**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**

# Department of Computer Science and Engineering

## BENGALURU, KARNATAKA, INDIA.

## B. TECH. (CSE)

## V SEMESTER

## Aug. – Dec. 2023

**UE21CS351A – DATABASE MANAGEMENT SYSTEM**

**PROJECT REPORT ON**

# Ride sharing app

# Hey-taxii

**Submitted by**



| PES1UG21CS527 | PES1UG21CS481 |
|---|---|
| Sampreethi Y | Rashmi P R |

**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**

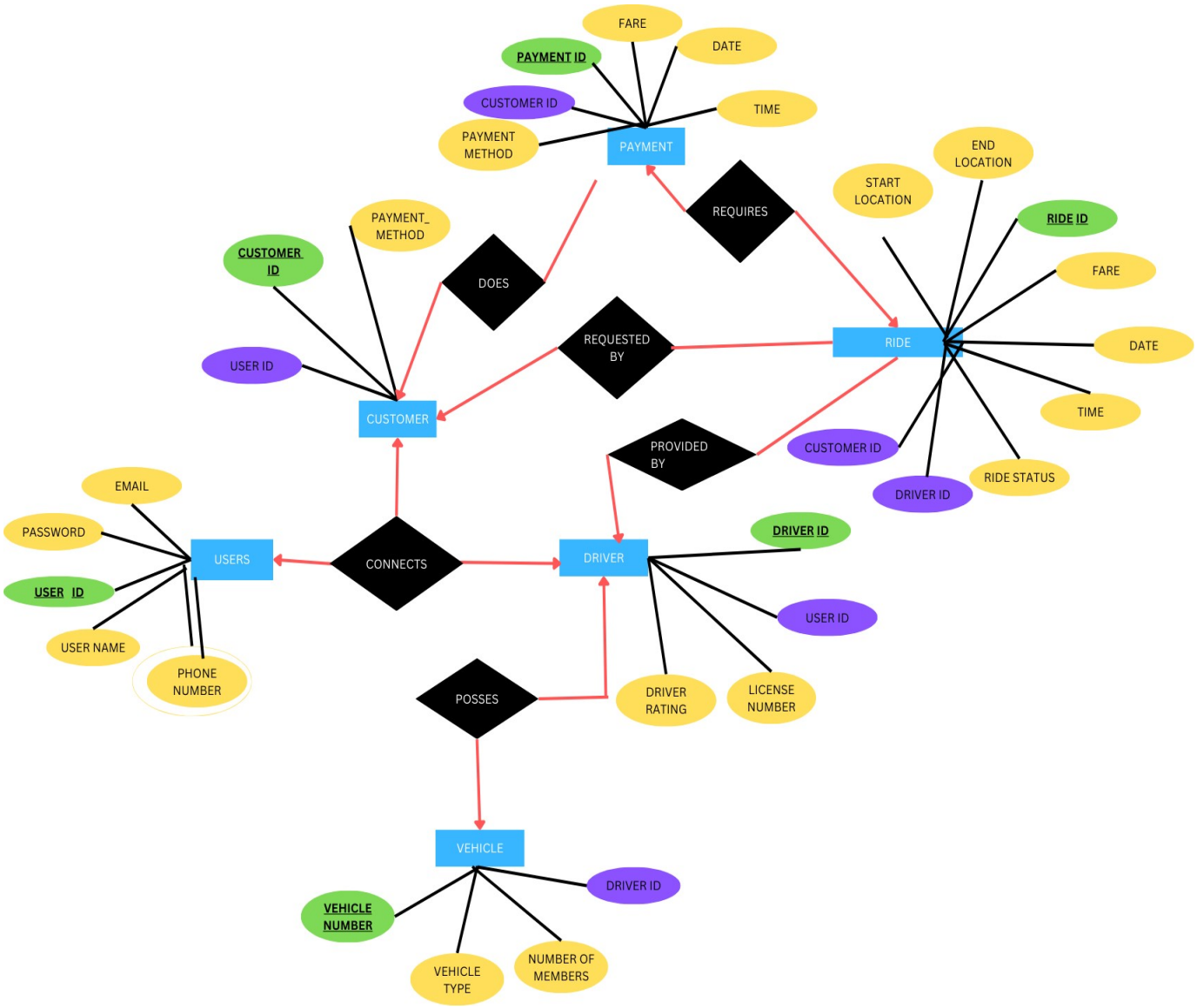
## Short Abstract

**The Hey-Taxii project introduces a dynamic on-demand ride service that prioritizes user convenience and transparency. Users can effortlessly book rides, complete with estimated distances and fares, ensuring a straightforward and predictable experience. The innovative prebooking feature allows users to plan rides for future dates, adding a strategic dimension to the platform.**

**User and driver profiles, established during registration, ensure a secure and personalized interaction. Both parties can update their details as needed, fostering a sense of reliability. Admin oversight, inclusive of customer and driver information, enables efficient monitoring and management, ensuring a smooth and trustworthy on-demand transportation solution. With a commitment to user-centric design and forward-thinking features, Hey-Taxii aims to redefine the on-demand ride service experience.**

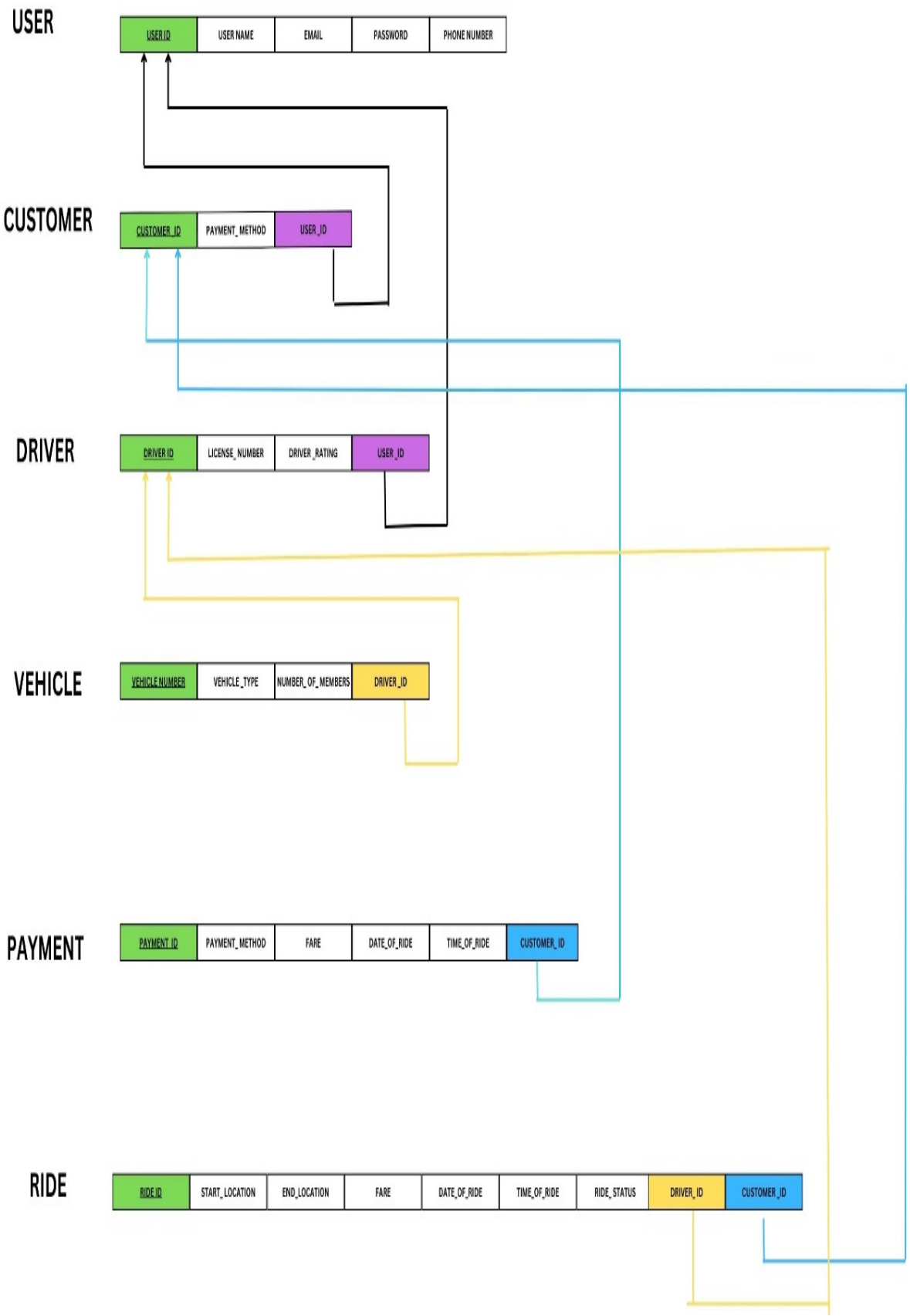
**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**

