AIN SHAMS UNIVERSITY
FACULTY OF ENGINEERING
Credit Hours Engineering Programs - CHEP
Computer Engineering and Software Systems
Engineering Programs

CSE431: Mobile Programming
Fall 2023

Final Delivery

Carpool App

Submitted by:

Basmala Abdullah Ahmed Hassan **19P2617**

Table of Contents

Introduction:

In response to the growing demand for efficient and student-centric transportation services, we are delighted to present the Carpool Rideshare Application tailored exclusively for the Faculty of Engineering Community at Ainshams University. This revolutionary application is designed to simplify the commuting experience for students, fostering a sense of community and reliability through a closed network of trusted users.

The Carpool app, operated by students for students, introduces a novel strategy in recruiting drivers and orchestrating rides, creating a platform that not only addresses transportation needs but also promotes a sense of camaraderie within the academic community. With a specific focus on enhancing user experience, the app establishes a trusted closed community by requiring users to sign in with their active @eng.asu.edu.eg accounts, ensuring a secure and reliable environment.

In this pilot project, we have set forth a streamlined schedule for rides, incorporating one morning ride at 7:30 am from various locations to Gate 3 or Gate 4, and one return ride at 5:30 pm from the same gates. To regulate this service, customers are required to reserve their seats by specific deadlines — before 10:00 pm the previous day for the morning ride and before 1:00 pm on the same day for the afternoon ride.

This report details the development process and features of the Carpool Rideshare App. From user authentication and route listings to reservation systems, payment processing, and real-time order tracking, the app aims to provide a comprehensive solution to the transportation needs of the Faculty of Engineering Community. Leveraging Firebase for user authentication and real-time database functionalities, as well as Room Database for profile data management, the application embodies cutting-edge technologies to ensure efficiency and reliability.

As we unveil this innovative rideshare platform, we anticipate that the Carpool app will not only streamline transportation logistics but also contribute to the sense of community and collaboration within the Faculty of Engineering at Ainshams University. This report offers insights into the development process, features, and functionalities of the app, setting the stage for a new era of student-driven transportation solutions.

App Features:

1- User Authentication:

Description: Users are required to log in using their @eng.asu.edu.eg email accounts, ensuring a closed and trusted community.

Benefits: Enhances security and builds a reliable network within the Faculty of Engineering Community.

2- Routes Listing:

Description: Displays a list of available routes to and from Gate 3 or Gate 4 using a user-friendly recycler view.

Benefits: Facilitates easy navigation and selection of preferred commuting routes.

3- Reservation System:

Description: Allows users to reserve seats for rides, with specific deadlines for morning and afternoon trips.

Benefits: Ensures timely planning and seat availability, optimizing the commuting experience.

4- Cart and Payments:

Description: Users can review their ride selections in a cart and proceed to make payments.

Benefits: Streamlines the booking process.

5- Order History:

Description: Provides a comprehensive view of users' ride history and a real-time status page for upcoming orders.

Benefits: Enhances user transparency and allows for efficient tracking of ride statuses.

6- Firebase Real-time Database Integration:

Description: Utilizes Firebase for real-time updates on route information and order statuses.

Benefits: Ensures the latest and most accurate information is available to users and drivers.

7- Room Database for Profile Data:

Description: Utilizes Room Database to manage and store user profile data securely.

Benefits: Safely stores user information, enhancing data integrity and accessibility.

8- Driver Application:

Description: Employs a separate application for drivers to confirm orders and update status data.

Benefits: Enables efficient coordination between drivers and users, ensuring timely order confirmations.

9- Driver Order Confirmation Deadline Management:

Description: Allows drivers to confirm ride orders before specific deadlines for morning and afternoon rides.

Benefits: Ensures timely coordination and confirmation of ride requests, optimizing the scheduling process.

10- Driver Order History:

Description: Offers drivers access to their order.

Benefits: Allows drivers to review their scheduled rides.

11- Driver Profile Management:

Description: Allows drivers to manage their profiles, including updating personal information.

Benefits: Ensures that driver profiles remain accurate.

12- Driver Real-time Order Updates:

Description: Provides drivers with real-time updates on order statuses, ensuring they have the latest information on confirmed rides.

Benefits: Enables drivers to plan their routes and schedules effectively.

13- Scheduled Rides:

Description: Implements fixed start times for morning and afternoon rides, adding predictability to the service.

Benefits: Establishes a structured and dependable transportation schedule for users.

Test Cases:

Test Account:

Email: test@eng.asu.edu.eg

Password: 123456

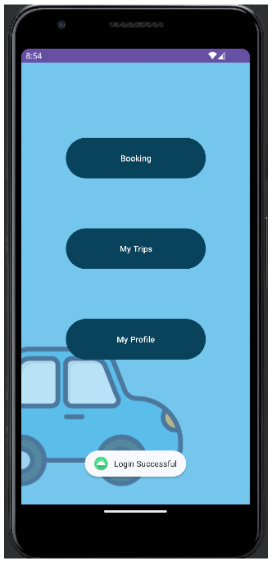
Note: The application can be and is tested by changing the system time of the emulator device.

User Authentication:

1- Valid Login Credentials:

Input: Valid @eng.asu.edu.eg email and password.

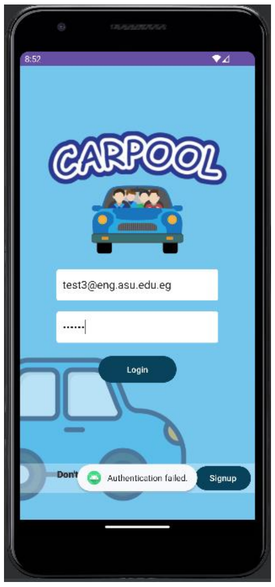
Expected Output: Successfully logged in.



2- Invalid Login Credentials:

Input: Invalid email or password.

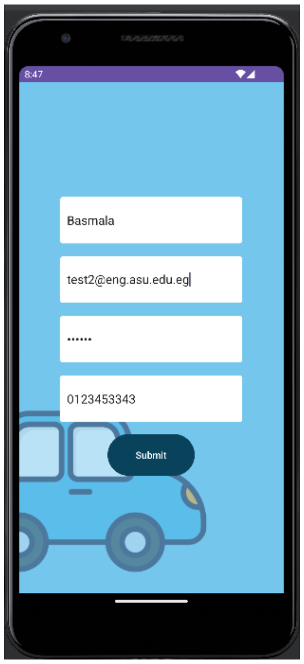
Expected Output: Display error message and prevent login.



3- Sign Up:

Input: Valid @eng.asu.edu.eg email and password.

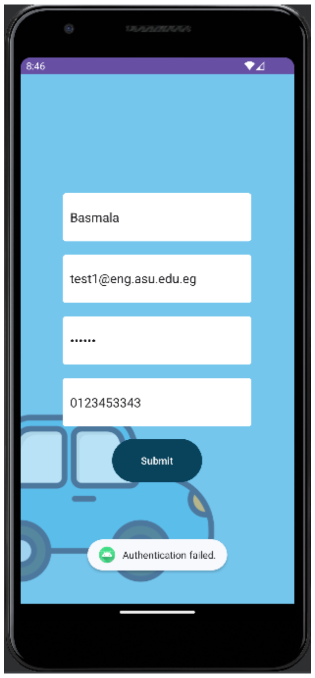
Expected Output: New account created successfully.



4- Sign Up with Existing Email:

Input: Email already registered.

Expected Output: Display error message.


8:46

Basmala

test1@eng.asu.edu.eg

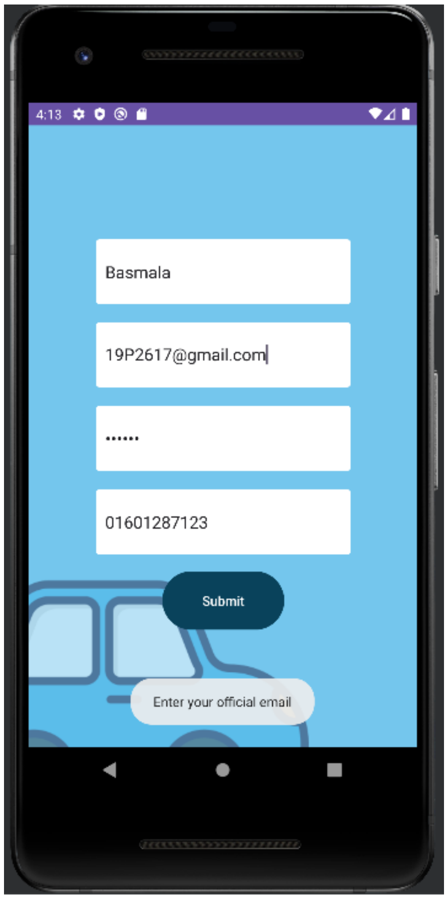
••••••

0123453343

Authentication failed.

5- Signup with non-official mail

If the user didn't enter his official email, a toast message appears saying "Enter your official email." And failing the signup process.


4:13

Basmala

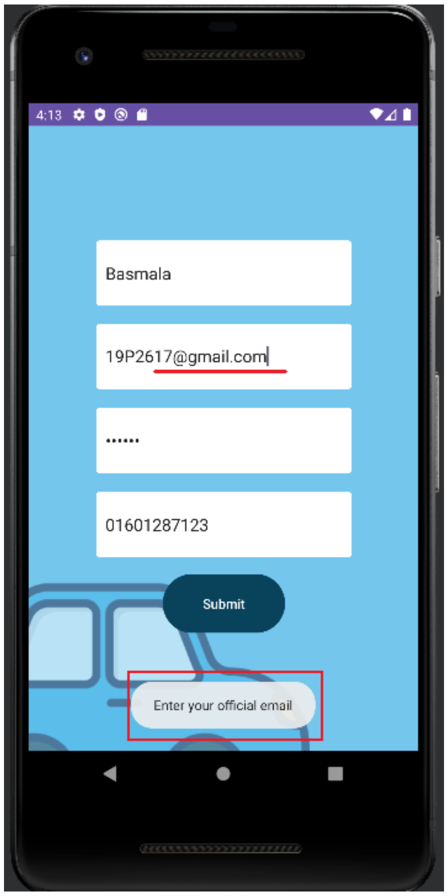
19P2617@gmail.com

••••••

01601287123

Submit

Enter your official email


4:13

Basmala

19P2617@gmail.com

••••••

01601287123

Submit

Enter your official email

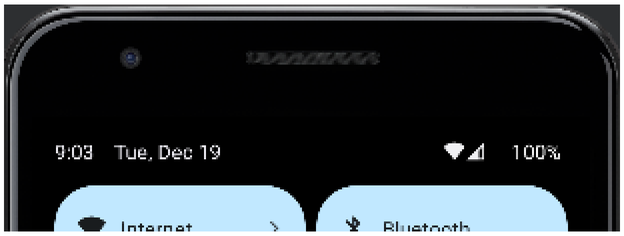
Rider Ride Reservation:

1- Morning Ride Reservation:

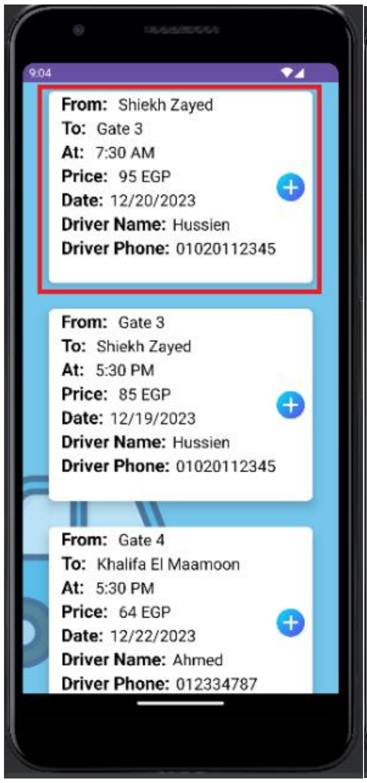
Input: Reserve a seat for the morning ride before 10:00 pm the previous day.

Expected Output: Reservation successful.

