CSE 431 – Documentation Report

Name: Ahmed Sameh Mohamed Mourad

ID: 19P5861

Extra Functionalities:

1- **Dark Mode:** Implemented a visually comfortable and straightforward dark mode for improved user experience.
2- **Flutter Best Practices:** Applied Flutter best practices consistently across the entire app to ensure efficiency and maintainability, such as managing the state of each widget, separating the widgets, initializing variables, disposing of controllers and focus nodes, named navigation using the "routeName" static variable.
3- **Functional Programming for Error Handling:** Utilized functional programming principles for error handling to enhance modularity and code readability.
4- **Providers:** Used providers throughout the 2 apps.
5- **Flexible Reservation:** Allowed users to make reservations for any day within the current week, providing flexibility beyond restricting it to today only.
6- **Real-Time Date:** Both apps don't depend on the devices' date to operate (DateTime.now), it operates on the correct time of Google's server, so that no user can bypass the time constraints.
7- **Route Sorting:** Enable users to easily sort and organize routes according to their preferences, offering options for alphabetical sorting and sorting by price.
8- **TDD Pattern:** The 2 projects were constructed using the TDD architecture pattern.

Test Credentials (for both apps):

Email: test@eng.asu.edu.eg

Password: 123456

Introduction

User Version:

Welcome to our carpool app designed to make your daily commute convenient and eco-friendly! The user version of our app simplifies ride booking for you. Input your destination, and browse through available drivers heading your way.

Our app provides transparent pricing and easy payment options for a hassle-free experience. Additionally, enjoy the social aspect of commuting by connecting with other users, creating a community of like-minded individuals.

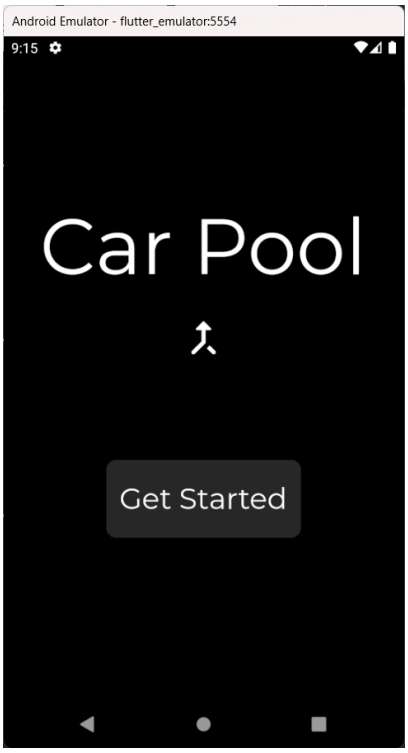
Driver Version:

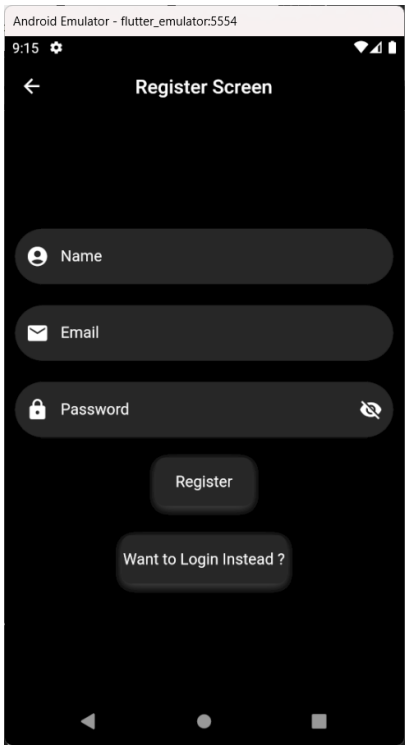
On the driver side of our carpool app, you can contribute to a greener planet while earning extra income. Set your daily routes, and let the app match you with passengers heading in the same direction.

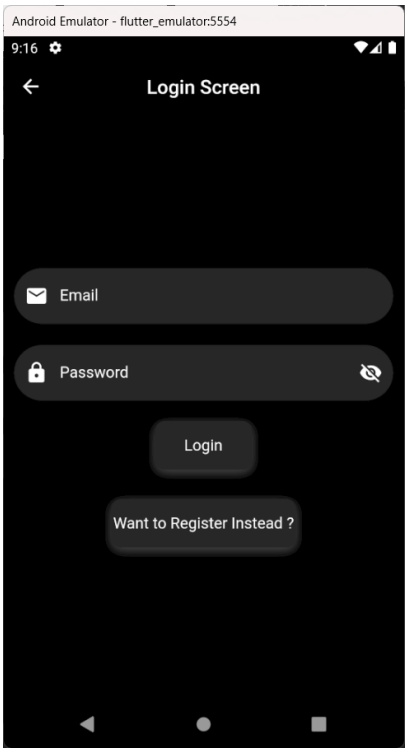
Feel secure with our verification system, ensuring that passengers are genuine from Ain Shams University. Earn fair compensation for your contribution to a more sustainable commute, and easily manage payouts through the app.

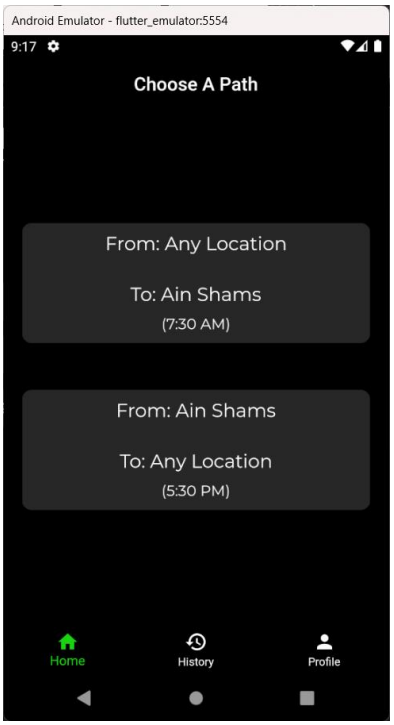
Building a sense of community among college colleagues is a key aspect of our app. Connect with your passengers, enjoy a more sociable commute, and be part of the revolution of shared commuting. Join us in making daily travel more affordable, social, and environmentally friendly!

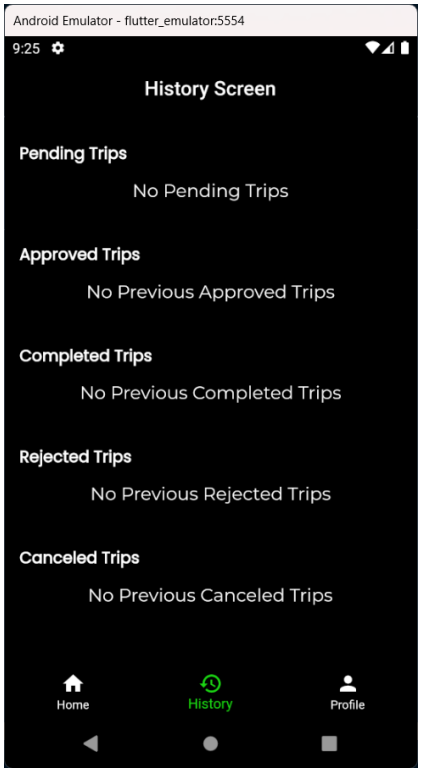
UI Design (User)

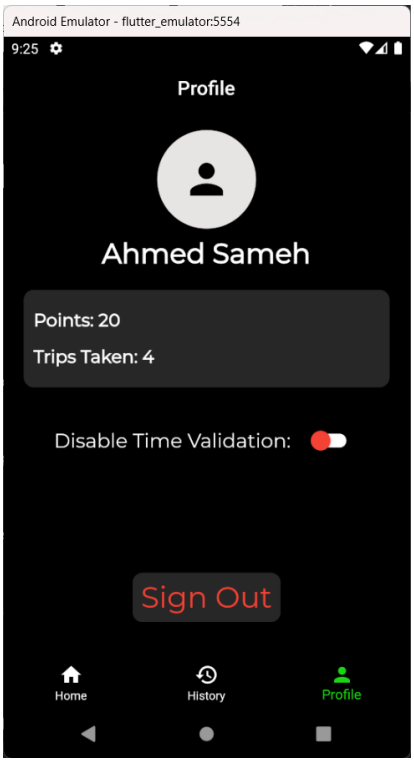


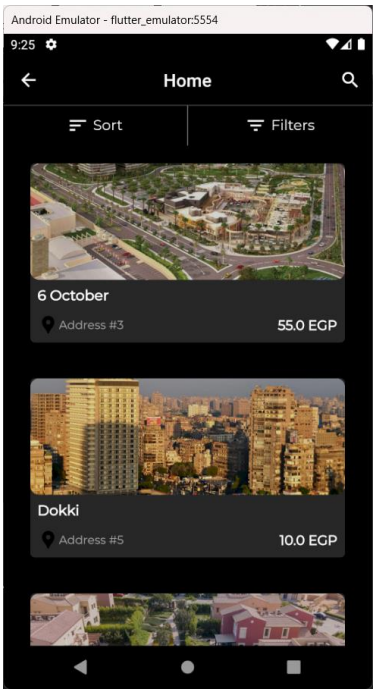


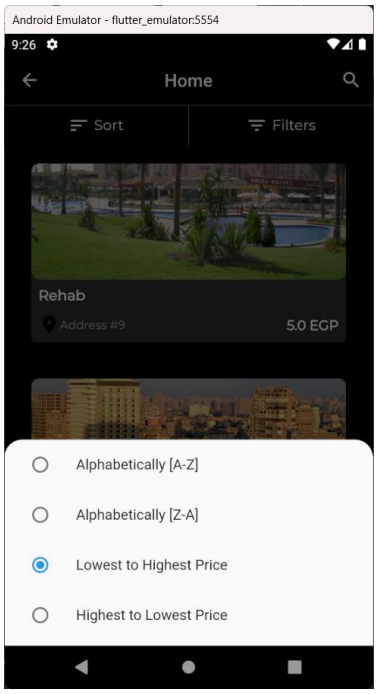


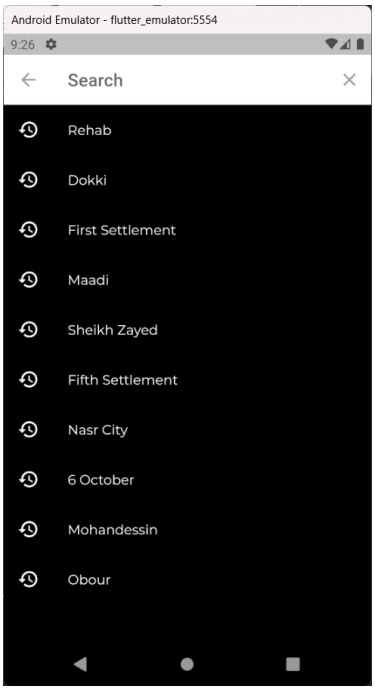


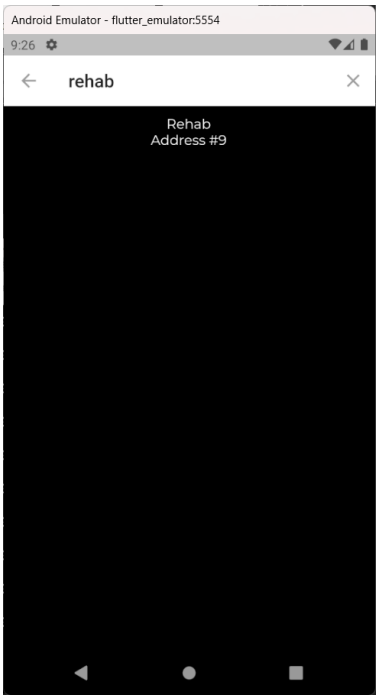


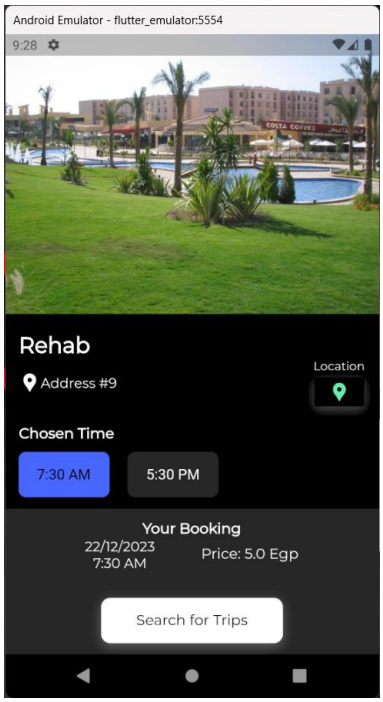


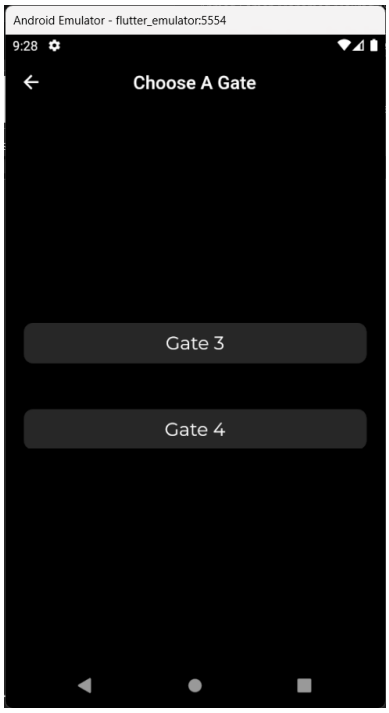

Android Emulator - flutter_emulator:5554
9:25
Home
Sort
Filters
6 October
Address #3
55.0 EGP
Dokki
Address #5
10.0 EGP

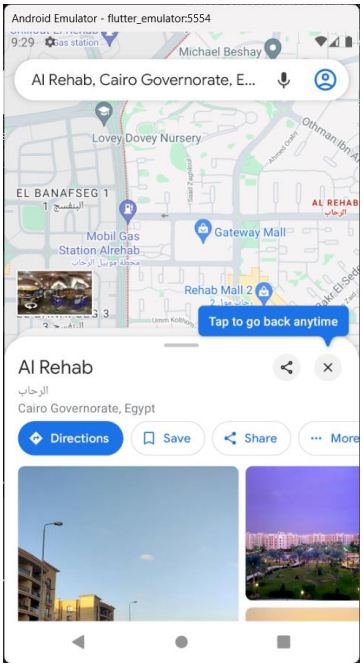

Android Emulator - flutter_emulator:5554
9:26
Home
Sort
Filters
Rehab
Address #9
5.0 EGP
Alphabetically [A-Z]
Alphabetically [Z-A]
Lowest to Highest Price
Highest to Lowest Price

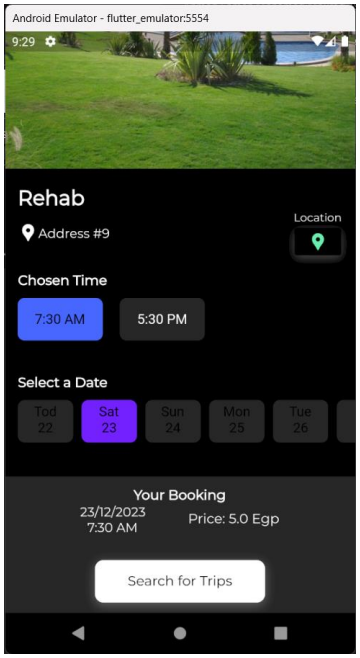

Android Emulator - flutter_emulator:5554
9:26
Search
Rehab
Dokki
First Settlement
Maadi
Sheikh Zayed
Fifth Settlement
Nasr City
6 October
Mohandessin
Obour

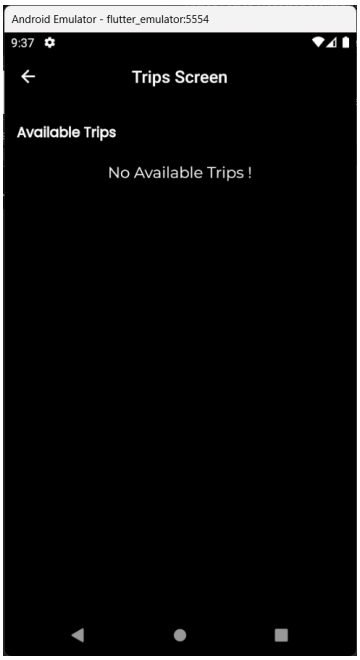

Android Emulator - flutter_emulator:5554
9:26
rehab
Rehab
Address #9

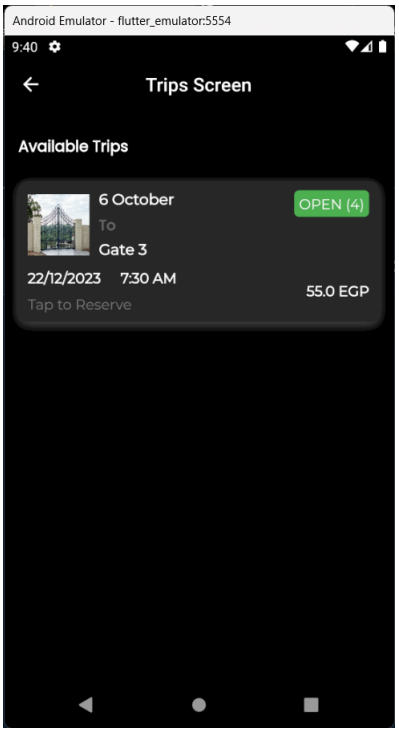

Android Emulator - flutter_emulator:5554
9:28
Rehab
Address #9
Location
Chosen Time
7:30 AM
5:30 PM
Your Booking
22/12/2023
7:30 AM
Price: 5.0 Egp
Search for Trips