# E-R DIAGRAM



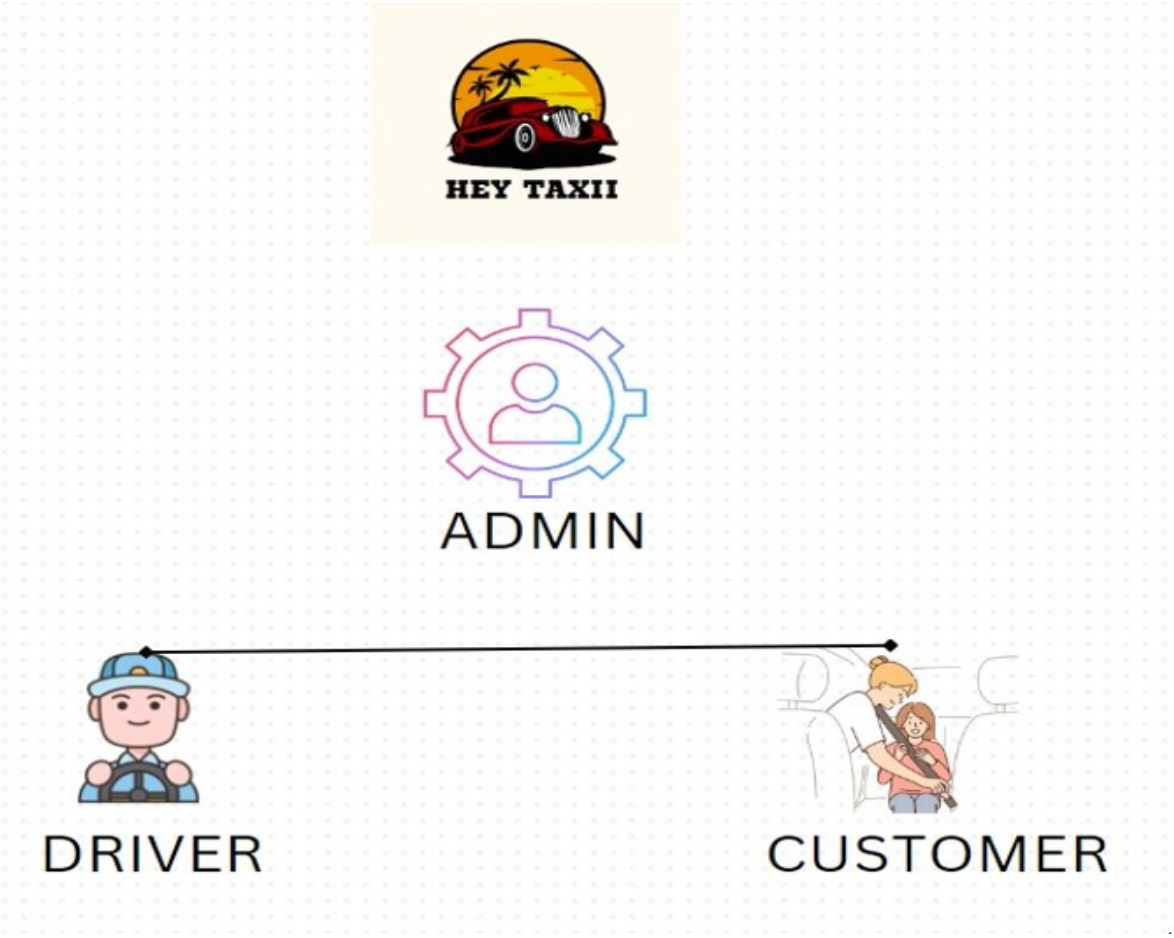
**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**

# E-R SCHEMA



**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**

# USER PRIVILEGES



PES UNIVERSITY
BENGALURU
Department of Computer Science and Engineering

# DDL AND SQL COMMANDS

## USERS table:

```
4  • CREATE TABLE USERS (
5        USER_ID VARCHAR(50) NOT NULL,
6        USERNAME VARCHAR(50) NOT NULL,
7        EMAIL VARCHAR(100) NOT NULL,
8        PASSWD CHAR(6) NOT NULL,
9        PHONE_NUMBER INT NOT NULL,
10       ROLE VARCHAR(20) NOT NULL,
11       PRIMARY KEY (USER_ID)
12   );
13
14 •  desc  users;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| USER_ID | varchar(50) | NO | PRI | NULL | |
| USERNAME | varchar(50) | NO | | NULL | |
| EMAIL | varchar(100) | NO | | NULL | |
| PASSWD | char(6) | NO | | NULL | |
| PHONE_NUMBER | int | NO | | NULL | |
| ROLE | varchar(20) | NO | | NULL | |

## CUSTOMER table:

```
16 • CREATE TABLE CUSTOMER(CUSTOMER_ID VARCHAR(50) NOT NULL,
17   USER_ID VARCHAR(50) NOT NULL,
18   PAYMENT_METHOD VARCHAR (50) NOT NULL,
19   PRIMARY KEY(CUSTOMER_ID),
20   FOREIGN KEY(USER_ID) REFERENCES USERS(USER_ID));
21 • desc CUSTOMER;
22 • CREATE TABLE DRIVER(DRIVER_ID VARCHAR(50) NOT NULL,
23   USER ID VARCHAR(50) NOT NULL,
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| CUSTOMER_ID | varchar(50) | NO | PRI | NULL | |
| USER_ID | varchar(50) | NO | MUL | NULL | |
| PAYMENT_METHOD | varchar(50) | NO | | NULL | |

## DRIVER table:

```
22 • CREATE TABLE DRIVER(DRIVER_ID VARCHAR(50) NOT NULL,
23   USER_ID VARCHAR(50) NOT NULL,
24   LICENSE_NUMBER VARCHAR(50) NOT NULL,
25   DRIVER_RATING INT,
26   PRIMARY KEY(DRIVER_ID),
27   FOREIGN KEY(USER_ID) REFERENCES USERS(USER_ID));
28 • desc DRIVER;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| DRIVER_ID | varchar(50) | NO | PRI | NULL | |
| USER_ID | varchar(50) | NO | MUL | NULL | |
| LICENSE_NUMBER | varchar(50) | NO | | NULL | |
| DRIVER_RATING | int | YES | | NULL | |

## VEHICLE TABLE:

```
30 • CREATE TABLE VEHICLE(VEHICLE_NUMBER VARCHAR(25) NOT NULL,
31   VEHICLE_TYPE VARCHAR(20) NOT NULL,
32   NUMBER_OF_MEMBERS INT NOT NULL,
33   DRIVER_ID VARCHAR(50),
34   PRIMARY KEY(VEHICLE_NUMBER),
35   FOREIGN KEY (DRIVER_ID) REFERENCES DRIVER(DRIVER_ID));
36 • desc VEHICLE;
37
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| VEHICLE_NUMBER | varchar(25) | NO | PRI | NULL | |
| VEHICLE_TYPE | varchar(20) | NO | | NULL | |
| NUMBER_OF_MEMBERS | int | NO | | NULL | |
| DRIVER_ID | varchar(50) | YES | MUL | NULL | |

YES

## PAYMENT TABLE:

```
38 • CREATE TABLE PAYMENT(PAYMENT_ID VARCHAR(50) NOT NULL,
39   PAYMENT_METHOD VARCHAR(50) NOT NULL,
40   FARE INT NOT NULL,
41   DATE_OF_RIDE DATE NOT NULL,
42   TIME_OF_RIDE TIME NOT NULL,
43   CUSTOMER_ID VARCHAR(50) NOT NULL,
44   PRIMARY KEY(PAYMENT_ID),
45   FOREIGN KEY(CUSTOMER_ID) REFERENCES CUSTOMER(CUSTOMER_ID));
46 • desc PAYMENT;
47
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| PAYMENT_ID | varchar(50) | NO | PRI | NULL | |
| PAYMENT_METHOD | varchar(50) | NO | | NULL | |
| FARE | int | NO | | NULL | |
| DATE_OF_RIDE | date | NO | | NULL | |
| TIME_OF_RIDE | time | NO | | NULL | |
| CUSTOMER_ID | varchar(50) | NO | MUL | NULL | |

## RIDE TABLE:

```
48 • CREATE TABLE RIDE(RIDE_ID VARCHAR(50) NOT NULL,
49   START_LOCATION VARCHAR(255) NOT NULL,
50   END_LOCATION VARCHAR(255) NOT NULL,
51   FARE INT NOT NULL,
52   DATE_OF_RIDE DATE NOT NULL,
53   TIME_OF_RIDE VARCHAR(100) NOT NULL,
54   CUSTOMER_ID VARCHAR(50) NOT NULL,
55   DRIVER_ID VARCHAR(50) NOT NULL,
56   PRIMARY KEY(RIDE_ID),
57   FOREIGN KEY(CUSTOMER_ID) REFERENCES CUSTOMER(CUSTOMER_ID),
58   FOREIGN KEY(DRIVER_ID) REFERENCES DRIVER(DRIVER_ID));
59 • desc RIDE;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| RIDE_ID | varchar(50) | NO | PRI | NULL | |
| START_LOCATION | varchar(255) | NO | | NULL | |
| END_LOCATION | varchar(255) | NO | | NULL | |
| FARE | int | NO | | NULL | |
| DATE_OF_RIDE | date | NO | | NULL | |
| TIME_OF_RIDE | varchar(100) | NO | | NULL | |
| CUSTOMER_ID | varchar(50) | NO | MUL | NULL | |
| DRIVER_ID | varchar(50) | NO | MUL | NULL | |

## ADMIN TABLE:

```
61 • create table Admin(admin_ID varchar(10) not null,passwd varchar(4) not null,primary key(admin_id));
62 • DESC Admin;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| admin_ID | varchar(10) | NO | PRI | NULL | |
| passwd | varchar(4) | NO | | NULL | |

## ADDING CONSTRAINT:

```
ALTER TABLE users
ADD CONSTRAINT check_ten_digit_number
CHECK (CHAR_LENGTH(PHONE_NUMBER) = 10 AND PHONE_NUMBER REGEXP '^[0-9]+$');
ALTER TABLE users
MODIFY COLUMN PHONE_NUMBER VARCHAR(15); -- Adjust the length as needed
```

# CRUD OPERATIONS

## INSERT

