CSE_431: Major Task
Carpool App

Shaimaa Mohamed 19P7484
Group 1 Sec 2

Contents

1.0 Introduction

"Carpool, the innovative rideshare application, aims to revolutionize the commuting experience for Ain Shams University's Faculty of Engineering Community. With a focus on enabling seamless rides to and from Abdu-Basha and Abasia square, Carpool caters exclusively to students, fostering a trusted and secure closed community environment. To ensure the reliability and safety of the service, users are required to sign in using their active @eng.asu.edu.eg accounts.

This bespoke version of Carpool is tailored to meet the unique needs of the university's engineering community. Operated by students, for students, it introduces a pioneering strategy in recruiting drivers and managing the service. Emphasizing simplicity and efficiency, Carpool's pilot project centers on two fixed destination points Gate 3 and Gate 4 with set departure times: 7:30 am from various locations and 5:30 pm from the Faculty of Engineering campus.

To streamline the service, reservations for the morning ride must be made before 10:00 pm the day before, while those for the afternoon ride require bookings by 1:00 pm on the same day.

2.0 App Specifications:

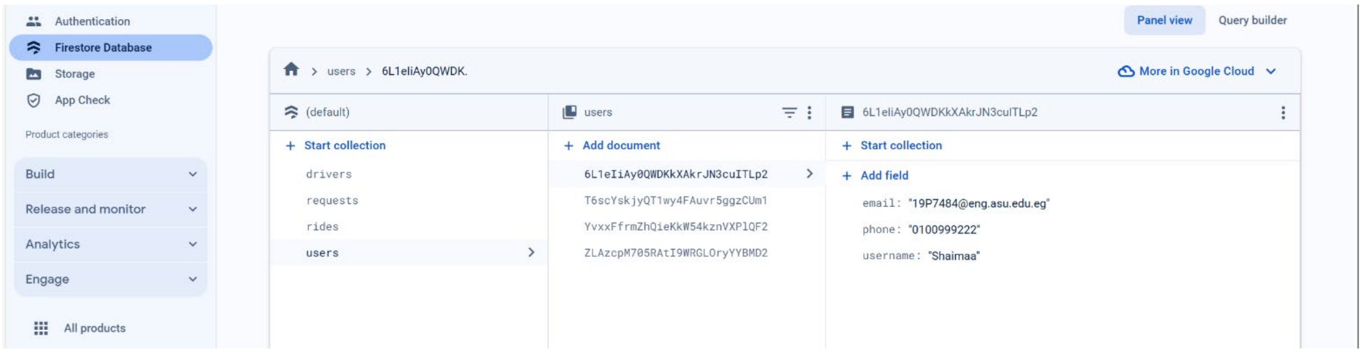
In this project, the primary goal is to develop a cutting-edge rideshare app that redefines the commuting experience for the Faculty of Engineering Community at Ain Shams University. The app entails a comprehensive set of features and functionalities, including:

- A secure login page with the option for new user registration through Firebase authentication for both user and driver.
- An extensive list of available routes comprising at different options offered by driver, displayed through a user-friendly List-View interface.
- A user-friendly cart page enabling riders to review their orders and make hassle-free payments also, equipped with tracking and status updates for user convenience.
- A web application for drivers to confirm orders and update status data, necessitating confirmation before 11:30 pm for the morning ride and 4:30 pm for the afternoon ride.
- Driver will be able to offer ride, see all the available requests and manage which requests will accept or reject then, the request status and ride status in both user app or driver app will be updated in real time using fire store service.
- Driver pages to see all available request and history of all the rides offered by him indicating ride status if it is completed, active, canceled and also tracking request status when the user was pending then converted to approved , rejected , expired requests if the driver exceeds the deadline in accepting this request.
- Implementation of Firebase real-time database for managing routes and order status, ensuring real-time updates and synchronization.
- Utilization of SQLite for storing and managing user profile data securely.

3.0 Database Structure

1. I am using **fire store database** as my main source in saving and retrieval data which supports updating and synchronizing data at real time so , it is very efficient in handling both user and driver apps.

My database scheme in fire store



users > 6L1eliAy0QWDK.

(default): drivers, requests, rides, users

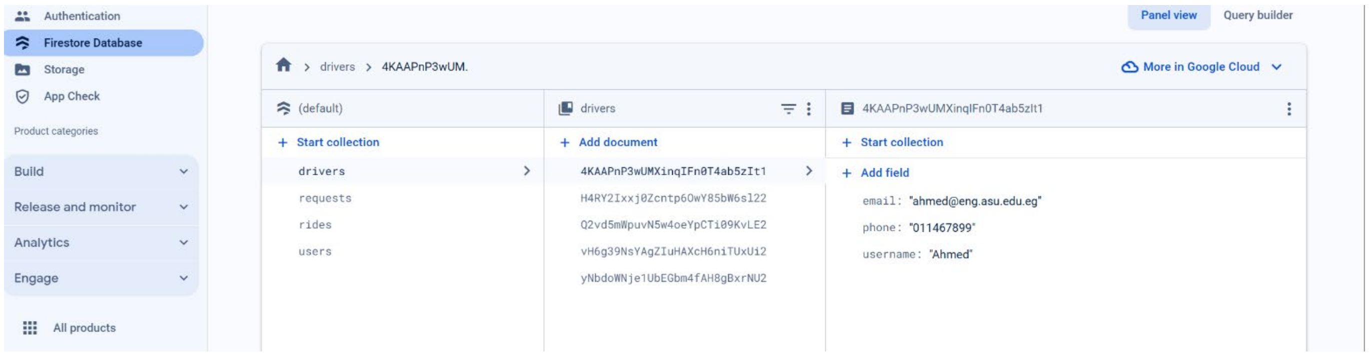
users:
6L1eIiAy0QWDKkXAkrJN3cuITLp2
T6scYskjyQT1wy4FAuvr5ggzCUm1
YvxxFfrmZhQieKkW54kznVXPlQF2
ZLAzcpM705RAtI9WRGLOryYYBMD2

6L1eliAy0QWDKkXAkrJN3culTLp2:
email: "19P7484@eng.asu.edu.eg"
phone: "0100999222"
username: "Shaimaa"

Making 4 essential collections

1. Users collection which is used to store user profile data like the username, email & phone

2. Driver personal data that I used to display in the profile and check on his existence once in the database to check If the email is not used before or not.



drivers > 4KAAPnP3wUM.

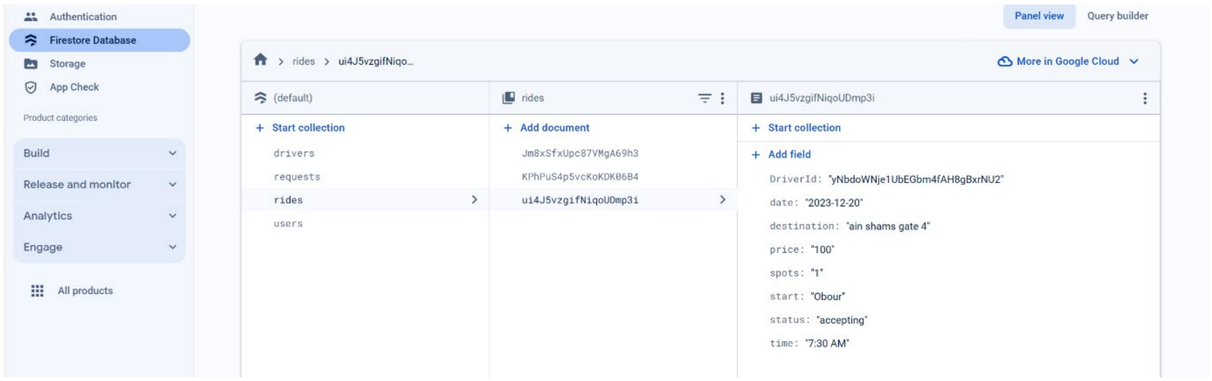
(default): drivers, requests, rides, users

drivers:
4KAAPnP3wUMXinqIFn0T4ab5zIt1
H4RY2Ixxj0Zcntp6OwY85bW6sl22
Q2vd5mWpuvN5w4oeYpCTi09KvLE2
vH6g39NsYAgZIuHAXcH6niTUxUi2
yNbdoWNje1UbEGbm4fAH8gBxrNU2

4KAAPnP3wUMXinqIFn0T4ab5zIt1:
email: "ahmed@eng.asu.edu.eg"
phone: "011467899"
username: "Ahmed"

3. **Rides collection** used for saving the ride information containing the driverId who offers this ride and all the ride data including start , destination , date , time , price , status of the ride , and spots . The one who writes in this collection is the driver while offering the ride.

   Status of the ride can take different values like active if the trip time equal the time now , canceled if the driver rejects the ride , completed when the request is approved and the time is after the trip time by delay time 2 hours and half as an indication that the ride is finished . The status of the ride is accepting means the driver is available and in service to take the ride. Other wise I make the ride status waiting in order to be check later.



rides > ui4J5vzgifNiqo...

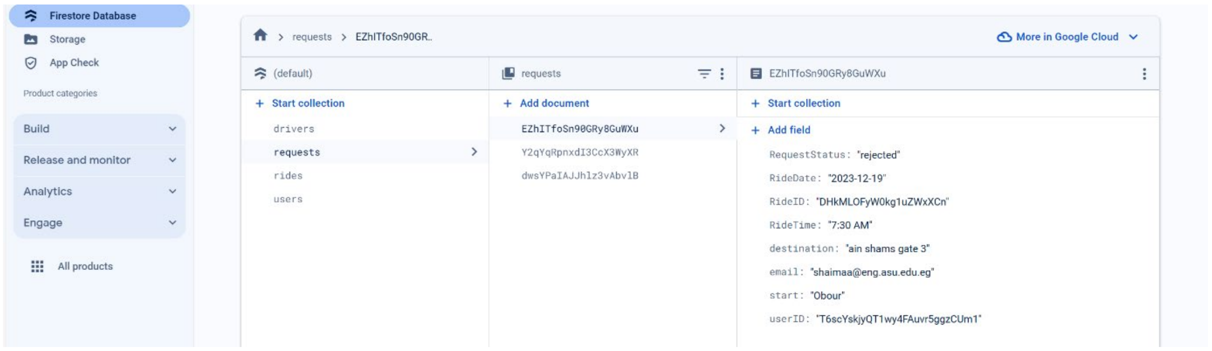
(default): drivers, requests, rides, users

rides: Jm8xSfxUpc87VMgA69h3, KPhPuS4p5vcKoKDK06B4, ui4J5vzgifNiqoUDmp3i

ui4J5vzgifNiqoUDmp3i:

```
DriverId: "yNbdoWNje1UbEGbm4fAH8gBxrNU2"
date: "2023-12-20"
destination: "ain shams gate 4"
price: "100"
spots: "1"
start: "Obour"
status: "accepting"
time: "7:30 AM"
```

4. **Requests Collection**



requests > EZhITfoSn90GR..

