Carpool Application

Yara Mostafa Ibrahim Mohamed

19P1120 G1 Sec 1

Contents

Intro

Welcome to the documentation for Carpool, a revolutionary rideshare application designed to streamline transportation services for the Faculty of Engineering Community at Ainshams University. This rideshare platform is specifically tailored to connect students, offering a convenient and trusted means of commuting to and from Abdu-Basha and Abbaseya square within the university campus (specifically Gate 4 and Gate 3).

Test accounts

User:

Email: yara@eng.asu.edu.eg

Password: sawsaw2

Driver:

Email: sama@eng.asu.edu.eg

Password: sawsaw2

Specs

User side

1- Login Page with Firebase Authentication:
   - Provide a seamless sign-up option using Firebase authentication.
   - User can only create an account with @eng.asu.edu.eg domain

2- Trips List:
   - Display a list of available routes added by drivers to and from Ainshams campus, trip price and car number.
   - Only trips with “ongoing” state are displayed
   - Once the maximum number of passenger are completed the trip disappears from the list also if the driver declares the trip to be completed/canceled it will also be removed from the list of trips.
   - User can book 5:30 PM trips before 1:00 PM same day and for 7:30 AM at 10 PM previous day.
   - Utilizes the Recycler View for an organized presentation.

3- Order summary and Payment Page:
   - Review some details of the trip and finalize order.
   - Can choose either to pay with cash or card.
   - Time constraint is applied where the user can’t book a trip after a certain time limit (for 7:30 AM trip user can’t book trip after 10PM same day, for 5:30PM trip user can book after 1:00PM next day)

4- Order History and Tracking/Status Page:
   - History rides are displayed as a list and user is able to see the status of each trip
   - States can be: pending, accepted, completed, canceled, not confirmed.

5- User Profile:
   - User can display his email and username in this page online (Firebase) and offline (Room).

6- Database Integration:
    - Utilize Firebase real-time database for route and order status.
    - Implement Room database for profile data

Driver side

1- Login Page with Firebase Authentication:
    - Provide a seamless sign-up option using Firebase authentication.
    - User can only create an account with @eng.asu.edu.eg domain

2- Add trips and view trips:
    - In add trips, driver adds trips details which includes: takeoff, destination, and price, and car number, maximum number of passengers, date and time).
    - The driver is capable of inputting details for up to two trips daily, with a stipulation that each trip must occur at a distinct time—either at 5:30 am or 7:30 pm. Additionally, when selecting the 5:30 am time slot, the destination is restricted to either Gate 3 or Gate 4. Similarly, for the 7:30 pm time slot, the departure point is limited to Gate 3 or Gate 4.
    - Driver is able to view the trip and have the option to cancel it or select that it's completed after competition of the ride.

3- User Profile
    - Driver is able to view his username and email offline/online
    - There are two buttons in the profile page one for driver to check history and other to check requests on his created trips.

4- Accept requests
    - Driver chose's whether to accept passengers or declines requests for the ride and status is updated accordingly
    - Requests are declined after certain time constraint and state updated at user to "not confirmed"

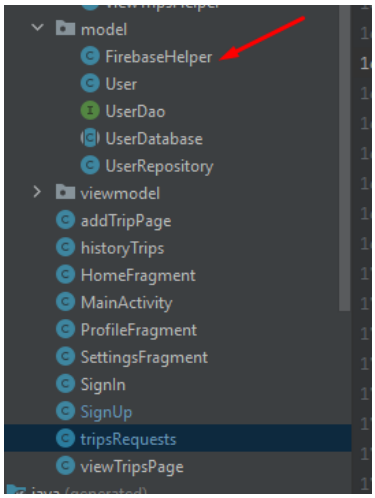
5- View history trips
   - States can be: ongoing, completed, canceled
   - Review trips history with the states stated above.

Additional features

- User can sign up from either apps and sign in both however when user signs in from carpool app (user side) info is saved in users, same applies at driver side, user can be a driver or/and client and get their data saved in real-time database as either driver, user or both.

- At 7:30 AM trips, destination must be either Gate 3 or Gate 4, also On 5:30 PM trips the takeoff must be from Gate 3 or Gate 4

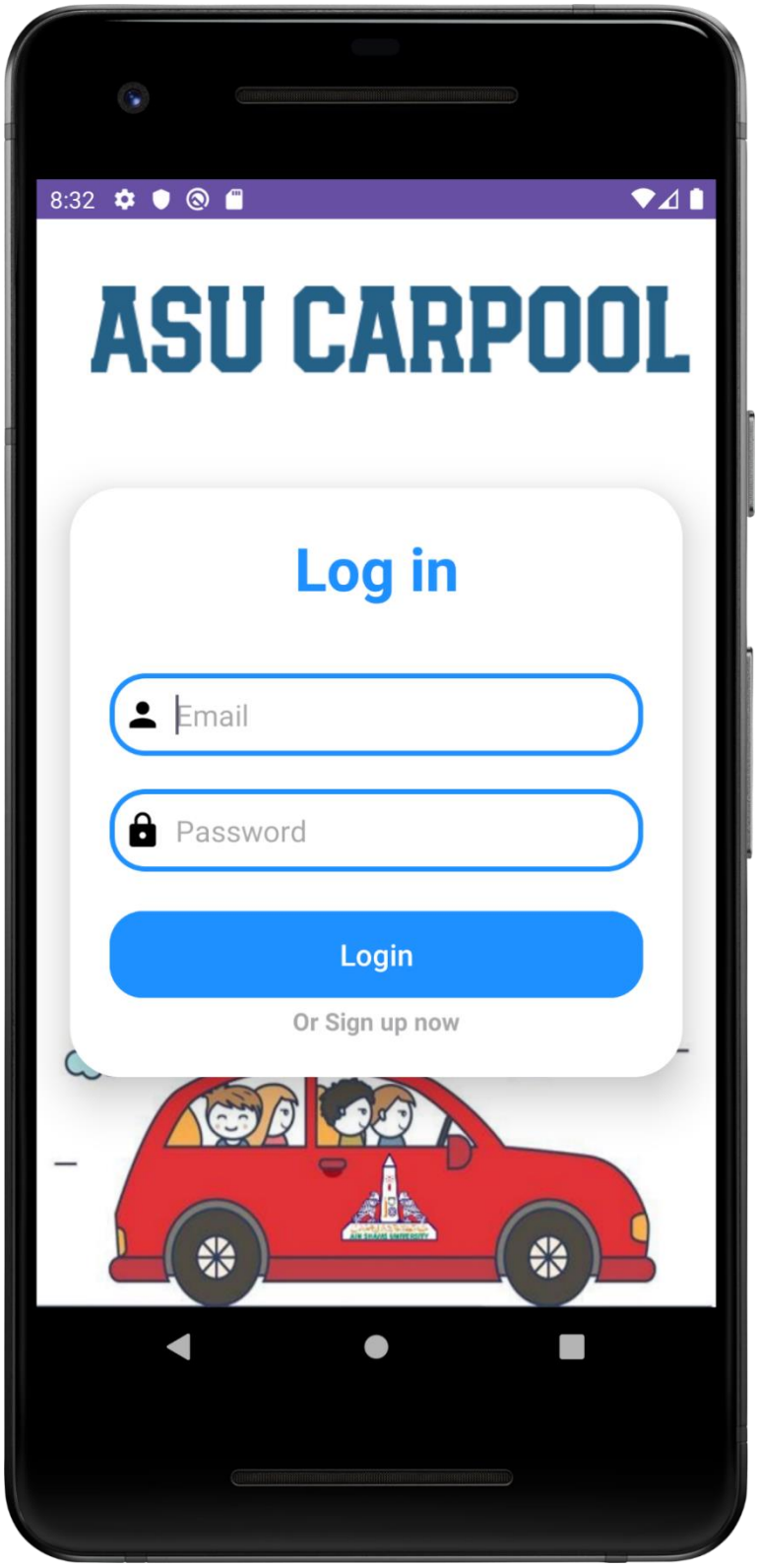
- Firebase is separated than View.

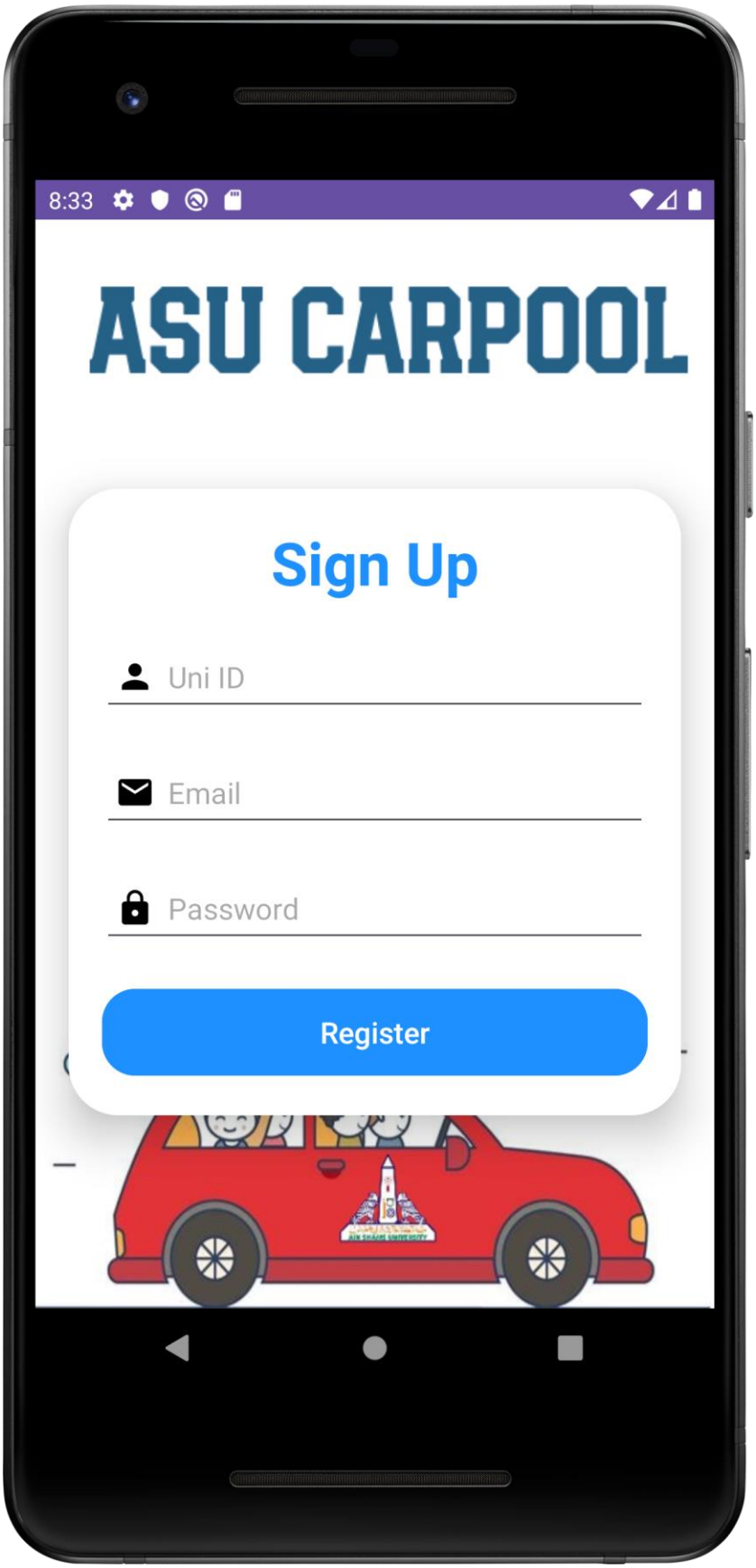


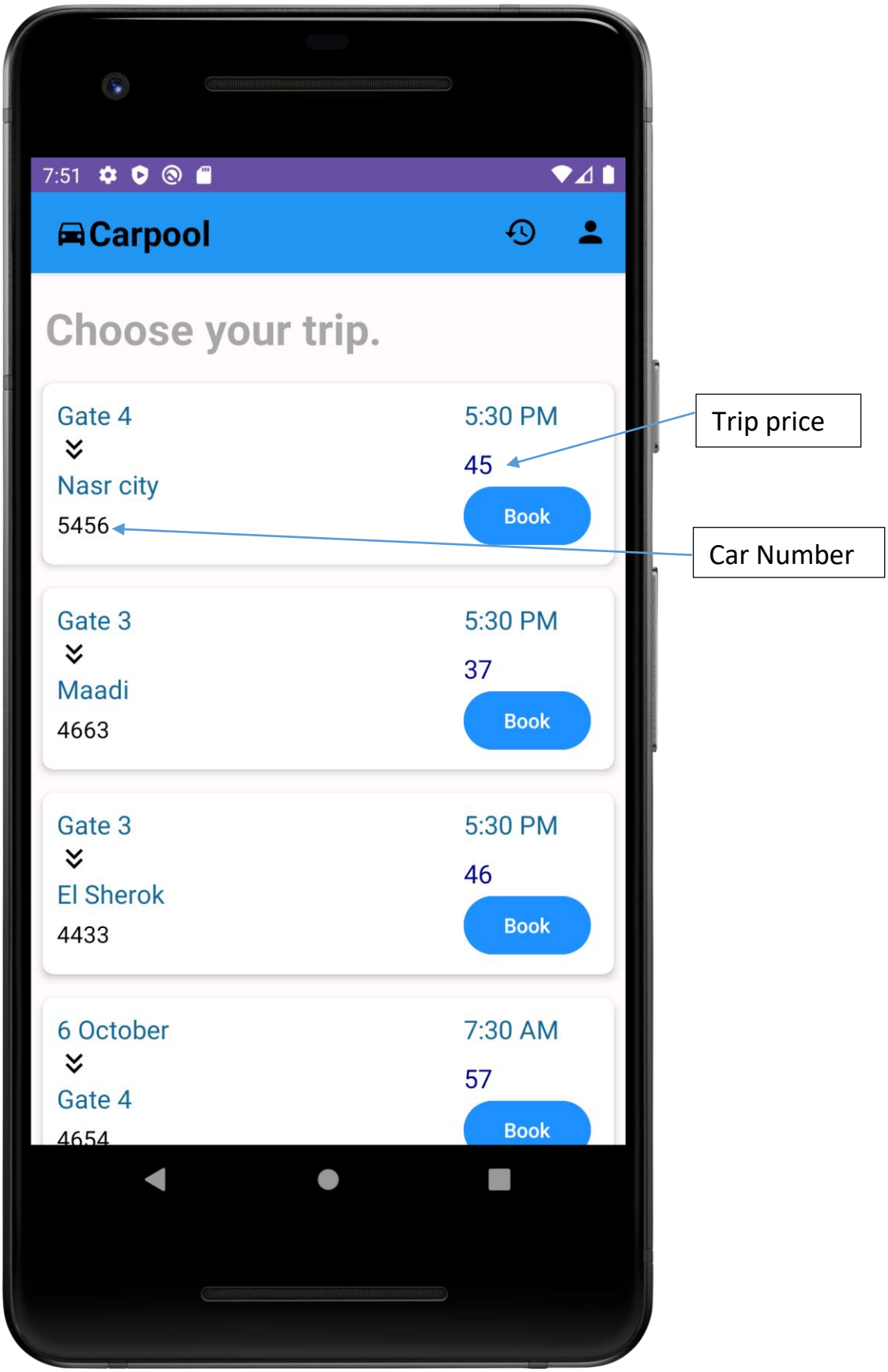
- If the driver decided to cancel a trip he can add another one same day and time, but as long as there is a trip ongoing driver can't add another trip with same time and date

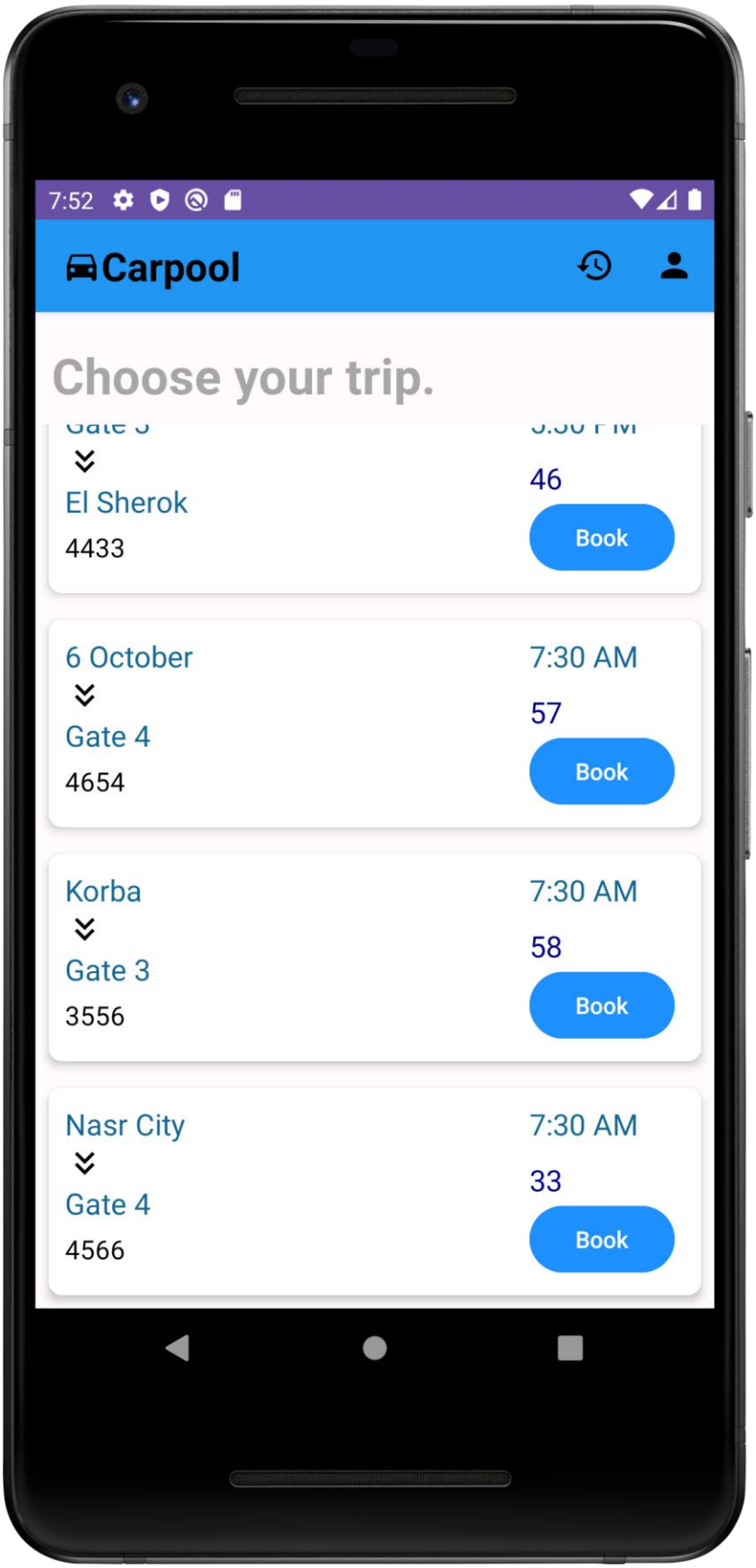
UI Design

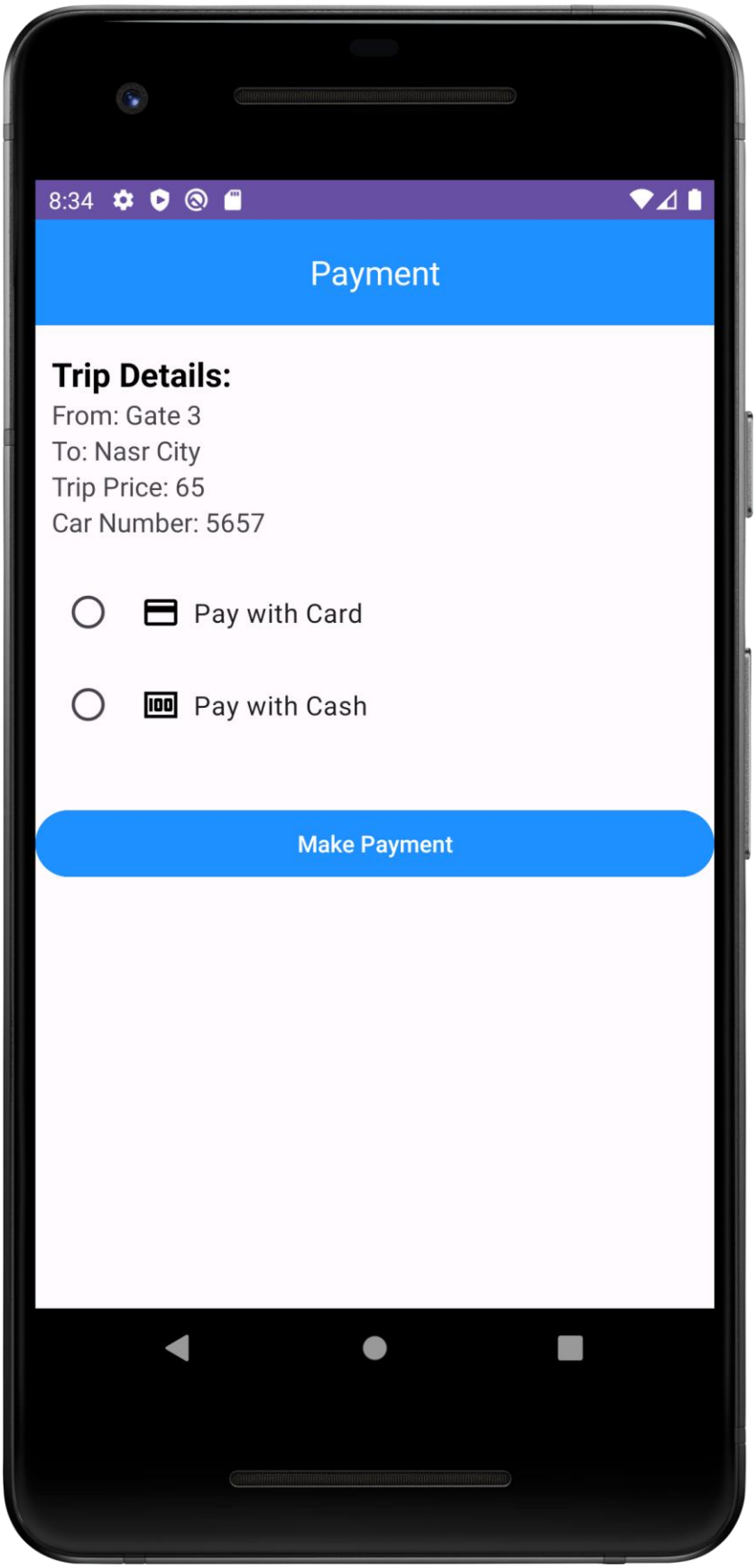
User side



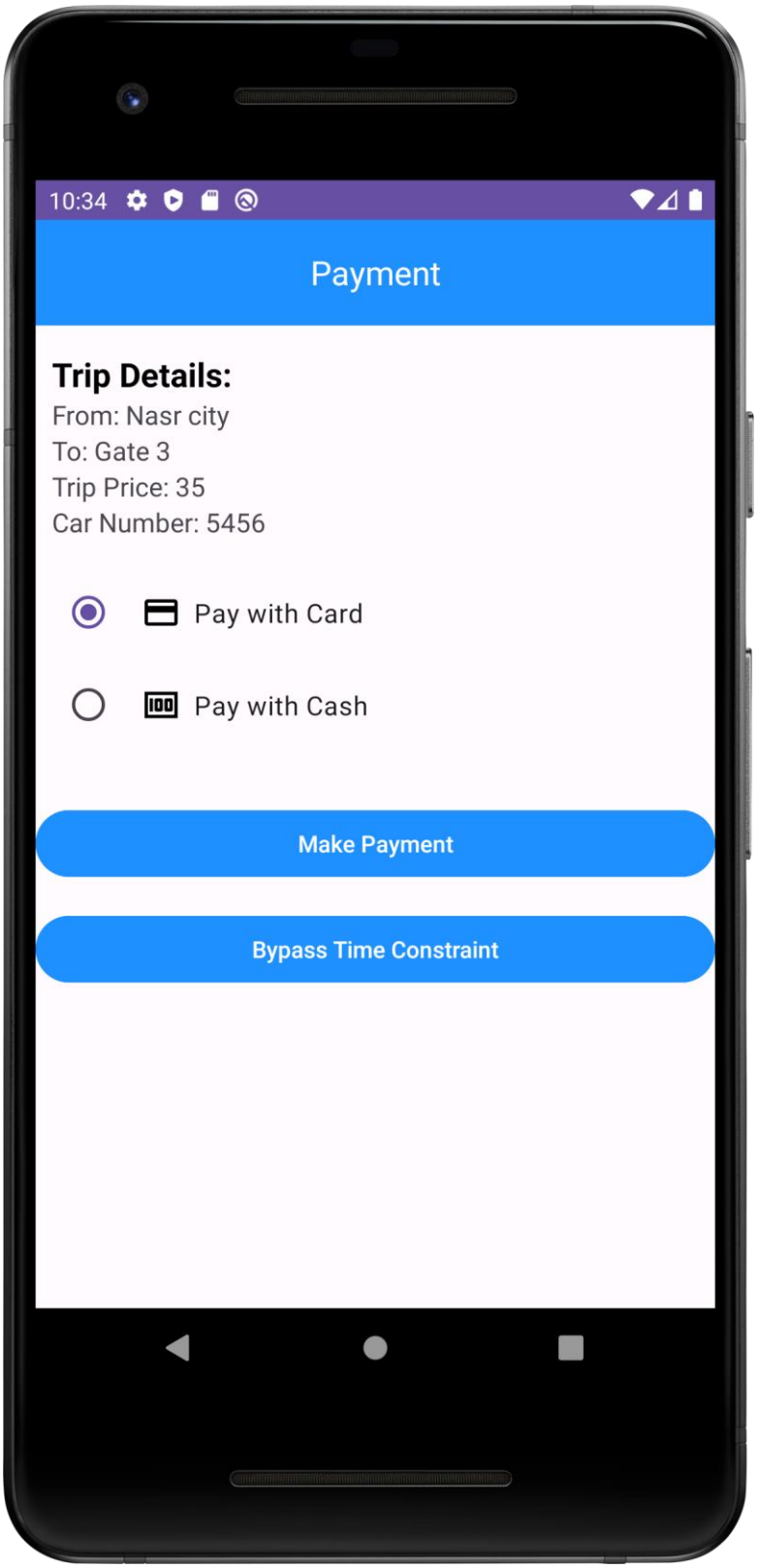

8:33
ASU CARPOOL
Sign Up
Uni ID
Email
Password
Register

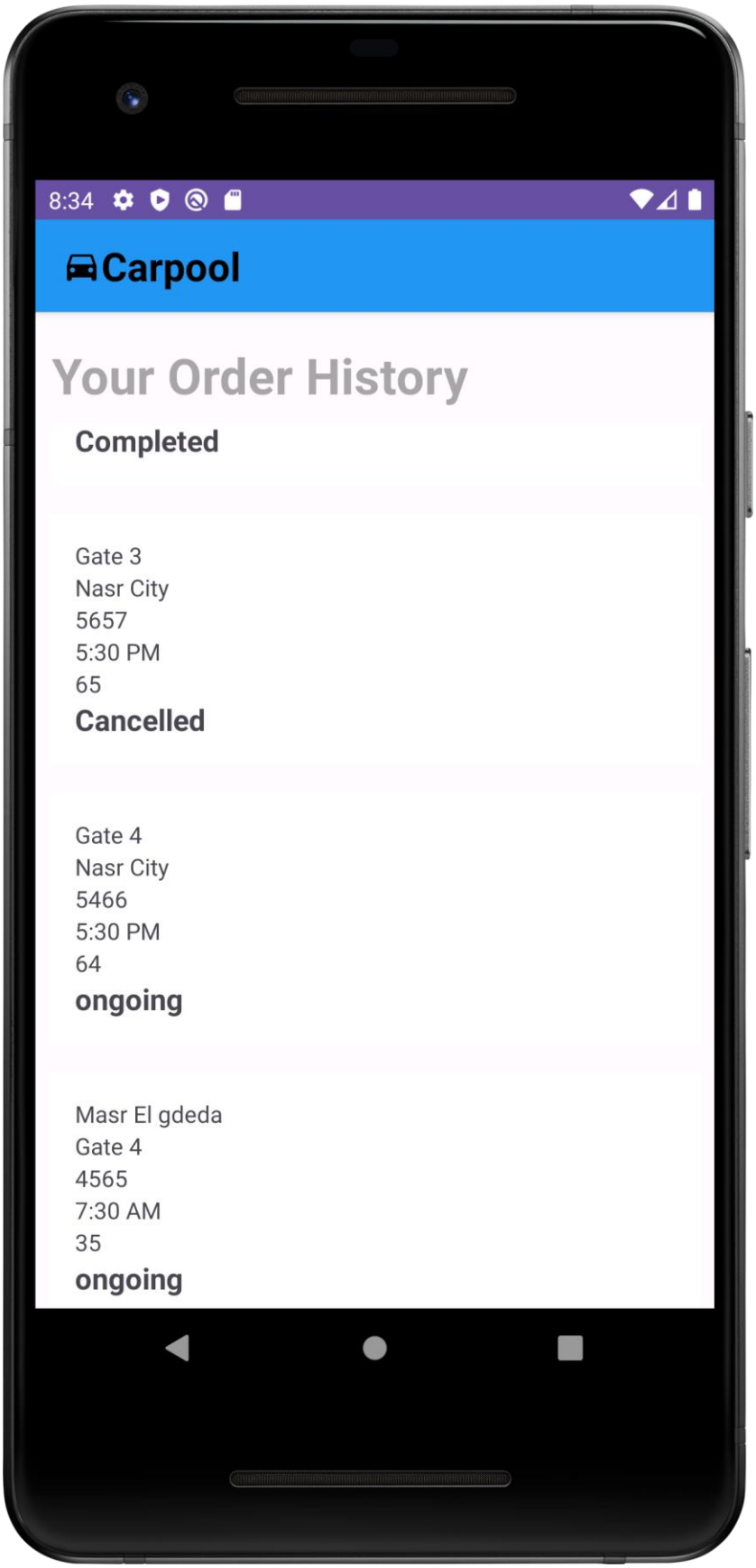

7:51
Carpool
Choose your trip.
Gate 4
Nasr city
5456
5:30 PM
45
Book
Gate 3
Maadi
4663
5:30 PM
37
Book
Gate 3
El Sherok
4433
5:30 PM
46
Book
6 October
Gate 4
4654
7:30 AM
57
Book
Trip price
Car Number