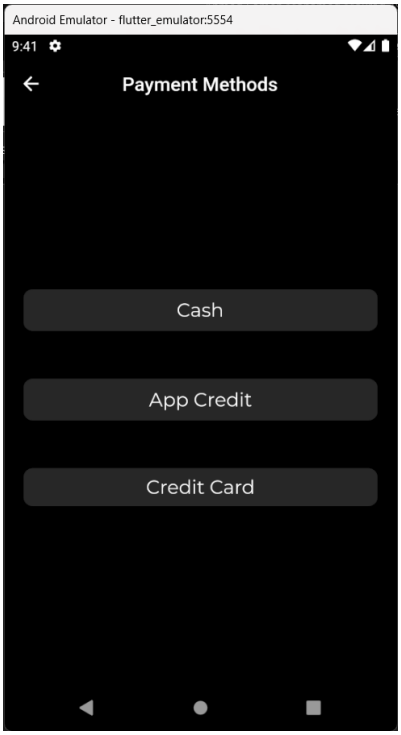

Android Emulator - flutter_emulator:5554
9:28
Choose A Gate
Gate 3
Gate 4

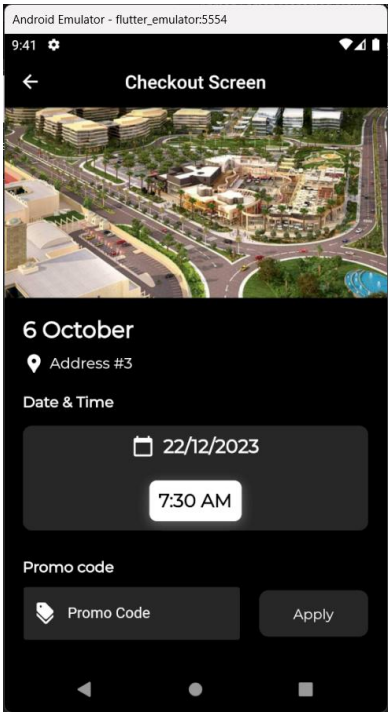

Android Emulator - flutter_emulator:5554
Al Rehab, Cairo Governorate, E...
Lovey Dovey Nursery
EL BANAFSEG 1
Mobil Gas Station Alrehab
Gateway Mall
Rehab Mall 2
Tap to go back anytime
Al Rehab
Cairo Governorate, Egypt
Directions
Save
Share

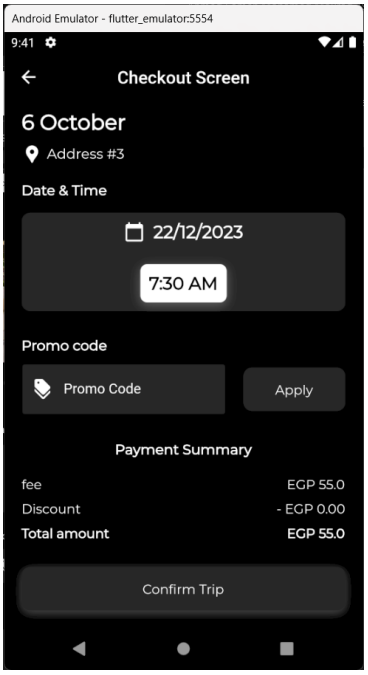

Android Emulator - flutter_emulator:5554
Rehab
Address #9
Location
Chosen Time
7:30 AM
5:30 PM
Select a Date
Sat 23
Your Booking
23/12/2023
7:30 AM
Price: 5.0 Egp
Search for Trips

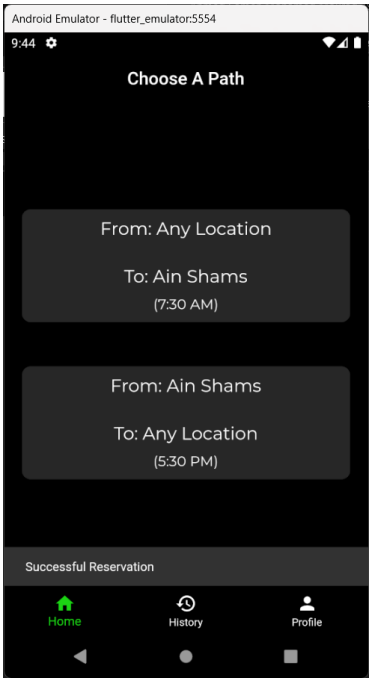

Android Emulator - flutter_emulator:5554
Trips Screen
Available Trips
No Available Trips !

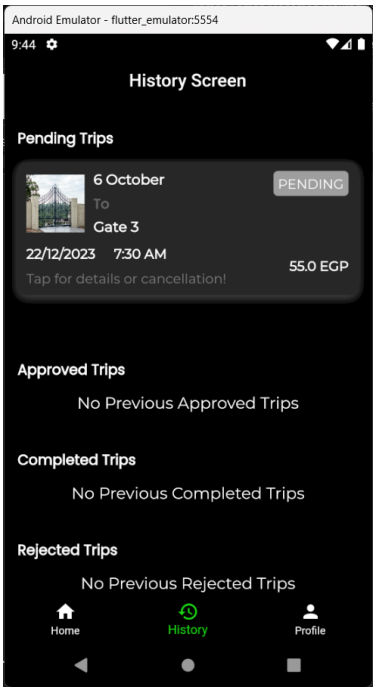

Android Emulator - flutter_emulator:5554
Trips Screen
Available Trips
6 October
To
Gate 3
OPEN (4)
22/12/2023 7:30 AM
Tap to Reserve
55.0 EGP

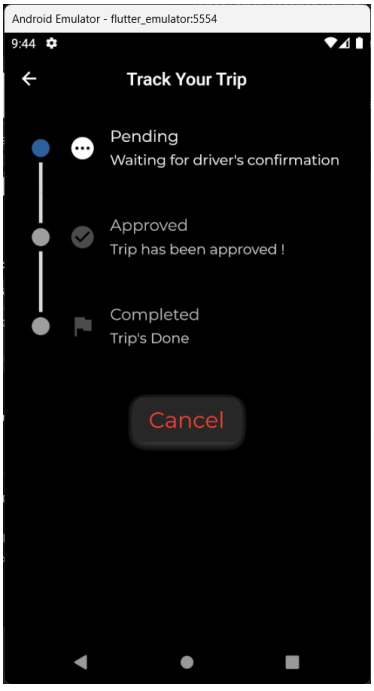

Android Emulator - flutter_emulator:5554
Payment Methods
Cash
App Credit
Credit Card

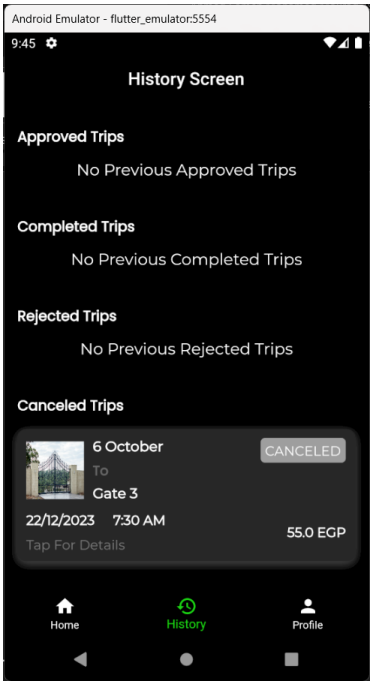

Android Emulator - flutter_emulator:5554
Checkout Screen
6 October
Address #3
Date & Time
22/12/2023
7:30 AM
Promo code
Promo Code
Apply

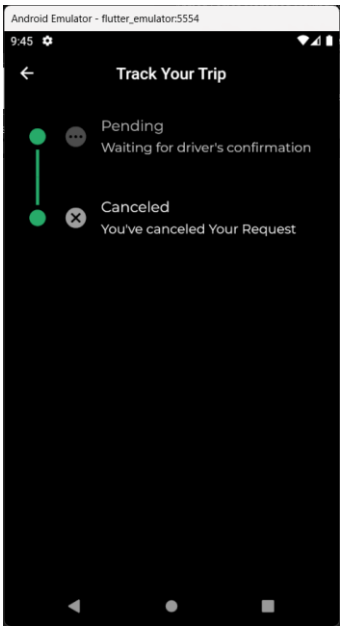

Android Emulator - flutter_emulator:5554
9:41
Checkout Screen
6 October
Address #3
Date & Time
22/12/2023
7:30 AM
Promo code
Promo Code
Apply
Payment Summary
fee
EGP 55.0
Discount
- EGP 0.00
Total amount
EGP 55.0
Confirm Trip

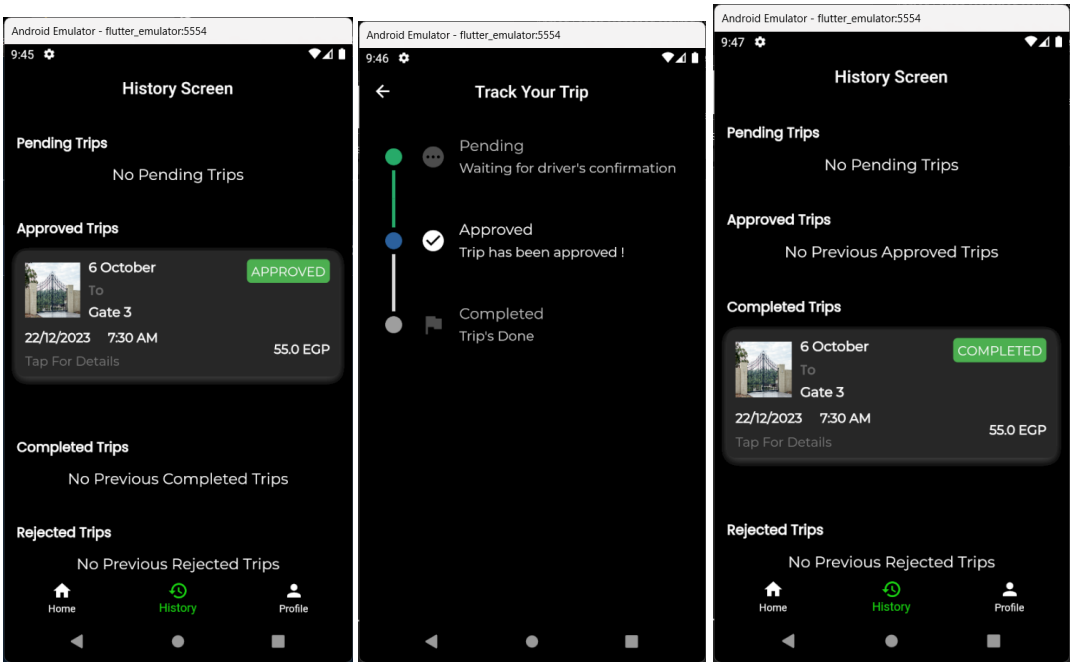

Android Emulator - flutter_emulator:5554
9:44
Choose A Path
From: Any Location
To: Ain Shams
(7:30 AM)
From: Ain Shams
To: Any Location
(5:30 PM)
Successful Reservation
Home
History
Profile

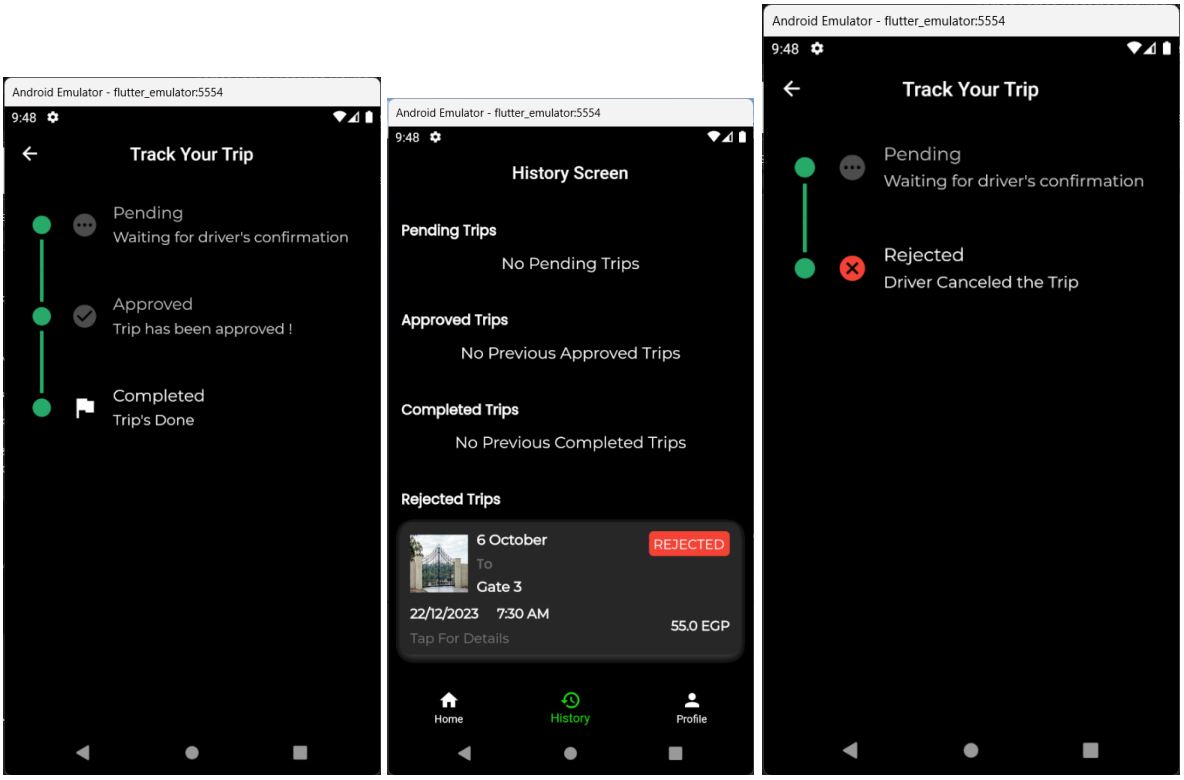

Android Emulator - flutter_emulator:5554
9:44
History Screen
Pending Trips
6 October
To
Gate 3
PENDING
22/12/2023 7:30 AM
Tap for details or cancellation!
55.0 EGP
Approved Trips
No Previous Approved Trips
Completed Trips
No Previous Completed Trips
Rejected Trips
No Previous Rejected Trips
Home
History
Profile


Android Emulator - flutter_emulator:5554
9:44
Track Your Trip
Pending
Waiting for driver's confirmation
Approved
Trip has been approved !
Completed
Trip's Done
Cancel

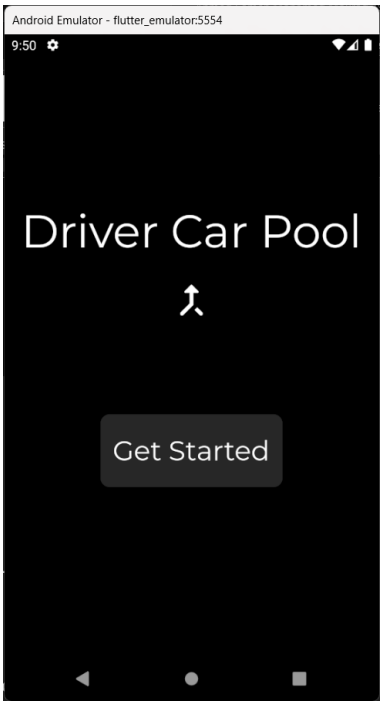

Android Emulator - flutter_emulator:5554
9:45
History Screen
Approved Trips
No Previous Approved Trips
Completed Trips
No Previous Completed Trips
Rejected Trips
No Previous Rejected Trips
Canceled Trips
6 October
To
Gate 3
CANCELED
22/12/2023 7:30 AM
Tap For Details
55.0 EGP
Home
History
Profile

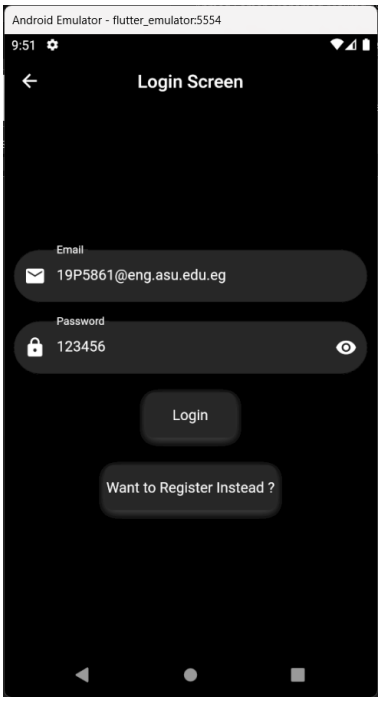

Android Emulator - flutter_emulator:5554
9:45
Track Your Trip
Pending
Waiting for driver's confirmation
Canceled
You've canceled Your Request

