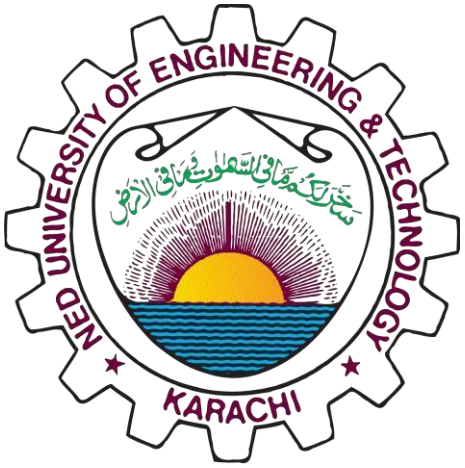
PROGRESS REPORT
DATABASE MANAGEMENT SYSTEMS(CS-222)
BATCH: 2021



NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Cabxury
A Cab Booking Application

G1-GROUP MEMBERS

Muhammad Kashif	**CS-21032**
Muhammad Aamir	**CS-21029**
Syed Maarij	**CS-21022**
Usama Khalid	**CS-21028**

Date of Submission: 16-07-2023

C E R T I F I C A T E

Certified that following students have successfully completed the DBMS project titled ***Cabxury (A cab Booking Android Application)*** assigned as a mandatory requirement for qualifying the practical examination of the course.

Muhammad Kashif	CS - 21032	_____
Muhammad Aamir	CS - 21029	_____
Syed Maarij	CS - 21022	_____
Usama Khalid	CS - 21028	_____

_____

Name & Signature

Project Examiner

ACKNOWLEDGEMENTS

Alhamdulillah, only by the Grace and help of Almighty Allah we've been able to complete this project and all of its functionalities. If it hadn't been for Him we wouldn't have reached this stage. Secondly, special regards to our DBMS course teacher Prof. S. Zafar Qasim also who helped us in the accomplishment of this task and satisfied our queries each time we required his help.

Furthermore, it's been a team work and all the participating members had been working hard and dedicated their time and effort for the completion of this project so we would also like to thank our group members each of whom was a primary individual associated with the consummation of this project.

Table of Contents

Contents

Abstract:

This report aims to present a comprehensive database design for a ride-sharing application. The application serves as a platform that connects users with drivers, allowing them to request rides, make payments, and provide feedback through reviews. In this report, we will delve into the different entities involved in the system, explore their relationships, and analyze the cardinalities that govern these relationships to ensure the effective and efficient functioning of the application's database structure. By understanding these key elements, we can facilitate seamless interactions between users and drivers while maintaining the integrity and reliability of the application's data management system.

Chapter 1:

Introduction:

The project aims to develop an online cab booking application similar to Uber, catering to the transportation needs of users. The advent of technology and the widespread use of smartphones have transformed the traditional taxi-hailing experience into a more streamlined and accessible process.

By creating the database of our application, we ensured the solution of the of the following problems we had been facing before which includes:

1) Data Management and Accessibility
2) Real-time matching and Scheduling
3) Driver Management and Performance tracking
4) Location–Based Services
5) Data Security and privacy

Entity Descriptions:

Considering the real-world application of a ride-sharing platform, the following entities have been chosen based on their relevance and assumptions regarding the system's functionality, user interactions, and data management requirements. These entities play vital roles in facilitating smooth and reliable operations within the application, ensuring a seamless experience for both users and drivers.

User:

The "User" entity represents individuals who engage with the ride-sharing application, playing a pivotal role in requesting rides, making payments, and providing feedback. This entity includes the attributes such as user_id (primary key), name, email, password, and phone number.

Driver:

The "Driver" entity represents individuals who offer transportation services

within the ride-sharing application. Drivers play a vital role in fulfilling ride requests and ensuring users' smooth and reliable experience. The entity is composed of the attributes such as driver_id (primary key), name, email, password, and car model.

Ride:

Individual transportation requests made by users inside the ride-sharing application are represented by the "Ride" entity. It has a number of characteristics that record crucial details about each voyage. The characteristics connected to the "Ride" object include ride_id (primary key), driver_ id (foreign key), user_id (foreign key), ride_distance, pickup_location, drop_location, and ride_fare.

Payment:

The "Payment" entity represents the financial transactions made for rides within the ride-sharing application. It is associated with the "User" entity through a relationship called "make," indicating that a user initiates the payment. The attributes associated with the payment entity are payment_id (primary key) ride_id (foreign key), payment_method, transportation_mode, and payment status.

Driver Rating:

The "Driver Rating" entity represents the feedback and rating provided by users for drivers within the ride-sharing application. It allows users to express their satisfaction or dissatisfaction with the driver's performance and helps other users make informed decisions when choosing drivers. The attributes include rating id (primary key), ride_id (foreign key), user_id(foreign key), driving_id, rating_value, rating_comment, and app_rating.

The "Driver Rating" entity enables users to share their experiences and opinions about drivers, contributing to the overall driver reputation and user confidence within the ride-sharing application. These ratings can help other users make informed decisions when selecting drivers for their rides and can also serve as valuable feedback for drivers to improve their services.

Passenger rating:

The "Passenger Rating" entity represents the feedback and rating provided by drivers for passengers within the ride-sharing application. It allows drivers to express their satisfaction or dissatisfaction with the passenger's behavior and helps other drivers make informed decisions when accepting ride requests.

It includes the same attributes as the driver rating.

The "Passenger Rating" entity enables drivers to share their experiences and opinions about passengers, contributing to the overall passenger reputation and driver confidence within the ride-sharing application.

These ratings can help other drivers make informed decisions when accepting ride requests and can also serve as valuable feedback for passengers to improve their behavior and cooperation during rides.

Relationships:

User makes a ride

The user initiates and engages in the process of making a ride within the ride-sharing application. This relationship signifies that a user is the one who requests and schedules a ride, specifying the pickup location, drop-off location, and other ride preferences. The user's interaction triggers the creation of a new ride record associated with their user account.

Driver performs the ride

Once a ride request is made by a user, the ride-sharing application assigns a driver to fulfill the requested ride. The driver is responsible for performing the ride and providing transportation services to the user.

User makes a payment

After a ride is successfully completed, the user is responsible for making the payment for the provided transportation services. This relationship signifies that the user initiates and completes the financial transaction associated with the ride.

User writes driver rating

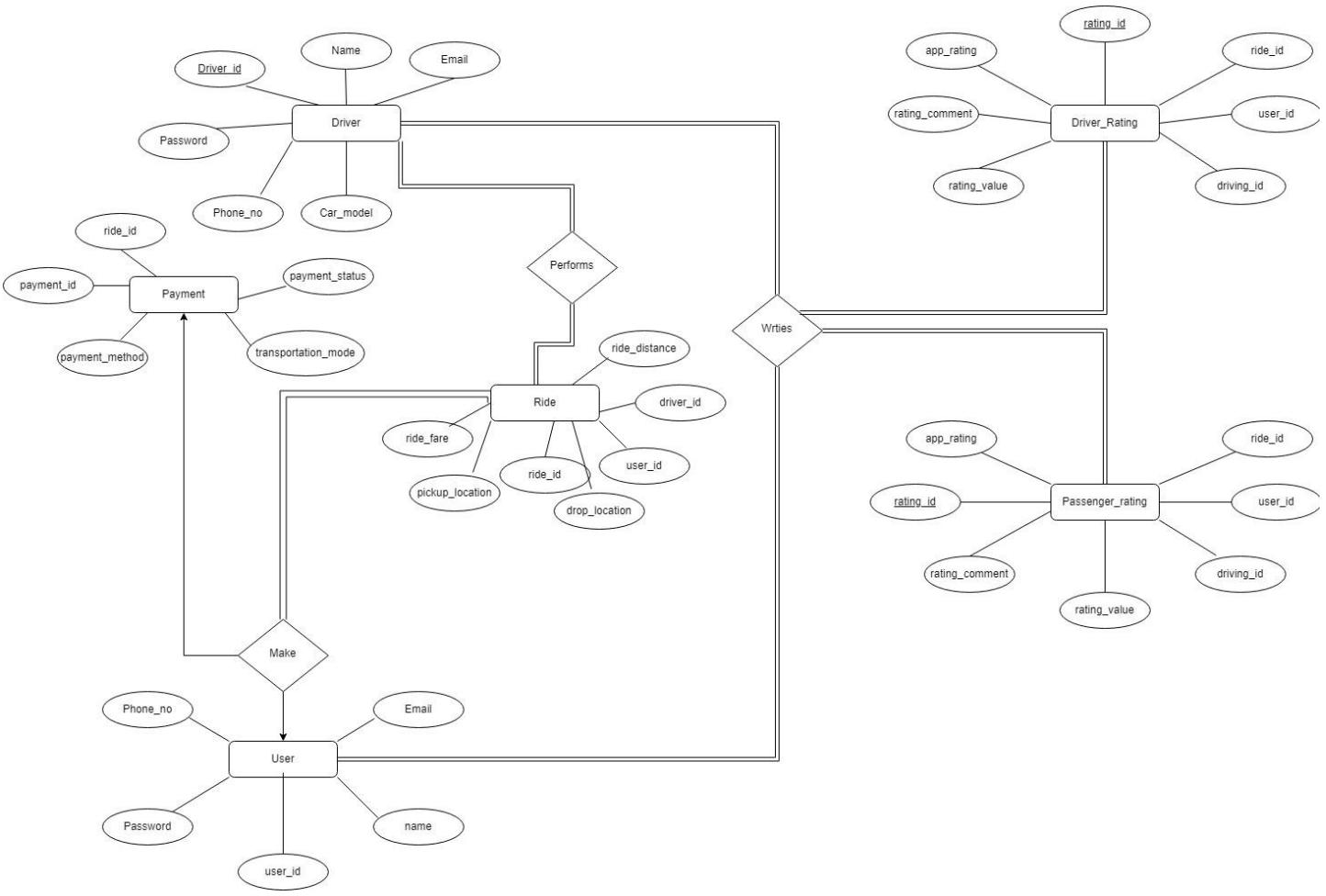
After a ride is completed, the user has the option to provide feedback and rate the performance of the driver who provided the transportation service. This relationship signifies that the user takes on the role of evaluating and providing a rating for the driver's service.