(default): drivers, requests, rides, users

requests: EZhITfoSn90GRy8GuWXu, Y2qYqRpnxdI3CcX3WyXR, dwsYPaIAJJhlz3vAbvlB

EZhITfoSn90GRy8GuWXu:

```
RequestStatus: "rejected"
RideDate: "2023-12-19"
RideID: "DHkMLOFyW0kg1uZWxXCn"
RideTime: "7:30 AM"
destination: "ain shams gate 3"
email: "shaimaa@eng.asu.edu.eg"
start: "Obour"
userID: "T6scYskjyQT1wy4FAuvr5ggzCUm1"
```

The requests collection is the collection where the rider request data will be saved like RideDate , RideId id for this selected ride , ride time ,destination and start locations and the user email & id who made this request also the Request status is saved. Request status by default when a rider requests the Request Status become pending then it can take possible values , approved when the driver accepts the ride within time limits , rejected if he exceeds the time limit or if he want to cancel it for any other reasons .

Local database handling for profile data using sqflite

My database handler code is in class called Databasev2

```dart
import 'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

class Databasev2 {
  Database? mydatabase;

  Future<Database?> checkdata() async {
    if (mydatabase == null) {
      mydatabase = await creating();
      return mydatabase;
    } else {
      return mydatabase;
    }
  }

  int Version = 2;
  creating() async {
    String databasepath = await getDatabasesPath();
    String mypath = join(databasepath, 'mynewdatafile2.db');
    Database mydb =
    await openDatabase(mypath, version: Version, onCreate: (db, version) {
      db.execute('''CREATE TABLE IF NOT EXISTS 'FILE1'(
      'ID' INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
      'UserName' TEXT NOT NULL,
      'Email' TEXT NOT NULL,
      'Phone' TEXT NOT NULL)''');
    });
    return mydb;
  }

  isexist() async {
    String databasepath = await getDatabasesPath();
    String mypath = join(databasepath, 'mynewdatafile2.db');
    await databaseExists(mypath) ? print("it exists") : print("not exist");
  }

  reading(sql) async {
    Database? somevar = await checkdata();
    var myesponse = somevar!.rawQuery(sql);
    return myesponse;
  }


write(sql) async {
    Database? somevar = await checkdata();
    var myesponse = somevar!.rawInsert(sql);
    return myesponse;
  }
update(sql) async {
    print("updating");
    Database? somevar = await checkdata();
    var myesponse = somevar!.rawUpdate(sql);
    print("done update");
    return myesponse;
  }

  delete(sql) async {
    Database? somevar = await checkdata();
    var myesponse = somevar!.rawDelete(sql);
    return myesponse;
  }

}
```

This Class includes main Database creating and SQL operations:

Creating Tables: Upon creating the database, it executes an SQL command to create a table named 'FILE1' if it doesn't exist. The table has columns for ID, UserName, Email, and Phone.

Checking Database Existence: isexist() checks if the database file exists at the specified path.

CRUD Operations: reading(), write(), update(), and delete() methods perform raw SQL operations on the database using the sqflite package.

Methods:

reading(sql): Executes a read operation on the database using the provided SQL query.

write(sql): Executes an insert operation into the database using the provided SQL query.

update(sql): Executes an update operation in the database using the provided SQL query.

delete(sql): Executes a delete operation in the database using the provided SQL query.

In my Profile UI page for each of user or driver

My handler code to choose where to get information at offline connection

```
final Databasev2 db = Databasev2();
Firestore_services _firestore = Firestore_services();

Future<bool> check() async {
  var connectivityResult = await (Connectivity().checkConnectivity());
  if (connectivityResult == ConnectivityResult.mobile) {
    return true;
  } else if (connectivityResult == ConnectivityResult.wifi) {
    return true;
  }
  return false;
}

List <Map> mylist = [];
Future<void> _readingData() async {
  List<Map> response = await db.reading('''SELECT * FROM 'FILE1' ''');
  mylist = [];
  mylist.addAll(response);
  setState(() {

  });
}

Future <void >_writingData() async {
  Map myProfile = _firestore.fetchUserProfile() as Map;
  await db.write(
      '''INSERT INTO 'FILE1' ('UserName', 'Email', 'Phone') VALUES
           ('${myProfile['username']}',
            '${myProfile['email']}',
            '${myProfile['phone']}')''');
}

Future<void> _initializeProfile() async {
  await _writingData(); // Wait for writing data to complete
  final isConnected = await check(); // Check connectivity

  if (isConnected) {
    // Fetch user profile from Firestore
    _firestore.fetchUserProfile();
  } else {
    setState(() {
      _readingData(); // Read user profile from the local SQLite database
      // db.checkdata();

    });

  }
}
```

```
@override
void initState() {
  super.initState();
  _initializeProfile();

}


Future<Map<String, dynamic>> fetchUserProfile() async {
  final uid = FirebaseAuth.instance.currentUser?.uid;

  if (uid != null) {
    var userRef = FirebaseFirestore.instance.collection('users').doc(uid);
    var documentSnapshot = await userRef.get();
    return documentSnapshot.data() ?? {};
  }
  return {};

}
```

Illustrations of syncing information between firestore and sqflite

Code demonstrates a profile page in a Flutter application, aiming to synchronize user profile data between a local SQLite database and Fire store. Let's break down the key parts of the code and illustrate how it manages the sync logic:

1. Database Initialization:

Local Database: Databasev2 appears to be the class handling the SQLite database.

Firestore: _firestore is an instance of Firestore_services, presumably managing Firestore interactions.

2. Sync Logic:

Reading Data: _readingData() reads user profile data from the local SQLite database using the db.reading() method.

Writing Data: _writingData() writes user profile data to the local SQLite database using the db.write() method. It uses data fetched from Firestore through _firestore.fetchUserProfile().

3. Handling Connectivity:

check() checks the device's connectivity status using the connectivity package.

4. Initialization:

_initializeProfile() initializes the user profile.

Writes data to the local SQLite database.

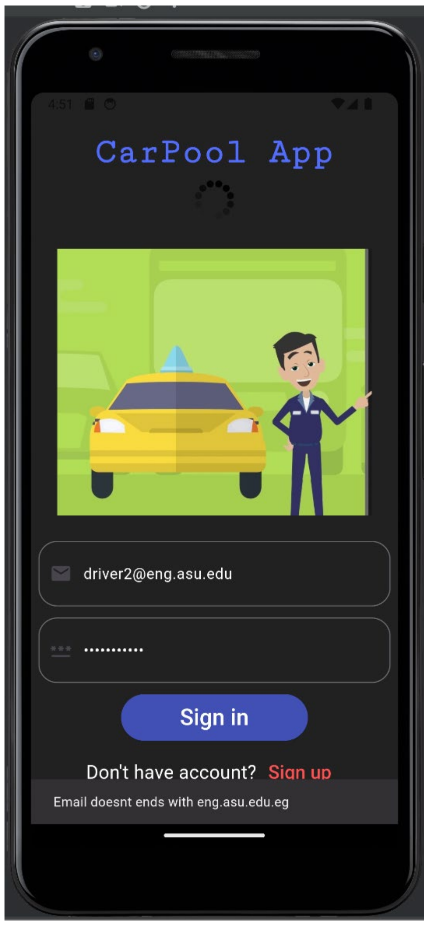
Checks connectivity and fetches user profile data from Firestore if connected, otherwise reads data from the local SQLite database.

So, The app ensures that user profile data is available offline in the SQLite database while also being synchronized with Firestore when the device is connected to the internet. This setup maintains data consistency and provides a smooth user experience across different network conditions.

