



# Ain Shams Carpool Project

Name : Ziad Ashraf Ahmed Ahmed

ID:19p7095

Group 1 Section 1

Video Link:

<https://drive.google.com/file/d/1Qa9fiA2buVEiXOmwkUKgDITGi76xyibK/view?usp=sharing>

GitHub Link: <https://github.com/zakoshgo/MobileProgramming-CarPool.git>

|  |    |
|--|----|
| Introduction: .....                                | 3  |
| Specs(Features) .....                              | 4  |
| For Passengers App .....                           | 4  |
| For Driver App .....                               | 6  |
| Screens Layouts:.....                              | 8  |
| Driver.....  | 8  |
| Login Screen & Signup Screen.....                  | 8  |
| HomeScreen .....                                   | 9  |
| Add Ride Screen .....                              | 10 |
| Profile Page .....                                 | 12 |
| Review Your ride .....                             | 13 |
| Passenger .....                                    | 14 |
| Login and Signup screens .....                     | 14 |
| Profile & order_Histore .....                      | 15 |
| Homepage .....                                     | 16 |
| Order_Tracking_page & Cart page .....              | 17 |
| For Testing Layouts.....                           | 18 |
| Passenger Homepage & driver homepage.....          | 18 |
| Database Structure .....                           | 19 |
| Local Database Structure .....                     | 19 |
| For Driver .....                                   | 19 |
| For passengers .....                               | 22 |
| Remote Database Structure and code .....           | 25 |
| Example at some instance.....                      | 25 |
| Code Authentication class For Passenger .....      | 26 |
| Code Authentication class For Driver.....          | 28 |
| Test Case scenarios .....                          | 31 |
| 1)Group of test scenarios for sign-Up page .....   | 31 |
| 2) Test case Login .....                           | 33 |
| 2)Test case :Add ride For Driver .....             | 34 |
| 3) Test case :Driver Navigate to profile Page..... | 34 |
| 4) Test Case :Edit Profile Page .....              | 35 |
| 5) Test case: Driver Time Constrains .....         | 35 |

|   |    |
|---|----|
| 6) Test Case : Logout for both drivers/passengers .....         | 36 |
| 7) Test case Request ride in passenger App .....                | 36 |
| 8)Test case Accept/reject ride in Driver App.....               | 37 |
| 9)Test case Passenger check the assigned ride .....             | 38 |
| 10)Test case : Passenger Time Constrains on reserve a ride..... | 38 |

## Introduction:

Carpool 2.0 represents a groundbreaking approach to ridesharing within the academic community, focusing on the transportation needs to and from Abdu-Basha to anywhere. As a testament to our commitment to safety and community, users are required to sign in with their active @eng.asu.edu.eg accounts, fostering a trusted closed community environment.

Operated by students for students, Carpool 2.0 introduces a revolutionary strategy in recruiting drivers and managing the service. This pilot project will streamline rides to two designated destination points – Gate 3 and 4 – with fixed departure and return times. Departure is scheduled for 7:30 am from various locations to one to the 2 gates, while the return ride is set at 5:30 pm from the Faculty of Engineering campus.

To ensure a seamless experience, customers are required to reserve their seats in advance. Those in need of a ride at 7:30 am must secure their seat before 10:00 pm the previous day, and for the 5:30 pm return ride, reservations must be made before 1:00 pm on the same day.

In addition, mobile App will be developed for drivers to confirm orders and update status data. Orders must be confirmed before 11:30 pm for the morning ride and before 4:30 pm for the afternoon ride, ensuring timely coordination between drivers and passengers.

# Specs(Features)

## For Passengers App

### 1. Authentication:

- **Login Page:**
  - Implement Firebase Authentication for secure user login.
  - Include a "Sign Up" option for new users.
- **Testing Credentials:**
  - Provide a test account with login information for testing purposes.

### 2. Home Page:

- **Route Information:**
  - Display a list of available routes to and from Ainsams Campus.
  - Utilize a RecyclerView for an organized and user-friendly display.
  - Include the status of each route.
- **Reservation:**
  - Allow users to select a route to reserve.

### 3. Cart Page:

- **Order Review:**
  - Provide a cart page for users to review their selected route.
  - Include options for making payments.
- **Confirmation:**
  - Implement a confirmation button for finalizing reservations.

### 4. Order History:

- **Tracking/Status Page:**
  - Enable users to view a history of their requested rides.
  - Include a tracking/status page for each ride.

### 5. Database Integration:

- **Firebase Real-time Database:**
  - Utilize Firebase Real-time Database for storing route information and order status.
- **Local Database (SQLite):**
  - Store a copy of profile data for passengers and drivers locally using SQLite.

- Ensure synchronization with Firebase for consistency.

#### **6. Order Tracking Page:**

- **Status Updates:**
  - Implement a page for users to track the status of their reservations.
  - Display detailed ride information.

#### **7. Profile Page:**

- **User Profile:**
  - Enable users to edit their profile information.
  - Ensure updates reflect in both the local database and Firebase.

## For Driver App

### 1. Authentication:

- **Login Page:**
  - Implement Firebase Authentication for secure user login.
  - Include a "Sign Up" option for new users.
- **Testing Credentials:**
  - Provide a test account with login information for testing purposes.

### 2. Home Page:

- **Route Information:**
  - Display a list of routes belongs to the current Driver to and from Ainsams Campus.
  - Utilize a Recycler View for an organized and user-friendly display.
  - Include the status of each route.

### 3. Add Route Page:

- **Order Review:**
  - Provide a Add route page for Drivers to Add route details needed as Date, Time, Source, Destination, and Price.
- **Confirmation:**
  - Implement a confirmation button for finalizing Adding route.

### 5. Database Integration:

- **Firebase Real-time Database:**
  - Utilize Firebase Real-time Database for storing route information and order status.
- **Local Database (SQLite):**
  - Store a copy of profile data for drivers locally using SQLite.
  - Ensure synchronization with Firebase for consistency.

### 6. Ride Tracking Page:

- **Status Updates:**
  - Implement a page for drivers to track the status of passengers reservations.
  - Accept or reject passengers
  - Change the State of Trip
  - Display detailed ride information.

## **7. Profile Page:**

- **User Profile:**
  - Enable drivers to edit their profile information.
  - Ensure updates reflect in both the local database and Firebase.




## Screens Layouts:

### Driver

#### Login Screen & Signup Screen

17:17 | 4.2KB/s



**Welcome to Ainshams CarPool**  
**Driver: Login**

User Email

---


User Password

---

[Login](#)

[Don't have an account? Signup Here](#)