Driver writes user rating

After completing a ride, the driver has the option to provide feedback and rate the behavior and cooperation of the user during the ride. This relationship signifies that the driver assumes the role of evaluating and providing a rating for the user's conduct.

Chapter 2:

ER Model:



Reduction Of ER-Model to Tables:

1) ## R1:

Ride(ride_id,user_id(fk),driver_id(fk),name, email,phone_number,password,car_model,ride_distance,ride_fare,pickup_location,drop_location)

2) ## R2:

Payment(payment_id,ride_id(fk),payment_status,payment_method, transportation_mode)

3) ## R3:

Driver_Rating(rating_id,ride_id(fk),user_id(fk),driver_id(fk),app_rating,rating_comment,rating_value)

4) ## R4:

Passenger_Rating(rating_id,ride_id(fk),user_id(fk),driver_id(fk),app_rating,rating_comment,rating_values)

Chapter 3:

Normalization:

R1: Ride

Ride(ride_id,user_id(fk),driver_id(fk),name, email,phone_number,password,car_model,ride_distance,ride_fare,pickup_location,drop_location)

Anomalies

Update Anomaly:

If we need to update the name, phone or password, we will have to update it everywhere in the table.

Insertion Anomaly:

If we need to insert the information of user/ driver, then we will have to link it with ride which is not possible for all cases.

Deletion Anomaly:

If some user or driver is only connected to one ride id and that ride id gets deleted then the data of user/driver will also be lost.

For 1NF:

ride_id → user_id, driver_id, ride_distance, ride_fare, pickup_location, drop_location

For 2NF:

According to 2nd Normal Form

- The tables should be in 1st Normal Form
- It should not have any partial dependencies

R2: user_id → name, email, phone_number, password

R3: driver_id → name, email, phone_number, password, car_model

For 3NF:

In this case, no non-key attributes are functionally dependent on other non-key attributes. All attributes are functionally dependent on the primary key, ride_id.

For BCNF:

The table is already in 1NF, 2NF, and 3NF. Since there are no partial or transitive dependencies, the table is already in BCNF.

R4: Payment

(payment_id,ride_id(fk),payment_status,payment_method,transportation_mode)

For 1NF:

payment_id → ride_id, payment_status, payment_method, transportation_mode

For 2NF:

There are no partial dependencies since all attributes are functionally dependent on the full primary key.

For 3NF:

In this case, no non-key attributes are functionally dependent on other non-key attributes. All attributes are functionally dependent on the primary key, payment_id.

For BCNF:

The table is already in 1NF, 2NF, and 3NF. Since there are no partial or transitive dependencies, the table is already in BCNF.

R5: Driver_Rating

(rating_id,ride_id(fk),user_id(fk),driver_id(fk),app_rating,rating_comment,rating_value)

For 1NF:

rating_id→ride_id,user_id,driver_id,app_rating,rating_comment, rating_value

For 2NF:

There are no partial dependencies since all attributes are functionally dependent on the full primary key.

For 3NF:

In this case, no non-key attributes are functionally dependent on other non-key attributes. All attributes are functionally dependent on the primary key, rating_id.

For BCNF:

The table is already in 1NF, 2NF, and 3NF. Since there are no partial or transitive dependencies, the table is already in BCNF.

R6: Passenger_Rating

(rating_id,ride_id(fk),user_id(fk),driver_id
(fk),app_rating,rating_comment,rating_value)

For 1NF:

rating_id→ride_id,user_id,driver_id,
app_rating,rating_comment,rating_value

For 2NF:

There are no partial dependencies since all attributes are functionally dependent on the full primary key.

For 3NF:

In this case, no non-key attributes are functionally dependent on other non-key attributes. All attributes are functionally dependent on the primary key, rating_id.

For BCNF:

The table is already in 1NF, 2NF, and 3NF. Since there are no partial or transitive dependencies, the table is already in BCNF.

Tables From Normalization:

1) R1:

Candidate key= user_id, email, phone_number
User (user_id, name, email, phone_number, password)

2) R2:

Candidate key= driver_id, email, phone_number
Driver (driver_id, name, email, phone_number, password, car_model)

3) R3:

Payment(payment_id,ride_id(fk),payment_status,payment_method,transportation_mode)

4) R4:

Ride(ride_id,user_id(fk),driver_id(fk),ride_distance,ride_fare,pickup_location,drop_location)

5) R5:

Driver_Rating(rating_id,ride_id(fk),user_id(fk),driver_id(fk),app_rating,rating_comment,rating_value)

6) R6:

Passenger_Rating(rating_id,ride_id(fk),user_id(fk),driver_id (fk),app_rating,rating_comment,rating_value)

Chapter 4:

SQL Commands to Create Tables:

1) *R1:*

```
CREATE TABLE User (
    user_id INT(11)
    PRIMARY KEY
    AUTO_INCREMENT,
    name VARCHAR(50)
    NOT NULL, email
    VARCHAR(50) NOT
    NULL UNIQUE,
    phone_number
    VARCHAR(11) NOT
    NULL UNIQUE
    Constraint ch_phone
    CHECK(length('phone_
    number')=11), password
    VARCHAR(255) NOT
    NULL Constraint
    ch_pass CHECK
    (length('password')>=8)
    );
```

2) *R2:*

```
CREATE TABLE Driver (
```

driver_id INT(11) PRIMARY KEY AUTO_INCREMENT, name VARCHAR(50) NOT NULL, email VARCHAR(50) NOT NULL UNIQUE, phone_number VARCHAR(11) NOT NULL UNIQUE Constraint ch_phone CHECK(length('phone_number')=11), password VARCHAR(255) NOT NULL Constraint ch_pass CHECK (length('password')>=8), car_model VARCHAR(50) DEFAULT NULL);

3) R3:

CREATE TABLE Payment (payment_id INT(11) PRIMARY KEY AUTO_INCREMENT, ride_id INT(11) NOT NULL, payment_status VARCHAR(50) DEFAULT 'SUCCESSFUL', payment_method VARCHAR(50) DEFAULT NULL, transportation_mode VARCHAR(50) DEFAULT NULL);

4) R4:

CREATE TABLE Ride (
ride_id INT(11) PRIMARY KEY AUTO_INCREMENT, user_id INT(11) NOT NULL,
driver_id INT(11) NOT NULL,
ride_distance VARCHAR(100) NOT NULL, ride_fare VARCHAR(100) NOT NULL,
pickup_location VARCHAR(100) NOT NULL, drop_location VARCHAR(100) NOT NULL,