4.0 UI design

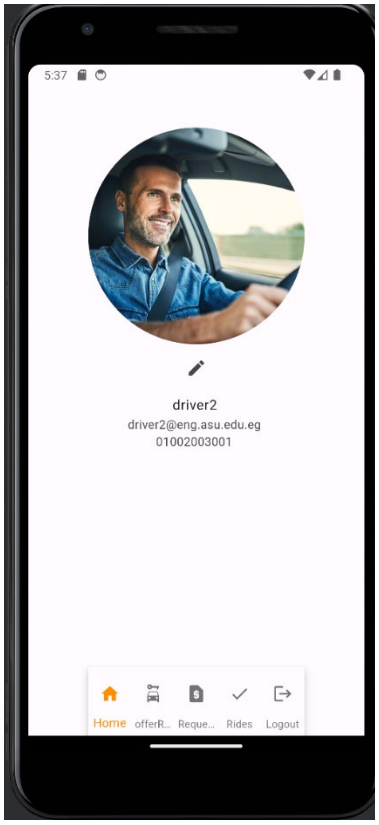
4.1 Driver App UI

1. Sign in



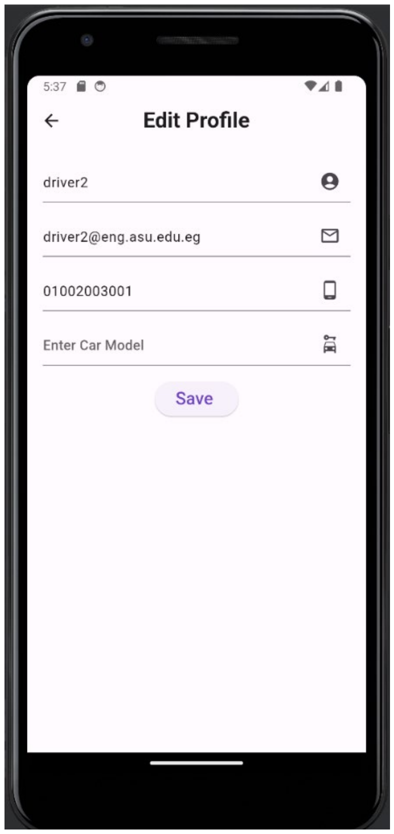
Email doesnt ends with eng.asu.edu.eg

2. Home Page


5:37

driver2
driver2@eng.asu.edu.eg
01002003001

3. Edit Profile Page


5:37

Edit Profile

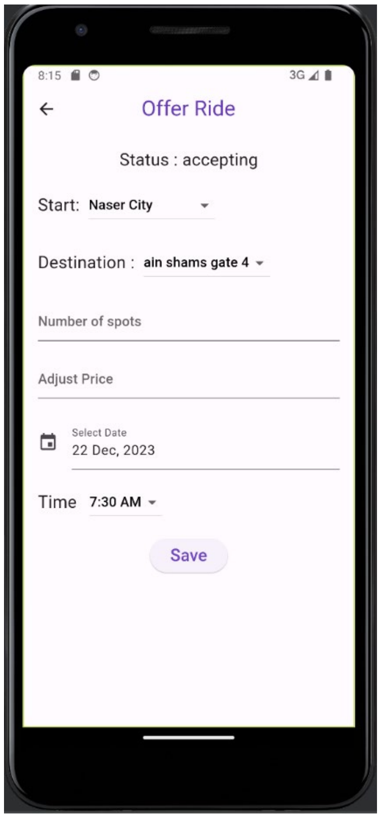
driver2

driver2@eng.asu.edu.eg

01002003001

Enter Car Model

4. Offering routes after pressing offer ride Icon.


8:15 3G

Offer Ride

Status : accepting

Start: Naser City

Destination : ain shams gate 4

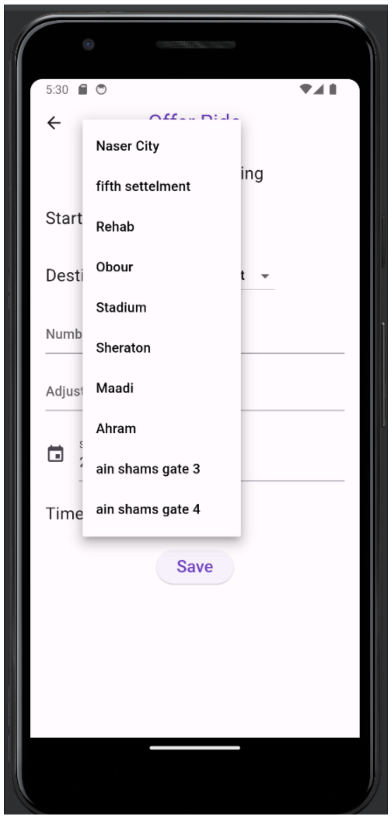
Number of spots

Adjust Price

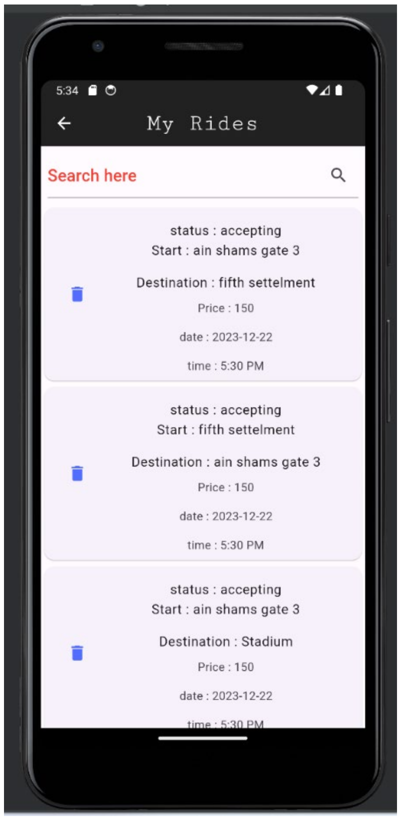
Select Date
22 Dec, 2023

Time 7:30 AM

Save



5. Click on Rides page stores all the offered rides and tracking its status , searching bar is used to search by ride status


5:34

My Rides

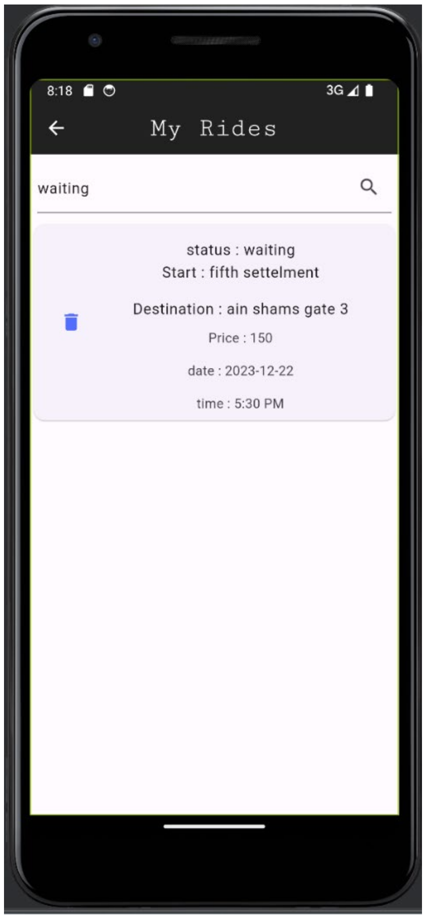
Search here

status : accepting
Start : ain shams gate 3
Destination : fifth settelment
Price : 150
date : 2023-12-22
time : 5:30 PM

status : accepting
Start : fifth settelment
Destination : ain shams gate 3
Price : 150
date : 2023-12-22
time : 5:30 PM

status : accepting
Start : ain shams gate 3
Destination : Stadium
Price : 150
date : 2023-12-22
time : 5:30 PM

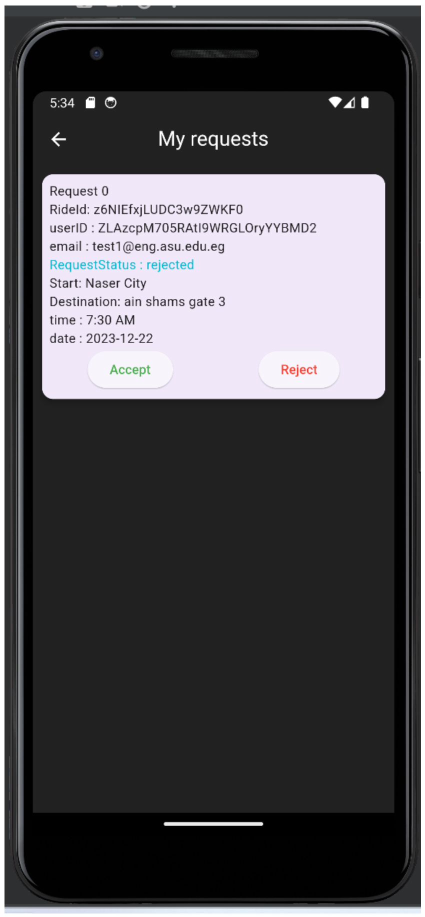
6. Activating the search bar to search by the ride status to facilitate seeing the history of the driver rides .



waiting

status : waiting
Start : fifth settelment
Destination : ain shams gate 3
Price : 150
date : 2023-12-22
time : 5:30 PM

7. Press on requests to see all available requests

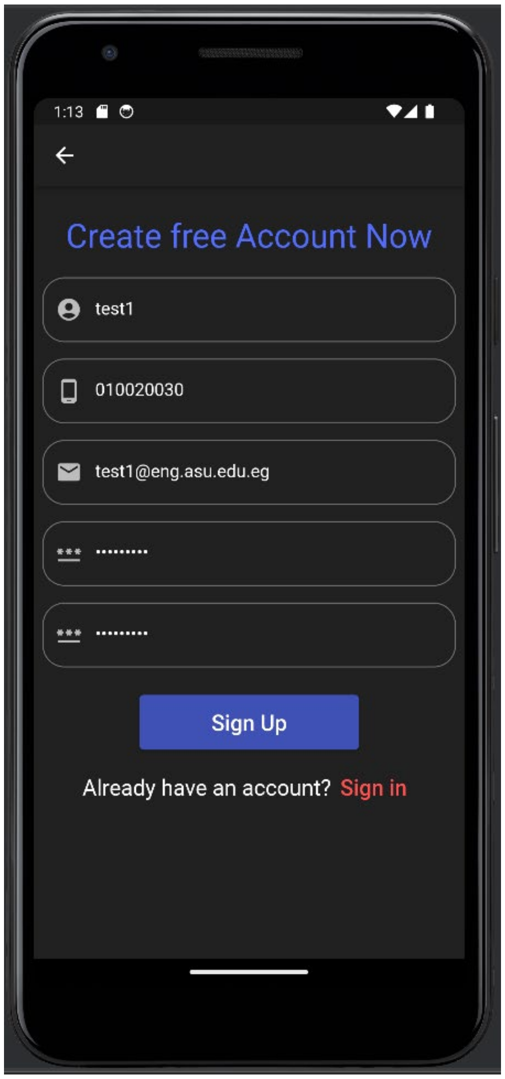

5:34

My requests

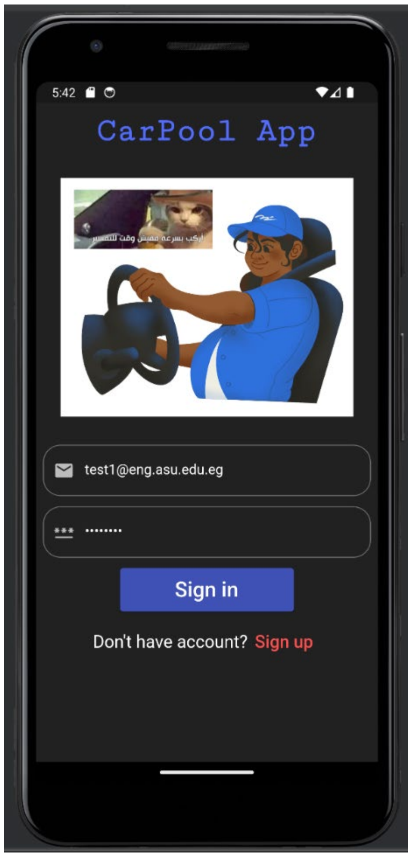
Request 0
RideId: z6NIEfxjLUDC3w9ZWKF0
userID : ZLAzcpM705RAtI9WRGLOryYYBMD2
email : test1@eng.asu.edu.eg
RequestStatus : rejected
Start: Naser City
Destination: ain shams gate 3
time : 7:30 AM
date : 2023-12-22