17:14 | 6.4KB/s



**Welcome to Ainshams CarPool**  
**Driver Signup**

User Name

---

User Phone number

---

User Email

---

User Password

---

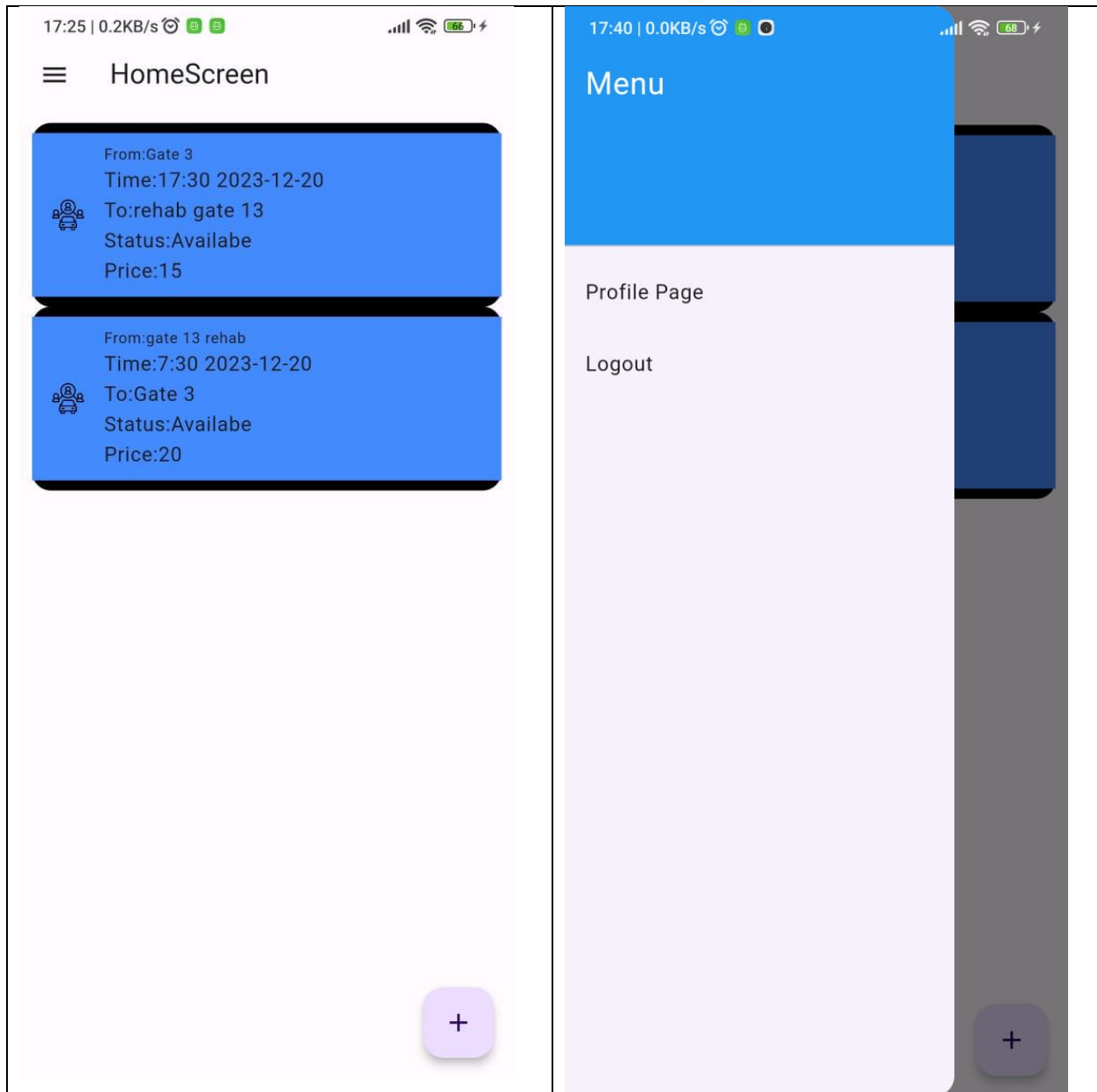
Confirm Password

---

[Sign Up](#)

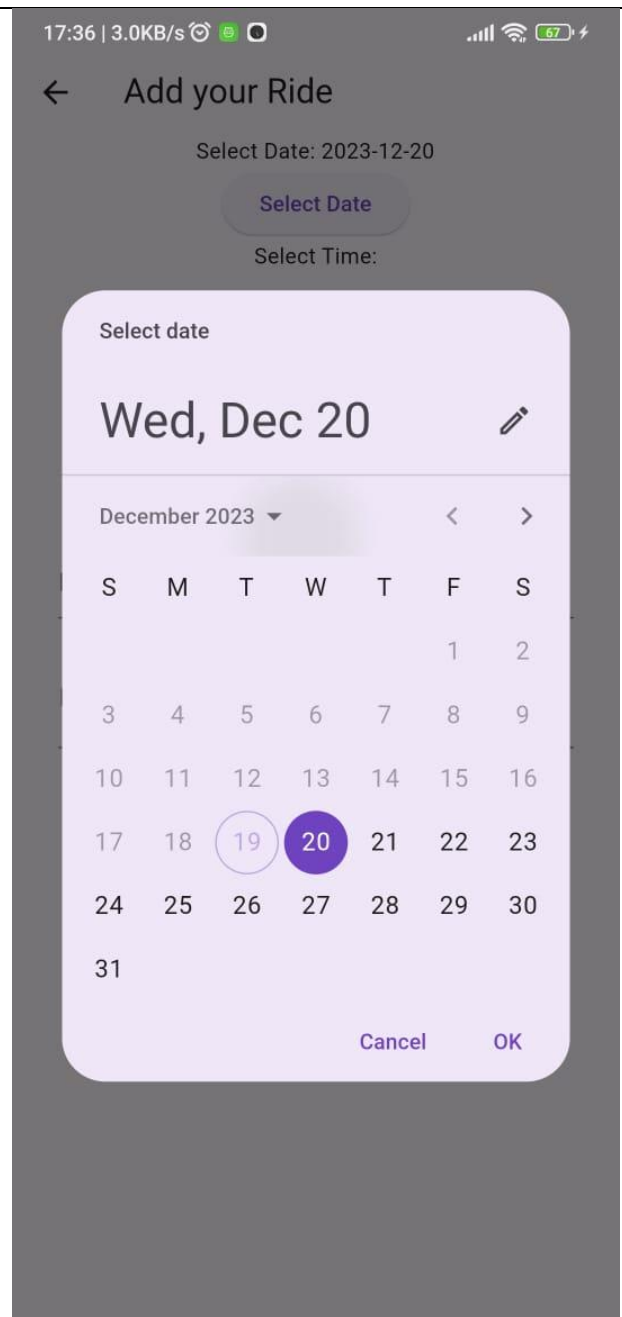
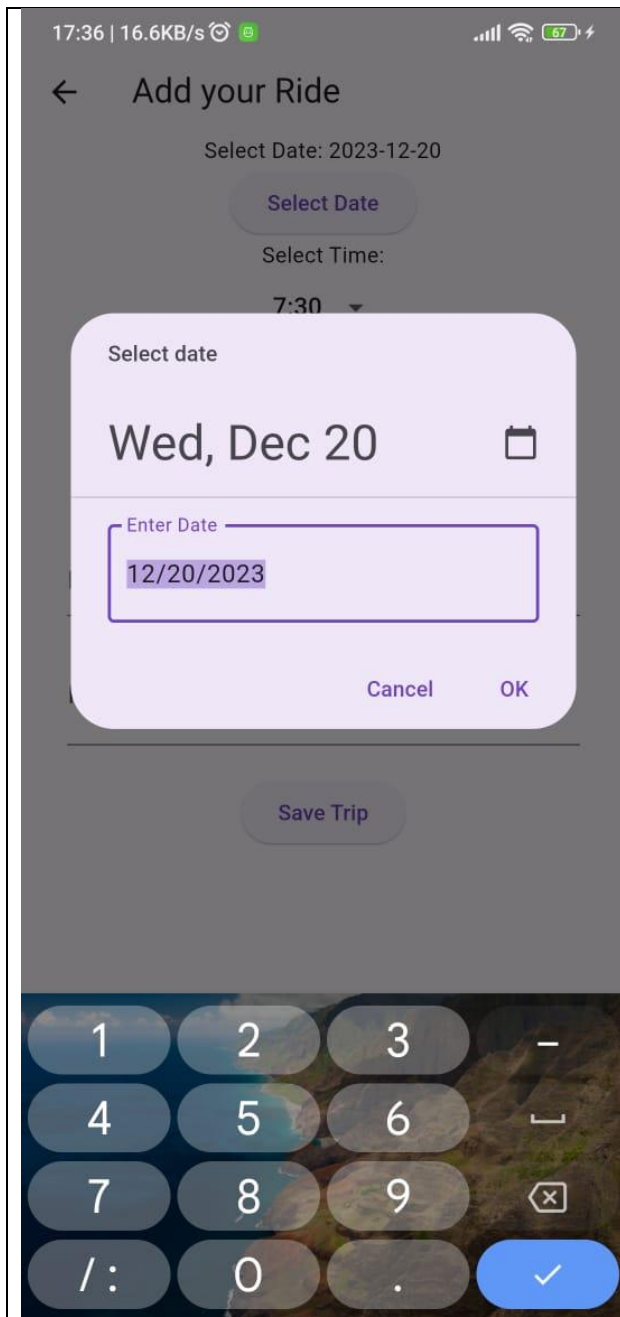
[Already have an account? Login Here](#)

