data of all users. Each file is named with a unique automatic ID. The file name will not be repeated. Each file is an independent driving record, including:

- **”createdAt”** : The creation time and date of the driving record, stored in timestamp format.
- **“geoPoint”** : The coordinates of the geographic location, it is a List that stores the value of each geopoint format.
- **“speed”** : Driving speed, it is a List that stores each value in int format.
- **“time”** : Driving time, it is a List that stores each value in String format;
- **“userUID”** : The ID of the user to which the driving record belongs, stored as a value in String format.

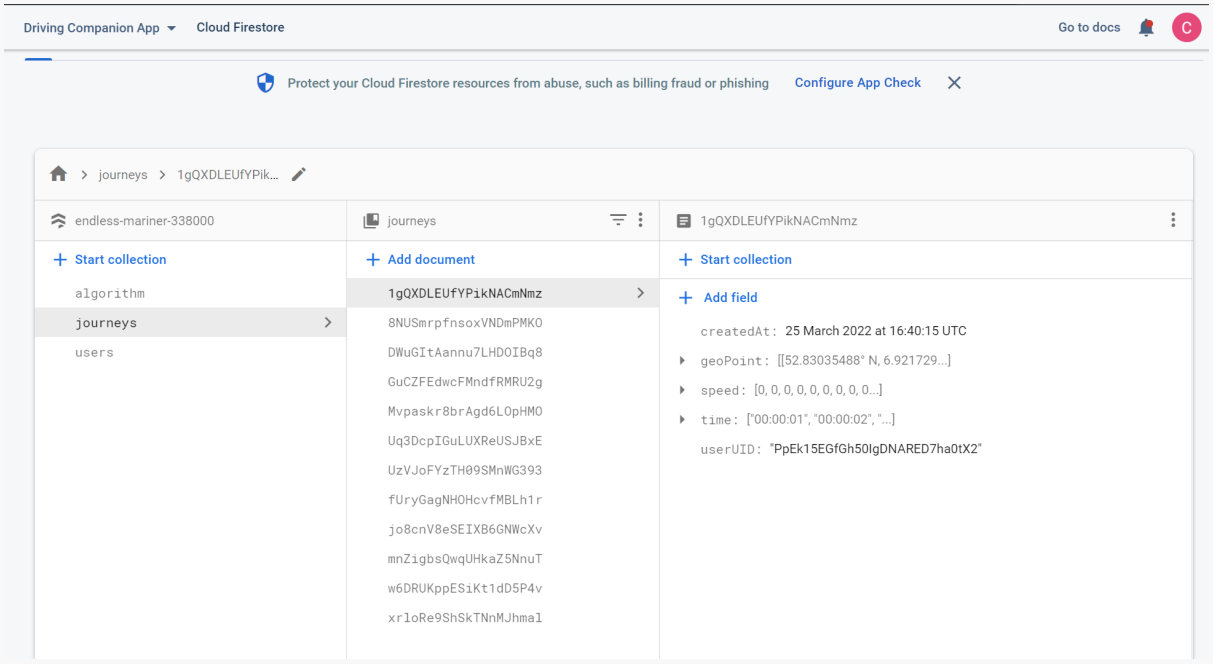


Fig 2 - Journeys data structure

4.3 Users data

It is the data responsible for all user information, each file is named with userUID (ie user id), and the data of each user includes:

“algorithm” : The scoring method selected by the user, the general default is "Intermediate", which is stored in String format.

“createdAt” : The date and time when the user was created, stored in timestamp format.

“email” : User's email address, stored in String format.

“firstname” : The real first name of the user, stored in String format.

“group” : The group to which the user belongs, generally divided into "admin" and "user", which are stored in String format.

“nickname” : The nickname of the user, used to avoid exposing the real name when ranking, it is stored in String format.

“surname” : The real surname of the user, stored in String format.

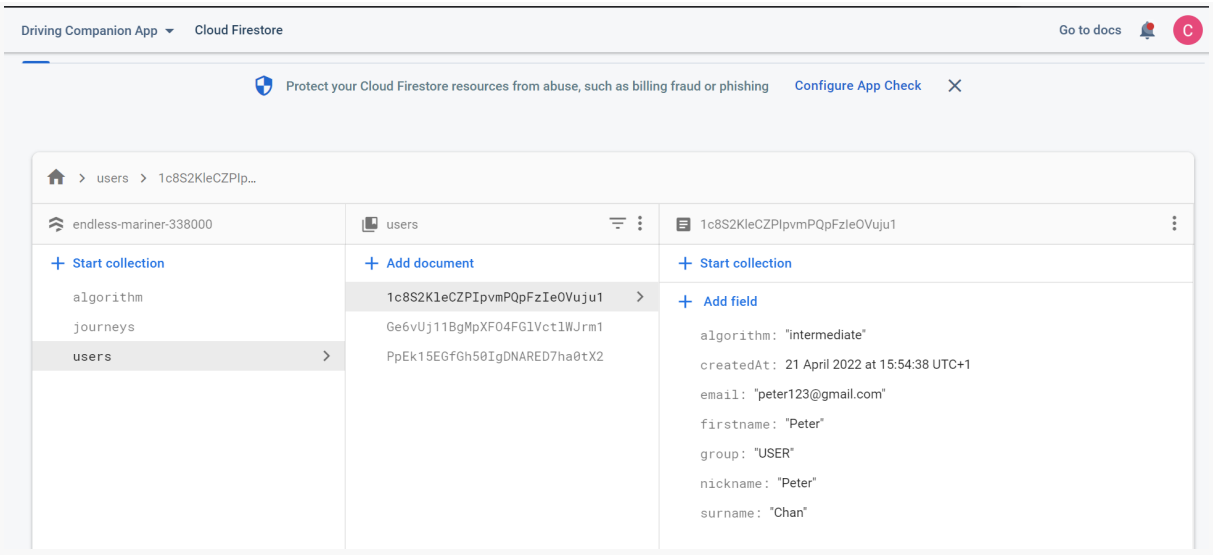


Fig 3 - Users data structure

In addition, the user account is managed by using Firebase's built-in Authentication system, which records the "User name", "Created date" , "Last Signed in date", "User UID" (automatically generated by the system).

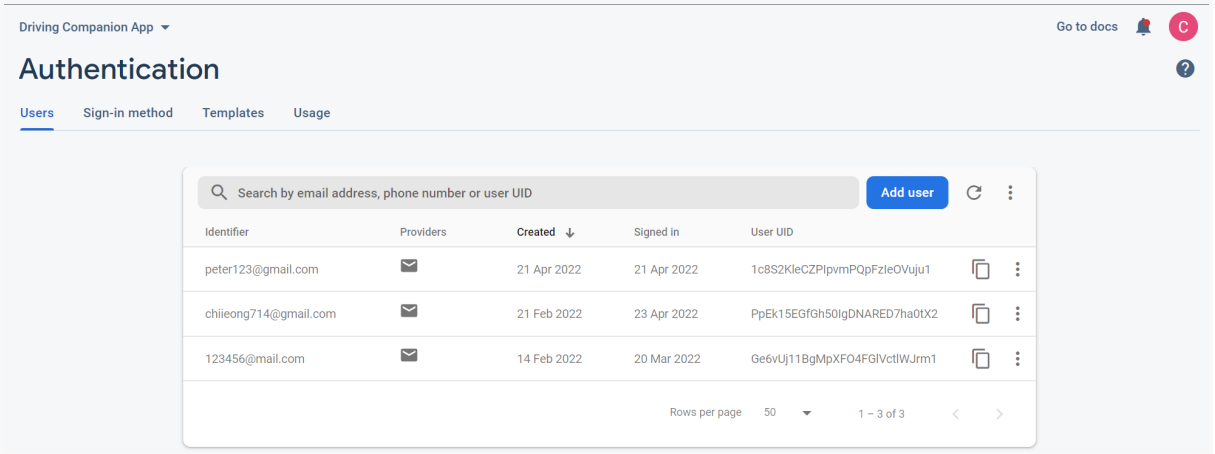


Fig 4 - Authentication data

5. Function description

The entire application is built with 16 dart files, each of which is responsible for the following:

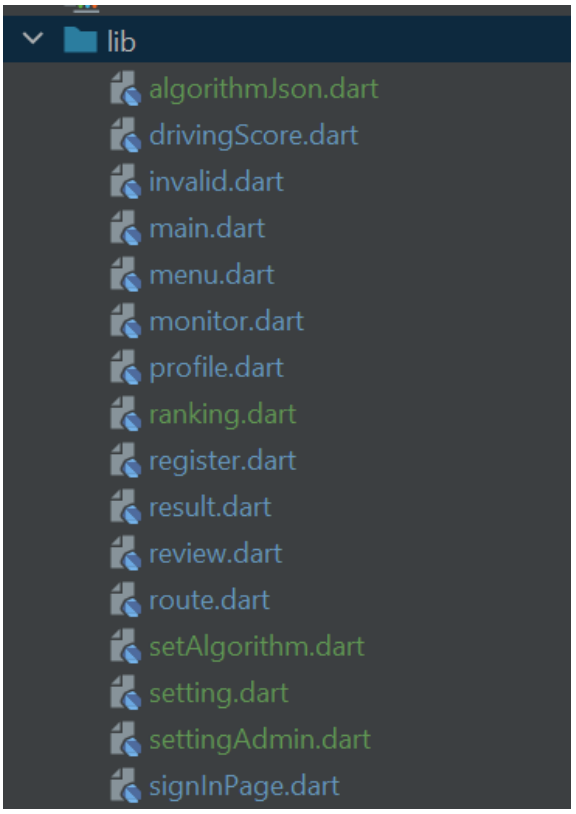


Fig 5 - Application structure

5.1 Main page

The "`main.dart`" is the first entry when the system opens the application. It has no UI. It is responsible for checking whether the user is logged in (`isLoggedIn = (user != null) ? false : user == null`), if the user is already logged in, It will read the user's data from the cloud, store the data in the local session for use, and then the system will move to `menu.dart` (Main Menu), if the user is not logged in, it will automatically enter `signInPage.dart` (login page), the user is required to log in to continue using the functions of this application.

```dart
class _MyState extends State<MyApp> {

  var isLoggedIn = (user != null) ? false: user == null;
  var userUID ="";

  @override
  void initState() {
    if (user != null) {
      getUserInfo(getUserUID());
    }
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: isLoggedIn ? SignInPage() : Menu()
    );  // MaterialApp
  }
```

Fig 6 - check user login code

```dart
Future<void> getUserInfo(String uid) async{
  var sessionManager = SessionManager();
  var collection = FirebaseFirestore.instance.collection('users');
  var document = await collection.doc(uid).get();

  if (document.exists) {
    Map<String, dynamic>? data = document.data();
    await sessionManager.set('firstname', data?['firstname']);
    await sessionManager.set('surname', data?['surname']);
    await sessionManager.set('nickname', data?['nickname']);
    await sessionManager.set('group', data?['group']);
    await sessionManager.set('algorithm', data?['algorithm']);
    Map<String, dynamic> rule = await setAlgorithmSession(await SessionManager().get('algorithm'));
    List<int> speeding = [];
    for (var i = 0; i < rule['speeding'].length; i++) {
      speeding.add(rule['speeding'][i].toInt());
    }
    List<int> braking = [];
    for (var i = 0; i < rule['braking'].length; i++) {
      braking.add(rule['braking'][i].toInt());
    }
    List<int> accelerating = [];
    for (var i = 0; i < rule['accelerating'].length; i++) {
      accelerating.add(rule['accelerating'][i].toInt());
    }
    AlgorithmJson scoring = AlgorithmJson(speeding,braking,accelerating);
    await sessionManager.set('scoring', scoring);
  }
}
```

Fig 7 - setup session code

5.2 Sign in page

It is the interface for sign in. The user needs to fill in the correct Email and password, and then press the "Sign In" button. If the user successfully logs in, the system will enter the main menu. If the user is a new user, he needs to press the "Register new account" button to enter `register.dart` to register as a new user.

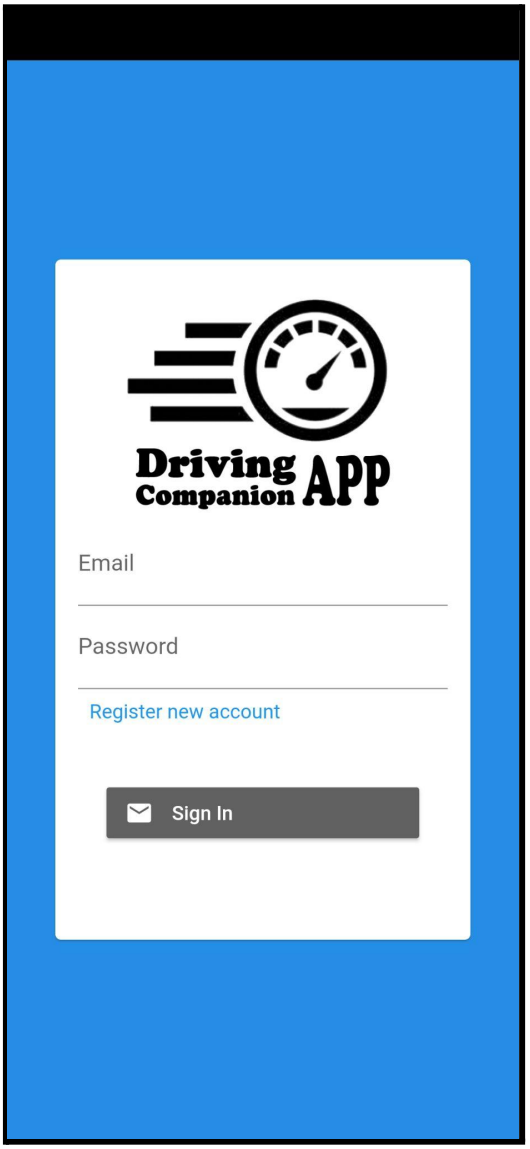


Fig 8 Login UI

```dart
Future<void> _signInWithEmailAndPassword() async {
  try {
    final User user = (await _auth.signInWithEmailAndPassword(
      email: _emailController.text,
      password: _passwordController.text,
    )).user!;
    Navigator.push(
        context, MaterialPageRoute(
        builder: (context) => Menu()
    ));  // MaterialPageRoute
  } catch (e) {
      print (e);
  }
}
```

Fig 9 - sign in and enter to menu() code

5.3 Register page

Users need to fill in all kinds of essential information and press the "Register" button to register. The Email address needs to be unique in the database.

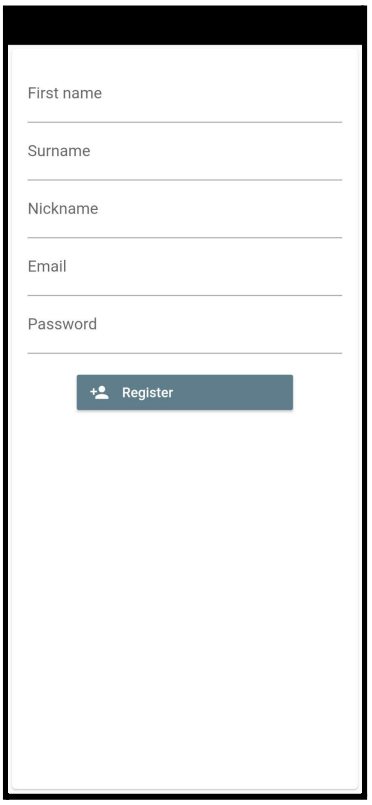


Fig 10 - register form

```
Future<void> saveData(String userUID, String email, String firstname, String surname, String nickname) async {

  DateTime now = DateTime.now();
  String timeInFormat = "${now.year.toString()}-${now.month.toString().padLeft(2,'0')}-${now.day.toString().pad
  DateTime savedTime = DateTime.parse(timeInFormat);

  CollectionReference firestore = FirebaseFirestore.instance.collection('users');

  return firestore
      .doc(userUID)
      .set({
        'algorithm' : 'intermediate',
        'createdAt' : savedTime,
        'email' : email,
        'firstname' : firstname,
        'group' : 'USER',
        'nickname' : nickname,
        'surname' : surname
      }
  )
      .then((value) => print("record saved"))
      .catchError((error) => print(error));
}
```

Fig 11 - save register data to cloud

5.4 Menu page

The role of the main menu is to provide a path to various functions, including:

- Monitor - Used to monitor driving, obtain driving data, and finally report driving score;
- Profile - Displays the user's comprehensive driving score and comprehensive data;
- History - Review the user's past driving records and driving report;
- Ranking - Compare users with other users by driving score;
- Setting - Let the user choose a scoring method;
- Sign out - sign out the current user.

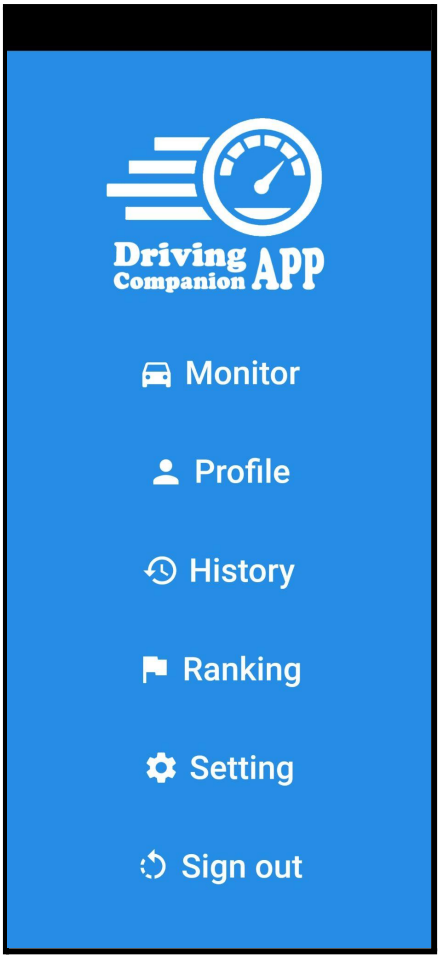


Fig 12 - Main Menu

5.5 Monitor page

The interface of real-time driving monitoring mainly displays a real-time map, real-time speed and timer.

- The real-time map shows the user's location during driving; it draws the driving route and marks the icon at the location of the user error.
- The real-time speed is updated every second, indicating the vehicle's current speed.
- The timer is to count the current driving time.
- After the user presses the green circular "Start" button at the bottom, it will turn red, and the monitoring will begin, getting real-time speed data every second in the backend.

- When the driving is finished, the user presses the red "Stop" button. The monitoring will end. All the obtained data will be sent to the following "Report" page for further analysis and scoring.

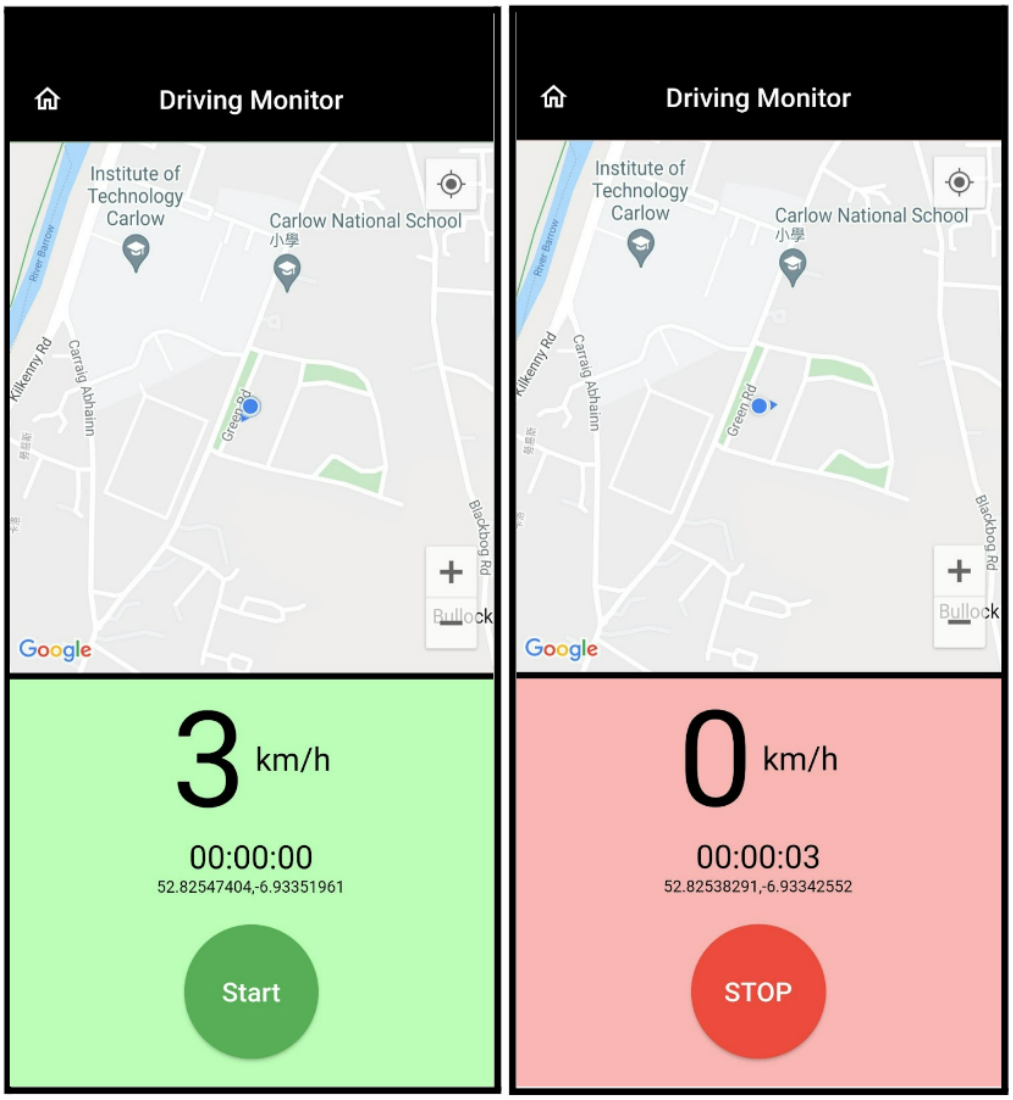


Fig 13 - Driving monitor UI

Before using the “monitor” function, the system will check whether the user's mobile phone has GPS permission. If not, a window will pop up asking the user to allow the system to obtain the mobile phone's GPS permission. If the user refuses, he will not be able to use this function.