5) R5:

```
CREATE TABLE Driver_Rating ( rating_id INT(11) PRIMARY KEY
AUTO_INCREMENT, ride_id INT(11) NOT NULL,
user_id INT(11) NOT NULL, driver_id INT(11) NOT NULL,
app_rating VARCHAR(50) DEFAULT NULL, rating_comment
VARCHAR(200) DEFAULT NULL, rating_value INT(50) DEFAULT NULL,
FOREIGN KEY (ride_id) REFERENCES ride(ride_id), FOREIGN KEY
(user_id) REFERENCES user(user_id), FOREIGN KEY (driver_id)
REFERENCES driver(driver_id));
```

7) R6:

CREATE TABLE Passenger_Rating (
rating_id INT(11) PRIMARY KEY AUTO_INCREMENT, ride_id INT(11) NOT NULL,
user_id INT(11) NOT NULL, driver_id INT(11) NOT NULL,
app_rating init(50) DEFAULT NULL, rating_comment VARCHAR(255) DEFAULT NULL, rating_value VARCHAR(50) DEFAULT NULL,
FOREIGN KEY (ride_id) REFERENCES ride(ride_id), FOREIGN KEY (user_id) REFERENCES user(user_id), FOREIGN KEY (driver_id) REFERENCES driver(driver_id)));

SQL Commands to Create Indices:

1) R1: CREATE INDEX idx_user_primarykey ON User (user_id);

2) R2: CREATE INDEX idx_driver_primarykey ON Driver (driver_id);

3) R3: CREATE INDEX idx_payment_primarykey ON Payment (payment_id);

4) R4: CREATE INDEX idx_ride_primarykey ON Ride (ride_id);

5) R5: CREATE INDEX idx_driver_rating_primarykey ON Driver_Rating (rating_id);

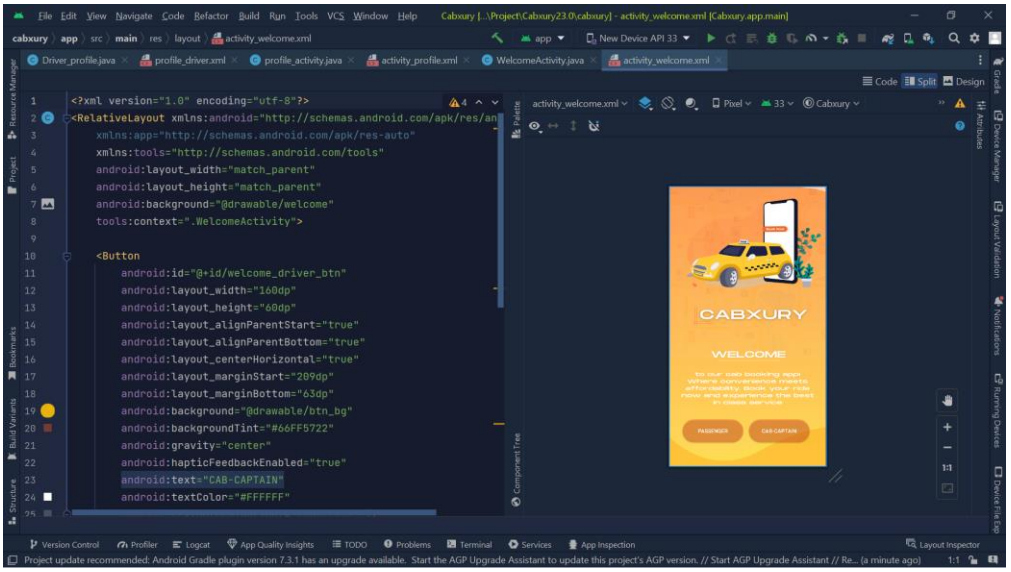
6) R6: CREATE INDEX idx_driver_rating_primarykey ON Driver_Rating (rating);

Chapter 5:

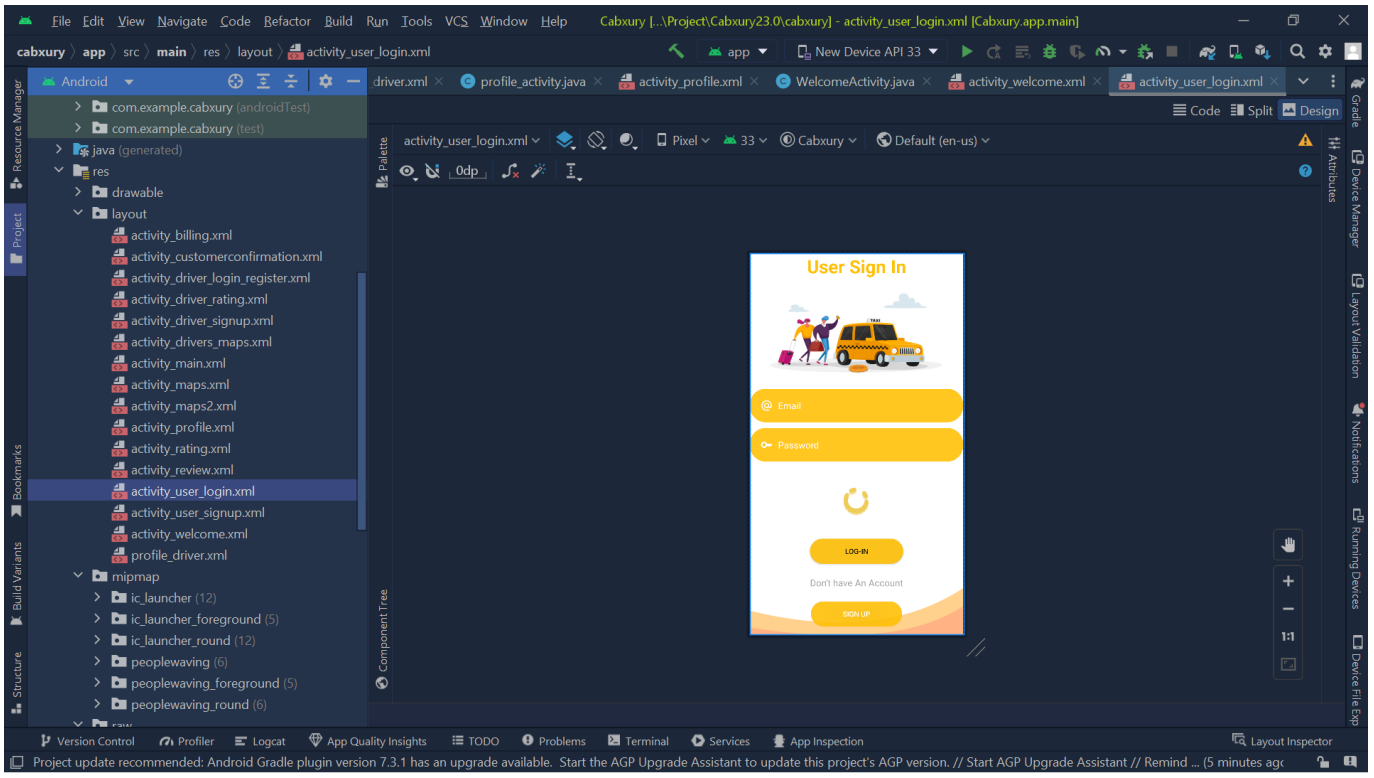
Implementation of Front-End & Back-End

Front-End Implementation:

We used Android Studio and Java programming language for our Front-End development.



We developed 16 different front-end activities in order to provide best user experience and to make our application as user friendly as technically possible.