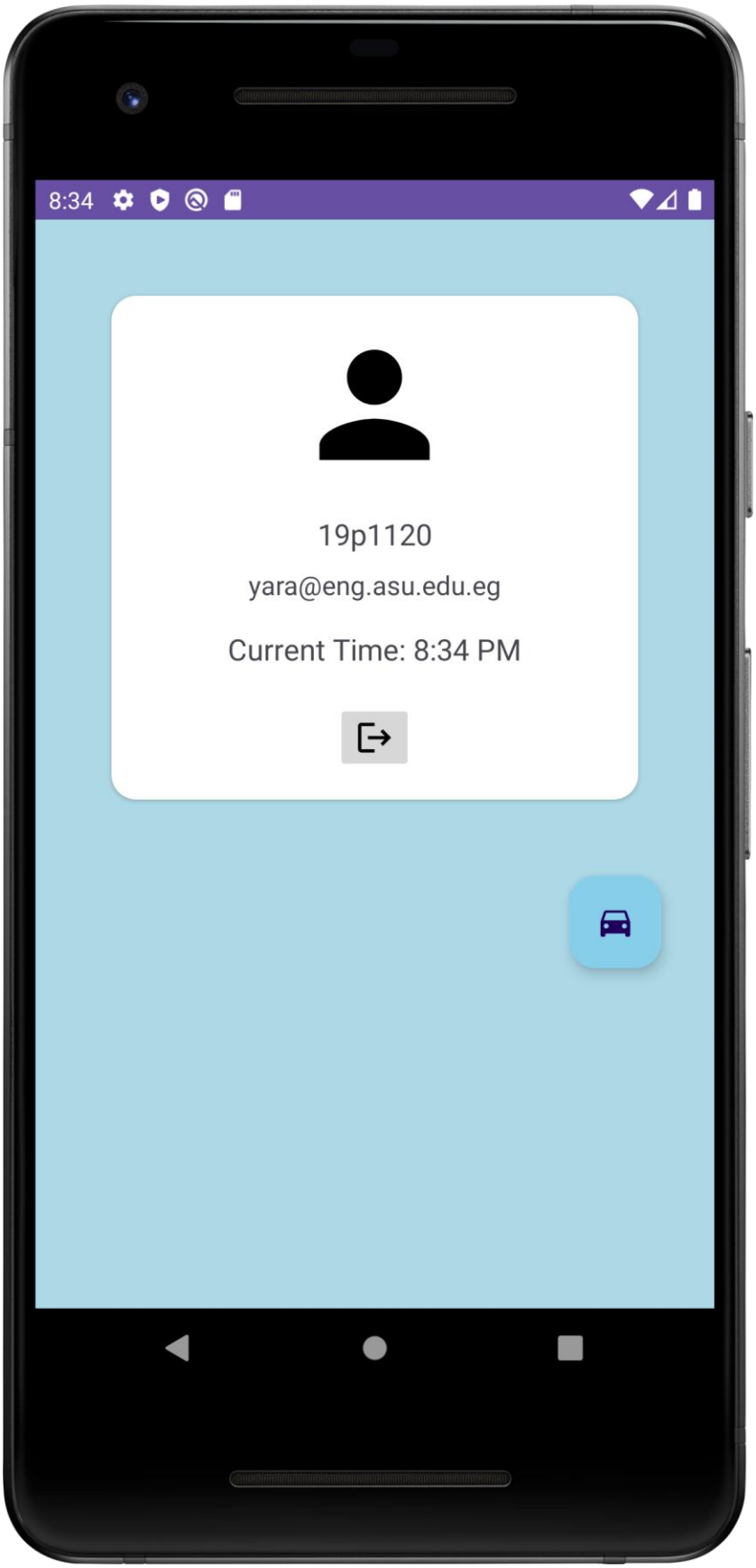

7:52
Carpool
Choose your trip.
El Sherok
4433
46
Book
6 October
Gate 4
4654
7:30 AM
57
Book
Korba
Gate 3
3556
7:30 AM
58
Book
Nasr City
Gate 4
4566
7:30 AM
33
Book


8:34
Payment
Trip Details:
From: Gate 3
To: Nasr City
Trip Price: 65
Car Number: 5657
Pay with Card
Pay with Cash
Make Payment

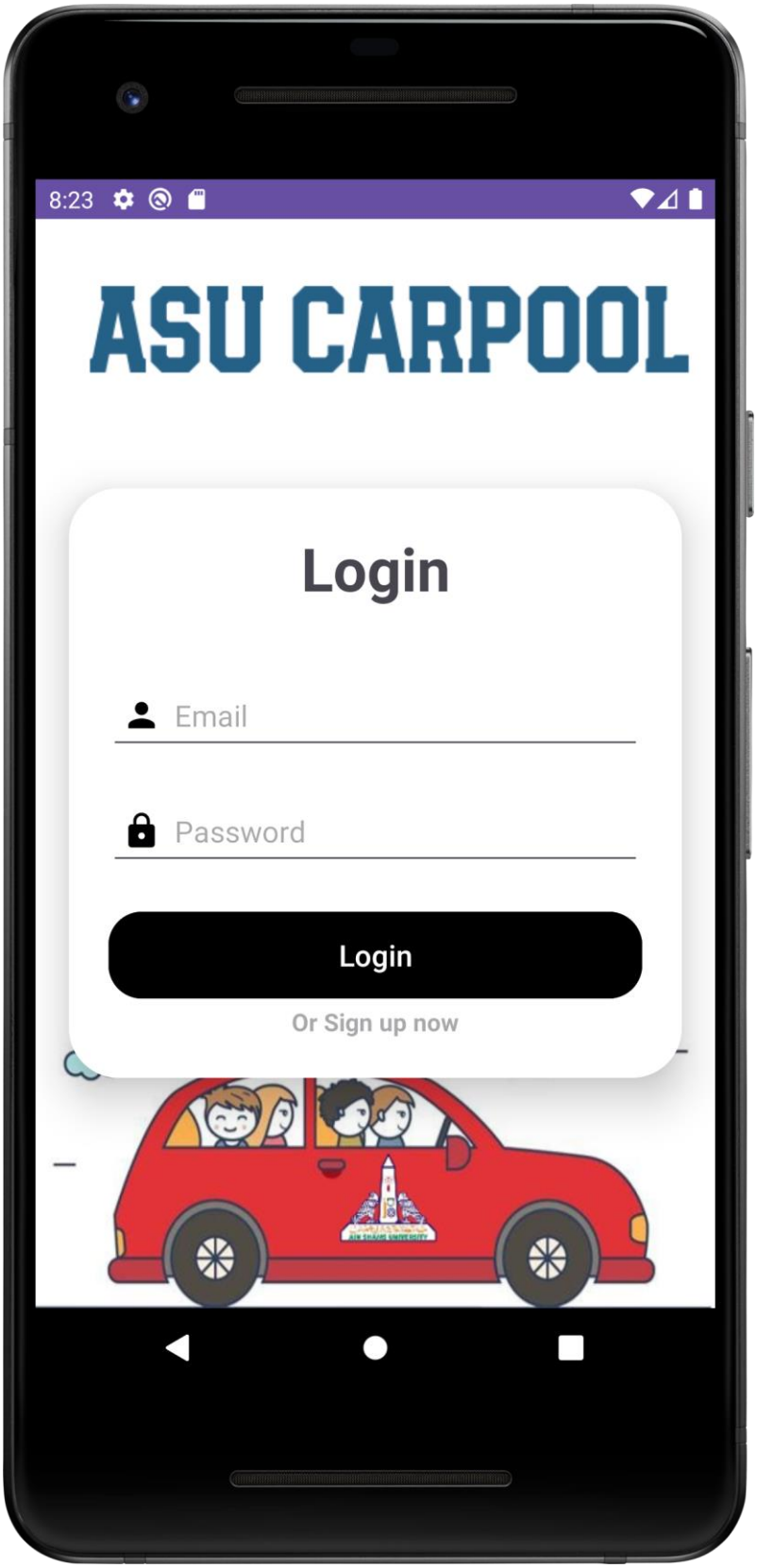
With bypass time constraint button

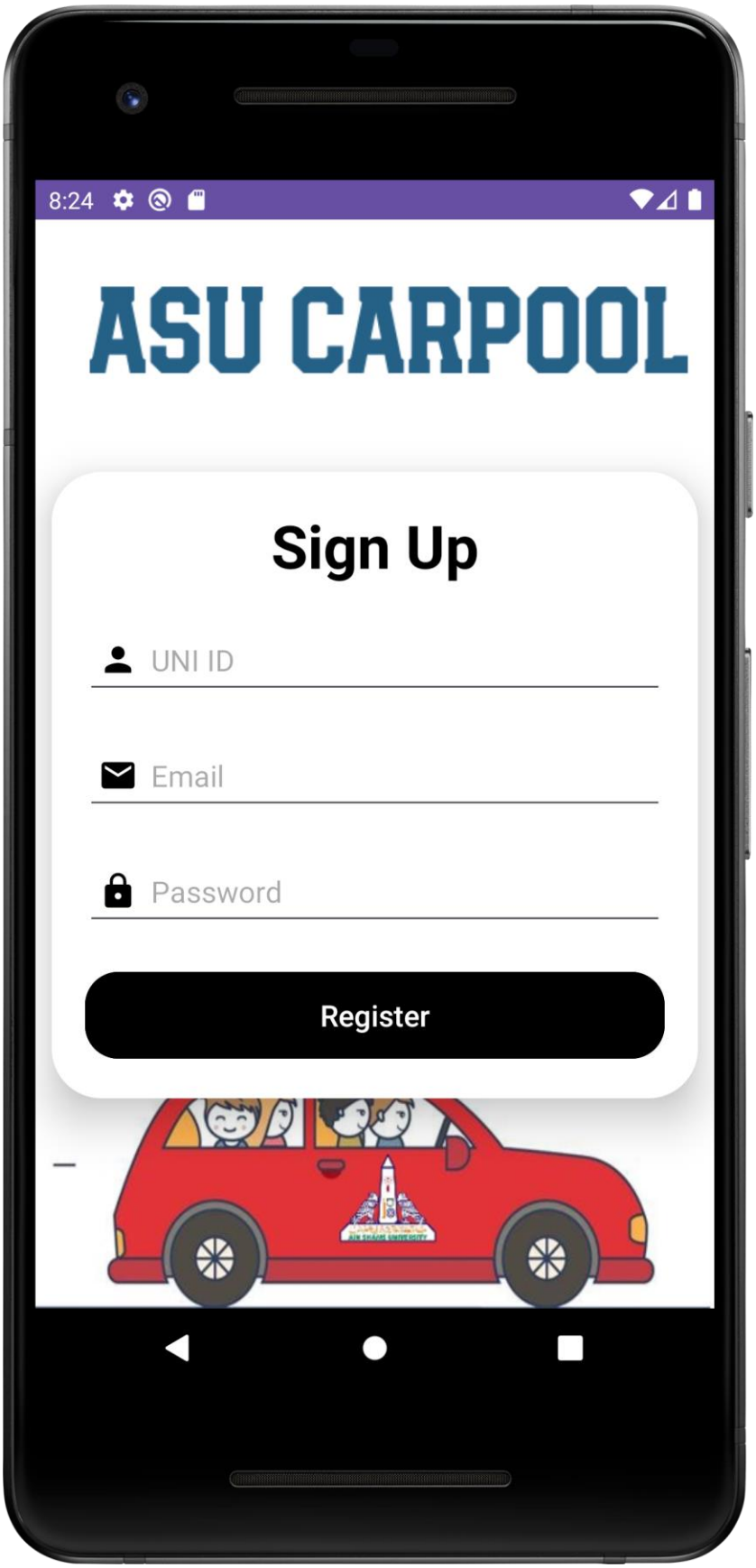


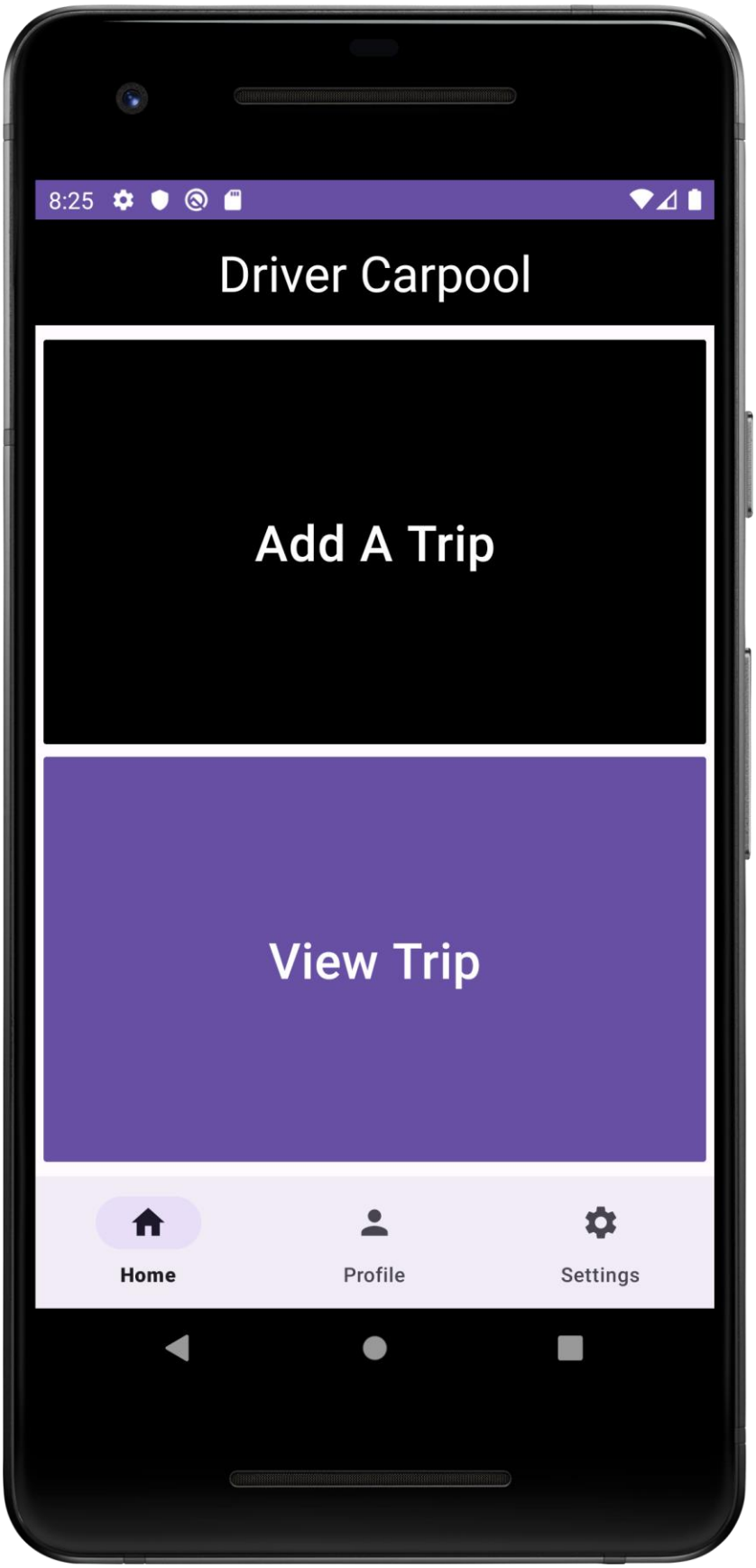

8:34
Carpool
Your Order History
Completed
Gate 3
Nasr City
5657
5:30 PM
65
Cancelled
Gate 4
Nasr City
5466
5:30 PM
64
ongoing
Masr El gdeda
Gate 4
4565
7:30 AM
35
ongoing

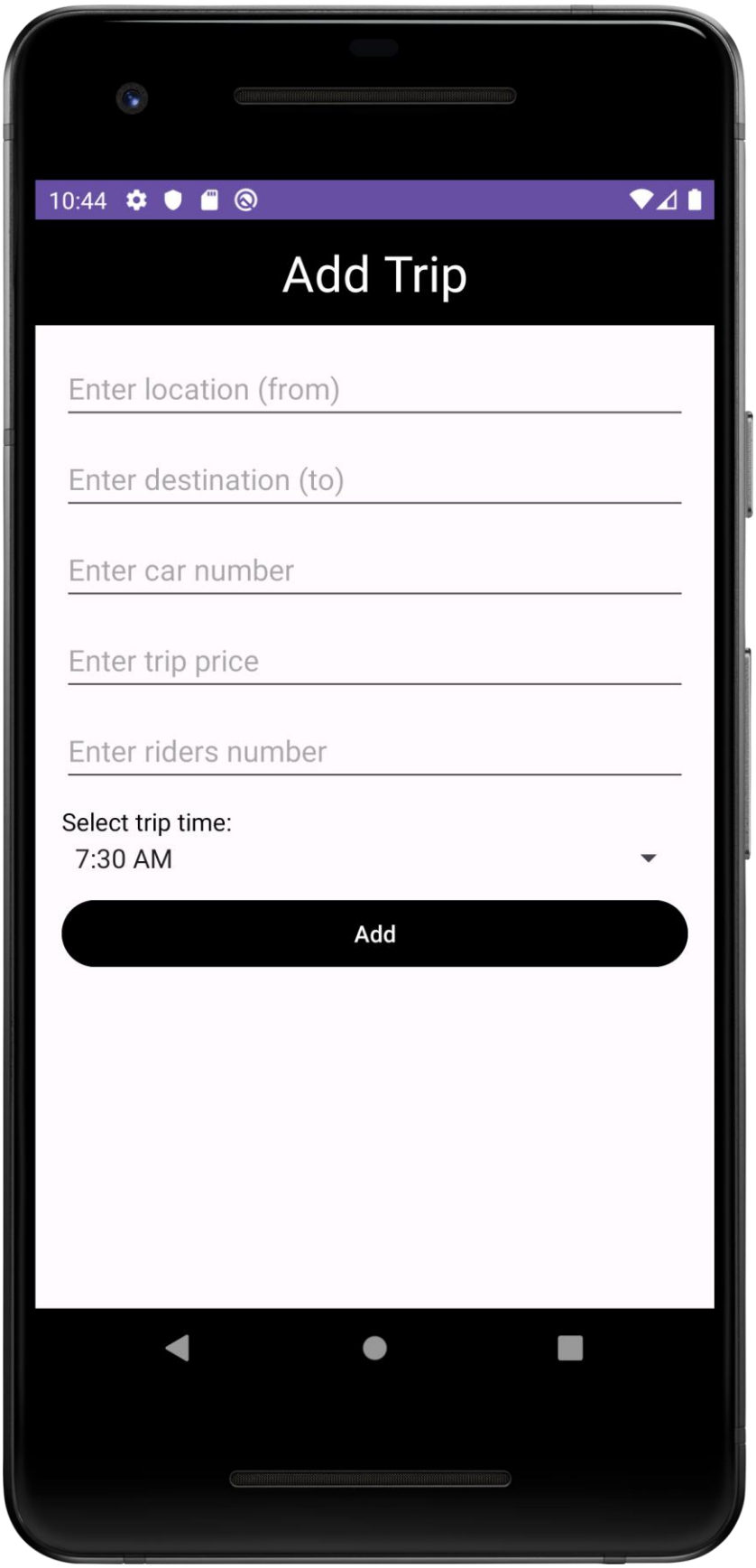

8:34
19p1120
yara@eng.asu.edu.eg
Current Time: 8:34 PM

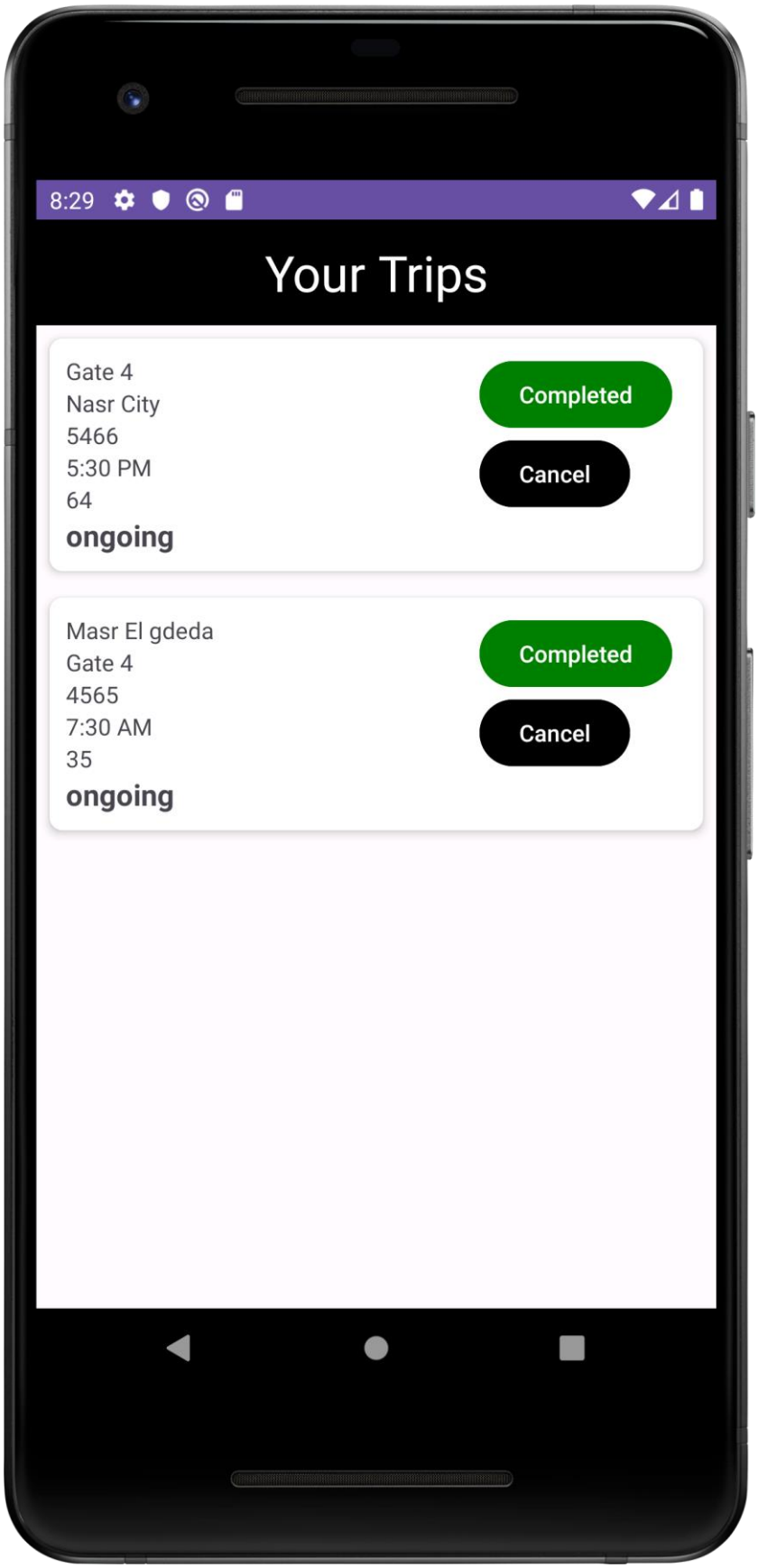
Driver side

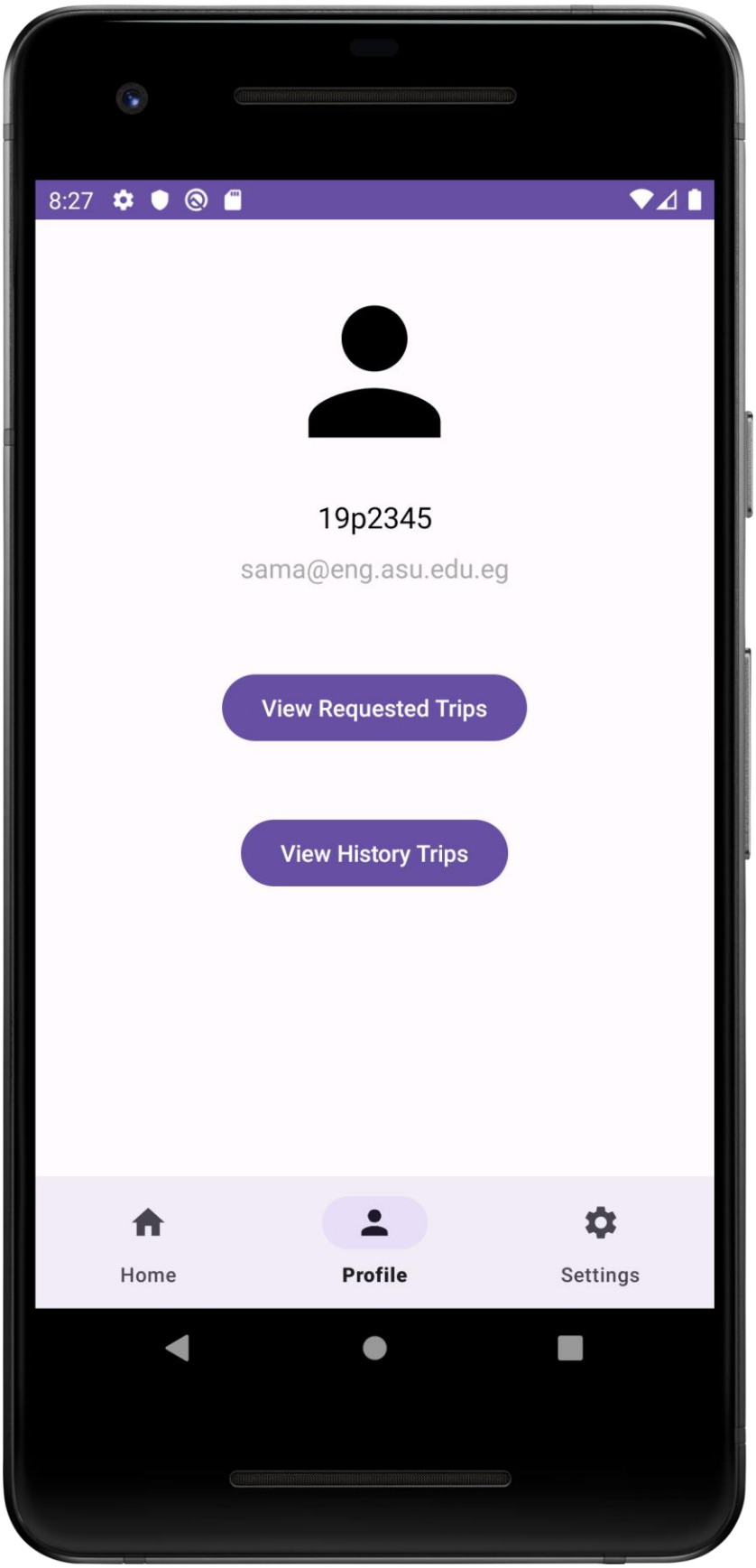


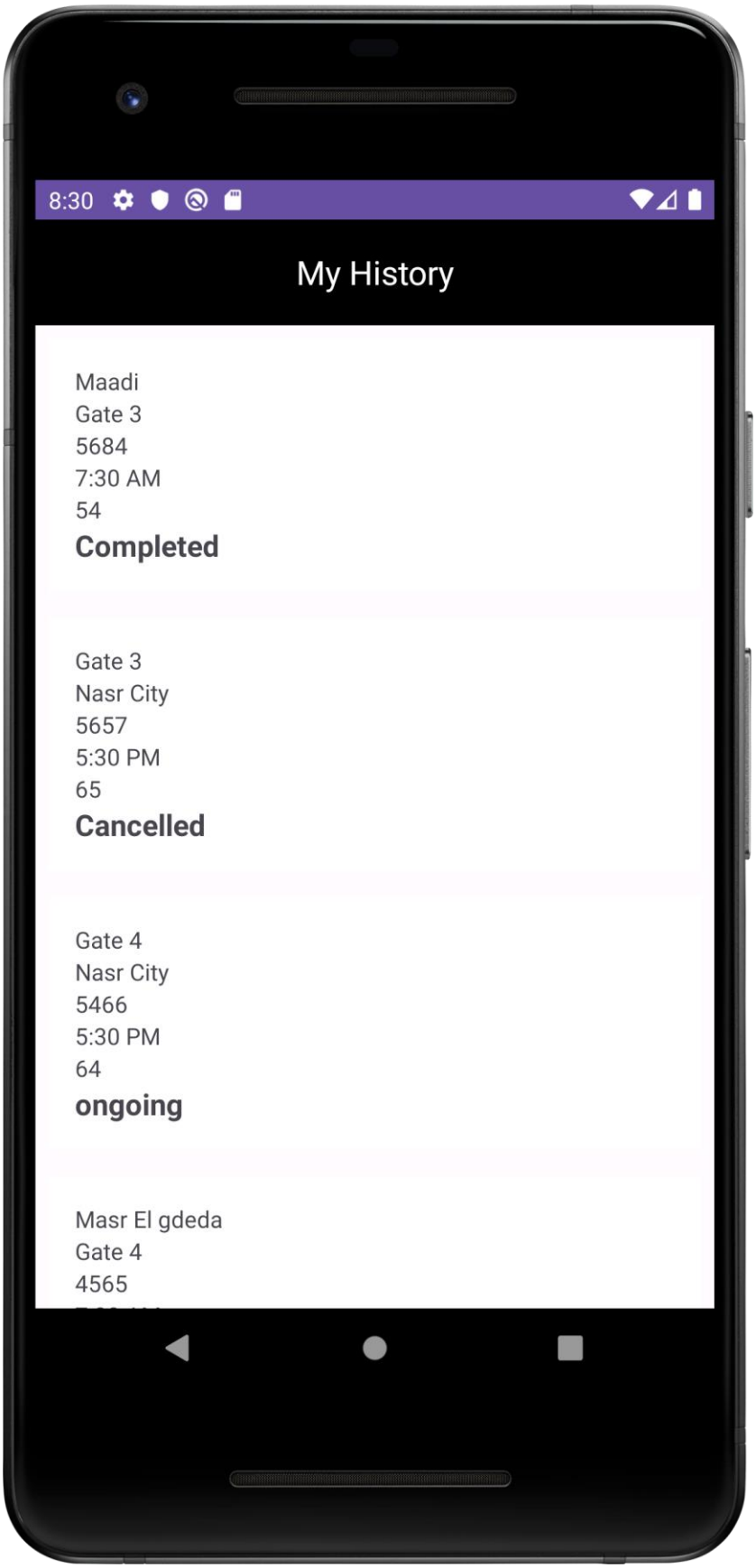

8:24
ASU CARPOOL
Sign Up
UNI ID
Email
Password
Register

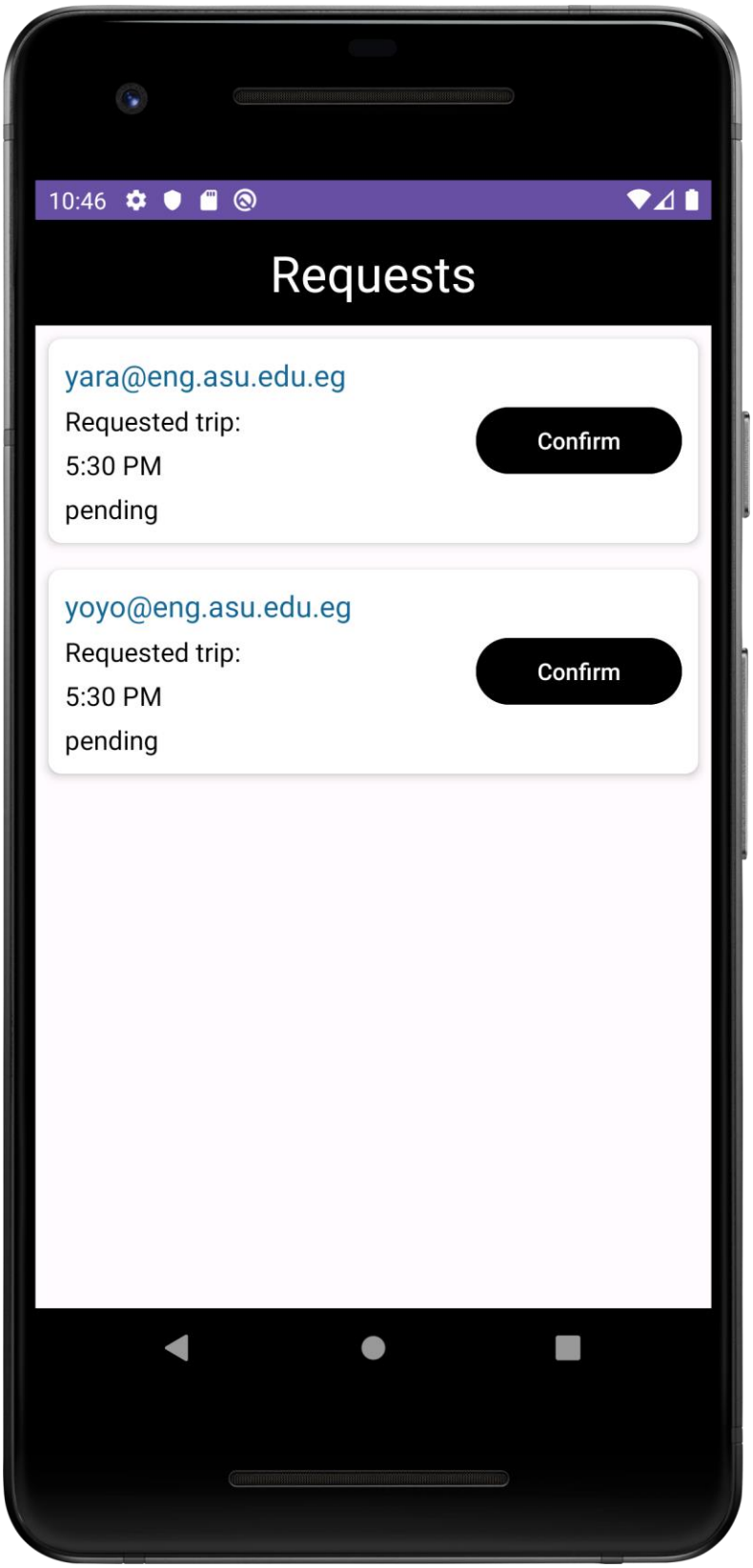

8:25
Driver Carpool
Add A Trip
View Trip
Home
Profile
Settings

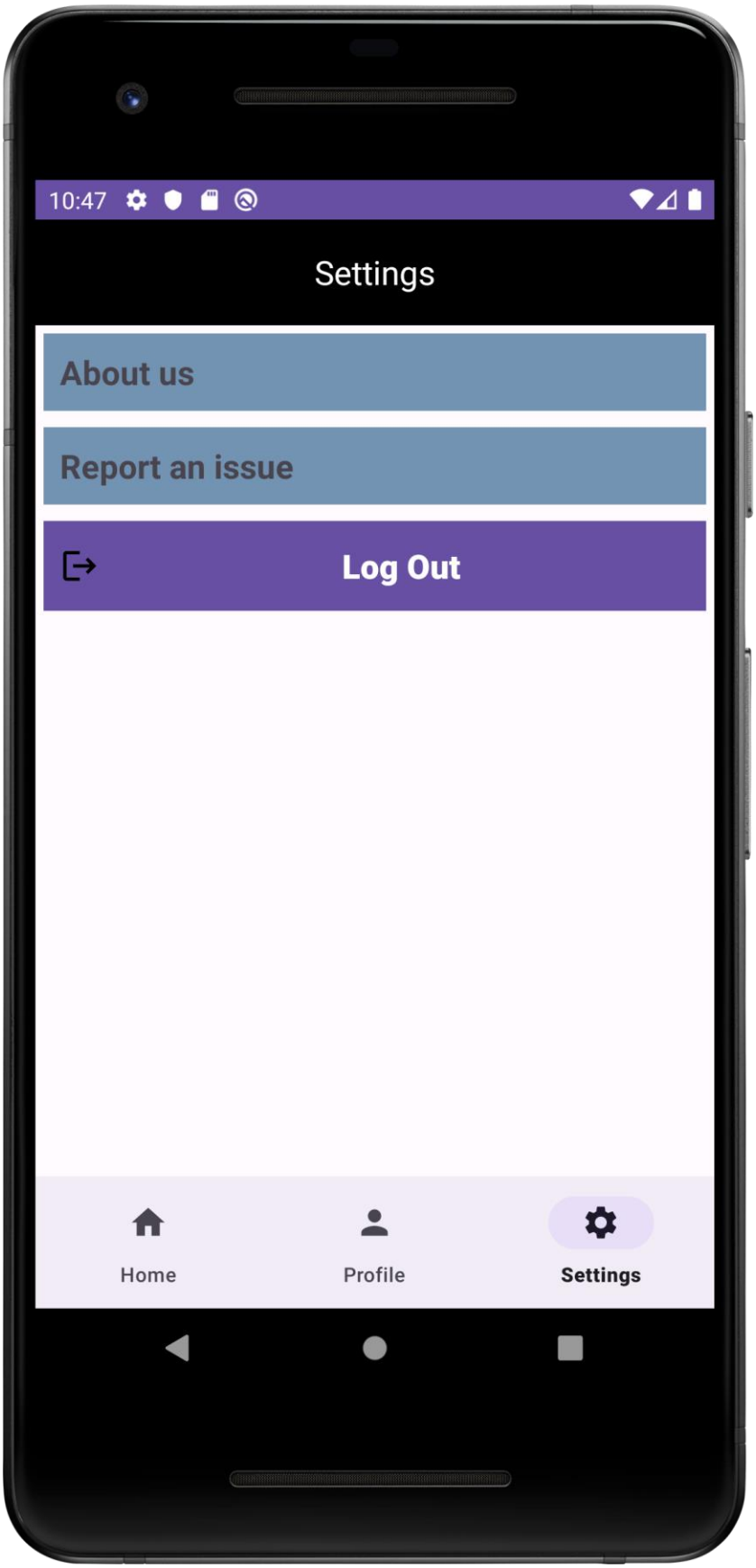

10:44
Add Trip
Enter location (from)
Enter destination (to)
Enter car number
Enter trip price
Enter riders number
Select trip time:
7:30 AM
Add