4.2 Rider UI design

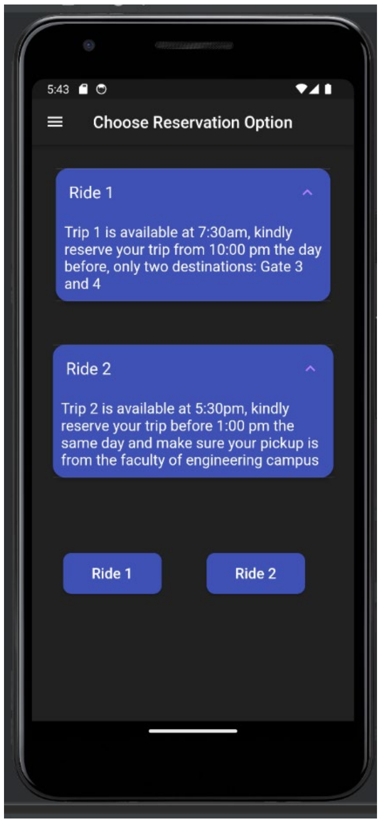
1. Sign Up Page



2. Sign In page



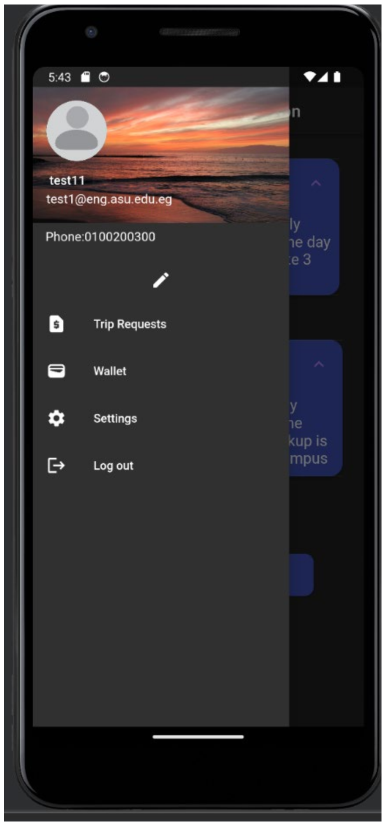
3. Home Page



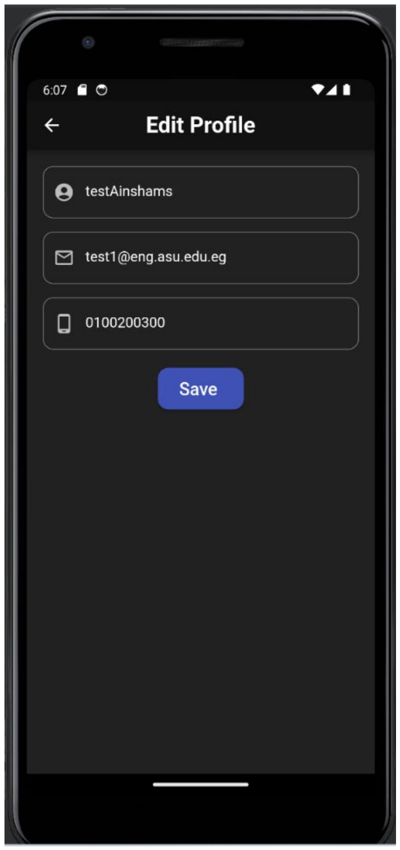
Trip 1 is available at 7:30am, kindly reserve your trip from 10:00 pm the day before, only two destinations: Gate 3 and 4

Trip 2 is available at 5:30pm, kindly reserve your trip before 1:00 pm the same day and make sure your pickup is from the faculty of engineering campus

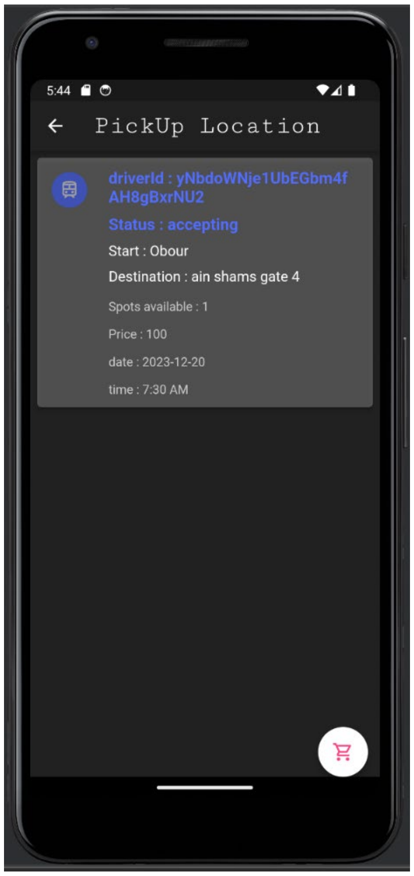
4. **Pressing on the drawer to see profile**



5. Click on edit button



6. Pressing on Ride will show all the available rides at 7:30 AM



driverId : yNbdoWNje1UbEGbm4fAH8gBxrNU2

Status : accepting

Start : Obour

Destination : ain shams gate 4

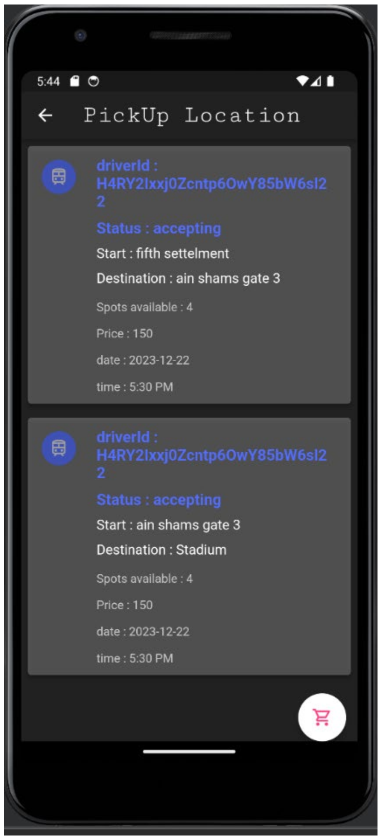
Spots available : 1

Price : 100

date : 2023-12-20

time : 7:30 AM

7. Pressing on Ride2 option


5:44

PickUp Location

driverId : H4RY2lxxj0Zcntp6OwY85bW6sl2 2

Status : accepting

Start : fifth settelment

Destination : ain shams gate 3

Spots available : 4

Price : 150

date : 2023-12-22

time : 5:30 PM

driverId : H4RY2lxxj0Zcntp6OwY85bW6sl2 2

Status : accepting

Start : ain shams gate 3

Destination : Stadium

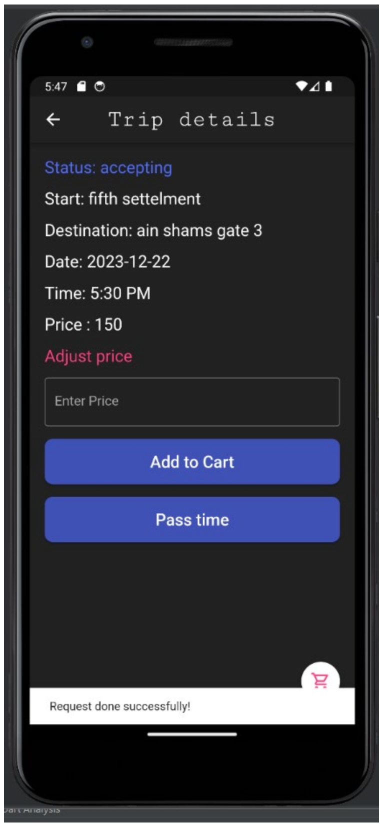
Spots available : 4

Price : 150

date : 2023-12-22

time : 5:30 PM

8. Tap on any card to show its details and have reservation option, pressed add to cart.
   It checks if the time is at the same date of the trip and check the day. so, if it is before deadline 1:00 pm at same day rider can request.
   this trip.


5:47

Trip details

Status: accepting

Start: fifth settelment

Destination: ain shams gate 3

Date: 2023-12-22

Time: 5:30 PM

Price : 150

Adjust price

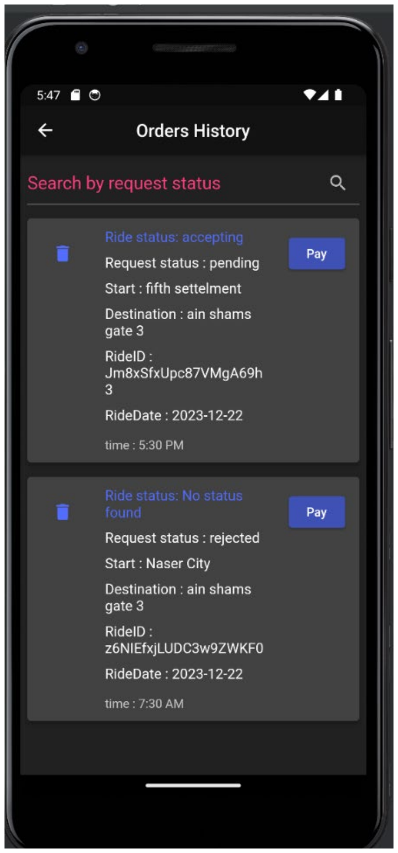
Enter Price

Add to Cart

Pass time

Request done successfully!

9. Press on the cart option, our request is saved in fire store and displayed on this screen, ready to make payment or delete the request from icon delete.
   Here we observe that the request is pending waiting for the driver connection.


5:47

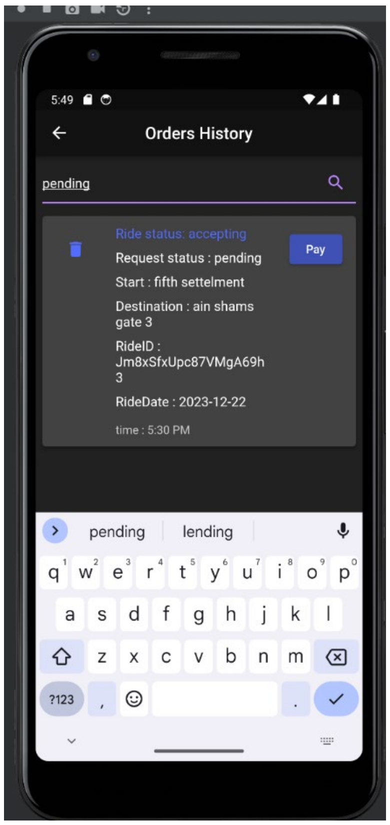
Orders History

Search by request status

Ride status: accepting
Request status : pending
Start : fifth settelment
Destination : ain shams gate 3
RideID : Jm8xSfxUpc87VMgA69h3
RideDate : 2023-12-22
time : 5:30 PM
Pay

Ride status: No status found
Request status : rejected
Start : Naser City
Destination : ain shams gate 3
RideID : z6NIEfxjLUDC3w9ZWKF0
RideDate : 2023-12-22
time : 7:30 AM
Pay

10. Searching feature by request status

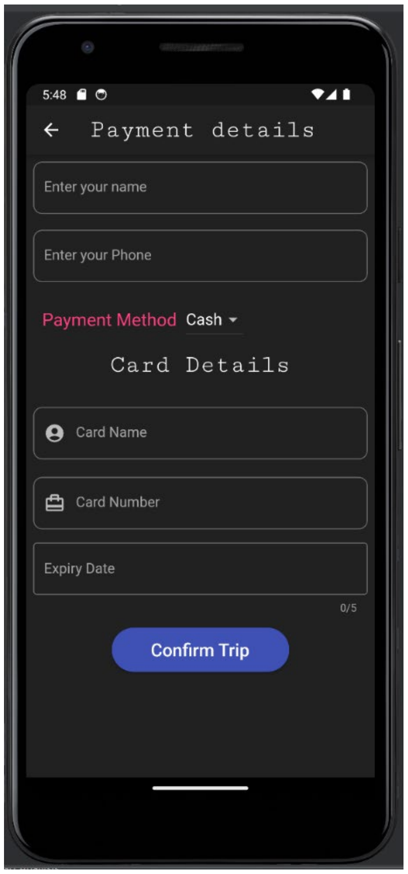

5:49

Orders History

pending

Ride status: accepting
Request status : pending
Start : fifth settelment
Destination : ain shams gate 3
RideID : Jm8xSfxUpc87VMgA69h3
RideDate : 2023-12-22
time : 5:30 PM
Pay