```
insert into users(USER_ID,USERNAME,EMAIL,PASSWD,PHONE_NUMBER,ROLE) values
('C531','sanjana','sanjanabt2003@gmail.com','2309','9141071392','customer'),
('C532','sanjana','mvsan@gmail.com','0205','9036755056','customer'),
('C527','sampreethi','ysampreethi26@gmail.com','2610','6366052151','admin'),
```

```
payment_method = request.form['payment_method']

# Perform database operations
# Insert into DRIVER table
new_driver = Driver(DRIVER_ID=driver_id, USER_ID=session['user_id'], LICENSE_NUMBER=license_number)
db.session.add(new_driver)

# Insert into VEHICLE table
new_vehicle = Vehicle(VEHICLE_NUMBER=vehicle_number, VEHICLE_TYPE=vehicle_type,
                      NUMBER_OF_MEMBERS=number_of_members, DRIVER_ID=driver_id)
db.session.add(new_vehicle)

# Insert into CUSTOMER table
new_customer = Customer(CUSTOMER_ID=customer_id, USER_ID=session['user_id'], PAYMENT_METHOD=payment_method)
db.session.add(new_customer)
```

## UPDATE

```
def update_driver_customer():
    if request.method == 'POST':
        # Retrieve data from the form
        driver_id_to_update = request.form['driver_id']
        new_license_number = request.form['new_license_number']
        customer_id_to_update = request.form['customer_id']
        new_payment_method = request.form['new_payment_method']

        # Perform database operations
        # Update DRIVER table
        driver_to_update = Driver.query.filter_by(DRIVER_ID=driver_id_to_update).first()
        if driver_to_update:
            driver_to_update.LICENSE_NUMBER = new_license_number
```

## READ

```
@app.route('/view_all_tables', methods=['POST','GET'])
def view_all_tables():
    if request.method == 'POST':
        # Retrieve data from the form
        view_criteria = request.form['view_criteria']
        selected_value = request.form['selected_value']

        # Define the base query
        base_query = """
            SELECT RIDE.RIDE_ID, RIDE.START_LOCATION, RIDE.END_LOCATION, RIDE.FARE, RIDE.DATE_OF_RIDE, RIDE.TIME_OF_RIDE,
                   CUSTOMER.USER_ID AS CUSTOMER_ID, DRIVER.USER_ID AS DRIVER_ID
            FROM RIDE
```

```
        # Define the base query
        base_query = """
            SELECT RIDE.RIDE_ID, RIDE.START_LOCATION, RIDE.END_LOCATION, RIDE.FARE, RIDE.DATE_OF_RIDE, RIDE.TIME_OF_RIDE,
                   CUSTOMER.USER_ID AS CUSTOMER_ID, DRIVER.USER_ID AS DRIVER_ID
            FROM RIDE
            INNER JOIN CUSTOMER ON RIDE.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
            INNER JOIN DRIVER ON RIDE.DRIVER_ID = DRIVER.DRIVER_ID
        """

        # Perform database operations based on view criteria
        if view_criteria == 'start_location':
            query = text(base_query + " WHERE RIDE.START_LOCATION = :value")
        elif view_criteria == 'end_location':
            query = text(base_query + " WHERE RIDE.END_LOCATION = :value")
        elif view_criteria == 'date':
            query = text(base_query + " WHERE RIDE.DATE_OF_RIDE = :value")
        elif view_criteria == 'customer':
            query = text(base_query + " WHERE CUSTOMER.USER_ID = :value")
        elif view_criteria == 'driver':
            query = text(base_query + " WHERE DRIVER.USER_ID = :value")
```

## DELETE OPERATION

```
def delete_driver_customer():
    if request.method == 'POST':
        # Retrieve data from the form
        driver_id_to_delete = request.form['driver_id']
        customer_id_to_delete = request.form['customer_id']
                # Perform database operations
        # Delete from RIDE table first
        rides_to_delete = Ride.query.filter_by(CUSTOMER_ID=customer_id_to_delete).all()
        for ride in rides_to_delete:
            db.session.delete(ride)

        # Delete from PAYMENT table
        payments_to_delete = Payment.query.filter_by(CUSTOMER_ID=customer_id_to_delete).all()
        for payment in payments_to_delete:
            db.session.delete(payment)
```

```
        driver_id_to_delete = request.form['driver_id']
        customer_id_to_delete = request.form['customer_id']
                # Perform database operations
        # Delete from RIDE table first
        rides_to_delete = Ride.query.filter_by(CUSTOMER_ID=customer_id_to_delete).all()
        for ride in rides_to_delete:
            db.session.delete(ride)

        # Delete from PAYMENT table
        payments_to_delete = Payment.query.filter_by(CUSTOMER_ID=customer_id_to_delete).all()
        for payment in payments_to_delete:
            db.session.delete(payment)