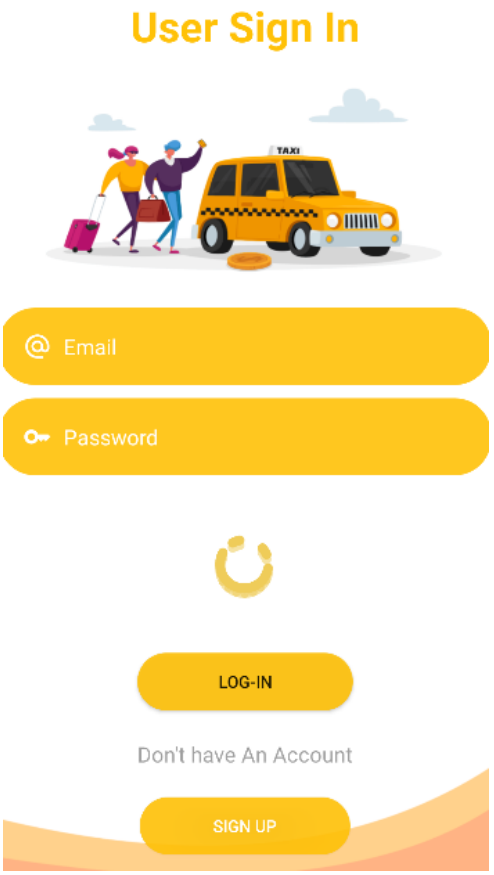


We implemented maps in our application using Google Maps API.

```
        To get one, follow the directions here:

           https://developers.google.com/maps/documentation/android-sdk/get-api-key

        Once you have your API key (it starts with "AIza"), define a new property in your
        project's local.properties file (e.g. MAPS_API_KEY=Aiza...), and replace the
        "YOUR_API_KEY" string in this file with "${MAPS_API_KEY}".
    -->
    <meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="AIzaSyDgg3sM1yQ1v_qA8rLPFp10KUyITi7ua_Y" />

    <activity
        android:name=".MapsActivity2"
        android:exported="false"
        android:label="MapsActivity2" />
```

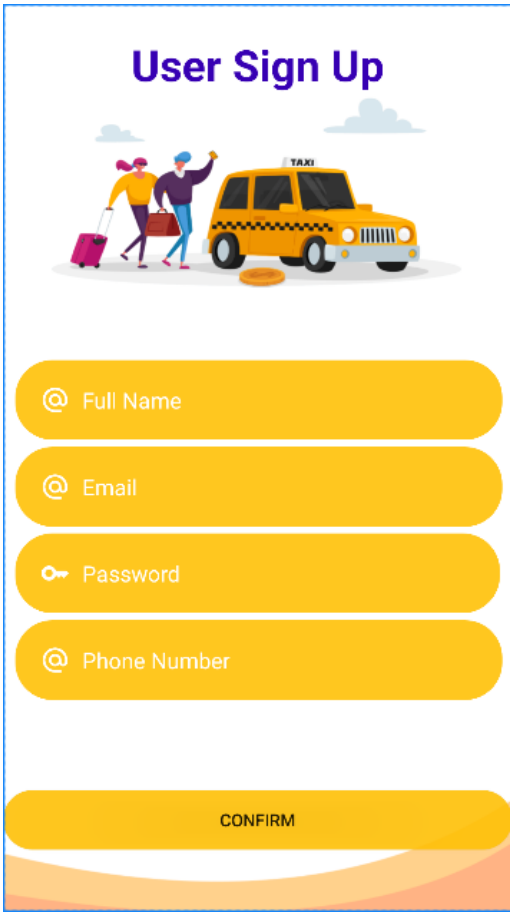
Flow of Application:

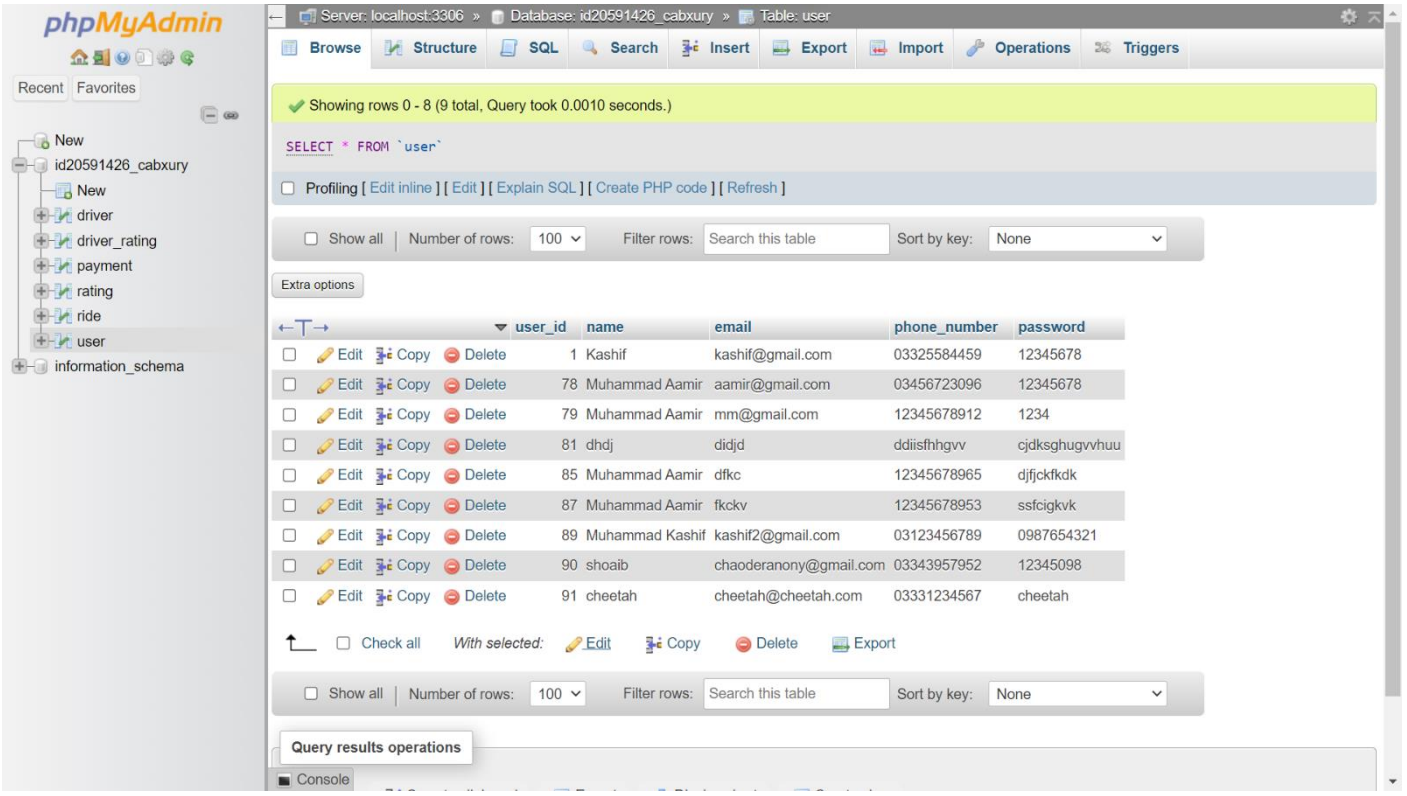
The application starts from the welcome splash activity then it is followed by log in activity where user can login using their existing account or can create a new one right away by clicking on SIGN UP button.



After clicking on SIGN UP button, sign up form opens up

User is supposed to enter the required information in the corresponding fields in order to sign up their account successfully. This information is saved in the SQL database as shown in the picture below.