Note that the date is 19 Dec at 9:03 PM and compare it with the date and time of the ride:


9:03 Tue, Dec 19 100%

Choosing the highlighted ride then navigating to Cart then submitting the order:

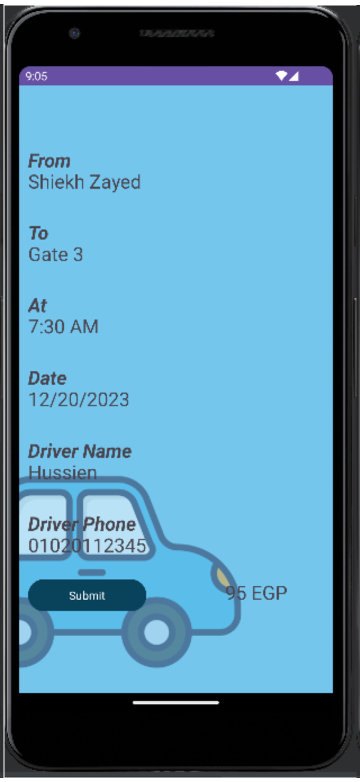


9:04

From: Shiekh Zayed
To: Gate 3
At: 7:30 AM
Price: 95 EGP
Date: 12/20/2023
Driver Name: Hussien
Driver Phone: 01020112345

From: Gate 3
To: Shiekh Zayed
At: 5:30 PM
Price: 85 EGP
Date: 12/19/2023
Driver Name: Hussien
Driver Phone: 01020112345

From: Gate 4
To: Khalifa El Maamoon
At: 5:30 PM
Price: 64 EGP
Date: 12/22/2023
Driver Name: Ahmed
Driver Phone: 012334787



9:05

From
Shiekh Zayed

To
Gate 3

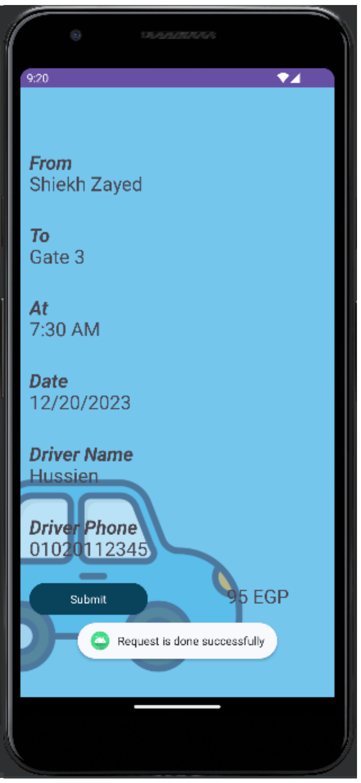
At
7:30 AM

Date
12/20/2023

Driver Name
Hussien

Driver Phone
01020112345

95 EGP



9:20

From
Shiekh Zayed

To
Gate 3

At
7:30 AM

Date
12/20/2023

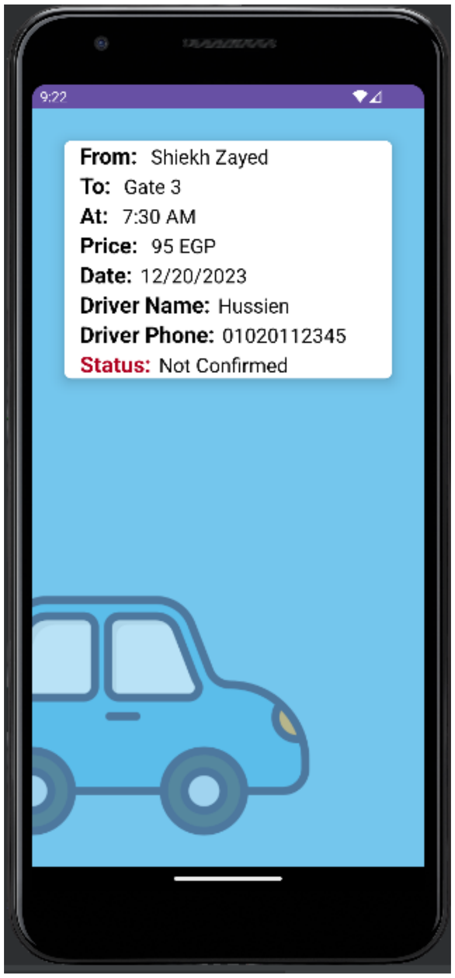
Driver Name
Hussien

Driver Phone
01020112345

95 EGP

Request is done successfully

The reserved ride is now shown in the pending trips of the rider after navigating to Dashboard then “My Trips” then “Pending Trips”


9:22

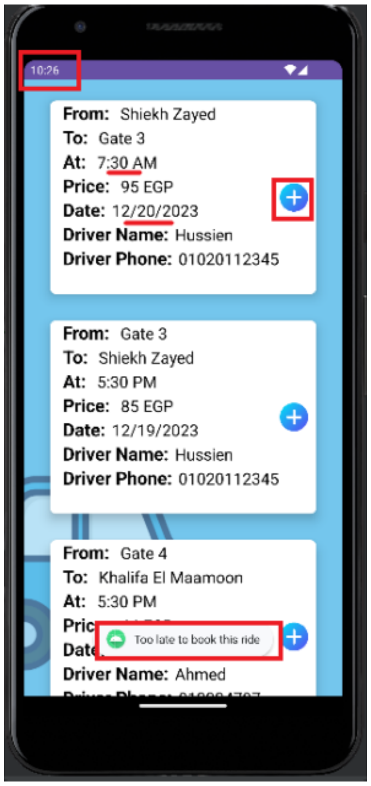
From: Shiekh Zayed
To: Gate 3
At: 7:30 AM
Price: 95 EGP
Date: 12/20/2023
Driver Name: Hussien
Driver Phone: 01020112345
Status: Not Confirmed

2- Late Morning Reservation:

Input: Attempt to reserve a seat for the morning ride after 10:00 pm the previous day.

Expected Output: Display error message, reservation not allowed.

The device date is 19 Dec on 10:26 PM and tries to reserve the morning ride on next day:



10:26

From: Shiekh Zayed
To: Gate 3
At: 7:30 AM
Price: 95 EGP
Date: 12/20/2023
Driver Name: Hussien
Driver Phone: 01020112345

From: Gate 3
To: Shiekh Zayed
At: 5:30 PM
Price: 85 EGP
Date: 12/19/2023
Driver Name: Hussien
Driver Phone: 01020112345

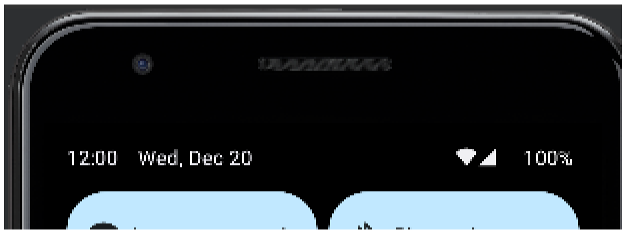
From: Gate 4
To: Khalifa El Maamoon
At: 5:30 PM
Pric
Date
Driver Name: Ahmed

Too late to book this ride

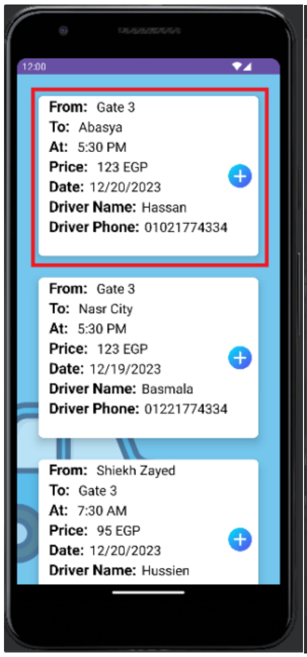
3- Afternoon Ride Reservation:

Input: Reserve a seat for the afternoon ride before 1:00 pm on the same day.

Expected Output: Reservation successful.



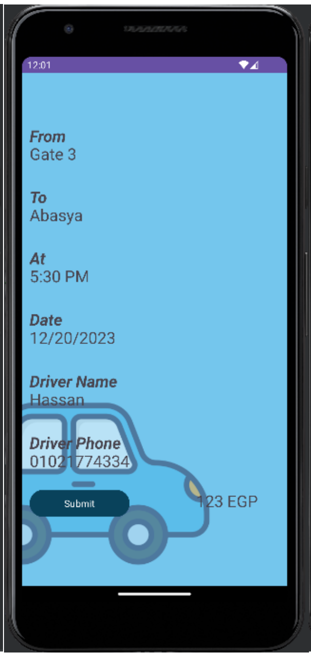
12:00 Wed, Dec 20 100%


12:00

From: Gate 3
To: Abasya
At: 5:30 PM
Price: 123 EGP
Date: 12/20/2023
Driver Name: Hassan
Driver Phone: 01021774334

From: Gate 3
To: Nasr City
At: 5:30 PM
Price: 123 EGP
Date: 12/19/2023
Driver Name: Basmala
Driver Phone: 01221774334

From: Shiekh Zayed
To: Gate 3
At: 7:30 AM
Price: 95 EGP
Date: 12/20/2023
Driver Name: Hussien


12:01

From
Gate 3

To
Abasya

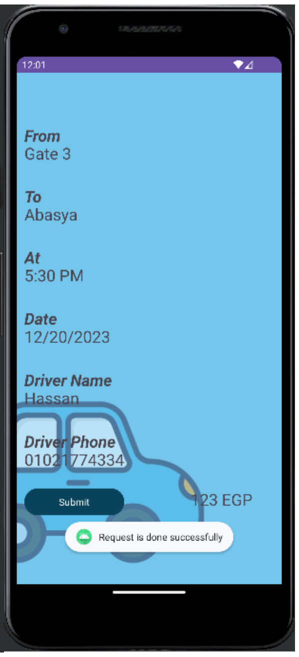
At
5:30 PM

Date
12/20/2023

Driver Name
Hassan

Driver Phone
01021774334

123 EGP


12:01

From
Gate 3

To
Abasya

At
5:30 PM

Date
12/20/2023

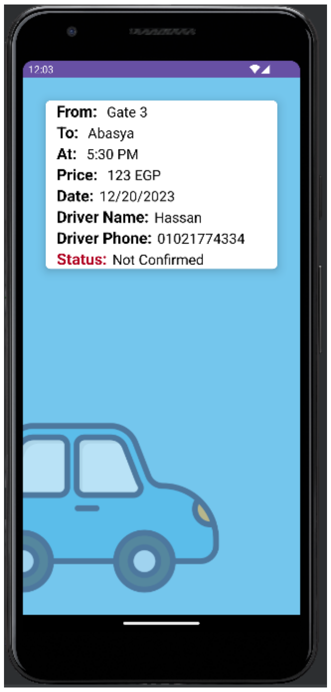
Driver Name
Hassan

Driver Phone
01021774334

123 EGP

Request is done successfully

The reserved ride is now shown in the pending trips of the rider after navigating to Dashboard then “My Trips” then “Pending Trips”


12:03

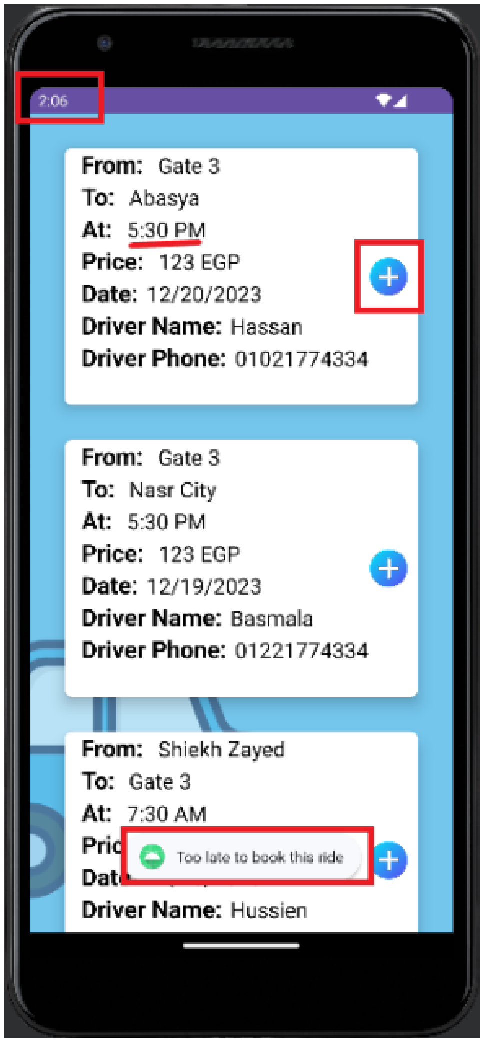
From: Gate 3
To: Abasya
At: 5:30 PM
Price: 123 EGP
Date: 12/20/2023
Driver Name: Hassan
Driver Phone: 01021774334
Status: Not Confirmed

Late Afternoon Ride Reservation:

Input: Reserve a seat for the afternoon ride After 1:00 pm on the same day.

Expected Output: Display error message, reservation not allowed.

The device date is 20 Dec on 2:06 PM and tries to reserve the afternoon ride on same day:



2:06

From: Gate 3
To: Abasya
At: 5:30 PM
Price: 123 EGP
Date: 12/20/2023
Driver Name: Hassan
Driver Phone: 01021774334

From: Gate 3
To: Nasr City
At: 5:30 PM
Price: 123 EGP
Date: 12/19/2023
Driver Name: Basmala
Driver Phone: 01221774334

From: Shiekh Zayed
To: Gate 3
At: 7:30 AM
Pric
Dat
Driver Name: Hussien

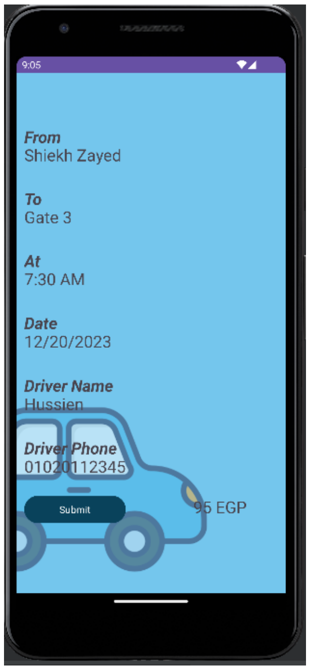
Too late to book this ride

Cart and Payments:

Review Order:

Input: Navigate to the cart and review the order.

Expected Output: Display a summary of the selected ride.