8:29
Your Trips
Gate 4
Nasr City
5466
5:30 PM
64
ongoing
Completed
Cancel
Masr El gdeda
Gate 4
4565
7:30 AM
35
ongoing
Completed
Cancel


8:27
19p2345
sama@eng.asu.edu.eg
View Requested Trips
View History Trips
Home
Profile
Settings


8:30
My History
Maadi
Gate 3
5684
7:30 AM
54
Completed
Gate 3
Nasr City
5657
5:30 PM
65
Cancelled
Gate 4
Nasr City
5466
5:30 PM
64
ongoing
Masr El gdeda
Gate 4
4565


10:46
Requests
yara@eng.asu.edu.eg
Requested trip:
5:30 PM
pending
Confirm
yoyo@eng.asu.edu.eg
Requested trip:
5:30 PM
pending
Confirm


10:47
Settings
About us
Report an issue
Log Out
Home
Profile
Settings

Database Structure

Real-time firebase

I chose to flatten out the structure of my real-time database as it is simpler to deal with and it was pointed out as best practice in firebase doc https://firebase.google.com/docs/database/web/structure-data

Avoid nesting data

Because the Firebase Realtime Database allows nesting data up to 32 levels deep, you might be tempted to think that this should be the default structure. However, when you fetch data at a location in your database, you also retrieve all of its child nodes. In addition, when you grant someone read or write access at a node in your database, you also grant them access to all data under that node. Therefore, in practice, it's best to keep your data structure as flat as possible.

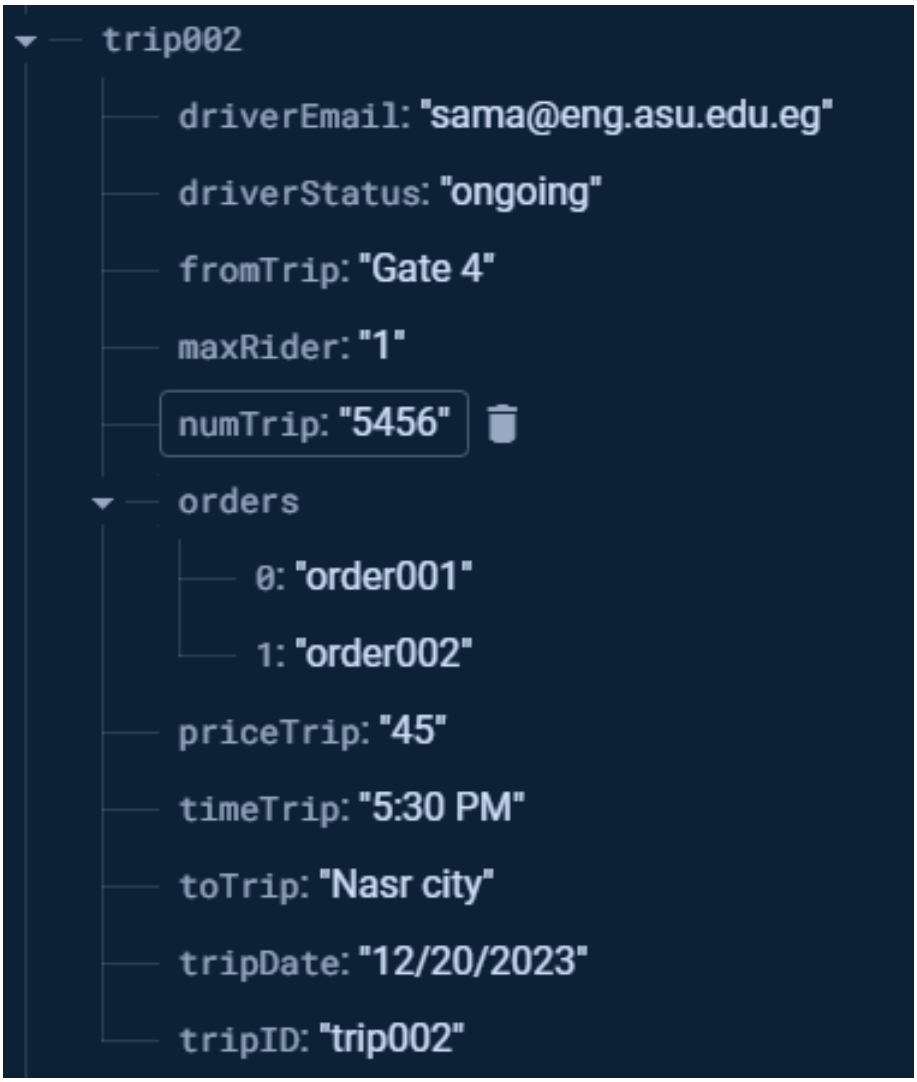
My database consists of 4 objects, drivers, orders, trips and users.

In drivers the email, id (from firebase Auth) and university id are saved.

Trips are where trips details are saved such as driver email, driver trip state (driverStatus) , take off (fromTrip) , destination (toTrip), max number of passengers in this trip(maxRider) (was set to 2 but was set to 1 as it decrements after each order made), car number (numTrip), orders ids on this trip, price of the trip, time of the trip, date and trip id so I can query on it.

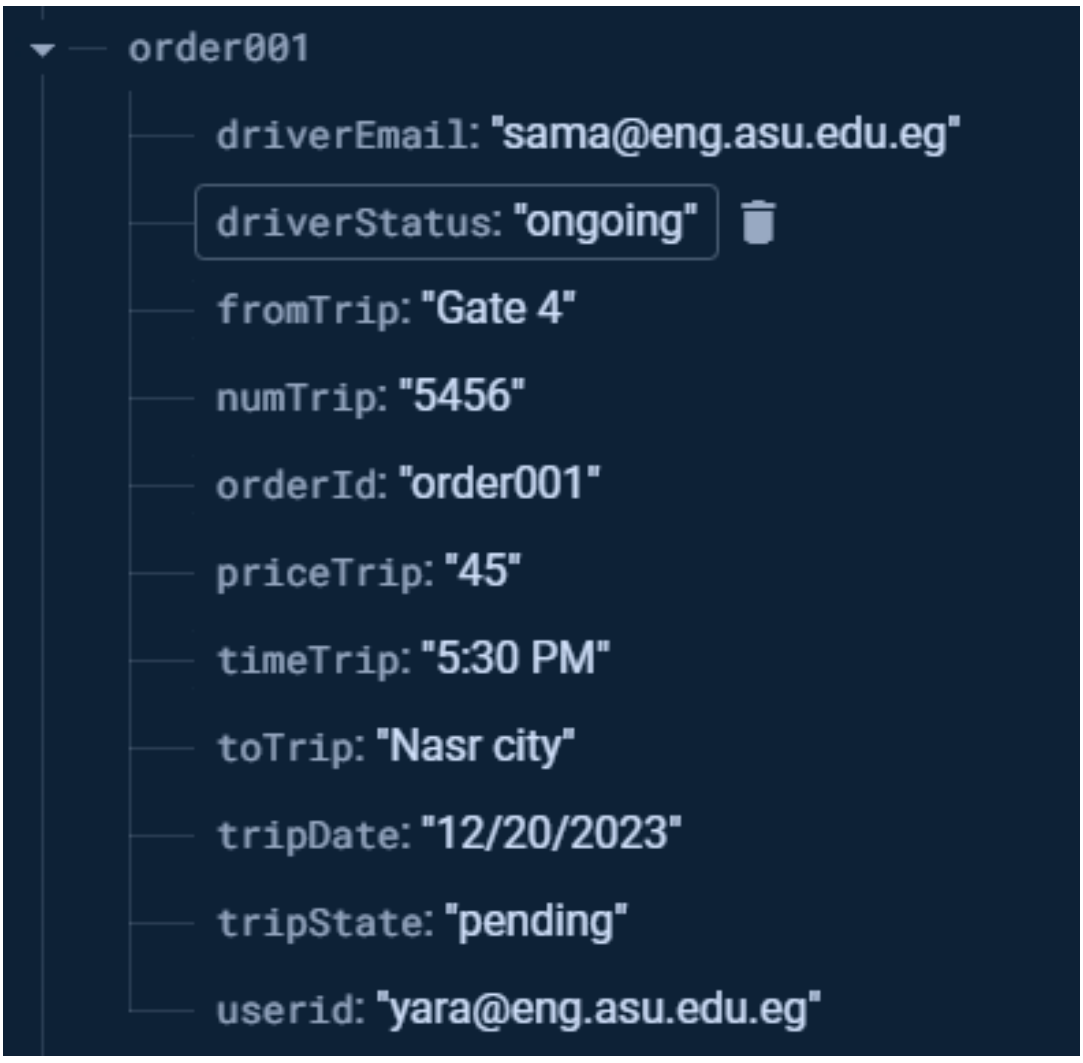
timeTrip can be 7:30 AM or 5:30 PM

driverStatus can be: ongoing (when trip is created), completed (after trip completation), canceled (if driver chooses to cancel the trip).



Orders are where orders details are saved such as driver email, driver trip state (driverStatus) , take off (fromTrip) , destination (toTrip) , car number (numTrip), orders ids on this trip, price of the trip, time of the trip, date and order id so I can query on it, email of user that ordered that trip (userid), trip state that shows the trip state to the user (tripState)

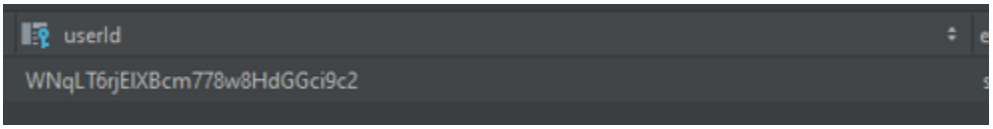
tripState: pending (when order state still not confirmed), accepted (when driver accepts the trip), completed (after trip completation), canceled (if driver chooses to cancel the trip), not confirmed (if deadline passes and driver doesn't confirm user request on a trip)

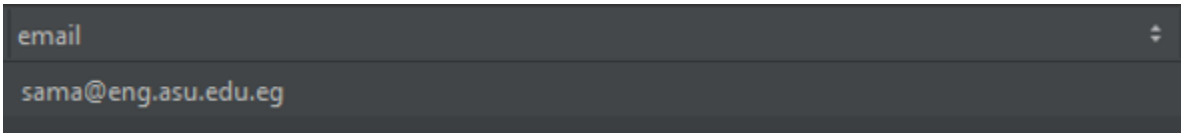


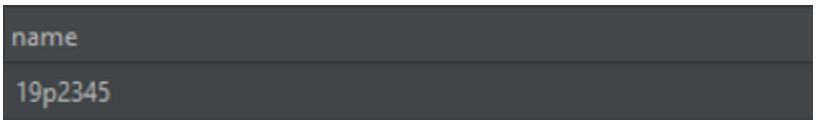
In Users the email, id (from firebase auth) and university id are saved.

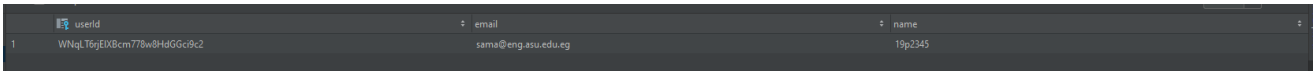
Room Database

I save the id, email and name of the users to be displayed on profile offline/online





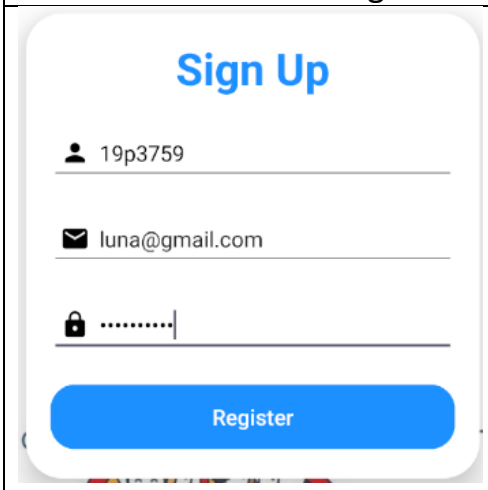
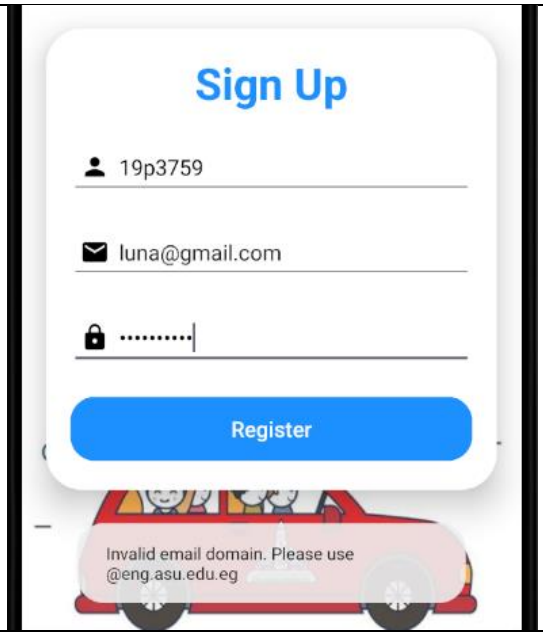


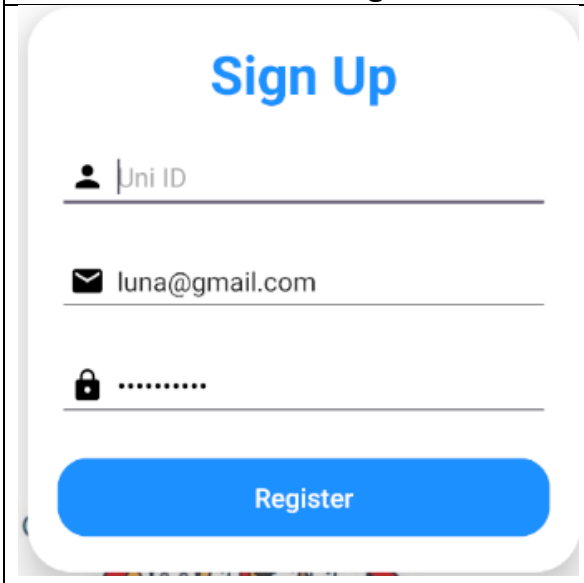
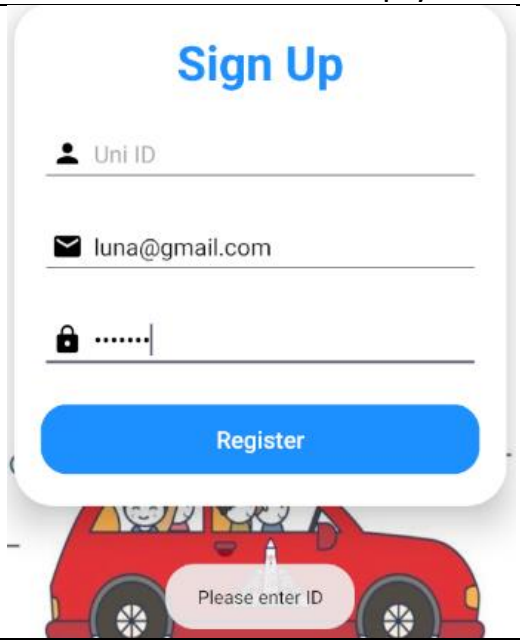


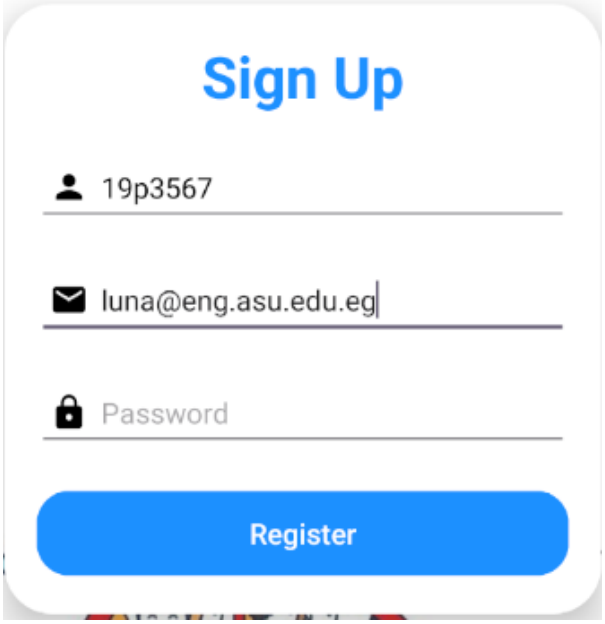
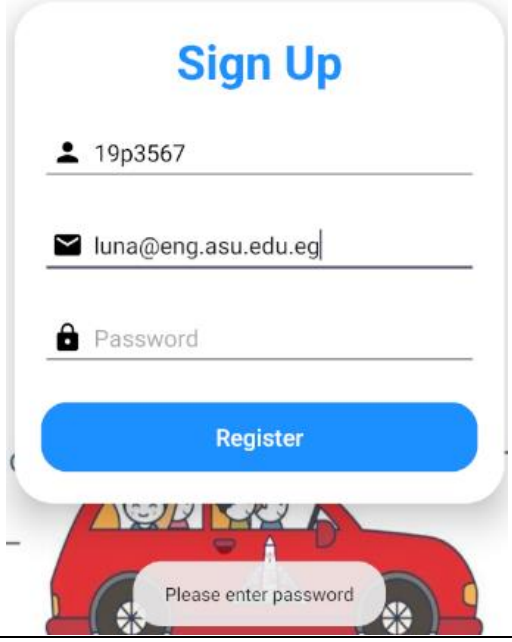

```
@Entity(tableName = "users")
public class User {
    @PrimaryKey
    @NonNull
    public String userId;
    @NonNull
    public String email;

    @NonNull
    public String name;

    public User(@NonNull String userId, @NonNull String email, @NonNull
String name) {
        this.userId = userId;
        this.email = email;
        this.name = name;
    }
```

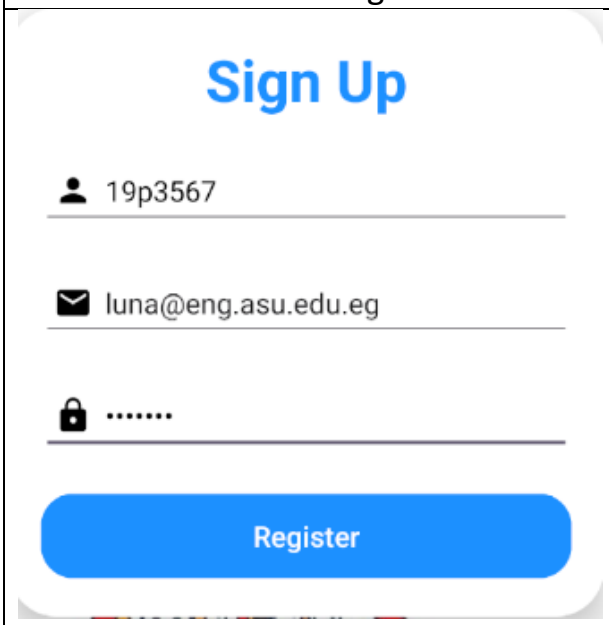
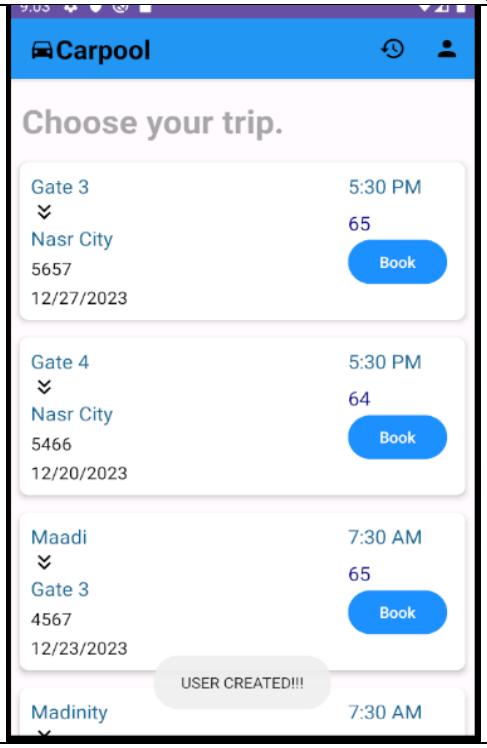
Test Cases

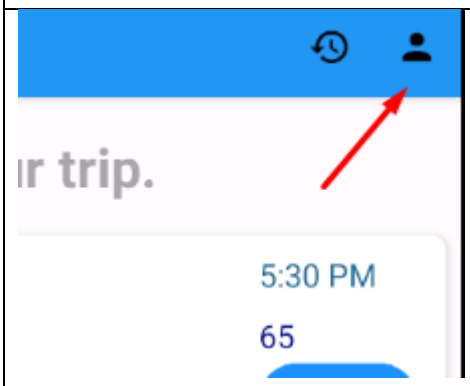
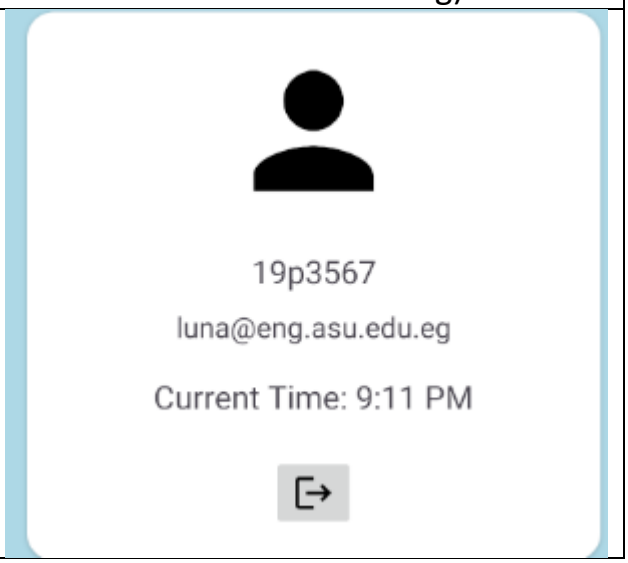
User side

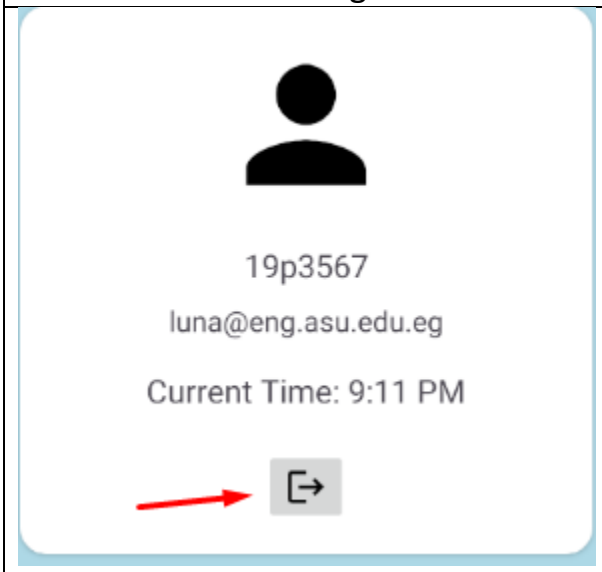
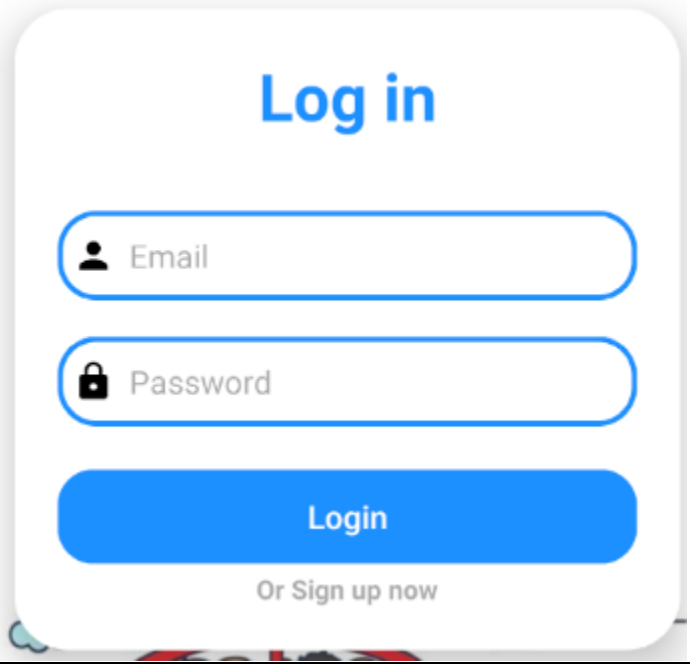
Scenario 1	
Click on Register	Entered invalid email domain
	

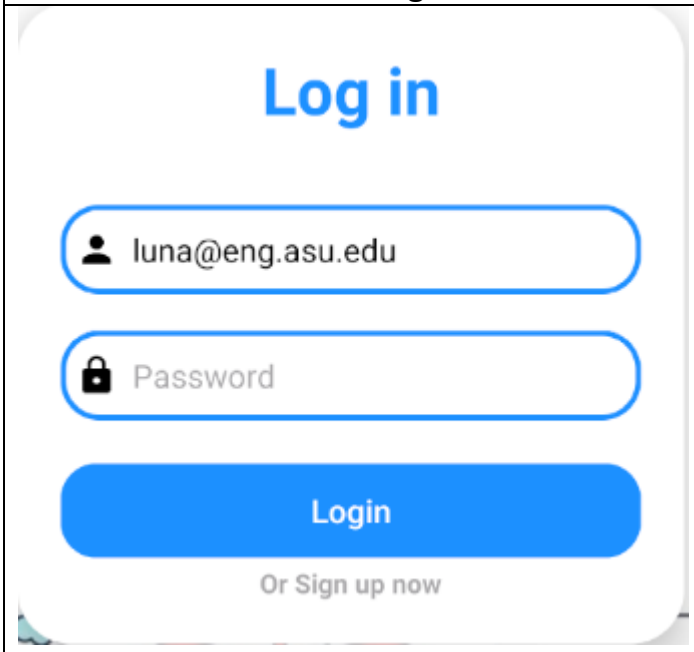
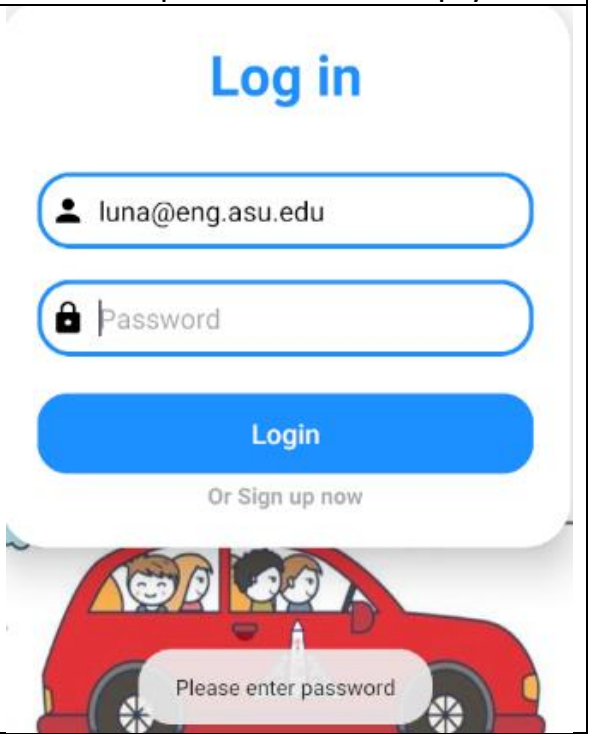
Scenario 2	
Click on Register	Left Id field empty
	