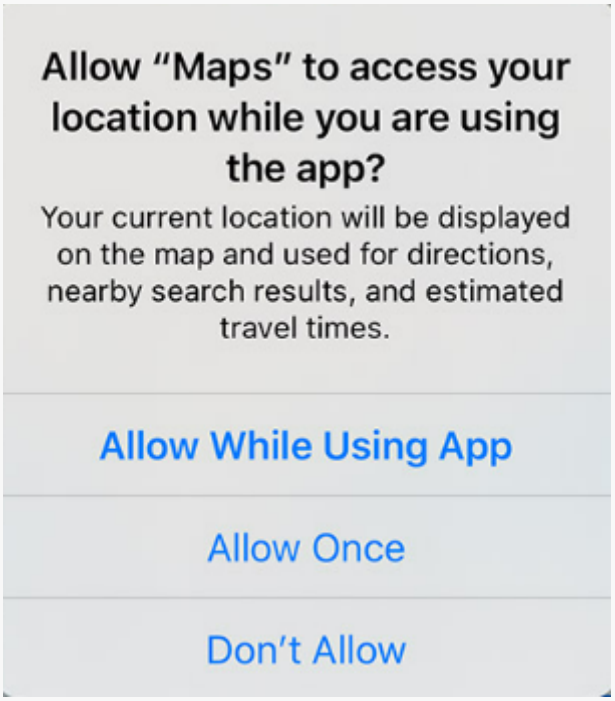


Fig 14 - Access location permission

```dart
checkGPSPermission() async {
  bool serviceEnabled;
  LocationPermission permission;
  serviceEnabled = await Geolocator.isLocationServiceEnabled();
  if (!serviceEnabled) {
    return Future.error('Location services are disabled.');
  }
  permission = await Geolocator.checkPermission();
  if (permission == LocationPermission.denied) {
    permission = await Geolocator.requestPermission();
    if (permission == LocationPermission.denied) {
      return Future.error('Location permissions are denied');
    }
  }
}
```

Fig 15 - check GPS permission code

Then the Start button is mainly responsible for changing the theme color from the original green to red, activating the timer, and turning on the map for real-time updates. It will start to draw blue lines for the route that has been driven, check for errors in real-time and display them on the map. In the backend, the system will collect three types of data, “time”, “speed”, and “geographic location”. These data will be collected and stored in a variable of type List as below:

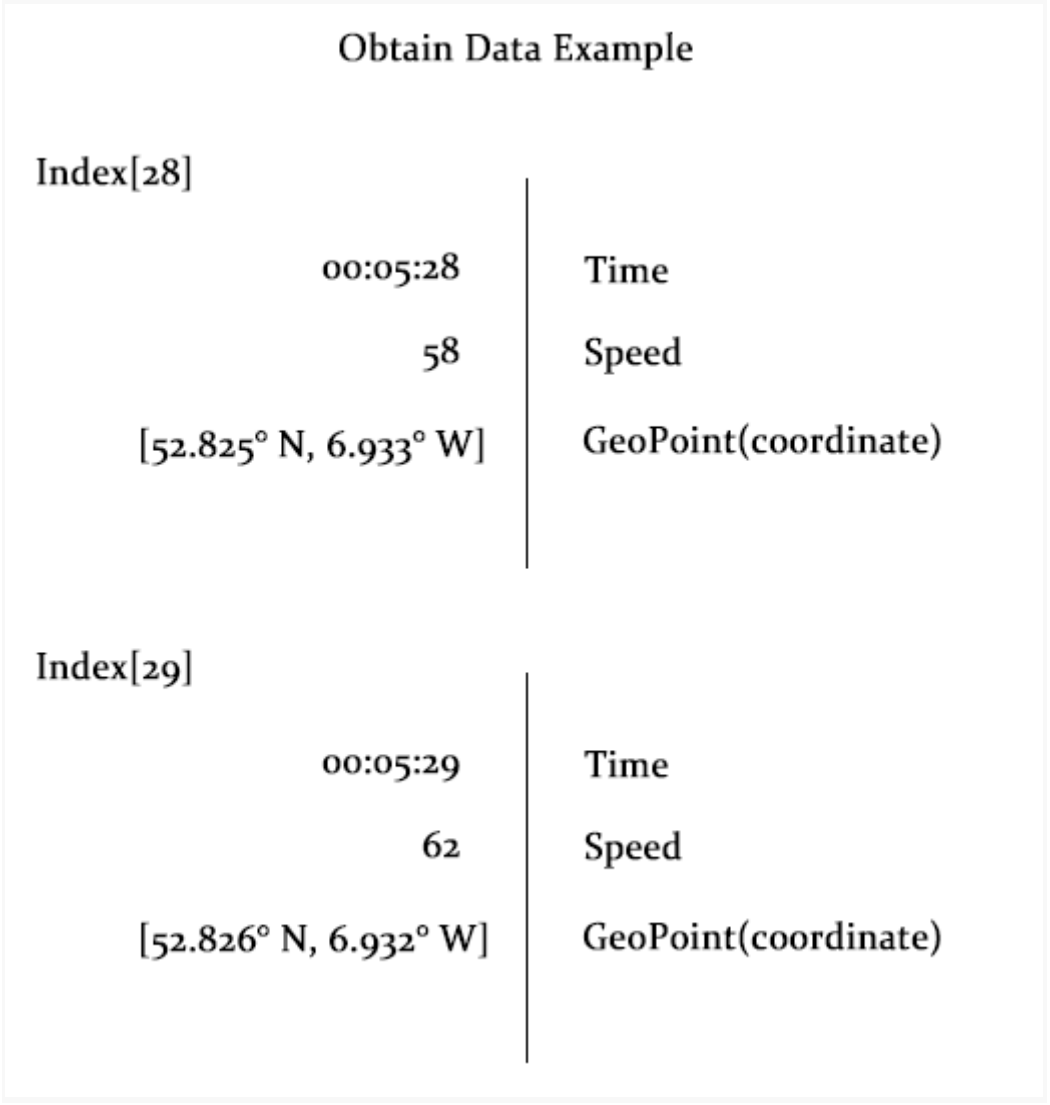


Fig 16 - Examples of data collected while driving

With these three types of data, the system can know the speed change from their previous index and the following index. The system can calculate the acceleration and deceleration. The system can also compare the speed limit of the road section to know the driver's driving speed. Have you exceeded the speed limit of the road?

```
startOrStop() {
  if(startStop) {
    watch.start();
    Wakelock.enable();
    setState(() {
      startStop = false;
      buttonText = "STOP";
      pressAttention = !pressAttention;
      backgroundChange = !backgroundChange;
    });
    timer = Timer.periodic(Duration(milliseconds: 1000), (Timer t) {
      googleMapUpdated();
      updateTime(timer);
      trip.add([elapsedTime,speedToInt,position]);
      setSpeedMarker();
      polylineCoordinates.add(
          LatLng(position.latitude, position.longitude)
      );
      Polyline polyline = Polyline(
          polylineId: PolylineId('poly'),
          color: polyColor,
          width: 6,
          points: polylineCoordinates
      );  // Polyline
      _polylines.add(polyline);
    });  // Timer.periodic
```

Fig 17 - Functions activated by the start button

```
getGeoLocation() async{
  _positionSubscription = await Geolocator.getPositionStream(
    locationSettings: locationSettings,
  ).listen((pos) {
    setState(() {
      position = pos;
      speedToInt = (position.speed * 3.85).toInt();
    });
  });
}
```

Fig 18 - get GPS location code

```
GoogleMap (
  myLocationEnabled: true,
  compassEnabled: true,
  tiltGesturesEnabled: false,
  mapType: MapType.normal,
  myLocationButtonEnabled: true,
  zoomControlsEnabled: true,
  zoomGesturesEnabled: true,
  scrollGesturesEnabled: true,
  rotateGesturesEnabled: true,
  polylines: _polylines,
  markers: markers,

  initialCameraPosition: CameraPosition(
    target: LatLng(position.latitude, position.longitude),
    zoom: mapZoom,
    tilt: mapTilt,
    bearing: position.heading
  ),  // CameraPosition

  onMapCreated: (GoogleMapController controller) {
    _controller.complete(controller);
  },
),  // GoogleMap
```

Fig 19 - google map code

```
Future<void> googleMapUpdated() async {
  final GoogleMapController controller = await _controller.future;
  controller.animateCamera (CameraUpdate.newCameraPosition(
      CameraPosition(
          target: LatLng(position.latitude, position.longitude),
          zoom: mapZoom,
          tilt: mapTilt,
          bearing: position.heading
      )  // CameraPosition
  ));
}
```

Fig 20 - real-time location update in google map

When the driver is driving under the monitoring function, the system will detect the driver's heavy acceleration or deceleration behavior. The system will immediately

display these errors on the real-time map. Heavy acceleration is the icon with an orange circular arrow up, and heavy deceleration is an icon with a red circular arrow down. The driver presses those icons to display the error levels, as shown below:

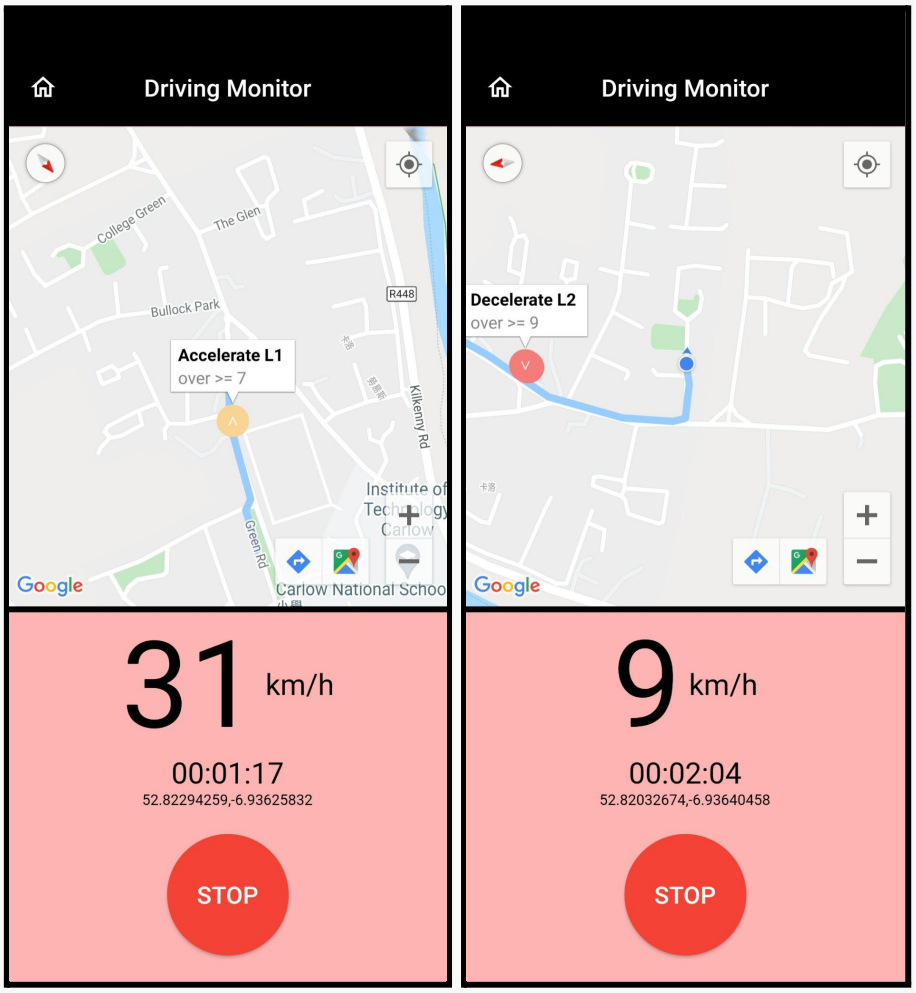


Fig 21 - Acceleration and deceleration errors are shown in the map

To effectively prevent some meaningless driving records, the minimum requirements for driving records are set to be 5 minutes and a driving distance of at least 1000 meters. `invalid.dart` is a page that notifies the user when the driver's driving trip is not recorded and prompts the conditions to meet the driving record requirement.

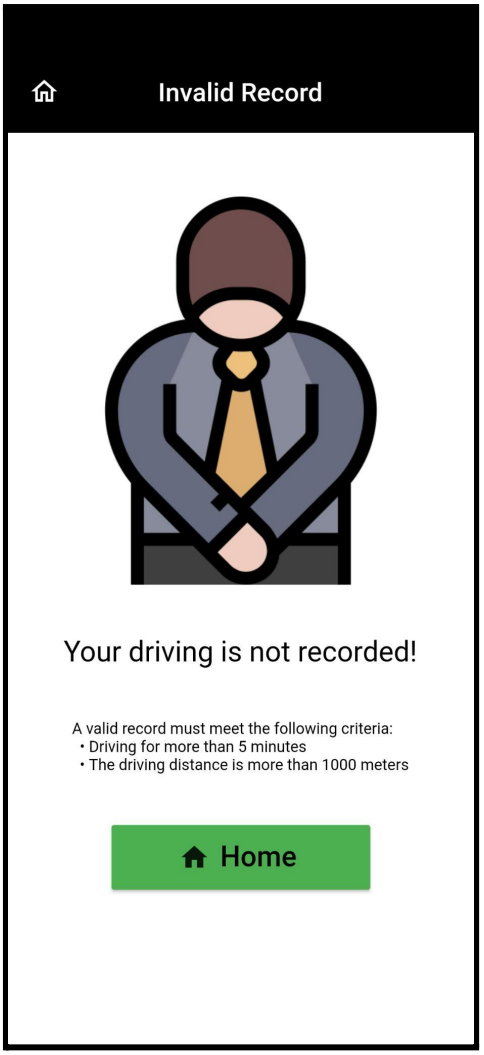


Fig 22 - invalid records

5.6 Report page

When the driver completes the driving monitor, it will show the driving score and short evaluation and the total driving time and distance traveled.

Short evaluations are divided into: Excellent - 90 points or more, Very Good - 80 points or more, Good - 70 points or more, Average - 60 points or more, Fair - 50 points or more, Poor - 40 points or more, Very Poor - 40 or less.

In addition, pressing the "Details" button will jump to another page, where the technical evaluation of speed, acceleration, and braking is displayed. The map can review the route of the driving journey and point out the error in the location.

Acceleration errors are orange, from shallow to deep according to the level, and deceleration errors are red, from shallow to deep according to the level. Click to see the time and speed changes.

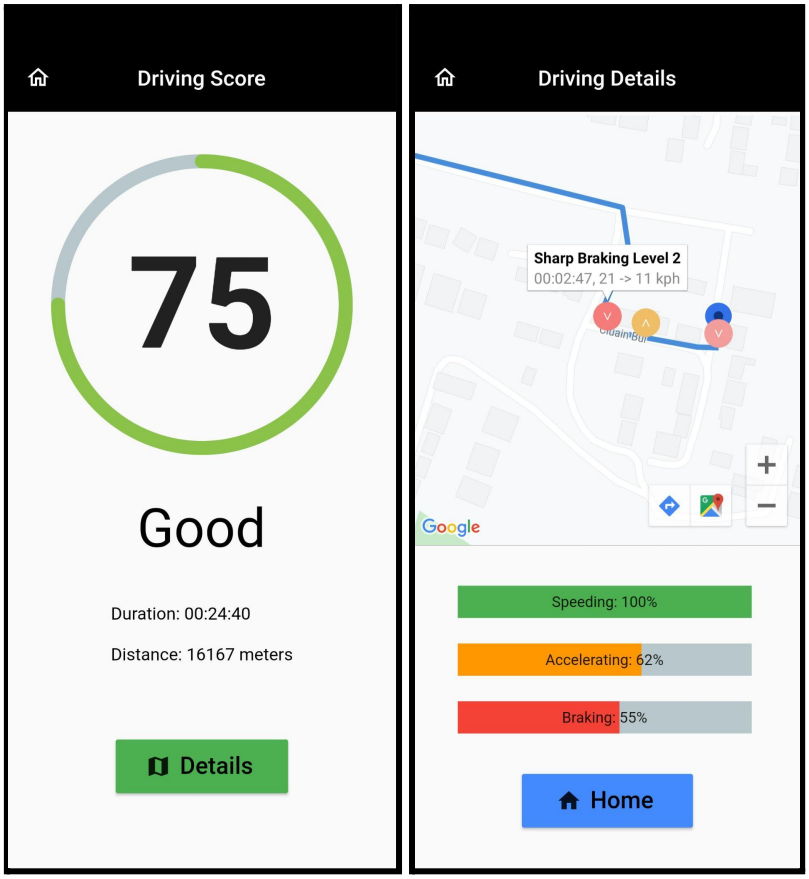


Fig 23 - Report UI

The driving score in this function is composed of "`drivingScore.dart`" and "`algorithmJson.dart`". "`drivingScore.dart`" is responsible for the calculation of the score, and "`algorithmJson.dart`" is responsible for converting the data. It is used to calculate the driving score (speeding, accelerating, braking) into Json Or List, Json is the required format for storing to Firebase, and List is the data format for calculating driving scores.

```
class AlgorithmJson {
  final List<dynamic> speeding;
  final List<dynamic> braking;
  final List<dynamic> accelerating;

  AlgorithmJson(this.speeding,this.braking,this.accelerating);

  AlgorithmJson.fromJson(Map<String,dynamic> json):
    speeding = json['speeding'],
    braking = json['braking'],
    accelerating = json['accelerating'];

  Map<String, dynamic> toJson() {
    final Map<String, dynamic> data = new Map<String, dynamic>();
    data['speeding'] = this.speeding;
    data['braking'] = this.braking;
    data['accelerating'] = this.accelerating;
    return data;
  }
}
```

Fig 24 - Convert between Json and List

Calculation of driving score is mainly influenced by three factors, including "Speeding", "Acceleration", and "Deceleration". According to the GPS positioning of the mobile phone, when the system obtains the two coordinates, then the distance can be calculated, and with the distance and time, the speed can be obtained, the formula is as follows:

- **speed =** `distance ÷ time`
- **distance =** `speed × time`
- **time =** `distance ÷ speed`

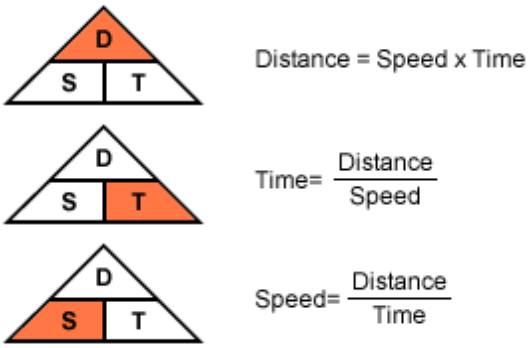


Fig 25 - The speed, distance, time relationship

After the system gets the value of the speed, the acceleration and deceleration are calculated from the speed between the two times.

if the result of `(current speed - previous speed) ÷ time` is:

- Positive value means speed of acceleration
- Negative value means speed of deceleration

The data used in the final calculation of the driving score are the following three:

Speeding: It is the driving speed, which is mainly about the driving speed and the speed limit of the road. If the user's current driving speed exceeds the speed limit specified by the road, the system will deduct points and divide them into five grades according to the percentage of the exceeding speed. Level 1 is 5% over the speed limit; Level 2 is 10% over the speed limit; level 3 is 15% over the speed limit; level 4 is 30% over the speed limit; level 5 Level is 50% of the speed over the speed limit. The higher the level, the more points will be deducted.

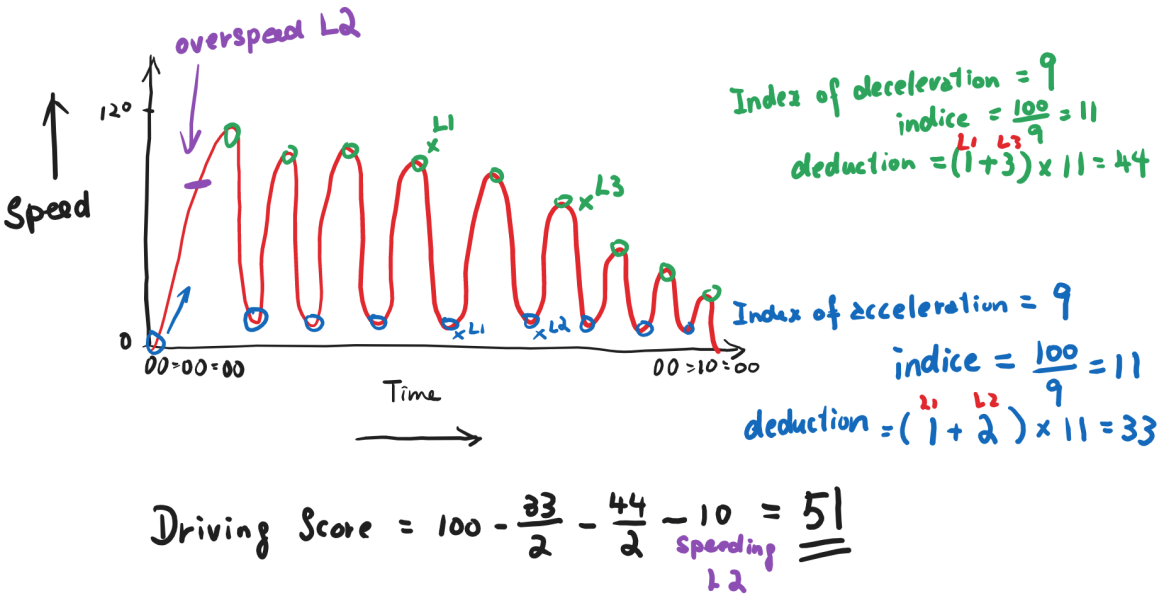
Acceleration: it is the acceleration of driving, according to the data obtained from the speed of the previous second and the increase of the current speed, according to the change of the acceleration of one second, divided into five levels; the first level is the speed increase of 6km/h per second, the second Level 1 is an increase of 8km/h per second, level 3 is an increase of 10km/h per second, level 4 is a speed increase of 12km/h per second, and level 5 is a speed increase of 14km/h per second. The higher the level, the more points will be deducted.