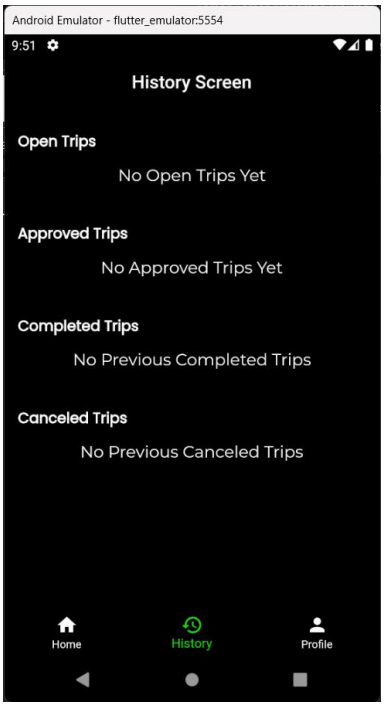

Android Emulator - flutter_emulator:5554
9:45
History Screen
Pending Trips
No Pending Trips
Approved Trips
6 October
To
Gate 3
APPROVED
22/12/2023 7:30 AM
Tap For Details
55.0 EGP
Completed Trips
No Previous Completed Trips
Rejected Trips
No Previous Rejected Trips
Home
History
Profile
Android Emulator - flutter_emulator:5554
9:46
Track Your Trip
Pending
Waiting for driver's confirmation
Approved
Trip has been approved !
Completed
Trip's Done
Android Emulator - flutter_emulator:5554
9:47
History Screen
Pending Trips
No Pending Trips
Approved Trips
No Previous Approved Trips
Completed Trips
6 October
To
Gate 3
COMPLETED
22/12/2023 7:30 AM
Tap For Details
55.0 EGP
Rejected Trips
No Previous Rejected Trips
Home
History
Profile

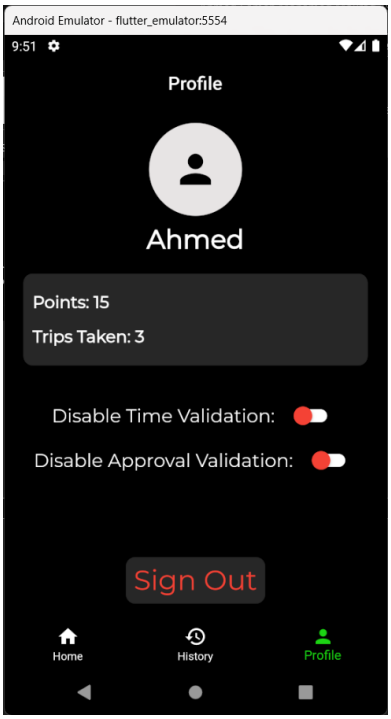

Android Emulator - flutter_emulator:5554
9:48
Track Your Trip
Pending
Waiting for driver's confirmation
Approved
Trip has been approved !
Completed
Trip's Done
Android Emulator - flutter_emulator:5554
9:48
History Screen
Pending Trips
No Pending Trips
Approved Trips
No Previous Approved Trips
Completed Trips
No Previous Completed Trips
Rejected Trips
6 October
To
Gate 3
REJECTED
22/12/2023 7:30 AM
Tap For Details
55.0 EGP
Home
History
Profile
Android Emulator - flutter_emulator:5554
9:48
Track Your Trip
Pending
Waiting for driver's confirmation
Rejected
Driver Canceled the Trip

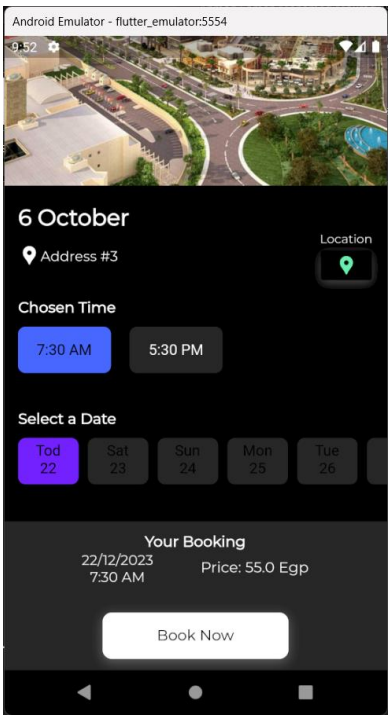
UI Design (Driver)

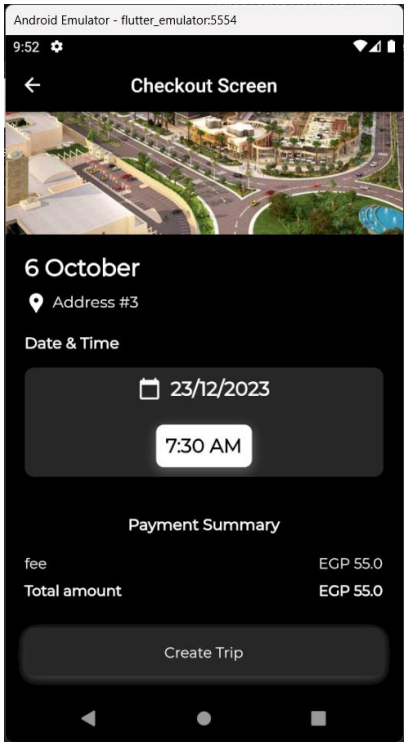


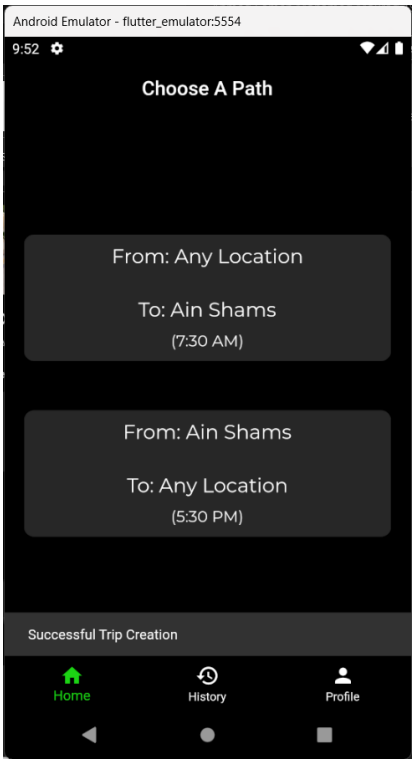


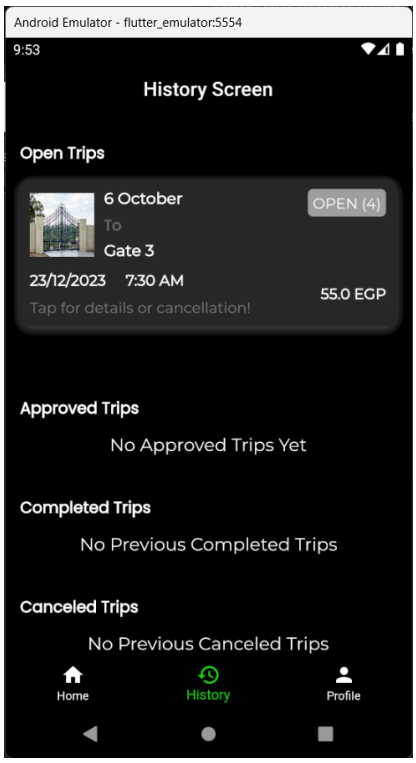


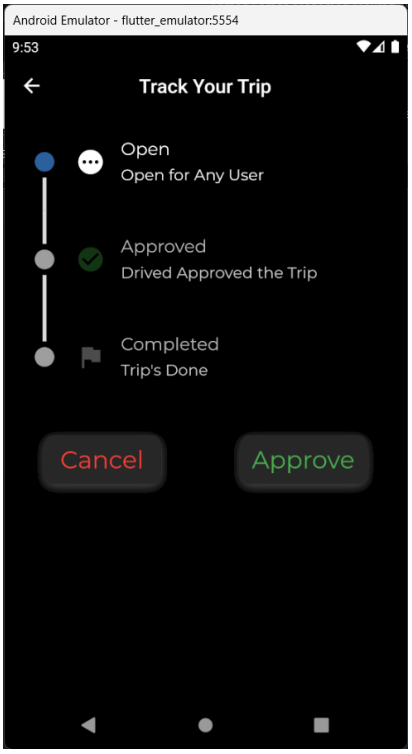


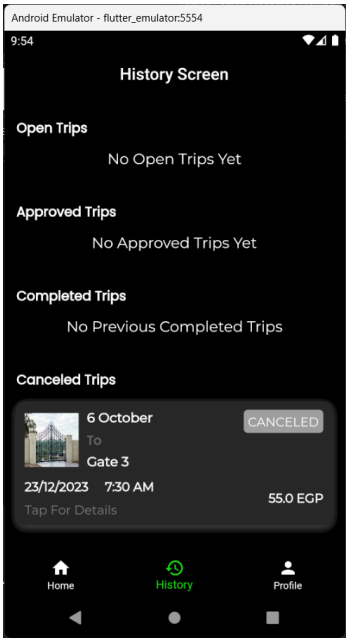


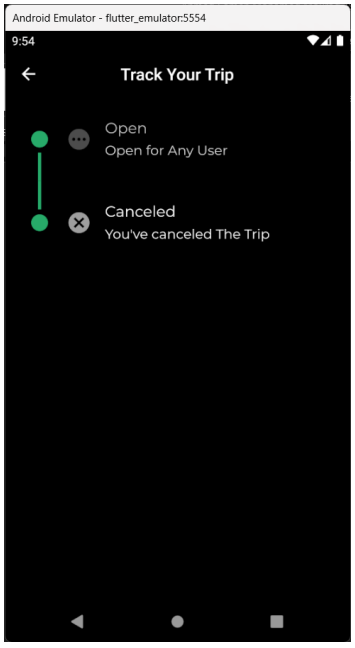


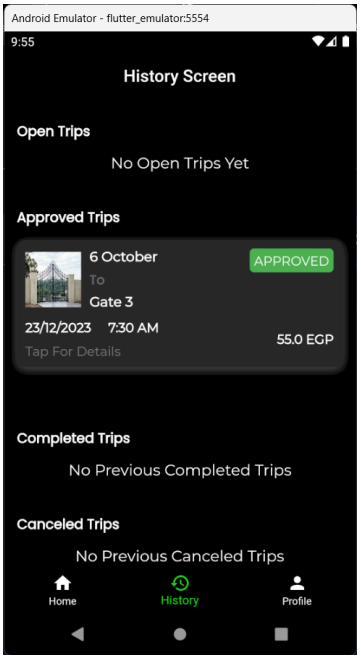

Android Emulator - flutter_emulator:5554
9:52
Choose A Path
From: Any Location
To: Ain Shams
(7:30 AM)
From: Ain Shams
To: Any Location
(5:30 PM)
Successful Trip Creation
Home
History
Profile

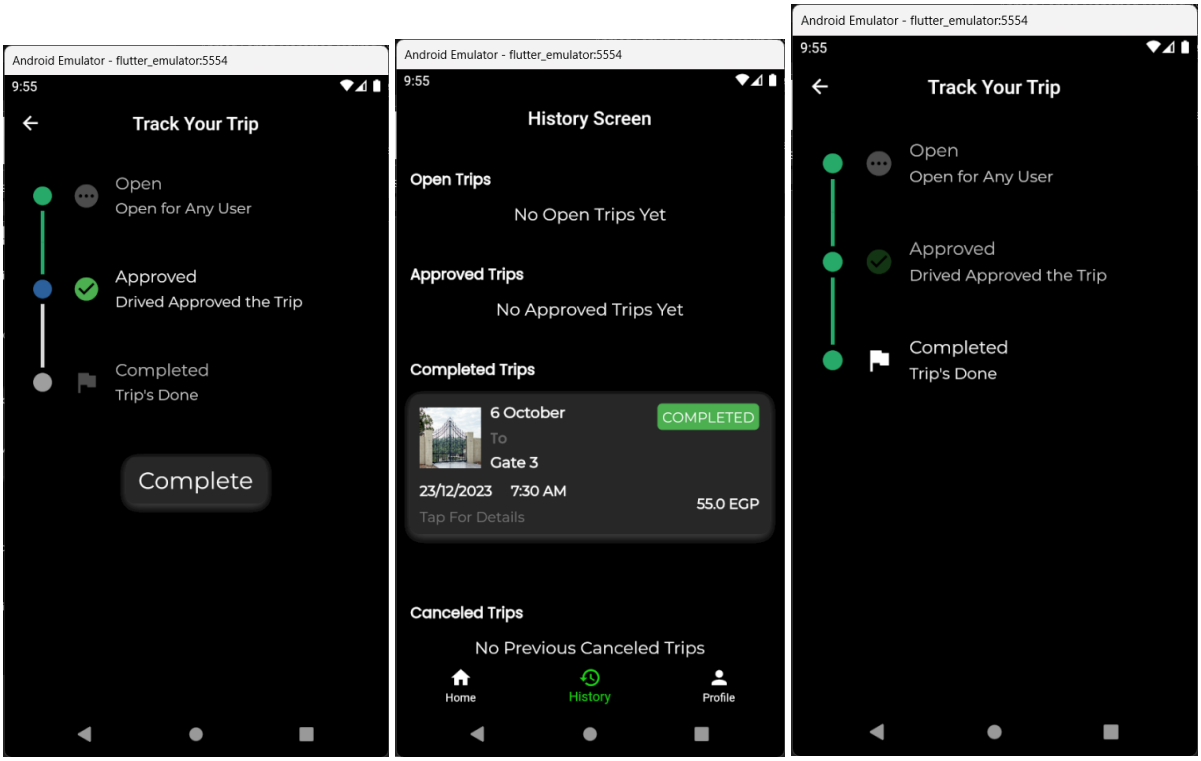

Android Emulator - flutter_emulator:5554
9:53
History Screen
Open Trips
6 October
To
Gate 3
OPEN (4)
23/12/2023 7:30 AM
Tap for details or cancellation!
55.0 EGP
Approved Trips
No Approved Trips Yet
Completed Trips
No Previous Completed Trips
Canceled Trips
No Previous Canceled Trips
Home
History
Profile


Android Emulator - flutter_emulator:5554
9:53
Track Your Trip
Open
Open for Any User
Approved
Drived Approved the Trip
Completed
Trip's Done
Cancel
Approve


Android Emulator - flutter_emulator:5554
9:54
History Screen
Open Trips
No Open Trips Yet
Approved Trips
No Approved Trips Yet
Completed Trips
No Previous Completed Trips
Canceled Trips
6 October
To
Gate 3
CANCELED
23/12/2023 7:30 AM
Tap For Details
55.0 EGP
Home
History
Profile


Android Emulator - flutter_emulator:5554
9:54
Track Your Trip
Open
Open for Any User
Canceled
You've canceled The Trip


Android Emulator - flutter_emulator:5554
9:55
History Screen
Open Trips
No Open Trips Yet
Approved Trips
6 October
To
Gate 3
APPROVED
23/12/2023 7:30 AM
Tap For Details
55.0 EGP
Completed Trips
No Previous Completed Trips
Canceled Trips
No Previous Canceled Trips
Home
History
Profile



Database Structure

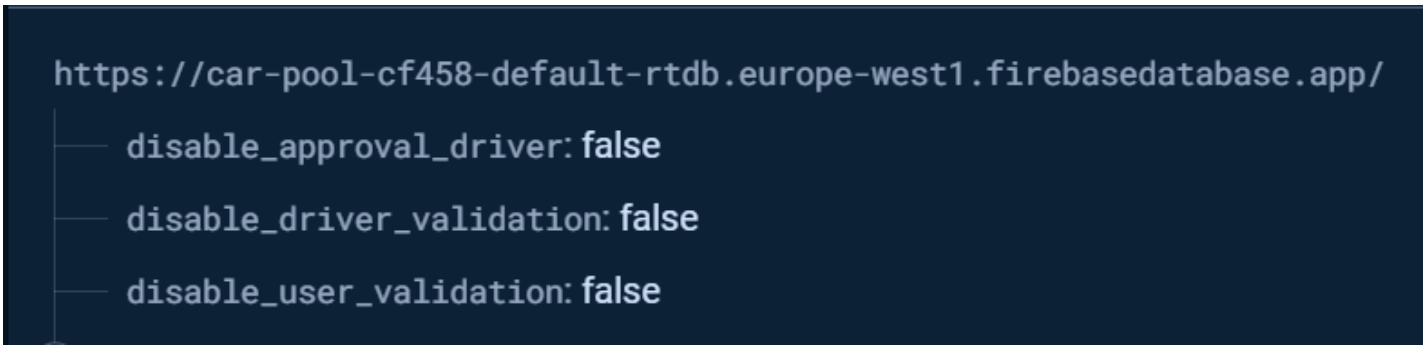
I've used the Realtime database provided by firebase, for its speed and real-time read/write operation.

The database consists of 5 main nodes, and 3 attributes.

Database View:

1- The 3 attributes are the “disable_user_validation” and “disable_driver_validation”, they are 3 Boolean attributes which controls the validation of the time constraints on both the user and driver apps, and controls the approval (confirmation) time constraint in the driver app; and all these Boolean values can be set through each app respectively.

Attributes View:



2- One of the main nodes is the “routes” node, it contains all the static routes that the driver can create a trip upon or a user can book a trip accordingly. It contains the route name, its address, its price, and its location on google maps which can be opened from both apps.