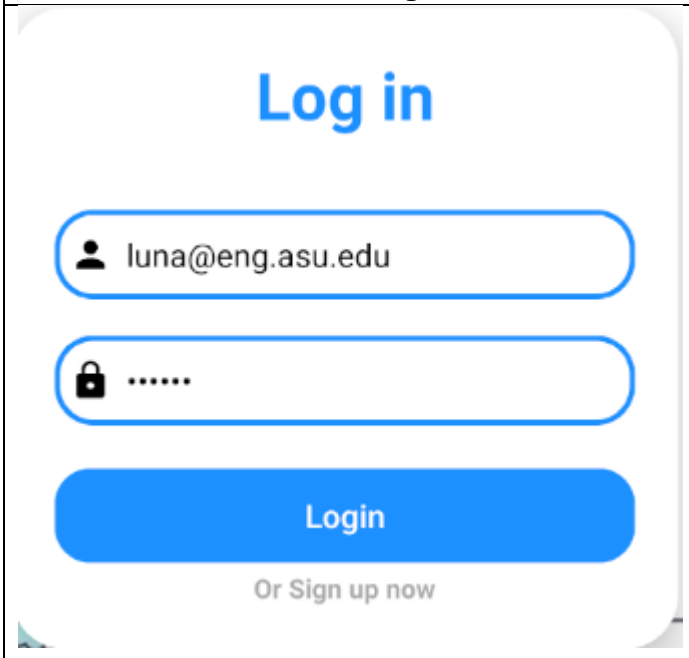
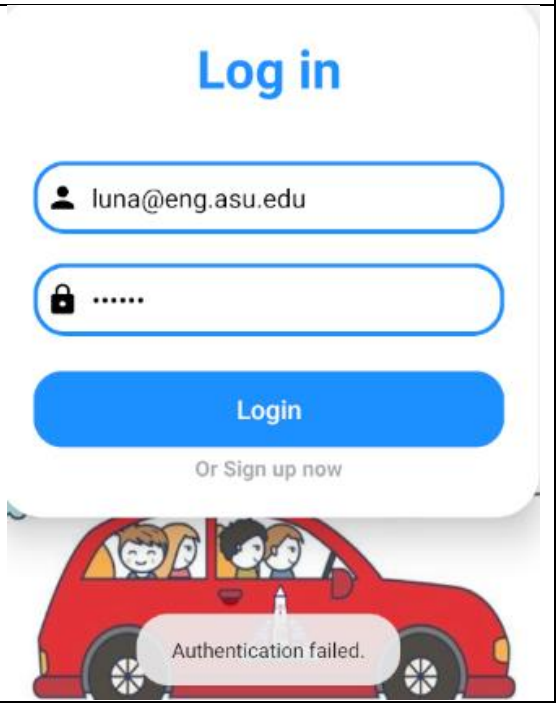
Scenario 3	
Click on Register	Left Password field empty
	

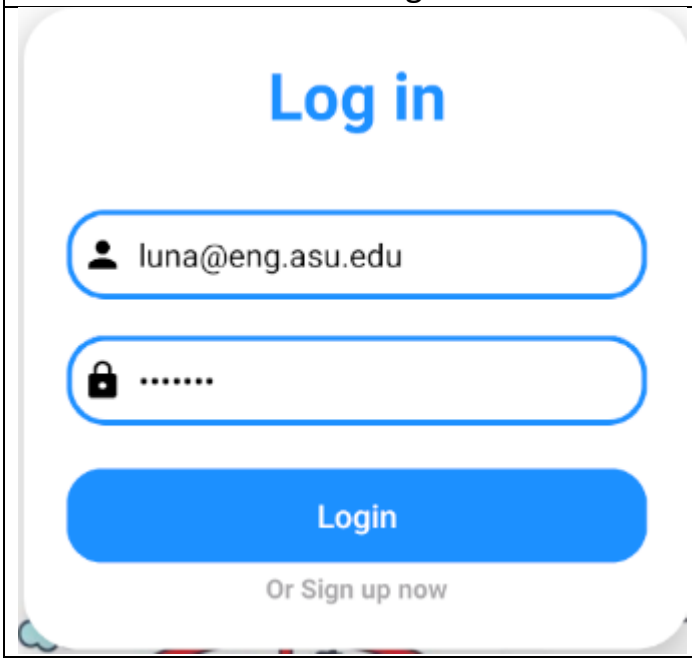
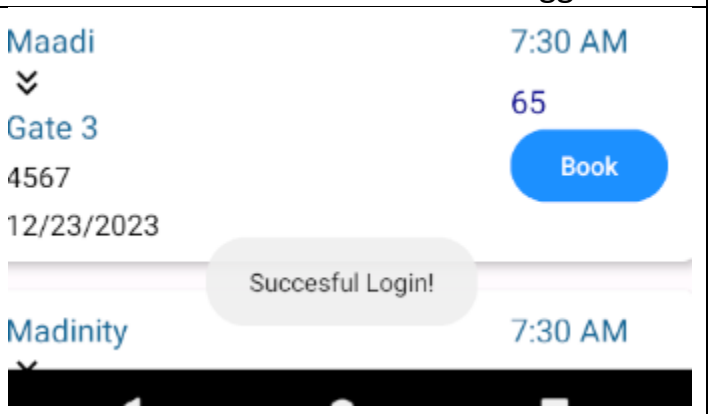
Scenario 4	
Click on Register	All fields are filled correctly
	

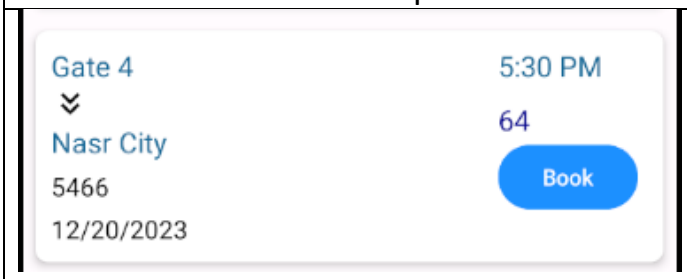
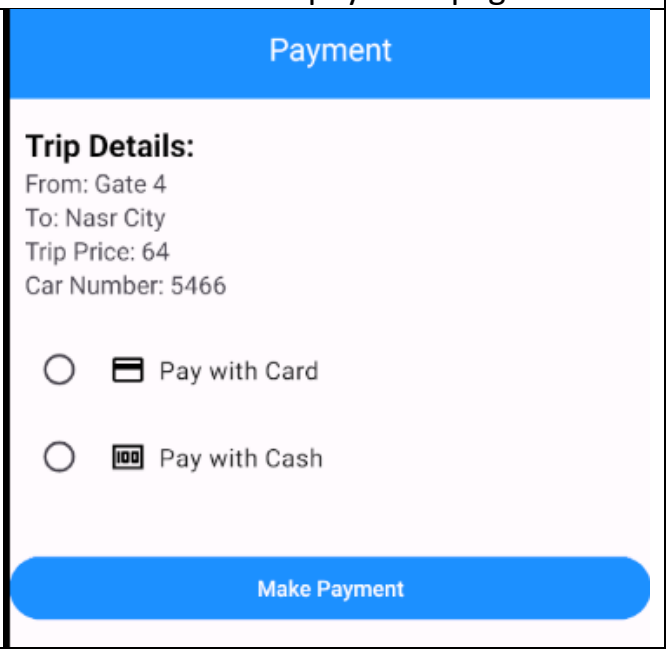
Scenario 5	
Click on Profile	Id and email of user is displayed (room database is working)
	

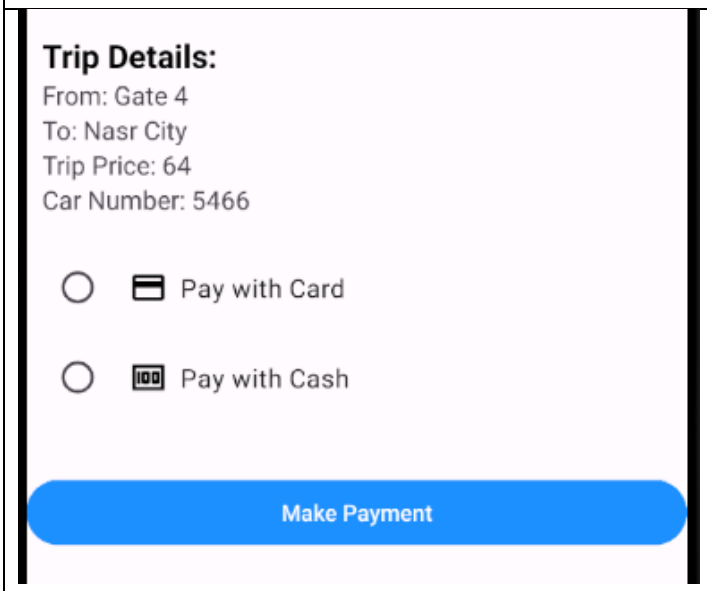
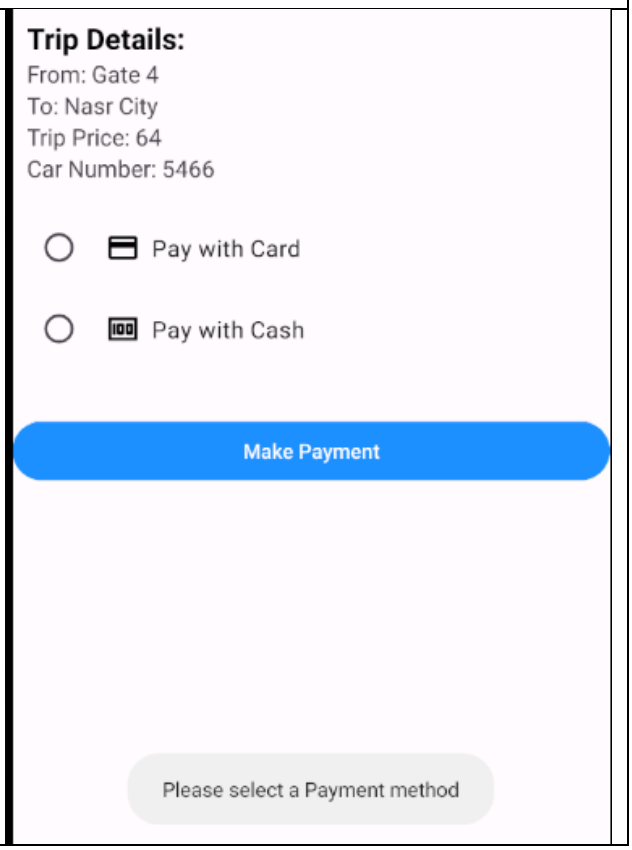
Scenario 6	
Click on Log out	Directed back to login page
	

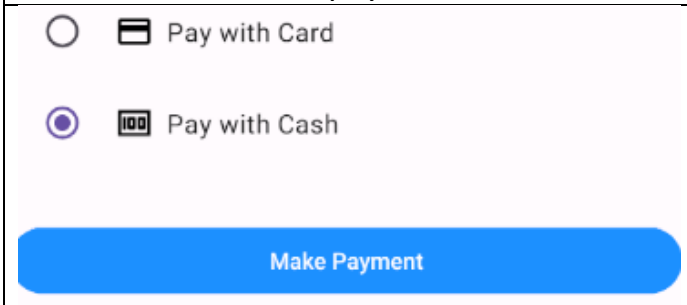
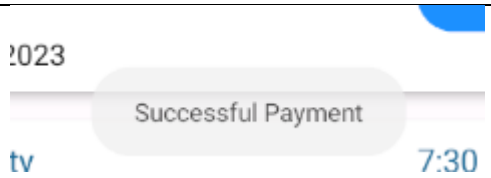
Scenario 7	
Click on Log in	Left password field empty
	

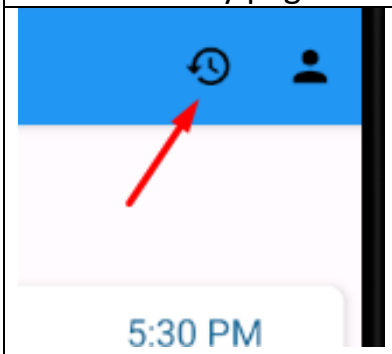
Scenario 8	
Click on Log in	Wrote an Incorrect Password
	

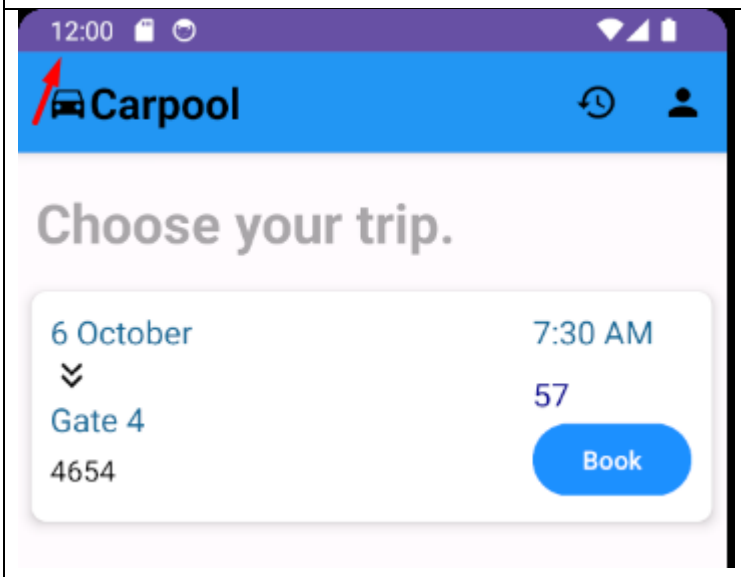
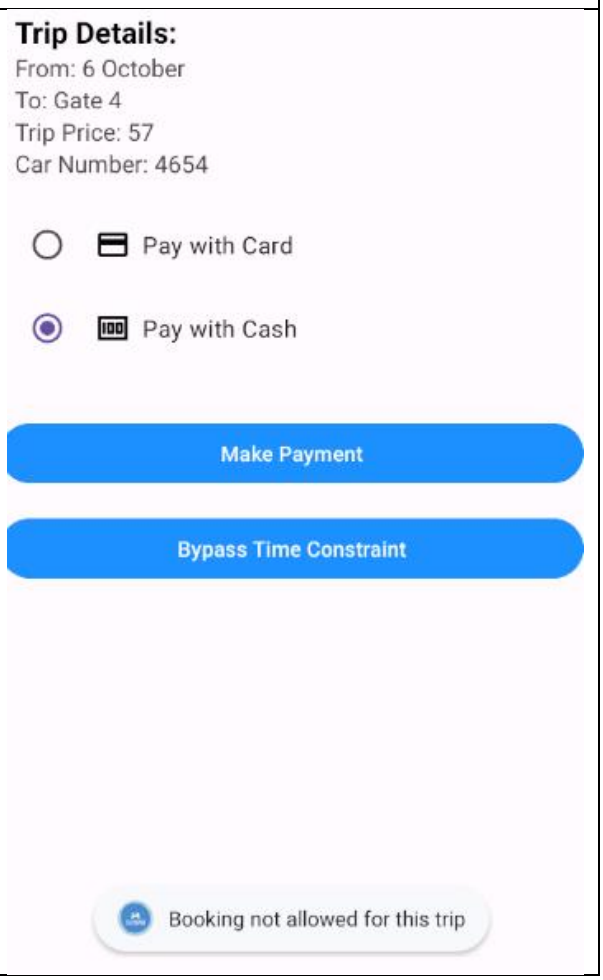
Scenario 9	
Click on Log in	Correct credentials written so logged in
	

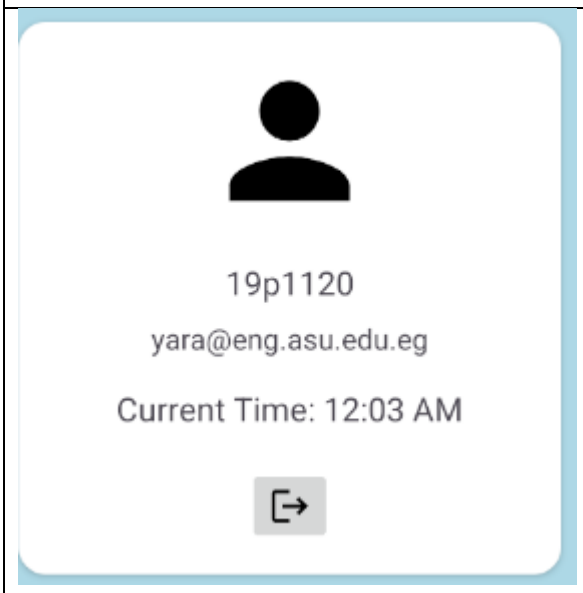
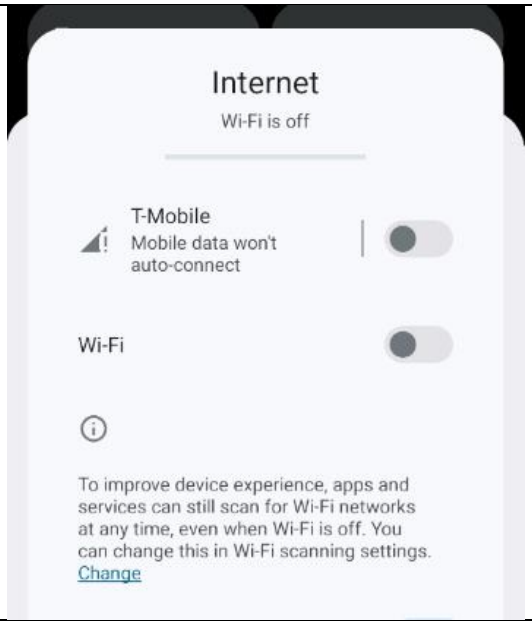
Scenario 10	
Book a trip	Directed to payment page
	

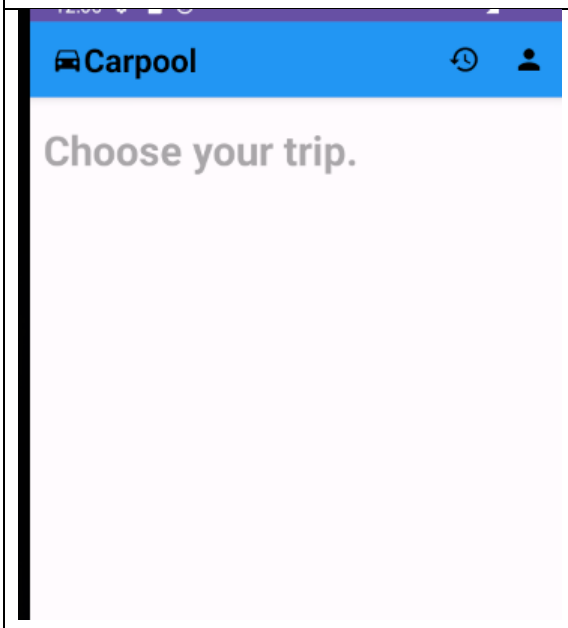
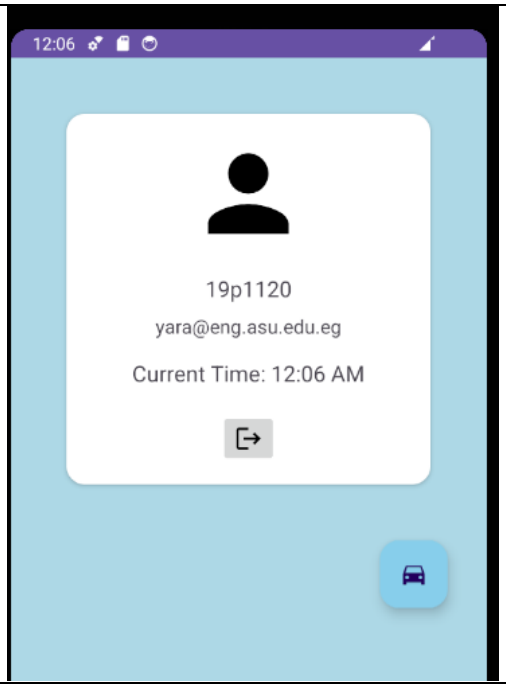
Scenario 11	
Click on make payment	Must select one of the payment methods
	

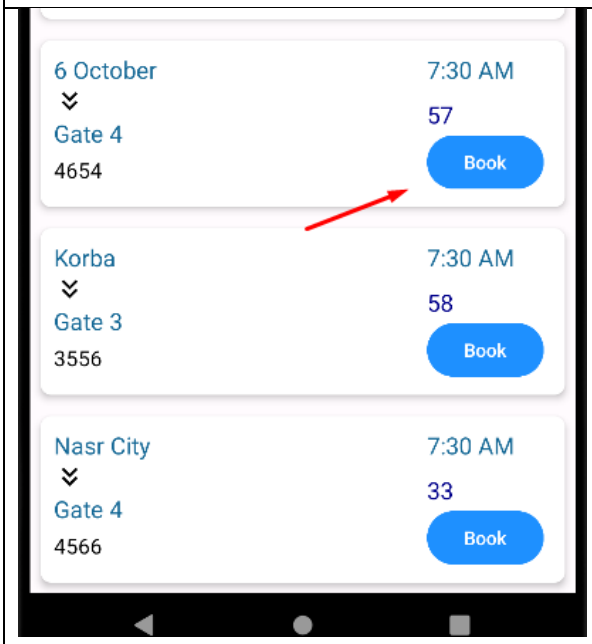
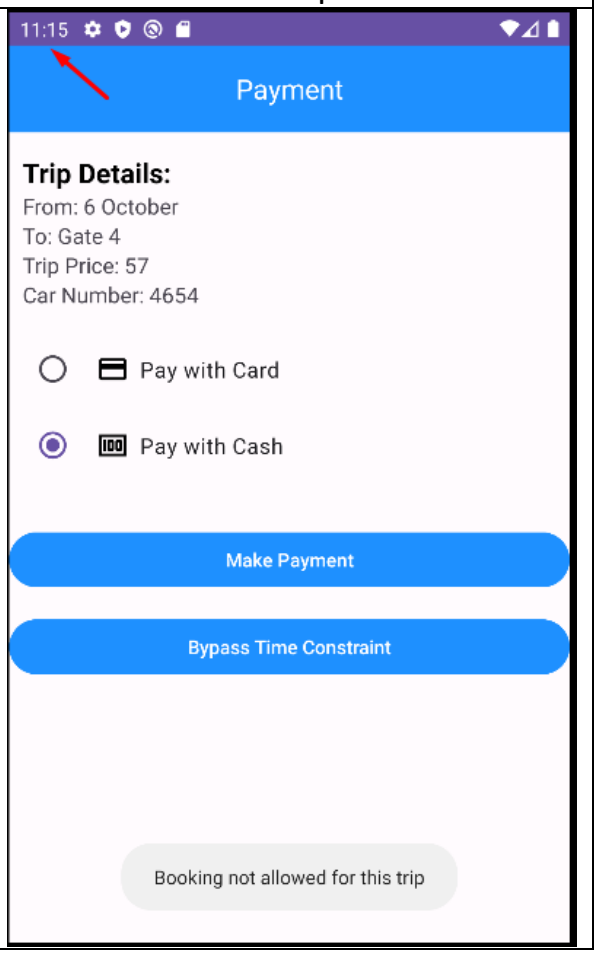
Scenario 12	
Choose a payment method then click on make payment	Payment done successfully and order is created
	

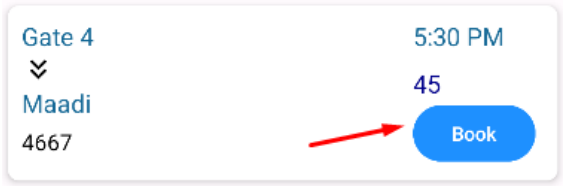
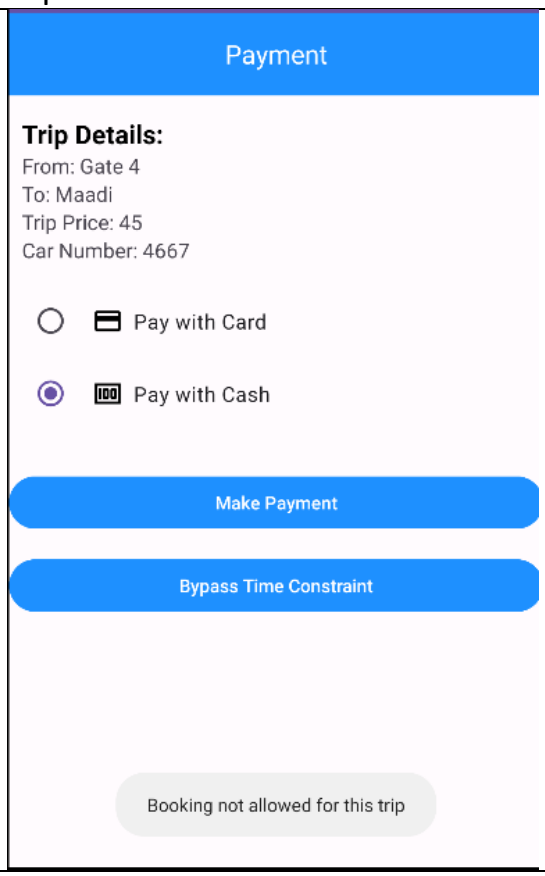
Scenario 13	
View history page to view trips	View the trip created
	

Scenario 14	
Go back to trips and book a trip after time constraint	Can't book the trip by normal "make payment" button as trip is at 7:30 AM but time is 12:00AM
	

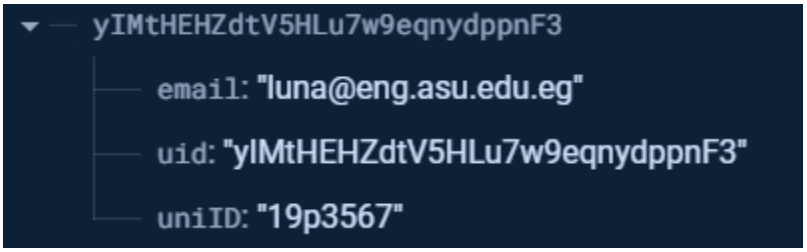
Scenario 15	
View profile info that's fetched from firebase	Now disconnect internet from phone
	

Scenario 16	
No trips displayed because no internet	Profile data is shown offline fetched from Room
	

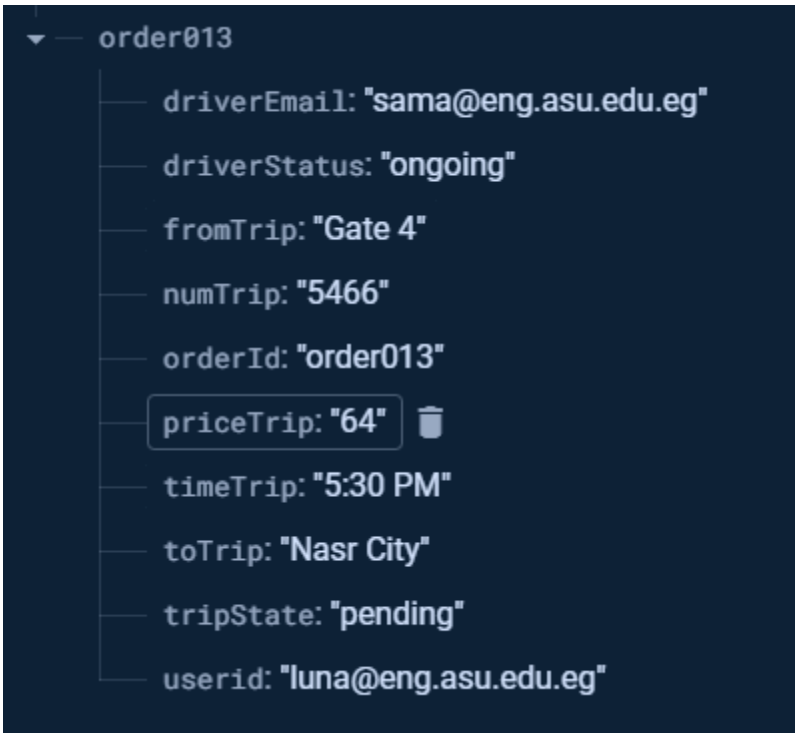
Scenario 17	
Book a 7:30 AM trip at 11:15 PM	Booking not allowed after 10 PM for 7:30 trip ☺
	

<table>
<tr><th colspan="2">Scenario 18</th></tr>
<tr><td>Book a 5:30 PM trip at 2:15 PM</td><td>Booking not allowed after 1 PM (same day of the trip) for 5:30 PM trip ☺</td></tr>
<tr><td> 
</td><td> 
</td></tr>
</table>

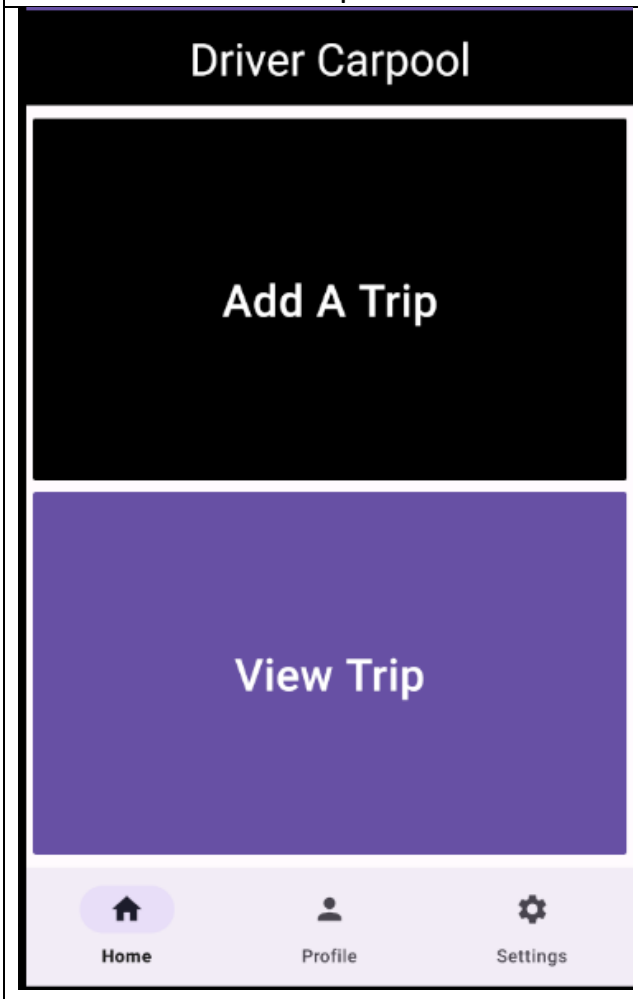
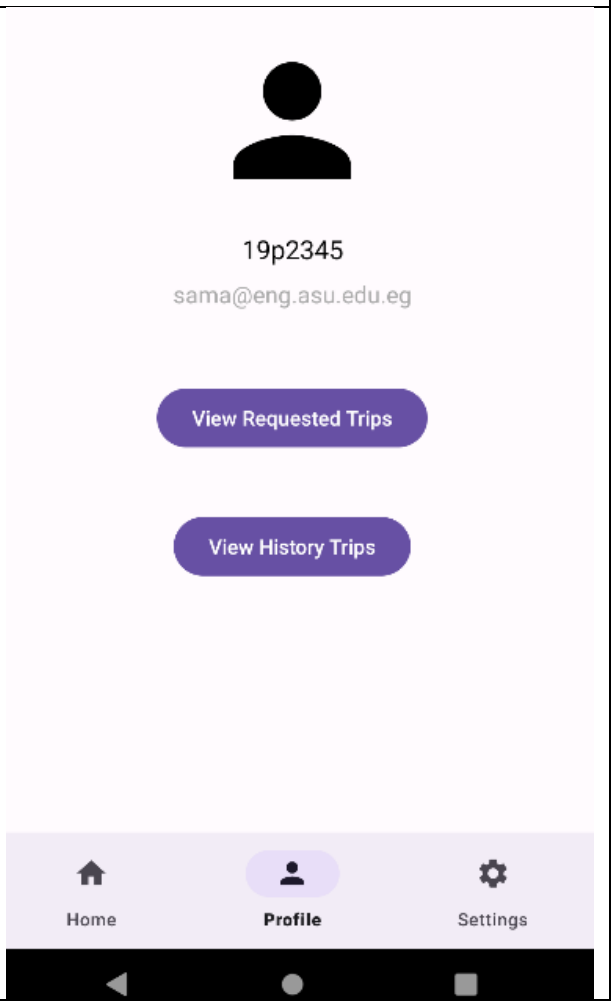
Firebase updates