        # Perform database operations
        # Delete from DRIVER table
        driver_to_delete = Driver.query.filter_by(DRIVER_ID=driver_id_to_delete).first()
        if driver_to_delete:
```

- BY DEFINING THE RELATIONSHIP IN BETWEEN THE VEHCILE AND DRIVER BY ADDING A CONSTRAINT (CASCADE) WHEREBY DELETING DRIVER IN DRIVER TABLE CORRESPONDING VEHICLE IS ALSO DELETED.

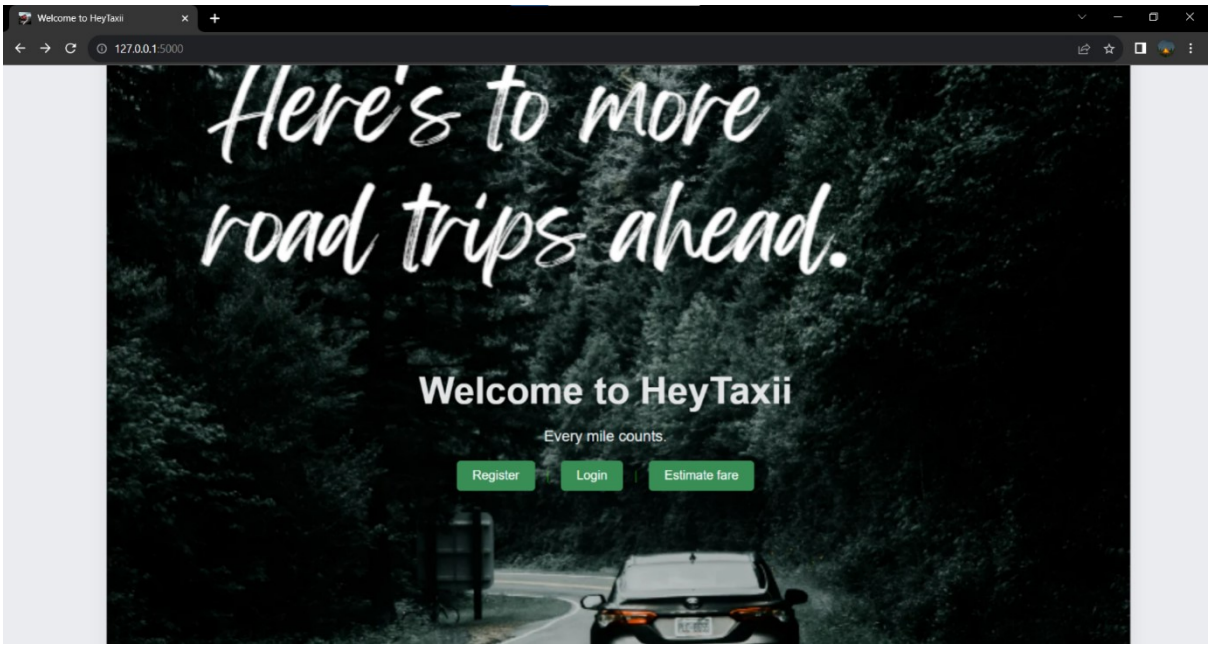
```
    # Define the relationship with the Vehicle table
    vehicles = db.relationship('Vehicle', backref='driver', cascade='all, delete-orphan')
```

- BY DEFINING THE RELATIONSHIP IN BETWEEN THE PAYMENT AND CUSTOMER BY ADDING A CONSTRAINT (CASCADE) WHEREBY DELETING CUSTOMER IN CUSTOMER TABLE CORRESPONDING RIDE AND PAYMENT DETAILS IS ALSO DELETED.

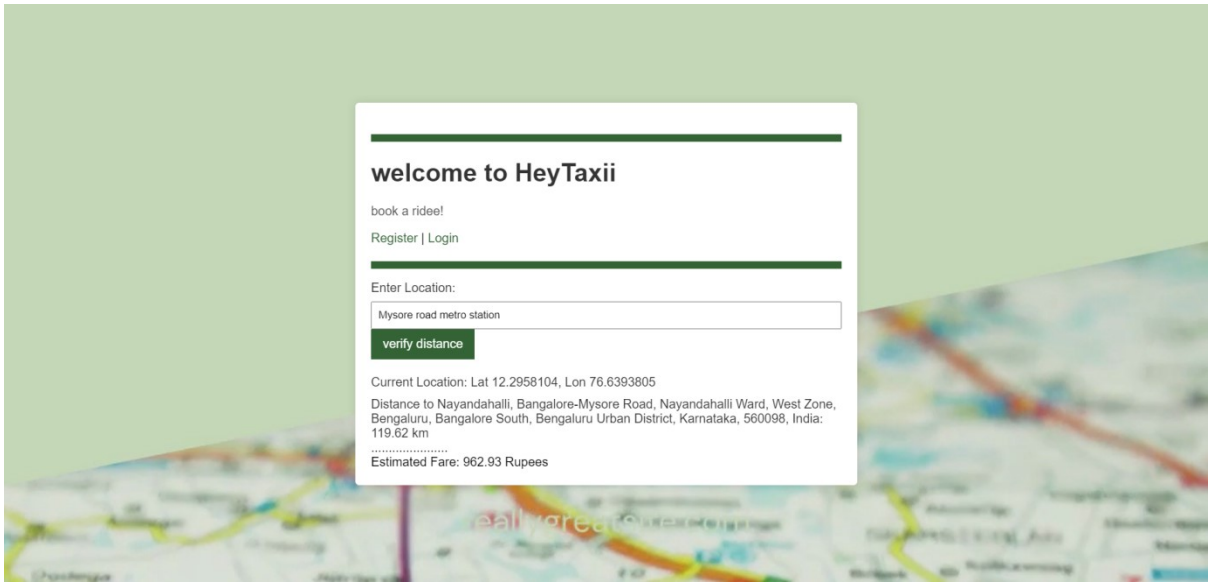
```
# Define the relationship with the Payment table
payments = db.relationship('Payment', backref='customer', cascade='all, delete-orphan')
```

- **List of Functionalities and its associated Query screenshots from front end**

This is the landing page of our website where the user has the options to register if he doesn't have an account, or can login if he has an account and even can just estimate the fare to his destination without necessarily logging in.



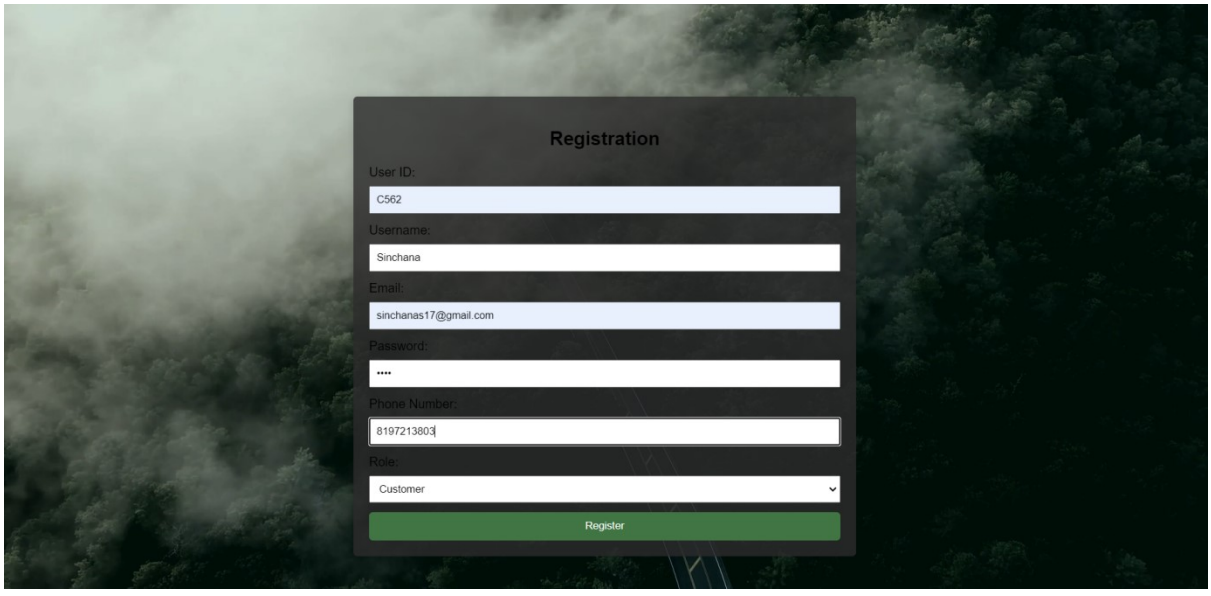

On clicking the Estimate fare button , the user will be directed to the page where he can estimate the fare to his desired destination by entering the valid place name /address/postal code. The current location of the user is fetched with the help of openstreetmap API and the distance between the current location and destination of the user is calculated and respective price and distance is displayed to the user. Price is 30 INR  for within 3km radius of the current location and 8rupees is charged for every extra km .




On clicking Register button user can register to our website by providing relevant information.

3 different kinds of account are supported by our website with different functionalities for each of them.