11. Payment page


5:48

Payment details

Enter your name

Enter your Phone

Payment Method Cash

Card Details

Card Name

Card Number

Expiry Date

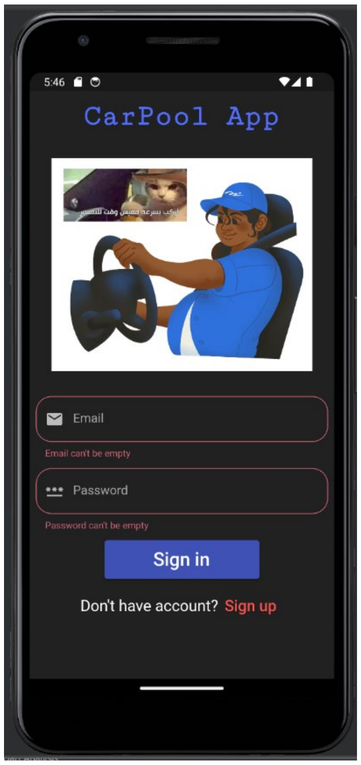
0/5

Confirm Trip

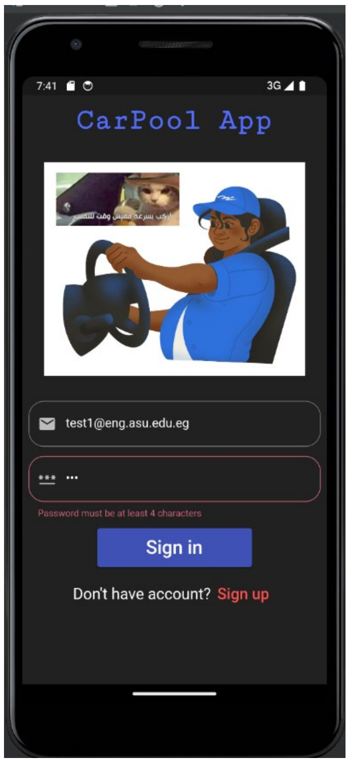
5.0 TestCases

5.1 Login & Register handling

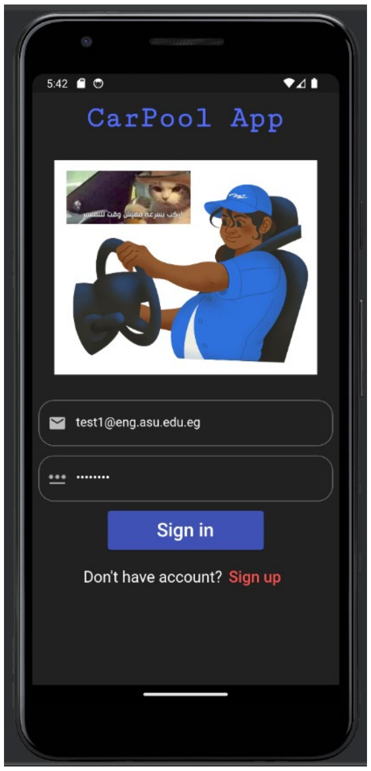
1. Validate that email and password can’t be empty



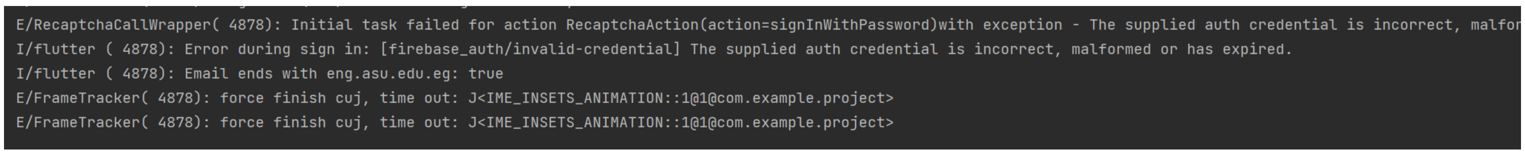
2. Handle matching if the password is weak


7:41
3G
CarPool App
test1@eng.asu.edu.eg
•••
Password must be at least 4 characters
Sign in
Don't have account? Sign up

3. Handle incorrect credentials by firebase

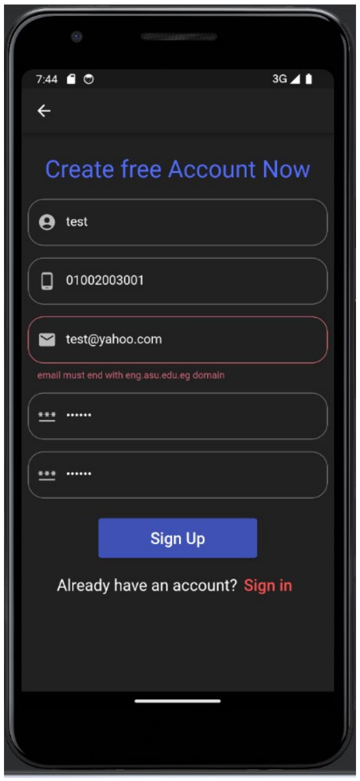


Output is



```
E/RecaptchaCallWrapper( 4878): Initial task failed for action RecaptchaAction(action=signInWithPassword)with exception - The supplied auth credential is incorrect, malfor
I/flutter ( 4878): Error during sign in: [firebase_auth/invalid-credential] The supplied auth credential is incorrect, malformed or has expired.
I/flutter ( 4878): Email ends with eng.asu.edu.eg: true
E/FrameTracker( 4878): force finish cuj, time out: J<IME_INSETS_ANIMATION::1@1@com.example.project>
E/FrameTracker( 4878): force finish cuj, time out: J<IME_INSETS_ANIMATION::1@1@com.example.project>
```

4. Sign up handling , the email must be in the domain eng.asu.edu.edu.eg



test

01002003001

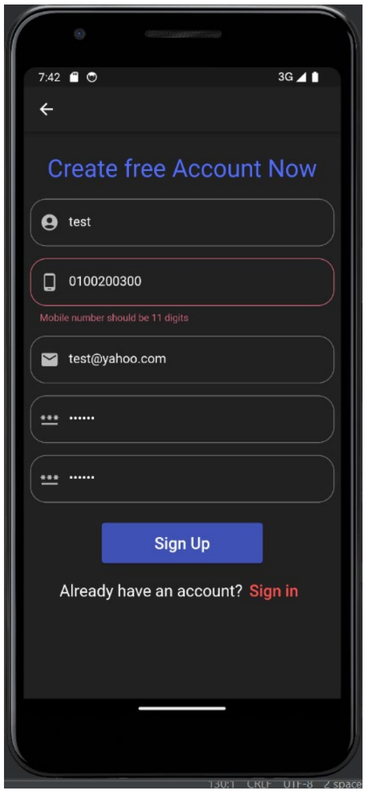
test@yahoo.com

email must end with eng.asu.edu.eg domain

••••••

••••••

5. handle if the phone consists of 11 digits.



test

0100200300

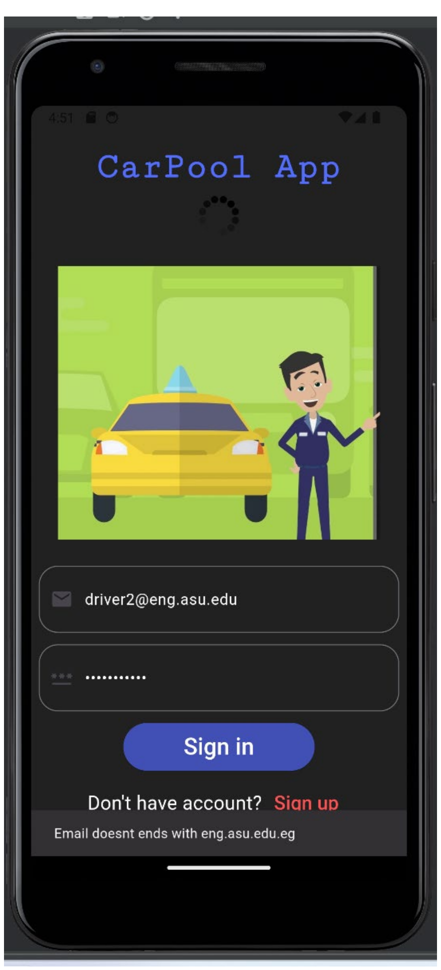
Mobile number should be 11 digits

test@yahoo.com

••••••

••••••

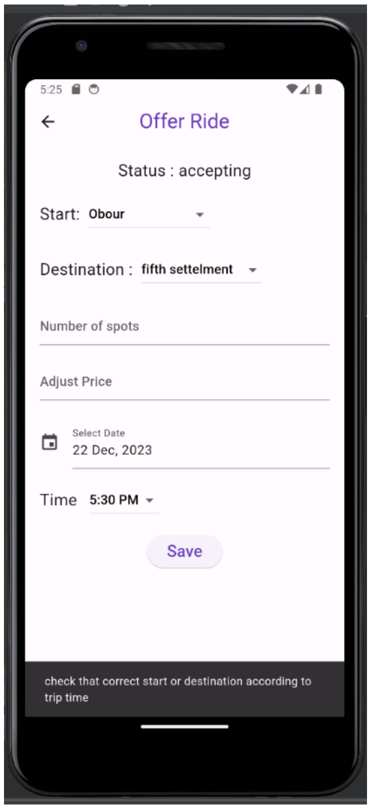
6. **Check also Signing in as a driver**
   Check that the email ends with the correct domain '@eng.asu.edu.eg'



Email doesnt ends with eng.asu.edu.eg

7. **Handle offering the rides on the Driver Application**

   I check that if the selected time is 5:30 so, the trip start must be ain shams gate 3 or 4 , moreover I check that of the start and destination is ain shams gate 3 or 4.