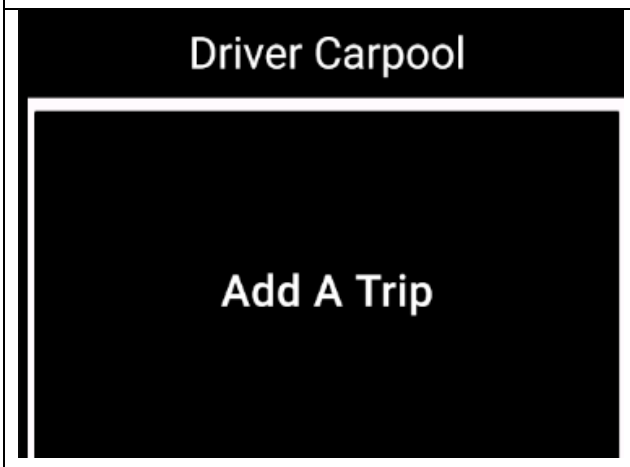
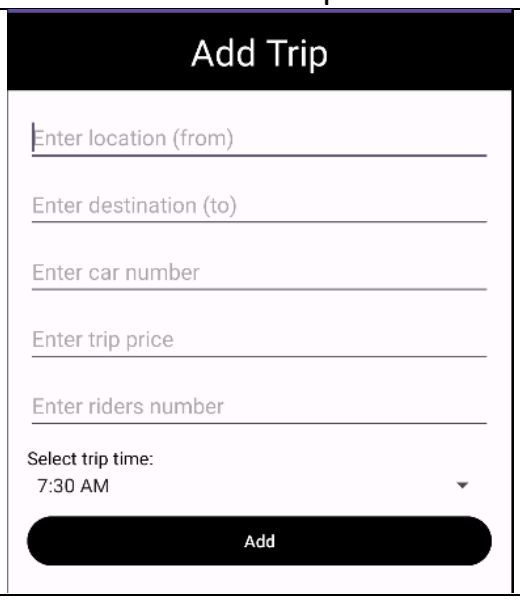


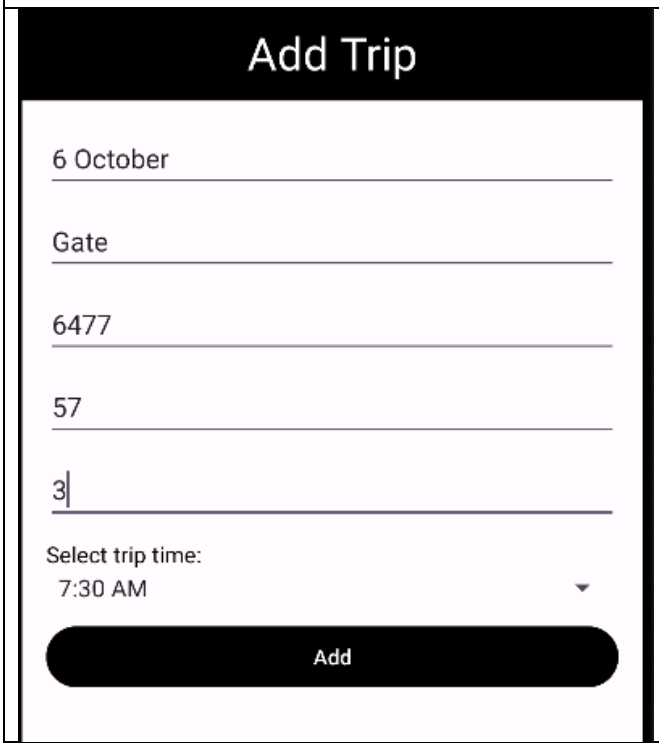
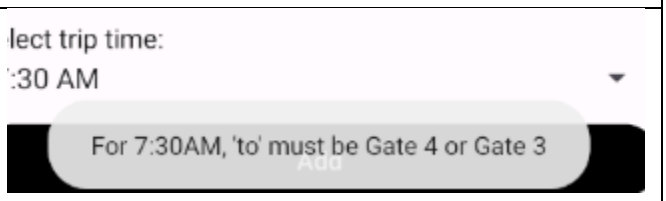

trip007
driverEmail: "sama@eng.asu.edu.eg"
driverStatus: "ongoing"
fromTrip: "Gate 4"
maxRider: "1"
numTrip: "5466"
orders
0: "order011"
1: "order013"
priceTrip: "64"
timeTrip: "5:30 PM"
toTrip: "Nasr City"
tripDate: "12/20/2023"
tripID: "trip007"

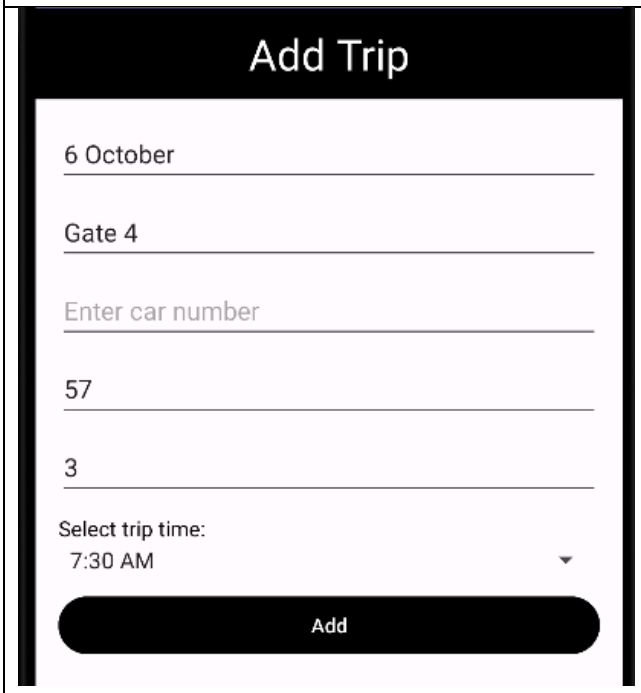
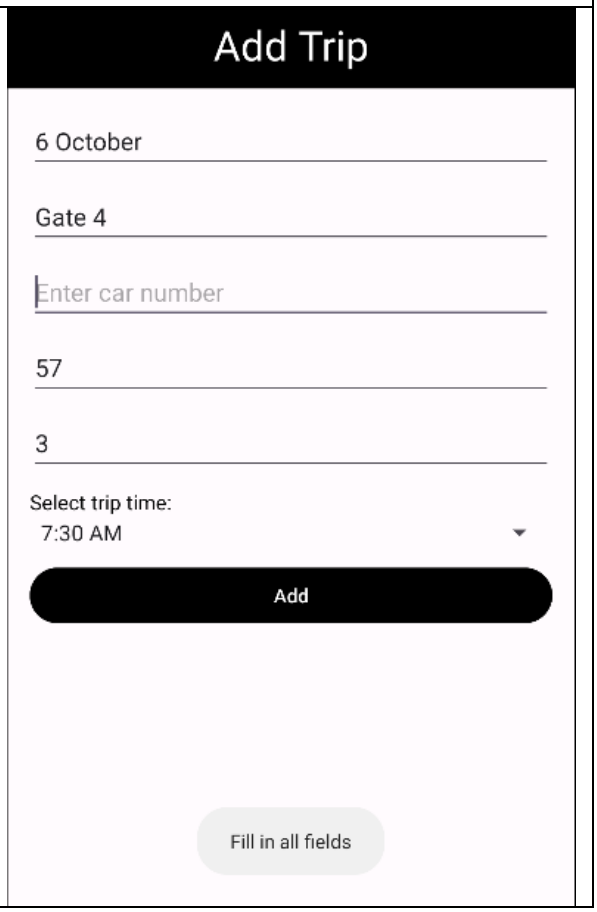

order013
driverEmail: "sama@eng.asu.edu.eg"
driverStatus: "ongoing"
fromTrip: "Gate 4"
numTrip: "5466"
orderId: "order013"
priceTrip: "64"
timeTrip: "5:30 PM"
toTrip: "Nasr City"
tripState: "pending"
userid: "luna@eng.asu.edu.eg"

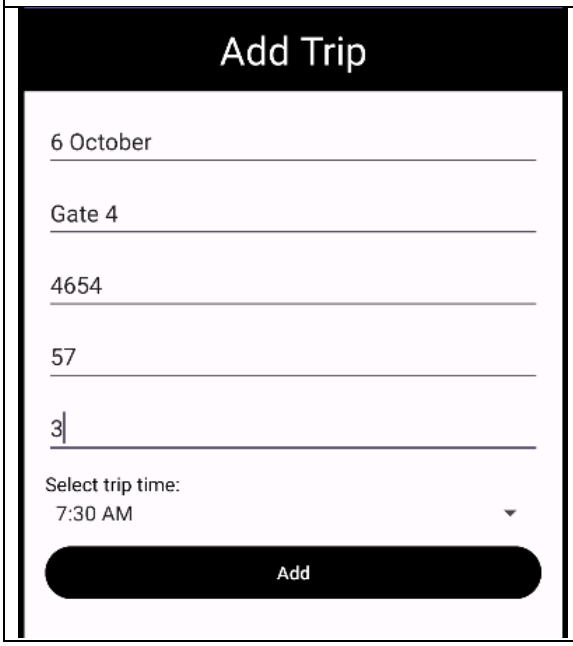
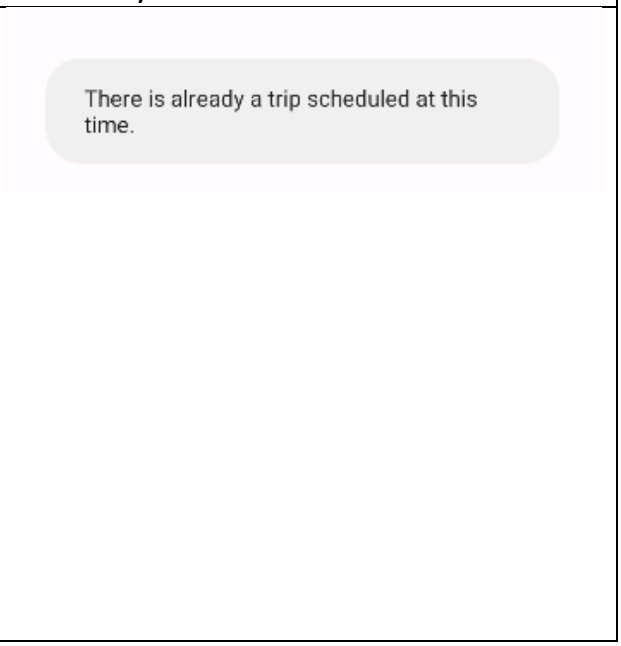
Driver side

Scenario 1	
Click on profile	Driver details is shown
	

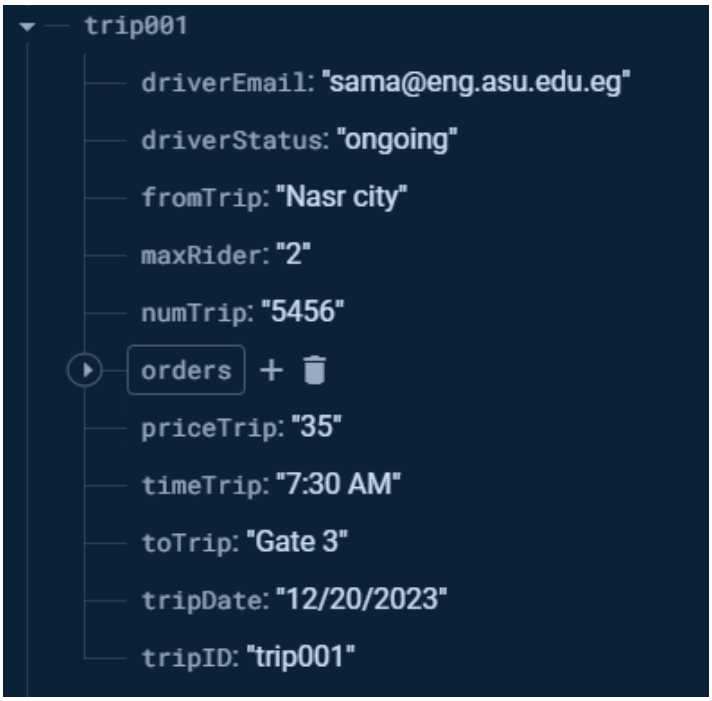
Scenario 2	
Click on Add trip	Page opens with Text fields to add the trips
	

Scenario 3	
Add trip but specify wrong destination	Trip can't be added because of incorrect destination
	

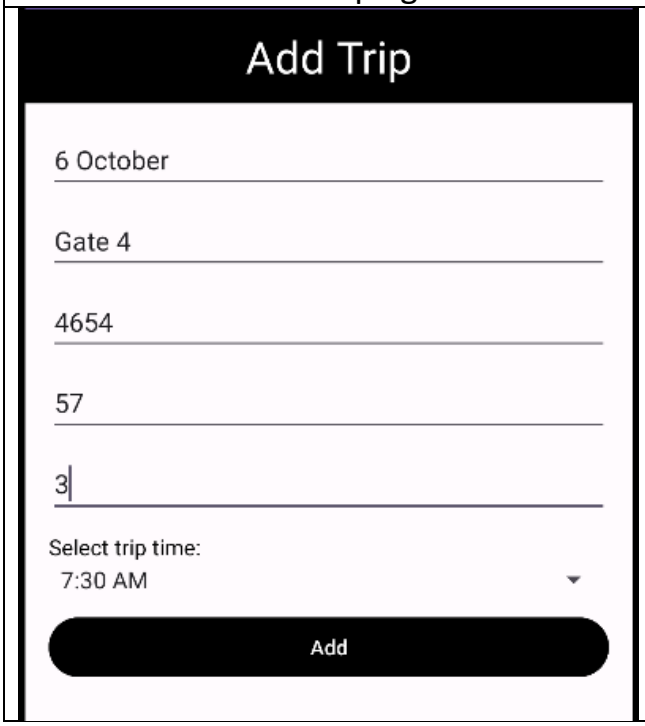
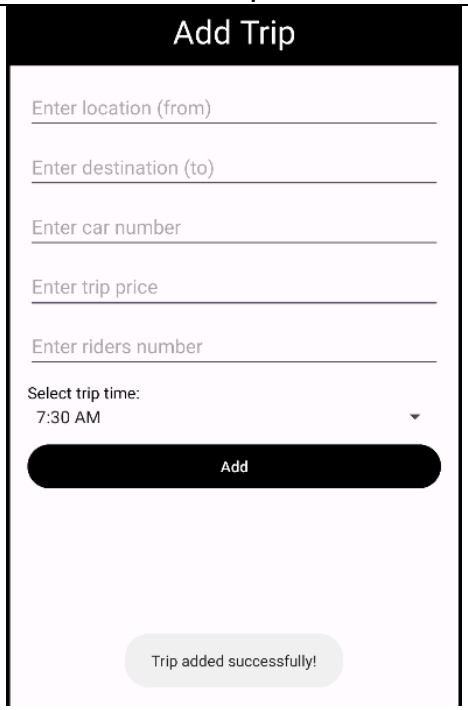
Scenario 4	
Leave any text field empty	Must fill that text field, trip not added.
	

Senario 5	
Add trip	Trip with same time and on same day is already created
	

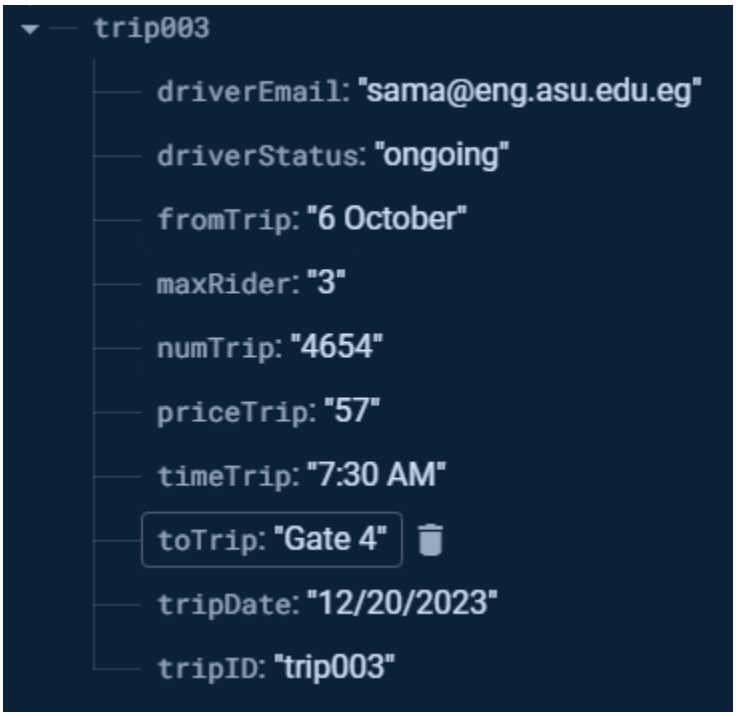
There’s a trip from that trip exiting on created on same day and time from that driver :/

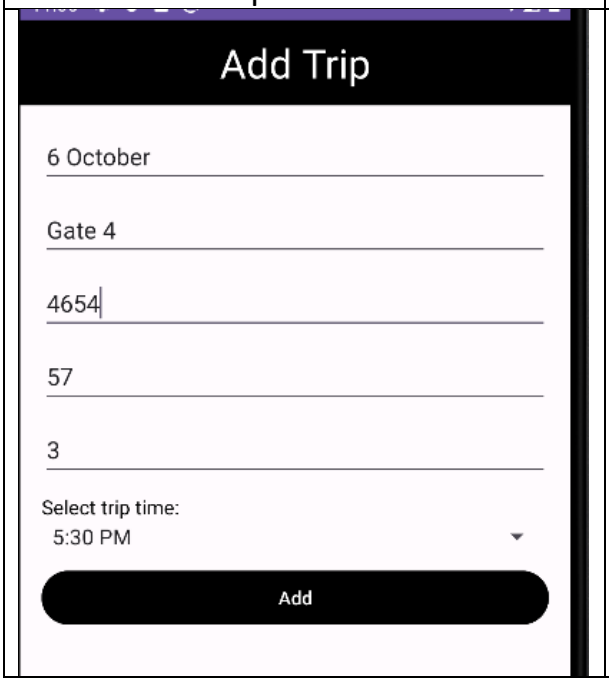
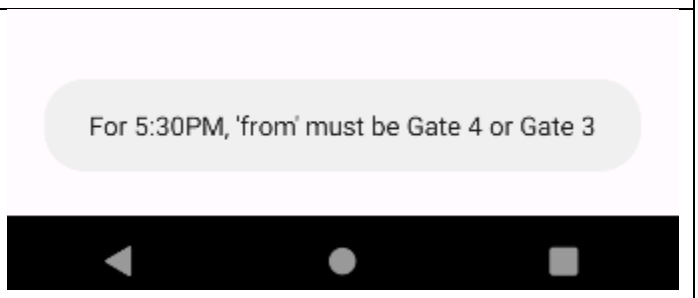


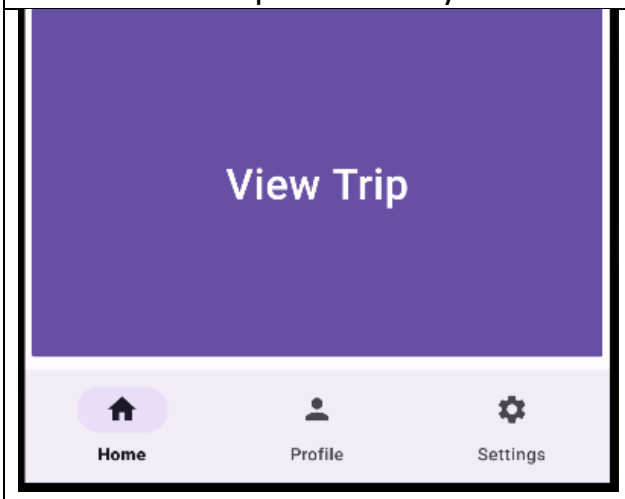
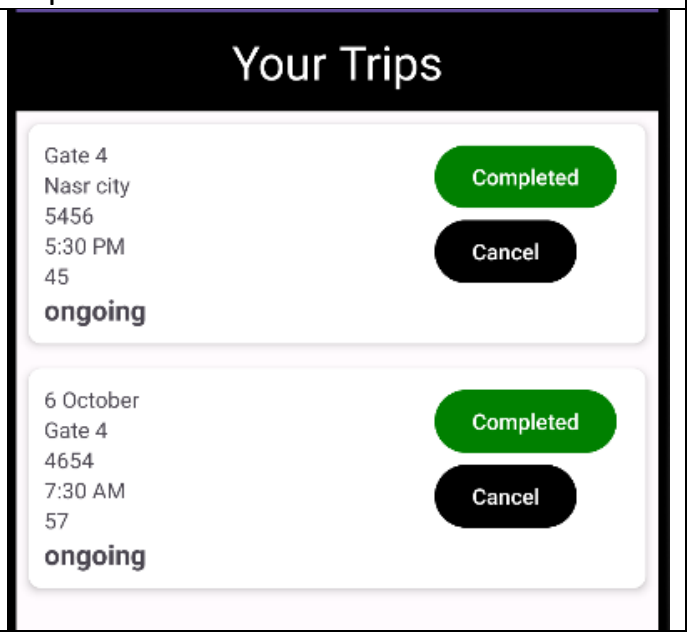
Let’s delete it and retry adding for testing purposes ☺

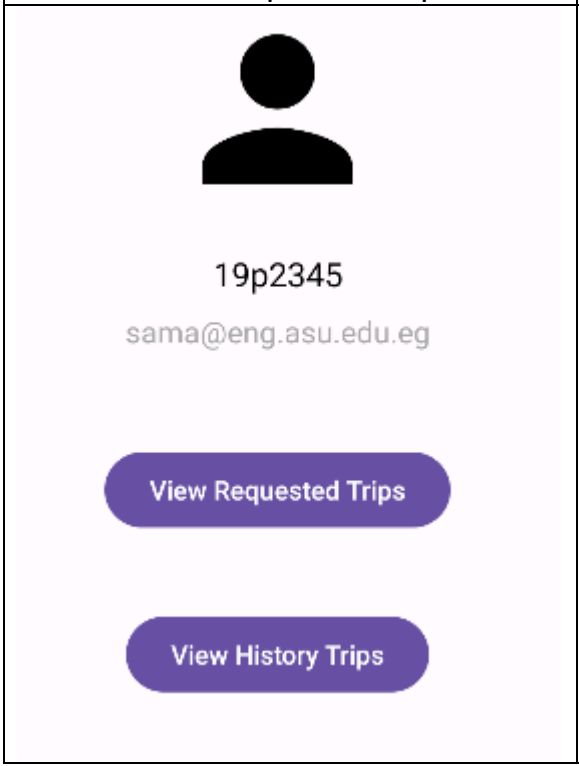
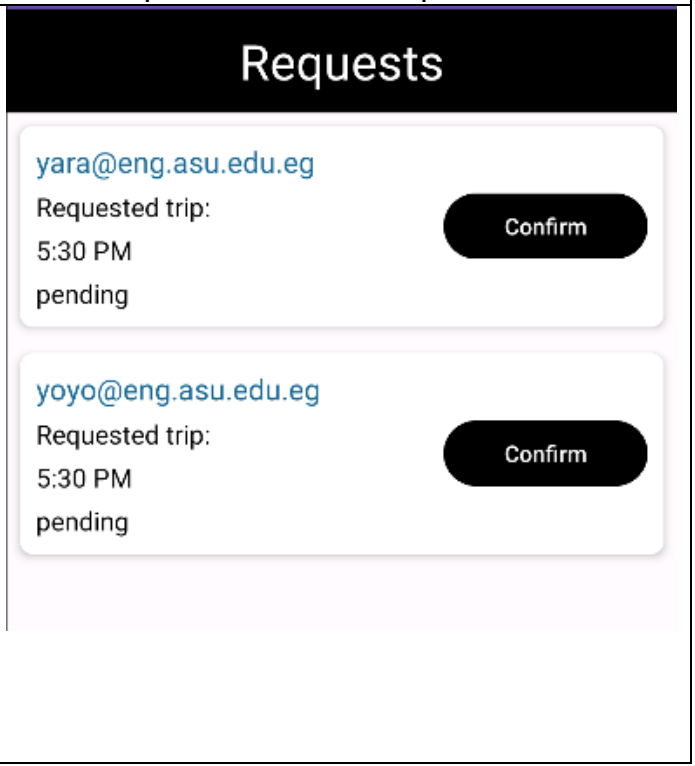
Scenario 6	
Add the trip again	Trip added
	

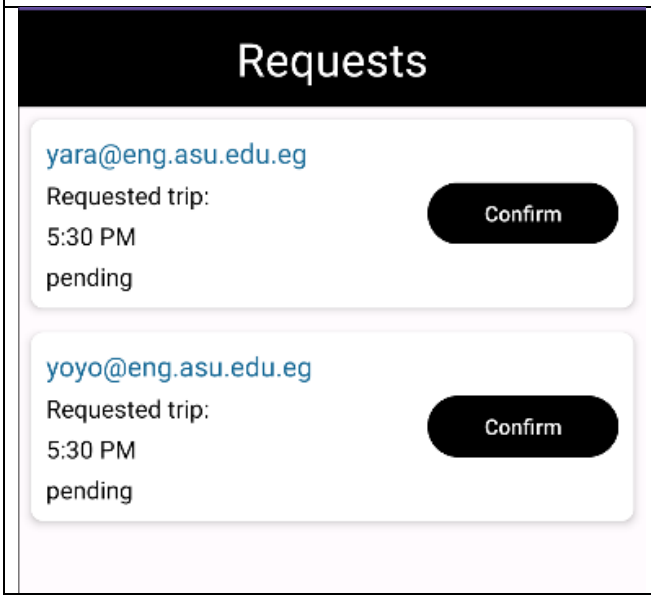
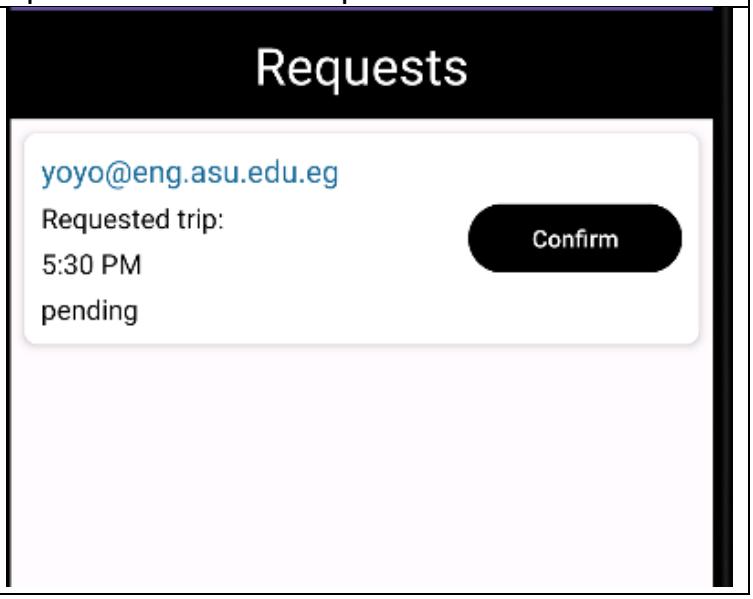
Updated trip on firebase

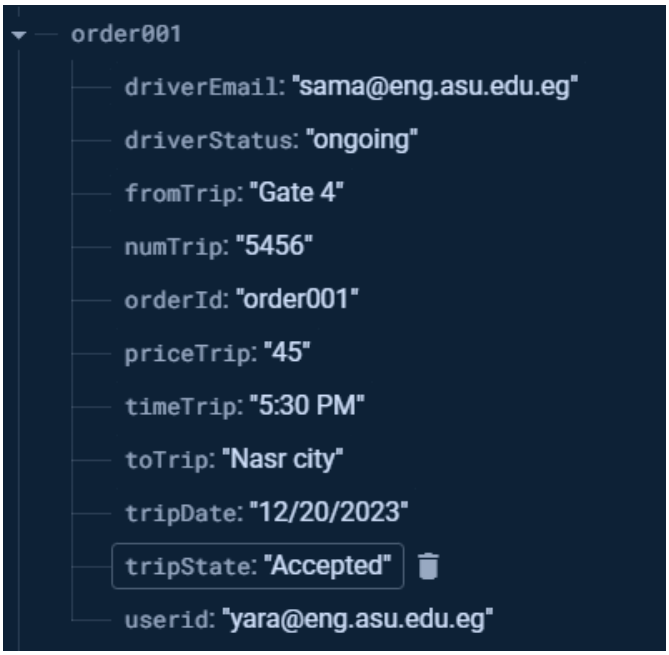


Scenario 7	
Add another trip for 5:30	Takeoff must be Gate 4 or Gate 3
	

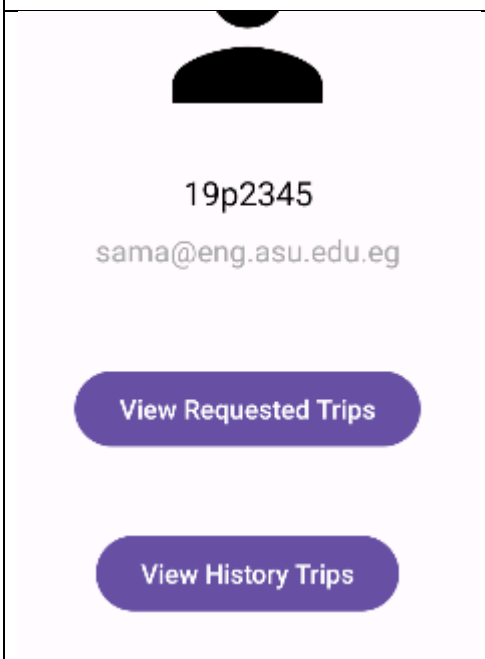
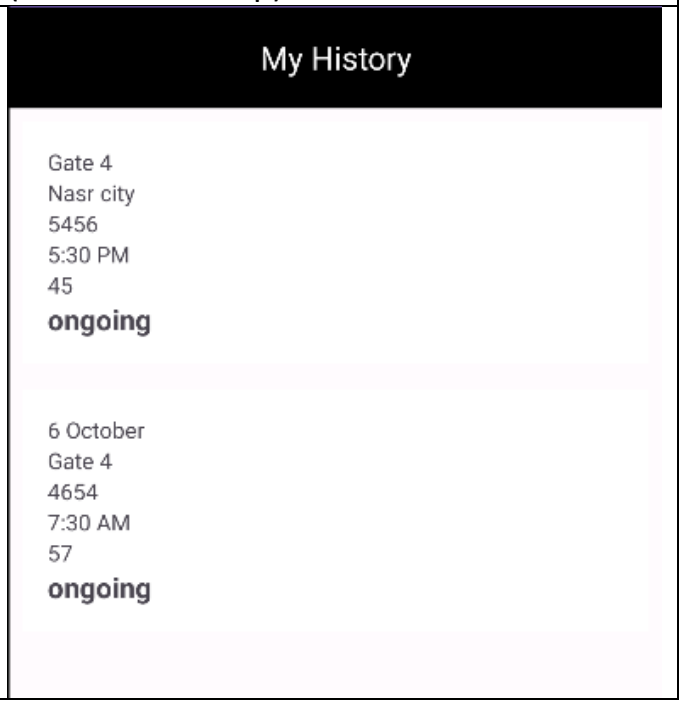
Scenario 8	
Click to View trips created by driver	Trips reviewed
	