Deceleration: It is the deceleration of driving, according to the data obtained from the speed of the previous second and the reduction of the current speed, it is divided

into five levels according to the change of the deceleration of one second. The first level is the speed reduction of 7km/h per second. The second level is a reduction of 10km/h per second, the third level is a reduction of 13km/h per second, the fourth level is a reduction of 16km/h per second, and the fifth level is a reduction of 19km/h per second. The higher the level, the more points will be deducted. The specific number of deductions will be explained later.

The scoring calculation method first gives 100 total points and then deducts points from the user's mistakes according to the level of the above three factors. In terms of acceleration and deceleration, the system will calculate the total index of acceleration and deceleration. These indices will be proportional, and each has a certain percentage of points. For example, if the acceleration index is 20 (The algorithm for deceleration is the same), its proportion is calculated as 100%, and each index worth 5%. If the user makes mistakes in the corresponding index, points will be deducted according to the error level. If it is level 1 error, it will lose 1. Level 2 error loses 2, level 3 error loses 3, and level 4 error loses 4. Level 5 error loses 5. Finally, these deductions are multiplied by the percentage of the single index, divided by the Total Index, and multiplied by 100.

For example, If the total error of acceleration level adds up to 3, and the total acceleration index is 9, so each indice will be 11, then total error 3 will be multiplied by indice (11) and come out 33, Since acceleration and deceleration each account for 50% of the score, divide this number by 2. So, finally, the deduction for acceleration is 16.5. Then assume that the “deceleration” deduction is 22 and the “speeding” deduction is 10. Finally the acceleration deduction and deceleration deduction will directly subtract from the original score of 100, come out 100 - 16.5 - 22 - 10 = 51(int), The result of the driving score is 51 points.



Although the driving score has been tested many times, it may not be perfect. In order to take this into account, the user can select the driving classification according to their own driving score requirements in the settings of this program. It is divided into "Beginner", "Intermediate", and "Advanced". The previous rating is "Intermediate", which is the default level for all users, "Beginner" has lower requirements than "Intermediate", and "Advanced" has higher requirements than "Intermediate". Acceleration and deceleration are more sensitive than other methods. Suppose these three settings are not perfect if the user has admin permission. In that case, the admin users can change the values of these three factors in the settings of this application by themselves and by modifying these values to get the most satisfactory score calculation method.

```dart
Future<int> getDrivingScore(AlgorithmJson json, List<dynamic> list) async{
  Map<String, dynamic> rule = {};
  List<int> speeding = [];
  List<int> braking = [];
  List<int> accelerating = [];
  for (var i = 0; i < json.speeding.length; i++) {
    speeding.add(json.speeding[i].toInt());
  }
  for (var i = 0; i < json.braking.length; i++) {
    braking.add(json.braking[i].toInt());
  }
  for (var i = 0; i < json.accelerating.length; i++) {
    accelerating.add(json.accelerating[i].toInt());
  }
  rule['speeding'] = speeding;
  rule['braking'] = braking;
  rule['accelerating'] = accelerating;
  return drivingScoreAlgorithm(rule,list);
}
```

Fig 26 - prepare data format for driving score calculation

```dart
Future<int> drivingScoreAlgorithm(Map<String, dynamic> rule,List<dynamic> list) async {

  var drivingScore = 100;
  List<int> speeding = rule['speeding'];
  List<int> accelerating = rule['accelerating'];
  List<int> braking = rule['braking'];

  int sharpAccelerated = getSharpAccelerated(accelerating,list);
  int sharpDecelerated = getSharpDecelerated(braking,list);
  int overSpeed = await getOverSpeeding(speeding, list);
  print("overspeed: $overSpeed");

  drivingScore = drivingScore - sharpAccelerated - sharpDecelerated - overSpeed;

  //no negative score
  if (drivingScore < 0) {
    drivingScore = 0;
  }

  return drivingScore.toInt();
}
```

Fig 27 - Driving score calculation

Acceleration and deceleration, the system will accumulate the speed peak for the entire driving trip and use the percentage of its accumulated total to determine what percentage of each acceleration or deceleration point is accounted for.

```
int getSharpAccelerated(List<int> accelerating,List<dynamic> list) {

  var indexOfSharpAccelerated = 1;
  bool wavePeak = false;
  var level1 = 0;
  var level2 = 0;
  var level3 = 0;
  var level4 = 0;
  var level5 = 0;

  for (var i = 0; i < list.length -1; i++) {

    //speed wave peak
    if (wavePeak == false && list[i][1] - list[i+1][1] >= 1 ) {wavePeak = true;}
    else if (wavePeak == true && list[i+1][1] - list[i][1] >= 1 ) {
      indexOfSharpAccelerated++;
      wavePeak = false;
    }

    if (list[i+1][1] - list[i][1] >= accelerating[4]) {level5++;}
    else if (list[i+1][1] - list[i][1] >= accelerating[3]) {level4++;}
    else if (list[i+1][1] - list[i][1] >= accelerating[2]) {level3++;}
    else if (list[i+1][1] - list[i][1] >= accelerating[1]) {level2++;}
    else if (list[i+1][1] - list[i][1] >= accelerating[0]) {level1++;}
  }

  var indice = 100 /indexOfSharpAccelerated;
  var mistakes = level1 + (level2 * 2) + (level3 * 3) + (level4 * 4) + (level5 * 5);
  var deduction= (mistakes * indice) / 2;
  return deduction.toInt();
}
```

Fig 28 - Sharp Acceleration deduction

```
int getSharpDecelerated(List<int> braking,List<dynamic> list) {

  var indexOfSharpDecelerated = 1;
  bool wavePeak = false;
  var level1 = 0;
  var level2 = 0;
  var level3 = 0;
  var level4 = 0;
  var level5 = 0;

  for (var i = 0; i < list.length -1; i++) {

    //speed wave peak
    if (wavePeak == false && list[i+1][1] - list[i][1] >= 1 ) {
      wavePeak = true;
    }
    else if (wavePeak == true && list[i][1] - list[i+1][1] >= 1 ) {
      indexOfSharpDecelerated++;
      wavePeak = false;
    }

    if (list[i][1] - list[i+1][1] >= braking[4]) {level5++;}
    else if (list[i][1] - list[i+1][1] >= braking[3]) {level4++;}
    else if (list[i][1] - list[i+1][1] >= braking[2]) {level3++;}
    else if (list[i][1] - list[i+1][1] >= braking[1]) {level2++;}
    else if (list[i][1] - list[i+1][1] >= braking[0]) {level1++;}
  }

  var indice = 100 / indexOfSharpDecelerated;
  var mistakes = level1 + (level2 * 2) + (level3 * 3) + (level4 * 4) + (level5 * 5);
  var deduction= (mistakes * indice) / 2;
  return deduction.toInt();
}
```

Fig 29 - Sharp Deceleration deduction

Regarding the level definition of acceleration and deceleration, as mentioned earlier, the system itself provides three scoring options for users, including "Beginner", "Intermediate", and "Advanced".

The following are the acceleration and deceleration values for different levels:

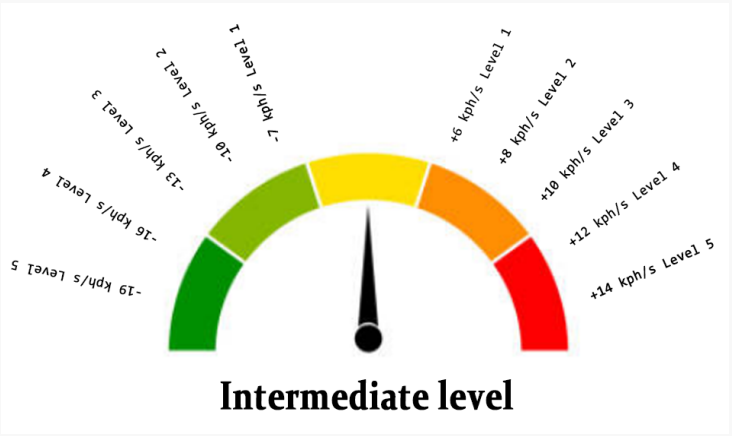


Fig 30 - The acceleration and deceleration level of Intermediate

Fig 31 - The acceleration and deceleration level of Intermediate

Fig 32 - The acceleration and deceleration level of Intermediate

The “Intermediate” level is the default scoring method for all users. After many tests, more reliable values can be found, and there may be better values with more experience in the future. Therefore, the system allows Admin users to modify these values.

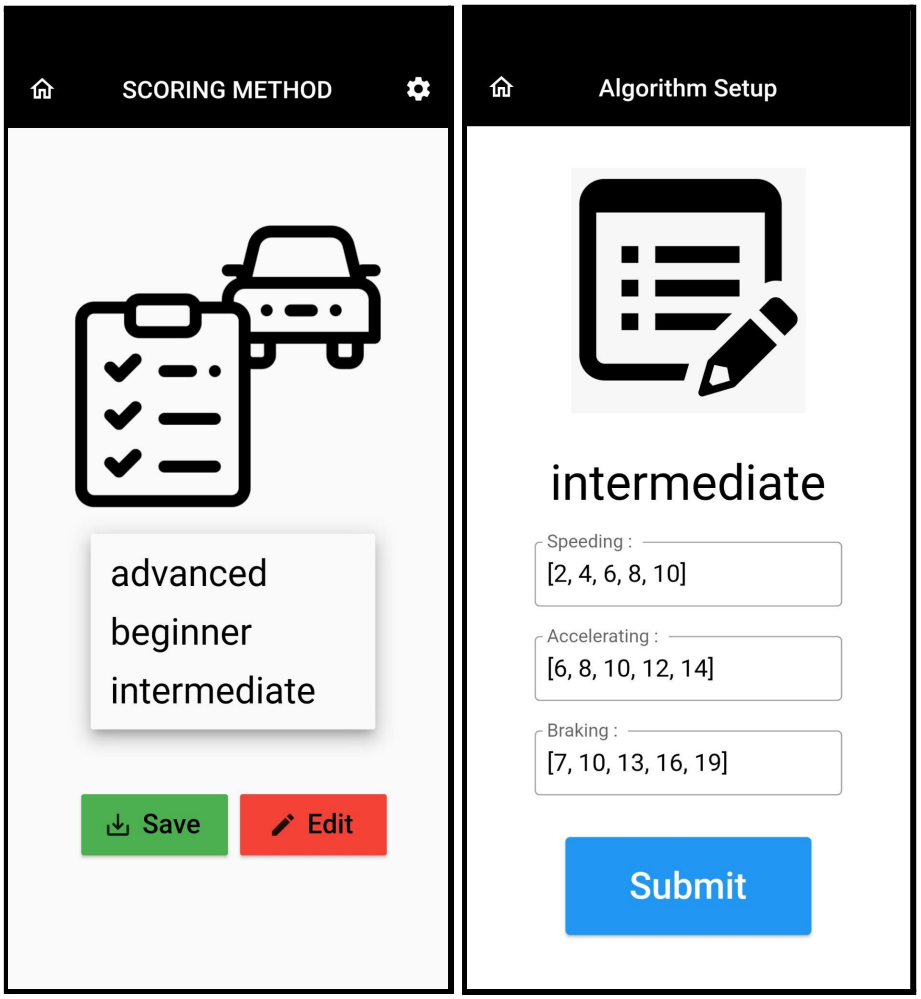


Fig 33 - Select scoring method and admin modification

The road speed limit is a List divided as Motorways and National Road. To realize the speed limit setting for all roads, it will need to classify the road name and speed limit and create a new List variable. The following is the geographic location information obtained by Google API.

```
{
   "plus_code" : {
      "compound_code" : "46RP+659 Kineagh Cross, County Kildare, Ireland",
      "global_code" : "9C5M46RP+659"
   },
   "results" : [
      {
         "address_components" : [
            {
               "long_name" : "M9",
               "short_name" : "M9",
               "types" : [ "route" ]
            },
            {
               "long_name" : "County Kildare",
               "short_name" : "County Kildare",
               "types" : [ "administrative_area_level_1", "political" ]
            },
            {
               "long_name" : "Ireland",
               "short_name" : "IE",
               "types" : [ "country", "political" ]
            }
         ],
         "formatted_address" : "M9, Co. Kildare, Ireland",
         "geometry" : {
            "bounds" : {
               "northeast" : {
                  "lat" : 53.1607824,
                  "lng" : -6.7516037
               },
               "southwest" : {
                  "lat" : 53.1405222,
                  "lng" : -6.7645559
               }
            },
            "location" : {
               "lat" : 53.1497964,
               "lng" : -6.7533514
            },
            "location_type" : "GEOMETRIC_CENTER",
            "viewport" : {
               "northeast" : {
                  "lat" : 53.1607824,
                  "lng" : -6.7516037
               },
               "southwest" : {
                  "lat" : 53.1405222,
                  "lng" : -6.7645559
               }
            }
         },
         "place_id" : "ChIJ_Znq54yAZ0gRcJY-O0kI52Y",
         "types" : [ "route" ]
      },
```


Fig 34, HTTP response from Google API in Json format

When the system calls an HTTP request for coordinates, Google API will respond to the relevant information of the coordinates, including the address name, Eircode and other valuable data. It will respond to the application in the format of Json, and then read it and find out where it's on the road. Since Motorways and National Road are the most manageable speed limits to correspond to, Motorways have a speed limit of 120km/h, and National Road has a speed limit of 100km/h. Moreover, the number of these two types of roads is not large, which is easy to be manually written into the variables of the two lists. Every time the speed is compared, the current position is found from the two lists to see if the current position is on the two types of roads. Thereby it can be determined whether the user has exceeded the speed limit on these two types of roads. But there is too much data on other roads. It takes a lot of time to collect these road names and its speed limit, especially for some Regional

roads with "R" head letter, such as R448, its speed limit will be different in different locations, it crosses the city center. It could be 50km/h, or 60km/h. After leaving the city center, it could be 80km/h or 100km/h, which makes it difficult to define its speed limit.

```
Future<int> getOverSpeeding(List<int> speeding, List<dynamic> list) async{
  int result = 0;
  List<dynamic> motorways = ['M1','M2','M3','M4','M6','M7','M8','M9','M11','M17','M18','M20','M50'];
  List<dynamic> nationalRoad = ['N1','N2','N3','N4','N5','N6','N7','N8','N9','N10','N11','N12','N13','N14','N15','N16','N17','N18',
  List<dynamic> specifiedRoad = ['Green Rd'];
  int highestSpeed = 0;
  int index = 0;
  for(var i = 0; i < list.length; i++) {
    if (list[i][1] > highestSpeed) {
      highestSpeed = list[i][1];
      index = i;
    }
  }
  String postsURL = "https://maps.googleapis.com/maps/api/geocode/json?latlng=${list[index][2].latitude},${list[index][2].longitude}
  Response res = await get(Uri.parse(postsURL));
  if (res.statusCode == 200) {
    Map<String, dynamic> geocode = jsonDecode(res.body);
    for (var j = 0; j < geocode['results'].length; j++) {
      var address = geocode['results'][j]['address_components'][0]['short_name'].toString();
      print('address: ${address}');
      //motorways
      if(motorways.contains(address)) {
        if ((list[index][1] * (100 / 120)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 120)) - 100 >= speeding[3]) { result += 30;}
        else if ((list[index][1] * (100 / 120)) - 100 >= speeding[2]) { result += 15;}
        else if ((list[index][1] * (100 / 120)) - 100 >= speeding[1]) { result += 10;}
        else if ((list[index][1] * (100 / 120)) - 100 >= speeding[0] ) { result += 5;}
        break;
      }
      //rationalRoad
      else if(nationalRoad.contains(address)) {
        if ((list[index][1] * (100 / 100)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 100)) - 100 >= speeding[3]) { result += 30;}
        else if ((list[index][1] * (100 / 100)) - 100 >= speeding[2]) { result += 15;}
        else if ((list[index][1] * (100 / 100)) - 100 >= speeding[1]) { result += 10;}
        else if ((list[index][1] * (100 / 100)) - 100 >= speeding[0]) { result += 5;}
        break;
      }
      //Regional roads
      else if(address.length == 4 && address[0] == 'R' && double.tryParse(address[1]) != null && double.tryParse(address[2]) != null
        if ((list[index][1] * (100 / 80)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 80)) - 100 >= speeding[3]) { result += 30;}
        else if ((list[index][1] * (100 / 80)) - 100 >= speeding[2]) { result += 15;}
        else if ((list[index][1] * (100 / 80)) - 100 >= speeding[1]) { result += 10;}
        else if ((list[index][1] * (100 / 80)) - 100 >= speeding[0]) { result += 5;}
      }
      //specificed speed road
      else if(specifiedRoad.contains(address)) {
        if ((list[index][1] * (100 / 50)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[3]) { result += 30;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[2]) { result += 15;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[1]) { result += 10;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[0]) { result += 5;}
        break;
      }
      //city max speed
      else {
        if ((list[index][1] * (100 / 50)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[3]) { result += 30;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[2]) { result += 15;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[1]) { result += 10;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[0]) { result += 5;}
        break;
      }
    }
  }
  else { throw "Unable to retrieve posts.";}
  return result;
}
```

Fig 35 - Speeding deduction

```
String getEvaluate(int drivingScore) {
  if (drivingScore >= 90) {
    return "Excellent";
  }
  else if (drivingScore >= 80) {
    return "Very Good";
  }
  else if (drivingScore >= 70) {
    return "Good";
  }
  else if (drivingScore >= 60) {
    return "Average";
  }
  else if (drivingScore >= 50) {
    return "Fair";
  }
  else if (drivingScore >= 40) {
    return "Poor";
  }
  else {
    return "Very Poor";
  }
}
```

Fig 36 - Driver evaluation

5.7 Profile page

The profile shows the current user's comprehensive evaluation, calculated based on the last ten driving records. It shows a linear graph of score changes, where users can see their driving improvement, total driving times, total driving time and total driving distance.

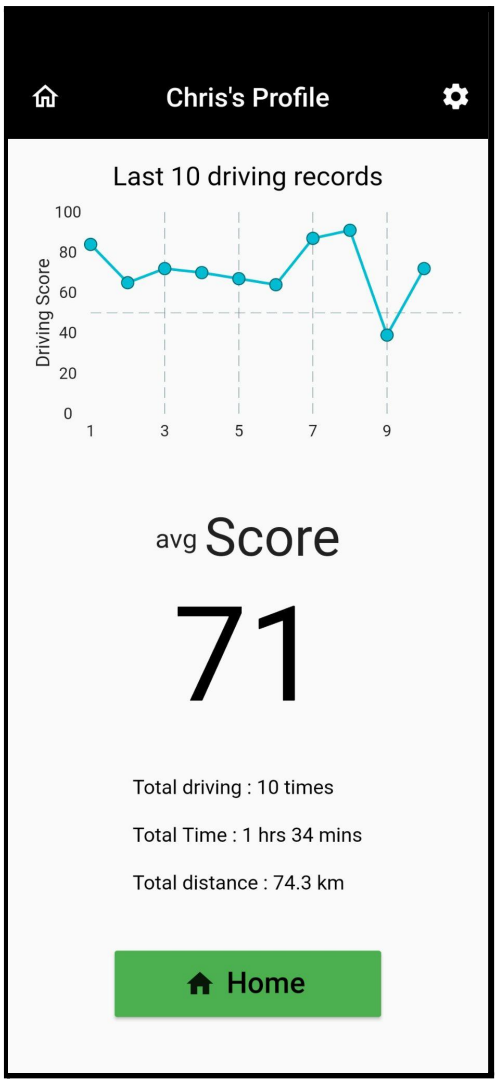


Fig 37 - Profile UI

5.8 History page

Review can review the past driving records of the current user. It displays in the form of a list and supports gestures to drag up and down. Each row is a driving record. The short display information includes the driving date, driving time and driving distance. Press the "Details" button to move to `report.dart` to display the driving report for that record.