From
Shiekh Zayed

To
Gate 3

At
7:30 AM

Date
12/20/2023

Driver Name
Hussien

Driver Phone
01020112345

95 EGP

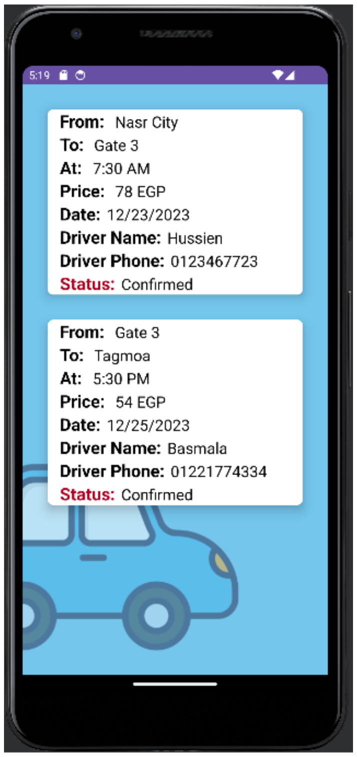
Rider Order History:

View Order History:

Input: Navigate to the “My Trips” order history section.

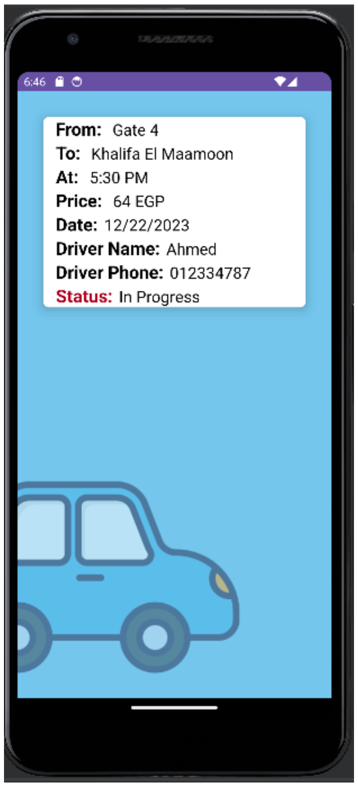
Expected Output: Display a list of rider’s ride orders of each status.

Pending:


From: Nasr City
To: Gate 3
At: 7:30 AM
Price: 78 EGP
Date: 12/23/2023
Driver Name: Hussien
Driver Phone: 0123467723
Status: Confirmed

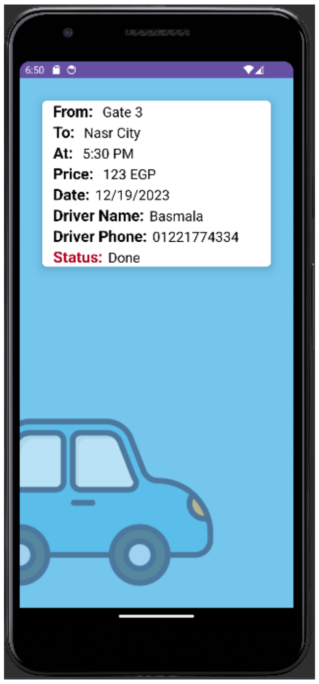
From: Gate 3
To: Tagmoa
At: 5:30 PM
Price: 54 EGP
Date: 12/25/2023
Driver Name: Basmala
Driver Phone: 01221774334
Status: Confirmed

In progress:


6:46

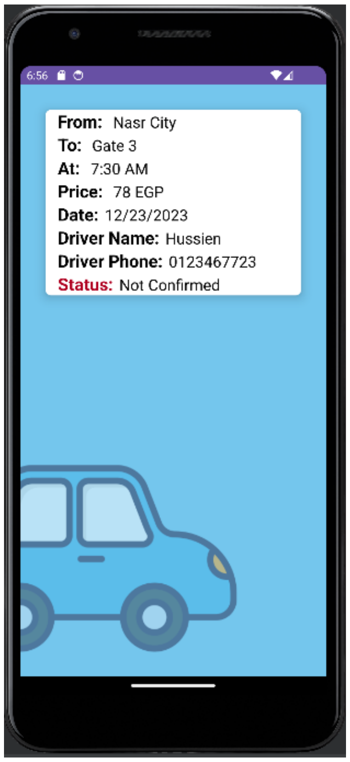
From: Gate 4
To: Khalifa El Maamoon
At: 5:30 PM
Price: 64 EGP
Date: 12/22/2023
Driver Name: Ahmed
Driver Phone: 012334787
Status: In Progress

Done:


6:50

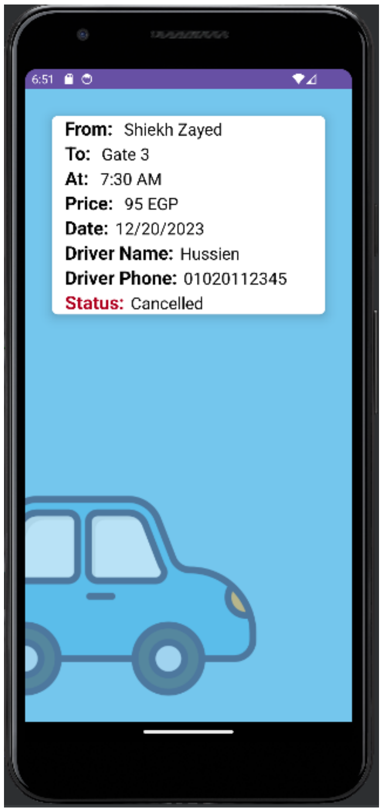
From: Gate 3
To: Nasr City
At: 5:30 PM
Price: 123 EGP
Date: 12/19/2023
Driver Name: Basmala
Driver Phone: 01221774334
Status: Done

Not confirmed:


6:56

From: Nasr City
To: Gate 3
At: 7:30 AM
Price: 78 EGP
Date: 12/23/2023
Driver Name: Hussien
Driver Phone: 0123467723
Status: Not Confirmed

Cancelled:


6:51

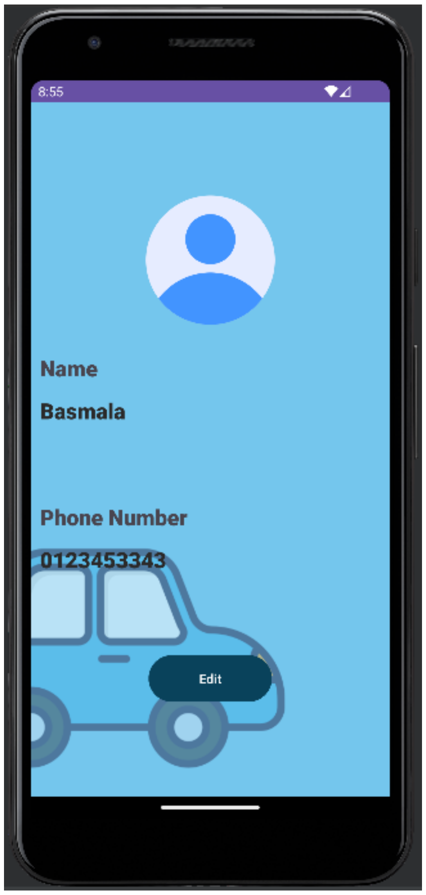
From: Shiekh Zayed
To: Gate 3
At: 7:30 AM
Price: 95 EGP
Date: 12/20/2023
Driver Name: Hussien
Driver Phone: 01020112345
Status: Cancelled

Room Database:

1- Save Profile Data:

Input: Save user profile data to the Room database.

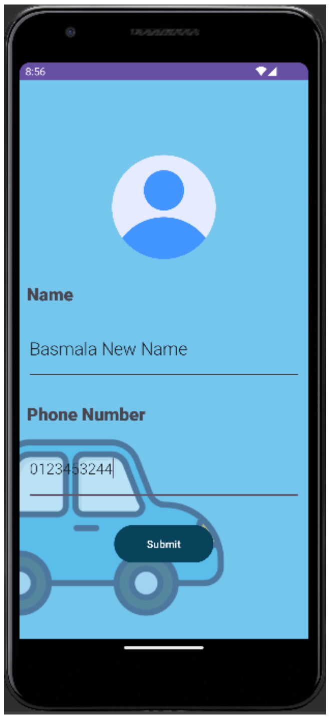
Expected Output: Profile data successfully saved.

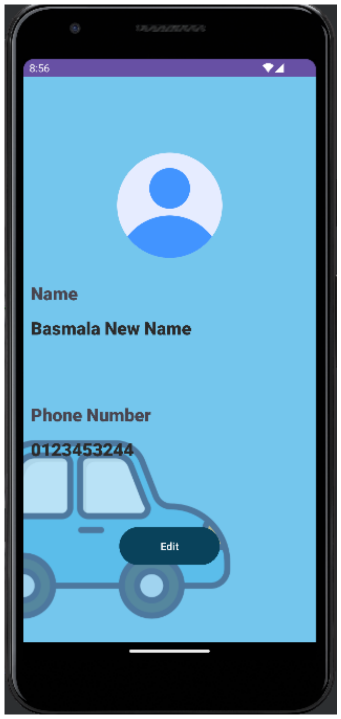


2- Edit Profile Data:

Input: Edit and Save user profile data to the Room database.

Expected Output: Profile data successfully edited.



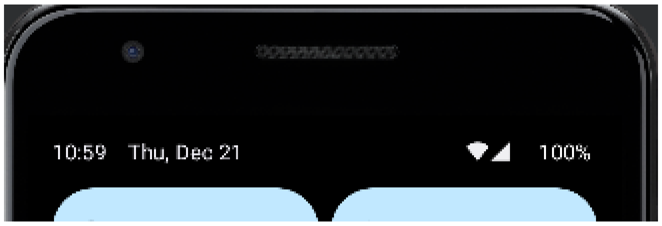


Driver Application:

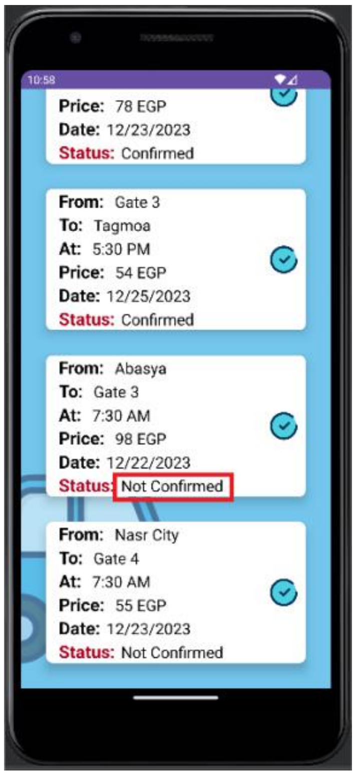
1- Confirm Morning Order:

Input: Confirm an order as a driver before 11:30 pm for the morning ride.

Expected Output: Order confirmed successfully.



Before pressing on (✓) icon:


10:58

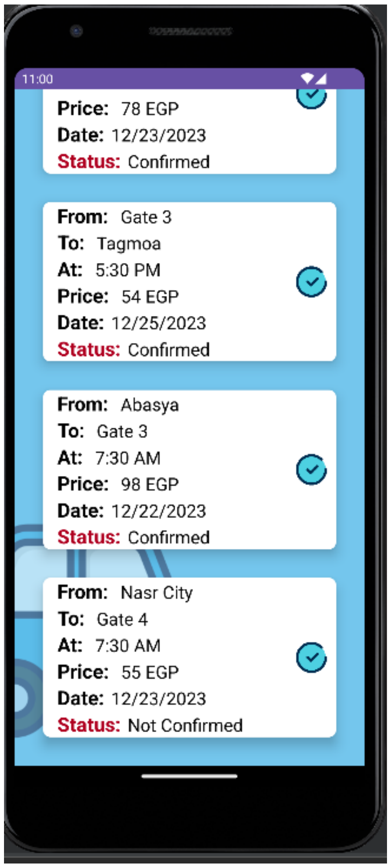
Price: 78 EGP
Date: 12/23/2023
Status: Confirmed

From: Gate 3
To: Tagmoa
At: 5:30 PM
Price: 54 EGP
Date: 12/25/2023
Status: Confirmed

From: Abasya
To: Gate 3
At: 7:30 AM
Price: 98 EGP
Date: 12/22/2023
Status: Not Confirmed

From: Nasr City
To: Gate 4
At: 7:30 AM
Price: 55 EGP
Date: 12/23/2023
Status: Not Confirmed

After:


11:00

Price: 78 EGP
Date: 12/23/2023
Status: Confirmed

From: Gate 3
To: Tagmoa
At: 5:30 PM
Price: 54 EGP
Date: 12/25/2023
Status: Confirmed

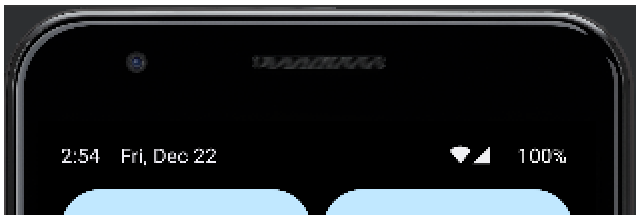
From: Abasya
To: Gate 3
At: 7:30 AM
Price: 98 EGP
Date: 12/22/2023
Status: Confirmed

From: Nasr City
To: Gate 4
At: 7:30 AM
Price: 55 EGP
Date: 12/23/2023
Status: Not Confirmed

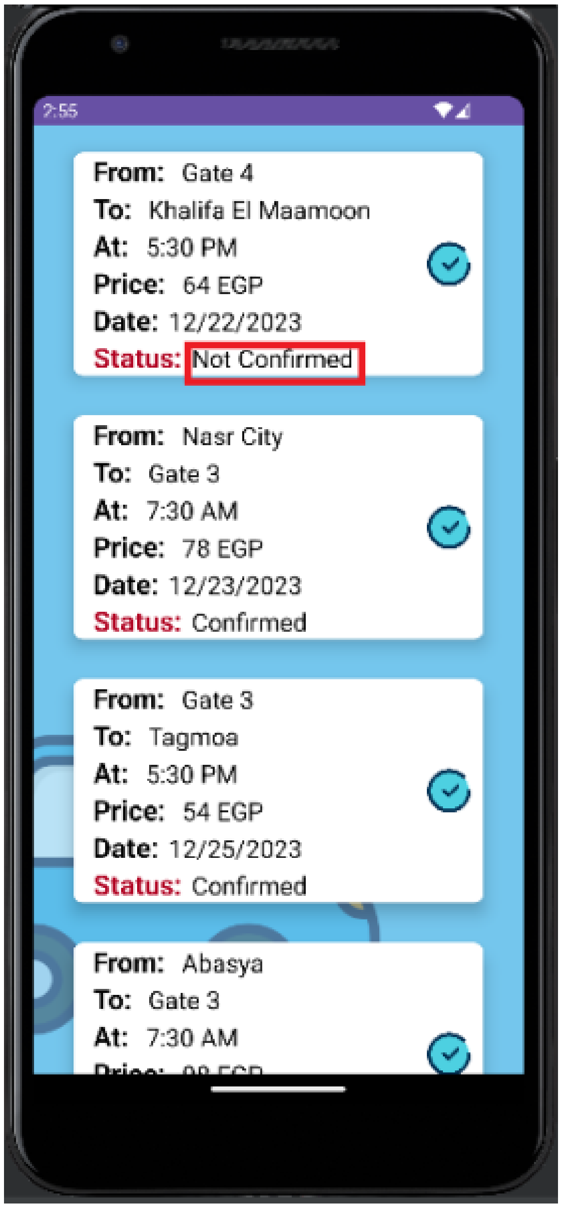
2- Confirm Afternoon Order:

Input: Confirm an order as a driver before 4:30 pm for the afternoon ride.