   It contains 10 routes across all over Egypt, each one is represented by a photo from that place

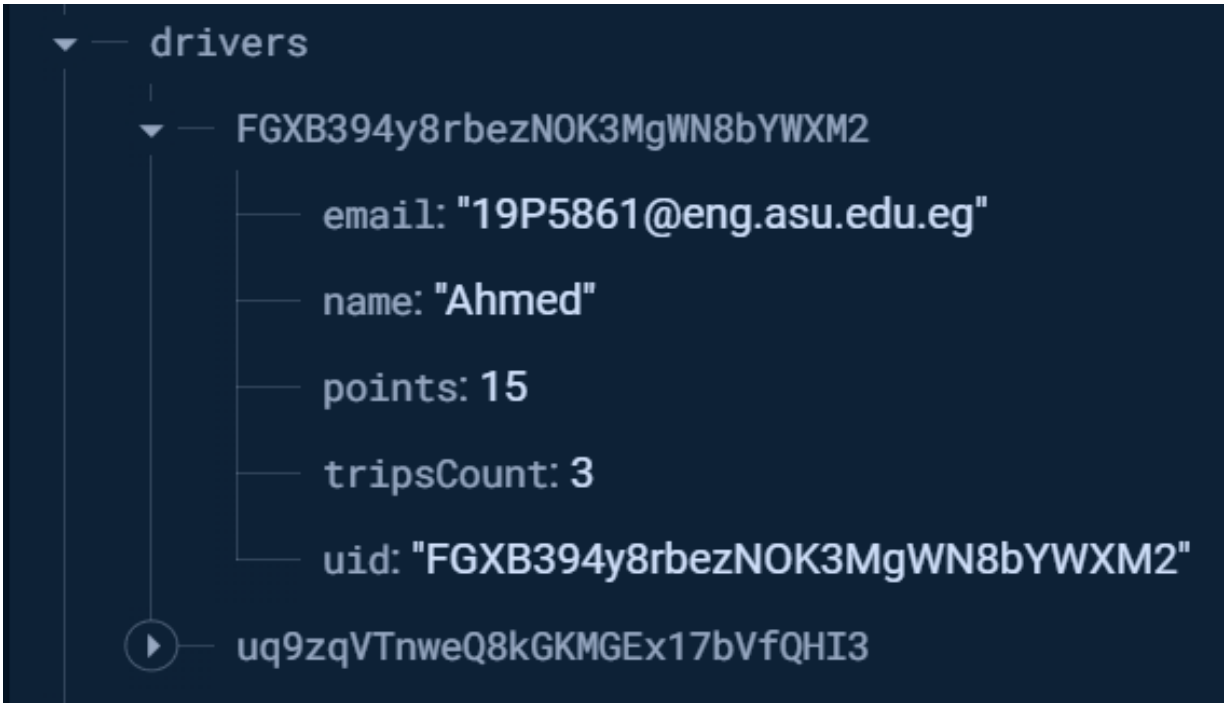
Routes View:

3- There is the “drivers” node which contains all the data about the registered drivers in the drivers app. It contains the name of the driver, his unique id, his trip count, his bonus points, and his raw email address.

   Each entry has the unique id as its key and the value is the drivers’ data.

   Each driver gets awarded 5 points when he successfully completes a trip, and the trip count increases by 1.

Drivers View:

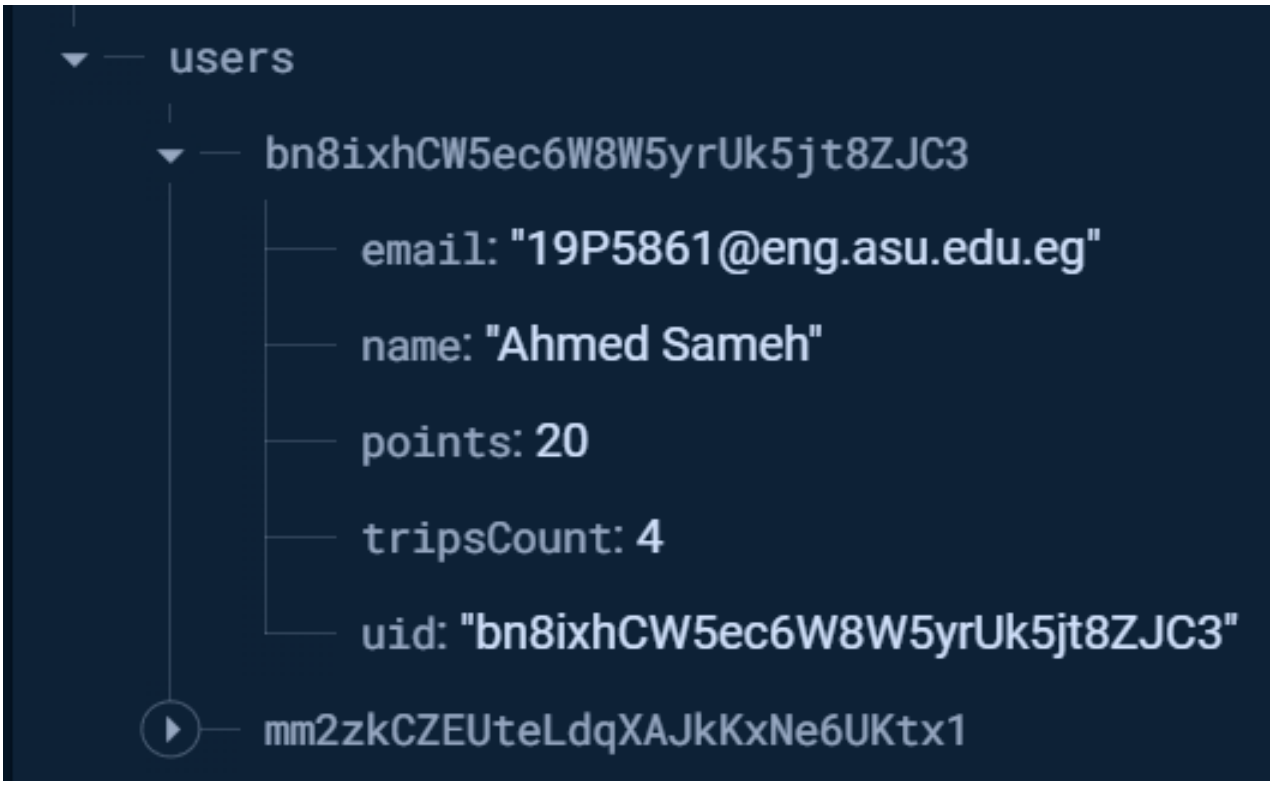


4- The “users” node contains all the data about the registered users in the users app. It contains the name of the user, his unique id, his trip count, his bonus points, and his raw email address.

   Each entry has the unique id as its key and the value is the users’ data.

   Each user gets awarded 5 points when he successfully completes a trip, and the trip count increases by 1.

Users View:



5- The user trips node contains all the trips that a certain user has subscribed to. It contains the trip id, source, destination, status, price, time and date (tripDate), the date when the user subscribed (currentDate) to the trip, and the driver id for this trip.

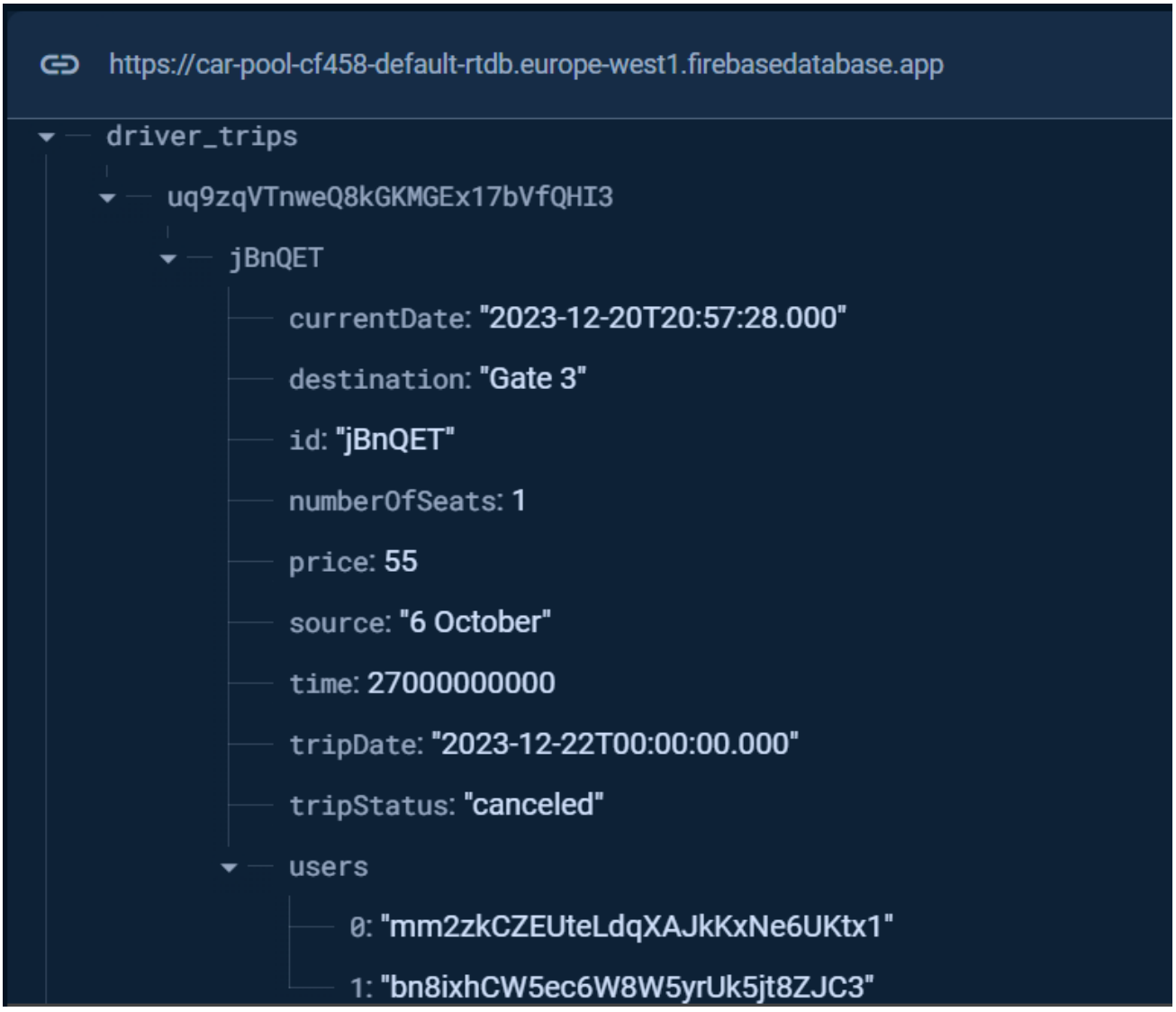
   Each user has his subscribed trips by his unique id as the key, and each unique id entry has one or many trips by the trip id as its key and the trip details for its values.

User Trips View:

6- The driver trips node contains all the trips that a certain driver has created. It contains the trip id, source, destination, status, price, time and date (tripDate), the date when the driver created the trip (currentDate), number of available seats, and the list of users subscribed to this specific trip.

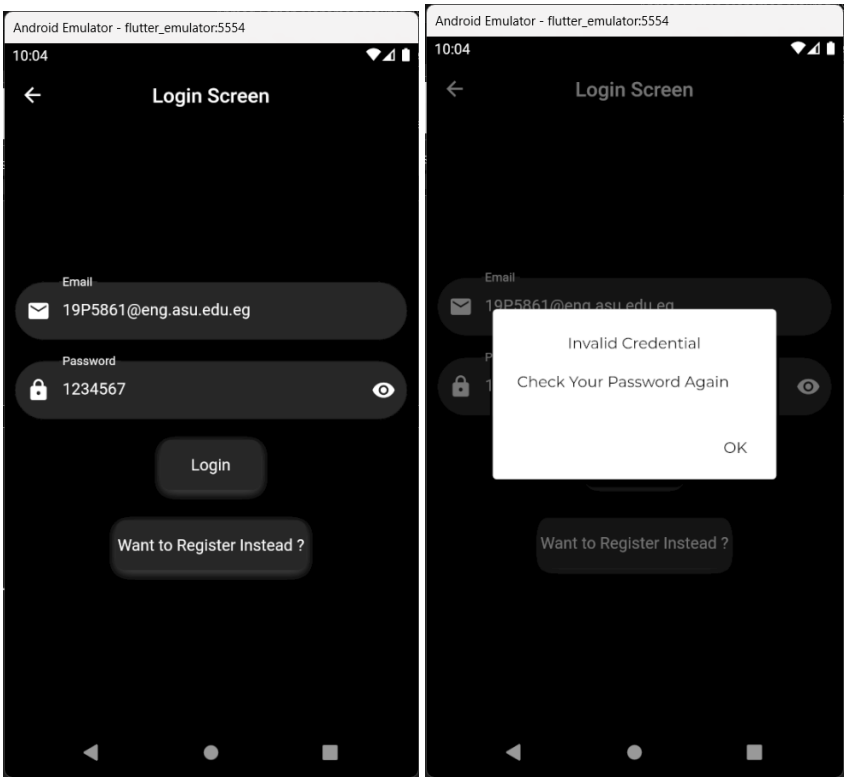
   Each driver has his created trips by his unique id as the key, and each unique id entry has one or many trips by the trip id as its key and the trip details for its values.

Driver Trips View:

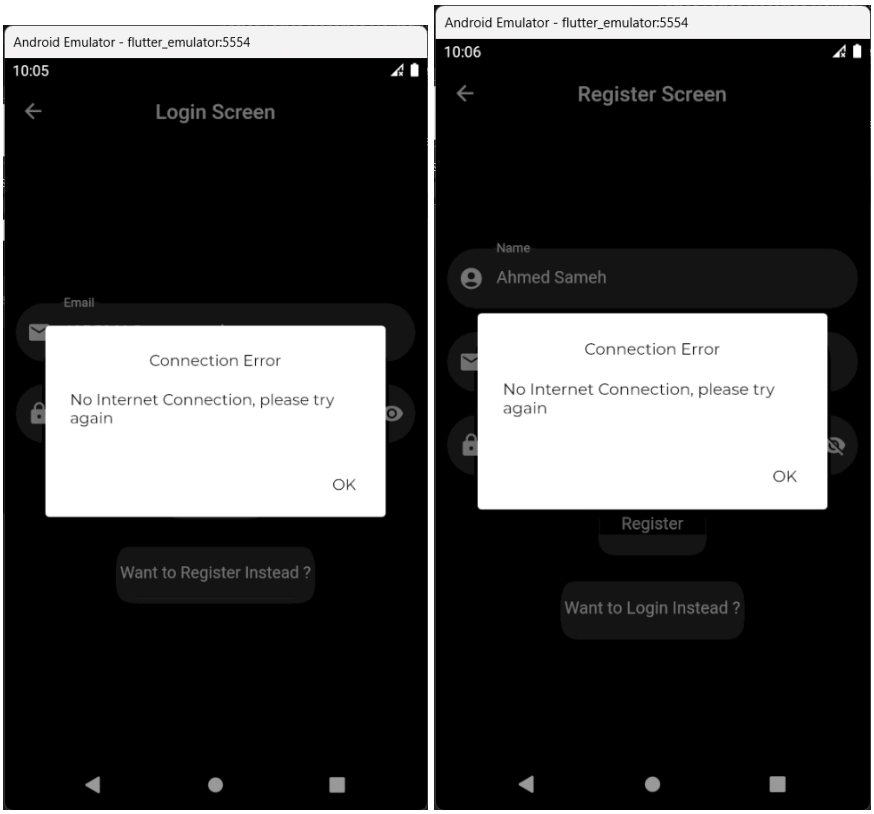


Test Cases (User & Driver)

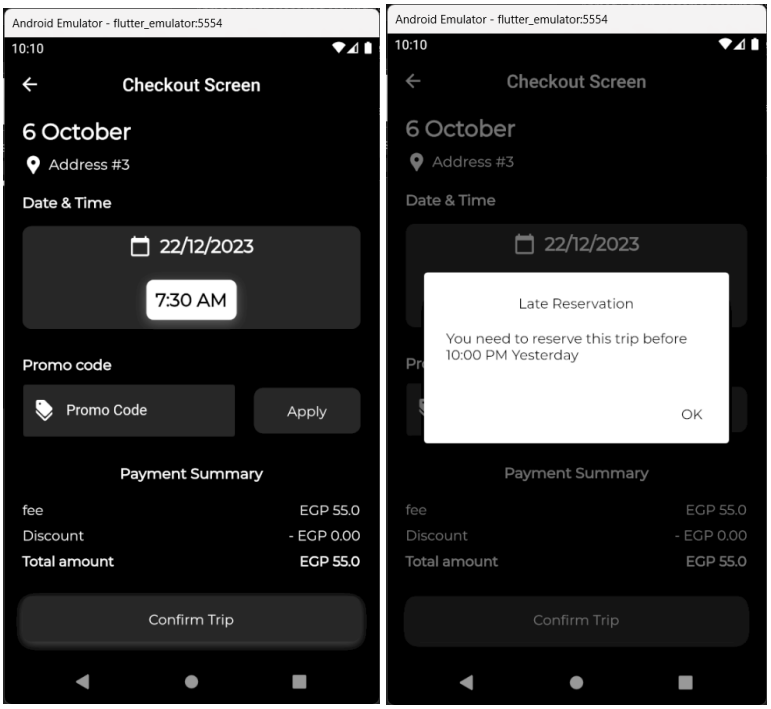
1- Wrong Password



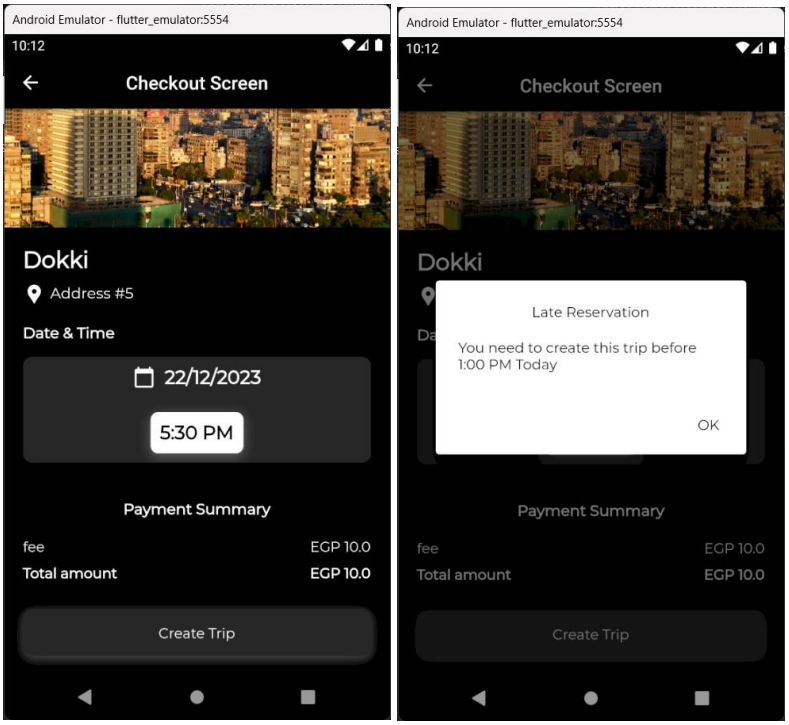
2- Offline Login & Registration



3- Reserving the 7:30 AM Trip after the time limit (User)