Offer Ride

Status : accepting

Start: Obour

Destination : fifth settelment

Number of spots

Adjust Price

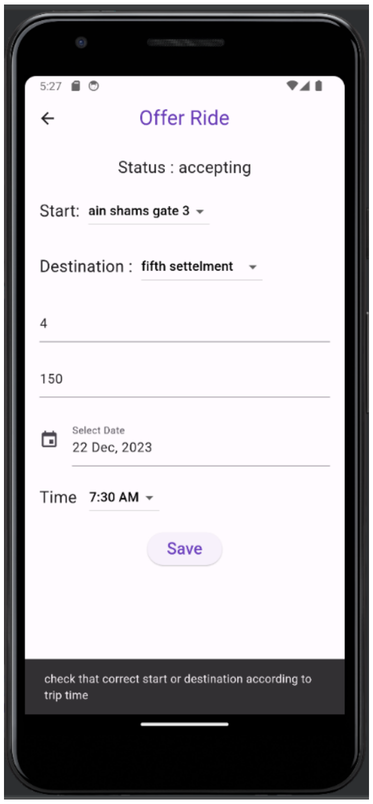
Select Date
22 Dec, 2023

Time 5:30 PM

Save

check that correct start or destination according to trip time

8. Check the correct destination if the time 7:30 AM so, the trip must go to ain shams gate 3 or 4.



Offer Ride

Status : accepting

Start: ain shams gate 3

Destination : fifth settelment

4

150

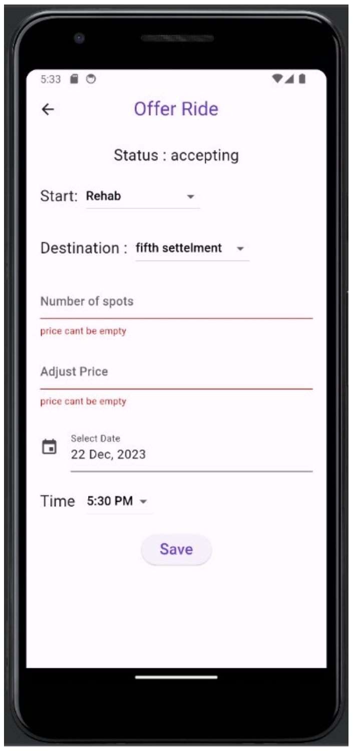
Select Date
22 Dec, 2023

Time 7:30 AM

Save

check that correct start or destination according to trip time

9. Check that the price is not empty field


5:33

Offer Ride

Status : accepting

Start: Rehab

Destination : fifth settelment

Number of spots

price cant be empty

Adjust Price

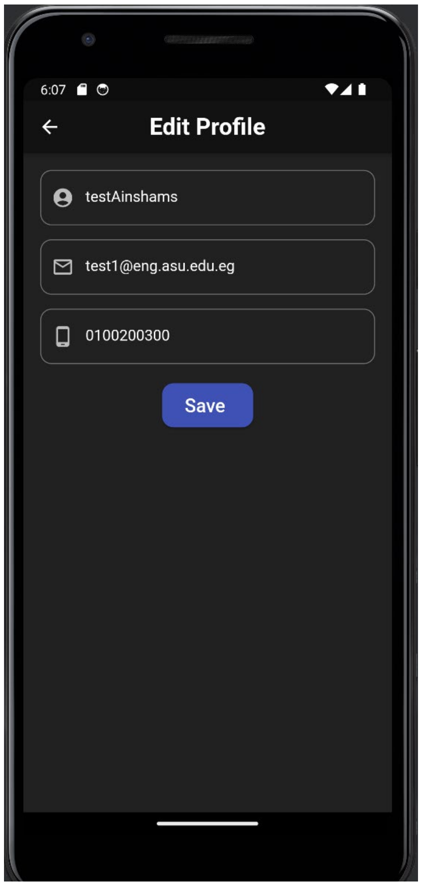
price cant be empty

Select Date
22 Dec, 2023

Time 5:30 PM

Save

10. Check that the data of profile is updated successfully.


6:07

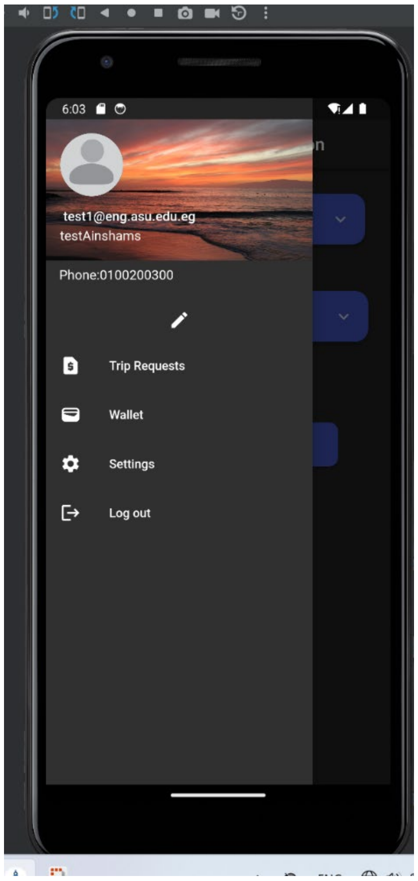
Edit Profile

testAinshams

test1@eng.asu.edu.eg

0100200300

Save


6:03

test1@eng.asu.edu.eg
testAinshams

Phone:0100200300

Trip Requests

Wallet

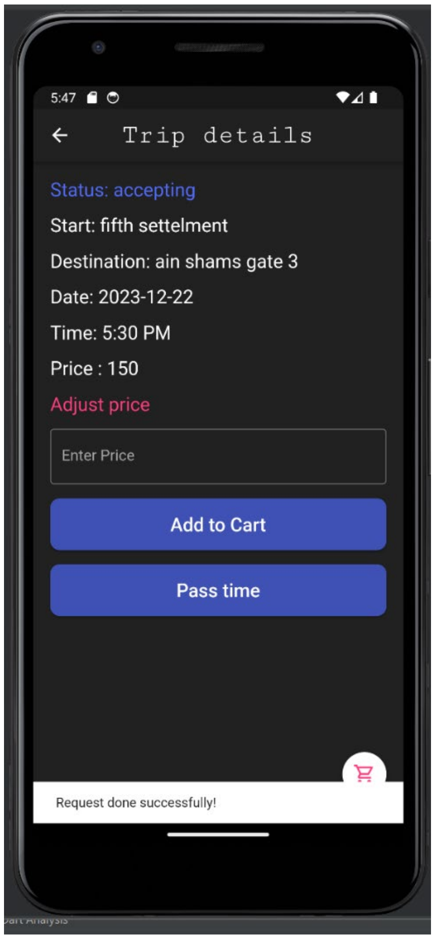
Settings

Log out

11. Check the reservation time for user based on conditions

1. If I am within the time constraints for the trip before 4:30 pm in the same day , If I am talking about 5:30 PM trip so, the ride must be added successfully , if not it shall give snack bar with message that the reservation closed.

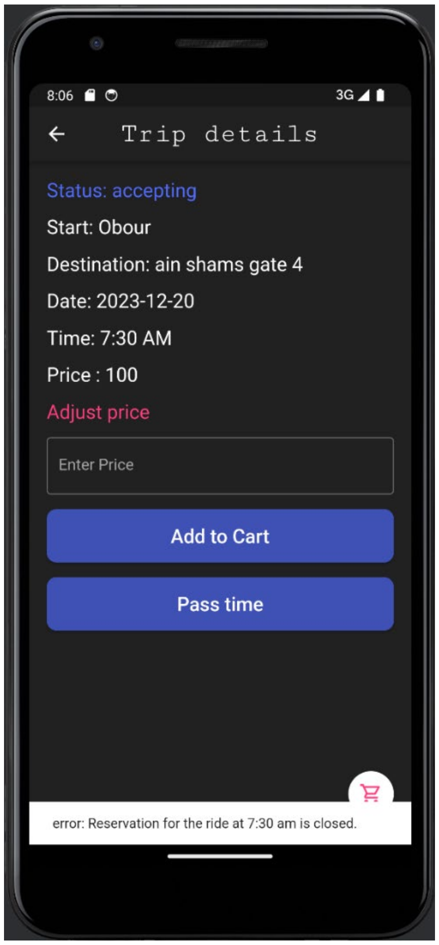
   The time was 3:00 am when I took the screen shot so, it is done successfully.



Status: accepting
Start: fifth settelment
Destination: ain shams gate 3
Date: 2023-12-22
Time: 5:30 PM
Price : 150
Adjust price

Request done successfully!

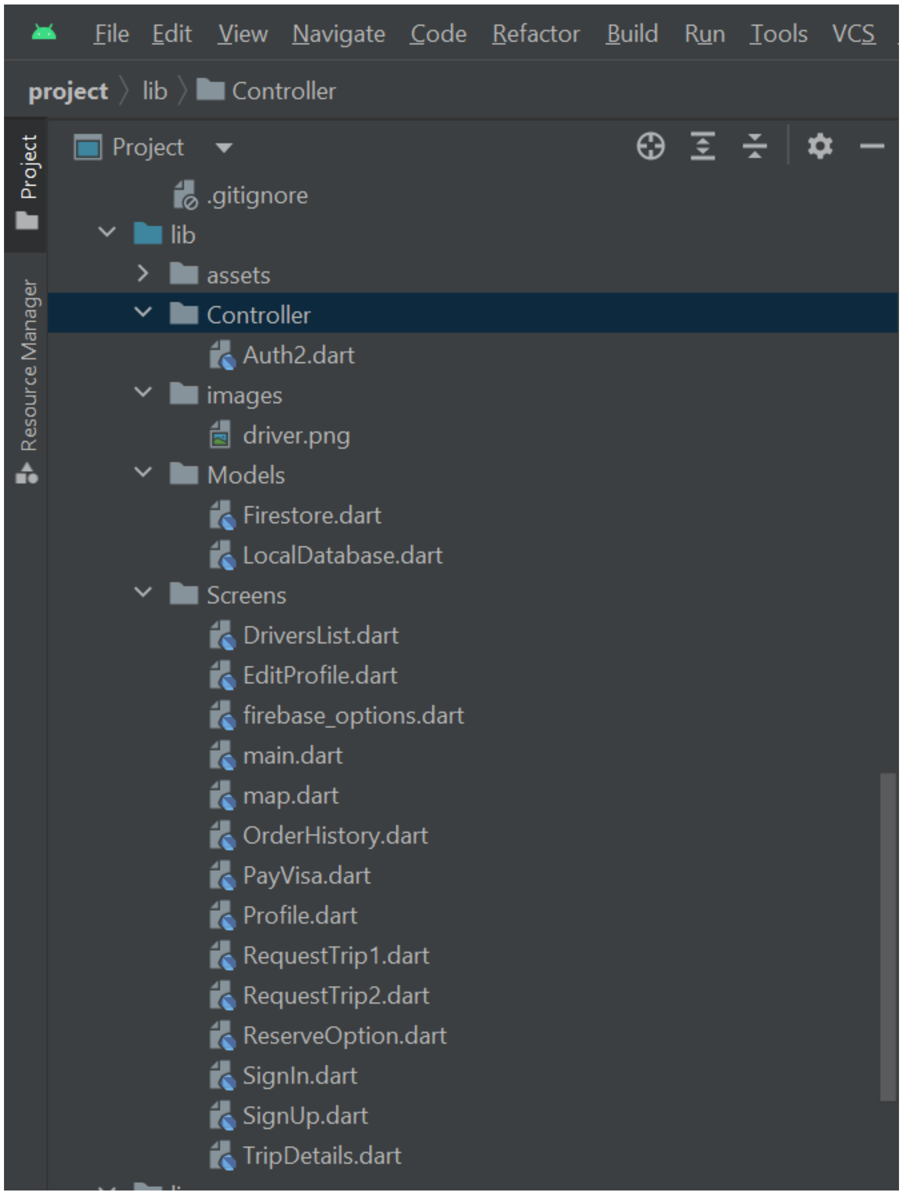
12. For this ride as the date of the trip is 20-12-2023 and I am trying to reserve at 8:00 Pm at 22-12-2023, it will show me an error message as the reservation closed.