Expected Output: Order confirmed successfully.



Before pressing on (✓) icon:



2:55

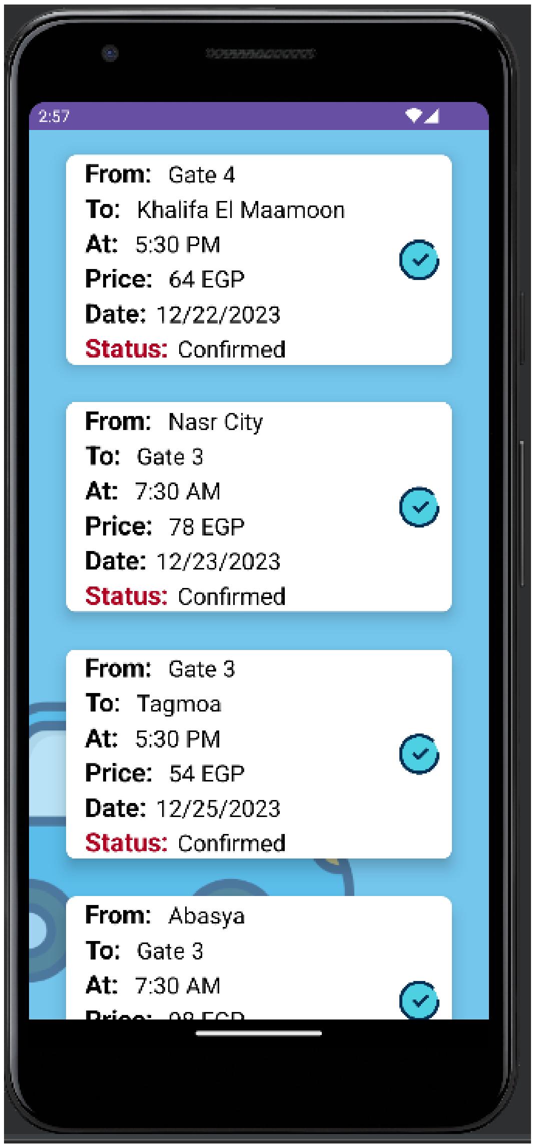
From: Gate 4
To: Khalifa El Maamoon
At: 5:30 PM
Price: 64 EGP
Date: 12/22/2023
Status: Not Confirmed

From: Nasr City
To: Gate 3
At: 7:30 AM
Price: 78 EGP
Date: 12/23/2023
Status: Confirmed

From: Gate 3
To: Tagmoa
At: 5:30 PM
Price: 54 EGP
Date: 12/25/2023
Status: Confirmed

From: Abasya
To: Gate 3
At: 7:30 AM

After:


2:57

From: Gate 4
To: Khalifa El Maamoon
At: 5:30 PM
Price: 64 EGP
Date: 12/22/2023
Status: Confirmed

From: Nasr City
To: Gate 3
At: 7:30 AM
Price: 78 EGP
Date: 12/23/2023
Status: Confirmed

From: Gate 3
To: Tagmoa
At: 5:30 PM
Price: 54 EGP
Date: 12/25/2023
Status: Confirmed

From: Abasya
To: Gate 3
At: 7:30 AM

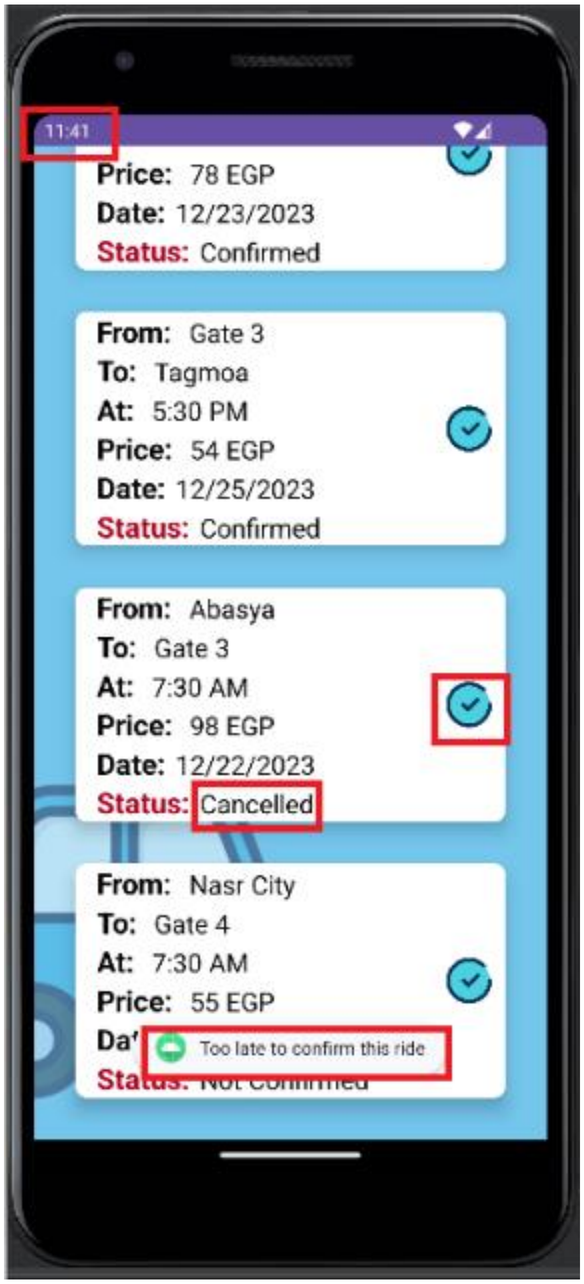
3- Late Morning Order Confirmation:

Input: Attempt to confirm an order after 11:30 pm for the morning ride.

Expected Output: Display error message, confirmation not allowed.

At 11:40 PM on 21 Dec 2023, the driver will try to confirm ride dated on 22 Dec 2023 at 7:30 AM

Any ride that was not confirmed, when deadline passes its status automatically is changed to cancelled, when the driver tries to confirm it, an error message is displayed saying" Too Late to confirm this ride"



11:41

Price: 78 EGP
Date: 12/23/2023
Status: Confirmed

From: Gate 3
To: Tagmoa
At: 5:30 PM
Price: 54 EGP
Date: 12/25/2023
Status: Confirmed

From: Abasya
To: Gate 3
At: 7:30 AM
Price: 98 EGP
Date: 12/22/2023
Status: Cancelled

From: Nasr City
To: Gate 4
At: 7:30 AM
Price: 55 EGP
Da
Status: Not Confirmed

Too late to confirm this ride

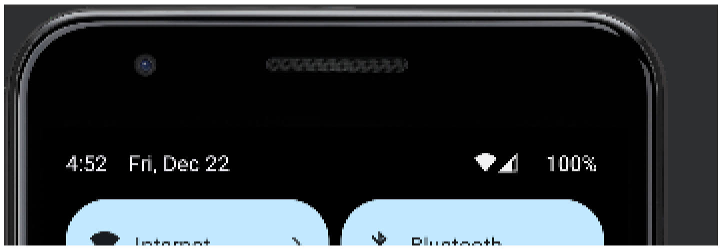
4- Late Afternoon Order Confirmation:

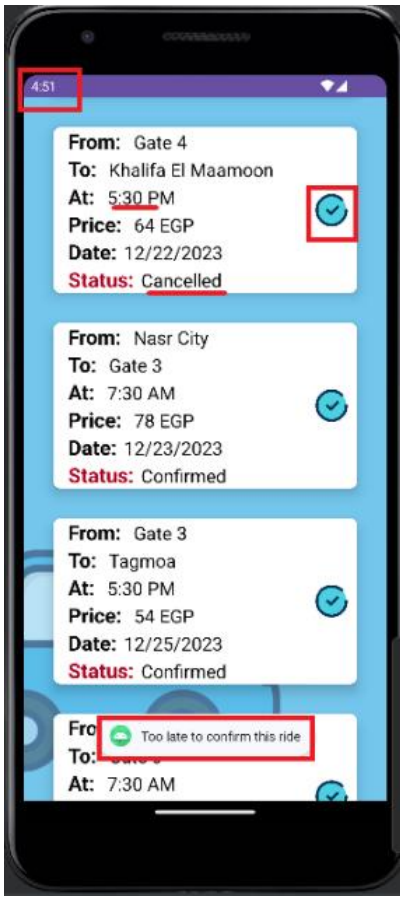
Input: Attempt to confirm an order after 4:30 pm for the afternoon ride.

Expected Output: Display error message, confirmation not allowed.

At 04:52 PM on 22 Dec 2023, the driver will try to confirm ride dated on 22 Dec 2023 at 5:30 PM.

Any ride that was not confirmed, when deadline passes, its status automatically is changed to cancelled, when the driver tries to confirm it, an error message is displayed saying" Too Late to confirm this ride"




4:51

From: Gate 4
To: Khalifa El Maamoon
At: 5:30 PM
Price: 64 EGP
Date: 12/22/2023
Status: Cancelled

From: Nasr City
To: Gate 3
At: 7:30 AM
Price: 78 EGP
Date: 12/23/2023
Status: Confirmed

From: Gate 3
To: Tagmoa
At: 5:30 PM
Price: 54 EGP
Date: 12/25/2023
Status: Confirmed

Fro
To:
At: 7:30 AM

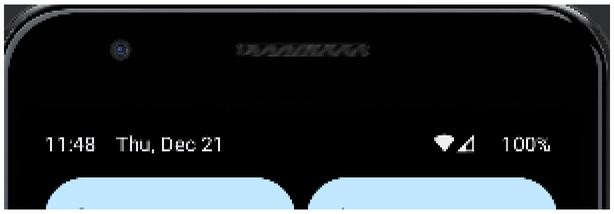
Too late to confirm this ride

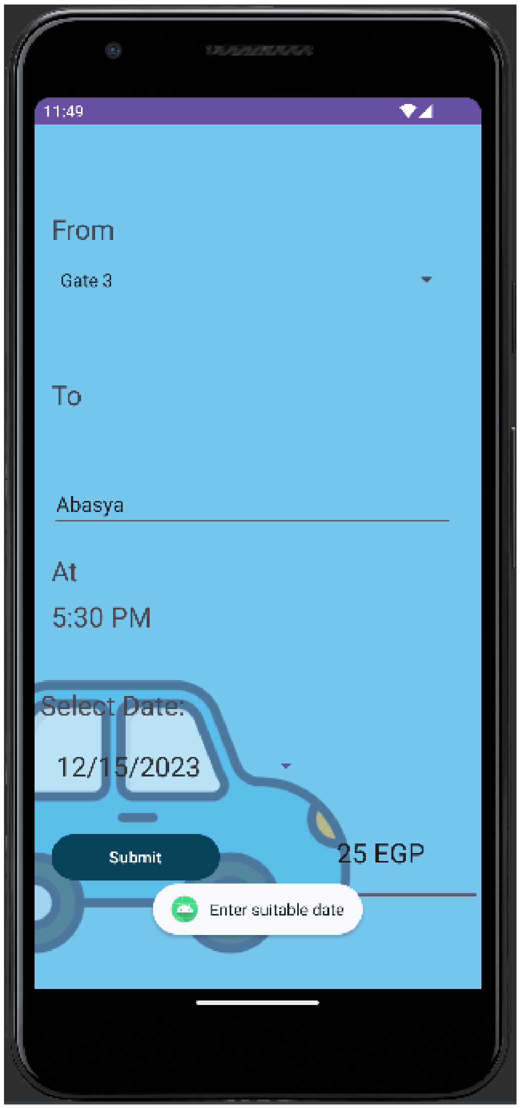
Driver Booking Page:

1- Late date order reservation:

Input: Trying to book a ride on an old date, On 21 Dec, The driver tries to book a ride on 15 Dec

Expected Output: Display error message to enter suitable date, reservation not allowed.