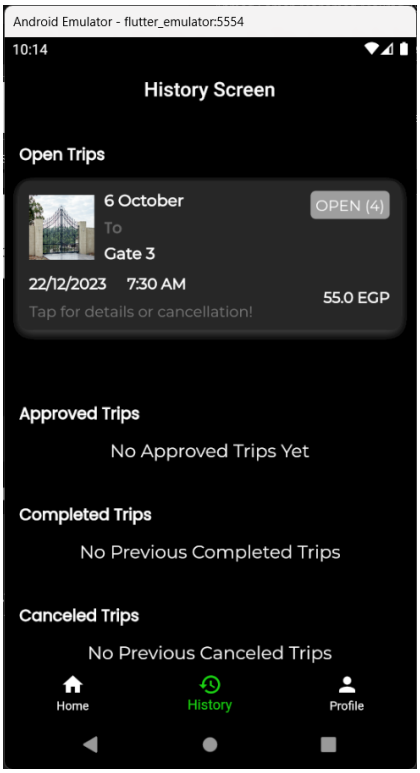


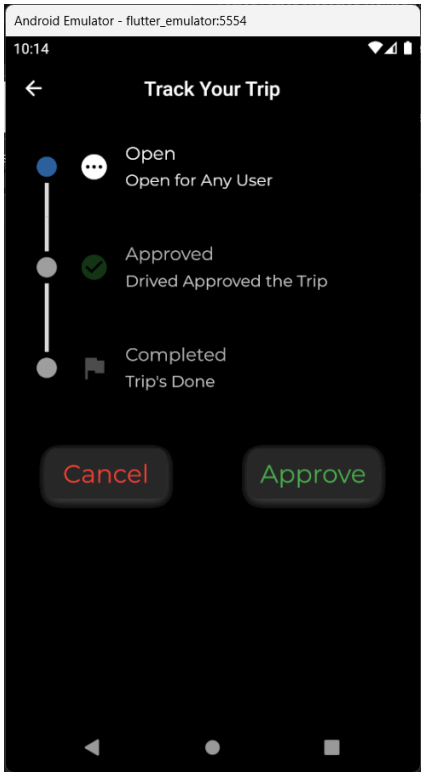
4- Creating the 5:30 PM trip after the time limit (Driver)

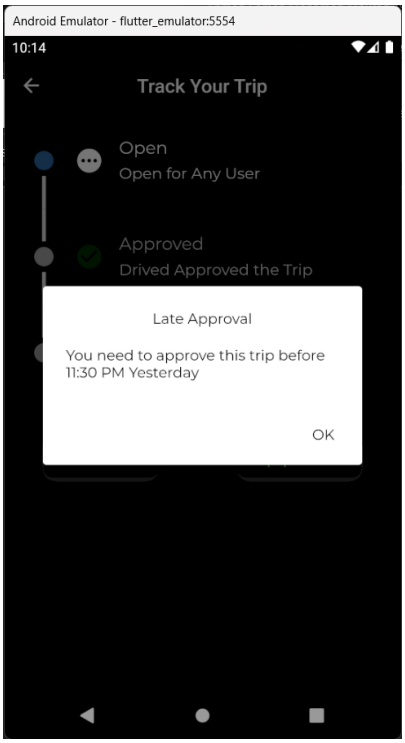


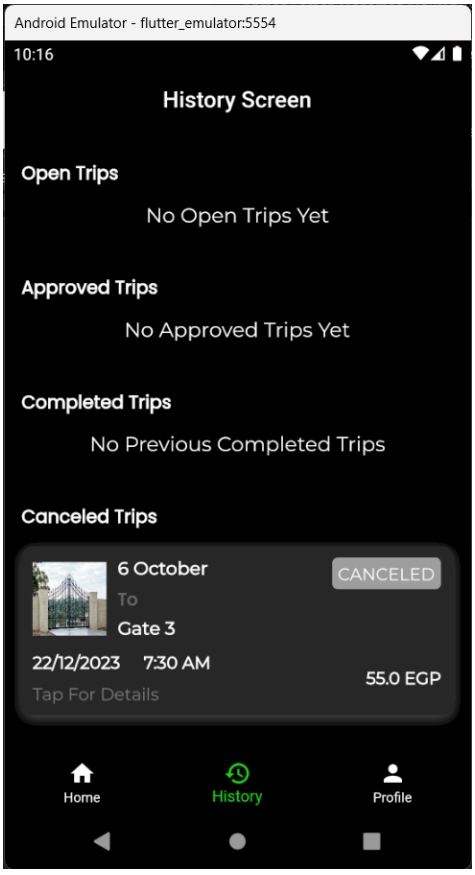
5- Approving (Confirming) a 7:30 AM trip after the time limit (Driver)

This trip gets cancelled automatically as a result.

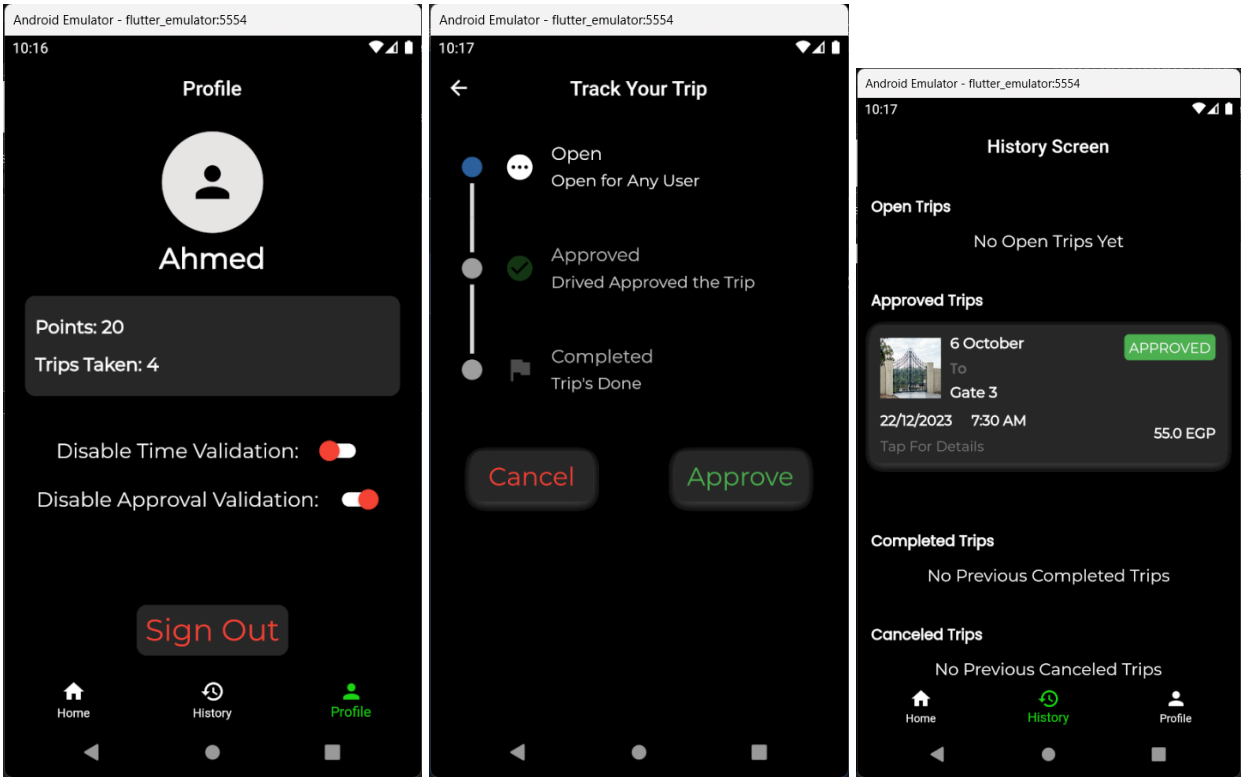

Android Emulator - flutter_emulator:5554
10:14
History Screen
Open Trips
6 October
To
Gate 3
OPEN (4)
22/12/2023 7:30 AM
55.0 EGP
Tap for details or cancellation!
Approved Trips
No Approved Trips Yet
Completed Trips
No Previous Completed Trips
Canceled Trips
No Previous Canceled Trips
Home
History
Profile


Android Emulator - flutter_emulator:5554
10:14
Track Your Trip
Open
Open for Any User
Approved
Drived Approved the Trip
Completed
Trip's Done
Cancel
Approve

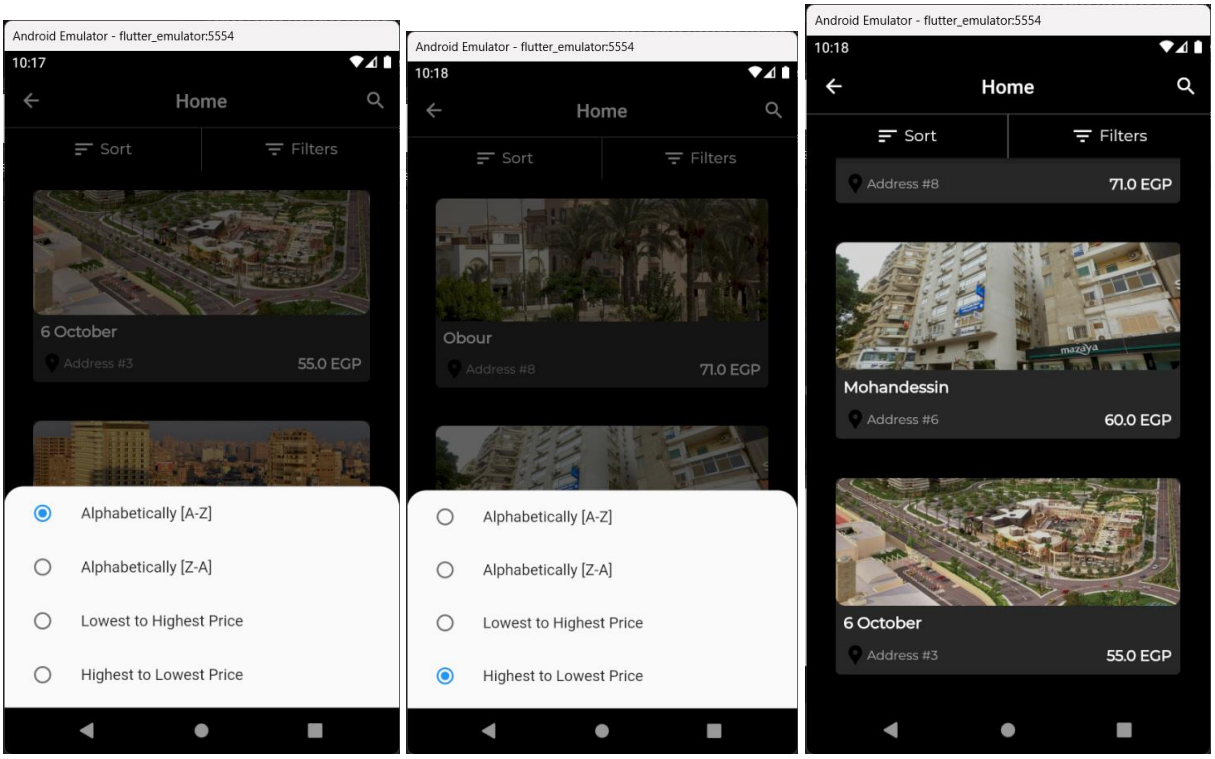

Android Emulator - flutter_emulator:5554
10:14
Track Your Trip
Open
Open for Any User
Approved
Drived Approved the Trip
Late Approval
You need to approve this trip before 11:30 PM Yesterday
OK


Android Emulator - flutter_emulator:5554
10:16
History Screen
Open Trips
No Open Trips Yet
Approved Trips
No Approved Trips Yet
Completed Trips
No Previous Completed Trips
Canceled Trips
6 October
To
Gate 3
CANCELED
22/12/2023 7:30 AM
55.0 EGP
Tap For Details
Home
History
Profile

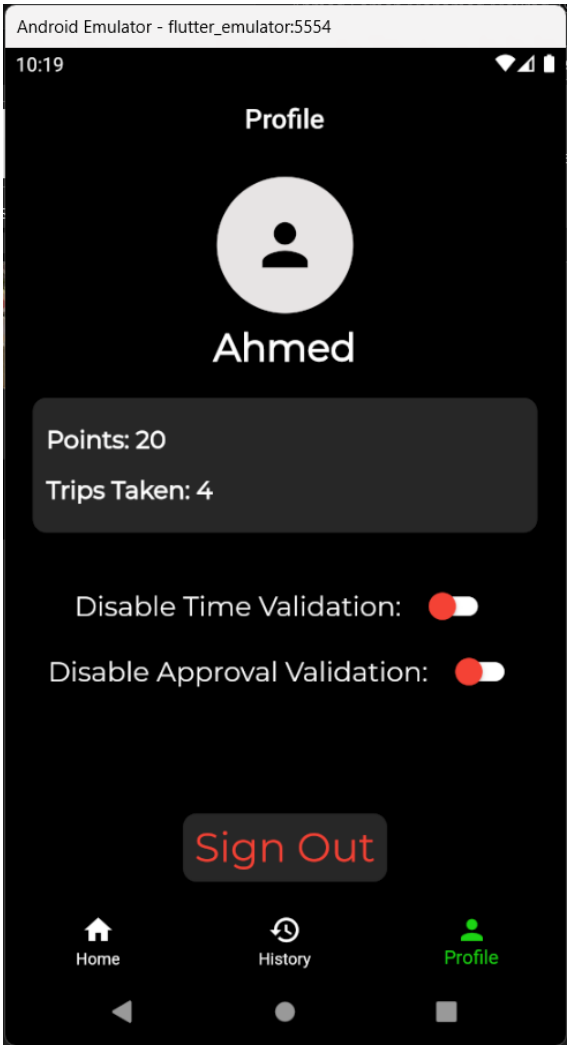
6- Disabling the approval validation (Driver)

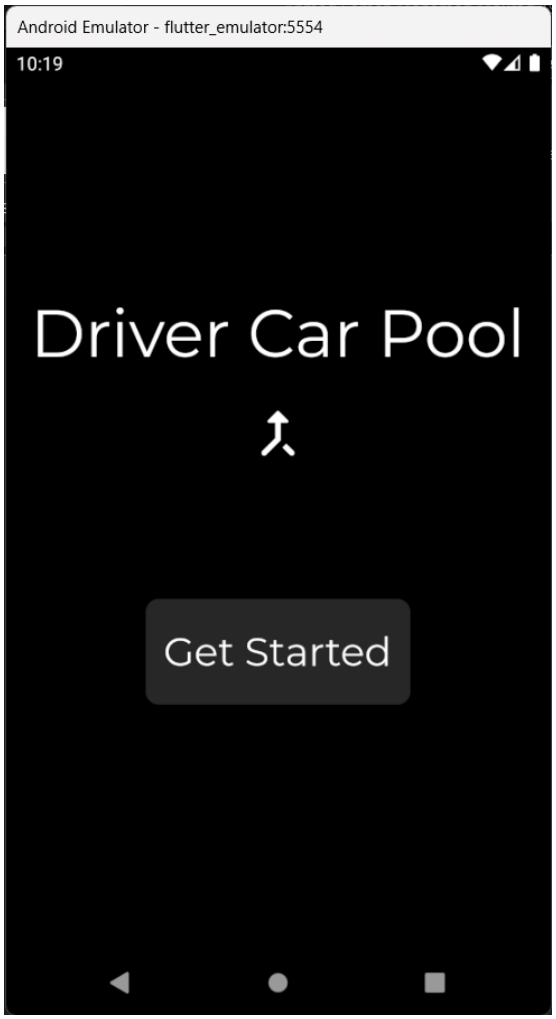


7- Sorting by highest price (Driver)