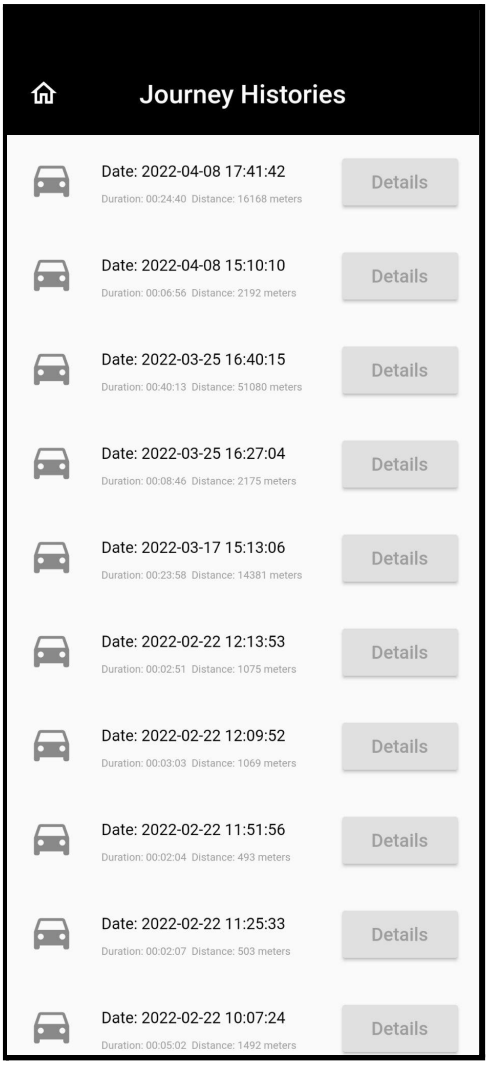


Fig 38 - review driving records

In “`review.dart`”, the system calls out all the journeys documents with the same userUID as the current user. It calculates “total time” and “total driving distance” from each of the driving data "Speeding", "Accelerating", and "Breaking". Those records will be sorted by date. The newest record will be placed at the top first, and the "Details" button will send the list of the journeys document to "`report.dart`" for processing.

```dart
Future<void> getData(String documentID) async {
  trip = [];
  List<dynamic> time = [];
  List<dynamic> speed = [];
  List<dynamic> location = [];

  var collection = FirebaseFirestore.instance.collection('journeys');
  var document = await collection.doc(documentID).get();

  if (document.exists) {
    Map<String, dynamic>? data = document.data();
    //test3 = document.data();

    data?["time"].forEach((v) => time.add(v));
    data?["speed"].forEach((v) => speed.add(v));
    data?["geoPoint"].forEach((v) => location.add(v));

    for (var i = 0; i < time.length; i++) {
      trip.add([time[i],speed[i],location[i]]);
    }
  }
}
```

Fig 39 - retrieve specific document of driving record data

The driving distance is calculated from the GeoPoint List. Between each index, it is calculated and accumulated one by one, accurately estimating the actual driving distance because the car does not go straight forward in the endpoint direction. It sometimes turns, Or return less, or takes some unnecessary roads.

```dart
int getTripDistance(List<dynamic> list) {
  var distance = 0.0;
  var totalDistance = 0.0;
  var currentlatitude = 0.0;
  var currentlongitude = 0.0;
  var lastlatitude = 0.0;
  var lastlongitude = 0.0;
  for (var i = 0; i < list.length -2; i++) {
    lastlatitude = list[i].latitude;
    lastlongitude = list[i].longitude;
    currentlatitude = list[i+1].latitude;
    currentlongitude = list[i+1].longitude;
    if (lastlatitude != 0.0 || lastlongitude != 0.0 || currentlatitude != 0.0 || currentlongitude != 0.0) {
      distance = Geolocator.distanceBetween(
          lastlatitude,lastlongitude,
          currentlatitude,currentlongitude
      );
      totalDistance += distance;
    }
  }
  return totalDistance.toInt();
}
```

Fig 40 - How to Calculate Driving Distance

5.9 Ranking page

In order to encourage drivers to improve their driving behavior, the Ranking system allows drivers to actively improve their scores to improve their rankings through the comparison of driving scores between users. This system also allows drivers to know which level their driving skills belong.

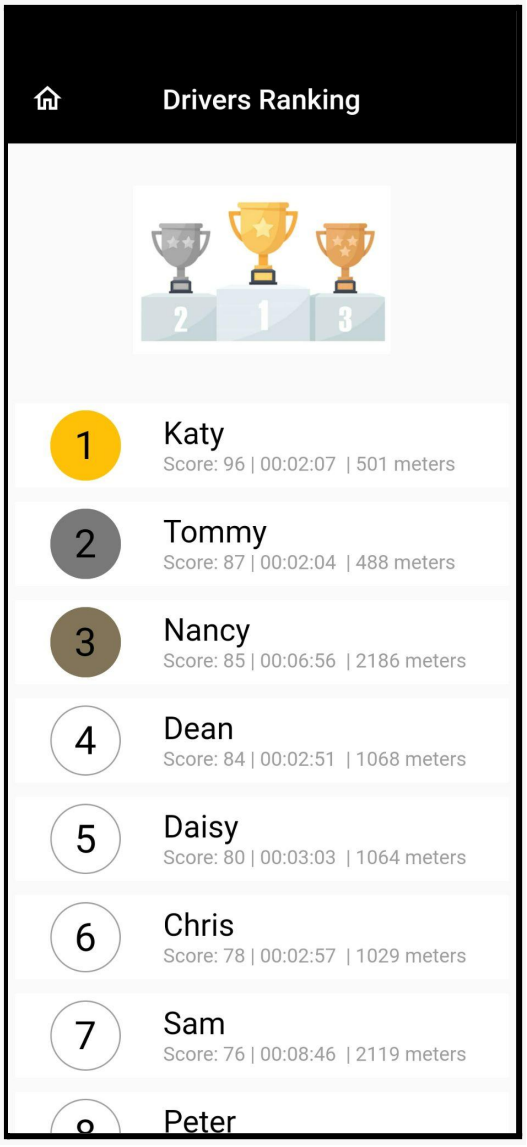


Fig 41 - Ranking system

6. Code

The section is about all the codes of this application; the previous part explains some important codes, and this part includes other codes and UI codes besides the main code.

6.1 algorithmJson.dart

Description: It is responsible for converting Map type variables and Json Format.

```
class AlgorithmJson {
 final List<dynamic> speeding;
 final List<dynamic> braking;
 final List<dynamic> accelerating;

 AlgorithmJson(this.speeding,this.braking,this.accelerating);

 AlgorithmJson.fromJson(Map<String,dynamic> json):
   speeding = json['speeding'],
   braking = json['braking'],
   accelerating = json['accelerating'];

 Map<String, dynamic> toJson() {
   final Map<String, dynamic> data = new Map<String, dynamic>();
   data['speeding'] = this.speeding;
   data['braking'] = this.braking;
   data['accelerating'] = this.accelerating;
   return data;
 }
}
```

6.2 drivingScore.dart

Description: It is responsible for calculating the driving score and detecting acceleration, deceleration, and speed. It also calculates the maximum speed, average speed, and total driving distance and evaluates the user's driving level.

```
import 'package:geolocator/geolocator.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'dart:convert';
import 'package:http/http.dart';
import 'algorithmJson.dart';

Future<int> getDrivingScore(AlgorithmJson json, List<dynamic> list) async{
 Map<String, dynamic> rule = {};
 List<int> speeding = [];
 List<int> braking = [];
 List<int> accelerating = [];
 for (var i = 0; i < json.speeding.length; i++) {
   speeding.add(json.speeding[i].toInt());
```

```
 }
 for (var i = 0; i < json.braking.length; i++) {
   braking.add(json.braking[i].toInt());
 }
 for (var i = 0; i < json.accelerating.length; i++) {
   accelerating.add(json.accelerating[i].toInt());
 }
 rule['speeding'] = speeding;
 rule['braking'] = braking;
 rule['accelerating'] = accelerating;
 return drivingScoreAlgorithm(rule,list);
}

Future<int> drivingScoreAlgorithm(Map<String, dynamic> rule,List<dynamic> list)
async {

 var drivingScore = 100;
 List<int> speeding = rule['speeding'];
 List<int> accelerating = rule['accelerating'];
 List<int> braking = rule['braking'];

 int sharpAccelerated = getSharpAccelerated(accelerating,list);
 int sharpDecelerated = getSharpDecelerated(braking,list);
 int overSpeed = await getOverSpeeding(speeding, list);
 print("overspeed: $overSpeed");

 drivingScore = drivingScore - sharpAccelerated - sharpDecelerated - overSpeed;

 //no negative score
 if (drivingScore < 0) {
   drivingScore = 0;
 }
 return drivingScore.toInt();
}

int getSharpAccelerated(List<int> accelerating,List<dynamic> list) {

 var indexOfSharpAccelerated = 1;
 bool wavePeak = false;
 var level1 = 0;
 var level2 = 0;
 var level3 = 0;
 var level4 = 0;
 var level5 = 0;

 for (var i = 0; i < list.length -1; i++) {

   //speed wave peak
   if (wavePeak == false && list[i][1] - list[i+1][1] >= 1 ) {wavePeak = true;}
   else if (wavePeak == true && list[i+1][1] - list[i][1] >= 1 ) {
     indexOfSharpAccelerated++;
     wavePeak = false;
   }

   if (list[i+1][1] - list[i][1] >= accelerating[4]) {level5++;}
   else if (list[i+1][1] - list[i][1] >= accelerating[3]) {level4++;}
   else if (list[i+1][1] - list[i][1] >= accelerating[2]) {level3++;}
   else if (list[i+1][1] - list[i][1] >= accelerating[1]) {level2++;}
   else if (list[i+1][1] - list[i][1] >= accelerating[0]) {level1++;}
 }

 var indice = 100 /indexOfSharpAccelerated;
 var mistakes = level1 + (level2 * 2) + (level3 * 3) + (level4 * 4) + (level5 *
5);
 print("Decelerating part");
 print("indexOfSharpAccelerated : $indexOfSharpAccelerated");
 print("indice : $indice");
```

```
 print("mistakes : $mistakes");
 var deduction= (mistakes * indice) / 2;
 print("deduction : $deduction");
 return deduction.toInt();
}

int getSharpDecelerated(List<int> braking,List<dynamic> list) {

 var indexOfSharpDecelerated = 1;
 bool wavePeak = false;
 var level1 = 0;
 var level2 = 0;
 var level3 = 0;
 var level4 = 0;
 var level5 = 0;

 for (var i = 0; i < list.length -1; i++) {

   //speed wave peak
   if (wavePeak == false && list[i+1][1] - list[i][1] >= 1 ) {
     wavePeak = true;
   }
   else if (wavePeak == true && list[i][1] - list[i+1][1] >= 1 ) {
     indexOfSharpDecelerated++;
     wavePeak = false;
   }

   if (list[i][1] - list[i+1][1] >= braking[4]) {level5++;}
   else if (list[i][1] - list[i+1][1] >= braking[3]) {level4++;}
   else if (list[i][1] - list[i+1][1] >= braking[2]) {level3++;}
   else if (list[i][1] - list[i+1][1] >= braking[1]) {level2++;}
   else if (list[i][1] - list[i+1][1] >= braking[0]) {level1++;}
 }

 var indice = 100 / indexOfSharpDecelerated;
 var mistakes = level1 + (level2 * 2) + (level3 * 3) + (level4 * 4) + (level5 *
5);
 print("Accelerating part");
 print("indexOfSharpDecelerated : $indexOfSharpDecelerated");
 print("indice : $indice");
 print("mistakes : $mistakes");
 var deduction= (mistakes * indice) / 2;
 print("deduction : $deduction");
 return deduction.toInt();
}

int getMaxSpeed(List<dynamic> list) {
 var maxSpeed = 0;
 for (var i = 0; i < list.length; i++) {
   if (list[i][1] > maxSpeed) {
     maxSpeed = list[i][1];
   }
 }
 return maxSpeed;
}

int getAvgSpeed(List<dynamic> list) {
 double sumOfSpeed = 0;
 int count = 0;
 int result = 0;

 for (var i = 0; i < list.length; i++) {
   if (list[i][1] >= 5) {
     count++;
     sumOfSpeed += list[i][1];
   }
 }
```

```
 result = sumOfSpeed ~/ count;
 return result;
}

int getTripDistance(List<dynamic> list) {
 var distance = 0.0;
 var totalDistance = 0.0;
 var currentLatitude = 0.0;
 var currentLongitude = 0.0;
 var lastLatitude = 0.0;
 var lastLongitude = 0.0;
 for (var i = 10; i < list.length -2; i++) {
   lastLatitude = list[i][2].latitude;
   lastLongitude = list[i][2].longitude;
   currentLatitude = list[i+1][2].latitude;
   currentLongitude = list[i+1][2].longitude;
   if (lastLatitude != 0.0 || lastLongitude != 0.0 || currentLatitude != 0.0 ||
currentLongitude != 0.0) {
     distance = Geolocator.distanceBetween(
         lastLatitude,lastLongitude,
         currentLatitude,currentLongitude
     );
     totalDistance += distance;
   }
 }
 return totalDistance.toInt();
}

String getEvaluate(int drivingScore) {
 if (drivingScore >= 90) {
   return "Excellent";
 }
 else if (drivingScore >= 80) {
   return "Very Good";
 }
 else if (drivingScore >= 70) {
   return "Good";
 }
 else if (drivingScore >= 60) {
   return "Average";
 }
 else if (drivingScore >= 50) {
   return "Fair";
 }
 else if (drivingScore >= 40) {
   return "Poor";
 }
 else {
   return "Very Poor";
 }
}

Future<List<double>> getRating(AlgorithmJson json, List<dynamic> list) async {

 List<double> output = [0,0,0];

 Map<String, dynamic> rule = {};
 List<int> speeding = [];
 List<int> braking = [];
 List<int> accelerating = [];
 for (var i = 0; i < json.speeding.length; i++) {
   speeding.add(json.speeding[i].toInt());
 }
 for (var i = 0; i < json.braking.length; i++) {
   braking.add(json.braking[i].toInt());
 }
 for (var i = 0; i < json.accelerating.length; i++) {
```

```
    accelerating.add(json.accelerating[i].toInt());
  }

  rule['speeding'] = speeding;
  rule['braking'] = braking;
  rule['accelerating'] = accelerating;

  int sharpAccelerated = getSharpAccelerated(rule['accelerating'],list);
  int sharpDecelerated = getSharpDecelerated(rule['braking'],list);
  int overSpeed = await getOverSpeeding(rule['speeding'], list);

  double speedingPercent = (50 - overSpeed) / 50;
  if (speedingPercent < 0) {
    speedingPercent = 0;
  }

  double acceleratingPercent = (50 - sharpAccelerated) / 50;
  if(acceleratingPercent < 0) {
    acceleratingPercent = 0;
  }

  double brakingPercent = (50 - sharpDecelerated) / 50;
  if(brakingPercent < 0) {
    brakingPercent = 0;
  }

  output = [speedingPercent,acceleratingPercent,brakingPercent];

  return output;
}

Future<int> getOverSpeeding(List<int> speeding, List<dynamic> list) async{
  int result = 0;
  List<dynamic> motorways =
['M1','M2','M3','M4','M6','M7','M8','M9','M11','M17','M18','M20','M50'];
  List<dynamic> nationalRoad =
['N1','N2','N3','N4','N5','N6','N7','N8','N9','N10','N11','N12','N13','N14','N15'
,'N16','N17','N18','N19','N20','N21','N22','N23','N24','N25','N26','N27','N28','N
29','N30','N31','N32','N33','N40'];
  List<dynamic> specifiedRoad = ['Green Rd'];
  int highestSpeed = 0;
  int index = 0;
  for(var i = 0; i < list.length; i++) {
    if (list[i][1] > highestSpeed) {
      highestSpeed = list[i][1];
      index = i;
    }
  }
  String postsURL =
"https://maps.googleapis.com/maps/api/geocode/json?latlng=${list[index][2].latitu
de},${list[index][2].longitude}&key=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
  Response res = await get(Uri.parse(postsURL));
  if (res.statusCode == 200) {
    Map<String, dynamic> geocode = jsonDecode(res.body);
    for (var j = 0; j < geocode['results'].length; j++) {
      var address =
geocode['results'][j]['address_components'][0]['short_name'].toString();
      print('address: ${address}');
      //motorways
      if(motorways.contains(address)) {
        if ((list[index][1] * (100 / 120)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 120)) - 100 >= speeding[3]) { result +=
30;}
        else if ((list[index][1] * (100 / 120)) - 100 >= speeding[2]) { result +=
15;}
        else if ((list[index][1] * (100 / 120)) - 100 >= speeding[1]) { result +=
10;}
```

```
        else if ((list[index][1] * (100 / 120)) - 100 >= speeding[0] ) { result +=
5;}
        break;
      }
      //rationalRoad
      else if(nationalRoad.contains(address)) {
        if ((list[index][1] * (100 / 100)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 100)) - 100 >= speeding[3]) { result +=
30;}
        else if ((list[index][1] * (100 / 100)) - 100 >= speeding[2]) { result +=
15;}
        else if ((list[index][1] * (100 / 100)) - 100 >= speeding[1]) { result +=
10;}
        else if ((list[index][1] * (100 / 100)) - 100 >= speeding[0]) { result +=
5;}
        break;
      }
      //Regional roads
      else if(address.length == 4 && address[0] == 'R' &&
double.tryParse(address[1]) != null && double.tryParse(address[2]) != null &&
double.tryParse(address[3]) != null) {
        if ((list[index][1] * (100 / 80)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 80)) - 100 >= speeding[3]) { result +=
30;}
        else if ((list[index][1] * (100 / 80)) - 100 >= speeding[2]) { result +=
15;}
        else if ((list[index][1] * (100 / 80)) - 100 >= speeding[1]) { result +=
10;}
        else if ((list[index][1] * (100 / 80)) - 100 >= speeding[0]) { result +=
5;}
      }
      //specificed speed road
      else if(specifiedRoad.contains(address)) {
        if ((list[index][1] * (100 / 50)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[3]) { result +=
30;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[2]) { result +=
15;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[1]) { result +=
10;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[0]) { result +=
5;}
        break;
      }
      //city max speed
      else {
        if ((list[index][1] * (100 / 50)) - 100 >= speeding[4]) { result += 50;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[3]) { result +=
30;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[2]) { result +=
15;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[1]) { result +=
10;}
        else if ((list[index][1] * (100 / 50)) - 100 >= speeding[0]) { result +=
5;}
        break;
      }
    }
  }
  else { throw "Unable to retrieve posts.";}
  return result;
}
```

6.3 invalid.dart

Description: When the driving record is invalid, the driver is notified of the reason why the driving record is not recorded by the system.

```
import 'package:flutter/material.dart';
import 'menu.dart';

class Invalid extends StatelessWidget {

 @override
 Widget build(BuildContext context) {
   return WillPopScope(
       onWillPop: () async {
     return Future.value(false);
   },
   child: MaterialApp(
     home: Scaffold(
       appBar: PreferredSize(
           preferredSize: Size(double.infinity, 60),
           child: AppBar(
             title: Text('Invalid Record'),
             centerTitle: true,
             backgroundColor: Colors.black,
             leading: IconButton(
               onPressed: () {
                 Navigator.push(
                     context, MaterialPageRoute(
                     builder: (context) => Menu()
                 ));
               },
               icon: Icon(Icons.home_outlined),
             ),
             elevation: 0, //remove shadow effect
           )
       ),
       body:
        Center(
         child: Container(
           alignment:Alignment.center,
           /*
         constraints: BoxConstraints(
           maxWidth: 300,
           maxHeight: 300,
           minWidth: 50,
           minHeight: 50
         ),
         margin: EdgeInsets.only(
           left: 10,
           top:10,
           right: 10,
           bottom: 10
         ),
         */
           padding: EdgeInsets.only(
               left: 10,
               top:10,
               right: 10,
               bottom: 10
           ),
```

```
            color: Color.fromARGB(255, 255, 255, 255),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: <Widget>[
                SizedBox(
                    height:30
                ),
                Container(
                  width: 300,
                  height: 300,
                  child: Image.asset(
                    'assets/images/sorry.png',
                  ),
                ),
                SizedBox(
                    height:30
                ),
                Text(
                  "Your driving is not recorded!",
                  style: TextStyle(color: Colors.black, fontSize: 22),
                ),
                SizedBox(
                    height:30
                ),
                Text(
                  "A valid record must meet the following criteria:\n  • Driving
for more than 5 minutes\n  • The driving distance is more than 1000 meters",
                  style: TextStyle(color: Colors.black, fontSize: 12),
                ),
                SizedBox(
                    height:30
                ),
                SizedBox(
                    width: 200,
                    height: 50,
                    child: RaisedButton.icon(
                        icon: Icon(Icons.home),
                        color: Colors.green,
                        label: Text(
                            'Home',
                            style: TextStyle(color: Colors.black, fontSize: 22),
                        ),
                        //textColor: Colors.grey,
                        onPressed:() {
                          Navigator.push(
                              context, MaterialPageRoute(
                              builder: (context) => Menu()
                          )
                          );
                        }
                    ),
                ),

                SizedBox(
                    height:100
                ),
              ] ,
            ),
          ),
        ),
      ),
    ));
  }
}
```

6.4 main.dart

Description: It is the entrance of this application and is responsible for checking whether the user is already logged in. If not, the system will move to the login page; or proceed to the main menu if already logged in. And it will take the user's information from the cloud database and temporarily store it in the local session.

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
//import 'package:shared_preferences/shared_preferences.dart';
import 'package:flutter_session_manager/flutter_session_manager.dart';
import 'package:flutter/services.dart';
import 'dart:convert';
import 'menu.dart';
import 'signInPage.dart';
import 'algorithmJson.dart';

final FirebaseAuth _auth = FirebaseAuth.instance;
final User? user = _auth.currentUser;


Future<void> main() async {

 //for firebase
 WidgetsFlutterBinding.ensureInitialized();
 await Firebase.initializeApp();

 //only allow portrait mode
 await SystemChrome.setPreferredOrientations(
   [
     DeviceOrientation.portraitUp,
     DeviceOrientation.portraitDown,
   ],
 );

 //fullscreen, disable status and navigation bars
 SystemChrome.setEnabledSystemUIMode(SystemUiMode.immersiveSticky);

 runApp(MyApp());
}

class MyApp extends StatefulWidget {
 @override
 _MyState createState() => _MyState();

}

class _MyState extends State<MyApp> {

 var isLoggedIn = (user != null) ? false: user == null;
 var userUID ="";

 @override
 void initState() {
   if (user != null) {
     getUserInfo(getUserUID());
   }
   super.initState();
 }
```

```
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
         home: isLoggedIn ? SignInPage() : Menu()
   );
 }

 String getUserUID() {
   final User? user = _auth.currentUser;
   if (user != null) {
     return user.uid.toString();
   }
   return "";
 }

 Future<void> getUserInfo(String uid) async{
   var sessionManager = SessionManager();
   var collection = FirebaseFirestore.instance.collection('users');
   var document = await collection.doc(uid).get();

   if (document.exists) {
     Map<String, dynamic>? data = document.data();
     await sessionManager.set('firstname', data?['firstname']);
     await sessionManager.set('surname', data?['surname']);
     await sessionManager.set('nickname', data?['nickname']);
     await sessionManager.set('group', data?['group']);
     await sessionManager.set('algorithm', data?['algorithm']);
     Map<String, dynamic> rule = await setAlgorithmSession(await
SessionManager().get('algorithm'));
     List<int> speeding = [];
     for (var i = 0; i < rule['speeding'].length; i++) {
       speeding.add(rule['speeding'][i].toInt());
     }
     List<int> braking = [];
     for (var i = 0; i < rule['braking'].length; i++) {
       braking.add(rule['braking'][i].toInt());
     }
     List<int> accelerating = [];
     for (var i = 0; i < rule['accelerating'].length; i++) {
       accelerating.add(rule['accelerating'][i].toInt());
     }
     AlgorithmJson scoring = AlgorithmJson(speeding,braking,accelerating);
     await sessionManager.set('scoring', scoring);
   }
 }

 Future <Map<String, dynamic>> setAlgorithmSession(String algorithmName) async {

   Map<String, dynamic> rule = {};
   //Map<String, dynamic> rule =
{'speeding':[5,10,15,20,25],'braking':[7,10,13,16,19],'accelerating':[6,8,10,12,1
4]};
   var collection = FirebaseFirestore.instance.collection('algorithm');
   var document = await collection.doc(algorithmName).get();
   if (document.exists) {
     Map<String, dynamic>? data = document.data();
     rule['speeding'] = data?['speeding'];
     rule['braking'] = data?['braking'];
     rule['accelerating'] = data?['accelerating'];
     return rule;
   }
   return rule;
 }

}
```

6.5 menu.dart

Description: It is primarily the main menu that provides users with pages to connect to various functions.