## HomeScreen

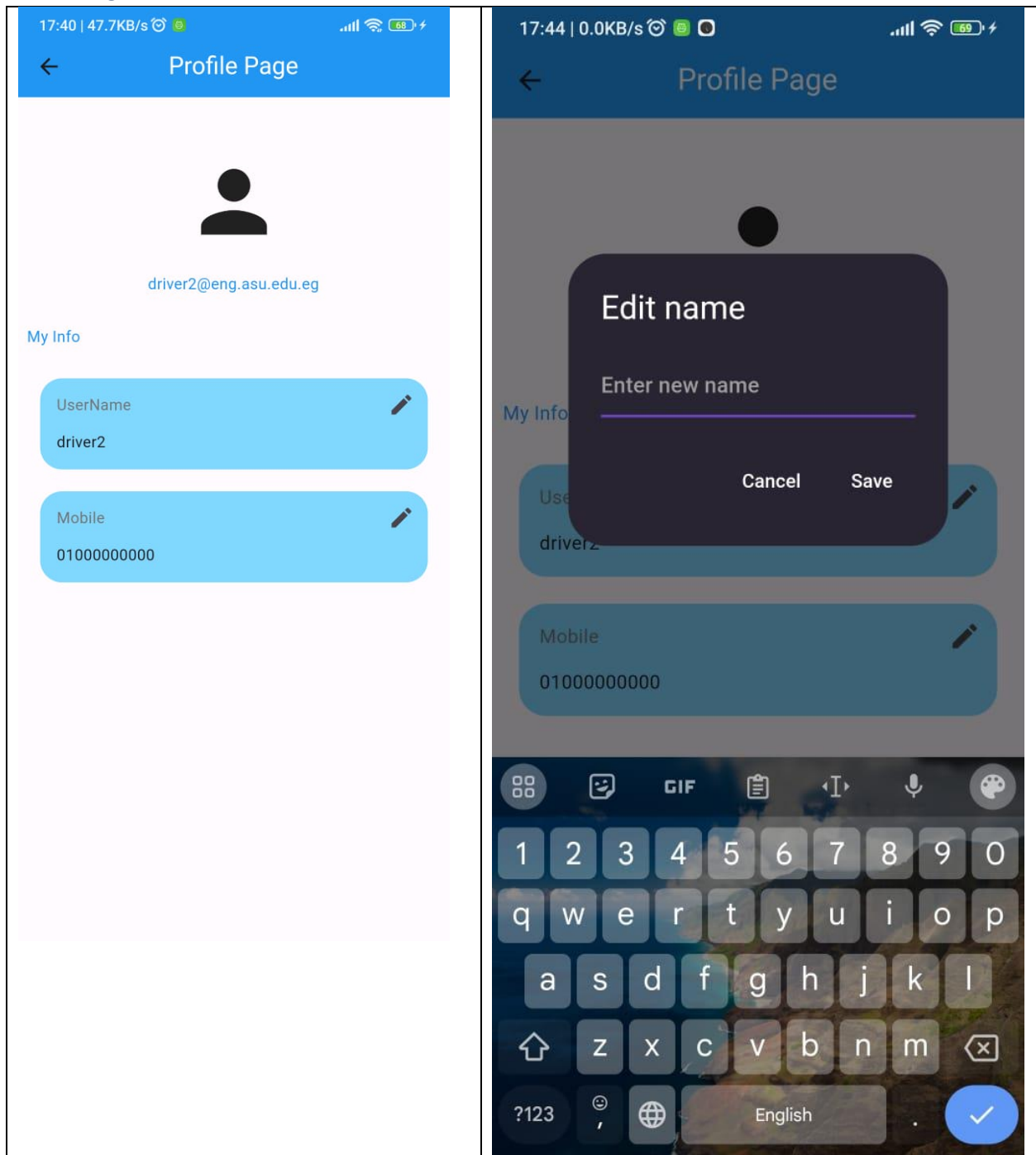


## Add Ride Screen

| 17:36   17.3KB/s 📶 🔋 67%   | 17:36   17.3KB/s 📶 🔋 67%   |
|--|--|
| <div>← Add your Ride</div> <div>Select Date: 2023-12-20</div> <div>Select Date</div> <div>Select Time:</div> <div>17:30 ▼</div> <div>Enter Destination:</div> <div>Enter destination</div> <div>Select Pickup Point:</div> <div>Gate 3 ▼</div> <div>Price</div> <div>Save Trip</div> | <div>← Add your Ride</div> <div>Select Date: 2023-12-20</div> <div>Select Date</div> <div>Select Time:</div> <div>7:30 ▼</div> <div>Select Destination:</div> <div>Gate 3 ▼</div> <div>Enter Pickup Point:</div> <div>Enter pickup point</div> <div>Price</div> <div>Save Trip</div> |