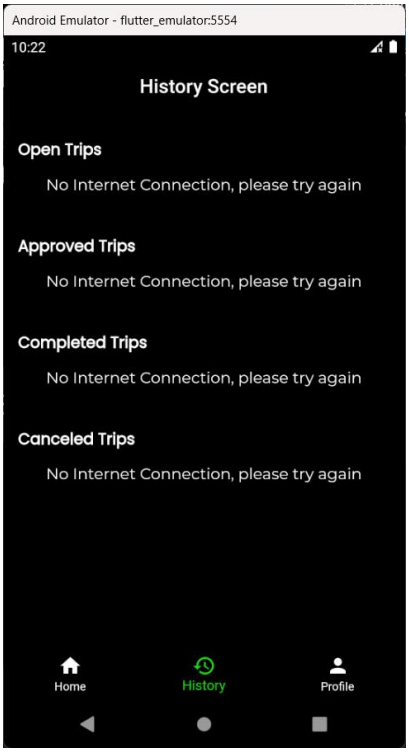


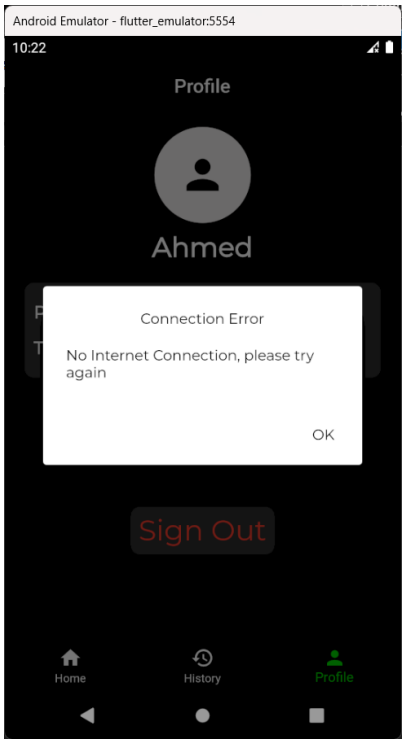
8- Signing out (Driver)

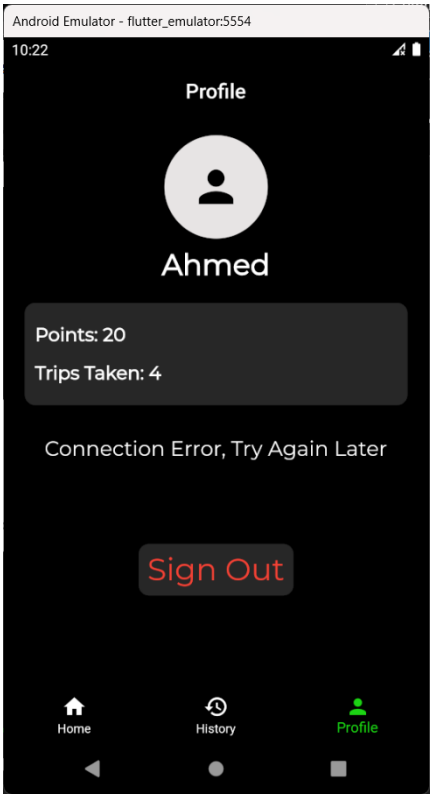


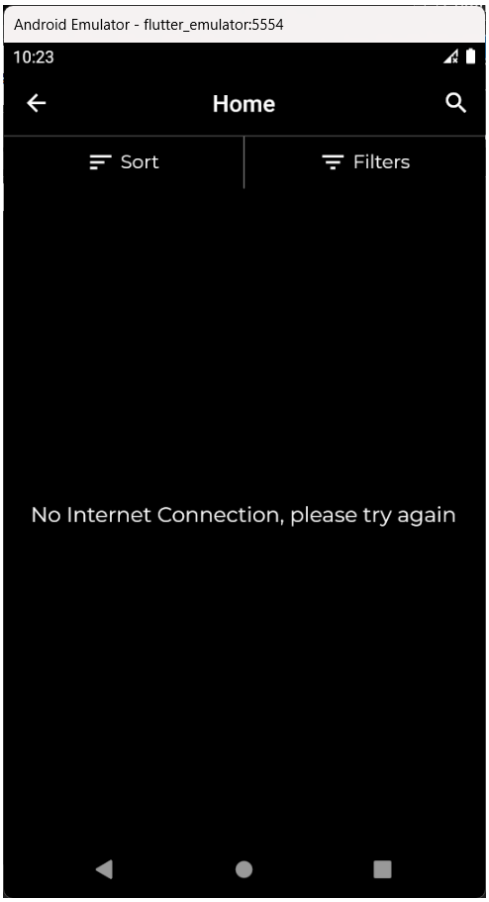


9- Different Screens on Offline Mode (Driver)

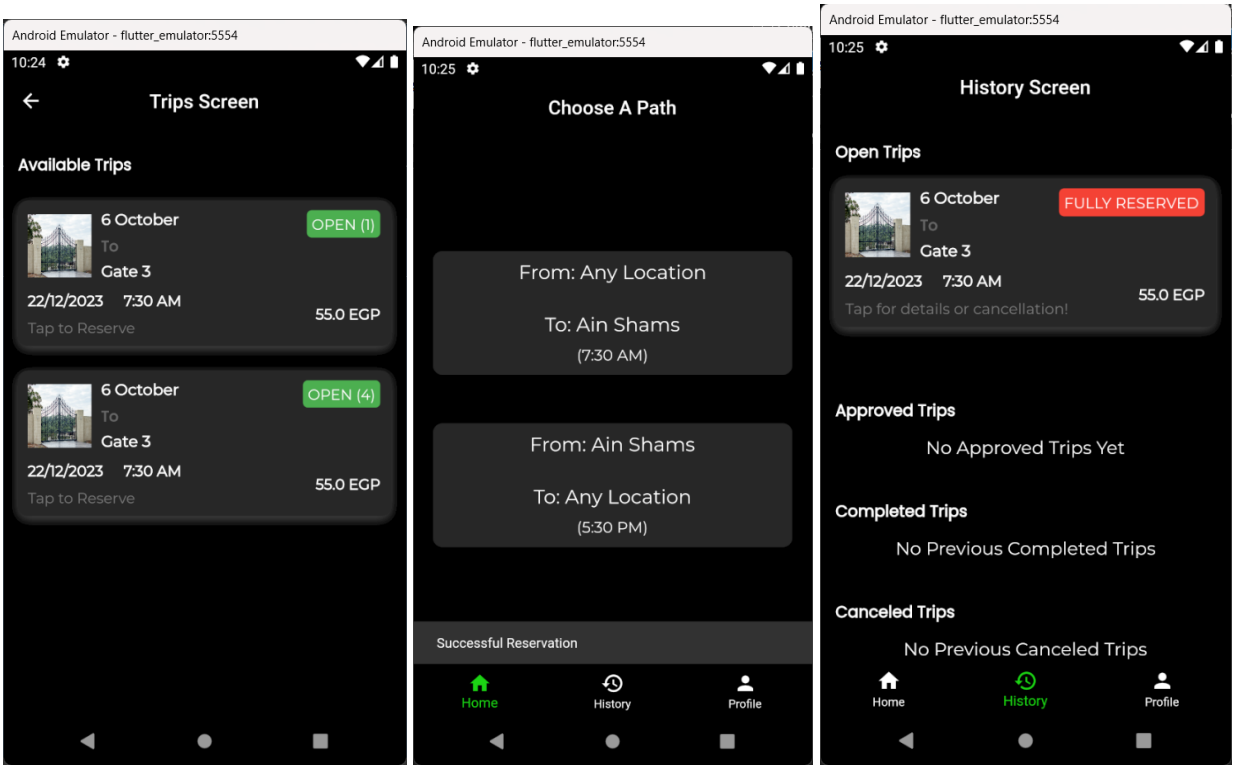

Android Emulator - flutter_emulator:5554
10:22
History Screen
Open Trips
No Internet Connection, please try again
Approved Trips
No Internet Connection, please try again
Completed Trips
No Internet Connection, please try again
Canceled Trips
No Internet Connection, please try again
Home
History
Profile


Android Emulator - flutter_emulator:5554
10:22
Profile
Ahmed
Connection Error
No Internet Connection, please try again
OK
Sign Out
Home
History
Profile

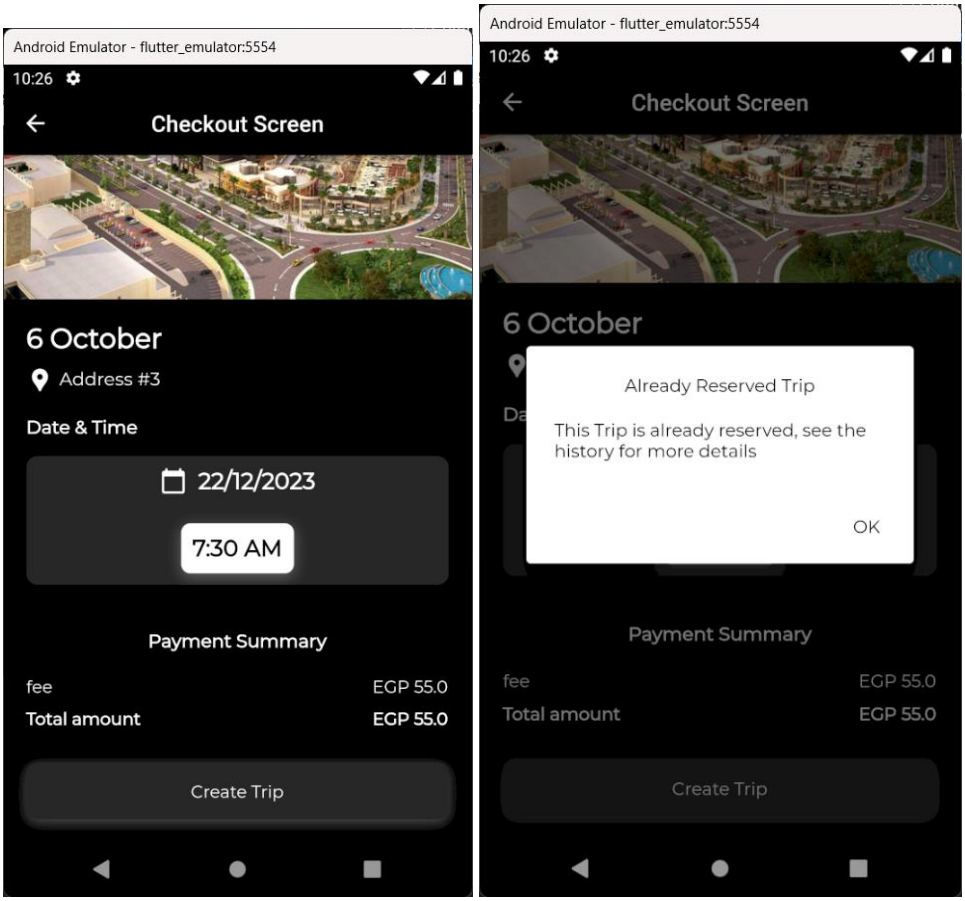

Android Emulator - flutter_emulator:5554
10:22
Profile
Ahmed
Points: 20
Trips Taken: 4
Connection Error, Try Again Later
Sign Out
Home
History
Profile


Android Emulator - flutter_emulator:5554
10:23
Home
Sort
Filters
No Internet Connection, please try again

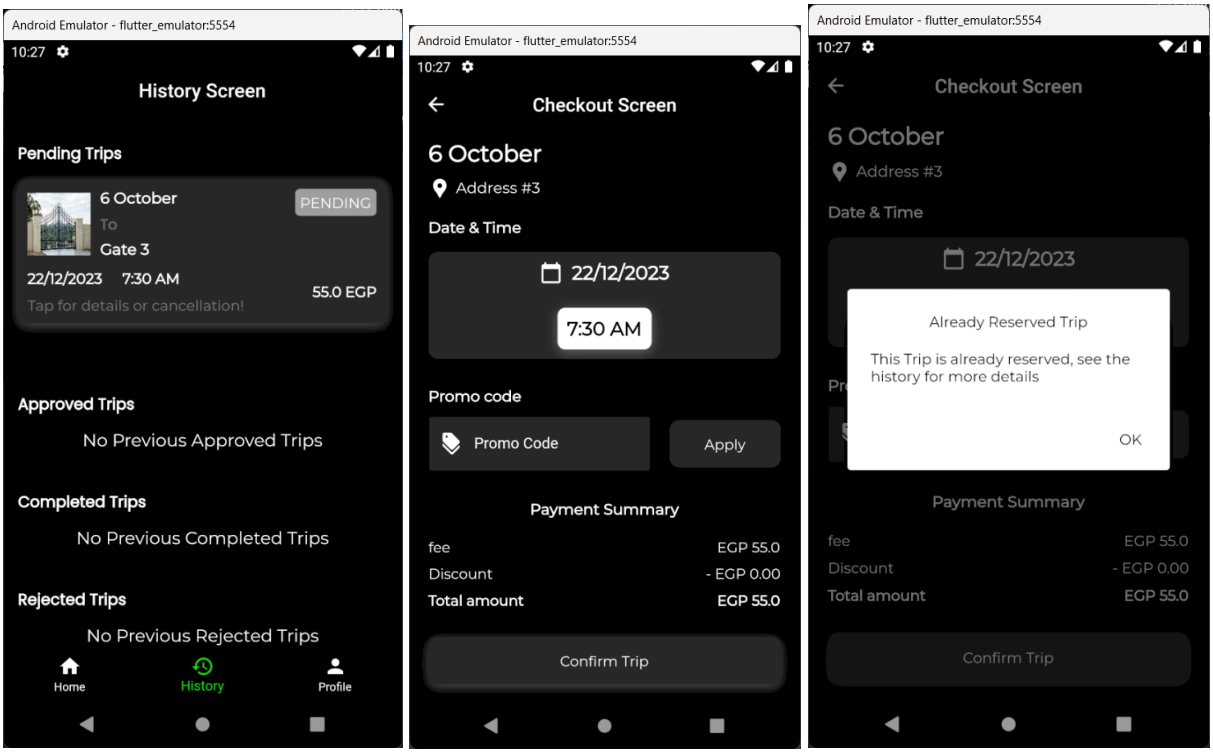
10- Reserving a trip with only 1 seat left (User & Driver)



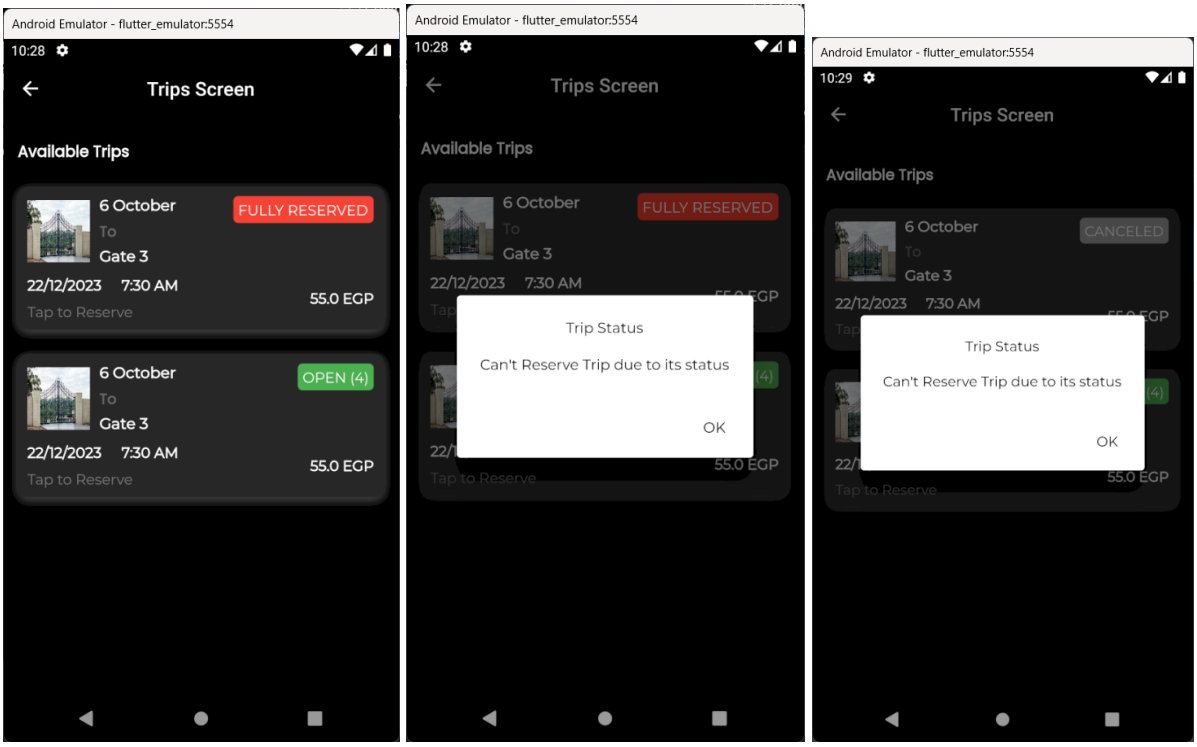
11- Driver creating the same trip



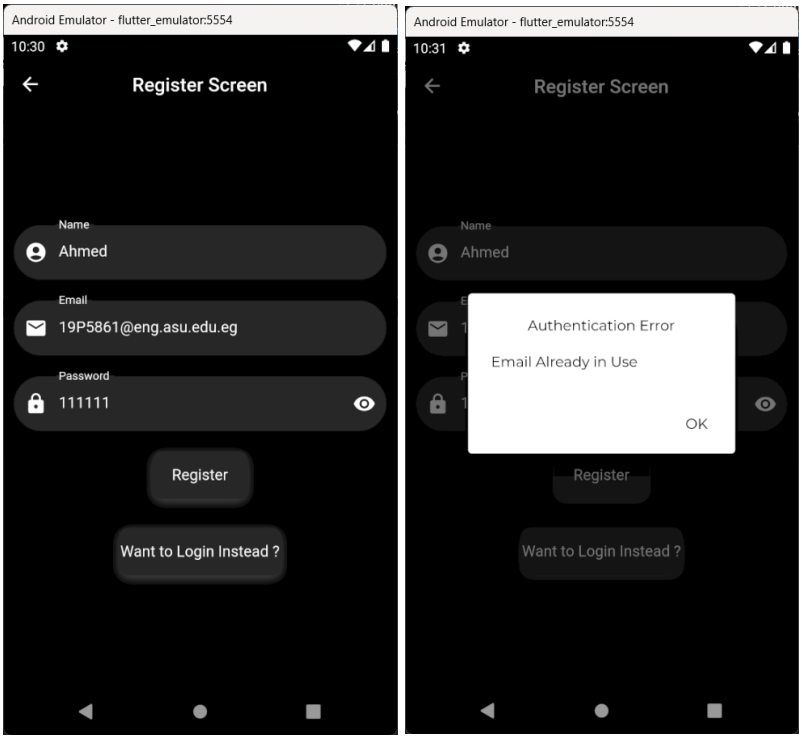
12- User subscribing to the same trip.