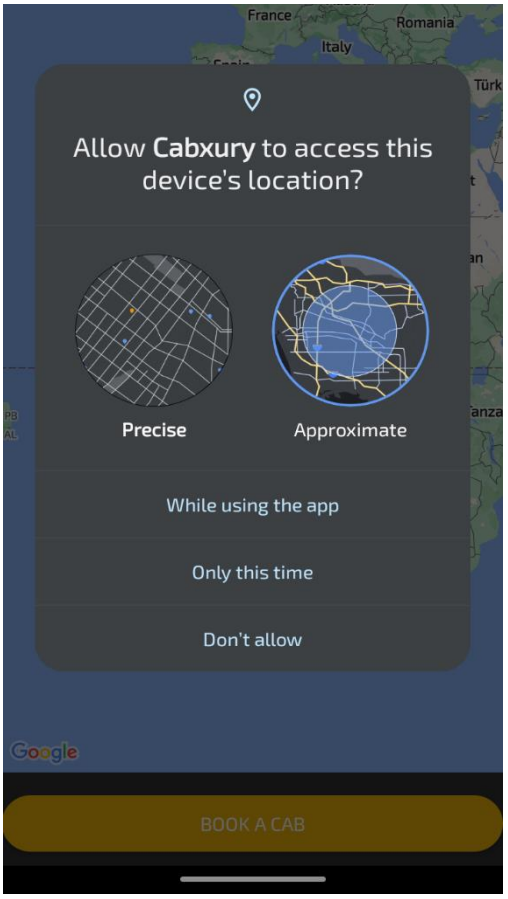


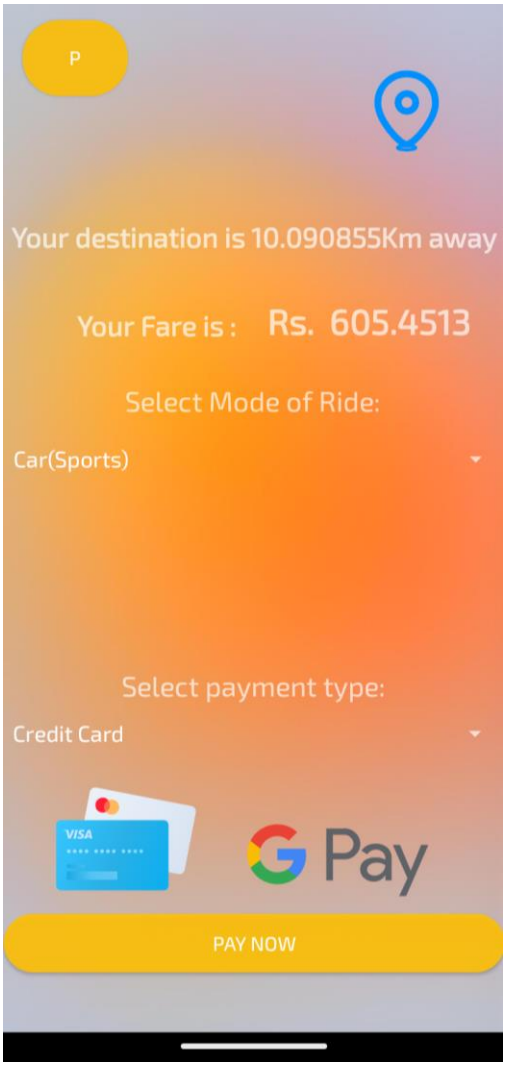
After logging in successfully, the user is transported to Maps activity where they can select their destination and book a cab for their wonderful journey.

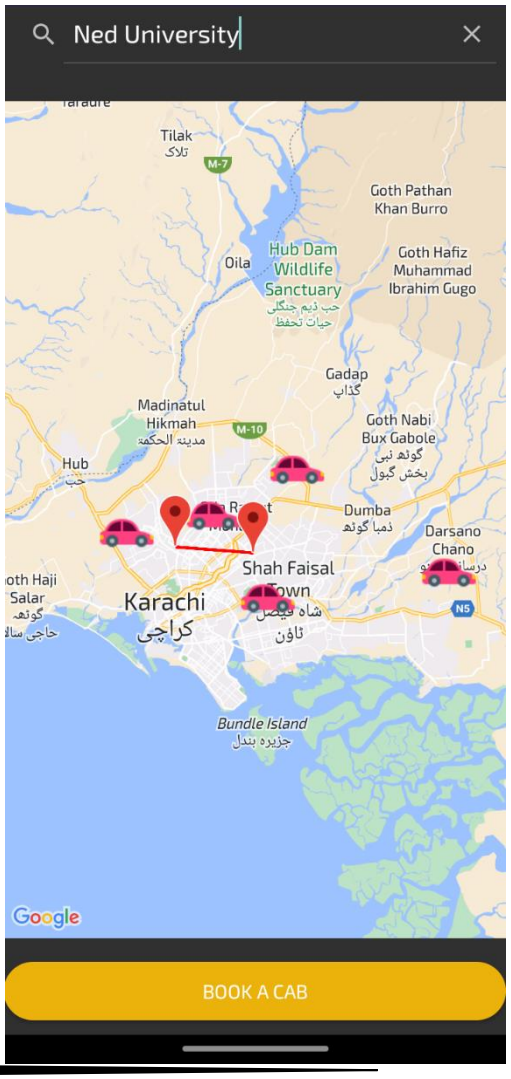
The application asks the user for Location access when initialized for the first time then locates user's current location on the map (when location is on)
automatically.

The user can search his/ her destination in the search bar on the top and application will mark the entered location on the map. The car icons show the current positions of the available driver which are currently simulated since right now this application is still a project and not launched online for general public use.

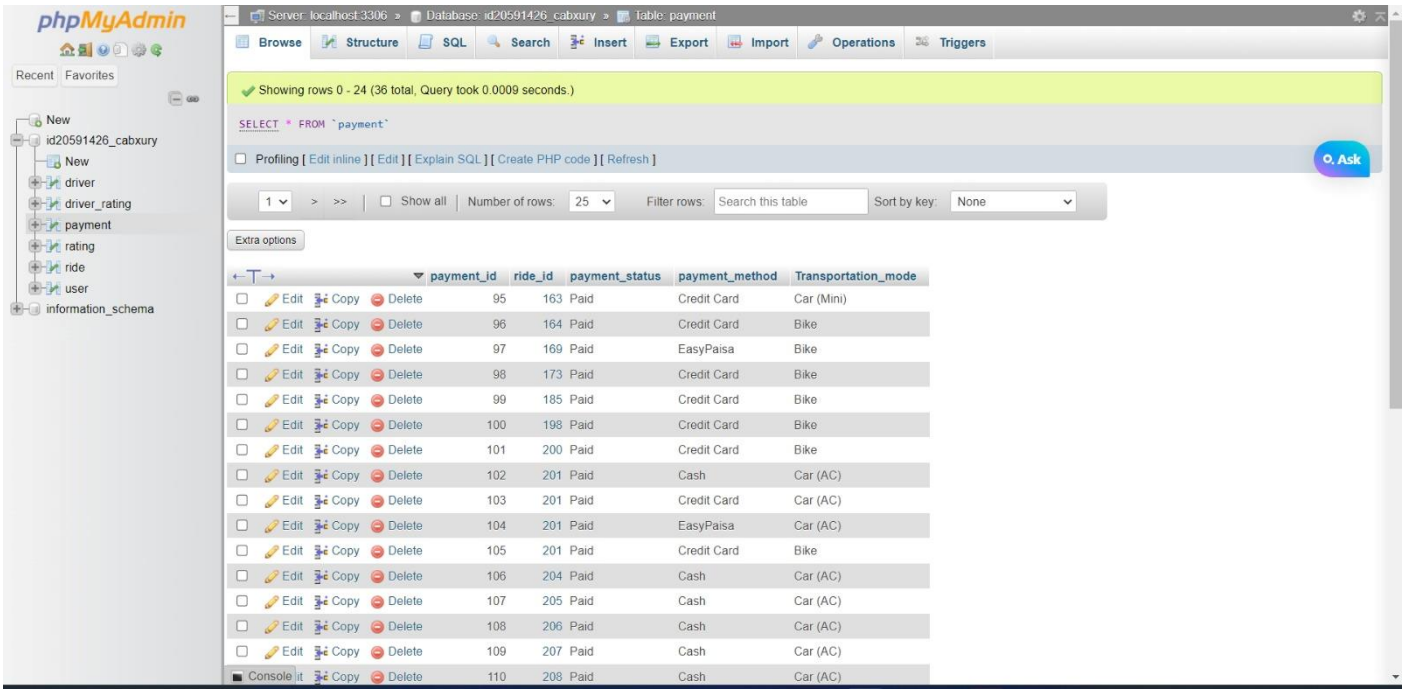
Once the user clicks on "Book a Cab" button, he gets a cab booked for himself. The user is then transported to "Billing Activity" where he can see the information about his ride (distance in KM and fare)
Along with the option to select from multiple methods of payments (including Credit Card, Easypaisa and Cash) and can select from multiple categories of cars (sports, Royal, Mini and Mini AC).