## Profile Page





## Review Your ride

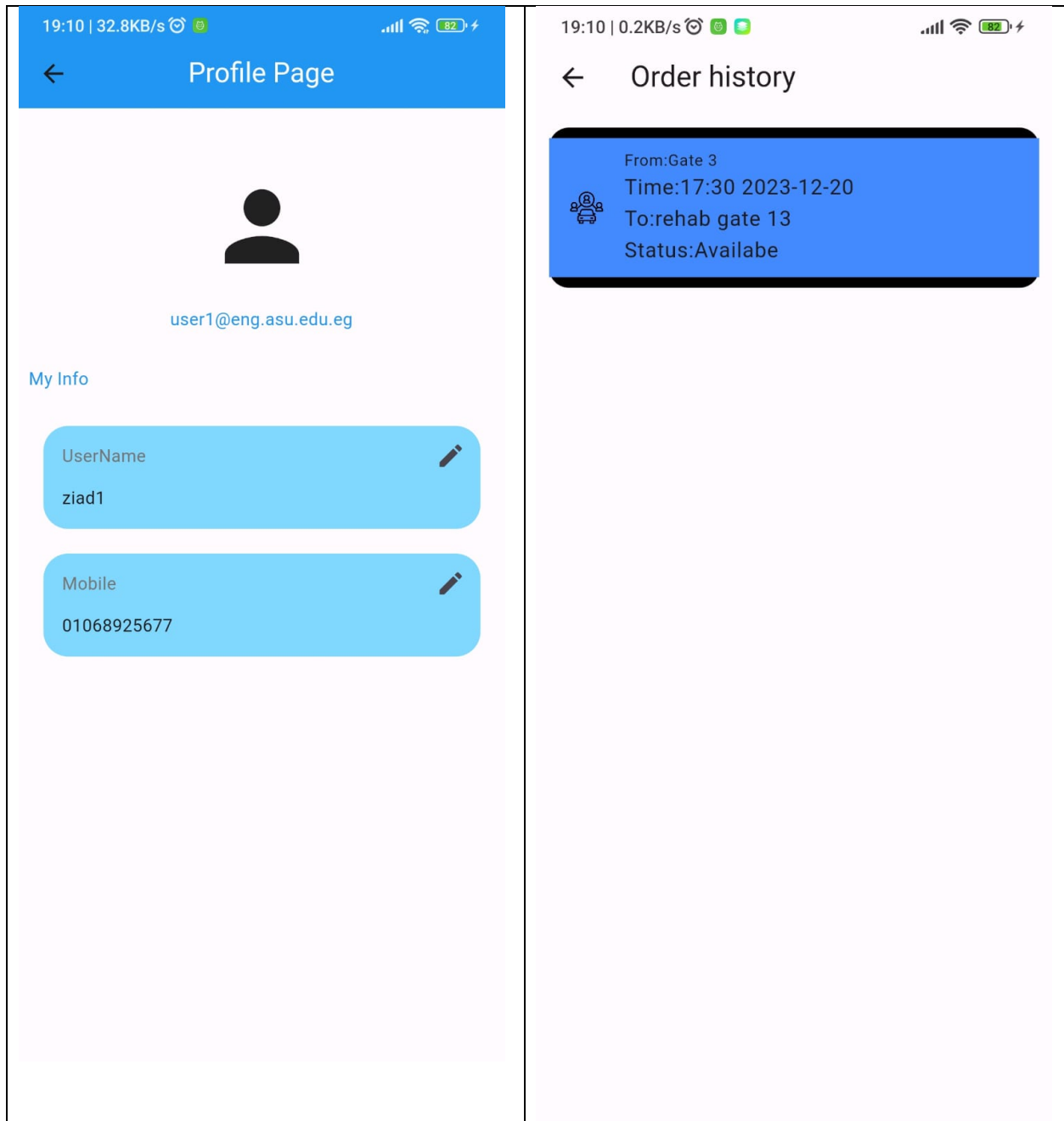
| 17:57   20.8KB/s   | 17:58   11.3KB/s  |
|--|---|
| <p>← Review your Ride</p> <p>Rider name:driver2</p> <p>PickUp point:Gate 3</p> <p>Destination point:rehab gate 13</p> <p>Time:17:30</p> <p>Price::15</p> <p>Status: Availabe</p> <p>Start Ride End Ride Cancel Ride</p> <p>NO accepted passengers yet</p> <div><div>Accept</div><div>ziad<br/>01068925677</div><div>Reject</div></div> | <p>← Review your Ride</p> <p>Rider name:driver2</p> <p>PickUp point:Gate 3</p> <p>Destination point:rehab gate 13</p> <p>Time:17:30</p> <p>Price::15</p> <p>Status: Availabe</p> <p>Start Ride End Ride Cancel Ride</p> <div><div>Name ziad<br/>Mobile 01068925677<br/>Status: Accepted</div></div> <p>NO passengers request the ride yet</p> |

## Passenger

### Login and Signup screens

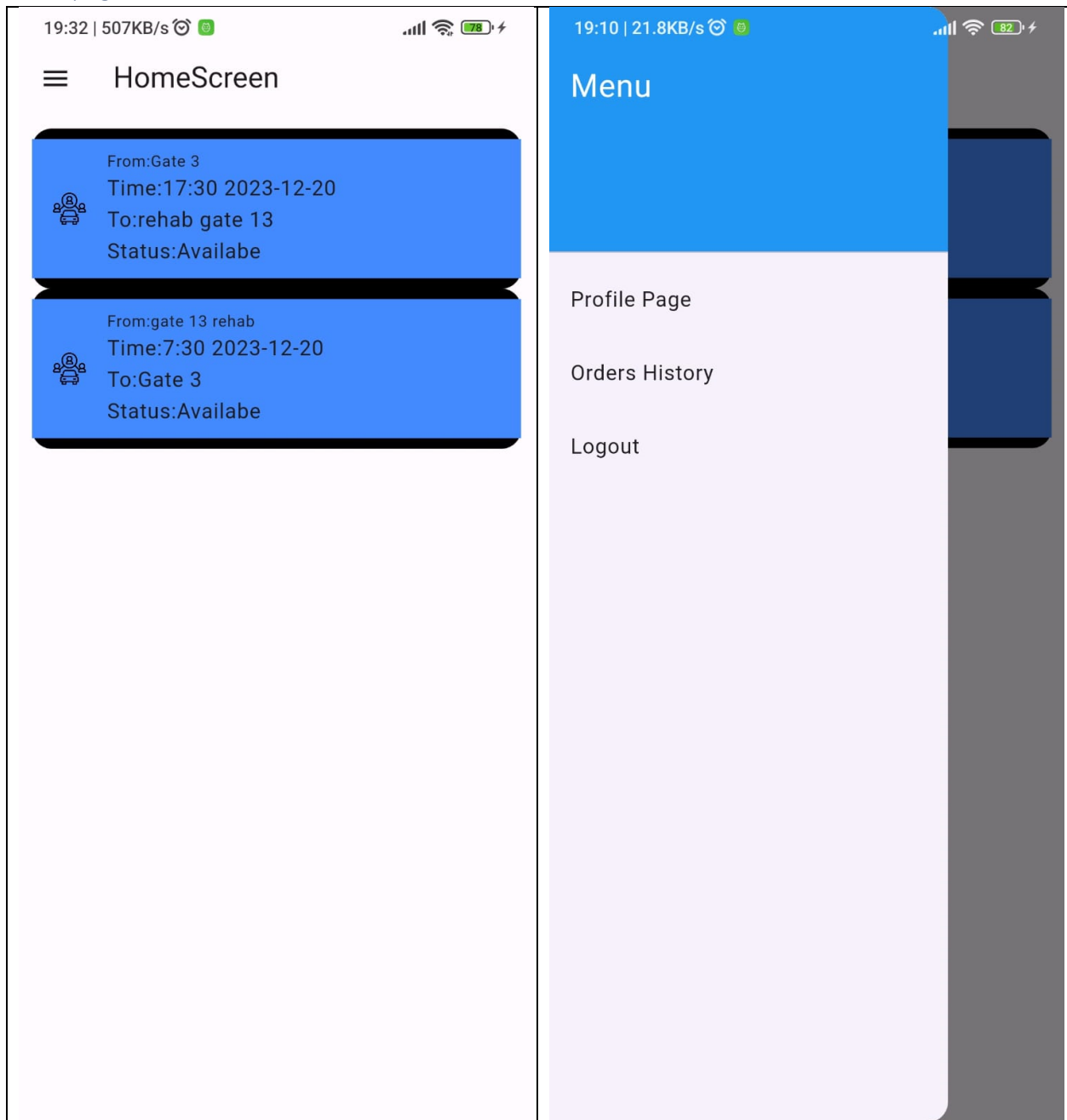
| 18:20   0.3KB/s 📶 🔋 73%   | 18:20   0.3KB/s 📶 🔋 73%  |
|---|--|
| <div></div> <div><b>Welcome to Ainshams CarPool<br/>Passenger Login</b></div> <div><div>User Email</div><div></div></div> <div><div>User Password</div><div>👁️</div></div> <div><div>Login</div></div> <div><a href="#">Don't have an account? Signup Here</a></div> | <div></div> <div><b>Welcome to Ainshams CarPool<br/>Passenger Signup</b></div> <div><div>User Name</div><div></div></div> <div><div>User Phone number</div><div></div></div> <div><div>User Email</div><div></div></div> <div><div>User Password</div><div></div></div> <div><div>Confirm Password</div><div></div></div> <div><div>Sign Up</div></div> <div><a href="#">Already have an account? Login Here</a></div> |