```
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
//import 'package:shared_preferences/shared_preferences.dart';
import 'package:flutter_session_manager/flutter_session_manager.dart';
import 'package:flutter/services.dart';
import 'dart:convert';
import 'menu.dart';
import 'signInPage.dart';
import 'algorithmJson.dart';

final FirebaseAuth _auth = FirebaseAuth.instance;
final User? user = _auth.currentUser;


Future<void> main() async {

 //for firebase
 WidgetsFlutterBinding.ensureInitialized();
 await Firebase.initializeApp();

 //only allow portrait mode
 await SystemChrome.setPreferredOrientations(
   [
     DeviceOrientation.portraitUp,
     DeviceOrientation.portraitDown,
   ],
 );

 //fullscreen, disable status and navigation bars
 SystemChrome.setEnabledSystemUIMode(SystemUiMode.immersiveSticky);

 runApp(MyApp());
}

class MyApp extends StatefulWidget {
 @override
 _MyState createState() => _MyState();

}

class _MyState extends State<MyApp> {

 var isLoggedIn = (user != null) ? false: user == null;
 var userUID ="";

 @override
 void initState() {
   if (user != null) {
     getUserInfo(getUserUID());
   }
   super.initState();
 }

 @override
 Widget build(BuildContext context) {
   return MaterialApp(
         home: isLoggedIn ? SignInPage() : Menu()
```

```
    );
  }

  String getUserUID() {
    final User? user = _auth.currentUser;
    if (user != null) {
      return user.uid.toString();
    }
    return "";
  }

  Future<void> getUserInfo(String uid) async{
    var sessionManager = SessionManager();
    var collection = FirebaseFirestore.instance.collection('users');
    var document = await collection.doc(uid).get();

    if (document.exists) {
      Map<String, dynamic>? data = document.data();
      await sessionManager.set('firstname', data?['firstname']);
      await sessionManager.set('surname', data?['surname']);
      await sessionManager.set('nickname', data?['nickname']);
      await sessionManager.set('group', data?['group']);
      await sessionManager.set('algorithm', data?['algorithm']);
      Map<String, dynamic> rule = await setAlgorithmSession(await
SessionManager().get('algorithm'));
      List<int> speeding = [];
      for (var i = 0; i < rule['speeding'].length; i++) {
        speeding.add(rule['speeding'][i].toInt());
      }
      List<int> braking = [];
      for (var i = 0; i < rule['braking'].length; i++) {
        braking.add(rule['braking'][i].toInt());
      }
      List<int> accelerating = [];
      for (var i = 0; i < rule['accelerating'].length; i++) {
        accelerating.add(rule['accelerating'][i].toInt());
      }
      AlgorithmJson scoring = AlgorithmJson(speeding,braking,accelerating);
      await sessionManager.set('scoring', scoring);
    }
  }

  Future <Map<String, dynamic>> setAlgorithmSession(String algorithmName) async {

    Map<String, dynamic> rule = {};
    //Map<String, dynamic> rule =
{'speeding':[5,10,15,20,25],'braking':[7,10,13,16,19],'accelerating':[6,8,10,12,1
4]};
    var collection = FirebaseFirestore.instance.collection('algorithm');
    var document = await collection.doc(algorithmName).get();
    if (document.exists) {
      Map<String, dynamic>? data = document.data();
      rule['speeding'] = data?['speeding'];
      rule['braking'] = data?['braking'];
      rule['accelerating'] = data?['accelerating'];
      return rule;
    }
    return rule;
  }

}
```

6.6 monitor.dart

Description: It is the primary function responsible for taking data while driving, displaying a map and real-time location and road traveled, and pointing out errors while driving.

```
import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:marker_icon/marker_icon.dart';
import 'package:geolocator/geolocator.dart';
import 'package:wakelock/wakelock.dart';
import 'dart:async';
import 'result.dart';
import 'drivingScore.dart';
import 'invalid.dart';

class Monitor extends StatefulWidget {
 @override
 Journey createState() => Journey();
}

class Journey extends State<Monitor> {

 var position;
 late StreamSubscription _positionSubscription;
 var speedToInt = 0;
 Color speedColor = Color.fromARGB(255, 0, 0, 0);
 //var totalDistance = 0;
 var buttonText = "Start";
 var mapZoom = 15.5; //14.4746
 var mapTilt = 0.0;
 var mapHeading = 0.0;
 Color polyColor = Color.fromARGB(255, 153, 204, 255);
 bool pressAttention = false;
 bool backgroundChange = false;
 bool startStop = true; //for timer
 Stopwatch watch = Stopwatch();
 String elapsedTime = "00:00:00";
 late Timer timer;
 List<dynamic> trip = [];
 List<LatLng> polylineCoordinates = [];
 final Completer<GoogleMapController> _controller = Completer();
 final Set<Polyline> _polylines = {};
 final Set<Marker> markers = Set();

 Journey() {
   getCurrentLocation().then((val) => setState(() {
     position = val;
   }));
 }

 Future<Position> getCurrentLocation() async{
   return await Geolocator.getCurrentPosition(desiredAccuracy:
LocationAccuracy.high);;
 }

 startOrStop() {
   if(startStop) {
     watch.start();
     Wakelock.enable();
     setState(() {
       startStop = false;
       buttonText = "STOP";
```

```
      pressAttention = !pressAttention;
      backgroundChange = !backgroundChange;
    });
    timer = Timer.periodic(Duration(milliseconds: 1000), (Timer t) {
      googleMapUpdated();
      updateTime(timer);
      trip.add([elapsedTime,speedToInt,position]);
      setSpeedMarker();
      polylineCoordinates.add(
          LatLng(position.latitude, position.longitude)
      );
      Polyline polyline = Polyline(
          polylineId: PolylineId('poly'),
          color: polyColor,
          width: 6,
          points: polylineCoordinates
      );
      _polylines.add(polyline);
    });
  } else {
    startStop = true;
    watch.stop();
    timer.cancel();
    watch.reset();
    Wakelock.disable();
    if (trip.length > 300 && getTripDistance(trip) > 1000 ) {
      Navigator.push(
          context, MaterialPageRoute(
          builder: (context) => Result('monitor',trip))
      );
    }
    else {
      Navigator.push(
          context, MaterialPageRoute(
          builder: (context) => Invalid())
      );
    }
  }
}

@override
void initState() {
  super.initState();
  checkGPSPermission();
  getGeoLocation();
}

@override
Widget build(BuildContext context) {
  return WillPopScope (
      onWillPop: () async {
        return Future.value(false);
      },

      child:MaterialApp(
        home: Scaffold(
          backgroundColor: backgroundChange ? Color.fromARGB(255, 255, 180, 180)
: Color.fromARGB(
              255, 180, 255, 180),
          appBar: PreferredSize(
              preferredSize: Size(double.infinity, 60),
              child: AppBar(
                title: Text('Driving Monitor'),
                centerTitle: true,
                backgroundColor: Colors.black,
                //leading: Icon(Icons.account_circle_rounded),
                leading: IconButton(
```

```
                onPressed: () {
                  Navigator.pop(context);
                },
                icon: Icon(Icons.home_outlined),
              ),
              elevation: 0, //remove shadow effect
            )
          ),
          body: Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: [
                Container(
                  color: Colors.black,
                  //width: 300,
                  height: 400,
                  padding: EdgeInsets.only(
                    //top:18,
                    bottom:5,
                  ),
                  margin: EdgeInsets.only(
                    //top:18,
                    bottom:1,
                  ),
                  child:
                  GoogleMap (
                    myLocationEnabled: true,
                    compassEnabled: true,
                    tiltGesturesEnabled: false,
                    mapType: MapType.normal,
                    myLocationButtonEnabled: true,
                    zoomControlsEnabled: true,
                    zoomGesturesEnabled: true,
                    scrollGesturesEnabled: true,
                    rotateGesturesEnabled: true,
                    polylines: _polylines,
                    markers: markers,

                    initialCameraPosition: CameraPosition(
                      target: LatLng(position.latitude, position.longitude),
                      zoom: mapZoom,
                      tilt: mapTilt,
                      bearing: position.heading
                    ),

                    onMapCreated: (GoogleMapController controller) {
                      _controller.complete(controller);
                    },
                  ),
                ),

                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    Text(
                      speedToInt.toString(),
                      style: TextStyle(color: speedColor, fontSize: 100),
                    ),
                    Text(
                      " km/h",
                      style: TextStyle(color: speedColor, fontSize: 24),
                    ),
                  ],
                ),
                Text(
                  elapsedTime,
```

```
                  style: TextStyle(color: Colors.black, fontSize: 24),
                ),
                Text(

"${position.latitude.toString()},${position.longitude.toString()}",
                  style: TextStyle(color: Colors.black, fontSize: 12),
                ),
                SizedBox(
                    height:20
                ),
                Container(
                  // set width equal to height to make a square
                    width: 100,
                    height: 100,
                    child: RaisedButton(
                      color: pressAttention ? Colors.red : Colors.green,
                      shape: RoundedRectangleBorder(
                        // set the value to a very big number like 100, 1000...
                          borderRadius: BorderRadius.circular(100)),
                      child: Text(
                        buttonText,
                        style: TextStyle(color: Colors.white, fontSize: 20),
                      ),
                      onPressed: () {
                        startOrStop();
                      },
                    )
                ),
                SizedBox(
                    height:20
                ),
              ],
            ),
          ),
        ),
      ));
  }

  @override
  void dispose() {
    _positionSubscription.cancel();
    timer.cancel();
    watch.reset();
    super.dispose();
  }
  checkGPSPermission() async {
    bool serviceEnabled;
    LocationPermission permission;
    serviceEnabled = await Geolocator.isLocationServiceEnabled();
    if (!serviceEnabled) {
      return Future.error('Location services are disabled.');
    }
    permission = await Geolocator.checkPermission();
    if (permission == LocationPermission.denied) {
      permission = await Geolocator.requestPermission();
      if (permission == LocationPermission.denied) {
        return Future.error('Location permissions are denied');
      }
    }
  }

  Future<void> googleMapUpdated() async {
    final GoogleMapController controller = await _controller.future;
    controller.animateCamera (CameraUpdate.newCameraPosition(
        CameraPosition(
            target: LatLng(position.latitude, position.longitude),
            zoom: mapZoom,
```

```
          tilt: mapTilt,
          bearing: position.heading
      )
  ));
}

getGeoLocation() async{
  _positionSubscription = await Geolocator.getPositionStream(
    locationSettings: locationSettings,
  ).listen((pos) {
    setState(() {
      position = pos;
      speedToInt = (position.speed * 3.85).toInt();
    });
  });
}

final LocationSettings locationSettings = AndroidSettings(
  accuracy: LocationAccuracy.bestForNavigation,
  distanceFilter: 1,
  forceLocationManager: true,
  intervalDuration: Duration(milliseconds: 200),
);

updateTime(Timer timer) {
  if (watch.isRunning) {
    setState(() {
      elapsedTime = transformMilliSeconds(watch.elapsedMilliseconds);
    });
  }
}

transformMilliSeconds(int milliseconds) {
  int hundreds = (milliseconds / 10).truncate();
  int seconds = (hundreds / 100).truncate();
  int minutes = (seconds / 60).truncate();
  int hours = (minutes / 60).truncate();
  String hoursStr = (hours % 60).toString().padLeft(2, '0');
  String minutesStr = (minutes % 60).toString().padLeft(2, '0');
  String secondsStr = (seconds % 60).toString().padLeft(2, '0');
  return "$hoursStr:$minutesStr:$secondsStr";
}

setSpeedMarker() async {

  speedColor = Color.fromARGB(255, 0, 0, 0);

  BitmapDescriptor a1 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
      text: '^',
      circleColor:Color.fromARGB(255, 248, 213, 148),
      fontColor: Colors.white,fontSize: 35);

  BitmapDescriptor a2 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
      text: '^',
      circleColor:Color.fromARGB(255, 239, 190, 102),
      fontColor: Colors.white,fontSize: 35);

  BitmapDescriptor a3 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
      text: '^',
      circleColor:Color.fromARGB(255, 220, 176, 61),
      fontColor: Colors.white,fontSize: 35);

  BitmapDescriptor a4 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
```

```
      text: '^',
      circleColor:Color.fromARGB(255, 220, 148, 61),
      fontColor: Colors.white,fontSize: 35);

  BitmapDescriptor a5 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
      text: '^',
      circleColor:Color.fromARGB(255, 208, 108, 37),
      fontColor: Colors.white,fontSize: 35);

  BitmapDescriptor d1 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
      text: 'V',
      circleColor:Color.fromARGB(255, 243, 157, 157),
      fontColor: Colors.white,fontSize: 35);

  BitmapDescriptor d2 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
      text: 'V',
      circleColor:Color.fromARGB(255, 246, 123, 123),
      fontColor: Colors.white,fontSize: 35);

  BitmapDescriptor d3 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
      text: 'V',
      circleColor:Color.fromARGB(255, 239, 103, 103),
      fontColor: Colors.white,fontSize: 35);

  BitmapDescriptor d4 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
      text: 'V',
      circleColor:Color.fromARGB(255, 239, 73, 73),
      fontColor: Colors.white,fontSize: 35);

  BitmapDescriptor d5 = await MarkerIcon.circleCanvasWithText(
      size: Size(80, 80),
      text: 'V',
      circleColor:Color.fromARGB(255, 243, 19, 19),
      fontColor: Colors.white,fontSize: 35);

  if (trip.length > 1) {
    var currentSpeed = trip.elementAt(trip.length - 1)[1];
    var lastSpeed = trip.elementAt(trip.length - 2)[1];

    //Sharp Accelerate detection
    if (currentSpeed - lastSpeed >= 7) {
      if (currentSpeed - lastSpeed >= 13) {
        markers.add(Marker( //add first marker
          markerId: MarkerId(position.toString()),
          draggable: false,
          position: LatLng(position.latitude,position.longitude), //position of
marker
          infoWindow: InfoWindow( //popup info
            title: 'Accelerate L5 ',
            snippet: 'over >= 13',
          ),
          icon: a5, //Icon for Marker
        ));

        speedColor = Color.fromARGB(255, 255, 153, 0);
      }
      else if (currentSpeed - lastSpeed >= 12) {

        markers.add(Marker( //add first marker
          markerId: MarkerId(position.toString()),
          draggable: false,
          position: LatLng(position.latitude,position.longitude), //position of
```

```
marker
            infoWindow: InfoWindow( //popup info
              title: 'Accelerate L4 ',
              snippet: 'over >= 12',
            ),
            icon: a4, //Icon for Marker
          ));

          speedColor = Color.fromARGB(255, 139, 101, 6);
        }
        else if (currentSpeed - lastSpeed >= 10) {

          markers.add(Marker( //add first marker
            markerId: MarkerId(position.toString()),
            draggable: false,
            position: LatLng(position.latitude,position.longitude), //position of
marker
            infoWindow: InfoWindow( //popup info
              title: 'Accelerate L3 ',
              snippet: 'over >= 10',
            ),
            icon: a3, //Icon for Marker
          ));

          speedColor = Color.fromARGB(255, 113, 63, 0);
        }
        else if (currentSpeed - lastSpeed >= 8) {

          markers.add(Marker( //add first marker
            markerId: MarkerId(position.toString()),
            draggable: false,
            position: LatLng(position.latitude,position.longitude), //position of
marker
            infoWindow: InfoWindow( //popup info
              title: 'Accelerate L2 ',
              snippet: 'over >= 8',
            ),
            icon: a2,
            //icon: BitmapDescriptor.defaultMarker, //Icon for Marker
          ));

          speedColor = Color.fromARGB(255, 83, 47, 0);
        }
        else {

          markers.add(Marker( //add first marker
            markerId: MarkerId(position.toString()),
            draggable: false,
            position: LatLng(position.latitude,position.longitude), //position of
marker
            infoWindow: InfoWindow( //popup info
              title: 'Accelerate L1 ',
              snippet: 'over >= 7',
            ),
            icon: a1, //Icon for Marker
          ));

          speedColor = Color.fromARGB(255, 56, 34, 0);
        }
      }
      //Sharp Decelerate detection
      else if (lastSpeed - currentSpeed >= 7) {
        if (lastSpeed - currentSpeed >= 21) {

          markers.add(Marker( //add first marker
            markerId: MarkerId(position.toString()),
            draggable: false,
```

```
            position: LatLng(position.latitude,position.longitude), //position of
marker
            infoWindow: InfoWindow( //popup info
              title: 'Decelerate L5 ',
              snippet: 'over >= 21',
            ),
            icon: d5, //Icon for Marker
          ));

          speedColor = Color.fromARGB(255, 204, 0, 0);
        }
        else if (lastSpeed - currentSpeed >= 18) {

          markers.add(Marker( //add first marker
            markerId: MarkerId(position.toString()),
            draggable: false,
            position: LatLng(position.latitude,position.longitude), //position of
marker
            infoWindow: InfoWindow( //popup info
              title: 'Decelerate L4 ',
              snippet: 'over >= 18',
            ),
            icon: d4, //Icon for Marker
          ));

          speedColor = Color.fromARGB(255, 200, 0, 0);
        }
        else if (lastSpeed - currentSpeed >= 14) {

          markers.add(Marker( //add first marker
            markerId: MarkerId(position.toString()),
            draggable: false,
            position: LatLng(position.latitude,position.longitude), //position of
marker
            infoWindow: InfoWindow( //popup info
              title: 'Decelerate L3 ',
              snippet: 'over >= 14',
            ),
            icon: d3, //Icon for Marker
          ));

          speedColor = Color.fromARGB(255, 150, 0, 0);
        }
        else if (lastSpeed - currentSpeed >= 9) {

          markers.add(Marker( //add first marker
            markerId: MarkerId(position.toString()),
            draggable: false,
            position: LatLng(position.latitude,position.longitude), //position of
marker
            infoWindow: InfoWindow( //popup info
              title: 'Decelerate L2 ',
              snippet: 'over >= 9',
            ),
            icon: d2, //Icon for Marker
          ));

          speedColor = Color.fromARGB(255, 100, 0, 0);
        }
        else {

          markers.add(Marker( //add first marker
            markerId: MarkerId(position.toString()),
            draggable: false,
            position: LatLng(position.latitude,position.longitude), //position of
marker
            infoWindow: InfoWindow( //popup info
```