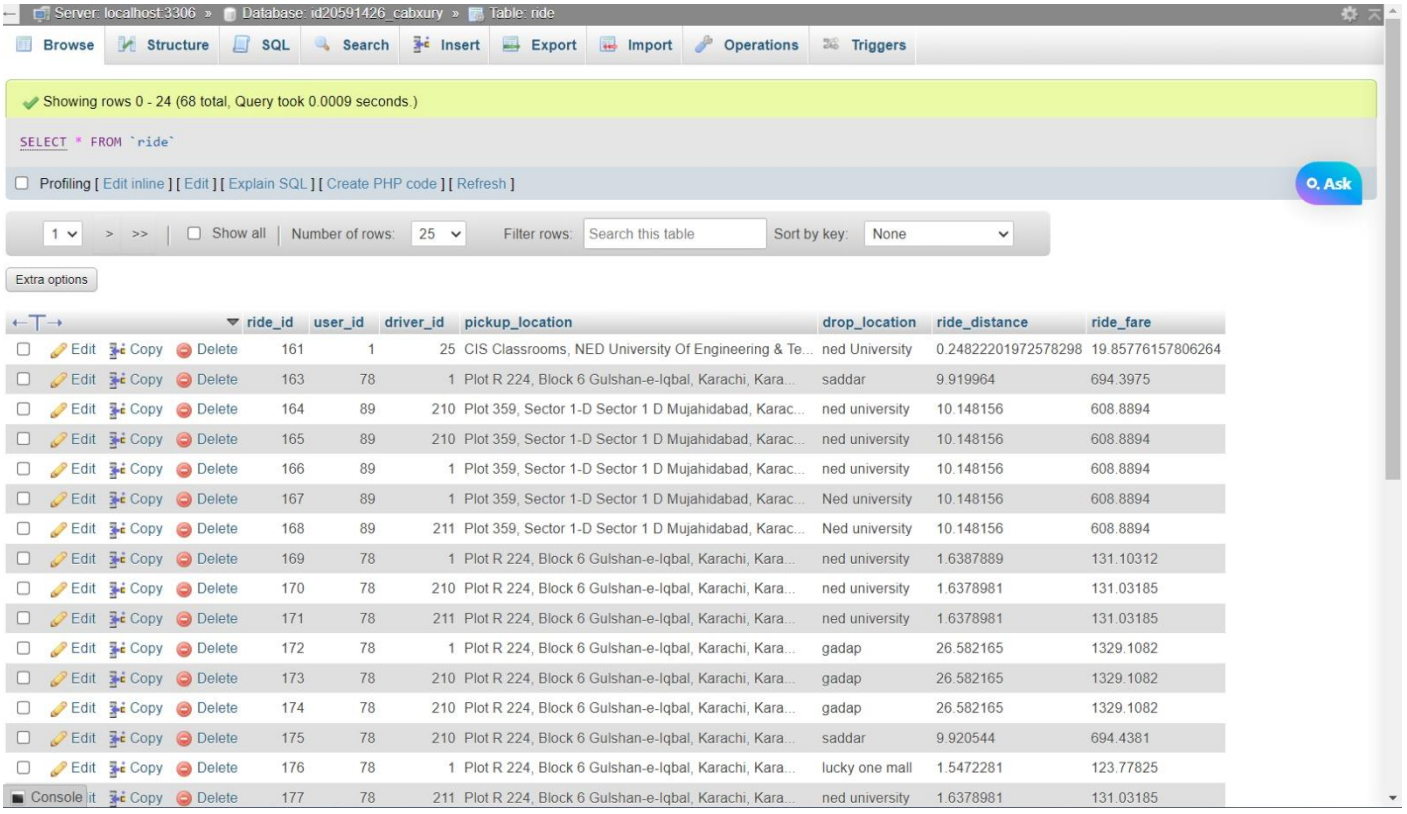




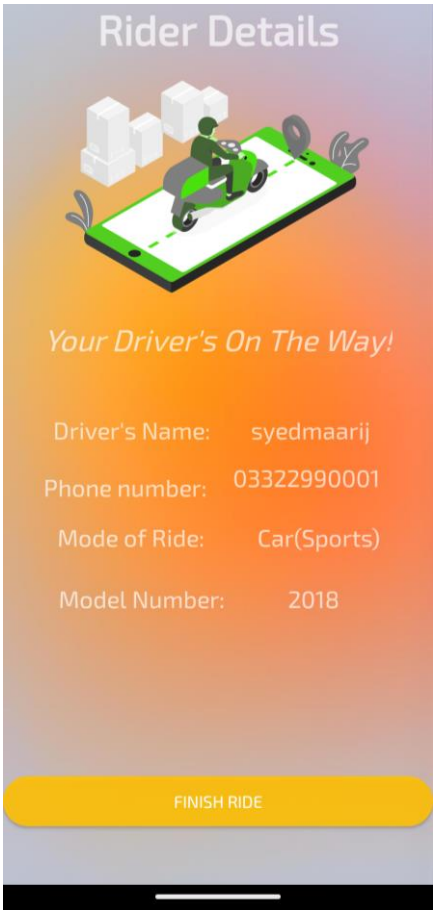
The payment information gathered from the above activities are stored in the database in “Payment” table as shown below in the picture:



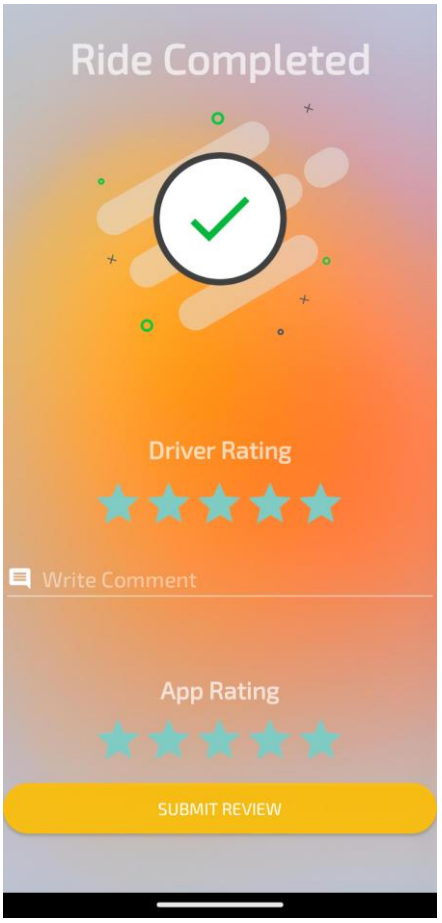
And ride information (i.e. fare, destination, pick up location, ride id) is stored in the ride table as shown below:



After that the rider details are displayed to the user which include driver's name, phone number, mode of ride (i.e. sports, mini etc.) and model number of the vehicle.



Then after clicking on finish ride the user is transported to the final activity. "Review Activity", where user can enter give stars as rating and give comments about his experience.



Server: localhost:3306 » Database: id20591426_cabxury » Table: rating

Browse | Structure | SQL | Search | Insert | Export | Import | Operations | Triggers

	rating_id	ride_id	user_id	driver_id	rating_value	rating_comment	app_rating
Edit Copy Delete	17	201	111	215	5.0	driver is a cheetah	5.0
Edit Copy Delete	18	201	111	212	4.0	good	0.0
Edit Copy Delete	19	201	111	212	4.0	good	4.0
Edit Copy Delete	20	204	131	1	4.5		4.5
Edit Copy Delete	21	204	131	1	4.5		4.5
Edit Copy Delete	22	207	140	217	5.0	good	5.0
Edit Copy Delete	23	211	78	217	5.0	good	5.0
Edit Copy Delete	24	215	78	212	0.0		0.0
Edit Copy Delete	25	216	78	217	1.0	poor driver	4.5
Edit Copy Delete	26	217	78	217	4.0		3.5
Edit Copy Delete	27	219	78	215	4.5		3.0
Edit Copy Delete	28	221	78	211	5.0		4.5
Edit Copy Delete	29	222	78	217	5.0		3.5
Edit Copy Delete	30	227	158	210	5.0	thanks	5.0

Check all *With selected:* Edit Copy Delete Export

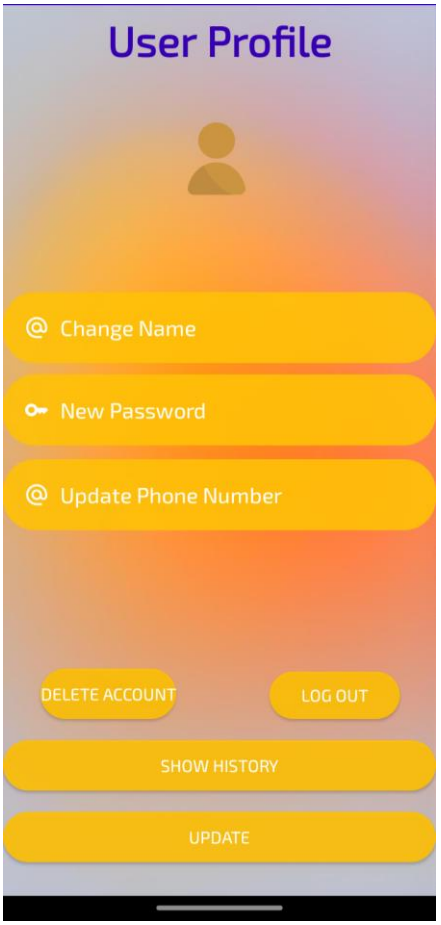
Show all | Number of rows: 25 Filter rows: Search this table Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

Console

The rating information is stored in the rating table according to the corresponding user_id and driver_id.