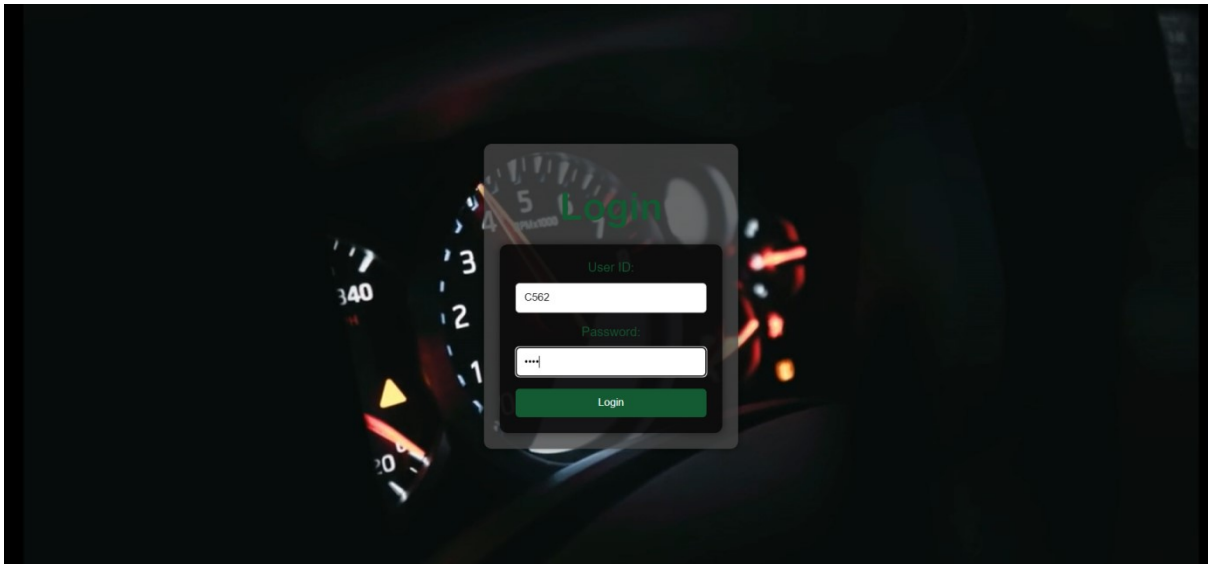
- **REGISTERING AS CUSTOMER**




The details entered in this page will be reflected in USERS table of hey_taxii database.

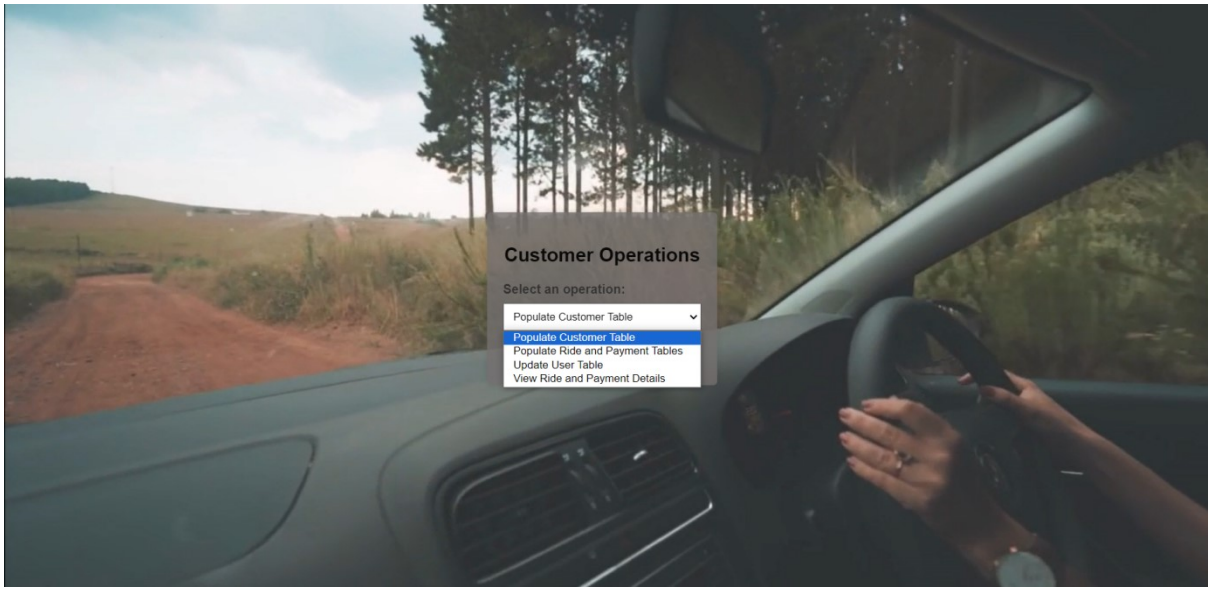
| | | | | | | |
|---|---|---|---|---|---|---|
| | C562 | Sinchana | sinchanas17@gmail.com | 1908 | 8197213803 | customer |

After registering, the user account is created with the customer role. And its get redirected to Login page Now the user can login via his/her valid credentials.

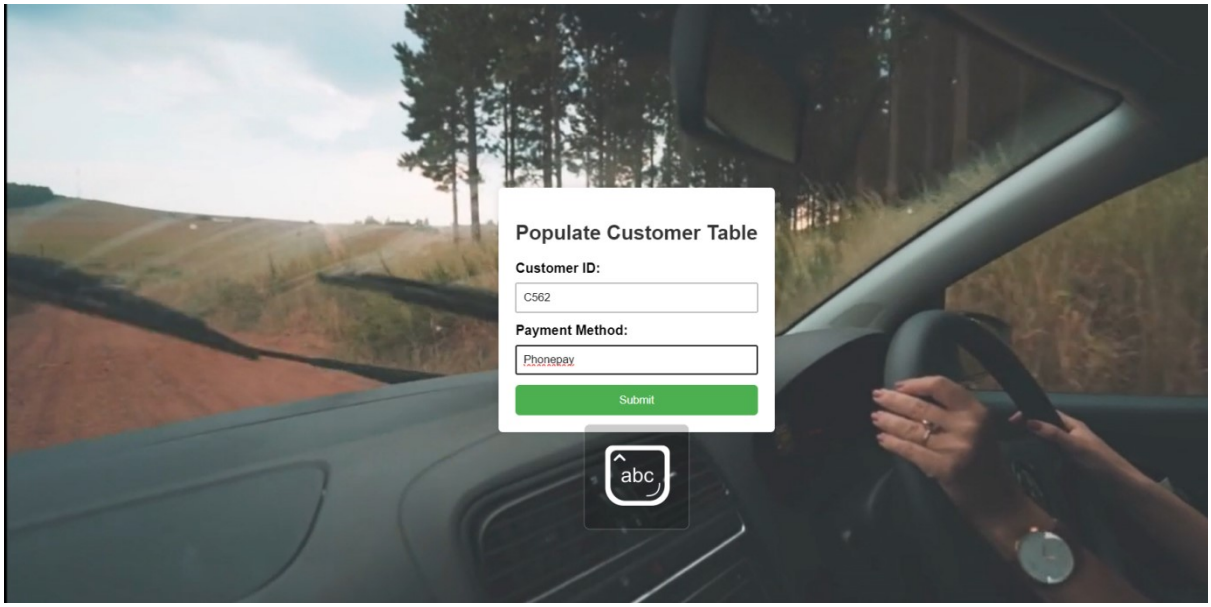



After logging in, the customer will be directed to the intermediary page where he/she can choose.

The actions to perform. Customer has 4 actions to perform.

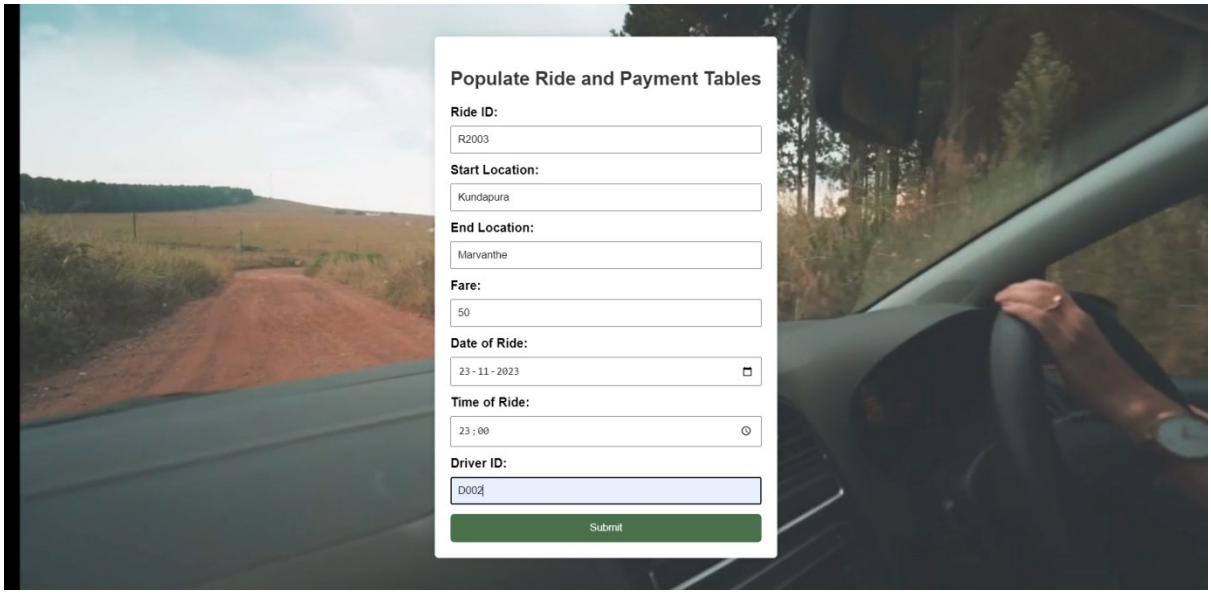



On choosing the first option, the customer will be directed to a page where he/she can enter the mode of payment that customer opts for




These details will be reflected in customers table .


On clicking the second option the customer can book a ride to his/ her desired location and time by entering all the necessary details and estimated fare from previous steps as discussed earlier the details of this will be reflected in ride and payments table of the database




Payment table will be populated with the information

| | | | | | | |
|---|---|---|---|---|---|---|
| | R2003 | Phonepay | 50 | 2023-11-23 | 23:00:00 | C562 |

Ride table will be populated with the information

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | R2003 | Kundapura | Marvanthe | 50 | 2023-11-23 | 23:00 | C562 | D002 |

On choosing update user table option from the drop down menu of the customer, the customer is redirected to a page where he/she can update the information.

Before update:

| | | | | | | |
|---|---|---|---|---|---|---|
| | C562 | Sinchana | sinchanas17@gmail.com | 1908 | 8197213803 | customer |

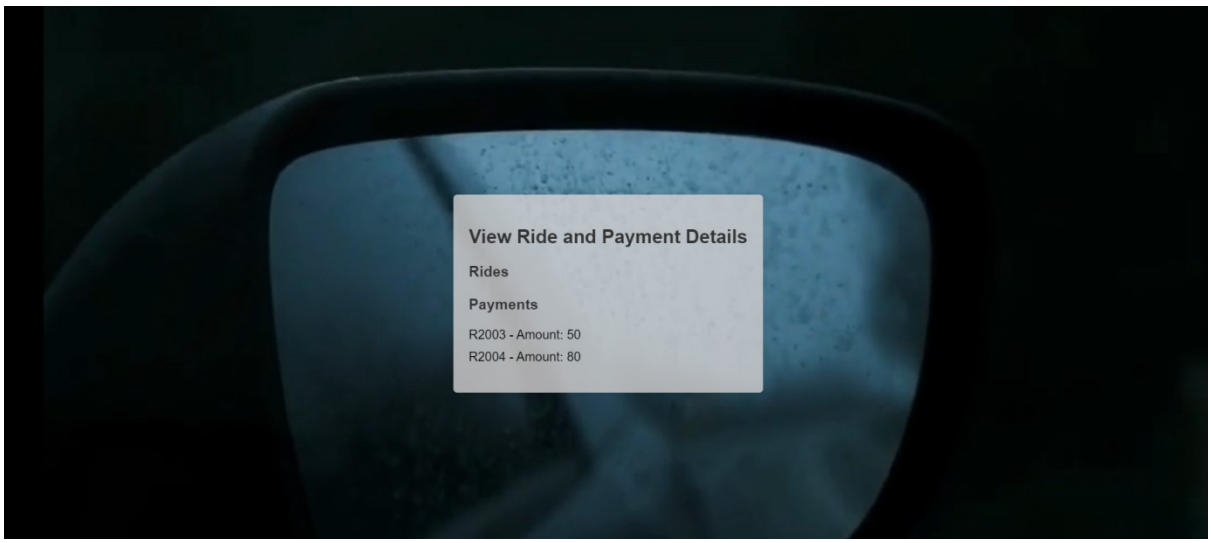
After Update:

| | | | | | | |
|---|---|---|---|---|---|---|