## Profile & order\_Histore



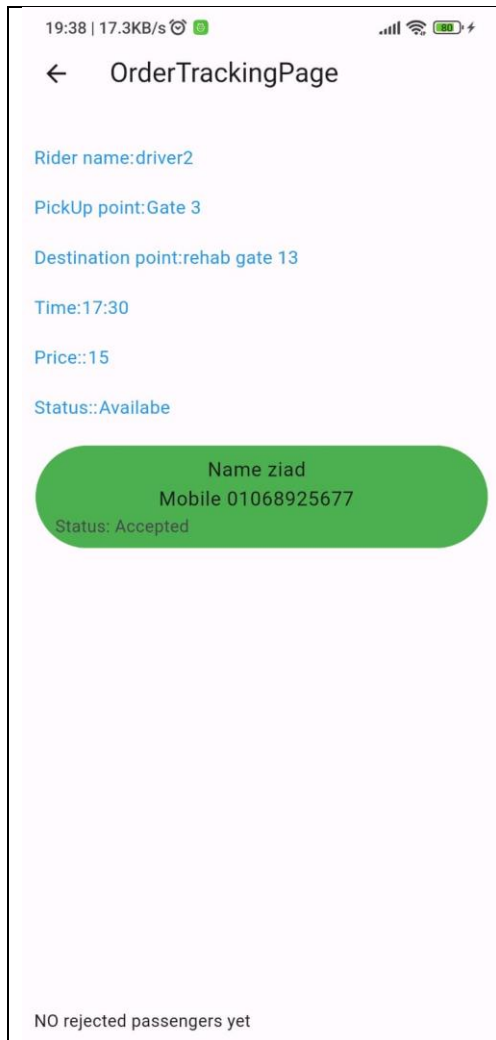


## Homepage

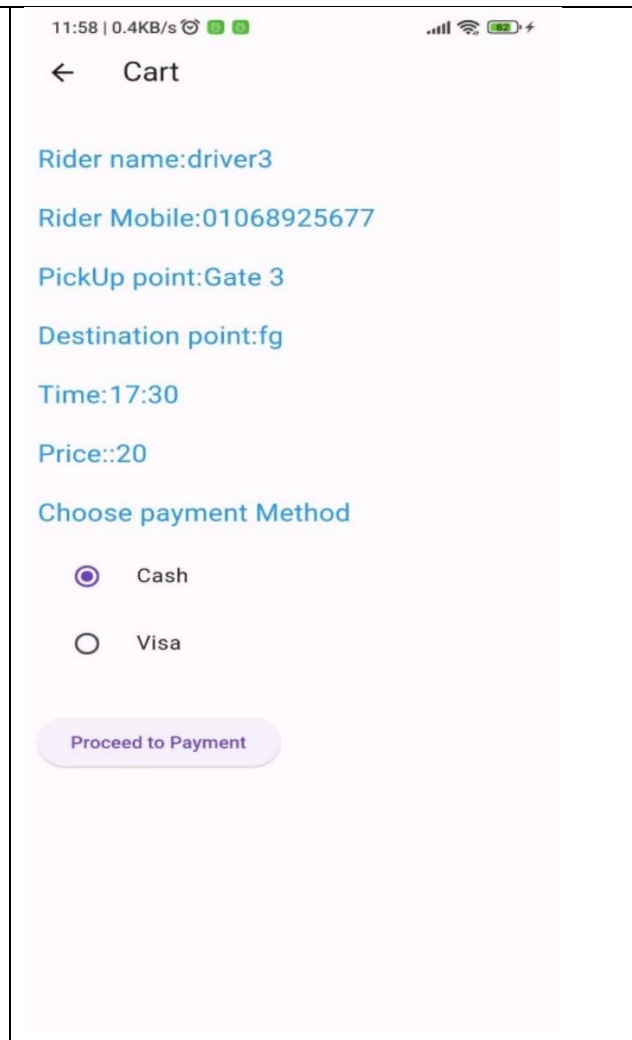


## Order\_Tracking\_page & Cart page

### Order Tracking page

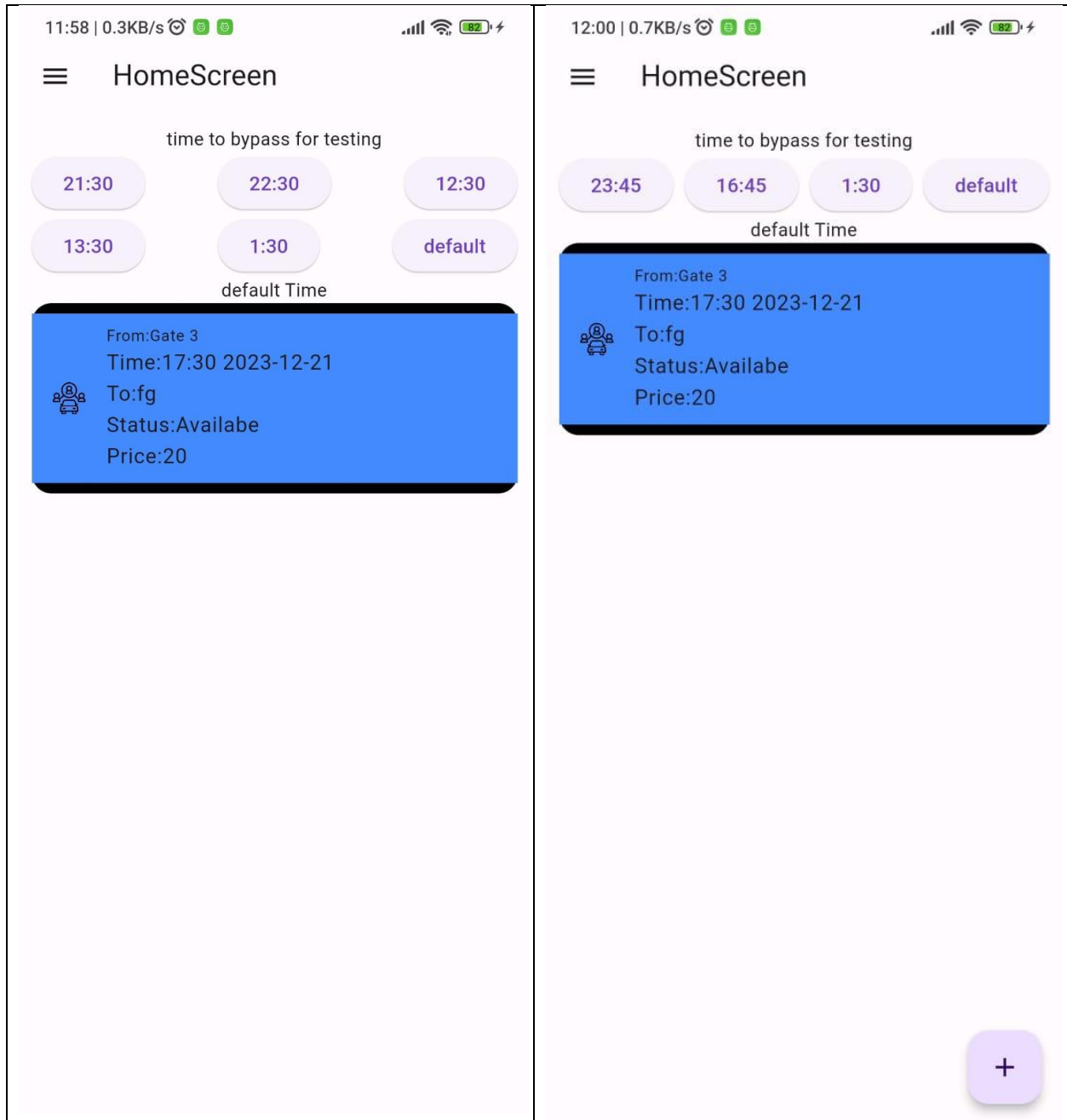


### Cart\_page



## For Testing Layouts

### Passenger Homepage & driver homepage



## Database Structure

### Local Database Structure

For Driver

Table:drivers

| ID(primary key) | name | email | phone |
|-----------------|------|-------|-------|
|-----------------|------|-------|-------|

Table Name: drivers

Columns:

- id (TEXT, PRIMARY KEY): This column uniquely identifies each driver. It is mandatory and cannot be null.
- name (TEXT): This column stores the driver's name. It is also mandatory and cannot be null.
- email (TEXT): This column stores the driver's email address.
- phone (TEXT): This column stores the driver's phone number.

Additional Notes:

- The database file is named "DRIVERPROJECTDB" and is stored in the application's data directory.
- The database version is 1.
- The code includes functions to create the table, get all users, get a specific user by ID, and insert or update a user.

```

import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
import '../Test_file/GlobalVariableForTesting.dart';

class MyDatabaseClass {
  Database? mydb;

  Future<Database?> mydbcheck() async {
    if (mydb == null) {
      mydb = await initiatingDatabase();
      return mydb;
    } else
      return mydb;
  }

  int Version = 1;
  Future<Database?> initiatingDatabase() async {
    String destinationPath = await getDatabasesPath();
    String Path = join(destinationPath, "DRIVERPROJECTDB");
    Database mydatabase = await openDatabase(
      Path,
      version: Version,
      onCreate: (db, version) {
        db.execute('''
          CREATE TABLE IF NOT EXISTS drivers (
            id TEXT PRIMARY KEY,
            name TEXT,
            email TEXT,
            phone TEXT
          )
        ''');
      },
    );
    return mydatabase;
  }

  Future<List<Map<String, dynamic>>> getUsers() async {
    Database? temp = await mydbcheck();
    var response = await temp!.rawQuery('SELECT * FROM drivers');
    return response;
  }

  Future<void> printTableContents() async {
    // Print the contents of the drivers table
    List<Map<String, dynamic>> drivers = await getUsers();

    drivers.forEach((user) {
      print('from local Database User ID: ${user['id']}, Name:
${user['name']}, Email: ${user['email']}, Phone: ${user['phone']}');
    });
  }

  Future<List<Map<String, dynamic>>> getSpecificUser(String uid) async {
    Database? temp = await mydbcheck();
    var response = await temp!.rawQuery(''
    SELECT * FROM drivers WHERE id = ?

```

```

        '', [uid]);
        return response;
    }

    Future<void> InsertOrUpdateUser(var uid,String username,String mobile)
    async {
        Database? temp = await mydbcheck();
        if(TESTMODE==0){
            print("not storing tester values in local database");
            var response = await temp!.rawInsert(''
            INSERT OR REPLACE INTO drivers (id, name, phone)