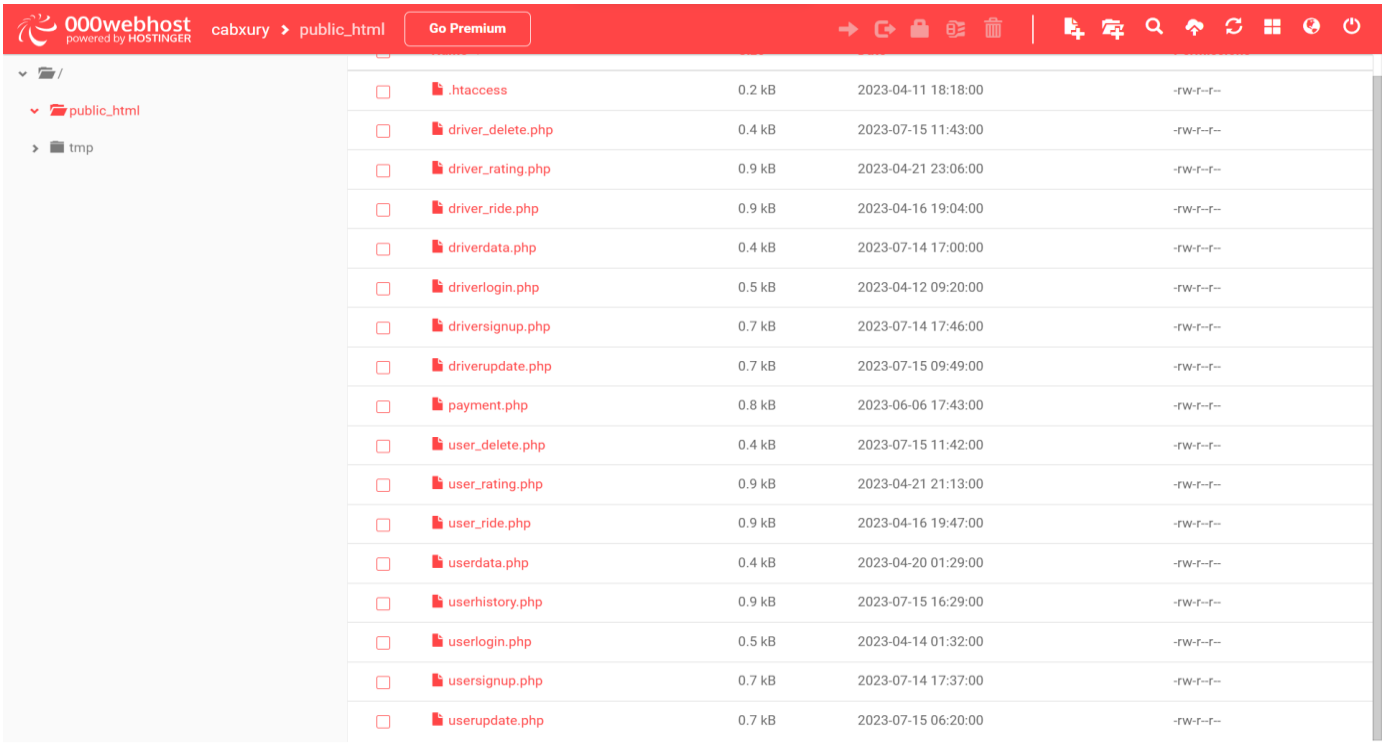
The user and the driver are also able to update their profiles such as changing the name, updating password and update phone number. From the “Profile Activity”, the user/driver can also delete their accounts entirely.

History of the last ride can also be accessed by clicking on the “Show History” button.

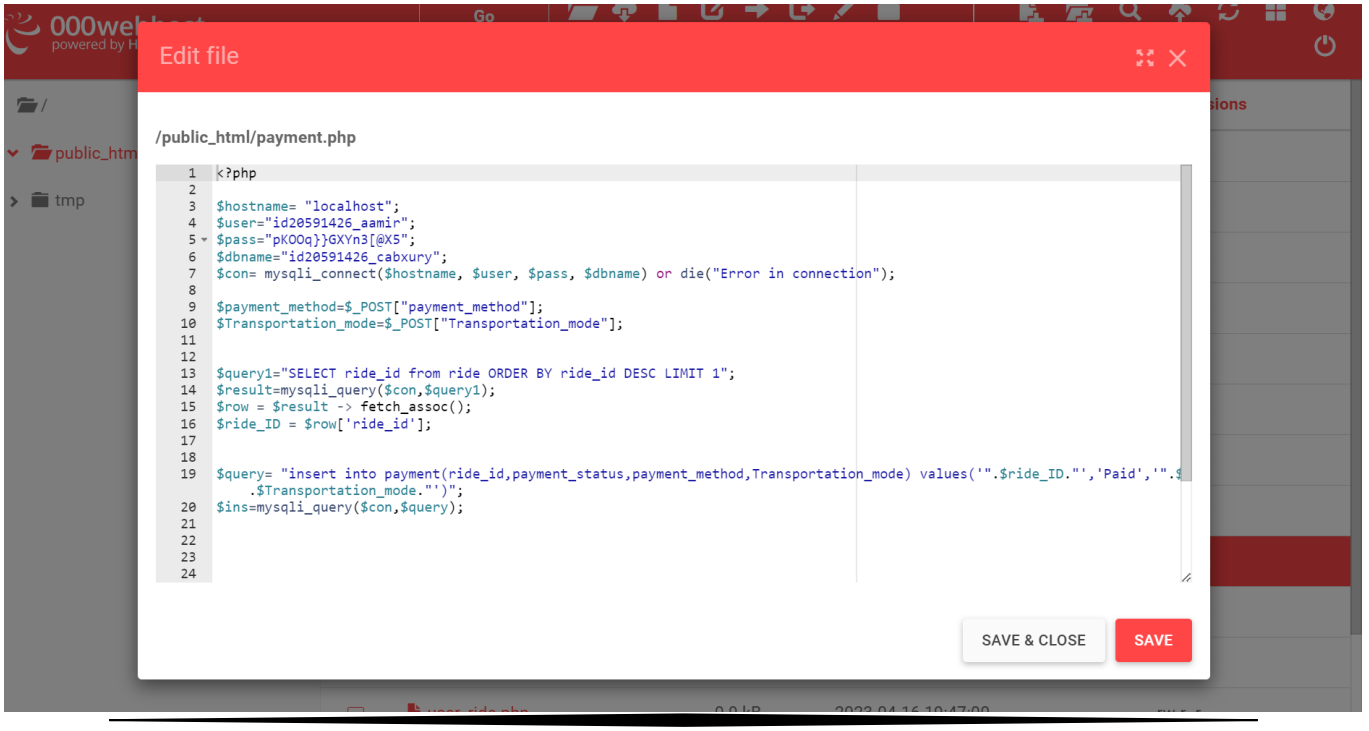
Back-end Implementation:

We implemented the back-end of our database using php. We used a local host service provider at www.000webhost.com.