| | C562 | Sinchana | Sinchanaa@gmail.com | 1426 | 8105874789 | customer |

The details can be updated .

On choosing the 4th option , customer can view their ride and payment details, ride ID along with the

Amount paid of all the rides that customer has taken is displayed.




Ride table :

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | R2003 | Kundapura | Marvanthe | 50 | 2023-11-23 | 23:00 | C562 | D002 |
| | R2004 | Girinagar | nayandahalli metro station | 80 | 2023-11-02 | 14:20 | C562 | D003 |

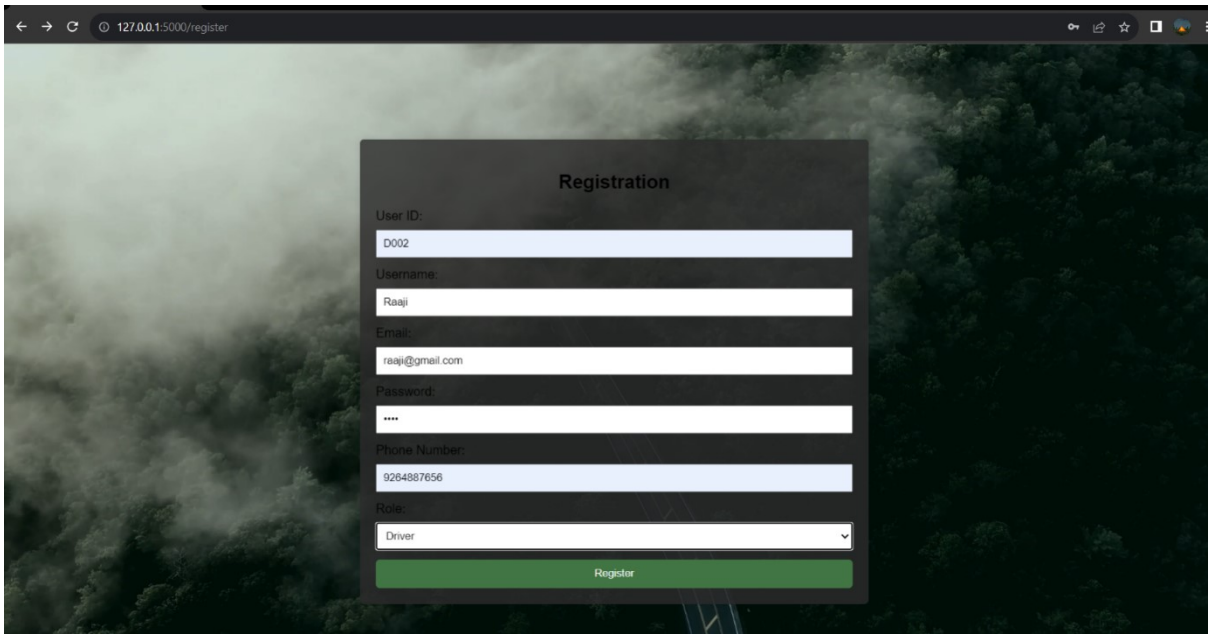
Payment table:

| | | | | | | |
|---|---|---|---|---|---|---|
| | R2003 | Phonepay | 50 | 2023-11-23 | 23:00:00 | C562 |
| ▶ | R2004 | Phonepay | 80 | 2023-11-02 | 14:20:00 | C562 |

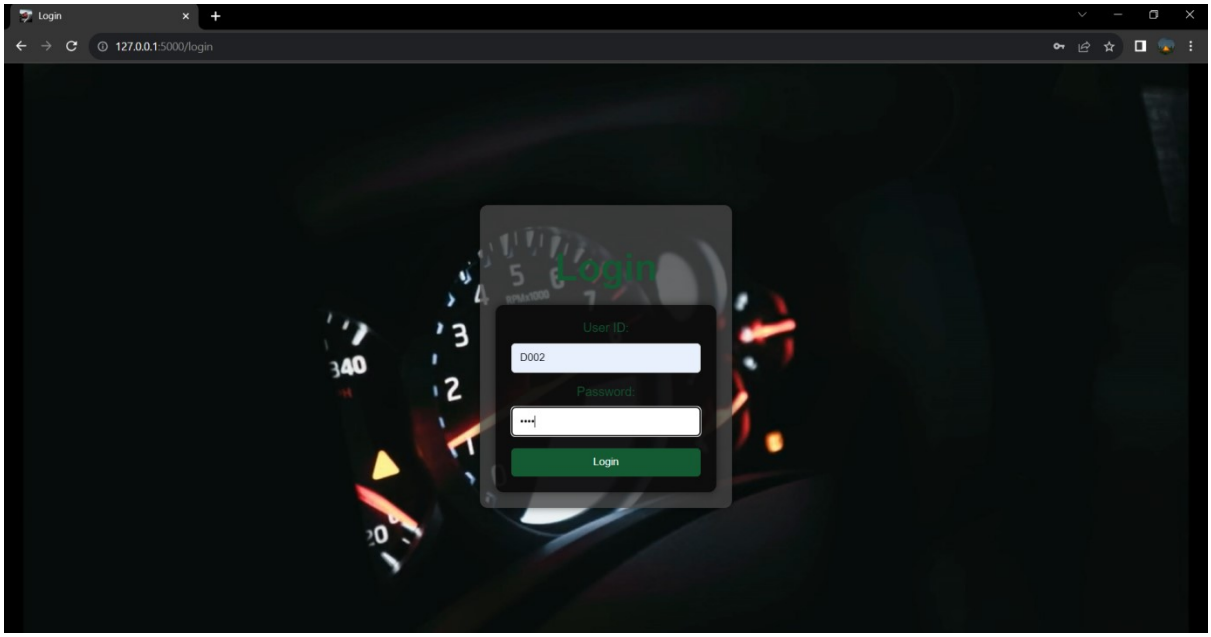
The ride and payment details from the these tables are displayed to the customer based on the customer id , the rides taken by the customer and the amount paid by the customer is displayed .

- **REGISTERING AS A DRIVER**

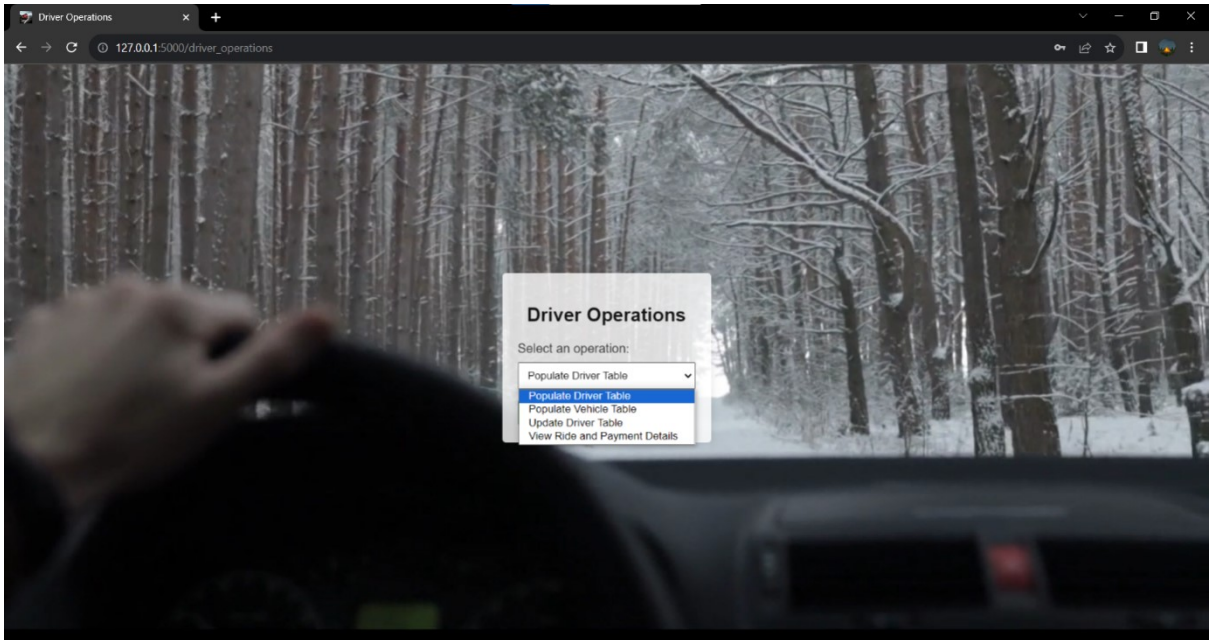
Inorder to register as a driver the user should input all the relevant information and choose the role to be driver from the drop down menu.




On successful registration the page will be directed to the login page where the user can login as a driver by entering the user id and password, if the user id and password matches correctly then the login will direct the driver to a page where he/she can choose the actions that he wishes to perform.




For the driver role certain functionalities are defined where the driver can choose the actions to perform.

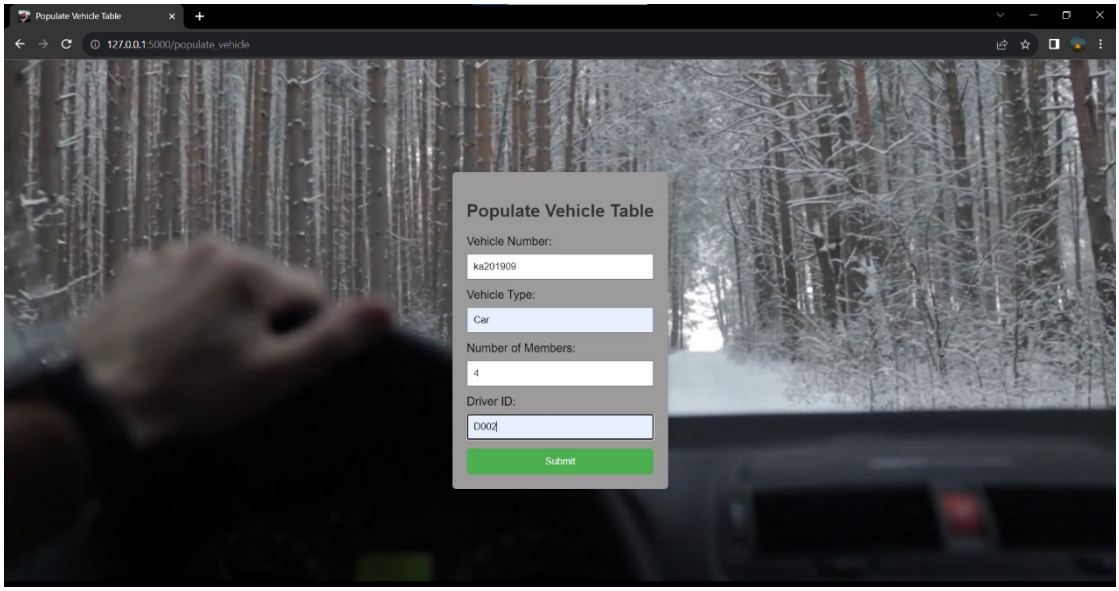



On choosing the first option i.e Populate the driver table , the driver is directed to a page where he can enter the details of him. The driver id user id and the license number of the driver is added upon giving the inputs

The screenshot of Driver table:

| DRIVER_ID | USER_ID | LICENSE_NUMBER |
|---|---|---|
| D002 | D002 | L002 |

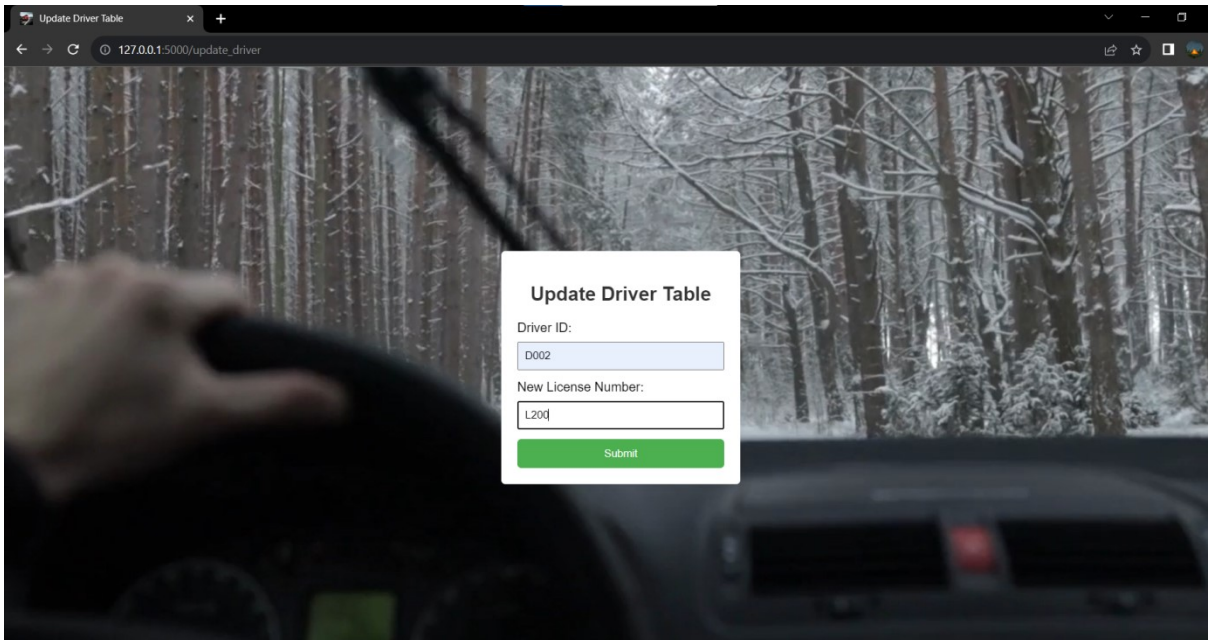