```
            title: 'Decelerate L1 ',
            snippet: 'over >= 7',
          ),
          icon: d1, //Icon for Marker
        ));

        speedColor = Color.fromARGB(255, 50, 0, 0);
      }
    }
  }
 }
}
```

6.7 profile.dart

Description: It reports comprehensive user profiles such as the average score, a linear graph of the last ten driving ratings, total driving time and total driving distance.

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:flutter_session_manager/flutter_session_manager.dart';
import 'package:fl_chart/fl_chart.dart';
import 'menu.dart';
import 'drivingScore.dart';
import 'setting.dart';
import 'algorithmJson.dart';


final FirebaseAuth _auth = FirebaseAuth.instance;

class Profile extends StatefulWidget {
 @override
 _Profile createState() => _Profile();

}

class _Profile extends State<Profile> {

 List<int> _score = [];
 int _avgScore = 0;
 var _email ="";
 int _numberOfDriving = 0;
 late int _totalTime = 0;
 double _totalDistance = 0;
 String groupName = '';
 String nickname = "Anonymous";

 _Profile() {
   getData().then((val) => setState(() {
     _score = val;
     _avgScore = getAvgScore(_score);
     _email = userEmail();
     _numberOfDriving = _score.length;
     _totalDistance = (_totalDistance / 1000);
   }));
   getNickName(userUID()).then((val) => setState(() {
     nickname = val;
   }));
 }

 @override
```

```
void initState() {
  /*getAlgorithm().then((val) => setState(() {
    groupName = val;
  }));*/
  super.initState();
}

@override
Widget build(BuildContext context) {
  return WillPopScope(
      onWillPop: () async {
    return Future.value(false);
  },
  child: MaterialApp(
    home: Scaffold(
        appBar: PreferredSize(
            preferredSize: Size(double.infinity, 60),
            child: AppBar(
              title: Text("${nickname}'s Profile"),
              centerTitle: true,
              backgroundColor: Colors.black,
              leading: IconButton(
                onPressed: () {
                  Navigator.push(
                      context, MaterialPageRoute(
                      builder: (context) => Menu()
                  ));
                },
                icon: Icon(Icons.home_outlined),
              ),
              actions: <Widget>[
                IconButton(
                  icon: const Icon(Icons.settings),
                  tooltip: 'Setting',
                  onPressed: () {
                    Navigator.push(
                        context, MaterialPageRoute(
                        builder: (context) => Setting()
                    ));
                  },
                ),
                ],
              elevation: 0, //remove shadow effect
            )
        ),
      body: Center(
        child: Container(
        alignment:Alignment.center,
            padding: EdgeInsets.only(
                left: 10,
                top:10,
                right: 10,
                bottom: 10
            ),
            child: Column (
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                crossAxisAlignment: CrossAxisAlignment.center,
                children: <Widget>[
                  Text(
                    "Last 10 driving records",
                    style: TextStyle(color: Colors.black, fontSize: 20),
                  ),
                  Container(
                    padding: EdgeInsets.all(10),
                    width: 380,
                    height: 200,
                    child: LineChart(LineChartData(
```

```
                        minX: 0,
                        minY: 0,
                        maxX: 10,
                        maxY: 100,
                          borderData: FlBorderData(show: false),
                          titlesData: FlTitlesData(
                            topTitles: SideTitles(
                              showTitles: false,
                            ),
                            rightTitles: SideTitles(
                              showTitles: false,
                            ),
                            leftTitles: SideTitles(
                              showTitles: true,
                              getTextStyles: (context, value) => const
TextStyle(fontSize: 12),
                              interval: 20,
                            ),
                            bottomTitles: SideTitles(
                              showTitles: true,
                              getTextStyles: (context, value) => const
TextStyle(fontSize: 12),
                                  getTitles: (value) {
                                    switch (value.toInt()) {
                                      case 0:
                                        return '1';
                                      case 1:
                                        return '2';
                                      case 2:
                                        return '3';
                                      case 3:
                                        return '4';
                                      case 4:
                                        return '5';
                                      case 5:
                                        return '6';
                                      case 6:
                                        return '7';
                                      case 7:
                                        return '8';
                                      case 8:
                                        return '9';
                                      case 9:
                                        return '10';
                                      default:
                                        return '';
                                    }
                                  }
                                //interval: 1,
                            ),
                          ),

                          axisTitleData: FlAxisTitleData(
                              leftTitle: AxisTitle(
                                  showTitle: true,
                                  titleText: 'Driving Score',
                                  margin: 0
                              ),
                          ),

                          gridData: FlGridData(
                            show: true,
                          ),

                          lineBarsData: [
                            LineChartBarData(
                                isCurved: false,
```

```
                                dotData: FlDotData(
                                  show: true,
                                ),
                                spots: getLineChartValue(_score)
                            )
                        ]
                    ),
                    ),
                ),
                SizedBox(
                    height:20
                ),
                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: const [
                    Text(
                      "avg ",
                      style: TextStyle(fontSize: 20),
                    ),
                    Text(
                      "Score",
                      style: TextStyle(fontSize: 40),
                    ),
                  ],
                ),
                Text(
                  "${_avgScore.toString()}",
                  style: TextStyle(color: Colors.black, fontSize: 100),
                ),
                SizedBox(
                    height:20
                ),
                Text(
                  "Total driving : ${_numberOfDriving.toString()}
times\n\nTotal Time : ${transformSeconds(_totalTime)}\n\nTotal distance :
${_totalDistance.toStringAsPrecision(3)} km",
                  style: TextStyle(color: Colors.black, fontSize: 15),
                ),


                SizedBox(
                    height:30
                ),
                SizedBox(
                  width: 200,
                  height: 50,
                  child: RaisedButton.icon(
                      icon: Icon(Icons.home),
                      color: Colors.green,
                      label: Text(
                        'Home',
                        style: TextStyle(color: Colors.black, fontSize: 22),
                      ),
                      //textColor: Colors.grey,
                      onPressed:() {
                        Navigator.push(
                            context, MaterialPageRoute(
                            builder: (context) => Menu()
                        )
                        );
                      }
                  ),
                ),
                SizedBox(
                    height:20
                ),
              ]
```

```
            )
          )
    )
      )
    )
    );
  }

  Future<String> getAlgorithm() async{
    final prefs = await SharedPreferences.getInstance();
    return prefs.getString('algorithm').toString();
  }

  int getAvgScore(List<int> drivingScore) {
    var sum = 0;
    for (var i = 0; i < drivingScore.length; i++) {
      sum += drivingScore[i];
    }
    return sum ~/ drivingScore.length;
  }

  Future<List<int>> getData() async{
    late List<int> score = [];
    AlgorithmJson json = AlgorithmJson.fromJson(await
SessionManager().get('scoring'));
    await FirebaseFirestore.instance
        .collection('journeys')
        .limit(10)
        .where('userUID', isEqualTo: userUID())
        .orderBy('createdAt', descending: false)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) async {
        List<dynamic> list = [];
        for (var i = 0; i < doc["speed"].length; i++) {
          list.add([doc["time"][i],doc["speed"][i],doc["geoPoint"][i]]);
          _totalTime ++;
        }
        await getDrivingScore(json,list).then((drivingScore) {
          setState(() {
            score.add(drivingScore);
            _totalDistance += getTripDistance(list).toDouble();
          });
        } );
      });
    });
    return Future.delayed(Duration(seconds: 1), () =>score);
  }

  transformSeconds(int seconds) {
    //int hundreds = (milliseconds / 10).truncate();
    //int seconds = (hundreds / 100).truncate();
    int minutes = (seconds / 60).truncate();
    int hours = (minutes / 60).truncate();
    String hoursStr = (hours % 60).toString();
    String minutesStr = (minutes % 60).toString();
    //String secondsStr = (seconds % 60).toString().padLeft(2, '0');
    return "$hoursStr hrs $minutesStr mins";
  }

  List<FlSpot> getLineChartValue(List<int> list) {
    List<FlSpot> spot = [];
    double index;
    double value;
    for (var i = 0; i < list.length; i++) {
      index = i.toDouble();
      value = list[i].toDouble();
```

```
      spot.add(FlSpot(index,value));
    }
    return spot;
  }

  String userUID() {
    final User? user = _auth.currentUser;
    if (user != null) {
      return user.uid.toString();
    }
    return "";
  }

  String userEmail() {
    final User? user = _auth.currentUser;
    if (user != null) {
      return user.email.toString();
    }
    return "";
  }

  Future <String> getNickName(String userUID) async {

    String nickname = "Anonymous";

    await FirebaseFirestore.instance
        .collection('users')
        .doc(userUID)
        .get()
        .then((DocumentSnapshot documentSnapshot) {
          if (documentSnapshot.exists) {
            nickname = documentSnapshot['nickname'];
          }
    });

    return nickname;
  }

}
```

6.8 ranking.dart

Description: Ranks for each driver's driving score display the driver's name, driving time, and driving distance in the ranking.

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:flutter_session_manager/flutter_session_manager.dart';
import 'package:fl_chart/fl_chart.dart';
import 'menu.dart';
import 'drivingScore.dart';
import 'setting.dart';
import 'algorithmJson.dart';


final FirebaseAuth _auth = FirebaseAuth.instance;

class Profile extends StatefulWidget {
 @override
  _Profile createState() => _Profile();
```

```
}

class _Profile extends State<Profile> {

 List<int> _score = [];
 int _avgScore = 0;
 var _email ="";
 int _numberOfDriving = 0;
 late int _totalTime = 0;
 double _totalDistance = 0;
 String groupName = '';
 String nickname = "Anonymous";

 _Profile() {
   getData().then((val) => setState(() {
     _score = val;
     _avgScore = getAvgScore(_score);
     _email = userEmail();
     _numberOfDriving = _score.length;
     _totalDistance = (_totalDistance / 1000);
   }));
   getNickName(userUID()).then((val) => setState(() {
     nickname = val;
   }));
 }

 @override
 void initState() {
   /*getAlgorithm().then((val) => setState(() {
     groupName = val;
   }));*/
   super.initState();
 }

 @override
 Widget build(BuildContext context) {
   return WillPopScope(
       onWillPop: () async {
     return Future.value(false);
   },
   child: MaterialApp(
     home: Scaffold(
         appBar: PreferredSize(
             preferredSize: Size(double.infinity, 60),
             child: AppBar(
               title: Text("${nickname}'s Profile"),
               centerTitle: true,
               backgroundColor: Colors.black,
               leading: IconButton(
                 onPressed: () {
                   Navigator.push(
                       context, MaterialPageRoute(
                       builder: (context) => Menu()
                   ));
                 },
                 icon: Icon(Icons.home_outlined),
               ),
               actions: <Widget>[
                 IconButton(
                   icon: const Icon(Icons.settings),
                   tooltip: 'Setting',
                   onPressed: () {
                     Navigator.push(
                         context, MaterialPageRoute(
                         builder: (context) => Setting()
                     ));
                   },
```

```
                ),
                ],
              elevation: 0, //remove shadow effect
            )
        ),
      body: Center(
        child: Container(
        alignment:Alignment.center,
            padding: EdgeInsets.only(
                left: 10,
                top:10,
                right: 10,
                bottom: 10
            ),
            child: Column (
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                crossAxisAlignment: CrossAxisAlignment.center,
                children: <Widget>[
                  Text(
                    "Last 10 driving records",
                    style: TextStyle(color: Colors.black, fontSize: 20),
                  ),
                  Container(
                    padding: EdgeInsets.all(10),
                    width: 380,
                    height: 200,
                    child: LineChart(LineChartData(
                      minX: 0,
                      minY: 0,
                      maxX: 10,
                      maxY: 100,
                        borderData: FlBorderData(show: false),
                        titlesData: FlTitlesData(
                          topTitles: SideTitles(
                            showTitles: false,
                          ),
                          rightTitles: SideTitles(
                            showTitles: false,
                          ),
                          leftTitles: SideTitles(
                            showTitles: true,
                            getTextStyles: (context, value) => const
TextStyle(fontSize: 12),
                            interval: 20,
                          ),
                          bottomTitles: SideTitles(
                            showTitles: true,
                            getTextStyles: (context, value) => const
TextStyle(fontSize: 12),
                              getTitles: (value) {
                                switch (value.toInt()) {
                                  case 0:
                                    return '1';
                                  case 1:
                                    return '2';
                                  case 2:
                                    return '3';
                                  case 3:
                                    return '4';
                                  case 4:
                                    return '5';
                                  case 5:
                                    return '6';
                                  case 6:
                                    return '7';
                                  case 7:
                                    return '8';
```

```
                                case 8:
                                  return '9';
                                case 9:
                                  return '10';
                                default:
                                  return '';
                              }
                            }
                          //interval: 1,
                        ),
                      ),

                      axisTitleData: FlAxisTitleData(
                          leftTitle: AxisTitle(
                              showTitle: true,
                              titleText: 'Driving Score',
                              margin: 0
                          ),
                      ),

                      gridData: FlGridData(
                        show: true,
                      ),

                      lineBarsData: [
                        LineChartBarData(
                            isCurved: false,
                            dotData: FlDotData(
                              show: true,
                            ),
                            spots: getLineChartValue(_score)
                        )
                      ]
                  ),
                  ),
              ),
              SizedBox(
                  height:20
              ),
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: const [
                  Text(
                    "avg ",
                    style: TextStyle(fontSize: 20),
                  ),
                  Text(
                    "Score",
                    style: TextStyle(fontSize: 40),
                  ),
                ],
              ),
              Text(
                "${_avgScore.toString()}",
                style: TextStyle(color: Colors.black, fontSize: 100),
              ),
              SizedBox(
                  height:20
              ),
              Text(
                "Total driving : ${_numberOfDriving.toString()}
times\n\nTotal Time : ${transformSeconds(_totalTime)}\n\nTotal distance :
${_totalDistance.toStringAsPrecision(3)} km",
                style: TextStyle(color: Colors.black, fontSize: 15),
              ),
```

```
                  SizedBox(
                      height:30
                  ),
                  SizedBox(
                    width: 200,
                    height: 50,
                    child: RaisedButton.icon(
                        icon: Icon(Icons.home),
                        color: Colors.green,
                        label: Text(
                          'Home',
                          style: TextStyle(color: Colors.black, fontSize: 22),
                        ),
                        //textColor: Colors.grey,
                        onPressed:() {
                          Navigator.push(
                              context, MaterialPageRoute(
                              builder: (context) => Menu()
                          )
                          );
                        }
                    ),
                  ),
                  SizedBox(
                      height:20
                  ),
                ]
            )
        )
    )
      )
    );
  }

  Future<String> getAlgorithm() async{
    final prefs = await SharedPreferences.getInstance();
    return prefs.getString('algorithm').toString();
  }

  int getAvgScore(List<int> drivingScore) {
    var sum = 0;
    for (var i = 0; i < drivingScore.length; i++) {
      sum += drivingScore[i];
    }
    return sum ~/ drivingScore.length;
  }

  Future<List<int>> getData() async{
    late List<int> score = [];
    AlgorithmJson json = AlgorithmJson.fromJson(await
SessionManager().get('scoring'));
    await FirebaseFirestore.instance
        .collection('journeys')
        .limit(10)
        .where('userUID', isEqualTo: userUID())
        .orderBy('createdAt', descending: false)
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) async {
        List<dynamic> list = [];
        for (var i = 0; i < doc["speed"].length; i++) {
          list.add([doc["time"][i],doc["speed"][i],doc["geoPoint"][i]]);
          _totalTime ++;
        }
        await getDrivingScore(json,list).then((drivingScore) {
          setState(() {
```

```
          score.add(drivingScore);
          _totalDistance += getTripDistance(list).toDouble();
        });
      } );
    });
  });
  return Future.delayed(Duration(seconds: 1), () =>score);
}

transformSeconds(int seconds) {
  //int hundreds = (milliseconds / 10).truncate();
  //int seconds = (hundreds / 100).truncate();
  int minutes = (seconds / 60).truncate();
  int hours = (minutes / 60).truncate();
  String hoursStr = (hours % 60).toString();
  String minutesStr = (minutes % 60).toString();
  //String secondsStr = (seconds % 60).toString().padLeft(2, '0');
  return "$hoursStr hrs $minutesStr mins";
}

List<FlSpot> getLineChartValue(List<int> list) {
  List<FlSpot> spot = [];
  double index;
  double value;
  for (var i = 0; i < list.length; i++) {
    index = i.toDouble();
    value = list[i].toDouble();
    spot.add(FlSpot(index,value));
  }
  return spot;
}

String userUID() {
  final User? user = _auth.currentUser;
  if (user != null) {
    return user.uid.toString();
  }
  return "";
}

String userEmail() {
  final User? user = _auth.currentUser;
  if (user != null) {
    return user.email.toString();
  }
  return "";
}

Future <String> getNickName(String userUID) async {

  String nickname = "Anonymous";

  await FirebaseFirestore.instance
      .collection('users')
      .doc(userUID)
      .get()
      .then((DocumentSnapshot documentSnapshot) {
        if (documentSnapshot.exists) {
          nickname = documentSnapshot['nickname'];
        }
  });

  return nickname;
}

}
```

6.9 register.dart

Description: Responsible for registering new accounts for new users, it displays a form for users to fill out and saves user data to the cloud.

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:flutter_session_manager/flutter_session_manager.dart';
import 'package:fl_chart/fl_chart.dart';
import 'menu.dart';
import 'drivingScore.dart';
import 'setting.dart';
import 'algorithmJson.dart';


final FirebaseAuth _auth = FirebaseAuth.instance;

class Profile extends StatefulWidget {
 @override
 _Profile createState() => _Profile();

}

class _Profile extends State<Profile> {

 List<int> _score = [];
 int _avgScore = 0;
 var _email ="";
 int _numberOfDriving = 0;
 late int _totalTime = 0;
 double _totalDistance = 0;
 String groupName = '';
 String nickname = "Anonymous";

 _Profile() {
   getData().then((val) => setState(() {
     _score = val;
     _avgScore = getAvgScore(_score);
     _email = userEmail();
     _numberOfDriving = _score.length;
     _totalDistance = (_totalDistance / 1000);
   }));
   getNickName(userUID()).then((val) => setState(() {
     nickname = val;
   }));
 }

 @override
 void initState() {
   /*getAlgorithm().then((val) => setState(() {
     groupName = val;
   }));*/
   super.initState();
 }

 @override
 Widget build(BuildContext context) {
   return WillPopScope(
       onWillPop: () async {
     return Future.value(false);
   },
   child: MaterialApp(
```

```
    home: Scaffold(
        appBar: PreferredSize(
            preferredSize: Size(double.infinity, 60),
            child: AppBar(
              title: Text("${nickname}'s Profile"),
              centerTitle: true,
              backgroundColor: Colors.black,
              leading: IconButton(
                onPressed: () {
                  Navigator.push(
                      context, MaterialPageRoute(
                      builder: (context) => Menu()
                  ));
                },
                icon: Icon(Icons.home_outlined),
              ),
              actions: <Widget>[
                IconButton(
                  icon: const Icon(Icons.settings),
                  tooltip: 'Setting',
                  onPressed: () {
                    Navigator.push(
                        context, MaterialPageRoute(
                        builder: (context) => Setting()
                    ));
                  },
                ),
                ],
              elevation: 0, //remove shadow effect
            )
        ),
      body: Center(
        child: Container(
        alignment:Alignment.center,
            padding: EdgeInsets.only(
                left: 10,
                top:10,
                right: 10,
                bottom: 10
            ),
            child: Column (
                mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                crossAxisAlignment: CrossAxisAlignment.center,
                children: <Widget>[
                  Text(
                    "Last 10 driving records",
                    style: TextStyle(color: Colors.black, fontSize: 20),
                  ),
                  Container(
                    padding: EdgeInsets.all(10),
                    width: 380,
                    height: 200,
                    child: LineChart(LineChartData(
                      minX: 0,
                      minY: 0,
                      maxX: 10,
                      maxY: 100,
                        borderData: FlBorderData(show: false),
                        titlesData: FlTitlesData(
                          topTitles: SideTitles(
                            showTitles: false,
                          ),
                          rightTitles: SideTitles(
                            showTitles: false,
                          ),
                          leftTitles: SideTitles(
                            showTitles: true,
```

```
                            getTextStyles: (context, value) => const
TextStyle(fontSize: 12),
                            interval: 20,
                          ),
                          bottomTitles: SideTitles(
                            showTitles: true,
                            getTextStyles: (context, value) => const
TextStyle(fontSize: 12),
                              getTitles: (value) {
                                switch (value.toInt()) {
                                  case 0:
                                    return '1';
                                  case 1:
                                    return '2';
                                  case 2:
                                    return '3';
                                  case 3:
                                    return '4';
                                  case 4:
                                    return '5';
                                  case 5:
                                    return '6';
                                  case 6:
                                    return '7';
                                  case 7:
                                    return '8';
                                  case 8:
                                    return '9';
                                  case 9:
                                    return '10';
                                  default:
                                    return '';
                                }
                              }
                            //interval: 1,
                          ),
                        ),

                        axisTitleData: FlAxisTitleData(
                            leftTitle: AxisTitle(
                                showTitle: true,
                                titleText: 'Driving Score',
                                margin: 0
                            ),
                        ),

                        gridData: FlGridData(
                          show: true,
                        ),

                        lineBarsData: [
                          LineChartBarData(
                              isCurved: false,
                              dotData: FlDotData(
                                show: true,
                              ),
                              spots: getLineChartValue(_score)
                          )
                        ]
                    ),
                    ),
                ),
                SizedBox(
                    height:20
                ),
                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
```

```
                    children: const [
                      Text(
                        "avg ",
                        style: TextStyle(fontSize: 20),
                      ),
                      Text(
                        "Score",
                        style: TextStyle(fontSize: 40),
                      ),
                    ],
                  ),
                  Text(
                    "${_avgScore.toString()}",
                    style: TextStyle(color: Colors.black, fontSize: 100),
                  ),
                  SizedBox(
                      height:20
                  ),
                  Text(
                    "Total driving : ${_numberOfDriving.toString()}
times\n\nTotal Time : ${transformSeconds(_totalTime)}\n\nTotal distance :
${_totalDistance.toStringAsPrecision(3)} km",
                    style: TextStyle(color: Colors.black, fontSize: 15),
                  ),


                  SizedBox(
                      height:30
                  ),
                  SizedBox(
                    width: 200,
                    height: 50,
                    child: RaisedButton.icon(
                        icon: Icon(Icons.home),
                        color: Colors.green,
                        label: Text(
                          'Home',
                          style: TextStyle(color: Colors.black, fontSize: 22),
                        ),
                        //textColor: Colors.grey,
                        onPressed:() {
                          Navigator.push(
                              context, MaterialPageRoute(
                              builder: (context) => Menu()
                          )
                          );
                        }
                    ),
                  ),
                  SizedBox(
                      height:20
                  ),
                ]
            )
        )
    )
      )
    )
    );
  }

  Future<String> getAlgorithm() async{
    final prefs = await SharedPreferences.getInstance();
    return prefs.getString('algorithm').toString();
  }

  int getAvgScore(List<int> drivingScore) {
```

```
  var sum = 0;
  for (var i = 0; i < drivingScore.length; i++) {
    sum += drivingScore[i];
  }
  return sum ~/ drivingScore.length;
}

Future<List<int>> getData() async{
  late List<int> score = [];
  AlgorithmJson json = AlgorithmJson.fromJson(await
SessionManager().get('scoring'));
  await FirebaseFirestore.instance
      .collection('journeys')
      .limit(10)
      .where('userUID', isEqualTo: userUID())
      .orderBy('createdAt', descending: false)
      .get()
      .then((QuerySnapshot querySnapshot) {
    querySnapshot.docs.forEach((doc) async {
      List<dynamic> list = [];
      for (var i = 0; i < doc["speed"].length; i++) {
        list.add([doc["time"][i],doc["speed"][i],doc["geoPoint"][i]]);
        _totalTime ++;
      }
      await getDrivingScore(json,list).then((drivingScore) {
        setState(() {
          score.add(drivingScore);
          _totalDistance += getTripDistance(list).toDouble();
        });
      } );
    });
  });
  return Future.delayed(Duration(seconds: 1), () =>score);
}

transformSeconds(int seconds) {
  //int hundreds = (milliseconds / 10).truncate();
  //int seconds = (hundreds / 100).truncate();
  int minutes = (seconds / 60).truncate();
  int hours = (minutes / 60).truncate();
  String hoursStr = (hours % 60).toString();
  String minutesStr = (minutes % 60).toString();
  //String secondsStr = (seconds % 60).toString().padLeft(2, '0');
  return "$hoursStr hrs $minutesStr mins";
}

List<FlSpot> getLineChartValue(List<int> list) {
  List<FlSpot> spot = [];
  double index;
  double value;
  for (var i = 0; i < list.length; i++) {
    index = i.toDouble();
    value = list[i].toDouble();
    spot.add(FlSpot(index,value));
  }
  return spot;
}

String userUID() {
  final User? user = _auth.currentUser;
  if (user != null) {
    return user.uid.toString();
  }
  return "";
}

String userEmail() {
```

```
    final User? user = _auth.currentUser;
    if (user != null) {
      return user.email.toString();
    }
    return "";
  }

  Future <String> getNickName(String userUID) async {

    String nickname = "Anonymous";

    await FirebaseFirestore.instance
        .collection('users')
        .doc(userUID)
        .get()
        .then((DocumentSnapshot documentSnapshot) {
          if (documentSnapshot.exists) {
            nickname = documentSnapshot['nickname'];
          }
    });

    return nickname;
  }

}
```

6.10 result.dart

Description: It is responsible for analyzing and reporting driving results to users and saving driving information to the cloud.