            VALUES (?, ?, ?)
            '', [uid, username, mobile]);
            // Print table contents after updating
            await printTableContents();
        }
        else{
            print("storing tester values in local database");
            var response = await temp!.rawInsert(''
            INSERT OR REPLACE INTO drivers (id, name, phone)
            VALUES (?, ?, ?)
            '', ["TEST", username, mobile]);
            // Print table contents after updating
            await printTableContents();
        }
    }
}

```

For passengers

Table:users

| ID(primary key) | name | email | phone |
|-----------------|------|-------|-------|
|-----------------|------|-------|-------|

Table Name: users

Columns:

- id (TEXT, PRIMARY KEY): This column uniquely identifies each user. It is mandatory and cannot be null.
- name (TEXT): This column stores the user's name.
- email (TEXT): This column stores the user's email address.
- phone (TEXT): This column stores the user's phone number.

Additional Notes:

- The database file is named "USERPROJECTDB" and is stored in the application's data directory.
- The database version is 1.
- The code includes functions to create the table, get all users, get a specific user by ID, and insert or update a user.

Differences from previous snippet:

- The table name changed from "drivers" to "users".
- The functions are named slightly differently (e.g., "getUsers" instead of "getUser").
- The functions seem to be used in the context of user profiles.

```

import 'package:path/path.dart';
import 'package:sqflite/sqflite.dart';
import '../Test_file/GlobalVariableForTesting.dart';

class MyDatabaseClass {
  Database? mydb;
  //used in profile
  Future<Database?> mydbcheck() async {
    if (mydb == null) {
      mydb = await initiatingDatabase();
      return mydb;
    } else
      return mydb;
  }

  int Version = 1;
  Future<Database?> initiatingDatabase() async {
    String destinationPath = await getDatabasesPath();
    String Path = join(destinationPath, "USERPROJECTDB");
    Database mydatabase = await openDatabase(
      Path,
      version: Version,
      onCreate: (db, version) {
        db.execute('''
          CREATE TABLE IF NOT EXISTS users (
            id TEXT PRIMARY KEY,
            name TEXT,
            email TEXT,
            phone TEXT
          )
        ''');
      },
    );
    return mydatabase;
  }

  Future<List<Map<String, dynamic>>> getUsers() async {
    Database? temp = await mydbcheck();
    var response = await temp!.rawQuery('SELECT * FROM users');
    return response;
  }

  //used in profile
  Future<void> printTableContents() async {
    // Print the contents of the users table
    List<Map<String, dynamic>> users = await getUsers();

    users.forEach((user) {
      print('from local Database User ID: ${user['id']}, Name:
${user['name']}, Email: ${user['email']}, Phone: ${user['phone']}');
    });
  }

  //used in profile
  Future<List<Map<String, dynamic>>> getSpecificUser(String uid) async {
    Database? temp = await mydbcheck();
    var response = await temp!.rawQuery('')