Status: accepting
Start: Obour
Destination: ain shams gate 4
Date: 2023-12-20
Time: 7:30 AM
Price : 100

error: Reservation for the ride at 7:30 am is closed.

5.0 MCV Structure


File Edit View Navigate Code Refactor Build Run Tools VCS
project 〉 lib 〉 Controller
Project
Resource Manager

```
.gitignore
lib
  assets
  Controller
    Auth2.dart
  images
    driver.png
  Models
    Firestore.dart
    LocalDatabase.dart
  Screens
    DriversList.dart
    EditProfile.dart
    firebase_options.dart
    main.dart
    map.dart
    OrderHistory.dart
    PayVisa.dart
    Profile.dart
    RequestTrip1.dart
    RequestTrip2.dart
    ReserveOption.dart
    SignIn.dart
    SignUp.dart
    TripDetails.dart
```

My Models directory contains all the dealing with databases either firestore database or sqflite local database , it contains all the logic like saving in firestore , fetching data from collections and even updating statues.

The controller directory consists of the authentication functions like signIn with firebase and contain creating new accounts , sign out.

The views directory contains all my UI screens addressing the user and the same structure in the driver app.

Importance of MCV structure

1. Modularity: MVC promotes modularity by breaking an application into three interconnected components: Model, View, and Controller. Each component has a specific role, making the codebase easier to manage, maintain, and scale.
2. Separation of Concerns: MVC enforces a clear separation of concerns, where each component handles specific aspects of the application:
   - Model: Represents the data and business logic of the application.
   - View: Deals with the presentation layer, displaying the data to users.
   - Controller: Acts as an intermediary between Model and View, handling user input, processing requests, and updating the Model accordingly.

6.0 Database code using fire store

Model (Data and Business Logic):

Firestore_services Class Code :

```
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:intl/intl.dart';

String status='';
class Firestore_services{


  Future<bool> checkUserExistsInFirestore(String email, String password)
async {
    // Ensure your Firestore query is correctly checking for user existence
    try {
      QuerySnapshot query = await FirebaseFirestore.instance
          .collection('users')
          .where('email', isEqualTo: email)
          .get();

      return query.docs
          .isNotEmpty; // Returns true if there are documents matching the
email
    } catch (e) {
      // Handle any potential errors during the Firestore query
      print('Error checking user existence: $e');
      return false;
    }
  }
```

```
//used to ensure that the user request the ride only once at a time
  Future<bool> checkExistingRequest(String rideId,String uid) async {
    try {
      // Check if a document exists with the same rideId and userID
      var rideRef = FirebaseFirestore.instance.collection('requests')
          .where('RideID', isEqualTo: rideId)
          .where('userID', isEqualTo: uid);

      var rideQuery = await rideRef.get();
      return rideQuery.docs.isNotEmpty;
    } catch (e) {
      print('Error checking existing request: $e');
      return false;
    }
  }

  // function used to save user data in collection users
  Future<void> saveUserData(String Uid, String email, String username,
      String phone) async {
    try {
      await FirebaseFirestore.instance.collection('users').doc(Uid).set({
        'username': username,
        'email': email,
        'phone': phone,

      });
    } catch (e) {
      print('Error saving user data: $e');
      // Handle data saving errors here.
    }
  }
  // function used to save request data in collection request

  Future <void> SaveRequestData (String ? uid, String ? email ,String ? start
,String ? destination
      ,String ?rideId , String ? RideDate,String ? RideTime,String
?RequestStatus , String ? DriverID) async{
    await FirebaseFirestore.instance.collection('requests').add({
      'userID': uid,
      'email': email,
      'start': start,
      'destination':destination,
      'RideID': rideId,
      'RideDate': RideDate,
      'RideTime':RideTime,
      'RequestStatus':RequestStatus,
      'DriverId':DriverID,
    });

  }
```

```
// fetch from requests collection and perform searching by status
  Future<List> fetchRequestdata(String status) async {
    final uid = FirebaseAuth.instance.currentUser?.uid;

    if (uid != null) {
      var userRef = FirebaseFirestore.instance.collection('requests');
      QuerySnapshot querySnapshot;

      if (status.isEmpty) {
        querySnapshot = await userRef.where('userID', isEqualTo: uid).get();
      } else {
        querySnapshot = await userRef.where('userID', isEqualTo: uid)
            .where('RequestStatus', isEqualTo: status)
            .get();
      }

      List mydoc = querySnapshot.docs;
      return mydoc;
    }
    return [];
  }
  // function used to fetchh rideStatus data in collection rides

  Future<String?> fetchRideStatus(String rideId) async {
    try {

      DocumentSnapshot rideSnapshot = await FirebaseFirestore.instance
          .collection('rides').doc(rideId).get();

      if (rideSnapshot.exists) {
        Map<String, dynamic> rideData = rideSnapshot.data() as Map<String,
dynamic>;
        String? rideStatus = rideData['status'] as String?;
        return rideStatus;
      } else {
        return null; // Ride not found or data is empty
      }
    }
    catch (e) {
      print('Error fetching ride status: $e');
      return null; // Error occurred during fetch
    }
  }
  Future<Map<String, dynamic>> fetchUserProfile() async {
    final uid = FirebaseAuth.instance.currentUser?.uid;

    if (uid != null) {
      var userRef = FirebaseFirestore.instance.collection('users').doc(uid);
      var documentSnapshot = await userRef.get();
      return documentSnapshot.data() ?? {};
    }
    return {};

  }
```

```
Future<List<QueryDocumentSnapshot<Object?>>> fetchOption1() async {
    final QuerySnapshot ridesSnapshot = await FirebaseFirestore.instance
        .collection('rides')
        .where('time', isEqualTo: '7:30 AM') // Replace 'time' with your time
field
        .get();

    return ridesSnapshot.docs;
  }
  Future<List<QueryDocumentSnapshot<Object?>>> fetchOption2() async {
    final QuerySnapshot ridesSnapshot = await FirebaseFirestore.instance
        .collection('rides')
        .where('time', isEqualTo: '5:30 PM') // Replace 'time' with your time
field
        .get();

    return ridesSnapshot.docs;
  }


//update the ride status
  Future<void> updateRideStatus(String ? rideId, String newStatus) async {
    try {
      print('Updating ride status for ride ID: $rideId'); // Debugging print
      await
FirebaseFirestore.instance.collection('rides').doc(rideId).update({'status':
newStatus});
      print('Ride status updated successfully!');

    }
    catch (e) {
      print('Error updating ride status: $e');
      // Handle the error as needed
    }
  }
  // var userRef1 = FirebaseFirestore.instance.collection('requests');

  Future<void> updateRideOnCondition() async {
    try {
      // Fetch data that you need to check conditions against
      List requests = await fetchRequestdata(status);

      if (requests.isNotEmpty) {
        for (int index = 0; index < requests.length; index++) {
          try {
            String dateTime = requests[index].data()['RideDate'];
            DateTime date =DateTime.parse(dateTime);
            print(date.year);
            String rideTime = requests[index].data()['RideTime']; // Assuming
'7:30 AM' is stored in rideTime
            //Trim leading/trailing spaces
            rideTime = rideTime.trim();
            // Replace non-breaking space with regular space if needed
            rideTime = rideTime.replaceAll('\u00A0', ' ');
            DateFormat format = DateFormat('h:mm a');
            DateTime time = format.parse(rideTime);
            print(time.minute);
            DateTime currentTime = DateTime.now();
            DateTime delayTime = DateTime(date.year, date.month, date.day,
time.hour + 2, time.minute +30);

            if (requests[index].data()['RequestStatus'] == 'approved' &&
currentTime==time) {
              await updateRideStatus(requests[index].data()['RideID'],
'active');
            } else if (requests[index].data()['RequestStatus'] == 'rejected'
) {

              await updateRideStatus(requests[index].data()['RideID'],
'canceled');
            } else if (requests[index].data()['RequestStatus'] == 'approved'
&& currentTime.isAfter(delayTime)) {

              await updateRideStatus(requests[index].data()['RideID'],
'completed');
            }
            else if (requests[index].data()['RequestStatus'] == 'expired' ) {

              await updateRideStatus(requests[index].data()['RideID'],
'canceled');
            }
            else if (requests[index].data()['RequestStatus'] == 'full trip' )
{

              await updateRideStatus(requests[index].data()['RideID'],
'Full');
            }

            else {
              await updateRideStatus(requests[index].data()['RideID'],
'waiting');
            }
          } catch (e) {
            print('Error in processing request
${requests[index].data()['RideID']}: $e');
          }
        }
      }
    } catch (e) {
      print('Error updating ride statuses: $e');
    }


  }

  DeleteData(CollectionReference userRef,String id){
    userRef.doc(id).delete();
  }
}
```

Firestore_class description

1. **checkUserExistsInFirestore:**

   Checks if a user exists in Firestore based on email and password and it handles the authentication-related data checks in Firestore.

2. **checkExistingRequest:**

   Verifies if a request for a ride already exists based on rideID and userID and manages checks for existing ride requests in Firestore.

3. **saveUserData:**

   Saves user data (username, email, phone) into the Firestore 'users' collection.

   Model Responsibility: Handles the storage of user-related data in Firestore.

4. **SaveRequestData:**

   Saves request data (ride details, user info) into the Firestore 'requests' collection.

   Model Responsibility: Manages the storage of ride request-related data in Firestore.

5. **fetchRequestdata:**

   Retrieves request data based on status or user ID from the 'requests' collection in Firestore.

   Model Responsibility: Fetches ride request-related data from Firestore based on specified criteria.

6. **fetchRideStatus:**

   It Fetches ride status based on a specific rideID from the 'rides' collection in Firestore.

   Model Responsibility: Retrieves ride status data from Firestore.

7. **fetchOption1, fetchOption2** : used to fetch from my rides collection based on time 7:30 Am or 5:30 AM

8. **fetchUserProfile**:

   Role: Fetches user profile data or ride data from Firestore based on specific criteria.

   Model Responsibility: Handles data retrieval for user profiles or ride options from Firestore.

9. **updateRideStatus:**

   Updates the ride status in the 'rides' collection in Firestore.

Model Responsibility: Manages updates to ride status data in Firestore based on rideID.

10. updateRideOnCondition:

Role: Updates ride statuses based on certain conditions (e.g., approval, rejection, expiration).

Model Responsibility: Implements complex business logic to update ride statuses in Firestore based on specific conditions.

11. DeleteData :

Role: Deletes data from a specified Firestore collection.

Model Responsibility: Handles deletion operations on Firestore data.

6.1 **updateRideOnCondition method Illustration** and when it is called

Method Illustration : It encapsulates when to update the rides status according to the requests status and the time constraint.