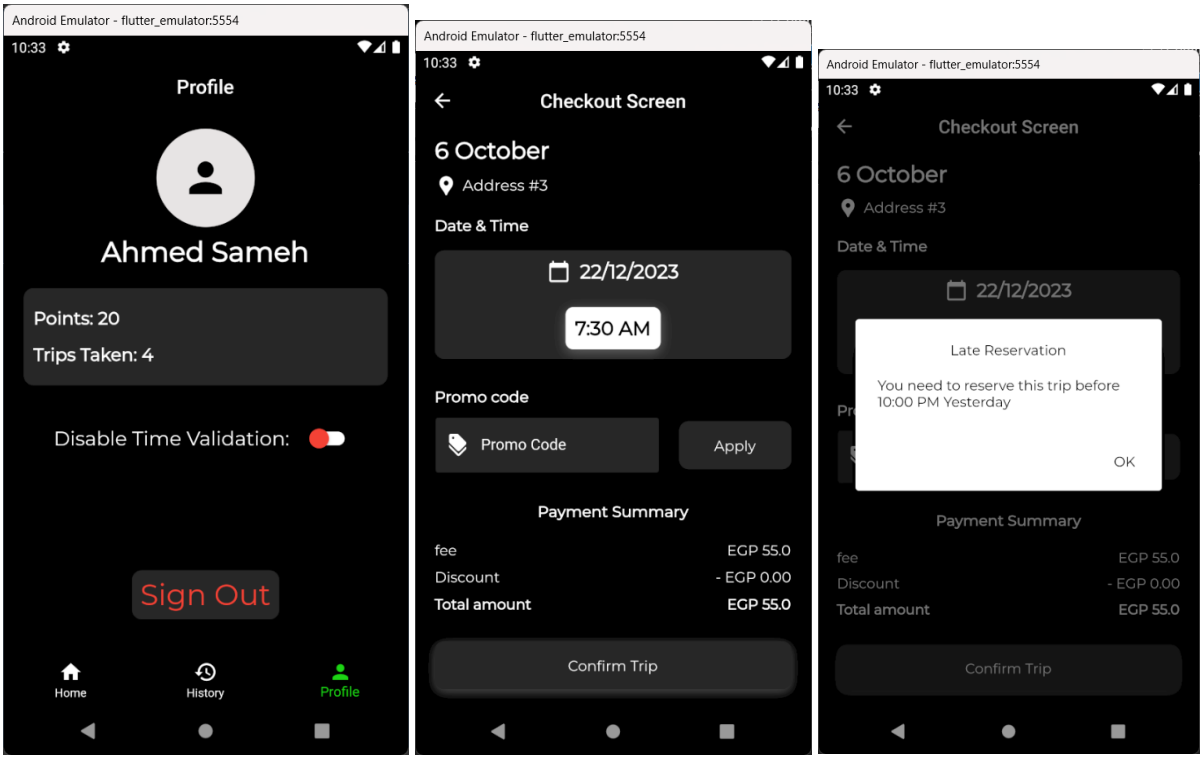
13- Reserving a fully reserved or cancelled trip (User)

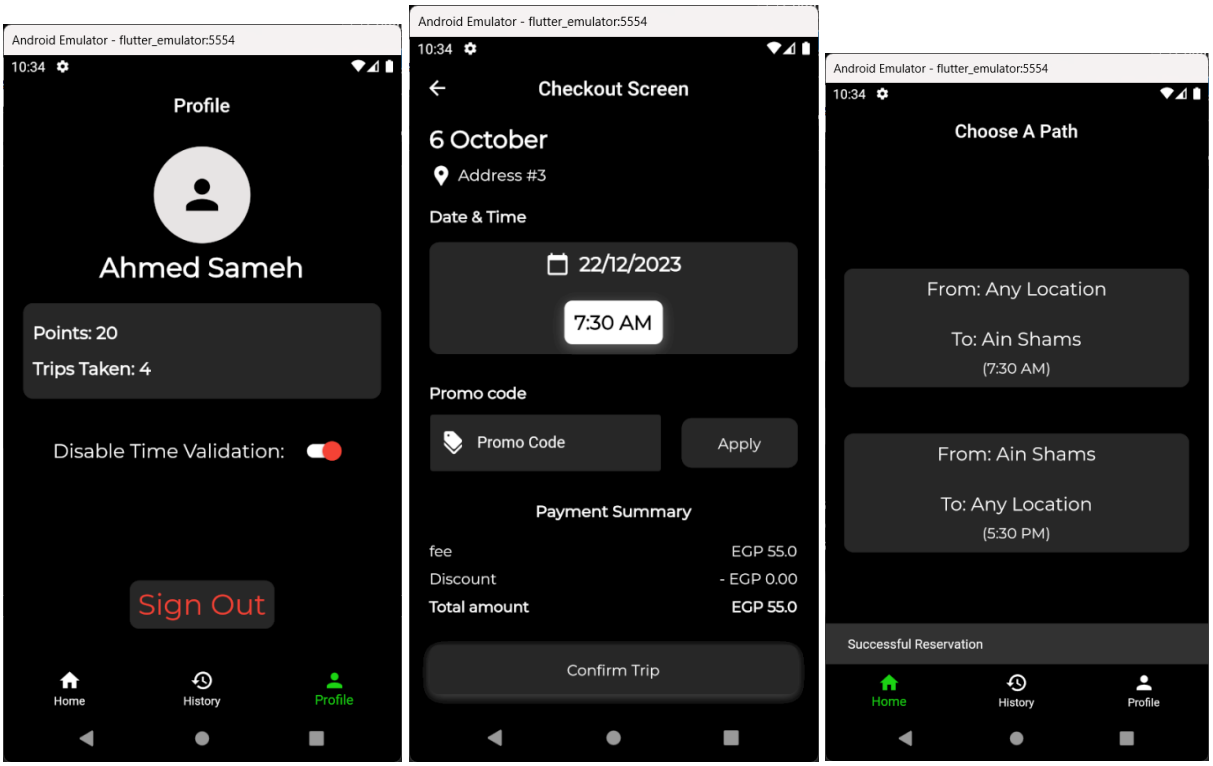


14- Registering an already existing account (User)



15- Bypassing the subscribe time constraints (User)





Realtime DB Code (User & Driver)

User Code

```dart
import 'package:car_pool_app/Services/lookup.dart';
import 'package:car_pool_app/Services/errors.dart';
import 'package:car_pool_app/Model%20Classes/custom_route.dart';
import 'package:car_pool_app/Model%20Classes/user.dart';
import 'package:car_pool_app/Model%20Classes/trip.dart';
import 'package:car_pool_app/Static%20Data/constants.dart';
import 'package:car_pool_app/Services/general_functions.dart';

import 'package:dartz/dartz.dart';
import 'package:firebase_database/firebase_database.dart';


final tripsReference = FirebaseDatabase.instance.ref("trips");

final driverTripsReference = FirebaseDatabase.instance.ref("driver_trips");

final routesReference = FirebaseDatabase.instance.ref("routes");
```

```
final rootReference = FirebaseDatabase.instance.ref();

class Realtime {

  final String uid;

  Realtime({
    required this.uid,
  });

  final usersReference = FirebaseDatabase.instance.ref("users");

  final driversReference = FirebaseDatabase.instance.ref("drivers");

  // Trip Reservation Methods

  Future< Either<ErrorTypes, bool> > reserveTrip({required Trip trip}) async {

    final connection = await LookUp.checkInternetConnection();

    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          final users = await
driverTripsReference.child(trip.driverUid!).child(trip.id).child("users").once().
then((event) => event.snapshot.value);

          final usersList = List<String>.from((users ?? []) as List);

          bool isDuplicate = false;

          for(int i = 0; i < usersList.length; i++) {
            if(uid == usersList[i]) {
              isDuplicate = true;
              break;
            }
          }

          if(isDuplicate) {
            return const Left(
              AlreadyReservedError(),
            );
```

```
          }

          final disableValidation = await
rootReference.child('disable_user_validation').once().then((event) =>
event.snapshot.value as bool);

          if(! disableValidation) {
            // checking the time constraints
            DateTime rawDate = DateTime(trip.tripDate.year, trip.tripDate.month,
trip.tripDate.day);

            rawDate = rawDate.add(timeConstraints[durationToTime(trip.time)]!);

            if(trip.currentDate.isAfter(rawDate)) {
              return Left(
                LateReservationError(
                  errorMessage: "You need to reserve this trip before
${timeConstraintsErrorMessages[trip.time]!}",
                ),
              );
            }
          }

          // reserving successfully
          await tripsReference.child(uid).child(trip.id).set(trip.toJson());

          // adding this user to the users list

          usersList.add(uid);

          // checking the number of seats, to update the tripStatus accordingly

          final numberOfSeats = await
driverTripsReference.child(trip.driverUid!).child(trip.id).child("numberOfSeats")
.once().then((event) => event.snapshot.value) as int;

          // Updating the driver trips reference

          Map<String, dynamic> temp = {
            'numberOfSeats' : ServerValue.increment(-1),
            'users' : usersList,
          };

          if(numberOfSeats == 1) {
            temp['tripStatus'] =
```

```
Trip.tripStatusToJsonString[TripStatus.fullyReserved];
          }

          await
driverTripsReference.child(trip.driverUid!).child(trip.id).update(temp);

          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 101,
            ),
          );
        }
      },
    );
  }

  Future< Either<ErrorTypes, bool> > cancelTrip({required Trip trip}) async {

    final connection = await LookUp.checkInternetConnection();

    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          // canceling successfully
          await tripsReference.child(uid).child(trip.id).update({'status':
'canceled'});

          // reading the users list

          final users = await
driverTripsReference.child(trip.driverUid!).child(trip.id).child("users").once().
then((event) => event.snapshot.value);

          final usersList = List<String>.from((users ?? []) as List);

          usersList.removeWhere((element) => (element == uid));

          // checking the number of seats, to update the tripStatus accordingly
```

```
          final numberOfSeats = await
driverTripsReference.child(trip.driverUid!).child(trip.id).child("numberOfSeats")
.once().then((event) => event.snapshot.value) as int;

          // Updating the drivers reference

          Map<String, dynamic> temp = {
            'numberOfSeats' : ServerValue.increment(1),
            'users' : usersList,
          };

          if(numberOfSeats == 0) {
            temp['tripStatus'] = Trip.tripStatusToJsonString[TripStatus.open];
          }

          await
driverTripsReference.child(trip.driverUid!).child(trip.id).update(temp);

          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 101,
            ),
          );
        }
      },
    );
  }

  Future< Either<ErrorTypes, List<Trip>> > getTrips() async {
    final connection = await LookUp.checkInternetConnection();
    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          final trips = await tripsReference.child(uid).once().then((event) {
            Map jsonMap = (event.snapshot.value ?? {}) as Map;

            final List<Trip> temp = [];
```

```
            jsonMap.forEach((key, value) {
              temp.add(Trip.fullFromJson(value));
            });

            return temp;
          });

          return Right(trips);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 103,
            ),
          );
        }
      },
    );
  }

  Future< Either<ErrorTypes, List<Trip>> > filterTrips({required Trip trip})
async {
    final connection = await LookUp.checkInternetConnection();
    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          final trips = await driverTripsReference.once().then((event) {
            Map jsonMap = (event.snapshot.value ?? {}) as Map;

            final List<Trip> driverTrips = [];

            jsonMap.forEach((key, value) {
              Map tripsMap = (value ?? {}) as Map;

              tripsMap.forEach((tripId, trip) {
                final driverTrip = Trip.fromJson(trip);
                driverTrip.driverUid = key;

                driverTrips.add(driverTrip);
              });
```

```
            });

            driverTrips.removeWhere((element) {
              return element.source != trip.source || element.destination !=
trip.destination || ! element.tripDate.isAtSameMomentAs(trip.tripDate) ||
element.time.compareTo(trip.time) != 0;
            });

            return driverTrips;
          });

          return Right(trips);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 103,
            ),
          );
        }
      },
    );
  }

  // End of Trip Reservation methods

  // UserData Methods

  Future< Either<ErrorTypes, bool> > addUserData(User user) async {
    final connection = await LookUp.checkInternetConnection();
    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          await usersReference.child(uid).set(user.toJson());
          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 103,
```

```
            ),
          );
        }
      },
    );
  }

  Future< Either<ErrorTypes, User> > getUserData() async {
    final connection = await LookUp.checkInternetConnection();
    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          final user = await usersReference.child(uid).once().then((event) =>
User.fromJson(event.snapshot.value as Map));
          return Right(user);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 103,
            ),
          );
        }
      },
    );
  }

  // End of UserData methods
}

// Switch Value Methods

Future< Either<ErrorTypes, bool> > getSwitchValue() async {
  final connection = await LookUp.checkInternetConnection();

  return connection.fold(
    (error) {
      return Left(error);
    },
    (right) async {
      try {
```

```
        final disableValidation = await
rootReference.child('disable_user_validation').once().then((event) =>
event.snapshot.value as bool);

        return Right(disableValidation);
      }
      catch(e) {
        return Left(
          FirebaseError(
            errorMessage: 'Server Error: $e',
            errorId: 101,
          ),
        );
      }
    },
  );
}

Future< Either<ErrorTypes, bool> > setSwitchValue(bool value) async {
    final connection = await LookUp.checkInternetConnection();

    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          await rootReference.child('disable_user_validation').set(value);

          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 101,
            ),
          );
        }
      },
    );
  }

  // CustomRoute Methods
```

```dart
  Future< Either<ErrorTypes, List<CustomRoute>> > getRoutes() async {
    final connection = await LookUp.checkInternetConnection();
    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          final routes = await routesReference.once().then((event) {
            final List jsonList = event.snapshot.value as List;

            return List<CustomRoute>.from(jsonList.map((route) =>
CustomRoute.fromJson(route.cast<String, dynamic>())));
          });

          return Right(routes);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 103,
            ),
          );
        }
      },
    );
  }

Stream<DatabaseEvent> dayStream(String schoolId, String day) {
  return tripsReference.child(schoolId).child(day).onValue;
}
```