```



```

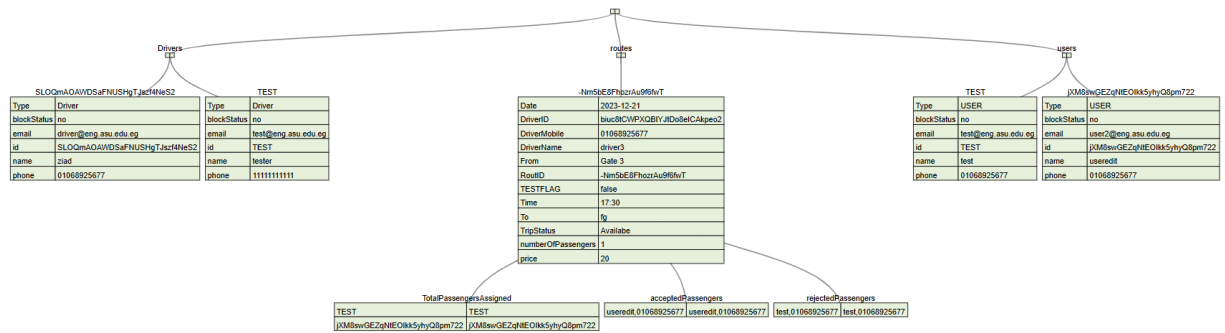
        SELECT * FROM users WHERE id = ?
        '', [uid]);
    return response;
}

Future<void> InsertOrUpdateUser(var uid,String username,String mobile)
async {
    Database? temp = await mydbcheck();
    if(TESTMODE==0){
        var response = await temp!.rawInsert('''
        INSERT OR REPLACE INTO users (id, name, phone)
        VALUES (?, ?, ?)
        '', [uid, username, mobile]);
        // Print table contents after updating
        await printTableContents();
    }
    else{
        var response = await temp!.rawInsert('''
        INSERT OR REPLACE INTO users (id, name, phone)
        VALUES (?, ?, ?)
        '', ["TEST", username, mobile]);
        // Print table contents after updating
        await printTableContents();
    }
}
}

```

## Remote Database Structure and code

Example at some instance



## Code Authentication class For Passenger

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_database/firebase_database.dart';
import 'package:flutter/material.dart';
import 'package:project/home_screen.dart';
import '../reusable/reusable_methods.dart';

class Authentication_class {
  ReusableMethods rMethods = ReusableMethods();

  Future<int> Sign_up(String emailTextEditingController
    , String passwordTextEditingController
    , String nameTextEditingController
    , String phoneTextEditingController
    , BuildContext context) async {
    final User? userFirebase = (
      await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: emailTextEditingController,
        password: passwordTextEditingController,
      ).catchError((errorMsg) {
        rMethods.displaySnackBar(errorMsg.toString(), context);
      })
    ).user;
    if (!context.mounted) return 0;

    DatabaseReference usersRef = FirebaseDatabase.instance.ref().child(
      "users").child(userFirebase!.uid);
    Map userDataMap = {
      "name": nameTextEditingController,
      "email": emailTextEditingController,
      "phone": phoneTextEditingController,
      "id": userFirebase.uid,
      "Type": "USER",
      "blockStatus": "no",
    };
    usersRef.set(userDataMap);

    Navigator.pushReplacement(
      context, MaterialPageRoute(builder: (c) => MyScreen()));
    return 0;
  }

  Log_in(String emailTextEditingController
    , String passwordTextEditingController
    , BuildContext context) async {
    final User? userFirebase = (
      await FirebaseAuth.instance.signInWithEmailAndPassword(
        email: emailTextEditingController,
        password: passwordTextEditingController,
      ).catchError((errorMsg) {
        rMethods.displaySnackBar(errorMsg.toString(), context);
      })
    ).user;
  }
```

```

    if (!context.mounted) return 0;

    if (userFirebase != null) {
      DatabaseReference DriverRef = FirebaseDatabase.instance.ref().child(
        "users").child(userFirebase!.uid);
      DriverRef.once().then((snap) {
        if (snap.snapshot.value != null) {
          if ((snap.snapshot.value as Map)["blockStatus"] == "no") {
            Navigator.pushReplacementNamed(context, '/home_screen');
          }
          else {
            FirebaseAuth.instance.signOut();
            rMethods.displaySnackBar("This Account Is Blocked", context);
          }
        } else {
          FirebaseAuth.instance.signOut();
          rMethods.displaySnackBar("The Account Not Found As Passenger",
context);
        }
      });
    }
  }

  Sign_out() async {
    await FirebaseAuth.instance.signOut();
  }
}

```

## Code Authentication class For Driver

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

import 'package:driver_app/home_screen.dart';
import 'package:driver_app/reusable/reusable_methods.dart';

import '../Test_file/GlobalVariableForTesting.dart';

class Authentication_class{
  ReusableMethods rMethods = ReusableMethods();

  Future<int> Sign_up(String emailTextEditingController
    ,String passwordTextEditingController
    ,String nameTextEditingController
    ,String phoneTextEditingController
    ,BuildContext context) async{
    final User? userFirebase = (
      await FirebaseAuth.instance.createUserWithEmailAndPassword(
        email: emailTextEditingController,
        password: passwordTextEditingController,
      ).catchError((errorMsg){
        rMethods.displaySnackBar(errorMsg.toString(), context);
        return 0;
      })
    ).user;
    if(!context.mounted) return 0;

    if(emailTextEditingController.trim() != "test@eng.asu.edu.eg"){
      DatabaseReference DriverRef =
        FirebaseDatabase.instance.ref().child("Drivers").child(userFirebase!.uid);
      Map driverDataMap = {
        "name":nameTextEditingController,
        "email":emailTextEditingController,
        "phone":phoneTextEditingController,
        "id":userFirebase.uid,
        "Type":"Driver",
        "blockStatus":"no",
      };
      DriverRef.set(driverDataMap);

      Navigator.pushReplacement(context,MaterialPageRoute(builder:
(c)=>MyScreen()));
      TESTMODE =0;
      return 1;
    }
    else{
      print("saving Test Node in Database");
      DatabaseReference DriverRef =
        FirebaseDatabase.instance.ref().child("Drivers").child("TEST");
      Map driverDataMap = {
        "name":nameTextEditingController,
        "email":emailTextEditingController,
        "phone":phoneTextEditingController,
        "id":"TEST",
      };
    }
  }
}
```

```

        "Type": "Driver",
        "blockStatus": "no",
    };
    DriverRef.set(driverDataMap);

    Navigator.pushReplacement(context, MaterialPageRoute(builder:
(c) => MyScreen()));
    TESTMODE = 1;
    return 1;
}

}

Log_in(String emailTextEditingController
, String passwordTextEditingController
, BuildContext context) async {

    final User? userFirebase = (
        await FirebaseAuth.instance.signInWithEmailAndPassword(
            email: emailTextEditingController,
            password: passwordTextEditingController,
        ).catchError((errorMsg) {
            rMethods.displaySnackBar(errorMsg.toString(), context);
        })
    ).user;

    if(!context.mounted) return;

    if(userFirebase != null) {
        DatabaseReference DriverRef =
        FirebaseDatabase.instance.ref().child("Drivers").child(userFirebase!.uid);
        DriverRef.once().then((snap) {
            if(snap.snapshot.value != null) {

                if((snap.snapshot.value as Map) ["blockStatus"] == "no") {
                    Navigator.pushReplacementNamed(context, '/home_screen');
                }
                else {
                    FirebaseAuth.instance.signOut();
                    rMethods.displaySnackBar("This Account Is Blocked", context);
                }
            }
            else {
                FirebaseAuth.instance.signOut();
                rMethods.displaySnackBar("The Account Not Found As Driver",
context);
            }
        }

    });
}

```

```
}

}

Sign_out() async{
  await FirebaseAuth.instance.signOut();
}

}
```

## Test Case scenarios

### 1)Group of test scenarios for sign-Up page

#### scenario 1.1:

- Open Application
- Go to sign-Up page
- Press sign Up button

EXPECTED: message "Name must be at least 4 characters"

#### scenario 1.2:

- Open Application
- Go to sign-Up page
- Enter valid name with more than 3 characters
- Press sign Up button

EXPECTED: message "Phone number must be at least 11 digits"