on choosing the Populate Vehicle table , the driver can register his vehicle details as provided in the below screenshot:




Screenshot of vehicle table:

| VEHICLE_NUMBER | VEHICLE_TYPE | NUMBER_OF_MEMBERS | DRIVER_ID |
|---|---|---|---|
| ka201909 | Car | 4 | D002 |

the driver can update his license id on choosing the 3rd option from the menu.



Before update:

| DRIVER_ID | USER_ID | LICENSE_NUMBER |
|---|---|---|
| D002 | D002 | L002 |

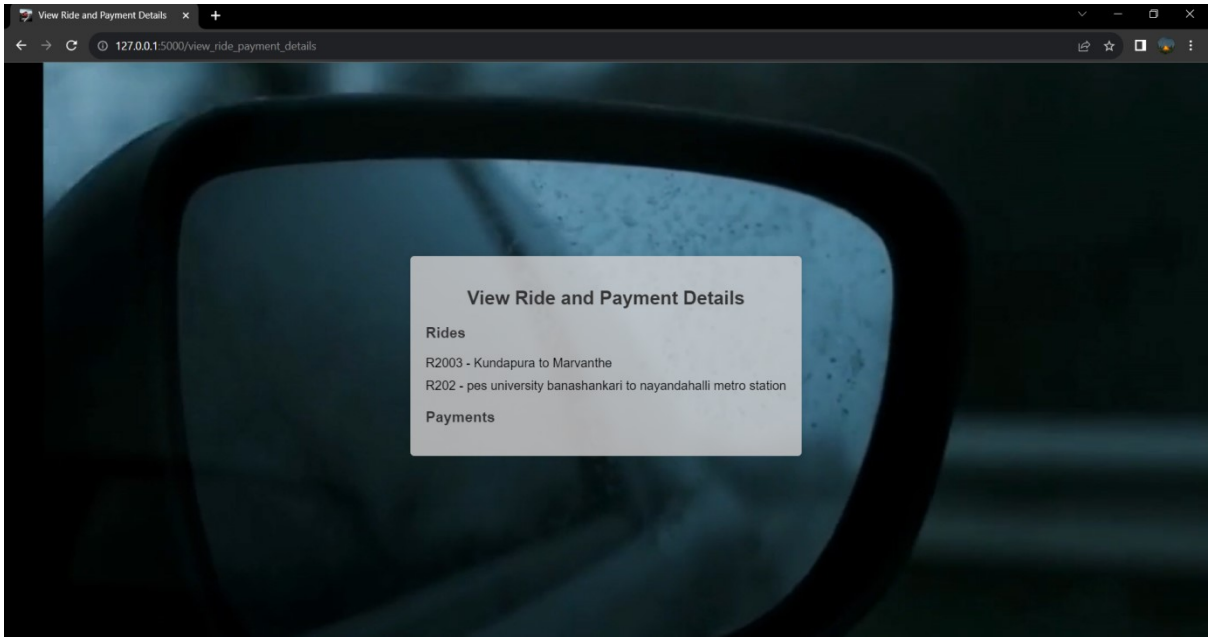
After Update:

| DRIVER_ID | USER_ID | LICENSE_NUMBER |
|---|---|---|
| D002 | D002 | L200 |

The Driver with Driver id D002 had License number **"L002",** now updated his license number to **"L200",** The change in the license number is also reflected in drivers table .

On choosing the 4th option from the menu, the driver is able to view the rides he has been taken

The screenshot of the same is provided below



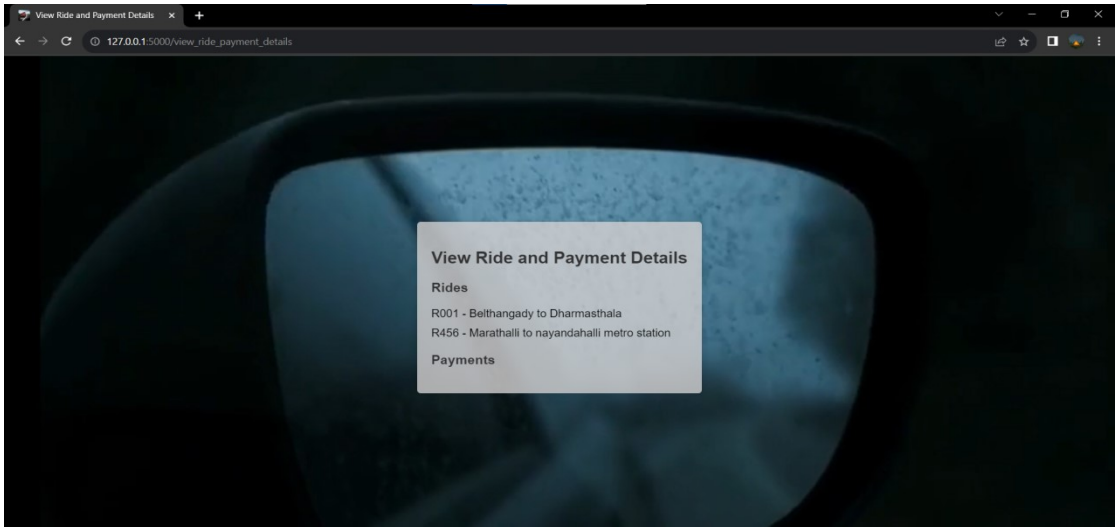

The ride table was populated by the customer, the customer with the customer id C562 and C531 has opted the driver D002 for their ride and the ride details entered by customer is displayed to the driver with driver id D002

| RIDE_ID | START_LOCATION | END_LOCATION | FARE | DATE_OF_RIDE | TIME_OF_RIDE | CUSTOMER_ID | DRIVER_ID |
|---|---|---|---|---|---|---|---|
| R2003 | Kundapura | Marvanthe | 50 | 2023-11-23 | 23:00 | C562 | D002 |
| R202 | pes university banashankari | nayandahalli metro station | 50 | 2023-11-17 | 14:54 | C531 | D002 |

Lets check for the driver with driver id D001

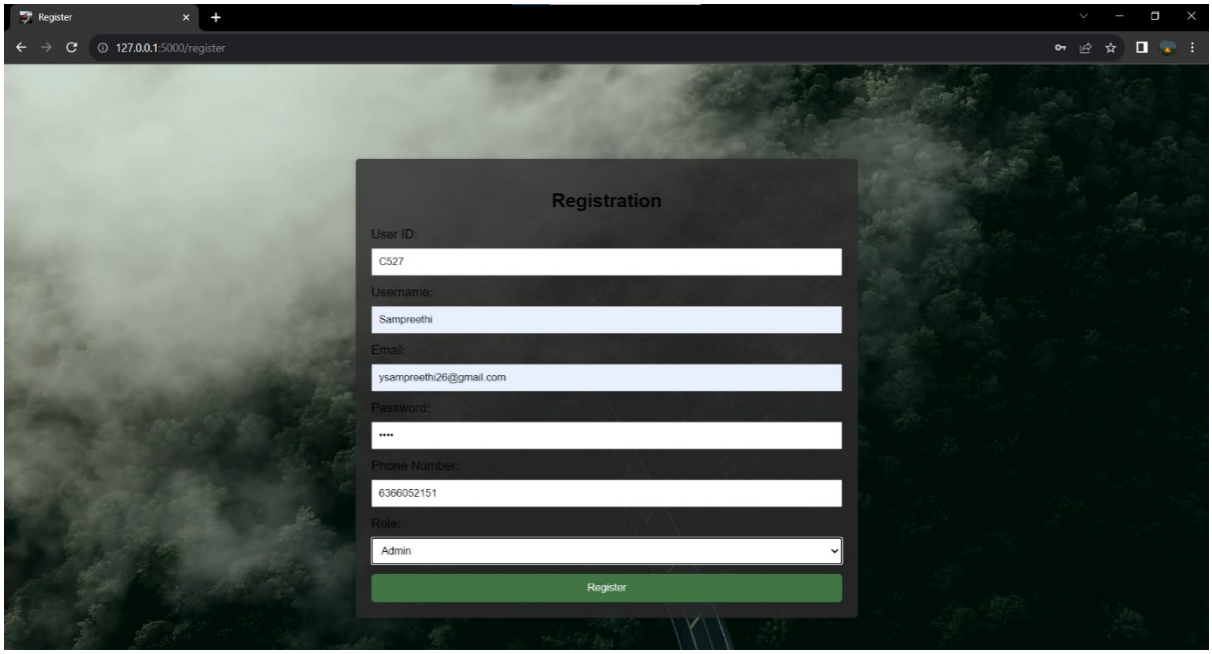
| RIDE_ID | START_LOCATION | END_LOCATION | FARE | DATE_OF_RIDE | TIME_OF_RIDE | CUSTOMER_ID | DRIVER_ID |
|---|---|---|---|---|---|---|---|
| R001 | Belthangady | Dharmasthala | 120 | 2023-11-08 | 15:48 | C526 | D001 |
| R456 | Marathalli | nayandahalli metro station | 120 | 2023-11-10 | 17:53 | C531 | D001 |

When the driver with the driver id D001 logins, the only ride details associated with him is displayed not all.