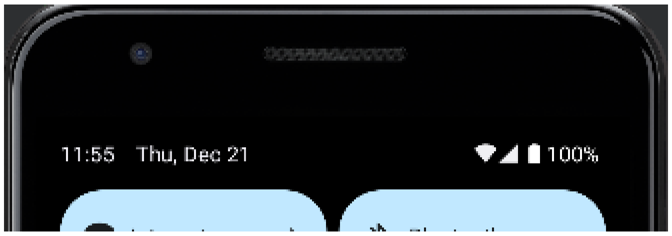
Enter suitable date

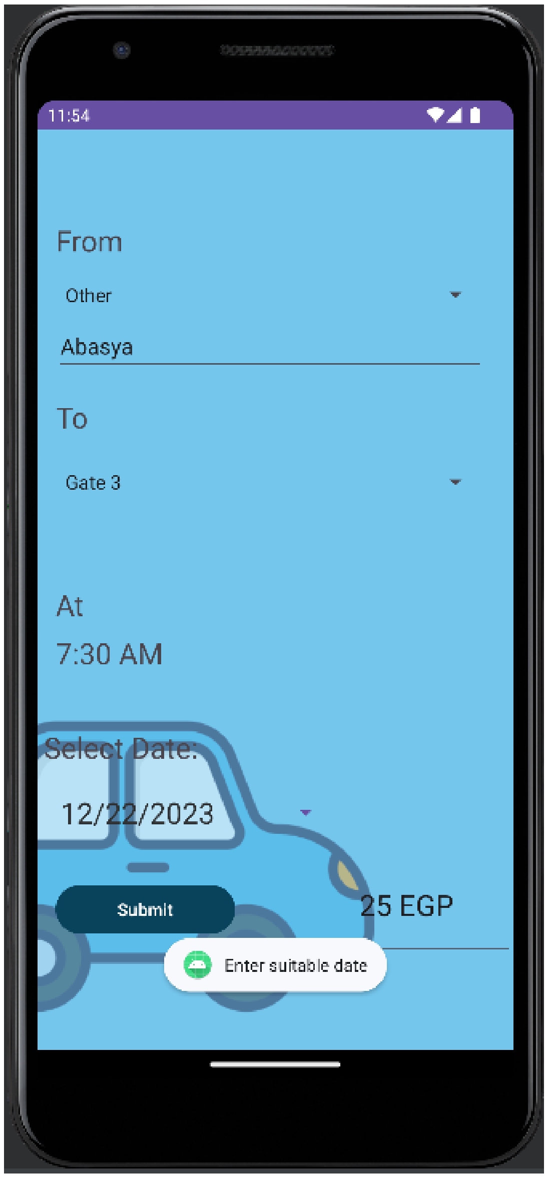
2- Trying to reserve a morning ride after the deadline.

Input: the driver tries to reserve a ride for next day morning after the deadline, this ride will be cancelled as the driver cannot confirm it now

Output: An error message is displayed saying "Enter suitable date"

And same happens for all driver afternoon reservation deadline (should be before 4:30 PM for afternoon ride)





From

Other

Abasya

To

Gate 3

At

7:30 AM

Select Date:

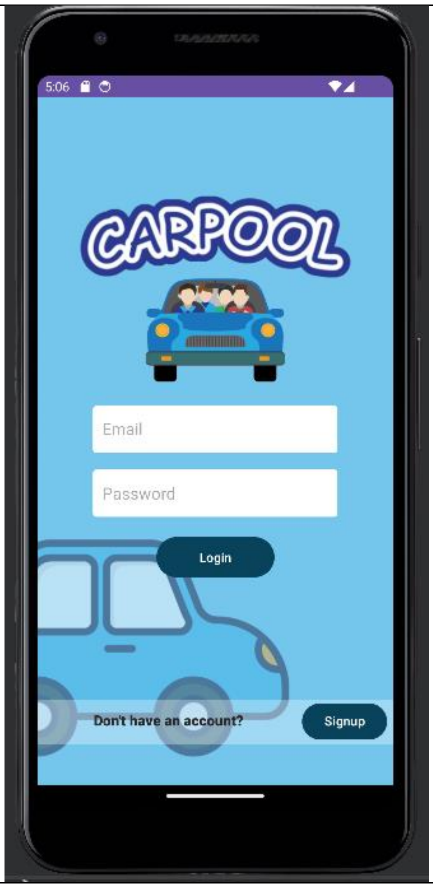
12/22/2023

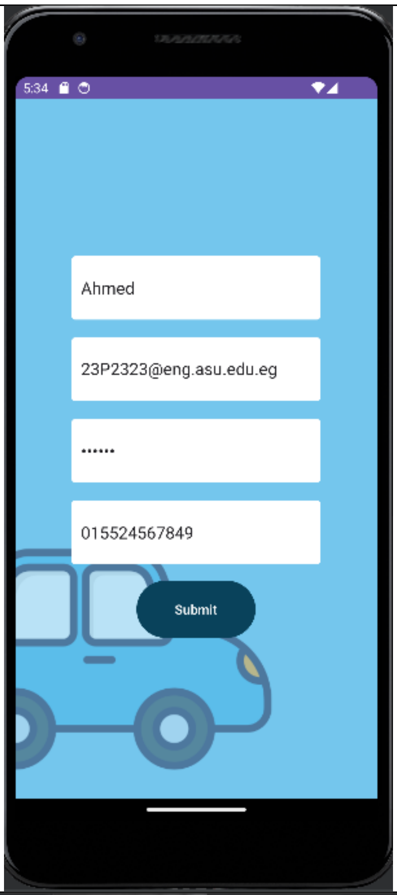
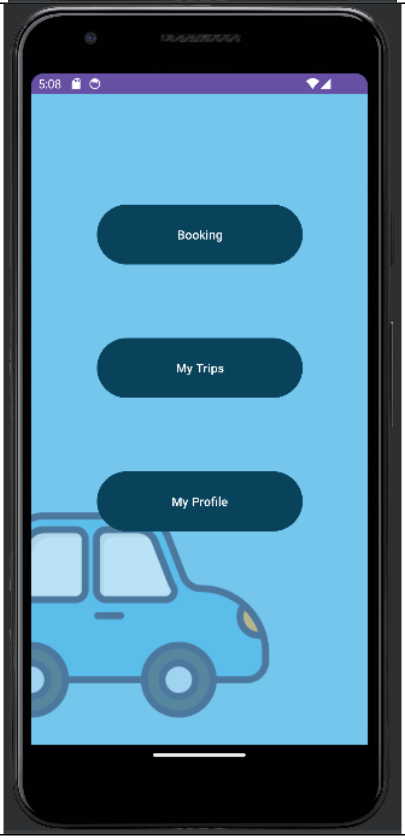
Submit

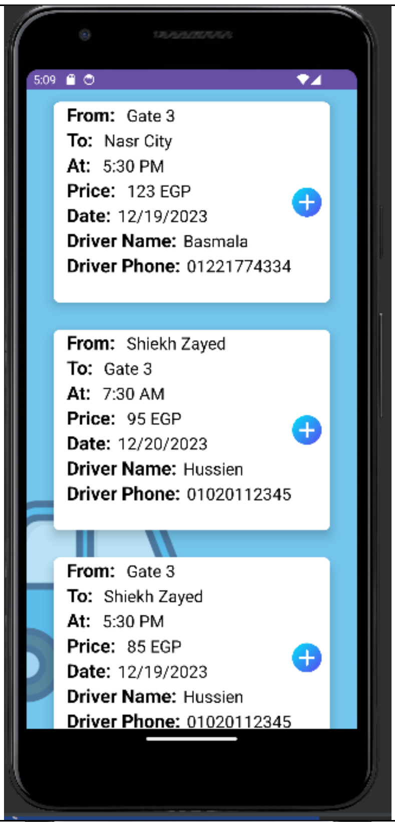
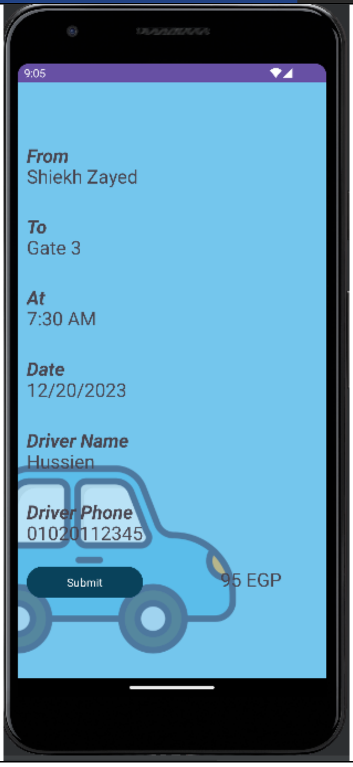
25 EGP

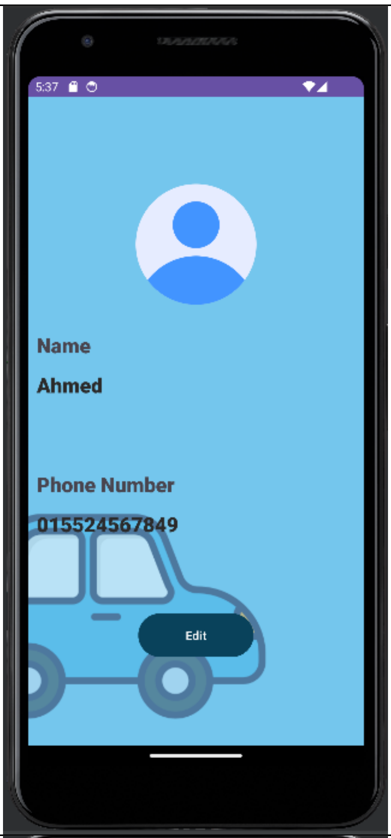
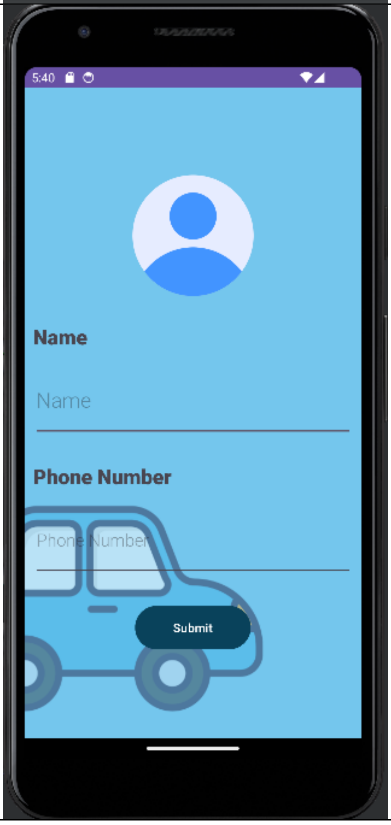
Enter suitable date

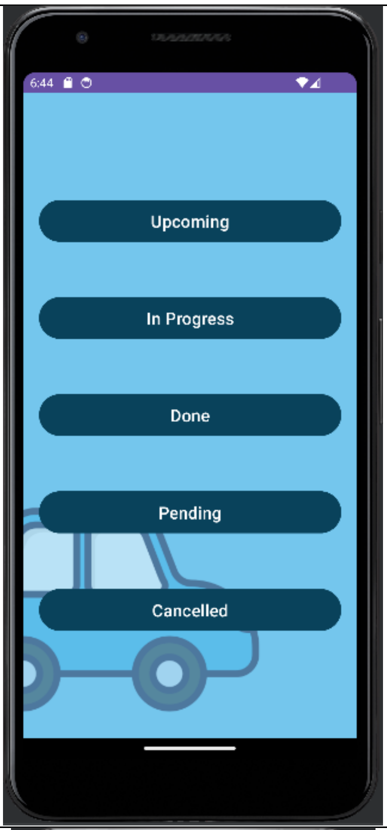
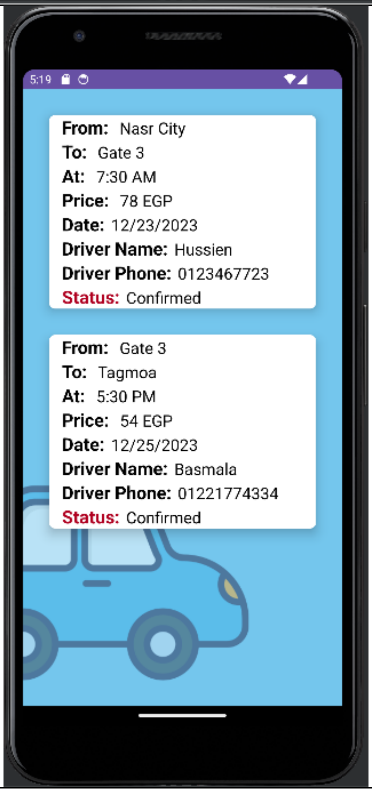
UI Design:

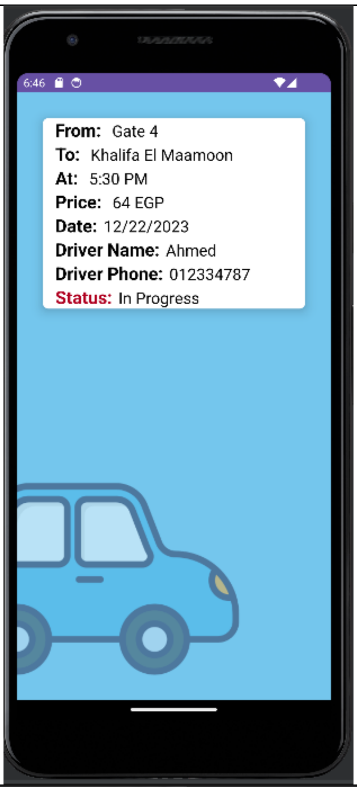
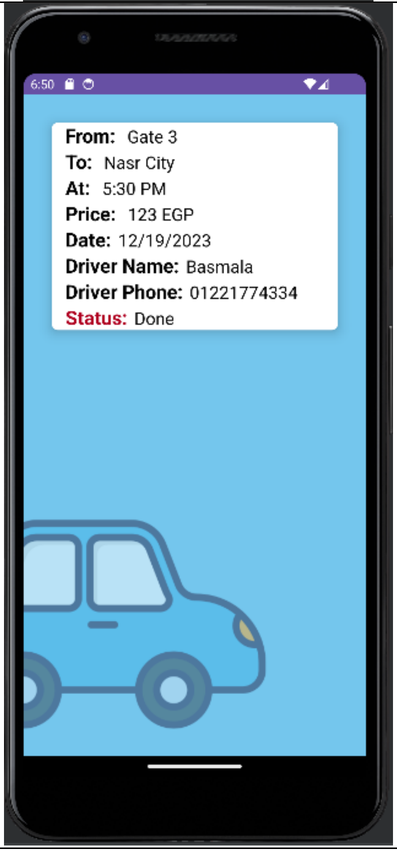
Rider Login Page:	

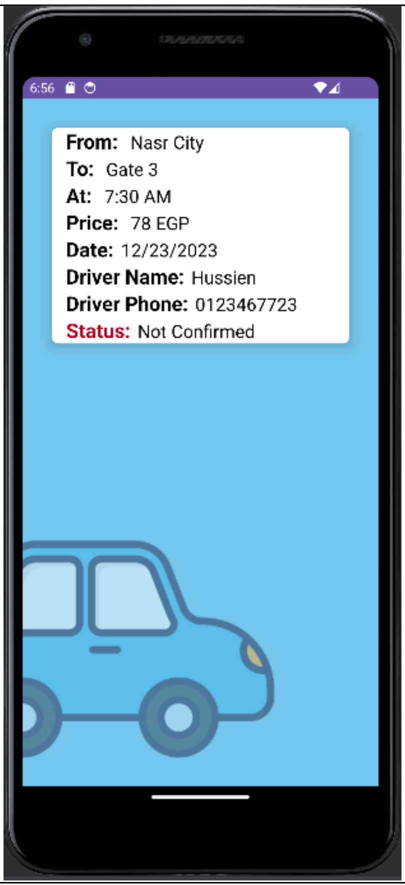
Signup Page:	 5:34 Ahmed 23P2323@eng.asu.edu.eg •••••• 015524567849
Dashboard	 5:08

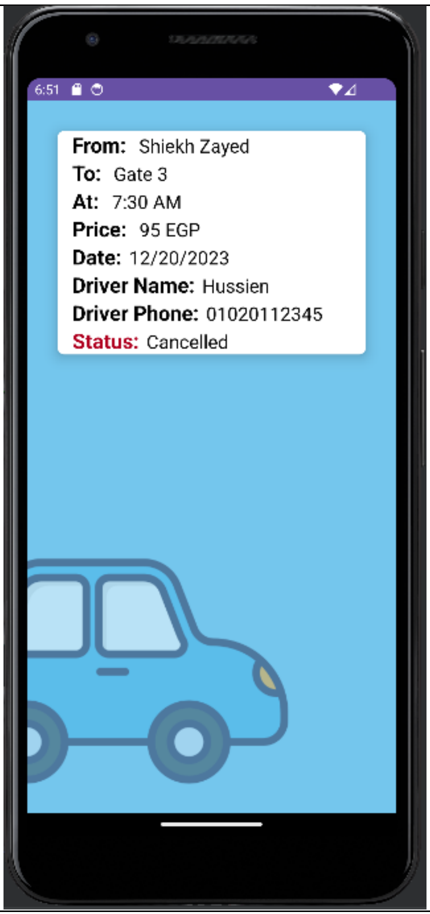
Rider Booking Page: The rider can view all the rides and choose from them by clicking on the (+) beside the required ride to be navigated to the cart page.	 5:09 **From:** Gate 3 **To:** Nasr City **At:** 5:30 PM **Price:** 123 EGP **Date:** 12/19/2023 **Driver Name:** Basmala **Driver Phone:** 01221774334 **From:** Shiekh Zayed **To:** Gate 3 **At:** 7:30 AM **Price:** 95 EGP **Date:** 12/20/2023 **Driver Name:** Hussien **Driver Phone:** 01020112345 **From:** Gate 3 **To:** Shiekh Zayed **At:** 5:30 PM **Price:** 85 EGP **Date:** 12/19/2023 **Driver Name:** Hussien **Driver Phone:** 01020112345
Rider Cart Page: This page is for the rider to review all the ride detail before submitting his/her request	 9:05 *From* Shiekh Zayed *To* Gate 3 *At* 7:30 AM *Date* 12/20/2023 *Driver Name* Hussien *Driver Phone* 01020112345 Submit 95 EGP

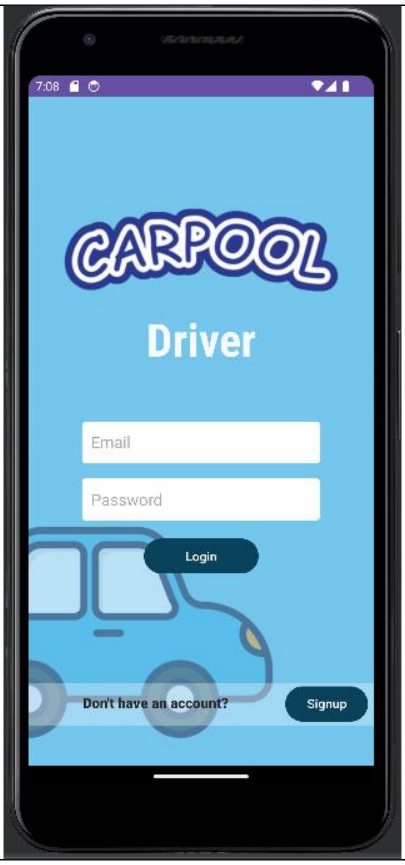
User Profile Page:	 5:37 Name Ahmed Phone Number 015524567849
User Profile Edit Page:	 5:40 Name Name Phone Number Phone Number

Rider Trips Navigation Page: After the user chooses "My Trips" from dashboard page, the user is navigated to another page to choose which kind of rides to show.	 6:44 Upcoming In Progress Done Pending Cancelled
Rider Upcoming Trips: Upcoming Trips are trips that are confirmed but not done yet	 **From:** Nasr City **To:** Gate 3 **At:** 7:30 AM **Price:** 78 EGP **Date:** 12/23/2023 **Driver Name:** Hussien **Driver Phone:** 0123467723 **Status:** Confirmed **From:** Gate 3 **To:** Tagmoa **At:** 5:30 PM **Price:** 54 EGP **Date:** 12/25/2023 **Driver Name:** Basmala **Driver Phone:** 01221774334 **Status:** Confirmed

<u>**Rider In progress Trips:**</u> By changing the device date to 12/22/2023 and the time to 6:45 PM This confirmed trip is now converted to In progress trip and shown in the in progress trips page	 6:46 **From:** Gate 4 **To:** Khalifa El Maamoon **At:** 5:30 PM **Price:** 64 EGP **Date:** 12/22/2023 **Driver Name:** Ahmed **Driver Phone:** 012334787 **Status:** In Progress
<u>**Rider Done Trips:**</u> Done trips are trips that were confirmed then its time and date passed so, it is converted to done status.	 6:50 **From:** Gate 3 **To:** Nasr City **At:** 5:30 PM **Price:** 123 EGP **Date:** 12/19/2023 **Driver Name:** Basmala **Driver Phone:** 01221774334 **Status:** Done

Rider Pending Trips: Pending Trips are trips that are booked by the rider but not yet confirmed from the driver. So, its status is “Not Confirmed” and viewed in the pending page.	 6:56 **From:** Nasr City **To:** Gate 3 **At:** 7:30 AM **Price:** 78 EGP **Date:** 12/23/2023 **Driver Name:** Hussien **Driver Phone:** 0123467723 **Status:** Not Confirmed

Rider Cancelled Trips: Cancelled Trips are trips that were not confirmed, and the deadline of confirming passed without confirming them, so it is now cancelled	 **From:** Shiekh Zayed **To:** Gate 3 **At:** 7:30 AM **Price:** 95 EGP **Date:** 12/20/2023 **Driver Name:** Hussien **Driver Phone:** 01020112345 **Status:** Cancelled

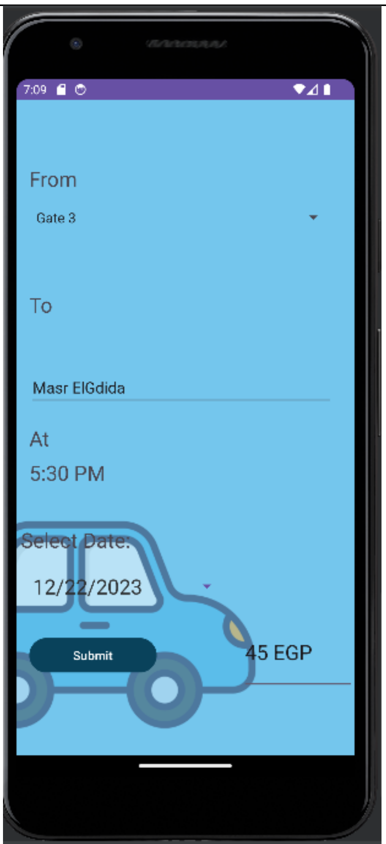
Driver Login Page:	

Driver Booking Page:

The driver can navigate to this page after clicking on “Booking” from dashboard page which common page between the rider and the driver (same design)

The driver chooses the source and destination, chooses whether make the source from Gate 3/ Gate 4 or from other destination.

In case Gates 3 or 4 chosen then he/she has to write the destination in text, otherwise he/she has to write the source in text.


7:09

From

Gate 3

To

Masr ElGdida

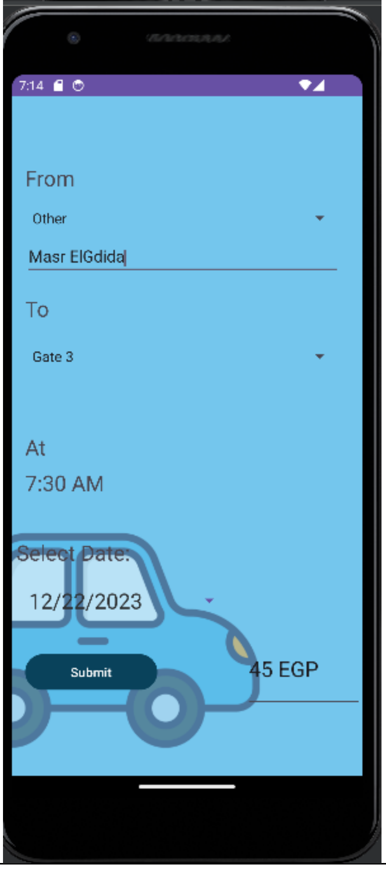
At

5:30 PM

Select Date:

12/22/2023

45 EGP


7:14

From

Other

Masr ElGdida

To

Gate 3

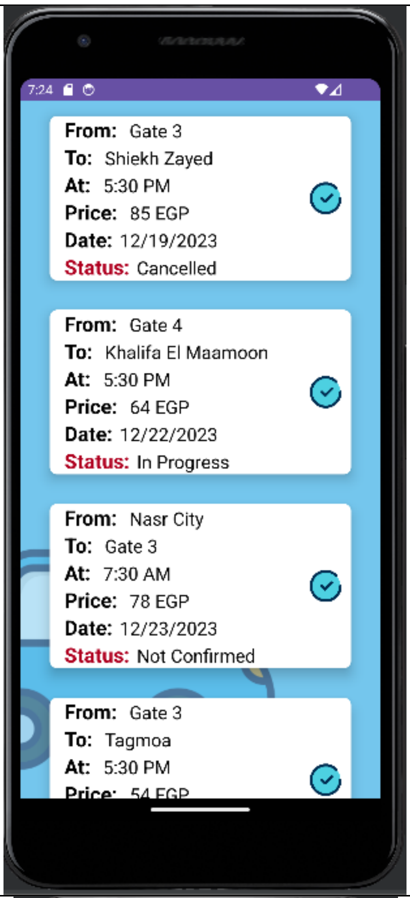
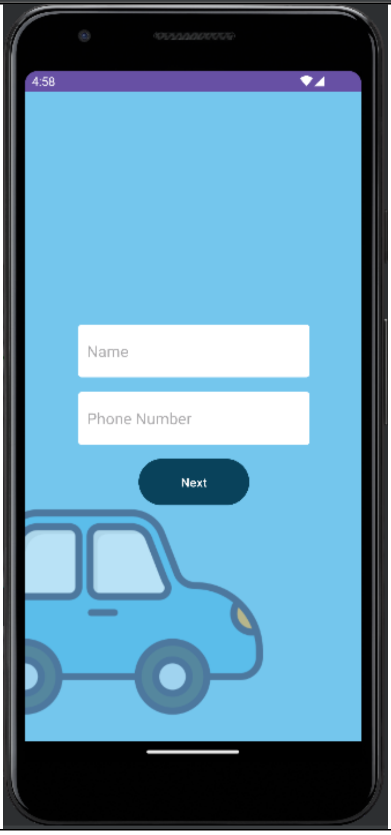
At

7:30 AM

Select Date:

12/22/2023

45 EGP

Driver Trips Page: The driver can navigate to this page after clicking on “My Trips” from dashboard page. The driver can review the status of each of his/her trips. This status is calculated based on time and date of the ride, also based on time and date of current device. The driver click on the (✓) of any ride its status will be converted from Not Confirmed to Confirmed. Not Confirmed is the only state that can be converted to Confirmed. Example: By changing the device date to 12/22/2023 and the time to 6:45 PM This confirmed trip is now converted to In progress trip.	 7:24 **From:** Gate 3 **To:** Shiekh Zayed **At:** 5:30 PM **Price:** 85 EGP **Date:** 12/19/2023 **Status:** Cancelled **From:** Gate 4 **To:** Khalifa El Maamoon **At:** 5:30 PM **Price:** 64 EGP **Date:** 12/22/2023 **Status:** In Progress **From:** Nasr City **To:** Gate 3 **At:** 7:30 AM **Price:** 78 EGP **Date:** 12/23/2023 **Status:** Not Confirmed **From:** Gate 3 **To:** Tagmoa **At:** 5:30 PM **Price:** 54 EGP
Pre driver booking page: This page is shown before the driver navigates to the booking page where he/she can reserve this ride. This page requires entering the name and phone number of the driver in order to be associated with the ride details and send to the rider and saved in a consistent manner with the asynchronous data base call that save the ride details in order to be accessed by the rider from the rider app.	