#### Scenario 1.3:

- Open Application
- Go to sign-Up page
- Enter valid name with more than 3 characters
- Enter valid Phone number with more than or equal 11 digits
- Press sign Up button

EXPECTED: message "Please sign up with ASU Domain email"

#### Scenario 1.4:

- Open Application
- Go to sign-Up page
- Enter valid name with more than 3 characters
- Enter valid Phone number with more than or equal 11 digits
- Enter Valid Email but not ASU domain
- Press sign Up button

EXPECTED: message "Please sign up with ASU Domain email"



Scenario 1.5:

- Open Application
- Go to sign-Up page
- Enter valid name with more than 3 characters
- Enter valid Phone number with more than or equal 11 digits
- Enter valid email address With ASU domain
- Press sign Up button

EXPECTED: message "Password must be atleast 6 characters"

Scenario 1.6:

- Open Application
- Go to sign-Up page
- Enter valid name with more than 3 characters
- Enter valid Phone number with more than or equal 11 digits
- Enter valid email address With ASU domain
- Enter valid password with more than or equal 11 characters
- Press sign Up button

EXPECTED: message "Confirm password is not the same"

Scenario 1.7:

- Open Application
- Go to sign-Up page
- Enter valid name with more than 3 characters
- Enter valid Phone number with more than or equal 11 digits
- Enter valid email address With ASU domain
- Enter valid password with more than or equal 11 characters
- Enter valid Confirm password equals the Password entered
- Press sign Up button

EXPECTED:

- Sign Up successfully
  - Navigation to HomeScreen
-

## 2) Test case Login

### Scenario 2.1 Entering empty fields

- Open Application
- Press Login

Expected Message "Please Login with ASU Domain Email"

### Scenario 2.2 Enter valid data

- Open application
- Enter valid email and password
- Press Login

Expected

- Logged in successfully
  - Navigated to HomePage
- 

### Scenario 2.3 enter Passenger account in Driver App

- Open Driver application
- Enter valid email and password but For passenger
- Press Login

Expected message "this account is nor Driver account"

### Scenario 2.4 enter Driver account in Passenger App

- Open Passenger application
- Enter valid email and password but For driver
- Press Login

Expected message "this account is nor Passenger account"

---

## 2)Test case :Add ride For Driver

### Scenario 2.1:

- Press Add Button
- Select date
- Select time
- Select Destination
- Enter pickup point
- Enter Price
- Click Save Trip

#### Expected

- Ride added to database with correct data
- Navigated to homeScreen
- Ride appears for passenger

### Scenario 2.2

- Press Add Button
- Leave one or more fields empty

#### Expected

- Message “one or more fields empty”
- 

## 3) Test case :Driver Navigate to profile Page

### Scenario 3.1

- Make sure internet connection available
- Press menu button then select Profile Page

#### Expected:

- Email, username and mobile number appears and correct
- And the data retrieved from Remote database

### Scenario 3.2

- Make sure internet connection is OFF
- Press menu button then select Profile Page

#### Expected

- Email, username and mobile number appears and correct

- And the data retrieved from Local database
- 

#### 4) Test Case :Edit Profile Page

##### Scenario 4.1

- Click the field edit icon
- Enter a new String
- Click Save

##### Expected

- Field displayed with the edited text
- Field updated in firebase
- Field updated in local database

##### Scenario 4.2

- Click the field edit icon
- Enter a Empty
- Click Save

##### Expected

- Message "field can't be empty"

##### Scenario 4.3

- Click the field edit icon
- Click Cancel button

##### Expected

- Back to profile Context

#### 5) Test case: Driver Time Constrains

##### Scenario 5.1: prerequisite Ride Created with time 7:30 and multiple passengers request that ride

- Select time button 16:45(or any time before 23:30)
- Click on the target ride

##### Expected

- Driver can accept or reject passengers

##### Scenario 5.2: prerequisite Ride Created with time 7:30 and multiple passengers request that ride

- Select time button 23:45
- Click on the target ride

##### Expected

- All passengers that was pending have being rejected automatically
- Driver can't accept or reject passengers

Scenario 5.3: prerequisite Ride Created with time 17:30 and multiple passengers request that ride

- Select time button 1:30(or any time before 16:30)
- Click on the target ride

Expected

- Driver can accept or reject passengers

Scenario 5.4: prerequisite Ride Created with time 17:30 and multiple passengers request that ride

- Select time button 16:45
- Click on the target ride

Expected

- All passengers that was pending have being rejected automatically
- Driver can't accept or reject passengers

## 6) Test Case : Logout for both drivers/passengers

Scenario 6.1

- Click menu button then select Logout

Expected

- Navigate to Login Screen
  - Close the app when press back button
- 

## 7) Test case Request ride in passenger App

Scenario 7.1

- Open Passenger App
- Select target ride
- Select payment method
- Press "proceed to payment"

Expected

- Passenger assigned to the route in database
- Navigate to home page
- Ride is added in order history page

Scenario 7.2 Ride is fullyBooked

- Open Passenger App
- Select target ride
- Select payment method
- Press "proceed to payment"

Expected message "this trip is FullyBooked"

## 8)Test case Accept/reject ride in Driver App

### Scenario 8.1

- Open driver App
- Check specific ride
- Click Accept Button of specific request

Expected

- User request turned into green
- User request removed from pending requests
- routes node child acceptedPassengers has username and mobile
- routes node child passengers doesn't have username and mobile anymore
- routes node child totalAssignedpassengers has username and mobile
- routes node child noOfAcceptedPassengeres value increased by 1

### Scenario 8.2

- Open driver App
- Check specific ride
- Click reject Button of specific request

Expected

- User request disappered
- User request removed from pending requests
- routes node child rejectedPassengers has username and mobile
- routes node child passengers doesn't have username and mobile anymore
- routes node child totalAssignedpassengers has username and mobile

### Scenario 8.3 noOfAcceptedPassengeres value = 4

- Open driver App
- Check specific ride
- Click Accept Button of specific request

Expected

- Message "You already accepted 4 passengers"
-

## 9)Test case Passenger check the assigned ride

Scenario 9.1 passenger accepted by driver at specific ride

- Open passenger App
- Press menu button and select order history
- Select specific ride

Expected

- See username and mobile and status accepted in green tile

Scenario 9.1 passenger rejected by driver at specific ride

- Open passenger App
- Press menu button and select order history
- Select specific ride

Expected

- See username and mobile and status rejected in red tile

## 10)Test case : Passenger Time Constrains on reserve a ride

Scenario 10.1 prerequisite ride at 7:30 In the next day is created

- Open passenger App
- Select bypass time to be 21:30

Expect

- To see the ride in homescreen normally
- Click on the ride
- Navigate to cart page normally

Scenario 10.2 prerequisite ride at 7:30 In the next day is created

- Open passenger App
- Select bypass time to be 22:30

Expected

- Do not see the ride in homescreen
- So passenger cant request it

Scenario 10.3 prerequisite ride at 17:30 In the same day is created

- Open passenger App
- Select bypass time to be 13:30

Expected

- Do not see the ride in homescreen

- So passenger cant request it

Scenario 10.4 prerequisite ride at 17:30 In the next day is created

- Open passenger App
- Select bypass time to be 12:30

Expect

- To see the ride in homescreen normally
- Click on the ride
- Navigate to cart page normally

Scenario 10.5 prerequisite ride at 7:30 In the next day is created

- Open passenger App
- Select bypass time to be 1:30

Expect

- Do not see the ride in homescreen
- So passenger can't request it



# APK build