1. 'approved' && currentTime == time:

Make the ride status active because it has been approved and it's time for the ride to start.

2. 'rejected':

If the request status is rejected then Cancel the ride because it has been explicitly rejected.

3. 'approved' && currentTime.isAfter(delayTime):

If the request status is approved and the current time is after a certain delay from the scheduled ride time.Then Mark the ride as completed because it's past the delayed time from the scheduled start.

4. 'expired':

If the request status is expired, Cancel the ride because it has expired without being initiated.

5. 'full trip':

If the request status is for a full trip when it exceeds the available seats in the car Then, Set the ride status as 'Full', possibly indicating that the vehicle or capacity for the ride is full.

6. Default Case:

If none of the above conditions match Then ,Set the ride status as 'waiting', likely indicating that the ride is pending or waiting for some action or condition to proceed.

When it is called?

At the init_state() function in the profile page of driver and the user to be continuously updating the status once anyone of them opens the application first.

6.2 Driver handling when to accept or reject inside the accept and the reject buttons I encapsulate the logic to change the request status ,

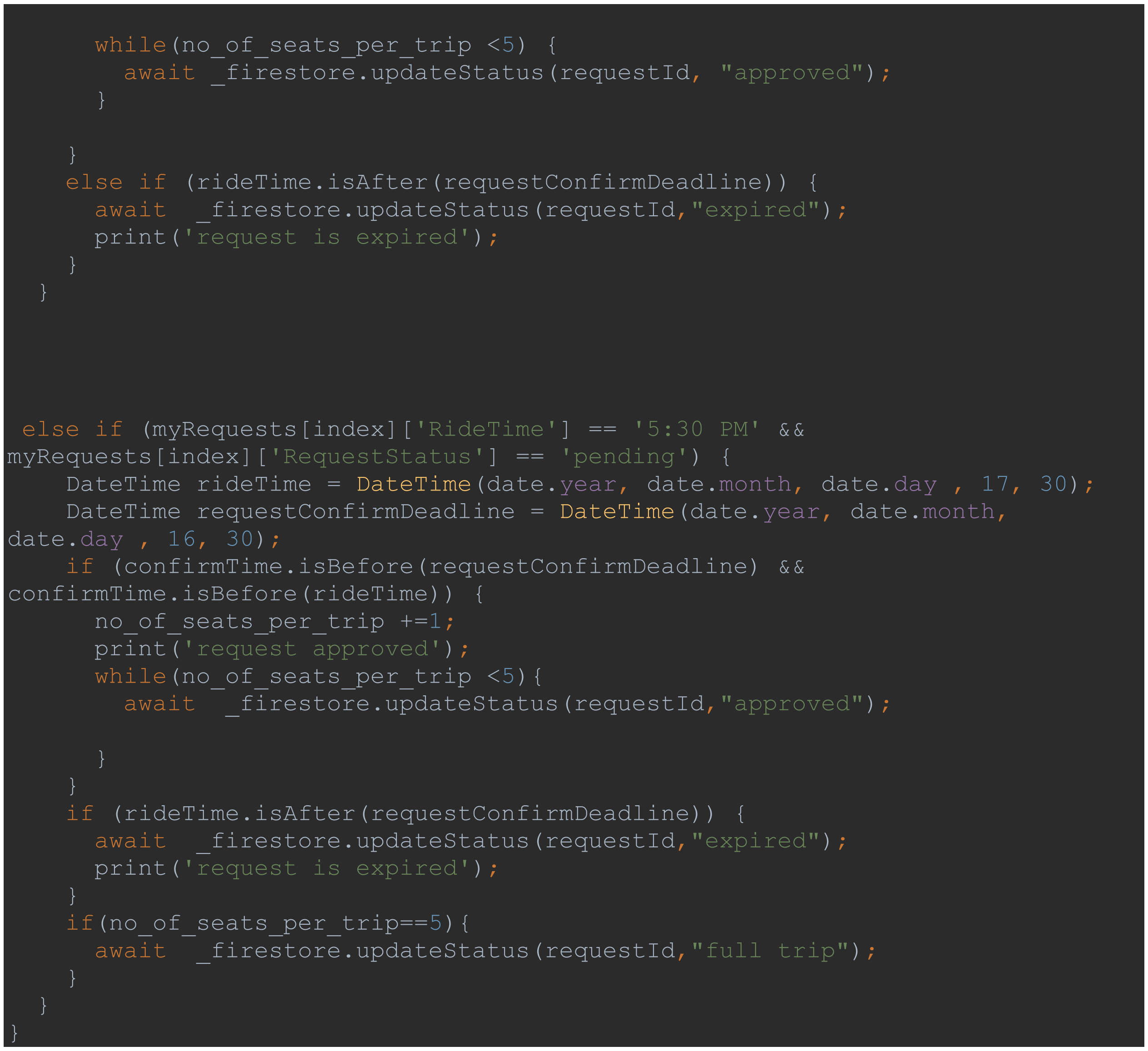
- Note that the default status for ride at offering is accepting which means the driver is in service.

- **Once the user makes a request status become pending directly**

Logic code in RequestPage in the driver App

```
if(no_of_seats_per_trip >5){
    ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('error : number of seats is completed
'),));
  }
var requestData = myRequests[index].data();
print(requestData);
if (requestData != null) {
  DateTime date= DateTime.parse(myRequests[index]['RideDate']);
  if (myRequests[index]['RideTime'] == '7:30 AM' &&
myRequests[index]['RequestStatus'] == 'pending') {
    DateTime rideTime = DateTime(date.year, date.month, date.day , 7, 30);
    DateTime requestConfirmDeadline = DateTime(date.year, date.month,
date.day -1, 23, 30);

    if (confirmTime.isBefore(requestConfirmDeadline) &&
confirmTime.isBefore(rideTime)) {
      print('request approved');
      no_of_seats_per_trip +=1;
```



```
      while(no_of_seats_per_trip <5) {
        await _firestore.updateStatus(requestId, "approved");
      }

    }
    else if (rideTime.isAfter(requestConfirmDeadline)) {
      await  _firestore.updateStatus(requestId,"expired");
      print('request is expired');
    }
  }




 else if (myRequests[index]['RideTime'] == '5:30 PM' &&
myRequests[index]['RequestStatus'] == 'pending') {
    DateTime rideTime = DateTime(date.year, date.month, date.day , 17, 30);
    DateTime requestConfirmDeadline = DateTime(date.year, date.month,
date.day , 16, 30);
    if (confirmTime.isBefore(requestConfirmDeadline) &&
confirmTime.isBefore(rideTime)) {
      no_of_seats_per_trip +=1;
      print('request approved');
      while(no_of_seats_per_trip <5){
        await  _firestore.updateStatus(requestId,"approved");

      }
    }
    if (rideTime.isAfter(requestConfirmDeadline)) {
      await  _firestore.updateStatus(requestId,"expired");
      print('request is expired');
    }
    if(no_of_seats_per_trip==5){
      await  _firestore.updateStatus(requestId,"full trip");
    }
  }
}
```

6.3 Conditions for Handling Ride Requests:

1. **Checking Available Seats**:
    - **Condition**: If the number of seats available for the trip is greater than 5.
    - **Action**: Displays an error message using a **SnackBar** if the number of available seats for the trip exceeds the limit of 5.
2. **Processing Ride Requests**:
    - Retrieves the request data and extracts details such as the ride date, time, and request status.
    - Based on the ride time and request status, different scenarios are handled:
        - For a ride scheduled at '7:30 AM' with a pending status:
            - Determines deadlines for confirmation and the ride time.
            - Checks if the confirmation time is within the deadline and approves the request if conditions are met. If the ride is about to start and seats are available, it tries to approve multiple times until the seat limit is reached.
            - Handles scenarios where the ride time exceeds the confirmation deadline, marking the request as 'expired'.
        - For a ride scheduled at '5:30 PM' with a pending status:
            - Similar to the previous scenario, but with different timings and conditions.
            - Approves the request based on confirmation deadlines and ride time, potentially marking it as 'full trip' if the seat limit is reached.
            - Handles cases where the ride time exceeds the confirmation deadline, marking the request as 'expired'.
3. **Seat Management**:
    - Throughout these conditions, there's a mechanism (**no_of_seats_per_trip**) to track and increment available seats. The code ensures that the approval loop continues until the available seats reach the maximum limit.

6.4 Conditions to make reservation in TripDetails.dart file in user app

Inside the Add to cart button

```
onPressed: () async {

        if(Data['time'].toString()=='7:30 AM') {
          DateTime rideTime = DateTime(date.year, date.month, date.day, 7,
30);
          DateTime reservationDeadline= DateTime(date.year, date.month,
date.day -1, 22, 00);
          if (reservationTime.isBefore(reservationDeadline) &&
reservationTime.isBefore(rideTime)) {
    print('Reservation done successfully');
    await _firestore.SaveRequestData(uid, email, start, destination, RideID,
RideDate, RideTime, RequestStatus,DriverID);

    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Request done successfully , waiting for driver
accept.'),
      ),

    );

  }
  else {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('error: Reservation for the ride at 7:30 am is
closed.'),
      ),
    );
  }
}
else if(Data['time'].toString()=='5:30 PM') {

          DateTime rideTime = DateTime(date.year, date.month, date.day, 17,
30);
          DateTime reservationDeadline= DateTime(date.year, date.month,
date.day , 13, 00);
          print(time);

  if (reservationTime.isBefore(reservationDeadline) &&
reservationTime.isBefore(rideTime)) {
    print('Reservation done successfully');
    await _firestore.SaveRequestData(uid, email, start, destination, RideID,
RideDate, RideTime, RequestStatus,DriverID);
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Request done successfully!'),
      ),
    );    }

  else {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('error: Reservation for the ride at 5:30pm is
closed.'),
      ),
    );
  }
}
```

Reservation Handling for Specific Ride Times:

1. **Reservation at 7:30 AM:**
   - Checks if the current reservation time is before a reservation deadline (reservationDeadline) and before the scheduled ride time (rideTime).
   - If the conditions are met, the reservation is successful. It saves the request data using _firestore.SaveRequestData and displays a success message.
   - If the reservation time exceeds the deadlines, it displays an error message indicating that the reservation for the 7:30 AM ride is closed.
2. **Reservation at 5:30 PM:**
   - Similar to the previous scenario but with different timing conditions (rideTime and reservationDeadline) specific to the 5:30 PM ride.
   - It checks if the reservation time is within the defined deadline and before the ride time, allowing successful reservation and displaying a success message.
   - If the reservation time surpasses the defined deadlines, it shows an error message indicating that the reservation for the 5:30 PM ride is closed.

7. Video link

https://drive.google.com/file/d/1NIpW6P1xM5pPyIDnTdb91IVqcF8tj-HT/view?usp=sharing

8. Github link

https://github.com/Shaimaa-moh/CarPoolApp