Database Structure:

1) RoomDB Local DB for profile data:

User profile data can be saved using the architecture shown in below image, incorporating RoomDB.

1. UI Controller Initiates the Save:

- The process begins when the user interacts with the UI to initiate a profile save action (e.g., When user signup into the account).
- The UI controller, typically an activity or fragment, calls a method on the ViewModel to trigger the save process.

2. ViewModel Interacts with Repository:

- The ViewModel, acting as the mediator between the UI and data layers, communicates with the Repository to execute the save operation.
- It passes the profile data to be saved to the Repository, which handles the interaction with the database.

3. Repository Interacts with RoomDB:

- The Repository, responsible for managing data access, calls the appropriate methods on the RoomDB DAO (Data Access Object) to insert or update the profile data.
- RoomDB handles the database interactions, including creating or updating the necessary SQL queries.

4. RoomDB Persists Data:

- RoomDB efficiently executes the SQL operations to store the profile data in the SQLite database.
- It handles any potential conflicts or errors that may arise during the database operation.

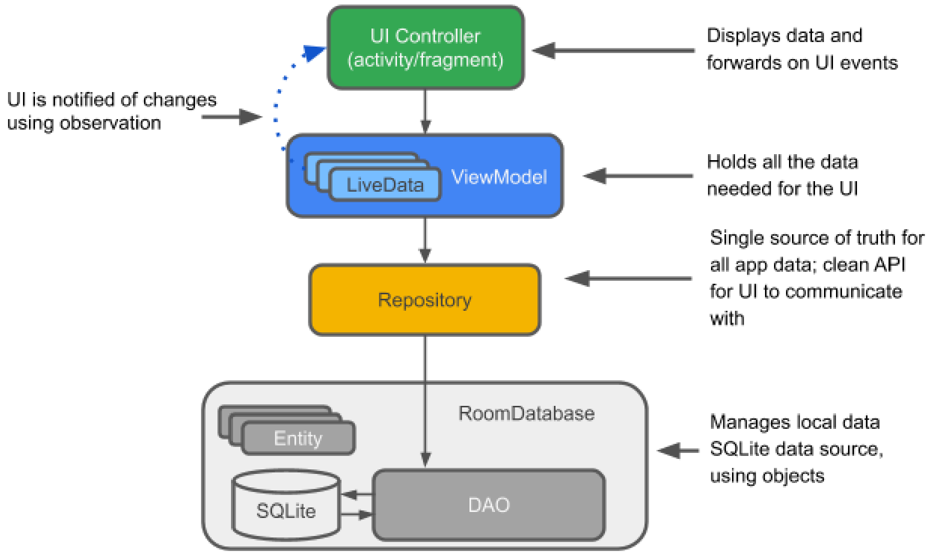
5. Repository Updates LiveData:

- Upon successful completion of the database operation, the Repository notifies the ViewModel by updating the relevant LiveData object(s).
- This triggers a data change notification, signaling the UI to refresh its content.

6. UI Reflects Changes:

- The UI, observing the LiveData, automatically receives the change notification and updates its views accordingly to reflect the saved profile data.

This ensures that the user sees the latest, persisted information on the screen.


UI Controller (activity/fragment)
Displays data and forwards on UI events
UI is notified of changes using observation
LiveData
ViewModel
Holds all the data needed for the UI
Repository
Single source of truth for all app data; clean API for UI to communicate with
RoomDatabase
Entity
SQLite
DAO
Manages local data SQLite data source, using objects

2) Firebase Structure to manage driver and rider requests and rides:

My initial approach is directly using Firebase functions within the code, the architecture is enhanced more to align more closely with the MVP pattern, as follows:

1. Model Layer:

Encapsulation of Data Access Logic: The Firebase class now acts as the Model, containing all API interactions with the Firebase database. This centralizes data-related operations and isolates them from other layers.

Clear Interface for Data Exchange: The Firebase class provides well-defined APIs that accept necessary inputs and return the required outputs from the database. This streamlines communication and simplifies data retrieval and manipulation.

2. Presenter Layer:

Mediation between View and Model: Presenter layer manages communication between the UI and the Firebase class. This Presenter handles actions initiated by the View, retrieves data from the Model, and updates the View accordingly.

3. View Layer:

Separation of UI Concerns: UI code remains in XML files, focusing solely on visual presentation and user interactions. It doesn't directly interact with Firebase, ensuring a clean separation of concerns.

Therefore, by encapsulating Firebase interactions within a dedicated class and maintaining a separation of concerns, we've structured the system for managing driver-rider requests and rides in a way that closely resembles the MVP architecture, leading to a more organized, testable, and maintainable codebase.

Code:

Profile DAO:

```java
package com.example.carpool.LocalDB;

import androidx.lifecycle.LiveData;
import androidx.room.Dao;
import androidx.room.Delete;
import androidx.room.Insert;
import androidx.room.OnConflictStrategy;
import androidx.room.Query;
import androidx.room.Update;

import java.util.List;


@Dao
public interface ProfileDao {

    // allowing the insert of the same word multiple times by passing a
    // conflict resolution strategy
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    public void insert(ProfileDataEntity ProfileData);

    @Update(onConflict = OnConflictStrategy.IGNORE)
    public void updateProfile(ProfileDataEntity NewProfileData);

    @Delete
    public void deleteProfile(ProfileDataEntity ProfileData);

    @Query("SELECT * FROM profile_data_table WHERE user_email==:userEmail
LIMIT 1")
    public LiveData<ProfileDataEntity> findByEmail(String userEmail);

    @Query("DELETE FROM profile_data_table")
    public void deleteAll();
    @Query("SELECT * FROM profile_data_table ORDER BY user_name ASC")
    public LiveData<List<ProfileDataEntity>> getAlphabetizedUsers();
}
```

ProfielDataEntity:

```java
package com.example.carpool.LocalDB;

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

@Entity(tableName = "profile_data_table")
public class ProfileDataEntity{

    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "user_id")
    int id;
    @NonNull
    @ColumnInfo(name = "user_email")
    private String email;

    @NonNull
    @ColumnInfo(name = "user_name")
    private String name;
    @NonNull
    @ColumnInfo(name = "user_phone_number")
    private String phoneNumber;
    //..other fields, getters, setters
    public ProfileDataEntity(@NonNull String name, @NonNull String
phoneNumber, String email) {
        this.name = name;
        this.phoneNumber = phoneNumber;
        this.email = email;
    }
    //Getters

    @NonNull
    public String getEmail() {
        return email;
    }

    @NonNull
    public String getName() {
        return name;
    }

    @NonNull
    public String getPhoneNumber() {
        return phoneNumber;
    }

    //Setters
    public void setUserName(String name){
        this.name = name;
    }
    public void setUserPhoneNumber(String phoneNumber){
        this.phoneNumber = phoneNumber;
    }

}
```

ProfileRepository:

```
package com.example.carpool.LocalDB;

import android.app.Application;

import androidx.lifecycle.LiveData;

import java.util.List;
```

```
class ProfileRepository {

    private ProfileDao mProfileDao;
    private LiveData<List<ProfileDataEntity>> mAllUsersProfileData;


    // Note that in order to unit test the WordRepository, you have to remove
the Application
    // dependency. This adds complexity and much more code, and this sample
is not about testing.
    // See the BasicSample in the android-architecture-components repository
at
    // https://github.com/googlesamples
    ProfileRepository(Application application) {
        ProfileRoomDatabase db =
ProfileRoomDatabase.getDatabase(application);
        mProfileDao = db.profileDao();
        mAllUsersProfileData = mProfileDao.getAlphabetizedUsers();
    }

    // Room executes all queries on a separate thread.
    // Observed LiveData will notify the observer when the data has changed.
    LiveData<List<ProfileDataEntity>> getAllUsersProfileData() {
        return mAllUsersProfileData;
    }



    // You must call this on a non-UI thread or your app will throw an
exception. Room ensures
    // that you're not doing any long running operations on the main thread,
blocking the UI.
    void insert(ProfileDataEntity profileData) {
        ProfileRoomDatabase.databaseWriteExecutor.execute(() -> {
            mProfileDao.insert(profileData);
        });
    }

    void deleteProfile(ProfileDataEntity profileData){
        ProfileRoomDatabase.databaseWriteExecutor.execute(() -> {
            mProfileDao.deleteProfile(profileData);
        });
    }


}
```

ProfileRoomDatabase:

```
package com.example.carpool.LocalDB;

import androidx.room.Database;
import androidx.room.Room;
import androidx.room.RoomDatabase;
import androidx.sqlite.db.SupportSQLiteDatabase;
```

```java
import android.content.Context;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import androidx.annotation.NonNull;


@Database(entities = {ProfileDataEntity.class}, version = 1, exportSchema =
false)
public abstract class ProfileRoomDatabase extends RoomDatabase {
    public abstract ProfileDao profileDao();
    private static volatile ProfileRoomDatabase INSTANCE;
    private static final int NUMBER_OF_THREADS = 4;
    static final ExecutorService databaseWriteExecutor =
            Executors.newFixedThreadPool(NUMBER_OF_THREADS);

    static ProfileRoomDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (ProfileRoomDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE =
Room.databaseBuilder(context.getApplicationContext(),
                                    ProfileRoomDatabase.class,
"user_profile_database")
                            .addCallback(sRoomDatabaseCallback).build();
                }
            }
        }
        return INSTANCE;
    }

    private static RoomDatabase.Callback sRoomDatabaseCallback = new
RoomDatabase.Callback() {
        @Override
        public void onCreate(@NonNull SupportSQLiteDatabase db) {
            super.onCreate(db);

            // If you want to keep data through app restarts,
            // comment out the following block
            databaseWriteExecutor.execute(() -> {
                // Populate the database in the background.
                // If you want to start with more words, just add them.
                ProfileDao dao = INSTANCE.profileDao();
                dao.deleteAll();



            });
        }
    };

}
```

ProfileViewModel:

```
package com.example.carpool.LocalDB;

import android.app.Application;

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

import java.util.List;

public class ProfileViewModel extends AndroidViewModel {

    private ProfileRepository mRepository;

    private final LiveData<List<ProfileDataEntity>> mAllUsersProfileData;

    public ProfileViewModel (Application application) {
        super(application);
        mRepository = new ProfileRepository(application);
        mAllUsersProfileData = mRepository.getAllUsersProfileData();
    }

    public LiveData<List<ProfileDataEntity>> getAllUsersProfileData() {
return mAllUsersProfileData; }

    public void insert(ProfileDataEntity profileData) {
mRepository.insert(profileData); }
    public void deleteProfile(ProfileDataEntity profileData) {
mRepository.deleteProfile(profileData); }
}
```