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'dart:async';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:fl_chart/fl_chart.dart';
import 'package:flutter_session_manager/flutter_session_manager.dart';
import 'package:percent_indicator/percent_indicator.dart';
import 'algorithmJson.dart';
import 'drivingScore.dart';
import 'route.dart';
import 'review.dart';
import 'menu.dart';

class Result extends StatefulWidget {
  final List<dynamic> trip;
  final String mode;
  Result(this.mode,this.trip);
  @override
  ResultState createState() => ResultState();
}

class ResultState extends State<Result> {

  final PageController _pageController = PageController();
  final FirebaseAuth _auth = FirebaseAuth.instance;
  CollectionReference journeys =
FirebaseFirestore.instance.collection('journeys');
  late List<dynamic> list = [];
  String groupName = 'default';
  int drivingScore = 0;
  Color ring = Colors.black;
```

```
 getJson() async{
   AlgorithmJson json = AlgorithmJson.fromJson(await
SessionManager().get('scoring'));
   return json;
 }

 @override
 void initState() {
   super.initState();
   list = widget.trip;
   if (widget.mode == "monitor") {
     addRecord(list);
   }
   getJson().then((json) => setState(() {
     getDrivingScore(json,list).then((val) => setState(() {
       drivingScore = val;
       if (drivingScore >= 90) {
         ring = Colors.blueAccent;
       }
       else if (drivingScore >= 80) {
         ring = Colors.green;
       }
       else if (drivingScore >= 70) {
         ring = Colors.lightGreen;
       }
       else if (drivingScore >= 60) {
         ring = Colors.yellow;
       }
       else if (drivingScore >= 50) {
         ring = Colors.orange;
       }
       else {
         ring = Colors.red;
       }


     }));
   }));
 }
 @override
 Widget build(BuildContext context) {
   return WillPopScope(
       onWillPop: () async {
         return Future.value(false);
       },
       child: MaterialApp(
         home: Scaffold(
             appBar: PreferredSize(
                 preferredSize: Size(double.infinity, 60),
                 child: AppBar(
                   title: Text('Driving Score'),
                   centerTitle: true,
                   backgroundColor: Colors.black,
                   //leading: Icon(Icons.account_circle_rounded),
                   leading: IconButton(
                     onPressed: () {
                       if (widget.mode == "review") {
                         Navigator.push(
                             context, MaterialPageRoute(
                             builder: (context) => Review()
                         ));
                       }
                       else if (widget.mode == "monitor") {
                         Navigator.push(
                             context, MaterialPageRoute(
                             builder: (context) => Menu()
```

```
                        ));
                    }

                    },
                    icon: Icon(Icons.home_outlined),
                ),
                elevation: 0, //remove shadow effect
            )
        ),
        body: Center(
          child: Column(
              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
              crossAxisAlignment: CrossAxisAlignment.center,
              children: <Widget>[
                SizedBox (
                  height: 30,
                ),
                CircularPercentIndicator(
                  radius: 140.0,
                  lineWidth: 13.0,
                  animation: true,
                  //percent: 0.7,
                  percent: drivingScore / 100,
                  center: Text(
                    drivingScore.toString(),
                    style:
                    TextStyle(fontWeight: FontWeight.bold, fontSize: 120.0),
                  ),
                  /*footer: Text(
                        "Sales this week",
                        style:
                        TextStyle(fontWeight: FontWeight.bold, fontSize:
17.0),
                      ),*/
                  circularStrokeCap: CircularStrokeCap.round,
                  progressColor: ring,
                ),
                SizedBox(
                    height:30
                ),
                Text(
                  getEvaluate(drivingScore),
                  style: TextStyle(color: Colors.black, fontSize: 50),
                ),
                SizedBox(
                    height:30
                ),
                Text(
                  "Duration: ${list.last[0]}\n\nDistance:
${getTripDistance(list)} meters",
                  style: TextStyle(color: Colors.black, fontSize: 16),
                ),
                SizedBox(
                    height: 60
                ),
                SizedBox(
                  width: 160,
                  height: 50,
                  child: RaisedButton.icon(
                      icon: Icon(Icons.map),
                      color: Colors.green,
                      label: Text(
                        'Details',
                        style: TextStyle(color: Colors.black, fontSize: 22),
                      ),
                      //textColor: Colors.grey,
                      onPressed:() {
```

```
                            Navigator.push(
                                context, MaterialPageRoute(
                                builder: (context) =>
RoutePage(widget.mode,list)
                            ));
                          }
                      ),
                    ),
                    SizedBox(
                        height: 60
                    ),
                  ]
              ),
            ),
        ),
      ));
 }
 @override
 void dispose() {
   _pageController.dispose();
   super.dispose();
 }

 Future<void> addRecord(List<dynamic> list) {

   DateTime now = DateTime.now();
   now = now.subtract(Duration(seconds: list.length));
   String timeInFormat =
"${now.year.toString()}-${now.month.toString().padLeft(2,'0')}-${now.day.toString
().padLeft(2,'0')}
${now.hour.toString().padLeft(2,'0')}:${now.minute.toString().padLeft(2,'0')}:${n
ow.second.toString().padLeft(2,'0')}";
   DateTime savedTime = DateTime.parse(timeInFormat);

   String uid = "Anonymous";
   final User? user = _auth.currentUser;
   if (user != null) {
     uid = user.uid;
   }

   List<String> time = [];
   List<int> speed = [];
   List<GeoPoint> geoPoint = [];
   for (var i = 0; i < list.length; i++) {
     time.add(list[i][0]);
     speed.add(list[i][1]);
     geoPoint.add(GeoPoint(list[i][2].latitude,list[i][2].longitude));
   }
   return journeys
       .add(
       {
         'createdAt' : savedTime,
         'userUID': uid,
         'time' : time,
         'speed' : speed,
         'geoPoint' : geoPoint
       }
   )
       .then((value) => print("record saved"))
       .catchError((error) => print(error));
 }

 List<FlSpot> getLineChartValue(List<dynamic> list) {
   List<FlSpot> spot = [];
   double index;
   double value;
   for (var i = 0; i < list.length; i++) {
```

```
      index = i.toDouble();
      value = list[i][1].toDouble();
      spot.add(FlSpot(index,value));
    }
    return spot;
  }
}
```

6.11 review.dart

Description: It is responsible for listing all previous driving records of the logged in user and providing a way back to those reports.

```
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:geolocator/geolocator.dart';
import 'result.dart';
import 'menu.dart';

final FirebaseAuth _auth = FirebaseAuth.instance;

class Review extends StatelessWidget {

 CollectionReference journeys =
FirebaseFirestore.instance.collection('journeys');
 late List<dynamic> trip = [];

 @override
 Widget build(BuildContext context) {
   return WillPopScope (
       onWillPop: () async {
     return Future.value(false);
   },
   child: MaterialApp(
       home: Scaffold(
         appBar: PreferredSize(
             preferredSize: Size(double.infinity, 60),
             child: AppBar(
               title: Text('Journey Histories'),
               centerTitle: true,
               backgroundColor: Colors.black,
               leading: IconButton(
                 onPressed: () {
                   Navigator.push(
                       context, MaterialPageRoute(
                       builder: (context) => Menu()
                   ));
                 },
                 icon: Icon(Icons.home_outlined),
               ),
               elevation: 0, //remove shadow effect
             )
         ),
         body: FutureBuilder<QuerySnapshot> (
           future: journeys
               .limit(30)
               .where('userUID', isEqualTo: userUID())
               .orderBy('createdAt', descending: true)
               .get(),
           builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot)
{
             if (snapshot.hasError) {
```

```
                return Center(
                    child: Text(
                        "Database Error",
                        style: TextStyle(color: Colors.black, fontSize: 28))
                );
              }
              if (snapshot.data!.size == 0) {
                return Center(
                    child: Text(
                        "No Record",
                        style: TextStyle(color: Colors.black, fontSize: 28))
                );
              }
              if (snapshot.connectionState == ConnectionState.done) {
                return ListView(
                  children: snapshot.data!.docs.map((DocumentSnapshot document) {
                    Map<String, dynamic> data = document.data()! as Map<String,
dynamic>;
                    var documentID = document.reference.id;

                    return ListTile(
                        leading: Icon(
                            Icons.directions_car,
                            size: 36.0,
                        ),
                        title: Text(
                            "Date:
${data['createdAt'].toDate().toString().substring(0,19)}",
                            style: TextStyle(
                                color: Colors.black,
                                fontSize: 12
                            )
                        ),
                        subtitle: Text(
                            "Duration: ${data['time'][data['time'].length -1]}
Distance: ${getTripDistance(data['geoPoint']).toString()} meters",
                            style: TextStyle(
                                color: Colors.grey,
                                fontSize: 8
                            )
                        ),

                        trailing: RaisedButton (
                          child: Text('Details'),
                          textColor: Colors.grey,
                          onPressed: () async{
                            await getData(documentID);
                            Navigator.push(
                              context, MaterialPageRoute(
                              builder: (context) => Result("review",trip))
                            );
                          }
                        )
                    );
                  }).toList(),
                );
              }
              return Center(
                child: CircularProgressIndicator()
              );
            }
          )
        ),
      )
    );
  }
```

```
 String userUID() {
   final User? user = _auth.currentUser;
   if (user != null) {
     return user.uid.toString();
   }
   return "";
 }

 Future<void> getData(String documentID) async {
   trip = [];
   List<dynamic> time = [];
   List<dynamic> speed = [];
   List<dynamic> location = [];

   var collection = FirebaseFirestore.instance.collection('journeys');
   var document = await collection.doc(documentID).get();

   if (document.exists) {
     Map<String, dynamic>? data = document.data();
     //test3 = document.data();

     data?["time"].forEach((v) => time.add(v));
     data?["speed"].forEach((v) => speed.add(v));
     data?["geoPoint"].forEach((v) => location.add(v));

     for (var i = 0; i < time.length; i++) {
       trip.add([time[i],speed[i],location[i]]);
     }
   }
 }

 int getTripDistance(List<dynamic> list) {
   var distance = 0.0;
   var totalDistance = 0.0;
   var currentlatitude = 0.0;
   var currentlongitude = 0.0;
   var lastlatitude = 0.0;
   var lastlongitude = 0.0;
   for (var i = 0; i < list.length -2; i++) {
     lastlatitude = list[i].latitude;
     lastlongitude = list[i].longitude;
     currentlatitude = list[i+1].latitude;
     currentlongitude = list[i+1].longitude;
     if (lastlatitude != 0.0 || lastlongitude != 0.0 || currentlatitude != 0.0 ||
currentlongitude != 0.0) {
       distance = Geolocator.distanceBetween(
           lastlatitude,lastlongitude,
           currentlatitude,currentlongitude
       );
       totalDistance += distance;
     }
   }
   return totalDistance.toInt();
 }
}
```

6.12 route.dart

Description: Its function is to display the driving map, import the driving data into the map, realize the planning of the route and indicate the location of driving mistakes. It will also display the driver's Speeding, Acceleration, Deceleration ability.

```dart
import 'package:flutter/material.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:geolocator/geolocator.dart';
import 'result.dart';
import 'menu.dart';

final FirebaseAuth _auth = FirebaseAuth.instance;

class Review extends StatelessWidget {

 CollectionReference journeys =
FirebaseFirestore.instance.collection('journeys');
 late List<dynamic> trip = [];

 @override
 Widget build(BuildContext context) {
   return WillPopScope (
       onWillPop: () async {
     return Future.value(false);
   },
   child: MaterialApp(
       home: Scaffold(
         appBar: PreferredSize(
             preferredSize: Size(double.infinity, 60),
             child: AppBar(
               title: Text('Journey Histories'),
               centerTitle: true,
               backgroundColor: Colors.black,
               leading: IconButton(
                 onPressed: () {
                   Navigator.push(
                       context, MaterialPageRoute(
                       builder: (context) => Menu()
                   ));
                 },
                 icon: Icon(Icons.home_outlined),
               ),
               elevation: 0, //remove shadow effect
             )
         ),
         body: FutureBuilder<QuerySnapshot> (
           future: journeys
               .limit(30)
               .where('userUID', isEqualTo: userUID())
               .orderBy('createdAt', descending: true)
               .get(),
           builder: (BuildContext context, AsyncSnapshot<QuerySnapshot> snapshot)
{
             if (snapshot.hasError) {
               return Center(
                   child: Text(
                       "Database Error",
                       style: TextStyle(color: Colors.black, fontSize: 28))
               );
             }
             if (snapshot.data!.size == 0) {
               return Center(
                   child: Text(
                       "No Record",
                       style: TextStyle(color: Colors.black, fontSize: 28))
               );
             }
             if (snapshot.connectionState == ConnectionState.done) {
               return ListView(
                 children: snapshot.data!.docs.map((DocumentSnapshot document) {
                   Map<String, dynamic> data = document.data()! as Map<String,
```

```
dynamic>;
                    var documentID = document.reference.id;

                    return ListTile(
                        leading: Icon(
                            Icons.directions_car,
                            size: 36.0,
                        ),
                        title: Text(
                            "Date:
${data['createdAt'].toDate().toString().substring(0,19)}",
                            style: TextStyle(
                                color: Colors.black,
                                fontSize: 12
                            )
                        ),
                        subtitle: Text(
                            "Duration: ${data['time'][data['time'].length -1]}
Distance: ${getTripDistance(data['geoPoint']).toString()} meters",
                            style: TextStyle(
                                color: Colors.grey,
                                fontSize: 8
                            )
                        ),

                        trailing: RaisedButton (
                          child: Text('Details'),
                          textColor: Colors.grey,
                          onPressed: () async{
                            await getData(documentID);
                            Navigator.push(
                              context, MaterialPageRoute(
                              builder: (context) => Result("review",trip))
                            );
                          }
                        )
                      );
                  }).toList(),
              );
            }
            return Center(
              child: CircularProgressIndicator()
            );
          }
        )
      ),
    )
  );
 }

 String userUID() {
   final User? user = _auth.currentUser;
   if (user != null) {
     return user.uid.toString();
   }
   return "";
 }

 Future<void> getData(String documentID) async {
   trip = [];
   List<dynamic> time = [];
   List<dynamic> speed = [];
   List<dynamic> location = [];

   var collection = FirebaseFirestore.instance.collection('journeys');
   var document = await collection.doc(documentID).get();
```

```
    if (document.exists) {
      Map<String, dynamic>? data = document.data();
      //test3 = document.data();

      data?["time"].forEach((v) => time.add(v));
      data?["speed"].forEach((v) => speed.add(v));
      data?["geoPoint"].forEach((v) => location.add(v));

      for (var i = 0; i < time.length; i++) {
        trip.add([time[i],speed[i],location[i]]);
      }
    }
  }

  int getTripDistance(List<dynamic> list) {
    var distance = 0.0;
    var totalDistance = 0.0;
    var currentlatitude = 0.0;
    var currentlongitude = 0.0;
    var lastlatitude = 0.0;
    var lastlongitude = 0.0;
    for (var i = 0; i < list.length -2; i++) {
      lastlatitude = list[i].latitude;
      lastlongitude = list[i].longitude;
      currentlatitude = list[i+1].latitude;
      currentlongitude = list[i+1].longitude;
      if (lastlatitude != 0.0 || lastlongitude != 0.0 || currentlatitude != 0.0 ||
currentlongitude != 0.0) {
        distance = Geolocator.distanceBetween(
            lastlatitude,lastlongitude,
            currentlatitude,currentlongitude
        );
        totalDistance += distance;
      }
    }
    return totalDistance.toInt();
  }
}
```

6.13 setAlgorithm.dart

Description: It is mainly for admin users to modify the value of the scoring method, which is displayed in a table, which are the values of Speeding, Accelerating, and Decelerating three kinds of data.

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'dart:async';
import 'dart:convert';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:fl_chart/fl_chart.dart';
import 'package:flutter_session_manager/flutter_session_manager.dart';
import 'algorithmJson.dart';
import 'drivingScore.dart';
import 'route.dart';
import 'review.dart';
import 'menu.dart';

class SetAlgorithm extends StatefulWidget {
  final String algorithm;
  final Map<String, dynamic> list;
```

```
 SetAlgorithm(this.algorithm, this.list);
 @override
 SetAlgorithmState createState() => SetAlgorithmState();
}

class SetAlgorithmState extends State<SetAlgorithm> {

 final GlobalKey<FormState> _formKey = GlobalKey<FormState>();



 @override
 void initState() {

   super.initState();
 }

 @override
 Widget build(BuildContext context) {

   final TextEditingController _speedingController = TextEditingController(text:
widget.list['speeding'].toString());
   final TextEditingController _acceleratingController =
TextEditingController(text: widget.list['accelerating'].toString());
   final TextEditingController _brakingController = TextEditingController(text:
widget.list['braking'].toString());

   return WillPopScope(
       onWillPop: () async {
         return Future.value(false);
       },
       child: MaterialApp(
         home: Scaffold(
             appBar: PreferredSize(
                 preferredSize: Size(double.infinity, 60),
                 child: AppBar(
                   title: Text('Algorithm Setup'),
                   centerTitle: true,
                   backgroundColor: Colors.black,
                   //leading: Icon(Icons.account_circle_rounded),
                   leading: IconButton(
                     onPressed: () {
                       Navigator.push(
                           context, MaterialPageRoute(
                           builder: (context) => Menu()
                       ));
                     },
                     icon: Icon(Icons.home_outlined),
                   ),
                   elevation: 0, //remove shadow effect
                 )
             ),

             body: Form (
               key: _formKey,
               child: Container(
                 alignment:Alignment.center,
                 padding: EdgeInsets.only(
                     left: 10,
                     top:10,
                     right: 10,
                     bottom: 10
                 ),
                 color: Color.fromARGB(255, 255, 255, 255),
                 child: Column(
                   mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                   crossAxisAlignment: CrossAxisAlignment.center,
```

```
                  children: <Widget>[
                    SizedBox(
                        height:20
                    ),
                    Container(
                      width: 200,
                      height: 200,
                      child: Image.asset(
                        'assets/images/edit_list.png',
                      ),
                    ),
                    SizedBox(
                        height:30
                    ),
                    Text(
                      "${widget.algorithm}",
                      style: TextStyle(color: Colors.black, fontSize: 40),
                    ),
                    SizedBox(
                        height:20
                    ),
                    SizedBox(
                      width:250,
                      child: TextFormField(
                        controller: _speedingController,
                        decoration: const InputDecoration(
                            labelText: 'Speeding : ',
                            contentPadding: EdgeInsets.symmetric(vertical: 15.0,
horizontal: 10.0),
                            border: OutlineInputBorder(
                                borderSide: BorderSide(color: Colors.black)
                            )
                        ),
                        style: TextStyle(fontWeight: FontWeight.normal,
fontSize: 20, color: Colors.black),

                        validator: (String? value) {
                          if (value!.isEmpty || json.decode(value).length != 5)
{
                            return 'List Format: [1,2,3,4,5]';
                          }
                          else {
                            return null;
                          }
                        },
                      ),
                    ),
                    SizedBox(
                        height:20
                    ),
                    SizedBox(
                      width:250,
                      child: TextFormField(
                        controller: _acceleratingController,
                        decoration: const InputDecoration(
                            labelText: 'Accelerating : ',
                            contentPadding: EdgeInsets.symmetric(vertical: 15.0,
horizontal: 10.0),
                            border: OutlineInputBorder(
                                borderSide: BorderSide(color: Colors.black)
                            )
                        ),
                        style: TextStyle(fontWeight: FontWeight.normal,
fontSize: 20, color: Colors.black),
                        validator: (String? value) {
                          if (value!.isEmpty || json.decode(value).length != 5)
{
```

```
                            return 'List Format: [1,2,3,4,5]';
                          }
                          else {
                            return null;
                          }
                        },
                      ),
                    ),
                    SizedBox(
                        height:20
                    ),
                    SizedBox(
                      width:250,
                      child: TextFormField(
                        controller: _brakingController,
                        decoration: const InputDecoration(
                            labelText: 'Braking : ',
                            contentPadding: EdgeInsets.symmetric(vertical: 15.0,
horizontal: 10.0),
                            border: OutlineInputBorder(
                                borderSide: BorderSide(color: Colors.black)
                            )
                        ),
                        style: TextStyle(fontWeight: FontWeight.normal,
fontSize: 20, color: Colors.black),
                        validator: (String? value) {
                          if (value!.isEmpty || json.decode(value).length != 5)
{
                            return 'List Format: [1,2,3,4,5]';
                          }
                          else {
                            return null;
                          }
                        },
                      ),
                    ),
                    SizedBox(
                        height:30
                    ),
                    SizedBox(
                      width: 200,
                      height: 80,
                      child: ElevatedButton(
                        onPressed: () async {
                          var method = widget.algorithm;
                          var speeding = json.decode(_speedingController.text);
                          var accelerating =
json.decode(_acceleratingController.text);
                          var braking = json.decode(_brakingController.text);

                          /*if (_formKey.currentState!.validate()) {
                          await saveData(method,speeding,accelerating,braking);
                        }*/
                        },
                        child: const Text(
                            'Submit',
                            style: TextStyle(color: Colors.white, fontSize: 30),
                        ),
                      ),
                    ),
                    SizedBox(
                        height:30
                    ),
                  ] ,
                ),
              ),
            )
```

```
          ),
        ));
  }

  @override
  void dispose() {
    super.dispose();
  }

  Future<void> saveData(String method, List<dynamic> speeding, List<dynamic>
accelerating, List<dynamic> braking) async {
    FirebaseFirestore.instance.collection('algorithm')
        .doc(method)
        .update({'speeding': speeding, 'accelerating' : accelerating, 'braking' :
braking})
        .then((value) => print("Data updated"))
        .catchError((error) => print("Failed to update user: $error"));
  }
}
```

6.14 setting.dart

Description: It provides options for regular users to choose their preferred scoring method.