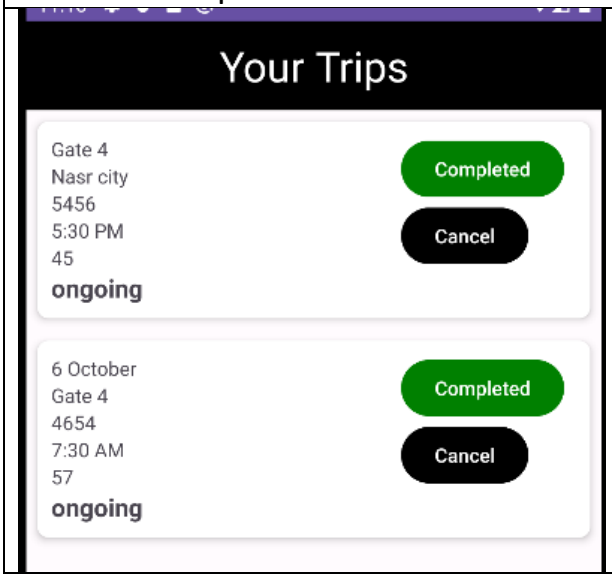
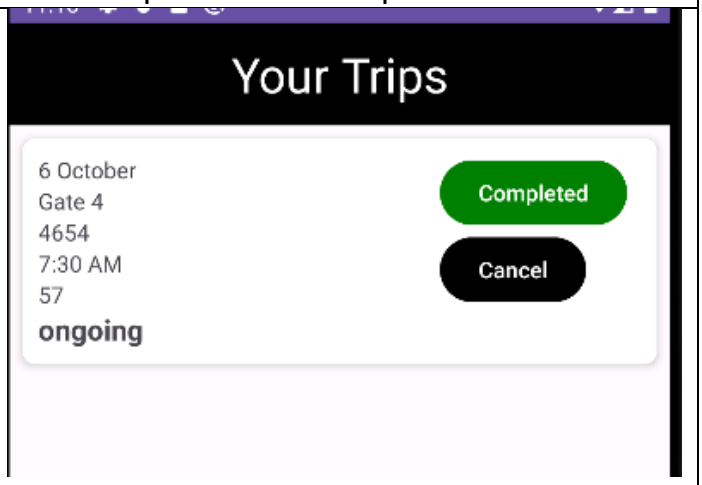
Scenario 9	
Click on view requested trips	Users requests on driver trips are shown
	

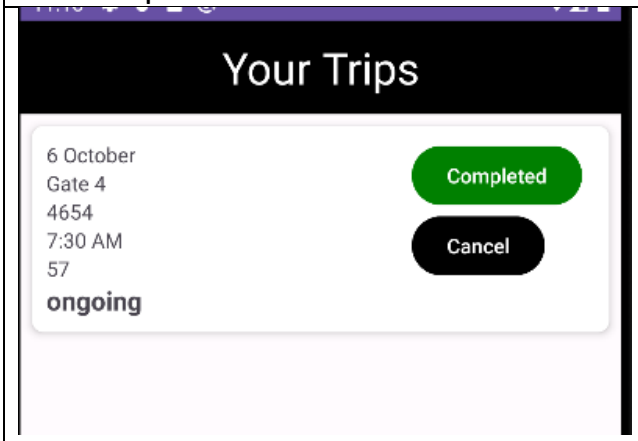
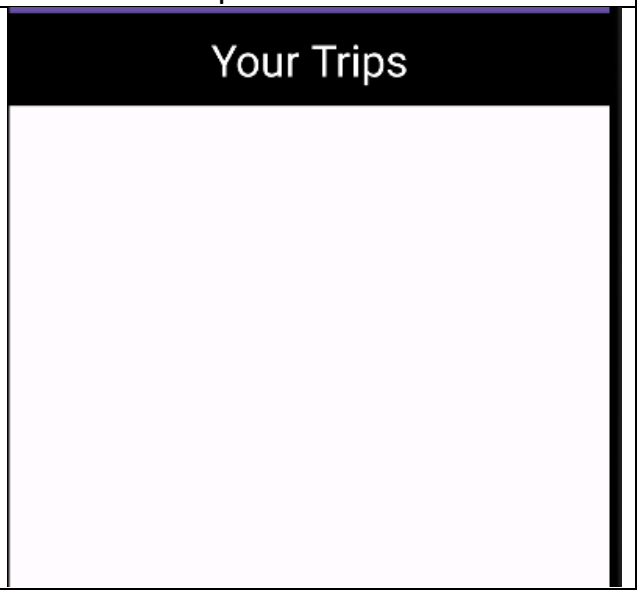
Scenario 10	
Confirm one of the requests	Items disappears from view and trip state is updated to user “accepted”
	

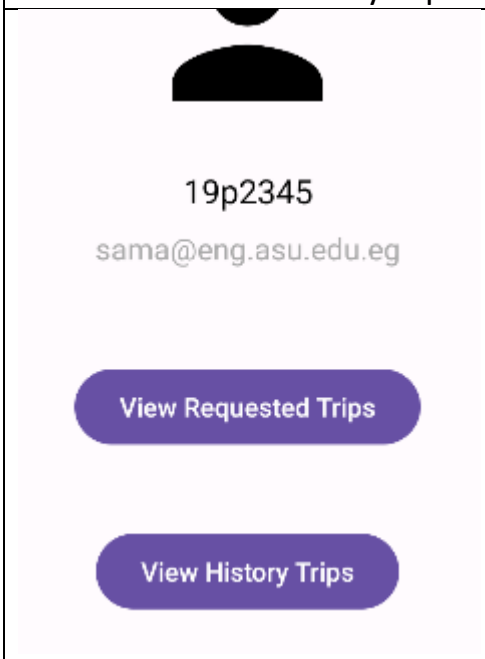
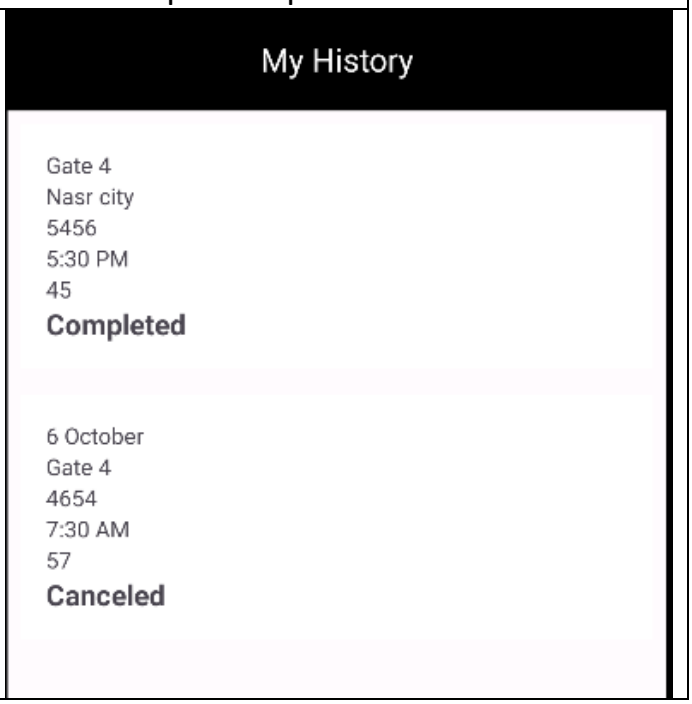


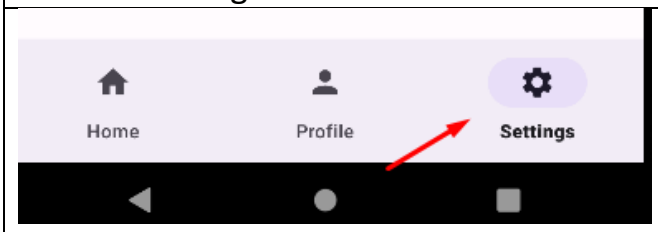
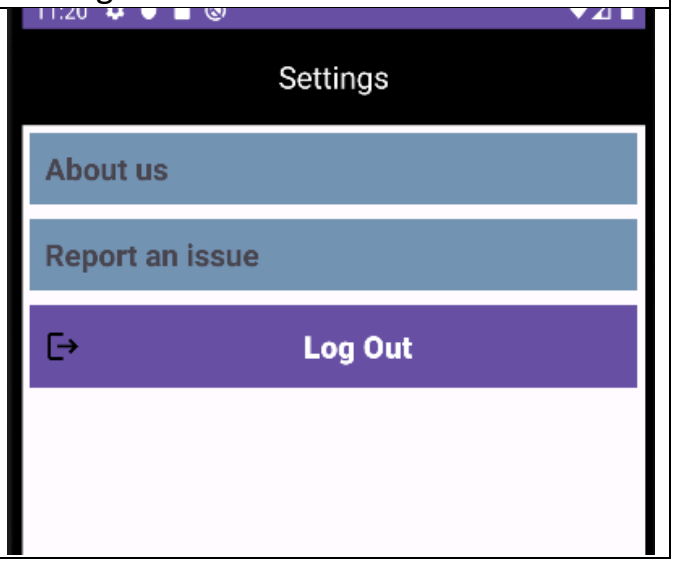
Trip state is updated in database which will be visible to the user that their request was accepted (confirmed).

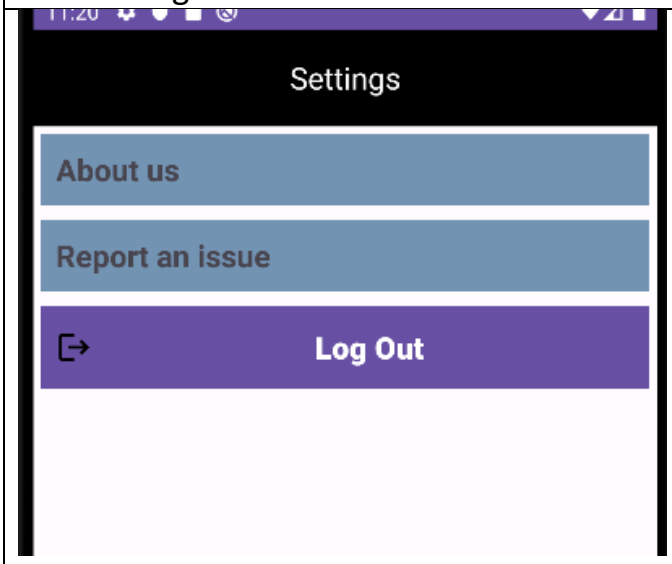
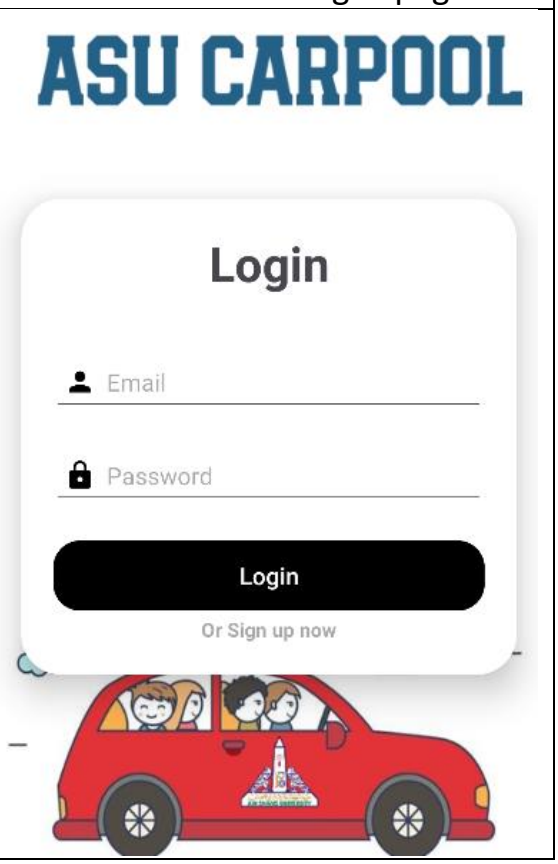
Scenario 11	
Click on view History Trips	History trips created by driver are shown (with state of trip)
	

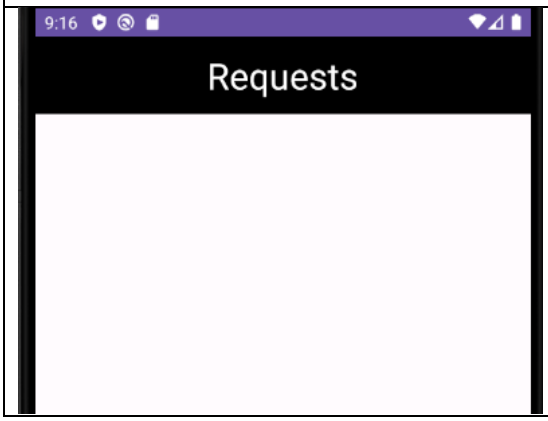
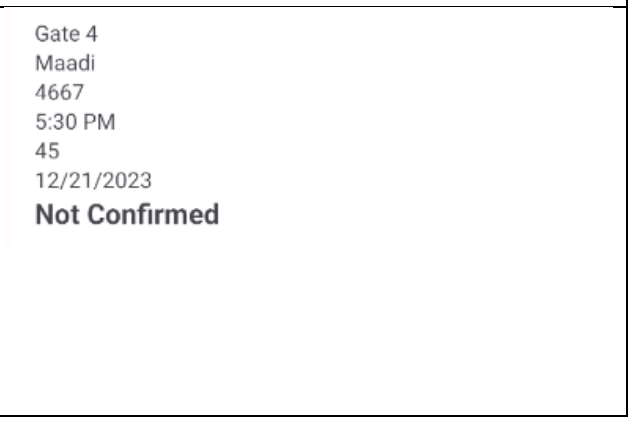
Scenario 12	
Click on completed to update state of one of the trips	Trip item disappears, driver state and user state is updated to “completed”
	

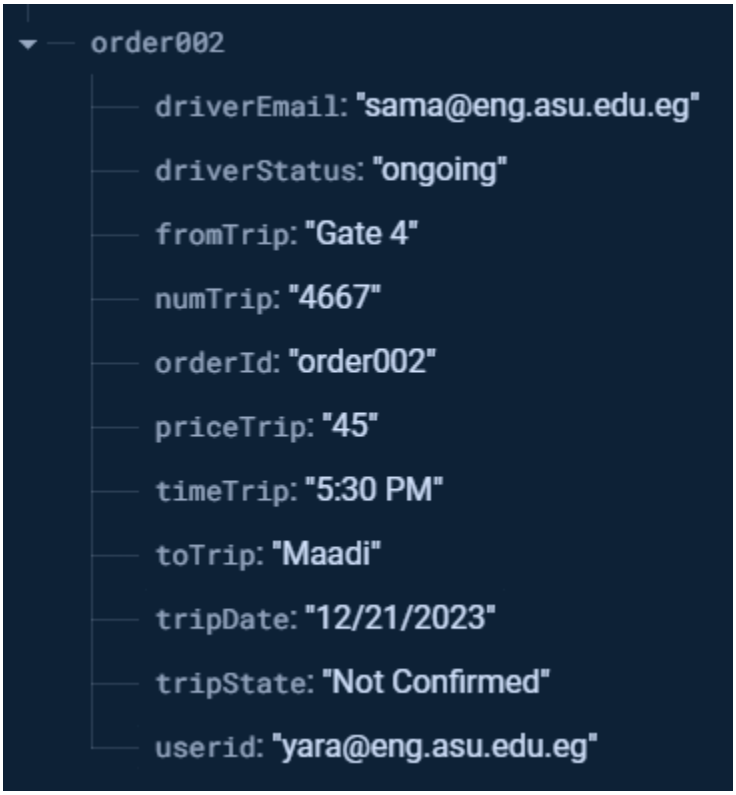
Scenario 13	
Click on cancel to update state of the other trip	Trip item disappears, driver state and user state is updated to “canceled”
	

Scenario 14	
Now view Driver history trips	State of trips are updated ☺
	

Scenario 15	
Click on setting	Setting are shown
	

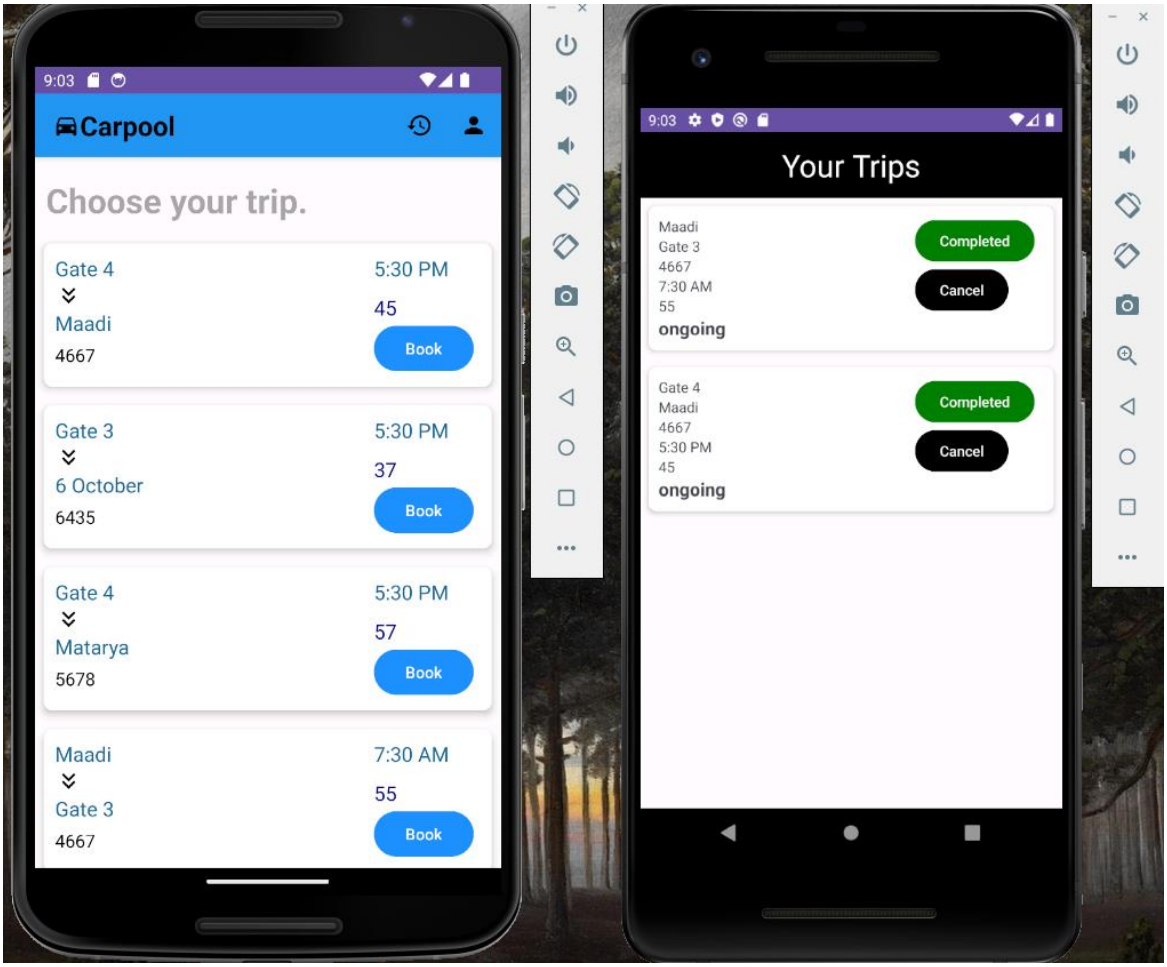
Scenario 16	
Click on log out	Driver is directed to log in page
	

Scenario 19	
Time constraint for requests (request) won't be shown	Will be updated at user side as not confirmed after the time limit
	

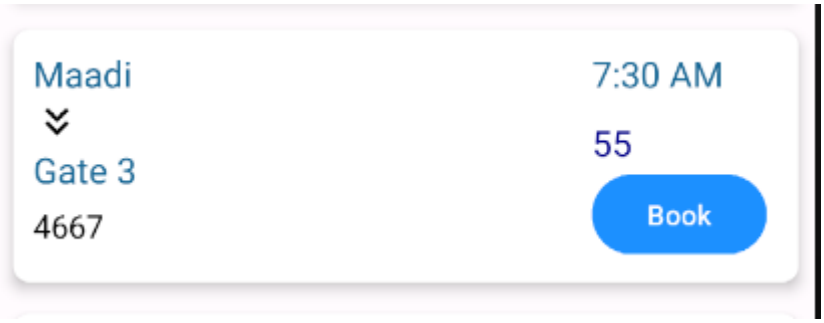

order002
driverEmail: "sama@eng.asu.edu.eg"
driverStatus: "ongoing"
fromTrip: "Gate 4"
numTrip: "4667"
orderId: "order002"
priceTrip: "45"
timeTrip: "5:30 PM"
toTrip: "Maadi"
tripDate: "12/21/2023"
tripState: "Not Confirmed"
userid: "yara@eng.asu.edu.eg"

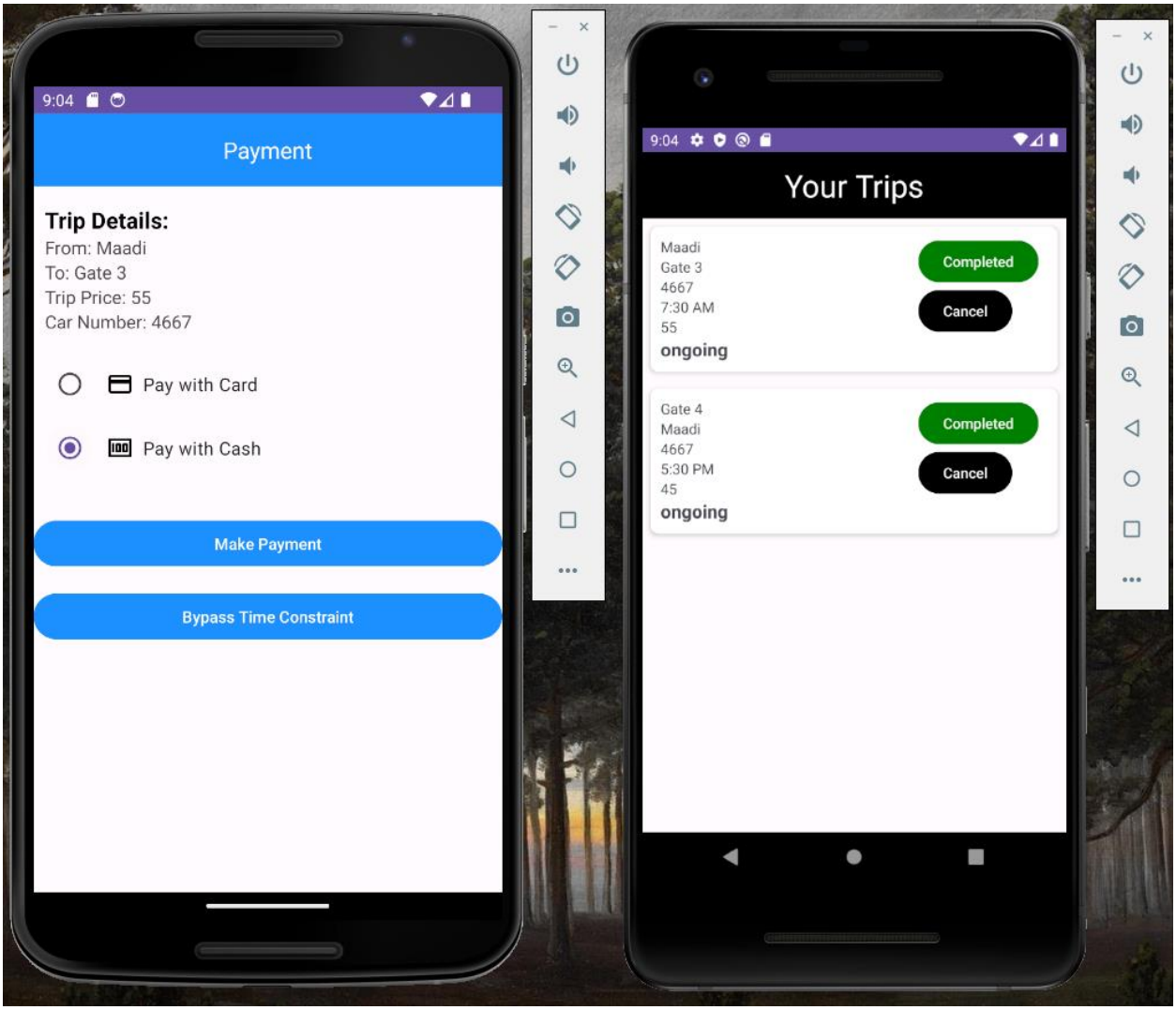
Interaction between both

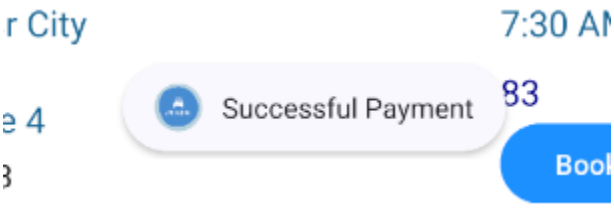
View available trips user side and trips made by a certain driver in driver side