Driver Code

```dart
import 'package:driver_car_pool_app/Services/date.dart';
import 'package:driver_car_pool_app/Services/lookup.dart';
import 'package:driver_car_pool_app/Services/errors.dart';
import 'package:driver_car_pool_app/Model%20Classes/custom_route.dart';
import 'package:driver_car_pool_app/Model%20Classes/driver.dart';
import 'package:driver_car_pool_app/Model%20Classes/trip.dart';
import 'package:driver_car_pool_app/Static%20Data/constants.dart';
import 'package:driver_car_pool_app/Services/general_functions.dart';

import 'package:dartz/dartz.dart';
import 'package:firebase_database/firebase_database.dart';


final driverTripsReference = FirebaseDatabase.instance.ref("driver_trips");

final userTripsReference = FirebaseDatabase.instance.ref("trips");

final routesReference = FirebaseDatabase.instance.ref("routes");

final usersReference = FirebaseDatabase.instance.ref("users");

class Realtime {

  final String uid;

  Realtime({
    required this.uid,
  });

  final driversReference = FirebaseDatabase.instance.ref("drivers");

  // Trip Reservation Methods

  Future< Either<ErrorTypes, bool> > createTrip({required Trip trip}) async {

    final check = await checkTrip(trip: trip);

    return check.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
```

```
          final disableValidation = await
FirebaseDatabase.instance.ref().child('disable_driver_validation').once().then((e
vent) => event.snapshot.value as bool);

          if(! disableValidation) {
            // checking the time constraints
            DateTime rawDate = DateTime(trip.tripDate.year, trip.tripDate.month,
trip.tripDate.day);

            rawDate = rawDate.add(timeConstraints[durationToTime(trip.time)]!);

            if(trip.currentDate.isAfter(rawDate)) {
              return Left(
                LateReservationError(
                  errorMessage: "You need to create this trip before
${timeConstraintsErrorMessages[trip.time]!}",
                ),
              );
            }
          }

          // reserving successfully
          await
driverTripsReference.child(uid).child(trip.id).set(trip.toJson());

          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 100,
            ),
          );
        }
      },
    );
  }

  Future< Either<ErrorTypes, bool> > cancelTrip({required String tripId}) async {

    final connection = await LookUp.checkInternetConnection();

    return connection.fold(
      (error) {
```

```
        return Left(error);
      },
      (right) async {
        try {
          // cancelling successfully
          await
driverTripsReference.child(uid).child(tripId).update({'tripStatus': 'canceled'});

          // rejecting the users trips

          final users = await
driverTripsReference.child(uid).child(tripId).child("users").once().then((event)
=> event.snapshot.value);

          final usersList = List<String>.from((users ?? []) as List);

          for(int i = 0; i < usersList.length; i++) {
            await
userTripsReference.child(usersList[i]).child(tripId).update({"status":
"rejected"});
          }

          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 101,
            ),
          );
        }
      },
    );
  }

  Future< Either<ErrorTypes, bool> > approveTrip({required Trip trip}) async {

    final connection = await LookUp.checkInternetConnection();

    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
```

```
        try {
          final disableApproval = await
FirebaseDatabase.instance.ref().child('disable_approval_driver').once().then((eve
nt) => event.snapshot.value as bool);

          if(! disableApproval) {
            // checking the time constraints
            DateTime rawDate = DateTime(trip.tripDate.year, trip.tripDate.month,
trip.tripDate.day);

            rawDate =
rawDate.add(approvalTimeConstraints[durationToTime(trip.time)]!);

            final currentDateFuture = await Date.fetchDate();

            return currentDateFuture.fold(
              (error) {
                return Left(error);
              },
              (currentDate) async {
                if(currentDate.isAfter(rawDate)) {
                  await cancelTrip(
                    tripId: trip.id,
                  );

                  return Left(
                    LateApprovalError(
                      errorMessage: "You need to approve this trip before
${approvalErrorMessages[trip.time]!}",
                    ),
                  );
                }

                // approving successfully
                await
driverTripsReference.child(uid).child(trip.id).update({'tripStatus':
'approved'});

                // approving the users trips

                final users = await
driverTripsReference.child(uid).child(trip.id).child("users").once().then((event)
=> event.snapshot.value);

                final usersList = List<String>.from((users ?? []) as List);
```

```
                for(int i = 0; i < usersList.length; i++) {
                  await
userTripsReference.child(usersList[i]).child(trip.id).update({"status":
"approved"});
                }

                return const Right(true);
              },
            );
          }

          // approving successfully
          await
driverTripsReference.child(uid).child(trip.id).update({'tripStatus':
'approved'});

          // approving the users trips

          final users = await
driverTripsReference.child(uid).child(trip.id).child("users").once().then((event)
=> event.snapshot.value);

          final usersList = List<String>.from((users ?? []) as List);

          for(int i = 0; i < usersList.length; i++) {
            await
userTripsReference.child(usersList[i]).child(trip.id).update({"status":
"approved"});
          }

          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 102,
            ),
          );
        }
      },
    );
  }
```

```dart
  Future< Either<ErrorTypes, bool> > completeTrip({required String tripId}) async
{

    Map<String, dynamic> temp = {
      'tripsCount' : ServerValue.increment(1),
      'points' : ServerValue.increment(5),
    };

    final connection = await LookUp.checkInternetConnection();

    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          // completing successfully
          await
driverTripsReference.child(uid).child(tripId).update({'tripStatus':
'completed'});

          // giving bonus to the drivers
          await driversReference.child(uid).update(temp);


          final users = await
driverTripsReference.child(uid).child(tripId).child("users").once().then((event)
=> event.snapshot.value);

          final usersList = List<String>.from((users ?? []) as List);

          for(int i = 0; i < usersList.length; i++) {
            // completing the users trips
            await
userTripsReference.child(usersList[i]).child(tripId).update({"status":
"completed"});

            // giving bonus to the users
            await usersReference.child(usersList[i]).update(temp);
          }

          return const Right(true);
        }
        catch(e) {
          return Left(
```

```
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 103,
            ),
          );
        }
      },
    );
  }

  Future< Either<ErrorTypes, List<Trip>> > getTrips() async {
    final connection = await LookUp.checkInternetConnection();
    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          final trips = await driverTripsReference.child(uid).once().then((event)
{
            Map jsonMap = (event.snapshot.value ?? {}) as Map;

            final List<Trip> temp = [];

            jsonMap.forEach((key, value) {
              temp.add(Trip.fromJson(value));
            });

            return temp;
          });

          return Right(trips);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 104,
            ),
          );
        }
      },
    );
  }
```

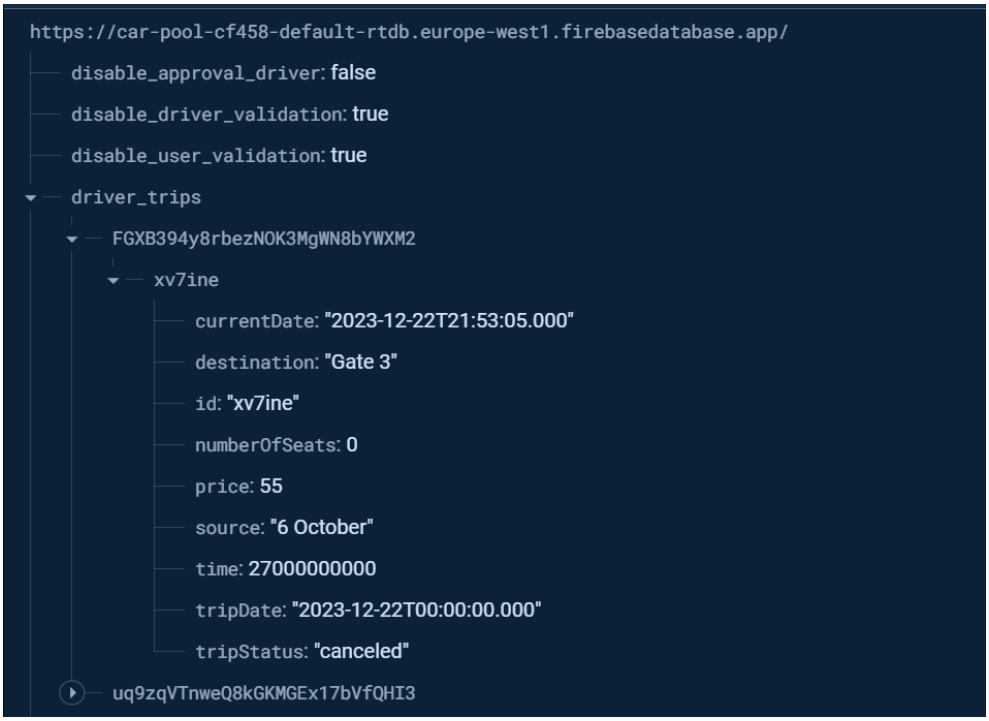
```
  Future< Either<ErrorTypes, bool> > checkTrip({required Trip trip}) async {

    final tripsFuture = await getTrips();

    return tripsFuture.fold(
      (error) {
        return Left(error);
      },
      (trips) {
        for(int i = 0; i < trips.length; i++) {
          if(trips[i].source == trip.source && trips[i].destination ==
trip.destination && trips[i].tripDate.isAtSameMomentAs(trip.tripDate) &&
trips[i].time.compareTo(trip.time) == 0) {
            return const Left(
              AlreadyReservedError()
            );
          }
        }

        return const Right(true);
      },
    );
  }

  // End of Trip Reservation methods

  // UserData Methods

  Future< Either<ErrorTypes, bool> > addDriverData(Driver driver) async {
    final connection = await LookUp.checkInternetConnection();
    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          await driversReference.child(uid).set(driver.toJson());
          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 105,
            ),
```

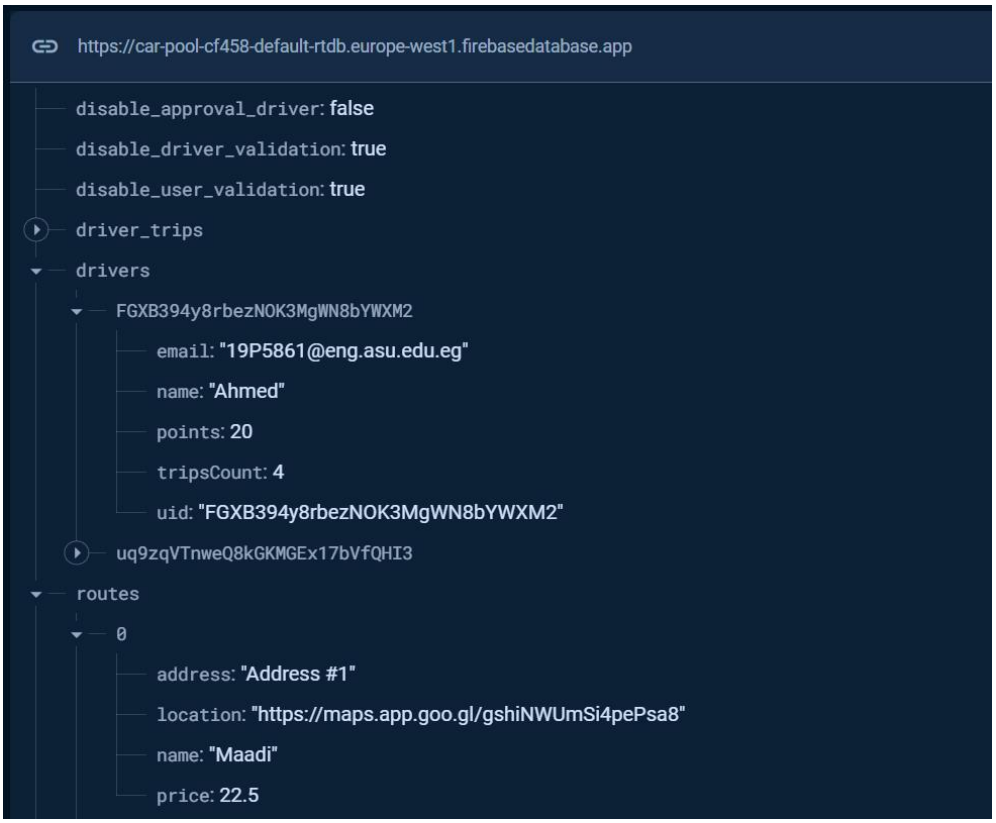
```
          );
        }
      },
    );
  }

  Future< Either<ErrorTypes, Driver> > getDriverData() async {
    final connection = await LookUp.checkInternetConnection();
    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          final user = await driversReference.child(uid).once().then((event) =>
Driver.fromJson(event.snapshot.value as Map));
          return Right(user);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 106,
            ),
          );
        }
      },
    );
  }

  // End of UserData methods
}

// Switch Value Methods

Future< Either<ErrorTypes, bool> > getValidationSwitchValue() async {
  final connection = await LookUp.checkInternetConnection();

  return connection.fold(
    (error) {
      return Left(error);
    },
    (right) async {
      try {
        final disableValidation = await
```

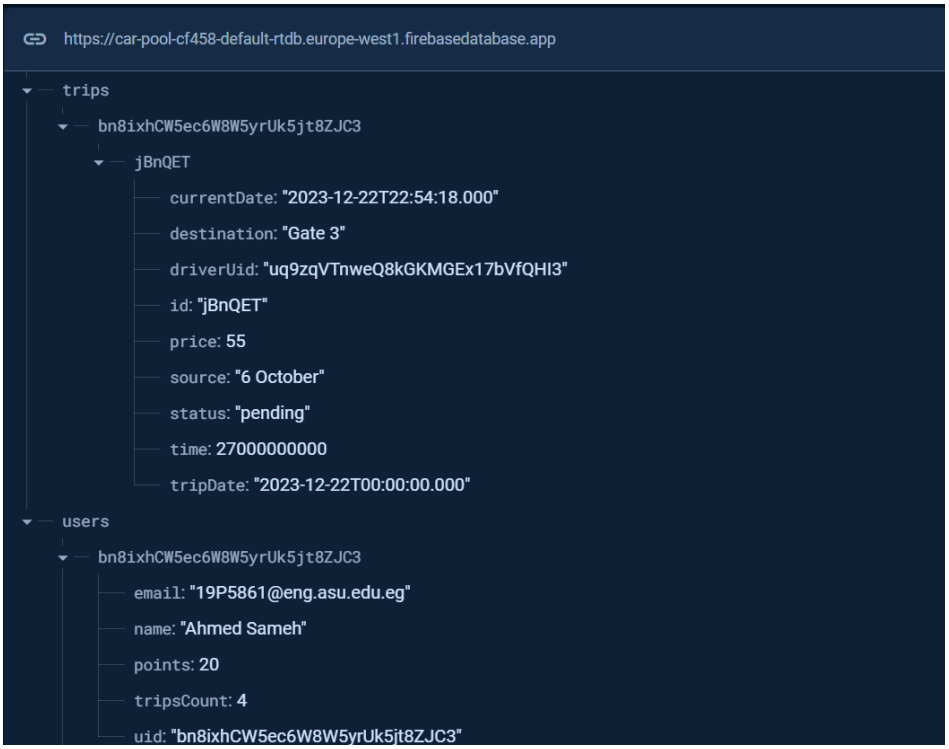
```
FirebaseDatabase.instance.ref().child('disable_driver_validation').once().then((e
vent) => event.snapshot.value as bool);

        return Right(disableValidation);
      }
      catch(e) {
        return Left(
          FirebaseError(
            errorMessage: 'Server Error: $e',
            errorId: 107,
          ),
        );
      }
    },
  );
}

Future< Either<ErrorTypes, bool> > setValidationSwitchValue(bool value) async {
    final connection = await LookUp.checkInternetConnection();

    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          await
FirebaseDatabase.instance.ref().child('disable_driver_validation').set(value);

          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 108,
            ),
          );
        }
      },
    );
  }

  Future< Either<ErrorTypes, bool> > getApprovalSwitchValue() async {
  final connection = await LookUp.checkInternetConnection();
```

```
  return connection.fold(
    (error) {
      return Left(error);
    },
    (right) async {
      try {
        final disableValidation = await
FirebaseDatabase.instance.ref().child('disable_approval_driver').once().then((eve
nt) => event.snapshot.value as bool);

        return Right(disableValidation);
      }
      catch(e) {
        return Left(
          FirebaseError(
            errorMessage: 'Server Error: $e',
            errorId: 117,
          ),
        );
      }
    },
  );
}

Future< Either<ErrorTypes, bool> > setApprovalSwitchValue(bool value) async {
    final connection = await LookUp.checkInternetConnection();

    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          await
FirebaseDatabase.instance.ref().child('disable_approval_driver').set(value);

          return const Right(true);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 118,
            ),
```

```
          );
        }
      },
    );
  }

  // CustomRoute Methods

  Future< Either<ErrorTypes, List<CustomRoute>> > getRoutes() async {
    final connection = await LookUp.checkInternetConnection();
    return connection.fold(
      (error) {
        return Left(error);
      },
      (right) async {
        try {
          final routes = await routesReference.once().then((event) {
            final List jsonList = event.snapshot.value as List;

            return List<CustomRoute>.from(jsonList.map((route) =>
CustomRoute.fromJson(route.cast<String, dynamic>())));
          });

          return Right(routes);
        }
        catch(e) {
          return Left(
            FirebaseError(
              errorMessage: 'Server Error: $e',
              errorId: 109,
            ),
          );
        }
      },
    );
  }

Stream<DatabaseEvent> dayStream(String schoolId, String day) {
  return driverTripsReference.child(schoolId).child(day).onValue;
}
```


https://car-pool-cf458-default-rtdb.europe-west1.firebasedatabase.app/
disable_approval_driver: false
disable_driver_validation: true
disable_user_validation: true
driver_trips
FGXB394y8rbezNOK3MgWN8bYWXM2
xv7ine
currentDate: "2023-12-22T21:53:05.000"
destination: "Gate 3"
id: "xv7ine"
numberOfSeats: 0
price: 55
source: "6 October"
time: 27000000000
tripDate: "2023-12-22T00:00:00.000"
tripStatus: "canceled"
uq9zqVTnweQ8kGKMGEx17bVfQHI3


https://car-pool-cf458-default-rtdb.europe-west1.firebasedatabase.app
disable_approval_driver: false
disable_driver_validation: true
disable_user_validation: true
driver_trips
drivers
FGXB394y8rbezNOK3MgWN8bYWXM2
email: "19P5861@eng.asu.edu.eg"
name: "Ahmed"
points: 20
tripsCount: 4
uid: "FGXB394y8rbezNOK3MgWN8bYWXM2"
uq9zqVTnweQ8kGKMGEx17bVfQHI3
routes
0
address: "Address #1"
location: "https://maps.app.goo.gl/gshiNWUmSi4pePsa8"
name: "Maadi"
price: 22.5


https://car-pool-cf458-default-rtdb.europe-west1.firebasedatabase.app
trips
bn8ixhCW5ec6W8W5yrUk5jt8ZJC3
jBnQET
currentDate: "2023-12-22T22:54:18.000"
destination: "Gate 3"
driverUid: "uq9zqVTnweQ8kGKMGEx17bVfQHI3"
id: "jBnQET"
price: 55
source: "6 October"
status: "pending"
time: 27000000000
tripDate: "2023-12-22T00:00:00.000"
users
bn8ixhCW5ec6W8W5yrUk5jt8ZJC3
email: "19P5861@eng.asu.edu.eg"
name: "Ahmed Sameh"
points: 20
tripsCount: 4
uid: "bn8ixhCW5ec6W8W5yrUk5jt8ZJC3"