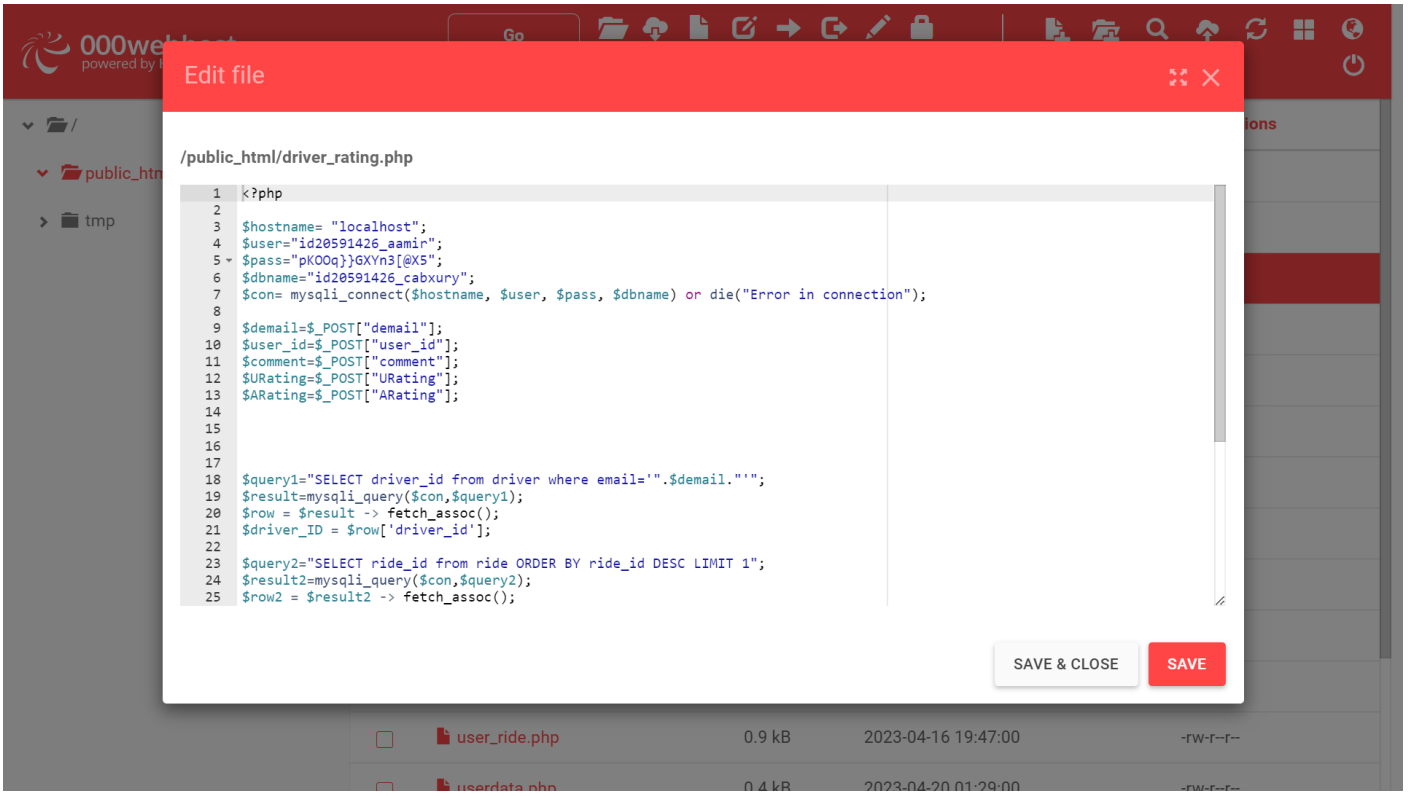
We have created following files for back-end information processing as shown in the picture below:



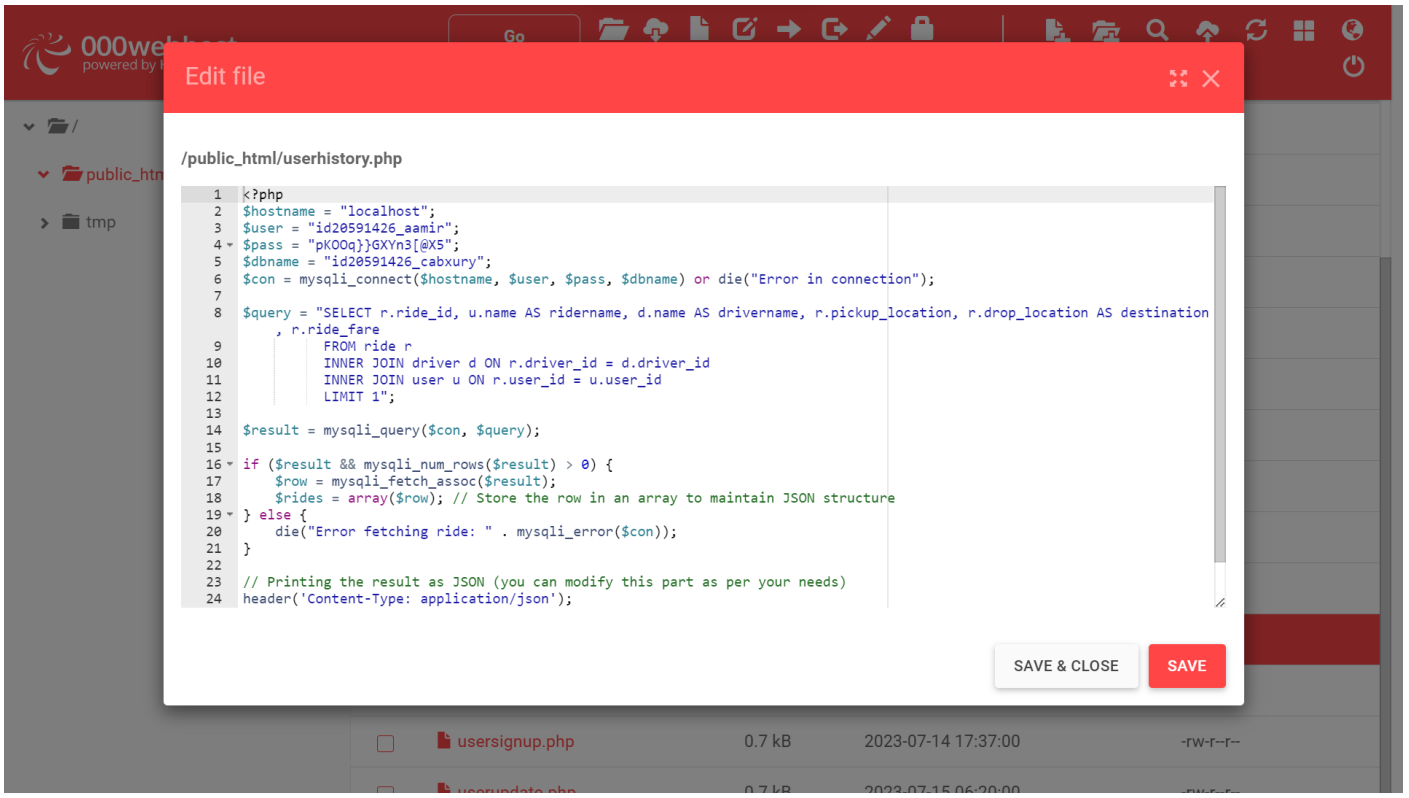
Each file contains the php script for our application's database which are executed in the back-end on the server side. Such as payment.php is implemented as:



And driver_rating.php is implemented as:



Similarly, user_history.php is implemented as:



Last Chapter:

Security of Database

The security of our database is implemented in such a way that user cannot view the confidential information of the driver or other users and similarly driver cannot view the information of the users or other drivers that is confidential. We have implemented constraints in the database in order to ensure security and integrity such as passwords cannot be less than 8 characters and phone numbers cannot be greater than 11 numbers.

Future Improvements

In future we are hoping to implement real-time users and drivers so that they won't have to be simulated as they are now. Also, we will implement one-time login feature so that users and drivers will not have to login always whenever they use the application. Further improving the security is also in our vision.

Nothing is perfect in this world. So, we are also no exception. Although, we have tried our best to present the information effectively, yet, there can be further enhancement in the Application. We have taken care of all the critical aspects, which need to take care of during the development of the Project.

Like the things this project also has some limitations and can further be enhances by someone, because there are certain drawbacks that do not permit the system to be 100%accurate.

Contribution

Everyone contributed in the implementation of database in SQL quires and problem finding.

Muhammad Aamir and Syed Maarij actively contributed in the implementation of back-end development of the application using PHP scripts.

Muhammad Kashif and Usama Khalid actively contributed in the implementation of front-end development of the application building user interfaces.

Syed Maarij and Muhammad Kashif also contributed in writing report with supervision of Muhammad Aamir.