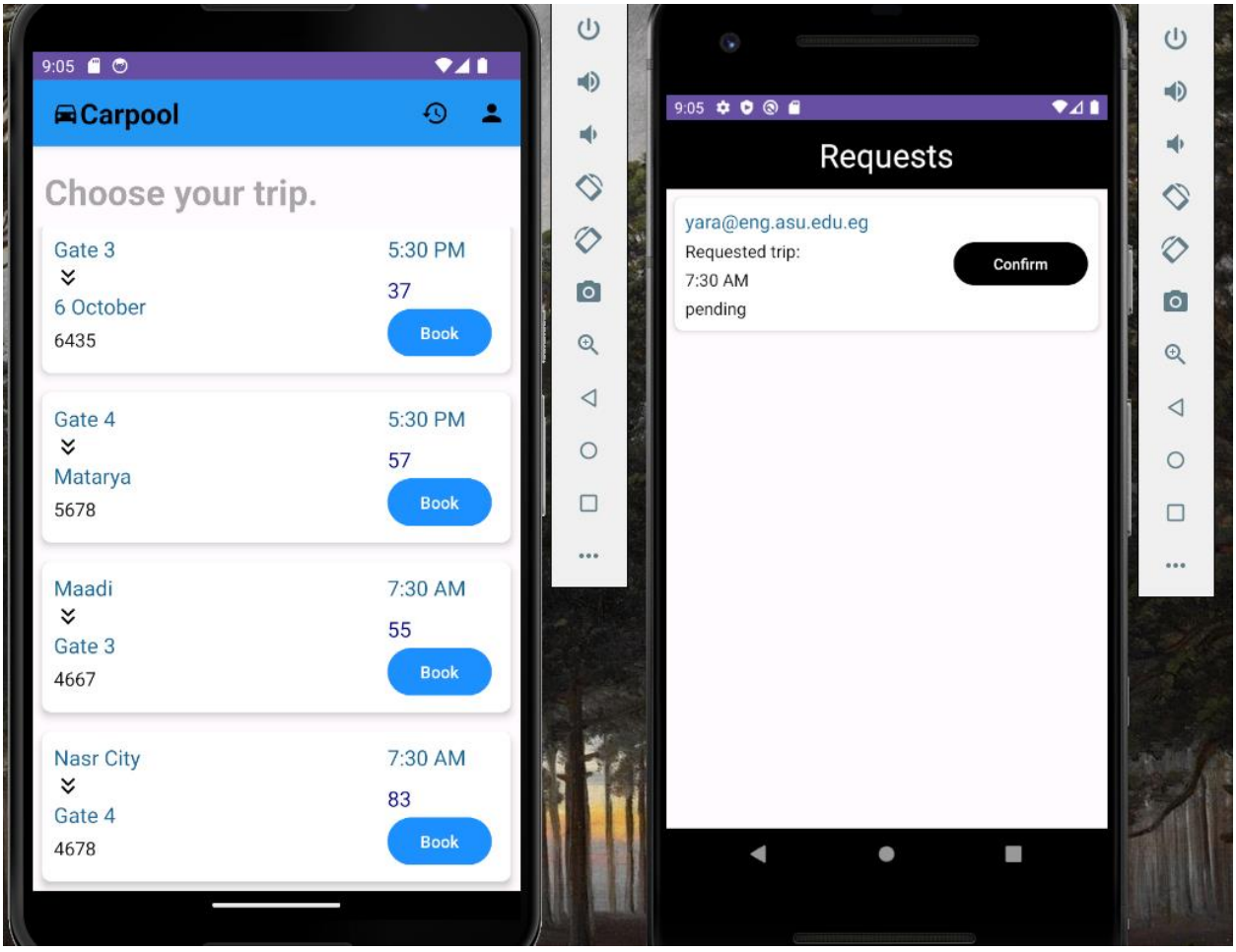
Book one of them



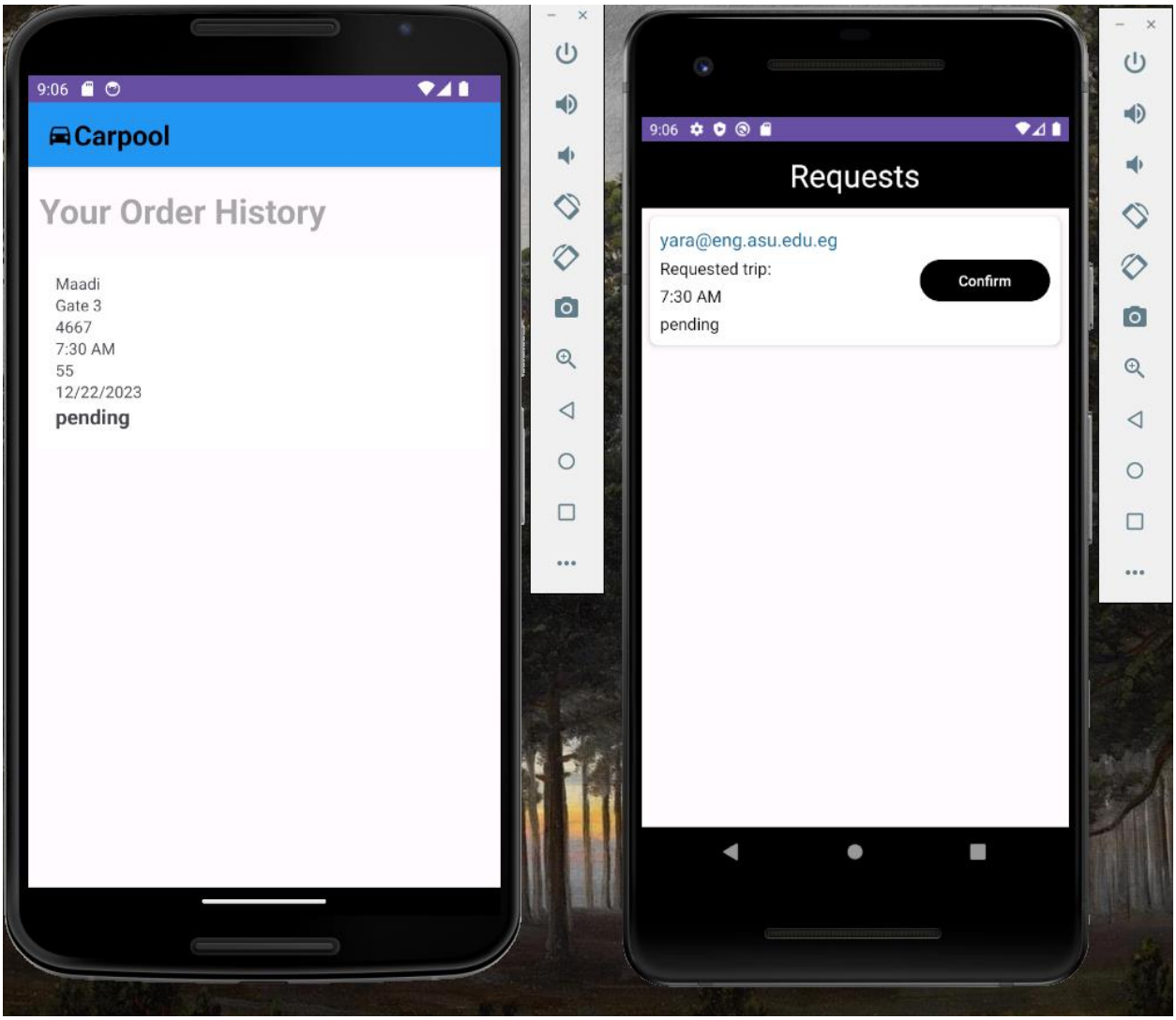




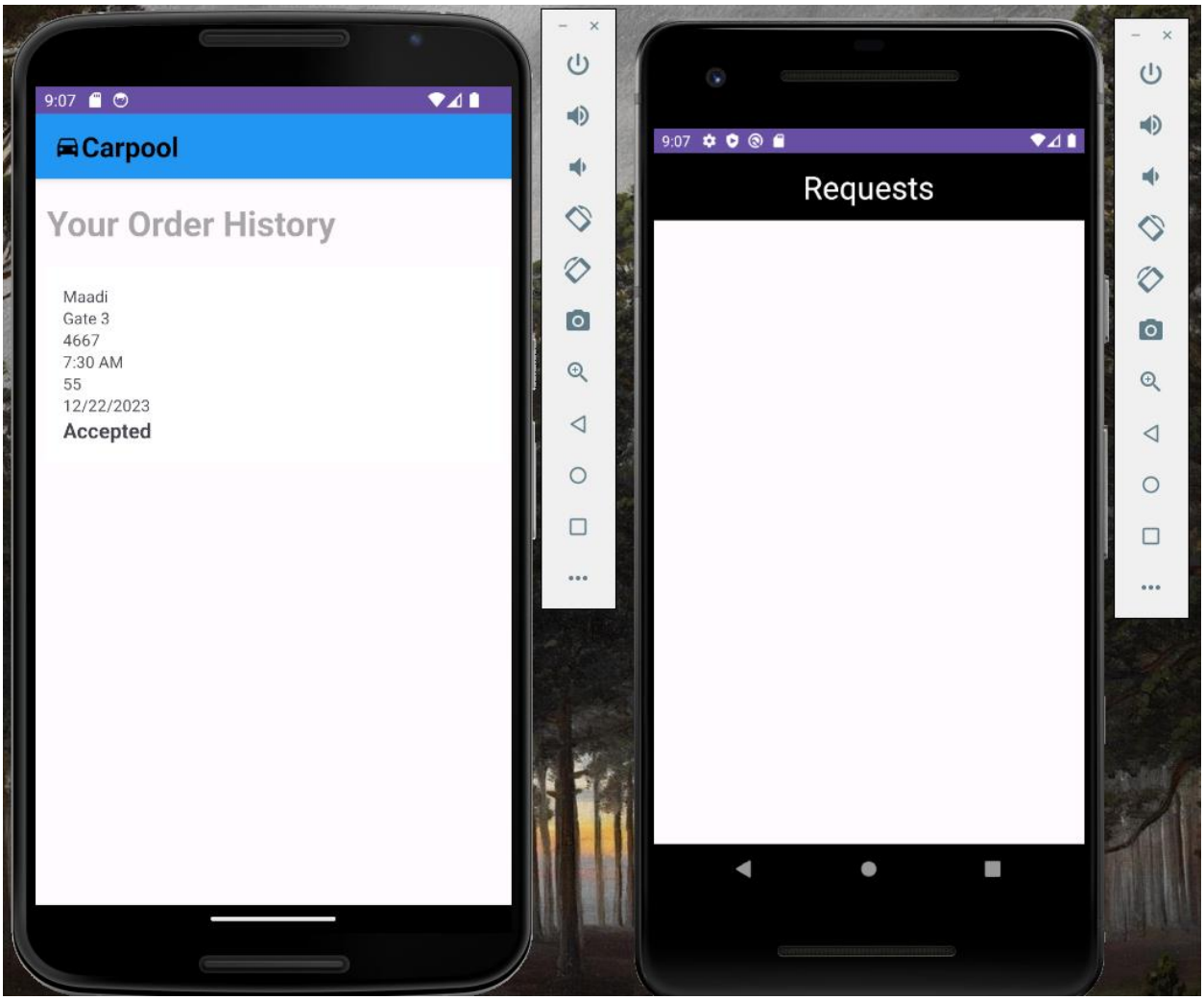
Show requests at driver side



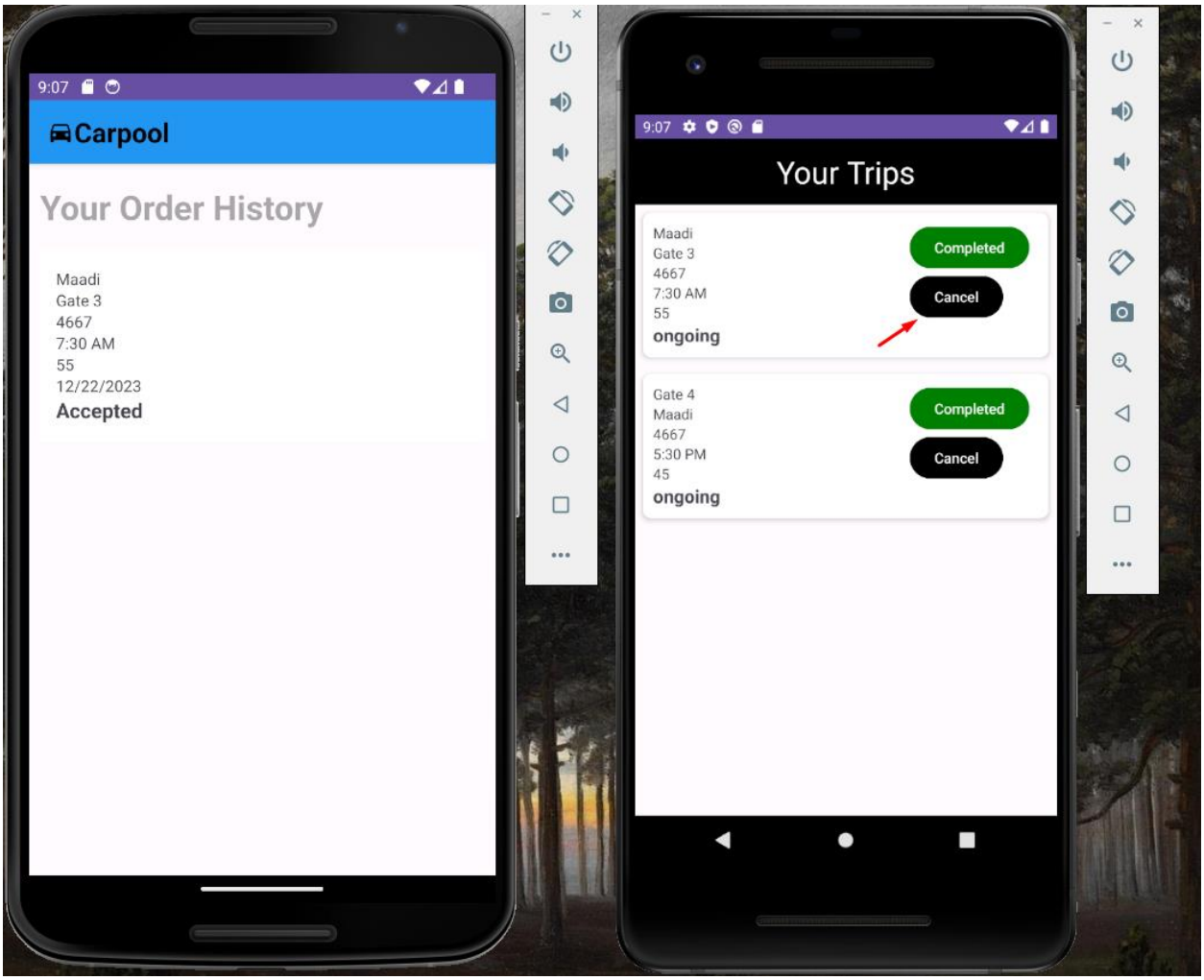
View history at user side



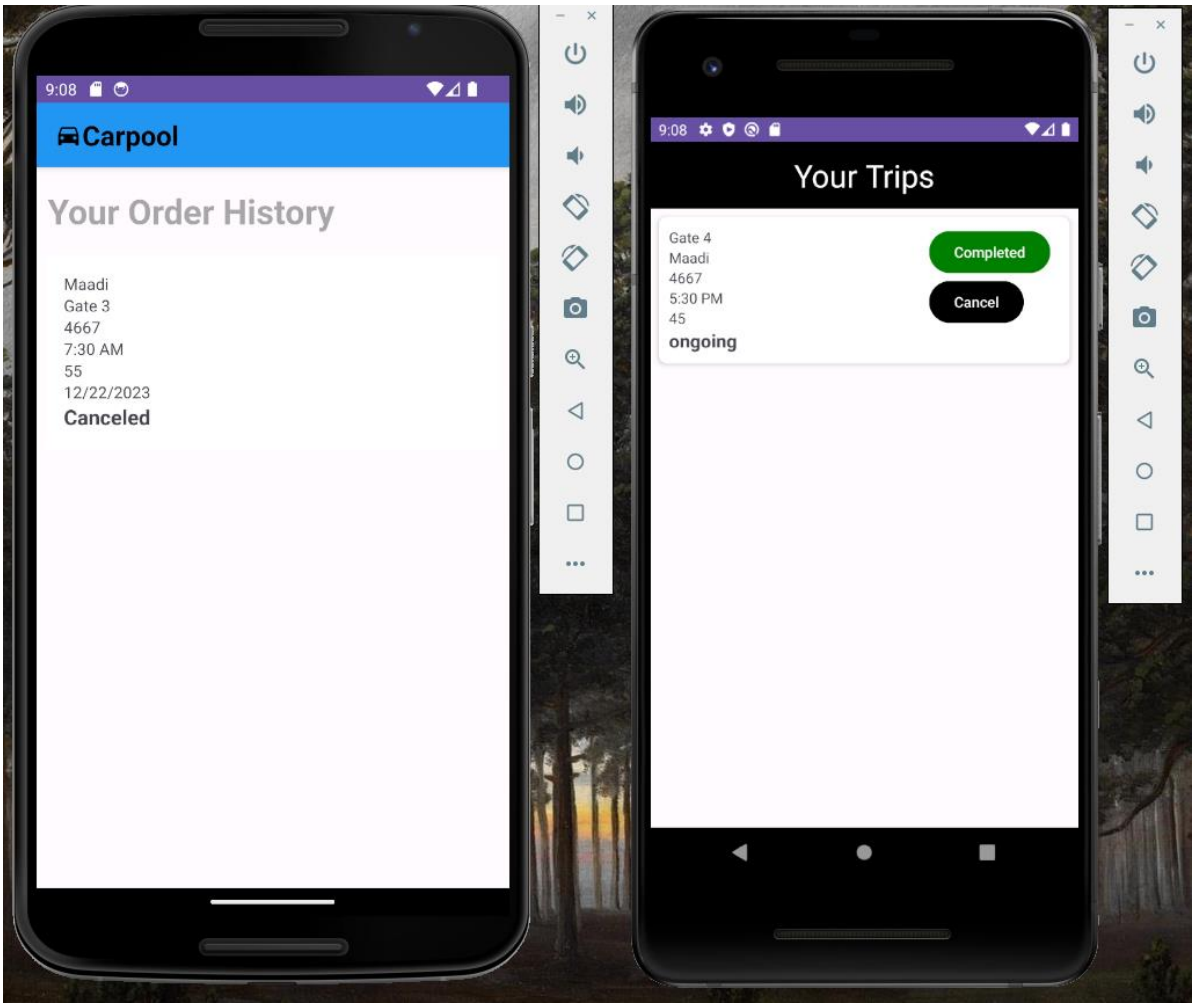
Confirm the request



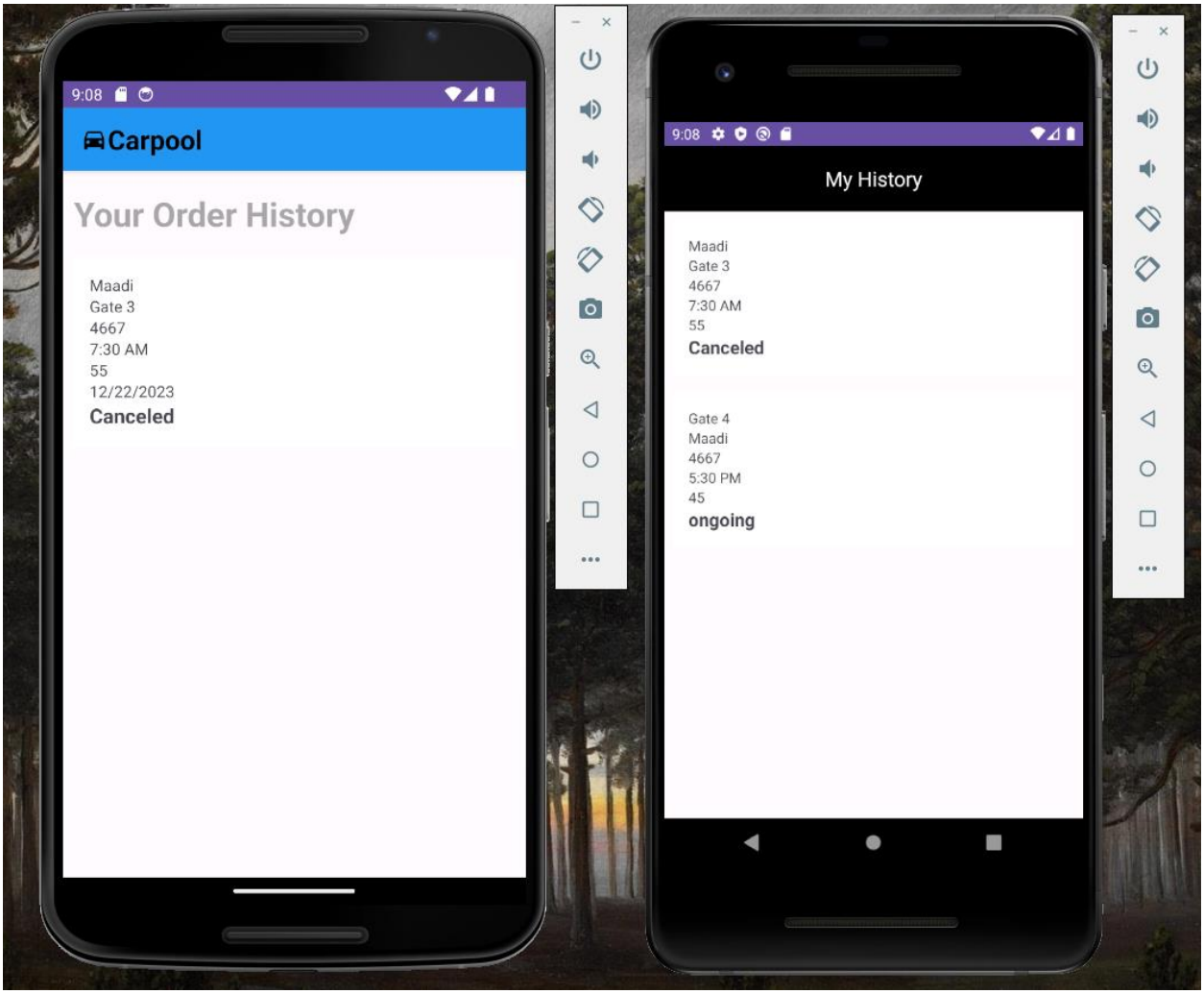
Test if I cancel the trip



State at user side updated to canceled

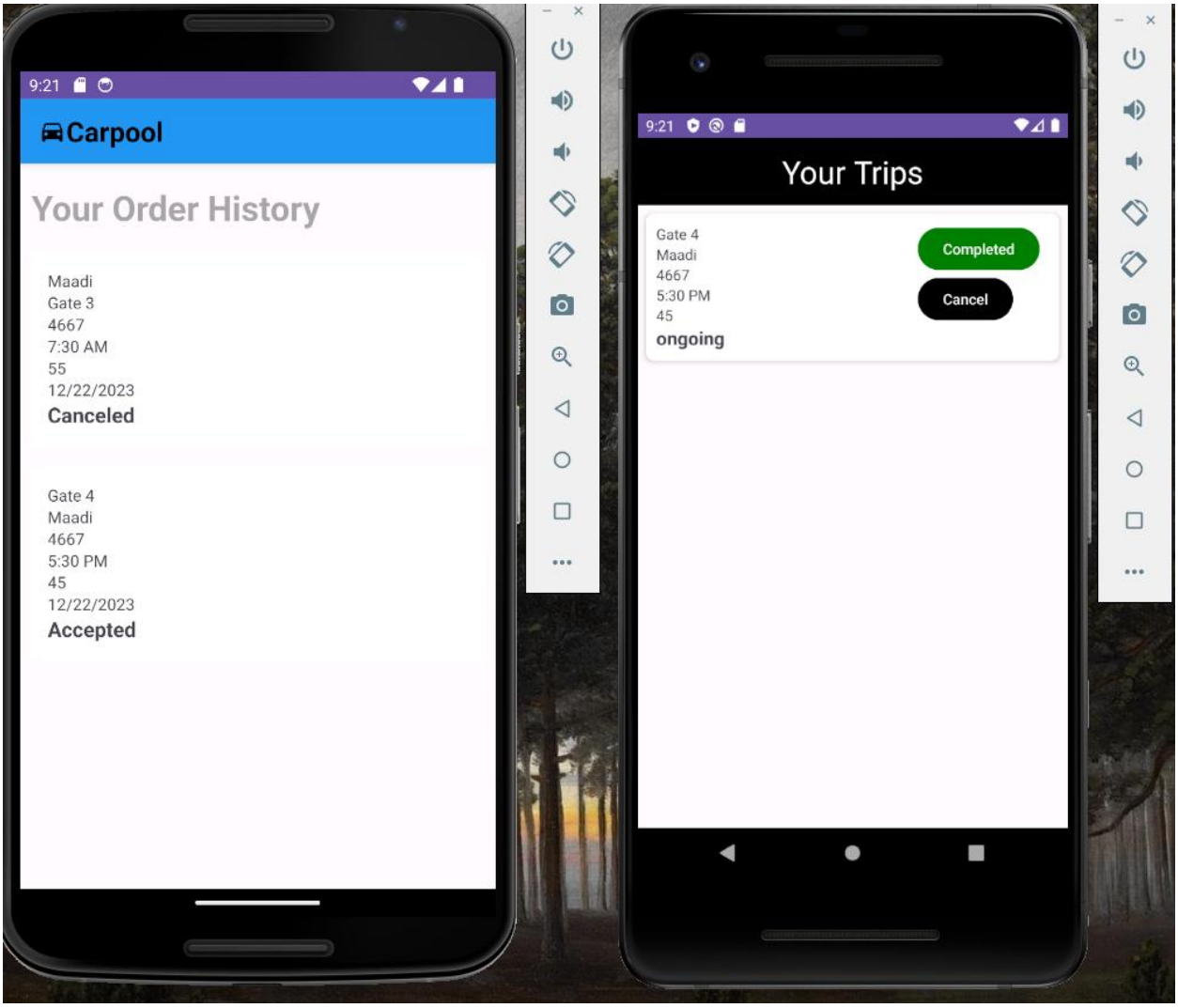


History at driver side

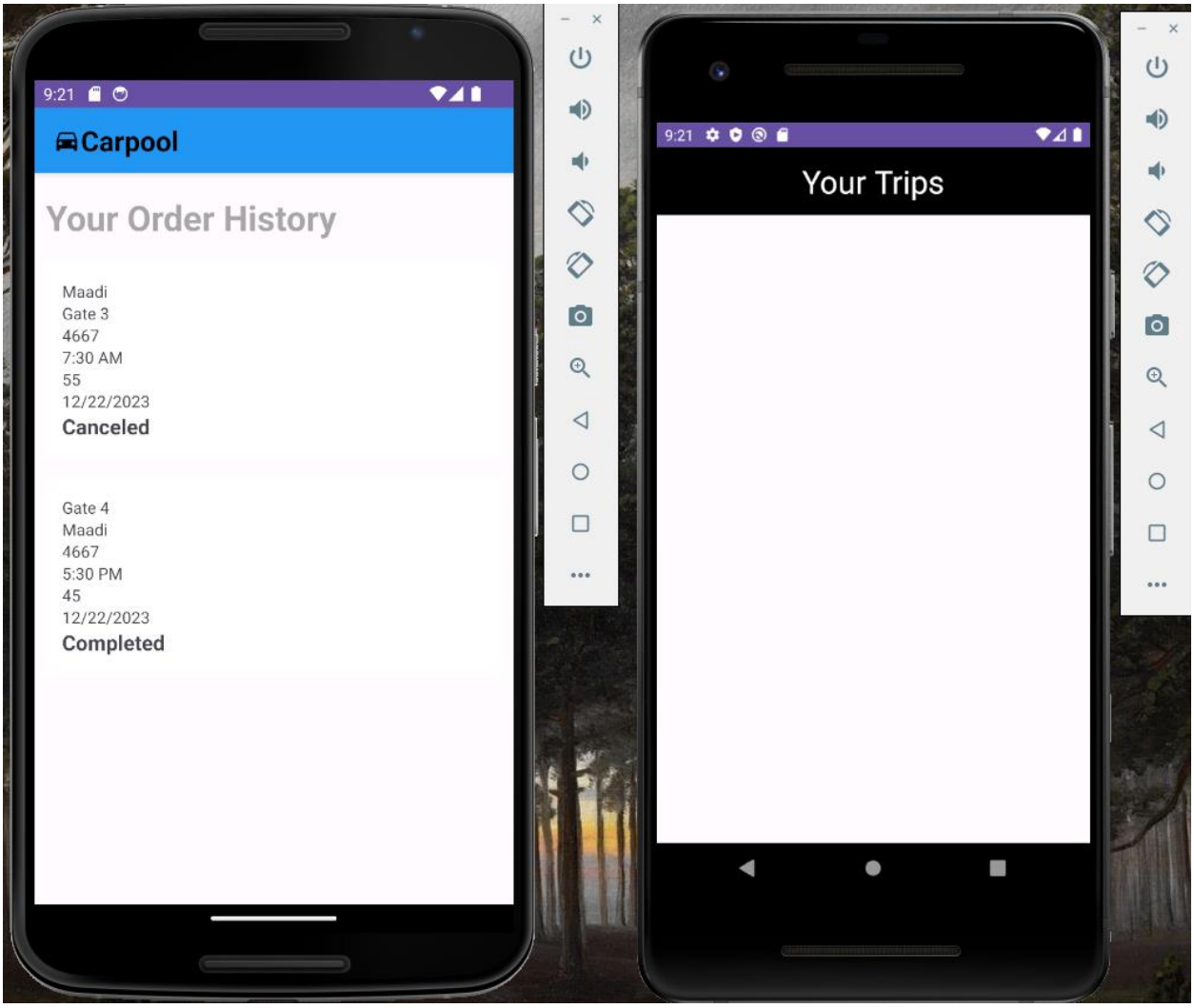


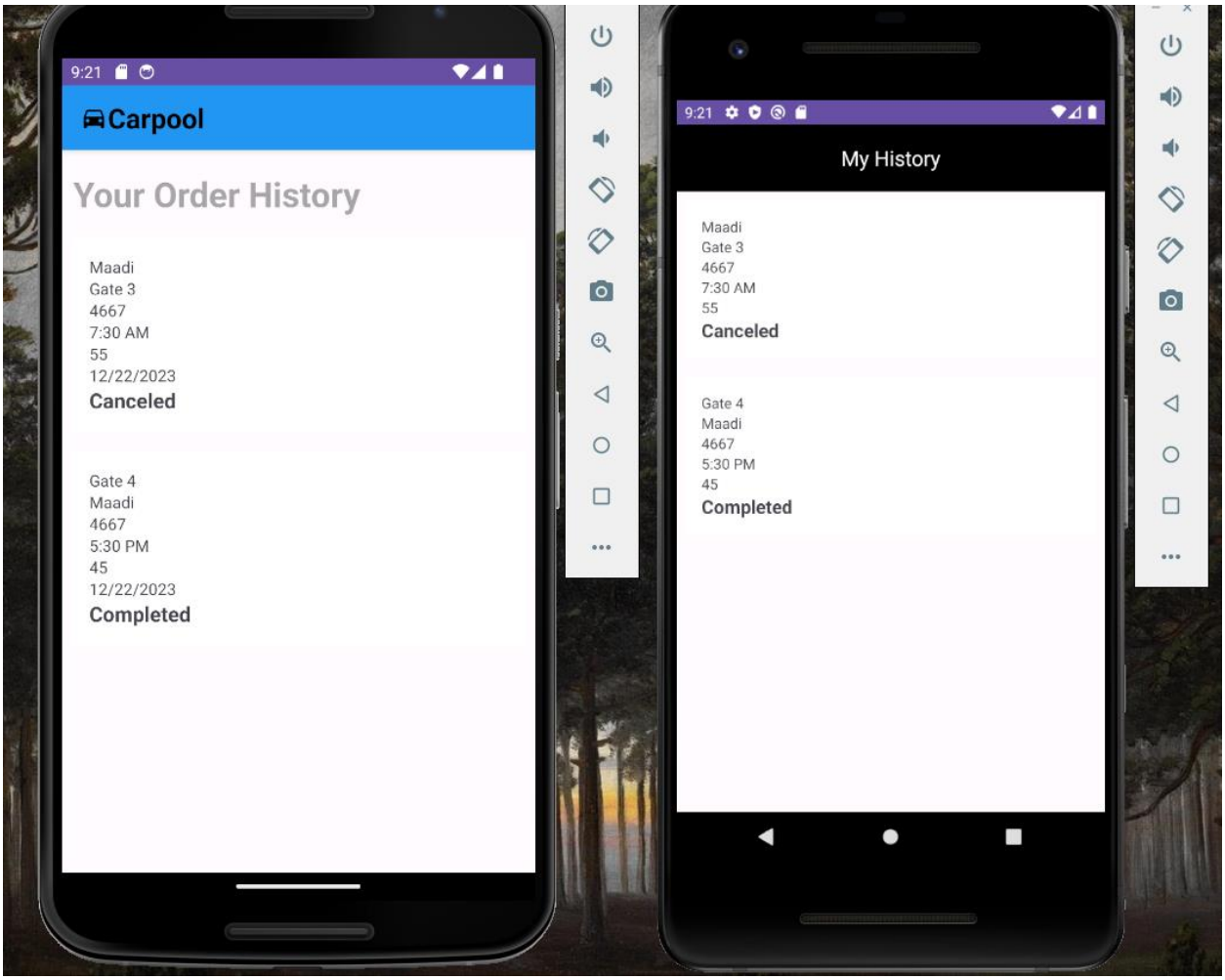
After each completing trip, driver should click on completed button for the trip state at driver and user history change to “Completed”

Example



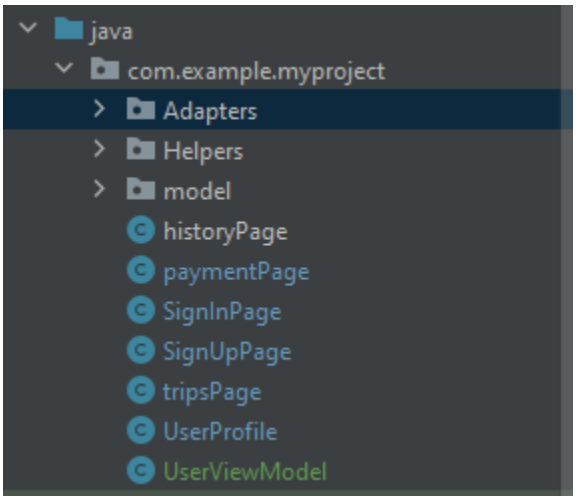
Click on completed


9:21
Carpool
Your Order History
Maadi
Gate 3
4667
7:30 AM
55
12/22/2023
Canceled
Gate 4
Maadi
4667
5:30 PM
45
12/22/2023
Completed
9:21
Your Trips

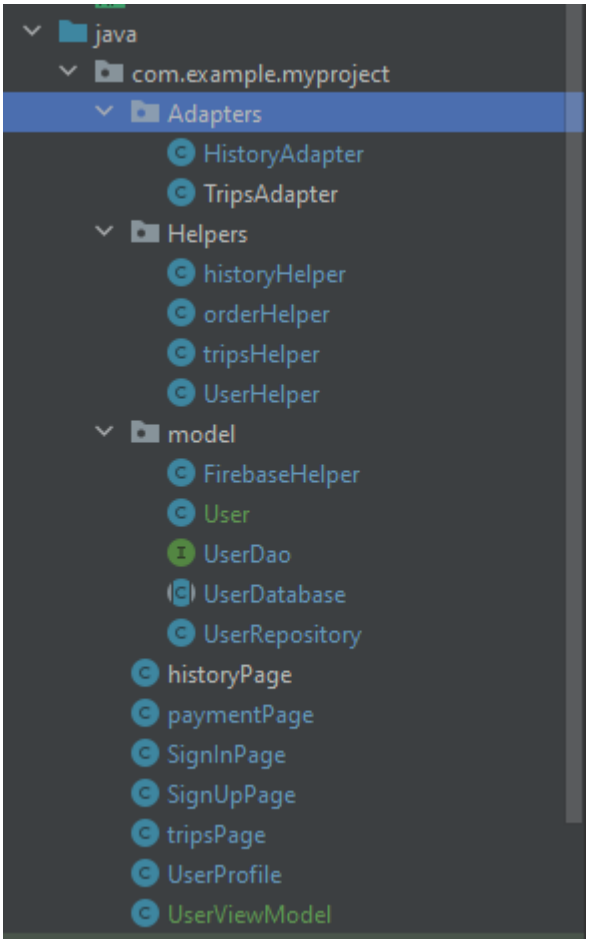

9:21
Carpool
Your Order History
Maadi
Gate 3
4667
7:30 AM
55
12/22/2023
Canceled
Gate 4
Maadi
4667
5:30 PM
45
12/22/2023
Completed
9:21
My History
Maadi
Gate 3
4667
7:30 AM
55
Canceled
Gate 4
Maadi
4667
5:30 PM
45
Completed

Code

User side



FirebaseHelper included all firebase operations such as queries and references (to separate model)



Entity

```
package com.example.myproject.model;

import androidx.annotation.NonNull;
import androidx.room.Entity;
import androidx.room.PrimaryKey;

@Entity(tableName = "users")
public class User {
    @PrimaryKey
    @NonNull
    public String userId;
    @NonNull
    public String email;

    @NonNull
    public String name;

    public User(@NonNull String userId, @NonNull String email, @NonNull
String name) {
        this.userId = userId;
        this.email = email;
        this.name = name;
    }
}
```

Dao

```
package com.example.myproject.model;

import androidx.room.Dao;
import androidx.room.Insert;
import androidx.room.OnConflictStrategy;
import androidx.room.Query;

import androidx.lifecycle.LiveData;

import java.util.List;

@Dao
public interface UserDao {

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void insertUser(User user);

    @Query("SELECT * FROM users")
    LiveData<List<User>> getAllUsers();

    @Query("SELECT * FROM users WHERE userId = :userId")
    User getUserById(String userId);

}
```

Database Room

```
package com.example.myproject.model;

import android.content.Context;

import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

@Database(entities = {User.class}, version = 1, exportSchema = false)
public abstract class UserDatabase extends RoomDatabase {
    public abstract UserDao userDao();

    private static volatile UserDatabase INSTANCE;
    private static final int NUMBER_OF_THREADS = 3;
    static final ExecutorService databaseWriteExecutor =
            Executors.newFixedThreadPool(NUMBER_OF_THREADS);

    static UserDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (UserDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE =
Room.databaseBuilder(context.getApplicationContext(),
                                    UserDatabase.class, "user_database")
                            .build();
                }
            }
        }
        return INSTANCE;
    }
}
```