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter_session_manager/flutter_session_manager.dart';
import 'algorithmJson.dart';
import 'menu.dart';


final FirebaseAuth _auth = FirebaseAuth.instance;

class Setting extends StatefulWidget {
  @override
  _SettingState createState() => _SettingState();

}

class _SettingState extends State<Setting> {

  String dropdownValue = 'default';
  late List<DropdownMenuItem<String>> dropdownMenu = [];

  @override
  void initState() {
    getScoringMethod().then((val) => setState(() {
      dropdownValue = val;
    }));
    dropdownItemList().then((items) => setState(() {
      dropdownMenu = items;
    }));
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return WillPopScope(
        onWillPop: () async {
          return Future.value(false);
        },
```

```
      child: MaterialApp(
          home: Scaffold(
              appBar: PreferredSize(
                  preferredSize: Size(double.infinity, 60),
                  child: AppBar(
                    title: Text('Profile'),
                    centerTitle: true,
                    backgroundColor: Colors.black,
                    leading: IconButton(
                      onPressed: () {
                        Navigator.push(
                            context, MaterialPageRoute(
                            builder: (context) => Menu()
                        ));
                      },
                      icon: Icon(Icons.home_outlined),
                    ),
                    actions: <Widget>[
                      IconButton(
                        icon: const Icon(Icons.settings),
                        tooltip: 'Setting',
                        onPressed: () {
                          ScaffoldMessenger.of(context).showSnackBar(
                              const SnackBar(content: Text('This is a
snackbar')));
                        },
                      ),
                    ],
                    elevation: 0, //remove shadow effect
                  )
              ),
              body: Center(
                  child: Container(
                      alignment:Alignment.center,
                      padding: EdgeInsets.only(
                          left: 10,
                          top:10,
                          right: 10,
                          bottom: 10
                      ),
                      child: Column (
                          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                          crossAxisAlignment: CrossAxisAlignment.center,
                          children: <Widget>[
                            SizedBox(
                                height:60
                            ),
                            Container(
                              width: 250,
                              height: 250,
                              child: Image.asset(
                                'assets/images/scoring.png',
                              ),
                            ),
                            SizedBox(
                                height:60
                            ),
                            Text(
                              "Difficulty Level",
                              style: TextStyle(color: Colors.grey, fontSize:
30),
                            ),
                            SizedBox(
                                height:20
                            ),
                            Container(
                              color: Colors.white,
```

```
                                height: 50,
                                width: 200,
                                alignment: Alignment.center,
                                // set width equal to height to make a square
                                child: DropdownButton<String>(
                                    value: dropdownValue,
                                    //elevation: 156,
                                    style: const TextStyle(color: Colors.black,
fontSize: 30),
                                    //itemHeight: 20,
                                    iconEnabledColor: Colors.black,
                                    //focusColor: Colors.purple,
                                    //dropdownColor: Colors.purple,
                                    //borderRadius:
BorderRadius.all(Radius.circular(20)),
                                    onChanged: (String? newValue){
                                      setState(() {
                                        dropdownValue = newValue!;
                                        print(newValue);
                                      });
                                    },
                                    items: dropdownMenu
                                ),
                            ),
                            SizedBox(
                                height:60
                            ),
                            Row(
                              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                              crossAxisAlignment: CrossAxisAlignment.center,
                              children: [
                                SizedBox(
                                    width:50
                                ),
                                SizedBox(
                                  width: 120,
                                  height: 50,
                                  child: RaisedButton.icon(
                                      icon: Icon(Icons.save_alt),
                                      color: Colors.green,
                                      label: Text(
                                        'Save',
                                        style: TextStyle(color: Colors.black,
fontSize: 22),
                                      ),
                                      //textColor: Colors.grey,
                                      onPressed:() {
                                        setScoringMethod(dropdownValue);
                                        setSession(dropdownValue);
                                        Navigator.push(
                                            context, MaterialPageRoute(
                                            builder: (context) => Menu()
                                        )
                                        );
                                      }
                                  ),
                                ),
                              ],
                            ),

                            SizedBox(
                                height:100
                            ),
                          ]
                      )
                  )
              )
```

```
          )
        )
    );
  }

  String userUID() {
    final User? user = _auth.currentUser;
    if (user != null) {
      return user.uid.toString();
    }
    return "";
  }

  String userEmail() {
    final User? user = _auth.currentUser;
    if (user != null) {
      return user.email.toString();
    }
    return "";
  }

  Future<void> setScoringMethod(String algorithmName) async {
    FirebaseFirestore.instance.collection('users')
        .doc(userUID())
        .update({'algorithm': algorithmName})
        .then((value) => print("Data updated"))
        .catchError((error) => print("Failed to update user: $error"));
  }

  Future<void> setSession(String algorithmName) async{
    var collection = FirebaseFirestore.instance.collection('algorithm');
    var document = await collection.doc(algorithmName).get();
    Map<String, dynamic> rule = {};
    if (document.exists) {
      Map<String, dynamic>? data = document.data();

      print(data.toString());
      rule['speeding'] = data?['speeding'];
      rule['braking'] = data?['braking'];
      rule['accelerating'] = data?['accelerating'];
      List<int> speeding = [];
      for (var i = 0; i < rule['speeding'].length; i++) {
        speeding.add(rule['speeding'][i].toInt());
      }
      List<int> braking = [];
      for (var i = 0; i < rule['braking'].length; i++) {
        braking.add(rule['braking'][i].toInt());
      }
      List<int> accelerating = [];
      for (var i = 0; i < rule['accelerating'].length; i++) {
        accelerating.add(rule['accelerating'][i].toInt());
      }
      AlgorithmJson scoring = AlgorithmJson(speeding,braking,accelerating);
      await SessionManager().remove('scoring');
      await SessionManager().set('scoring',scoring);
    }
  }


  Future<List<DropdownMenuItem<String>>> dropdownItemList() async {
    List<DropdownMenuItem<String>> menuItems = [];
    await FirebaseFirestore.instance
        .collection('algorithm')
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) async {
        menuItems.add(DropdownMenuItem(child:
```

```
Text(doc.reference.id.toString()),value: doc.reference.id.toString()),);
      });
    });
    return menuItems;
  }

  Future<String> getScoringMethod() async{
    return await SessionManager().get('algorithm');
  }

}
```

6.15 settingAdmin.dart

Description: Description: It provides options for admin users to choose their preferred scoring method.

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter_session_manager/flutter_session_manager.dart';
import 'algorithmJson.dart';
import 'menu.dart';
import 'setAlgorithm.dart';

final FirebaseAuth _auth = FirebaseAuth.instance;

class SettingAdmin extends StatefulWidget {
 @override
 _SettingAdminState createState() => _SettingAdminState();
}

class _SettingAdminState extends State<SettingAdmin> {

 String dropdownValue = '';
 late List<DropdownMenuItem<String>> dropdownMenu = [];

 @override
 void initState() {
   getScoringMethod().then((val) => setState(() {
     dropdownValue = val;
   }));
   dropdownItemList().then((items) => setState(() {
     dropdownMenu = items;
   }));
   super.initState();
 }

 @override
 Widget build(BuildContext context) {
   return WillPopScope(
       onWillPop: () async {
         return Future.value(false);
       },
       child: MaterialApp(
           home: Scaffold(
               appBar: PreferredSize(
                   preferredSize: Size(double.infinity, 60),
                   child: AppBar(
                     title: Text('SCORING METHOD'),
                     centerTitle: true,
                     backgroundColor: Colors.black,
```

```
                    leading: IconButton(
                      onPressed: () {
                        Navigator.push(
                            context, MaterialPageRoute(
                            builder: (context) => Menu()
                        ));
                      },
                      icon: Icon(Icons.home_outlined),
                    ),
                    actions: <Widget>[
                      IconButton(
                        icon: const Icon(Icons.settings),
                        tooltip: 'Setting',
                        onPressed: () {
                          ScaffoldMessenger.of(context).showSnackBar(
                              const SnackBar(content: Text('This is a
snackbar')));
                        },
                      ),
                    ],
                    elevation: 0, //remove shadow effect
                  )
              ),
              body: Center(
                  child: Container(
                      alignment:Alignment.center,
                      padding: EdgeInsets.only(
                          left: 10,
                          top:10,
                          right: 10,
                          bottom: 10
                      ),
                      child: Column (
                          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                          crossAxisAlignment: CrossAxisAlignment.center,
                          children: <Widget>[
                            SizedBox(
                                height:60
                            ),
                            Container(
                              width: 250,
                              height: 250,
                              child: Image.asset(
                                'assets/images/scoring.png',
                              ),
                            ),
                            SizedBox(
                                height:60
                            ),
                            Text(
                              "Difficulty Level",
                              style: TextStyle(color: Colors.grey, fontSize:
30),
                            ),
                            SizedBox(
                                height:20
                            ),
                            Container(
                              color: Colors.white,
                              height: 50,
                              width: 200,
                              alignment: Alignment.center,
                              // set width equal to height to make a square
                              child: DropdownButton<String>(
                                  value: dropdownValue,
                                  //elevation: 156,
                                  style: const TextStyle(color: Colors.black,
```

```
fontSize: 30),
                                    //itemHeight: 20,
                                    iconEnabledColor: Colors.black,
                                    //focusColor: Colors.purple,
                                    //dropdownColor: Colors.purple,
                                    //borderRadius:
BorderRadius.all(Radius.circular(20)),
                                    onChanged: (String? newValue){
                                      setState(() {
                                        dropdownValue = newValue!;
                                        print(newValue);
                                      });
                                    },
                                    items: dropdownMenu
                                ),
                            ),
                            SizedBox(
                                height:60
                            ),
                            Row(
                              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
                              crossAxisAlignment: CrossAxisAlignment.center,
                              children: [
                                SizedBox(
                                    width:50
                                ),
                                SizedBox(
                                  width: 120,
                                  height: 50,
                                  child: RaisedButton.icon(
                                      icon: Icon(Icons.save_alt),
                                      color: Colors.green,
                                      label: Text(
                                        'Save',
                                        style: TextStyle(color: Colors.black,
fontSize: 22),
                                      ),
                                      //textColor: Colors.grey,
                                      onPressed:() {
                                        setScoringMethod(dropdownValue);
                                        setSession(dropdownValue);
                                        Navigator.push(
                                            context, MaterialPageRoute(
                                            builder: (context) => Menu()
                                        )
                                        );
                                      }
                                  ),
                                ),
                                SizedBox(
                                    width:10
                                ),
                                SizedBox(
                                  width: 120,
                                  height: 50,
                                  child: RaisedButton.icon(
                                      icon: Icon(Icons.edit),
                                      color: Colors.red,
                                      label: Text(
                                        'Edit',
                                        style: TextStyle(color: Colors.black,
fontSize: 22),
                                      ),
                                      //textColor: Colors.grey,
                                      onPressed:() async{
                                        setScoringMethod(dropdownValue);
                                        setSession(dropdownValue);
```

```
                                        Map<String, dynamic> list = await
getData(dropdownValue);
                                        Navigator.push(
                                            context, MaterialPageRoute(
                                            builder: (context) =>
SetAlgorithm(dropdownValue,list)
                                        )
                                        );
                                    }
                                ),
                            ),
                            SizedBox(
                                width:50
                            ),
                        ],
                    ),

                    SizedBox(
                        height:100
                    ),
                ]
            )
        )
    )
  );
 }
 @override
 void dispose() {
   super.dispose();
 }

 String userUID() {
   final User? user = _auth.currentUser;
   if (user != null) {
     return user.uid.toString();
   }
   return "";
 }

 String userEmail() {
   final User? user = _auth.currentUser;
   if (user != null) {
     return user.email.toString();
   }
   return "";
 }

 Future<void> setScoringMethod(String algorithmName) async {
   FirebaseFirestore.instance.collection('users')
       .doc(userUID())
       .update({'algorithm': algorithmName})
       .then((value) => print("Data updated"))
       .catchError((error) => print("Failed to update user: $error"));
 }

 Future<void> setSession(String algorithmName) async{
   var collection = FirebaseFirestore.instance.collection('algorithm');
   var document = await collection.doc(algorithmName).get();
   Map<String, dynamic> rule = {};
   if (document.exists) {
     Map<String, dynamic>? data = document.data();

     print(data.toString());
     rule['speeding'] = data?['speeding'];
     rule['braking'] = data?['braking'];
```

```
      rule['accelerating'] = data?['accelerating'];
      List<int> speeding = [];
      for (var i = 0; i < rule['speeding'].length; i++) {
        speeding.add(rule['speeding'][i].toInt());
      }
      List<int> braking = [];
      for (var i = 0; i < rule['braking'].length; i++) {
        braking.add(rule['braking'][i].toInt());
      }
      List<int> accelerating = [];
      for (var i = 0; i < rule['accelerating'].length; i++) {
        accelerating.add(rule['accelerating'][i].toInt());
      }
      AlgorithmJson scoring = AlgorithmJson(speeding,braking,accelerating);
      await SessionManager().remove('scoring');
      await SessionManager().set('scoring',scoring);
      await SessionManager().remove('algorithm');
      await SessionManager().set('algorithm',dropdownValue);
    }
  }


  Future<List<DropdownMenuItem<String>>> dropdownItemList() async {
    List<DropdownMenuItem<String>> menuItems = [];
    await FirebaseFirestore.instance
        .collection('algorithm')
        .get()
        .then((QuerySnapshot querySnapshot) {
      querySnapshot.docs.forEach((doc) async {
        menuItems.add(DropdownMenuItem(child:
Text(doc.reference.id.toString()),value: doc.reference.id.toString()),);
      });
    });
    return menuItems;
  }

  Future<String> getScoringMethod() async{
    return await SessionManager().get('algorithm');
  }

  Future<Map<String, dynamic>> getData(String method) async{

    Map<String, dynamic> list = {};
    var collection = FirebaseFirestore.instance.collection('algorithm');
    var document = await collection.doc(method).get();
    if (document.exists) {
      Map<String, dynamic>? data = document.data();
      list['speeding'] = data?['speeding'];
      list['braking'] = data?['braking'];
      list['accelerating'] = data?['accelerating'];
    }
    return list;
  }


}
```

6.16 SignInPage.dart

Description: Fill out a form for the old user. After passing the Firebase authentication, create a user session and log in to use other functions.

```
import 'package:flutter/material.dart';
```

```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter_signin_button/flutter_signin_button.dart';
import 'menu.dart';
import 'register.dart';

final FirebaseAuth _auth = FirebaseAuth.instance;

/// Entrypoint example for various sign-in flows with Firebase.
class SignInPage extends StatefulWidget {
 SignInPage({Key? key}) : super(key: key);
 /// The page title.
 final String title = 'Sign In & Out';
 @override
 State<StatefulWidget> createState() => _SignInPageState();
}

class _SignInPageState extends State<SignInPage> {
 User? user;

 @override
 void initState() {
   _auth.userChanges().listen(
         (event) => setState(() => user = event),
   );
   super.initState();
 }

 @override
 Widget build(BuildContext context) {
   return WillPopScope(
       onWillPop: () async {
     return Future.value(false);
     },
     child: Scaffold(
       appBar: PreferredSize(
           preferredSize: Size(double.infinity, 0),
           child: AppBar(
             backgroundColor: Colors.black,
             elevation: 0, //remove shadow effect
           )
       ),
     body: Builder(
       builder: (BuildContext context) {
         return Center(
           child: Container(
             alignment:Alignment.center,
             padding: EdgeInsets.only(
                 left: 30,
                 top:10,
                 right: 30,
                 bottom: 10
             ),
             color: Color.fromARGB(255, 37, 141, 229),
             child: Column (
               mainAxisAlignment: MainAxisAlignment.spaceEvenly,
               crossAxisAlignment: CrossAxisAlignment.center,
               children: <Widget>[
                 _EmailPasswordForm(),
               ]
             )
           )
         );
       },
     ),
   ));
 }
 @override
```

```
 void dispose() {
   super.dispose();
 }
}

class _EmailPasswordForm extends StatefulWidget {
 const _EmailPasswordForm({Key? key}) : super(key: key);

 @override
 State<StatefulWidget> createState() => _EmailPasswordFormState();
}

class _EmailPasswordFormState extends State<_EmailPasswordForm> {
 final GlobalKey<FormState> _formKey = GlobalKey<FormState>();
 final TextEditingController _emailController = TextEditingController();
 final TextEditingController _passwordController = TextEditingController();

 @override
 Widget build(BuildContext context) {
   return Form(
     key: _formKey,
     child: Card(
       child: Padding(
         padding: const EdgeInsets.all(16),
         child: Column(
           crossAxisAlignment: CrossAxisAlignment.center,
           children: <Widget>[
             Container(
               width: 200,
               height: 170,
               child: Image.asset(
                 'assets/images/icon1.png',
               ),
             ),
             /*Container(
               alignment: Alignment.center,
               child: const Text(
                 'Sign in with email and password',
                 style: TextStyle(fontWeight: FontWeight.bold),
               ),
             ),
             SizedBox(
                 height:50
             ),*/
             TextFormField(
               controller: _emailController,
               decoration: const InputDecoration(labelText: 'Email'),
               validator: (String? value) {
                 if (value!.isEmpty) return 'Please enter some text';
                 return null;
               },
             ),
             TextFormField(
               controller: _passwordController,
               decoration: const InputDecoration(labelText: 'Password'),
               validator: (String? value) {
                 if (value!.isEmpty) return 'Please enter some text';
                 return null;
               },
               obscureText: true,
             ),
             TextButton(
               style: TextButton.styleFrom(
                 textStyle: const TextStyle(fontSize: 14),
               ),
               onPressed: () {
                 Navigator.push(
```

```
                    context, MaterialPageRoute(
                    builder: (context) => RegisterPage()
                )
                );
              },
              child: const Text('Register new account
\n'),
            ),
            Container(
              padding: const EdgeInsets.only(top: 16),
              alignment: Alignment.center,
              child: SignInButton(
                Buttons.Email,
                text: 'Sign In',
                onPressed: () async {
                  if (_formKey.currentState!.validate()) {
                    await _signInWithEmailAndPassword();
                  }
                },
              ),
            ),
            SizedBox(
                height:50
            ),
          ],
        ),
      ),
    ),
  );
 }

 @override
 void dispose() {
   _emailController.dispose();
   _passwordController.dispose();
   super.dispose();
 }

 // Example code of how to sign in with email and password.
 Future<void> _signInWithEmailAndPassword() async {
   try {
     final User user = (await _auth.signInWithEmailAndPassword(
       email: _emailController.text,
       password: _passwordController.text,
     )).user!;
     Navigator.push(
         context, MaterialPageRoute(
         builder: (context) => Menu()
     ));
   } catch (e) {
       print (e);
   }
 }
}
```

**The above are the 16 dart files that built this application. The privacy issues related to the Google API key have been removed and replaced with "xxxxxxxxxxxxxxxxxxxxxxxxxxxx".

7. Bibliography

RSA 2021, Rules of the Road, available on
https://www.rsa.ie/Documents/RotR%20BOOK%20for%20web%202019.pdf

Nauto 2021, What is Driver Behavior, available on
https://www.nauto.com/glossary/what-is-driver-behavior

Elitech 2021, Flutter App advantages and disadvantages, available on
https://www.elitechsystems.com/start-with-flutter-basics-the-good-and-the-bad/

Dart 2021, Dart overview, available on
https://dart.dev/tutorials

Firebase 2021, Configure hosting behavior, available on
https://firebase.google.com/docs/hosting/full-config

Google 2021, Road api overview, available on
https://developers.google.com/maps/documentation/roads/overview

Google 2021, Snapto Roads, available on
https://developers.google.com/maps/documentation/roads/snap

Google 2021, Speed limits, available on
https://developers.google.com/maps/documentation/roads/speed-limits

Google 2021, Write your first flutter app, available on
https://codelabs.developers.google.com/codelabs/first-flutter-app-pt1#0

Souvik Biswas 2019, Implementing Google Sign In, available on
https://medium.com/flutter-community/flutter-implementing-google-sign-in-71888bca24ed

Google 2021, Adding Google Maps to a Flutter app, available on
https://codelabs.developers.google.com/codelabs/google-maps-in-flutter#0

Peter Martinez, How to use flutter UI Design Tool Easily, available on
https://mockitt.wondershare.com/ui-ux-design/flutter-ui-design.html

Flutter 2021, Building layouts, available on
https://docs.flutter.dev/development/ui/layout/tutorial

Mockflow 2021, video tutorials, available on
https://support.mockflow.com/category/92-video-tutorials

rnfirebase 2021, Firebase JSON config, available on
https://rnfirebase.io/app/json-config

Firebase 2022, add Firebase to your android project, available on
https://firebase.google.com/docs/android/setup

Flutter 2022, Firebase documentation, available on
https://docs.flutter.dev/development/data-and-backend/firebase

Pushwoosh 2022, Android Firebase configuration, available on
https://docs.pushwoosh.com/platform-docs/manage-projects/project-configuration/configure-android-platform

FlutterFire 2022, Cloud Firestore, available on
https://firebase.flutter.dev/docs/firestore/usage/

Vijay R, Read Data from firestore in flutter web, available on
https://medium.com/vijay-r/read-data-from-firestore-in-flutter-web-32bf998843d2

PaulHalliday 2019, How to get a User's Location, available on
https://www.digitalocean.com/community/tutorials/flutter-geolocator-plugin

Varun Kamani 2021, How to calculate total distance, available on
https://flutteragency.com/total-distance-from-latlng-list-in-flutter/

Rakhi 2021, Location in flutter
https://medium.flutterdevs.com/location-in-flutter-27ca6fa1126c

Rajat Palanka 2021, Flutter get current location address, available on
https://protocoderspoint.com/flutter-get-current-location-address-from-latitude-longitude/

INSTITUTE of TECHNOLOGY CARLOW
Institiúid Teicneolaíochta Cheatharlach

PLAGIARISM DECLARATION

- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) Chi Ieong Ng
Student Number(s): C00223421

Signature(s):

Date: 29/04/2022