Firebase Class:

```
package com.example.carpool.MyClasses;

import android.content.Context;
import android.util.Log;
import android.widget.Toast;

import com.example.carpool.AdapterRequest;
import com.example.carpool.AdapterRoute;
import com.example.carpool.Route;
import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
```

```java
import java.util.Locale;
import java.util.Map;

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

public class Firebase{
    DatabaseReference databaseReference;
    DatabaseReference requestsDatabaseReference;
    FirebaseDatabase firebaseDatabase = FirebaseDatabase.getInstance();
    AdapterRoute adapterRoute;
    AdapterRequest adapterRequest;
    RecyclerView recyclerView;

    List<Route> AvailableRoutsList(){
        List list = new ArrayList<Route>();
        databaseReference =
FirebaseDatabase.getInstance().getReference("Rides");
        databaseReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                for(DataSnapshot dataSnapshot: snapshot.getChildren()){
                    Route route = dataSnapshot.getValue(Route.class);
                    list.add(route);
                }
                adapterRoute.notifyDataSetChanged();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {

            }
        });
        return list;
    }
    Context context = null;
    void makeRideBooking(String sanitizedPath, String rideID){


databaseReference.child("Requests").child(sanitizedPath).child(rideID).setVal
ue(rideID).addOnSuccessListener(new OnSuccessListener<Void>() {
            @Override
            public void onSuccess(Void unused) {
                Toast.makeText(context, "Request is done successfully",
Toast.LENGTH_SHORT).show();
            }
        }).addOnFailureListener(new OnFailureListener() {
            @Override
            public void onFailure(@NonNull Exception e) {
                Log.e("FirebaseError", "Error writing to Firebase", e);
                Toast.makeText(context, "Failed to request ride. Check logs
for details.", Toast.LENGTH_SHORT).show();
            }
        });

    }
    List<Route> classifyRideType(Route route, String tripType, List<Route>
requests){
        List reqList = new ArrayList<Route>();
        DatabaseReference ridesDatabaseReference=
FirebaseDatabase.getInstance().getReference("Rides");;
        ridesDatabaseReference.addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                // Handle rides data
                for (DataSnapshot ridesSnapshot : snapshot.getChildren()) {
                    Route route = ridesSnapshot.getValue(Route.class);
                    String newStatus =
getRouteNewStatus(route.Status,route.Date,route.Time,route);
                    route.Status= newStatus;
                    if(tripType.equals("Upcoming")){
                        if (route != null && route.Ride_ID != null &&
requests.contains(route.Ride_ID) && ((route.Status).equals("Confirmed"))) {
                            reqList.add(route);
                        }
                    }else if(tripType.equals("Pending")){
                        if (route != null && route.Ride_ID != null &&
requests.contains(route.Ride_ID) && ((route.Status).equals("Not Confirmed")))
{

                            reqList.add(route);
                        }
                    }
                    else if(tripType.equals("Done")){
                        if (route != null && route.Ride_ID != null &&
requests.contains(route.Ride_ID) && ((route.Status).equals("Done"))) {
                            reqList.add(route);
                        }
                    }
                    else if(tripType.equals("InProgress")){
                        if (route != null && route.Ride_ID != null &&
requests.contains(route.Ride_ID) && ((route.Status).equals("In Progress"))) {
                            reqList.add(route);
                        }
                    }
                    else if(tripType.equals("Cancelled")){
                        if (route != null && route.Ride_ID != null &&
requests.contains(route.Ride_ID) && ((route.Status).equals("Cancelled"))) {
                            reqList.add(route);
                        }
                    }
                    Log.d("MyTrips", "2Requests data: " +
requests.toString());
                    Log.d("MyTrips", "2Rides data: " + reqList.toString());
                }
                // Update the adapter

                updateAdapter();
            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                // Handle error
            }
```

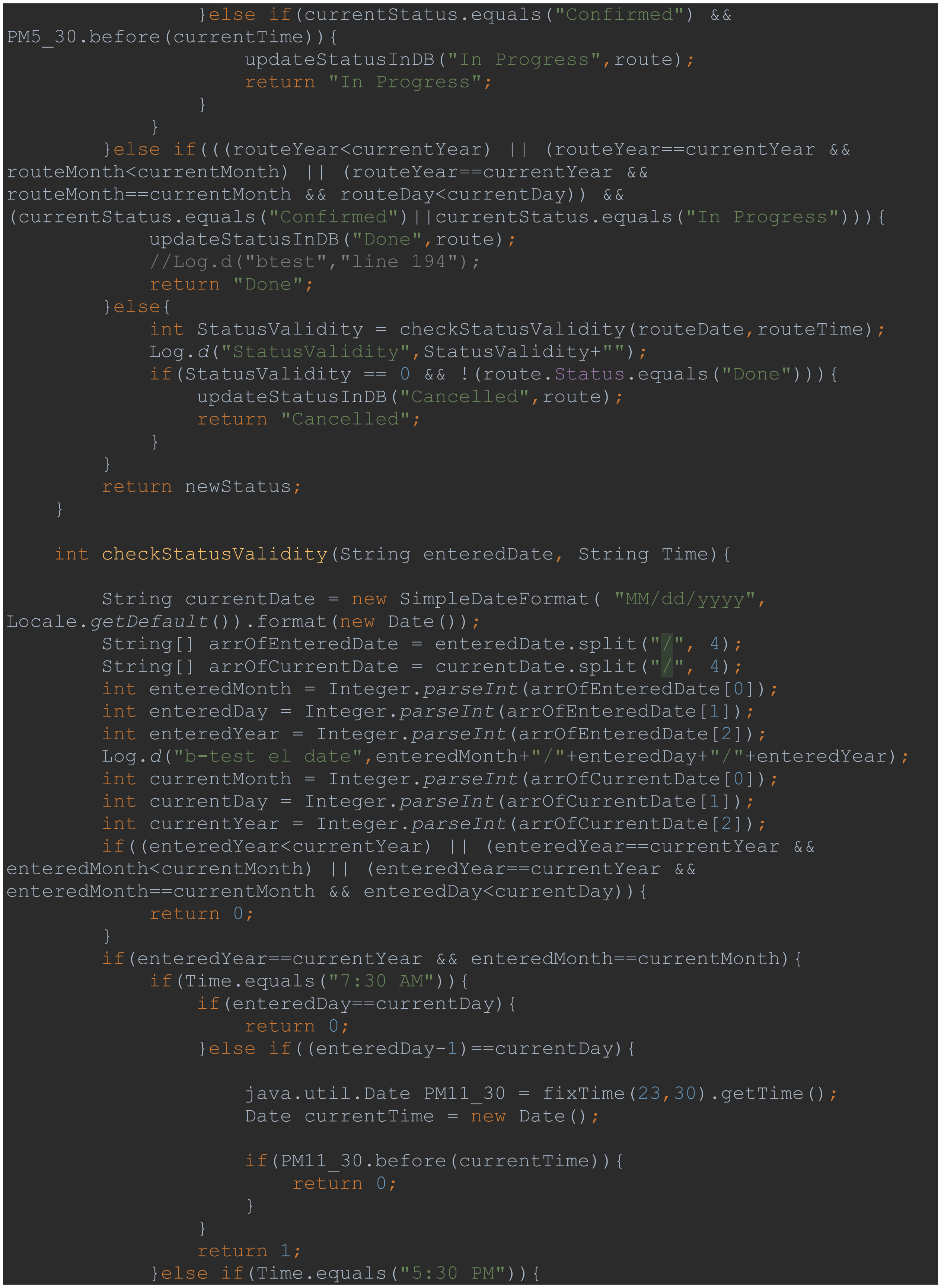
```java
        });
        return reqList;
    }
    void updateStatusInDB(String newStatus, Route route){
        Map<String, Object> updates = new HashMap<>();
        updates.put("Status", newStatus);

firebaseDatabase.getReference("Rides").child(route.Ride_ID).updateChildren(up
dates).addOnCompleteListener(task -> {
            if (task.isSuccessful()) {
                Log.d("Done", "updated");

            }else {
                Log.e("Error", "update failed", task.getException());
            }
        });
    }

    String getRouteNewStatus(String currentStatus, String routeDate, String
routeTime,Route route){
        String newStatus = currentStatus;
        String currentDate = new SimpleDateFormat( "MM/dd/yyyy",
Locale.getDefault()).format(new Date());
        String[] arrOfRouteDate = routeDate.split("/", 4);
        String[] arrOfCurrentDate = currentDate.split("/", 4);
        int routeMonth = Integer.parseInt(arrOfRouteDate[0]);
        int routeDay = Integer.parseInt(arrOfRouteDate[1]);
        int routeYear = Integer.parseInt(arrOfRouteDate[2]);
        int currentMonth = Integer.parseInt(arrOfCurrentDate[0]);
        int currentDay = Integer.parseInt(arrOfCurrentDate[1]);
        int currentYear = Integer.parseInt(arrOfCurrentDate[2]);
        if(currentYear == routeYear && routeDay ==  currentDay && routeMonth
== currentMonth &&
(currentStatus.equals("Confirmed")||currentStatus.equals("In Progress"))){

            Date currentTime = new Date();
            if(routeTime.equals("7:30 AM")){
                java.util.Date AM7_30 = fixTime(7,30).getTime();
                java.util.Date AM9_30 = fixTime(9,30).getTime();
                if(AM9_30.before(currentTime)){
                    Log.d("btest","AM9_30:"+AM9_30+"--
currentTime:"+currentTime+"-
AM9_30.before(currentTime)"+AM9_30.before(currentTime));
                    updateStatusInDB("Done",route);
                    return "Done";
                }else if(currentStatus.equals("Confirmed") &&
AM7_30.before(currentTime)){
                    updateStatusInDB("In Progress",route);
                    return "In Progress";
                }
            }else if(routeTime.equals("5:30 PM")){
                java.util.Date PM5_30 = fixTime(17,30).getTime();
                java.util.Date PM7_30 = fixTime(19,30).getTime();
                if( PM7_30.before(currentTime)){
                    updateStatusInDB("Done",route);
                    //Log.d("btest","line 182");
                    return "Done";
```



```
                }else if(currentStatus.equals("Confirmed") &&
PM5_30.before(currentTime)){
                    updateStatusInDB("In Progress",route);
                    return "In Progress";
                }
            }
        }else if(((routeYear<currentYear) || (routeYear==currentYear &&
routeMonth<currentMonth) || (routeYear==currentYear &&
routeMonth==currentMonth && routeDay<currentDay)) &&
(currentStatus.equals("Confirmed")||currentStatus.equals("In Progress"))){
            updateStatusInDB("Done",route);
            //Log.d("btest","line 194");
            return "Done";
        }else{
            int StatusValidity = checkStatusValidity(routeDate,routeTime);
            Log.d("StatusValidity",StatusValidity+"");
            if(StatusValidity == 0 && !(route.Status.equals("Done"))){
                updateStatusInDB("Cancelled",route);
                return "Cancelled";
            }
        }
        return newStatus;
    }

    int checkStatusValidity(String enteredDate, String Time){

        String currentDate = new SimpleDateFormat( "MM/dd/yyyy",
Locale.getDefault()).format(new Date());
        String[] arrOfEnteredDate = enteredDate.split("/", 4);
        String[] arrOfCurrentDate = currentDate.split("/", 4);
        int enteredMonth = Integer.parseInt(arrOfEnteredDate[0]);
        int enteredDay = Integer.parseInt(arrOfEnteredDate[1]);
        int enteredYear = Integer.parseInt(arrOfEnteredDate[2]);
        Log.d("b-test el date",enteredMonth+"/"+enteredDay+"/"+enteredYear);
        int currentMonth = Integer.parseInt(arrOfCurrentDate[0]);
        int currentDay = Integer.parseInt(arrOfCurrentDate[1]);
        int currentYear = Integer.parseInt(arrOfCurrentDate[2]);
        if((enteredYear<currentYear) || (enteredYear==currentYear &&
enteredMonth<currentMonth) || (enteredYear==currentYear &&
enteredMonth==currentMonth && enteredDay<currentDay)){
            return 0;
        }
        if(enteredYear==currentYear && enteredMonth==currentMonth){
            if(Time.equals("7:30 AM")){
                if(enteredDay==currentDay){
                    return 0;
                }else if((enteredDay-1)==currentDay){

                    java.util.Date PM11_30 = fixTime(23,30).getTime();
                    Date currentTime = new Date();

                    if(PM11_30.before(currentTime)){
                        return 0;
                    }
                }
                return 1;
            }else if(Time.equals("5:30 PM")){
                if(enteredDay==currentDay){
                    Date PM4_30 = fixTime(16,30).getTime();
                    Date currentTime = new Date();
                    if(PM4_30.before(currentTime)){
                        return 0;
                    }
                }
                return 1;
            }
        }
        return -1;
    }
    Calendar fixTime(int hour, int minutes){
        Calendar calendar = Calendar.getInstance();
        calendar.set(Calendar.HOUR_OF_DAY, hour);
        calendar.set(Calendar.MINUTE, minutes);
        calendar.set(Calendar.SECOND, 0);

        // Combine the current date with the specified time
        calendar.set(Calendar.YEAR, calendar.get(Calendar.YEAR));
        calendar.set(Calendar.MONTH, calendar.get(Calendar.MONTH));
        calendar.set(Calendar.DAY_OF_MONTH,
calendar.get(Calendar.DAY_OF_MONTH));
        return calendar;

    }
    private void updateAdapter() {
        adapterRequest = new AdapterRequest();
        adapterRequest.notifyDataSetChanged();
        recyclerView.setAdapter(adapterRequest);
    }

    List<Route> returnUserReq(String sanitizedPath){
        requestsDatabaseReference =
FirebaseDatabase.getInstance().getReference("Requests").child(sanitizedPath);

        List requests = new ArrayList<>();
        requestsDatabaseReference.addListenerForSingleValueEvent(new
ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                // Handle requests data
                for (DataSnapshot requestsSnapshot : snapshot.getChildren())
{
                    String rideID = requestsSnapshot.getValue().toString();
                    requests.add(rideID);
                    Log.d("MyTrips", "1Requests data: " +
requests.toString());
                }

            }

            @Override
            public void onCancelled(@NonNull DatabaseError error) {
                // Handle error
            }
        });
        return requests;
    }



}
```