Repository class

```
import android.content.Context;


import android.app.Application;

import androidx.annotation.NonNull;
import androidx.lifecycle.LiveData;

import com.example.myproject.Helpers.UserHelper;

import java.util.List;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Future;

public class UserRepository {

    private UserDao mUserDao;
    private LiveData<List<User>> mAllUsers;
    private Context mContext;
    private FirebaseHelper firebaseDB;

    public UserRepository(Application application) {
        UserDatabase db = UserDatabase.getDatabase(application);
        mUserDao = db.userDao();
        mAllUsers = mUserDao.getAllUsers();
        mContext = application.getApplicationContext();
        firebaseDB = FirebaseHelper.getInstance();
    }

    public LiveData<List<User>> getAllUsers() {

        return mAllUsers;
    }

    // You must call this on a non-UI thread or your app will throw an
exception. Room ensures
    // that you're not doing any long running operations on the main thread,
blocking the UI.
    public void insertUser(User user) {
        UserDatabase.databaseWriteExecutor.execute(() -> {
            mUserDao.insertUser(user);
        });
    }
    public User getUserById(String userId) {
        Future<User> future = UserDatabase.databaseWriteExecutor.submit(() ->
{
            return mUserDao.getUserById(userId);
        });

        try {
            User user = future.get();
            return user;
        } catch (InterruptedException | ExecutionException e) {
```

```
            // Handle exceptions as needed
            e.printStackTrace();
            return null;
        }
    }
    public void checkUserExistsInRoom(String userId){
        User user = getUserById(userId);
        if(user == null){
            firebaseDB.fetchDataFromFirebase(userId, new
FirebaseHelper.DataCallback<UserHelper>() {
                @Override
                public void onDataLoaded(UserHelper data) {
                    User user = new User(data.getUid(), data.getEmail(),
data.getUniID());
                    insertUser(user);
                }
                @Override
                public void onError(String errorMessage) {

                }
            });
        }
    }

}
```

ViewModel

```
package com.example.myproject;

import android.app.Application;

import androidx.lifecycle.AndroidViewModel;
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;


import com.example.myproject.model.User;
import com.example.myproject.model.UserRepository;

import java.util.List;

public class UserViewModel extends AndroidViewModel {

    private UserRepository mRepository;

    private final LiveData<List<User>> mAllUsers;

    public UserViewModel(Application application) {
        super(application);
        mRepository = new UserRepository(application);
        mAllUsers = mRepository.getAllUsers();
    }

    public LiveData<List<User>> getAllUsers() { return mAllUsers; }
    public LiveData<User> getUserById(String userId) {
```

```
        MutableLiveData<User> userLiveData = new MutableLiveData<>();

        User user = mRepository.getUserById(userId);
        userLiveData.postValue(user);

        return userLiveData;
    }
    public void checkUserExistsRoom(String userId){
        mRepository.checkUserExistsInRoom(userId);
    }
    public void insert(User user) { mRepository.insertUser(user); }

}
```

User data is fetched from firebase in Sign in page and is added to Room when user signs in to their account

```
mAuth.signInWithEmailAndPassword(Email, Password)
        .addOnCompleteListener(new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {
                    // Sign in success, update UI with the signed-in user's
information
                    Toast.makeText(SignInPage.this, "Succesful Login!",
                            Toast.LENGTH_SHORT).show();
                    firebaseDB.checkUser(mAuth);

UserViewModel.checkUserExistsRoom(mAuth.getCurrentUser().getUid());
                    Intent intent1=new Intent(SignInPage.this,
tripsPage.class);
                    startActivity(intent1);
                    finish();

                } else {
                    // If sign in fails, display a message to the user.
                    Toast.makeText(SignInPage.this, "Authentication failed.",
                            Toast.LENGTH_SHORT).show();
                }
            }
        });
```

And if user signs up user id, name and email is inserted into database in sign up page

```
firebaseAuth.createUserWithEmailAndPassword(createEmail, createPassword)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {
                    FirebaseUser user = firebaseAuth.getCurrentUser();
                    UserHelper helperClass=new
```

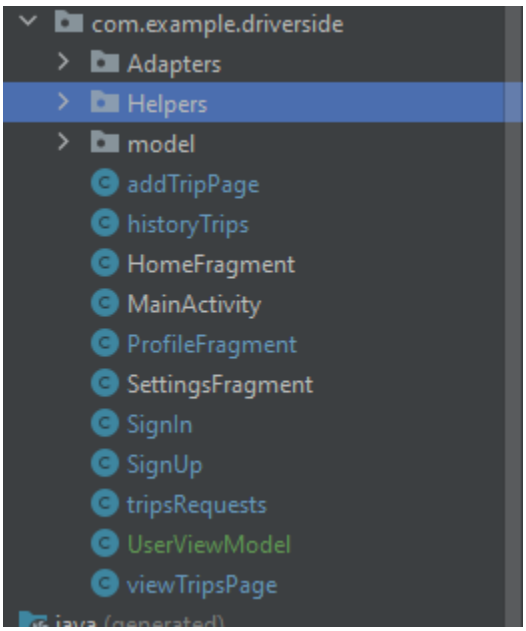
```
UserHelper(createID,createEmail, user.getUid());

firebaseDB.getUserReference().child(user.getUid()).setValue(helperClass);
                    User userx = new
User(firebaseAuth.getCurrentUser().getUid(), createEmail,createID);
                    UserViewModel.insert(userx);

                    Toast.makeText(SignUpPage.this, "USER CREATED!!!",
                            Toast.LENGTH_SHORT).show();
                    Intent intent1=new Intent(SignUpPage.this,
SignInPage.class);
                    startActivity(intent1);

                } else {
                    Toast.makeText(SignUpPage.this, "Authentication failed.",
                            Toast.LENGTH_SHORT).show();
                }
            }
        });
```

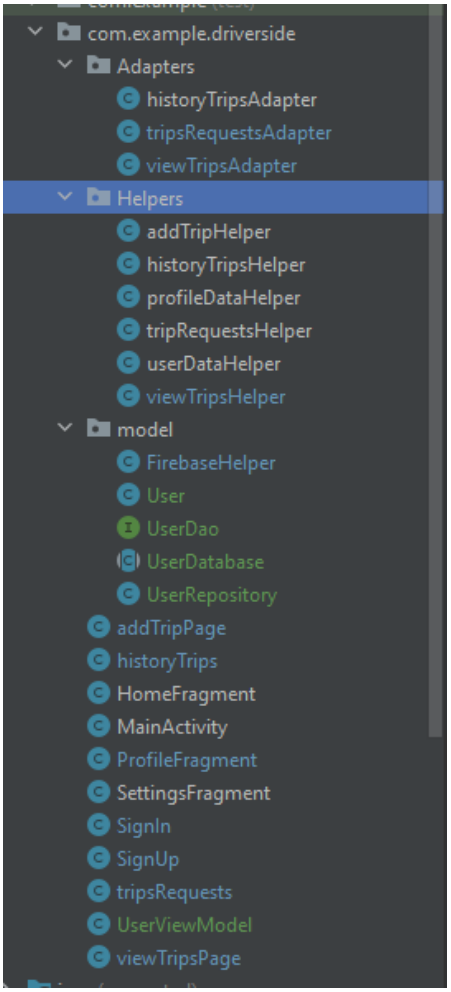
UserProfile.java

```
package com.example.myproject;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.ImageButton;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.ViewModelProvider;

import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Locale;

public class UserProfile extends AppCompatActivity {
    TextView email, name, time;
    private UserViewModel UserViewModel;
    FirebaseAuth auth;
    FirebaseUser user;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user_profile);

        email = findViewById(R.id.emailTextView);
        name = findViewById(R.id.nameTextView);
        time = findViewById(R.id.tripsTextView);
        auth = FirebaseAuth.getInstance();
        user = auth.getCurrentUser();

        UserViewModel = new ViewModelProvider(this).get(UserViewModel.class);
```

```
        UserViewModel.getUserById(user.getUid()).observe(this, userRoom -> {
            if(userRoom!=null) {
                name.setText(userRoom.name);
                email.setText(userRoom.email);
            }
        });

        Calendar calendar = Calendar.getInstance();
        SimpleDateFormat sdf = new SimpleDateFormat("h:mm a",
Locale.getDefault());
        String currentTime = sdf.format(calendar.getTime());
        time.setText("Current Time: " + currentTime);


        ImageButton btn = findViewById(R.id.logOut);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                FirebaseAuth.getInstance().signOut();
                Intent intent2 = new Intent(UserProfile.this,
SignInPage.class);
                startActivity(intent2);
                finish();
            }
        });
    }

}
```

Driver side


com.example.driverside
Adapters
Helpers
model
addTripPage
historyTrips
HomeFragment
MainActivity
ProfileFragment
SettingsFragment
SignIn
SignUp
tripsRequests
UserViewModel
viewTripsPage


com.example.driverside
Adapters
historyTripsAdapter
tripsRequestsAdapter
viewTripsAdapter
Helpers
addTripHelper
historyTripsHelper
profileDataHelper
tripRequestsHelper
userDataHelper
viewTripsHelper
model
FirebaseHelper
User
UserDao
UserDatabase
UserRepository
addTripPage
historyTrips
HomeFragment
MainActivity
ProfileFragment
SettingsFragment
SignIn
SignUp
tripsRequests
UserViewModel
viewTripsPage

The room at drive side however profile is a fragment so instead I call getActivity()

In observe.

```
UserViewModel = new ViewModelProvider(this).get(UserViewModel.class);

UserViewModel.getUserById(user.getUid()).observe(getActivity(), userRoom -> {
    if(userRoom!=null) {
        idText.setText(userRoom.name);
        emailText.setText(userRoom.email);
    }
});
```

Firebase (user side)

```
package com.example.myproject.model;

import androidx.annotation.NonNull;

import com.example.myproject.Helpers.UserHelper;
import com.example.myproject.Helpers.orderHelper;
import com.example.myproject.Helpers.tripsHelper;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.Query;
import com.google.firebase.database.ValueEventListener;

import java.util.ArrayList;

public class FirebaseHelper {
    private FirebaseDatabase database;
    private DatabaseReference userReference, driverReference, tripsReference,
orderReference;
    private static FirebaseHelper instance;
    private FirebaseHelper() {
        // Initialize Firebase Realtime Database
        database = FirebaseDatabase.getInstance();
        userReference = database.getReference("users");
        driverReference = database.getReference("drivers");
        tripsReference = database.getReference("trips");
        orderReference =
FirebaseDatabase.getInstance().getReference("orders");
    }
    public static synchronized FirebaseHelper getInstance() {
        if (instance == null) {
            instance = new FirebaseHelper();
        }
        return instance;
    }
    public DatabaseReference getUserReference() {
        return userReference;
    }

    public DatabaseReference getTripsReference() {
        return tripsReference;
    }
    public DatabaseReference getOrderReference() {
        return orderReference;
    }

    public void updateTripPassengerNumber(String tripId, String passengers){
        tripsReference.child(tripId).child("maxRider").setValue(passengers);
    }
    public void insertOrder(String orderId, orderHelper order){
        orderReference.child(orderId).setValue(order);
    }
```

```
    public void checkUser(FirebaseAuth mAuth){
        FirebaseUser user = mAuth.getCurrentUser();
        final String[] uniID = new String[1];
        DatabaseReference reference =
FirebaseDatabase.getInstance().getReference("users");
        Query checkUserDatabase=
reference.orderByChild("uid").equalTo(user.getUid());
        DatabaseReference reference2 =
FirebaseDatabase.getInstance().getReference("drivers");
        Query checkDriverDatabase  =
reference2.orderByChild("uid").equalTo(user.getUid());
        checkDriverDatabase.addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if(snapshot.exists()){
                    uniID[0] =
snapshot.child(user.getUid()).child("uniID").getValue(String.class);
                }
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
            }
        });
        checkUserDatabase.addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if(!snapshot.exists()){
                    UserHelper helperClass=new
UserHelper(uniID[0],user.getEmail(),user.getUid());
                    reference.child(user.getUid()).setValue(helperClass);
                }
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
            }
        });
    }
    public interface DataCallback<T> {
        void onDataLoaded(T data);
        void onError(String errorMessage);
    }
    public void getTrip(String tripId, DataCallback<tripsHelper> callback) {
        Query checkTripDatabase =
tripsReference.orderByChild("tripID").equalTo(tripId);
        checkTripDatabase.addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if (snapshot.exists()) {
                    tripsHelper trip =
snapshot.child(tripId).getValue(tripsHelper.class);
                    callback.onDataLoaded(trip);
                } else {
                    callback.onError("Trip not found");
```

```
                }
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                callback.onError(error.getMessage());
            }
        });
    }
    public void getTripOrders(String tripId, DataCallback<ArrayList<String>>
callback) {
        Query checkTripDatabase =
tripsReference.orderByChild("tripID").equalTo(tripId);
        checkTripDatabase.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if (snapshot.exists()) {
                    tripsHelper trip =
snapshot.child(tripId).child("").getValue(tripsHelper.class);

                    if (trip != null && trip.getOrders() != null) {
                        callback.onDataLoaded(new
ArrayList<>(trip.getOrders()));
                    } else {
                        callback.onError("Orders not found for the trip");
                    }
                } else {
                    callback.onError("Trip not found");
                }
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                callback.onError(error.getMessage());
            }
        });
    }
    public void updateTripOrders(String tripId, ArrayList<String> orders){
        tripsReference.child(tripId).child("orders").setValue(orders);
    }
    public void fetchDataFromFirebase(String userId, DataCallback<UserHelper>
callback) {
        userReference.child(userId).addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot snapshot) {
                if (snapshot.exists()) {
                    UserHelper user = snapshot.getValue(UserHelper.class);
                    callback.onDataLoaded(user);
                }
            }
            @Override
            public void onCancelled(DatabaseError error) {
                callback.onError(error.getMessage());
            }
        });
    }
```

```
}
```

Firebase (driver side)

```
package com.example.driverside.model;

import android.text.TextUtils;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.AsyncListUtil;

import com.example.driverside.Helpers.addTripHelper;
import com.example.driverside.Helpers.userDataHelper;
import com.example.driverside.Helpers.viewTripsHelper;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.Query;
import com.google.firebase.database.ValueEventListener;

import java.util.ArrayList;

public class FirebaseHelper {
    private FirebaseDatabase database;
    private DatabaseReference userReference, driverReference, tripsReference,
orderReference;
    private static FirebaseHelper instance;
    private FirebaseHelper() {
        // Initialize Firebase Realtime Database
        database = FirebaseDatabase.getInstance();
        userReference = database.getReference("users");
        driverReference = database.getReference("drivers");
        tripsReference = database.getReference("trips");
        orderReference =
FirebaseDatabase.getInstance().getReference("orders");
    }
    public static synchronized FirebaseHelper getInstance() {
        if (instance == null) {
            instance = new FirebaseHelper();
        }
        return instance;
    }
    public DatabaseReference getUserReference() {
        return userReference;
    }

    public DatabaseReference getDriverReference() {
        return driverReference;
    }

    public DatabaseReference getTripsReference() {
        return tripsReference;
    }
```

```
    public DatabaseReference getOrderReference() {
        return orderReference;
    }
    public void insertTrip(String orderId, addTripHelper trip){
        tripsReference.child(orderId).setValue(trip);
    }
    public void updateOrderStateAccept(String orderId){

orderReference.child(orderId).child("tripState").setValue("Accepted");
    }
    /*public void updateOrderStateDeclined(String orderId){

orderReference.child(orderId).child("tripState").setValue("Declined");
    }*/
    public void updateStateCompleted(String orderId){

tripsReference.child(orderId).child("driverStatus").setValue("Completed");
    }
    public void updateStateCanceled(String orderId){

tripsReference.child(orderId).child("driverStatus").setValue("Canceled");
    }
    public void updateRequestNotConfirmed(String orderId){
        orderReference.child(orderId).child("tripState").setValue("Not
Confirmed");
    }

    public void updateOrderStateCompleted(String orderId){

orderReference.child(orderId).child("driverStatus").setValue("Completed");

orderReference.child(orderId).child("tripState").setValue("Completed");
    }
    public void updateOrderStateCanceled(String orderId){

orderReference.child(orderId).child("driverStatus").setValue("Canceled");

orderReference.child(orderId).child("tripState").setValue("Canceled");
    }

    public void checkUser(FirebaseAuth mAuth){
        FirebaseUser user = mAuth.getCurrentUser();
        final String[] uniID = new String[1];
        DatabaseReference reference =
FirebaseDatabase.getInstance().getReference("users");
        Query checkUserDatabase=
reference.orderByChild("uid").equalTo(user.getUid());
        DatabaseReference reference2 =
FirebaseDatabase.getInstance().getReference("drivers");
        Query checkDriverDatabase  =
reference2.orderByChild("uid").equalTo(user.getUid());
        checkUserDatabase.addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if(snapshot.exists()){
```

```
                    uniID[0] =
snapshot.child(user.getUid()).child("uniID").getValue(String.class);

                }
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
            }
        });
        checkDriverDatabase.addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if(!snapshot.exists()){
                    userDataHelper helperClass=new
userDataHelper(uniID[0],user.getEmail(),user.getUid());
                    reference2.child(user.getUid()).setValue(helperClass);
                }
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
            }
        });
    }
    public interface TimeSlotCallback {
        void onCallback(boolean isAvailable);
    }
    public void isTimeSlotAvailable(String selectedTime, String
selectedDate,String driveremail ,TimeSlotCallback callback) {
        DatabaseReference tripsRef = getTripsReference();
        Query query =
tripsRef.orderByChild("timeTrip").equalTo(selectedTime);
        query.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
                for (DataSnapshot tripSnapshot : dataSnapshot.getChildren())
{
                    addTripHelper existingTrip =
tripSnapshot.getValue(addTripHelper.class);

                    // Check if the existing trip is on the same date and
time
                    if (existingTrip != null &&
TextUtils.equals(existingTrip.getDriverEmail(),driveremail) &&

!TextUtils.equals(existingTrip.getDriverStatus(),"Canceled") &&

TextUtils.equals(existingTrip.getTripDate(),selectedDate)
                            &&
TextUtils.equals(existingTrip.getTimeTrip(),selectedTime)) {
                        callback.onCallback(false);
                        return; // No need to continue checking

                    }
                }
                // Invoke the callback with the result
                callback.onCallback(true);
```

```
            }

            @Override
            public void onCancelled(@NonNull DatabaseError databaseError) {
                // Handle the error
                callback.onCallback(false); // Notify callback of failure
            }
        });
    }
    public interface DataCallback<T> {
        void onDataLoaded(T data);
        void onError(String errorMessage);
    }

    public void updateTripOrders(String tripId, String state){
        getTripOrders(tripId, new DataCallback<ArrayList<String>>() {
            @Override
            public void onDataLoaded(ArrayList<String> data) {
                for (String orderId : data) {
                    updateOrderState(orderId,state);
                }
            }
            @Override
            public void onError(String errorMessage) {

            }
        });
    }

    private void getTripOrders(String tripId, DataCallback<ArrayList<String>>
callback) {
        Query checkTripDatabase =
tripsReference.orderByChild("tripID").equalTo(tripId);
        checkTripDatabase.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                if (snapshot.exists()) {
                    viewTripsHelper trip =
snapshot.child(tripId).child("").getValue(viewTripsHelper.class);

                    if (trip != null && trip.getOrders() != null) {
                        callback.onDataLoaded(new
ArrayList<>(trip.getOrders()));
                    } else {
                        callback.onError("Orders not found for the trip");
                    }
                } else {
                    callback.onError("Trip not found");
                }
            }
            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                callback.onError(error.getMessage());
            }
        });
    }
```

```
    private void updateOrderState(String orderId, String state) {
        if (TextUtils.equals(state, "Completed")) {
            updateOrderStateCompleted(orderId);
        } else if (TextUtils.equals(state, "Canceled")) {
            updateOrderStateCanceled(orderId);
        }
    }
    public void fetchDataFromFirebase(String userId,
DataCallback<userDataHelper> callback) {
        driverReference.child(userId).addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot snapshot) {
                if (snapshot.exists()) {
                    userDataHelper user =
snapshot.getValue(userDataHelper.class);
                    callback.onDataLoaded(user);
                }
            }
            @Override
            public void onCancelled(DatabaseError error) {
                callback.onError(error.getMessage());
            }
        });
    }

}
```

Time constraint function for trips

Customers who need a ride at 7:30 must reserve their seat before 10:00 pm previous day. Customers who need a ride from campus at 5:30 pm must reserve their seat before 1:00 pm same day.

```
private boolean canBookTrip(String tripDate, String tripTime) {
    // Parsing trip date
    String[] dateParts = tripDate.split("/");
    int monthOfTrip = Integer.parseInt(dateParts[0]);
    int tripDay = Integer.parseInt(dateParts[1]);
    int yearOfTrip = Integer.parseInt(dateParts[2]);

    // Get the current date and time
    Calendar currentCalendar = Calendar.getInstance();
    int currentMonth = currentCalendar.get(Calendar.MONTH) + 1; // Month is
0-based
    int currentDay = currentCalendar.get(Calendar.DAY_OF_MONTH);
    int currentYear = currentCalendar.get(Calendar.YEAR);
    int currentHour = currentCalendar.get(Calendar.HOUR);
    int currentMinute = currentCalendar.get(Calendar.MINUTE);

    // Get the current hour in 12-hour format (AM/PM)
    SimpleDateFormat dateFormat = new SimpleDateFormat("hh:mm a");
    String currentTimeFormatted =
dateFormat.format(currentCalendar.getTime());

    Date parsedTime = null;
    try {
        parsedTime = dateFormat.parse(currentTimeFormatted);
    } catch (ParseException e) {
        throw new RuntimeException(e);
    }
    currentCalendar.setTime(parsedTime);
    currentCalendar.set(currentYear, currentMonth - 1, currentDay);

    // Create a calendar for the trip
    Calendar tripCalendar = Calendar.getInstance();
    tripCalendar.set(yearOfTrip, monthOfTrip - 1, tripDay);

    // Parsing trip time
    String[] timeParts = tripTime.split(":");
    int parsedHour = Integer.parseInt(timeParts[0]);
    String tripAmPm = timeParts[1].split(" ")[1].trim(); // Extract AM/PM
after minutes

    // Set trip calendar with parsed time
    tripCalendar.set(Calendar.HOUR_OF_DAY, parsedHour);
    tripCalendar.set(Calendar.MINUTE, Integer.parseInt(timeParts[1].split("
")[0]));

    // Extract current time components
    currentHour = currentCalendar.get(Calendar.HOUR);
    currentMinute = currentCalendar.get(Calendar.MINUTE);
```

```
    // Extract current AM/PM
    String currentAmPm = currentCalendar.get(Calendar.AM_PM) == Calendar.AM ?
"AM" : "PM";

    // Check if the trip is on the same day, month, and year
    boolean isSameDayAndMonthAndYear =
            currentCalendar.get(Calendar.YEAR) ==
tripCalendar.get(Calendar.YEAR) &&
                    currentCalendar.get(Calendar.MONTH) ==
tripCalendar.get(Calendar.MONTH) &&
                    currentCalendar.get(Calendar.DAY_OF_MONTH) ==
tripCalendar.get(Calendar.DAY_OF_MONTH);

    // Check if the trip is on the next day
    boolean isNextDay =
            // Check for the same month and consecutive days
            (currentCalendar.get(Calendar.YEAR) ==
tripCalendar.get(Calendar.YEAR) &&
                    currentCalendar.get(Calendar.MONTH) ==
tripCalendar.get(Calendar.MONTH) &&
                    currentCalendar.get(Calendar.DAY_OF_MONTH) ==
tripCalendar.get(Calendar.DAY_OF_MONTH) + 1) ||

                    // Check for the end of the month to the beginning of the
next month
                    (tripCalendar.get(Calendar.DAY_OF_MONTH) ==
tripCalendar.getActualMaximum(Calendar.DAY_OF_MONTH) &&
                            currentCalendar.get(Calendar.DAY_OF_MONTH) == 1
&&
                            currentCalendar.get(Calendar.MONTH) ==
tripCalendar.get(Calendar.MONTH) + 1) ||

                    // Check for the last day of the year to the first day of
the next year
                    (tripCalendar.get(Calendar.DAY_OF_MONTH) ==
tripCalendar.getActualMaximum(Calendar.DAY_OF_MONTH) &&
                            currentCalendar.get(Calendar.DAY_OF_MONTH) == 1
&&
                            currentCalendar.get(Calendar.YEAR) ==
tripCalendar.get(Calendar.YEAR) + 1);

    // Check specific conditions for allowing booking at 7:30 AM
    if (TextUtils.equals(tripTime, "7:30 AM")) {
        if (isSameDayAndMonthAndYear) {
            if ((currentHour < 10 && TextUtils.equals(currentAmPm, "PM")) ||
TextUtils.equals(currentAmPm, "AM") || (currentHour == 12 &&
TextUtils.equals(currentAmPm, "PM"))) {
                // Allow booking only if the trip time is before 10:00 pm or
12:00 pm (noon) on the trip date
                return true;
            } else {
                return false;
            }
        } else {
            return false;
        }
    }
```

```
    // Check specific conditions for allowing booking at 5:30 PM
    else if (tripTime.equals("5:30 PM")) {
        if (isSameDayAndMonthAndYear) {
            return true;
        } else if (isNextDay) {
            if (TextUtils.equals(currentAmPm, "AM") ||
(TextUtils.equals(currentAmPm, "PM") && currentHour < 1)) {
                // Allow booking if the trip is on the next day and the
current time is before 1:00 PM
                return true;
            } else {
                return false;
            }
        }
    }
    // Default case, return false if no conditions are met
    return false;
}
```

```
if (TextUtils.equals(tripTime, b: "7:30 AM")) {
    // Allow booking for 7:30 AM trip until 10:00 PM same day
    if(isSameDayAndMonthAndYear){
        if ((currentHour < 10 && TextUtils.equals(currentAmPm, b: "PM")) ||
                TextUtils.equals(currentAmPm, b: "AM")||
                (currentHour == 12 && TextUtils.equals(currentAmPm, b: "PM"))) {
            // Allow booking only if the trip time is before 10:00 pm or 12:00 pm (noon) on the trip date
            return true;
        }else{
            return false;
        }
    }else{
        return false;
    }
} else if (tripTime.equals("5:30 PM")) {
    // Allow booking for 5:30 PM trip until 1:00 PM next day
    if (isSameDayAndMonthAndYear) {
        return true;
    } else if (isNextDay) {
        if (TextUtils.equals(currentAmPm, b: "AM") || (TextUtils.equals(currentAmPm, b: "PM") &&
                currentHour < 1)) {
            return true;
        } else {
            return false;
        }
    }
}
```

Time constraint function for requests

Order must be confirmed before 11:30 pm for morning ride and before 4:30 pm for afternoon ride

```
private boolean requestTimeConstraint(String tripDate, String tripTime) {
    // Parsing trip date
    String[] dateParts = tripDate.split("/");
    int monthOfTrip = Integer.parseInt(dateParts[0]);
    int tripDay = Integer.parseInt(dateParts[1]);
    int yearOfTrip = Integer.parseInt(dateParts[2]);

    // Get the current date and time
    Calendar currentCalendar = Calendar.getInstance();
    int currentMonth = currentCalendar.get(Calendar.MONTH) + 1; // Month is
0-based
    int currentDay = currentCalendar.get(Calendar.DAY_OF_MONTH);
    int currentYear = currentCalendar.get(Calendar.YEAR);
    int currentHour = currentCalendar.get(Calendar.HOUR);
    int currentMinute = currentCalendar.get(Calendar.MINUTE);

    // Get the current hour in 12-hour format (AM/PM)
    SimpleDateFormat dateFormat = new SimpleDateFormat("hh:mm a");
    String currentTimeFormatted =
dateFormat.format(currentCalendar.getTime());

    Date parsedTime = null;
    try {
        parsedTime = dateFormat.parse(currentTimeFormatted);
    } catch (ParseException e) {
        throw new RuntimeException(e);
    }
    currentCalendar.setTime(parsedTime);
    currentCalendar.set(currentYear, currentMonth - 1, currentDay);

    // Create a calendar for the trip
    Calendar tripCalendar = Calendar.getInstance();
    tripCalendar.set(yearOfTrip, monthOfTrip - 1, tripDay);

    // Parsing trip time
    String[] timeParts = tripTime.split(":");
    int parsedHour = Integer.parseInt(timeParts[0]);
    String tripAmPm = timeParts[1].split(" ")[1].trim(); // Extract AM/PM
after minutes

    // Set trip calendar with parsed time
    tripCalendar.set(Calendar.HOUR_OF_DAY, parsedHour);
    tripCalendar.set(Calendar.MINUTE, Integer.parseInt(timeParts[1].split("
")[0]));

    // Extract current time components
    currentHour = currentCalendar.get(Calendar.HOUR);
    currentMinute = currentCalendar.get(Calendar.MINUTE);

    // Extract current AM/PM
    String currentAmPm = currentCalendar.get(Calendar.AM_PM) == Calendar.AM ?
```

```
"AM" : "PM";

    // Check if the trip is on the same day, month, and year
    boolean isSameDayAndMonthAndYear =
            currentCalendar.get(Calendar.YEAR) ==
tripCalendar.get(Calendar.YEAR) &&
                    currentCalendar.get(Calendar.MONTH) ==
tripCalendar.get(Calendar.MONTH) &&
                    currentCalendar.get(Calendar.DAY_OF_MONTH) ==
tripCalendar.get(Calendar.DAY_OF_MONTH);

    // Check if the trip is on the next day
    boolean isNextDay =
            // Check for the same month and consecutive days
            (currentCalendar.get(Calendar.YEAR) ==
tripCalendar.get(Calendar.YEAR) &&
                    currentCalendar.get(Calendar.MONTH) ==
tripCalendar.get(Calendar.MONTH) &&
                    currentCalendar.get(Calendar.DAY_OF_MONTH) ==
tripCalendar.get(Calendar.DAY_OF_MONTH) + 1) ||

                    // Check for the end of the month to the beginning of the
next month
                    (tripCalendar.get(Calendar.DAY_OF_MONTH) ==
tripCalendar.getActualMaximum(Calendar.DAY_OF_MONTH) &&
                            currentCalendar.get(Calendar.DAY_OF_MONTH) == 1
&&
                            currentCalendar.get(Calendar.MONTH) ==
tripCalendar.get(Calendar.MONTH) + 1) ||

                    // Check for the last day of the year to the first day of
the next year
                    (tripCalendar.get(Calendar.DAY_OF_MONTH) ==
tripCalendar.getActualMaximum(Calendar.DAY_OF_MONTH) &&
                            currentCalendar.get(Calendar.DAY_OF_MONTH) == 1
&&
                            currentCalendar.get(Calendar.YEAR) ==
tripCalendar.get(Calendar.YEAR) + 1);

    // Check specific conditions for allowing request at 7:30 AM
    if (TextUtils.equals(tripTime, "7:30 AM")) {
        if (isSameDayAndMonthAndYear) {
            if ((currentHour < 11 && TextUtils.equals(currentAmPm, "PM")) ||
                    (currentHour == 11 && currentMinute <= 30 &&
TextUtils.equals(currentAmPm, "PM")) ||
                    TextUtils.equals(currentAmPm, "AM")) {
                // Allow request only if the current time is before 11:30 PM
or 7:30 AM on the trip date
                return true;
            } else {
                return false; // Request declined
            }
        } else {
            return false; // Request declined
        }
    } else if (tripTime.equals("5:30 PM")) {
        // Check specific conditions for allowing request at 5:30 PM
```

```
        if (isSameDayAndMonthAndYear) {
            return true;
        } else if (isNextDay) {
            if (TextUtils.equals(currentAmPm, "AM") ||
(TextUtils.equals(currentAmPm, "PM") && currentHour < 4) ||
                    (TextUtils.equals(currentAmPm, "PM") && currentHour == 4
&& currentMinute <= 30)) {
                // Allow request if the trip is on the next day and the
current time is before 4:30 PM
                return true;
            } else {
                return false; // Request declined
            }
        }
    }
    // Default case, return false if no conditions are met
    return false;
}
```

```
if (TextUtils.equals(tripTime, b: "7:30 AM")) {
    // Allow request for 7:30 AM trip until 11:30 PM
    if(isSameDayAndMonthAndYear){
        if ((currentHour < 11 && TextUtils.equals(currentAmPm, b: "PM"))||
                (currentHour == 11 && currentMinute<=30 && TextUtils.equals(currentAmPm, b: "PM")||
                        TextUtils.equals(currentAmPm, b: "AM"))) {
            return true;
        }else{
            return false; //request declined
        }
    }else{
        return false;
    }
} else if (tripTime.equals("5:30 PM")) {
    // Allow requests for 5:30 PM trip until 4:30 PM next day
    if (isSameDayAndMonthAndYear) {
        return true;
    } else if (isNextDay) {
        if (TextUtils.equals(currentAmPm, b: "AM") || (TextUtils.equals(currentAmPm, b: "PM") && currentHour < 4) ||
                (TextUtils.equals(currentAmPm, b: "PM") && currentHour == 4 && currentMinute<=30)) {
            return true;
        } else {
            return false; //request declined
        }
    }
}
```