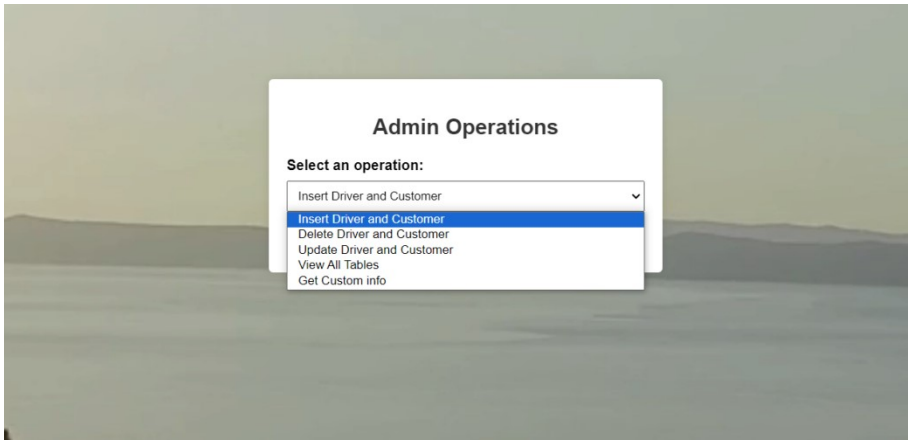

- **REGISTERING AS AN ADMIN**

The 3rd type of account supported by our website is admin account. Admin registeration is also done in the same way as the customer and driver but the role is to been choosen as admin



The admin login page:


The admin has the options to insert driver, customer, update the details view the details he has the authority to delete the driver and customer. He can do these by choosing any of the options from the drop down menu.




before inserting the driver and customer :

driver table:

Result Grid | Filter Rows:

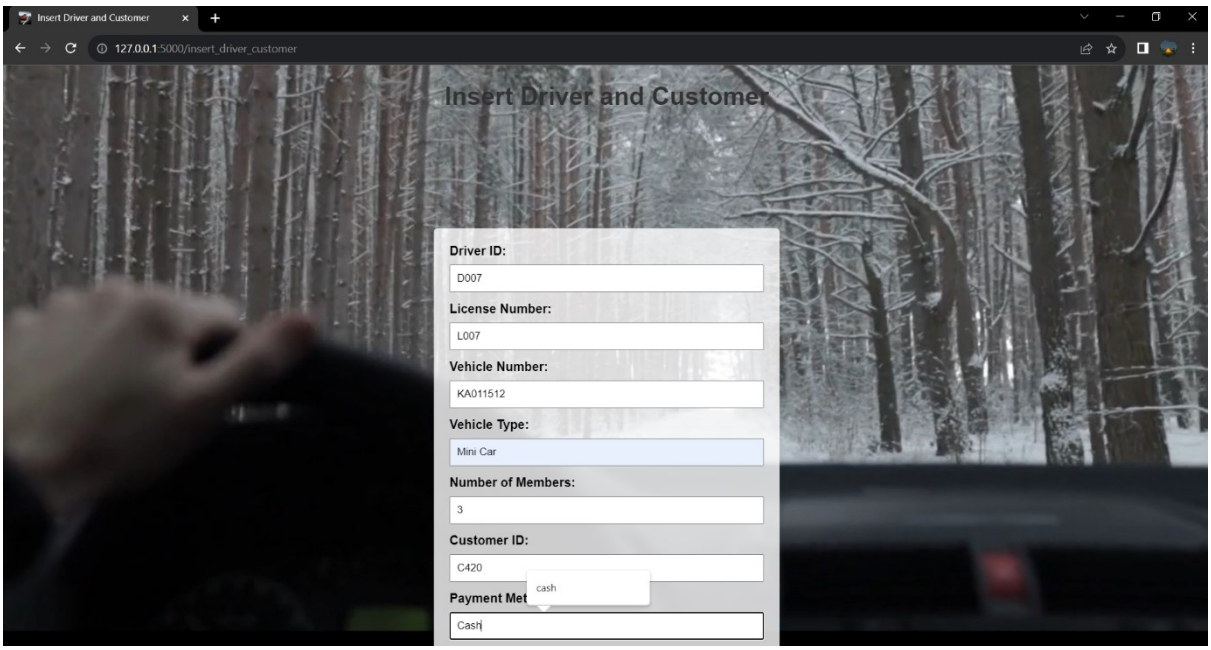
| DRIVER_ID | USER_ID | LICENSE_NUMBER |
|---|---|---|
| D001 | D001 | 1707 |
| D002 | D002 | L200 |
| D003 | D003 | L003 |
| D004 | C527 | L004 |
| NULL | NULL | NULL |

Customer table:

Result Grid | Filter Rows: | Edit

| CUSTOMER_ID | USER_ID | PAYMENT_METHOD |
|---|---|---|
| 526 | C526 | rupay |
| 531 | C531 | cash |
| C521 | C527 | phonepay |
| C526 | C526 | cash |
| C531 | C531 | gpay |
| C562 | C562 | Phonepay |
| NULL | NULL | NULL |

the admin can insert the driver as well the customer to the database by entering all the details asked.




After filling out the details , the driver and customer table is populated with these details.

Driver table has these values:

| DRIVER_ID | USER_ID | LICENSE_NUMBER |
|---|---|---|
| D007 | D007 | L007 |

Customer table has these values:

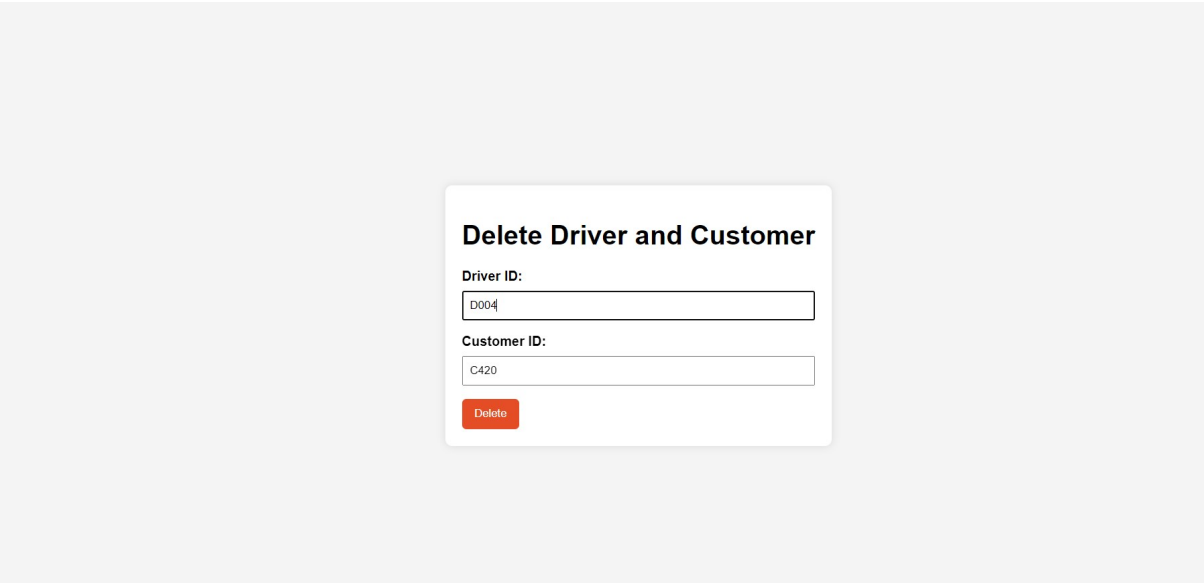
| CUSTOMER_ID | USER_ID | PAYMENT_METHOD |
|---|---|---|
| C420 | C420 | Cash |

The admin also have the option to delete a driver or customer

Now if the admin wishes to delete the driver with driver ID D007.

And he wants to delete the customer with customer ID C420

He can do so by entering the driver id and customer id of his choice that he wishes to delete:




The result of this operation is reflected in the table the table now doesn't have these values.

Driver table doesn't have D004 Now

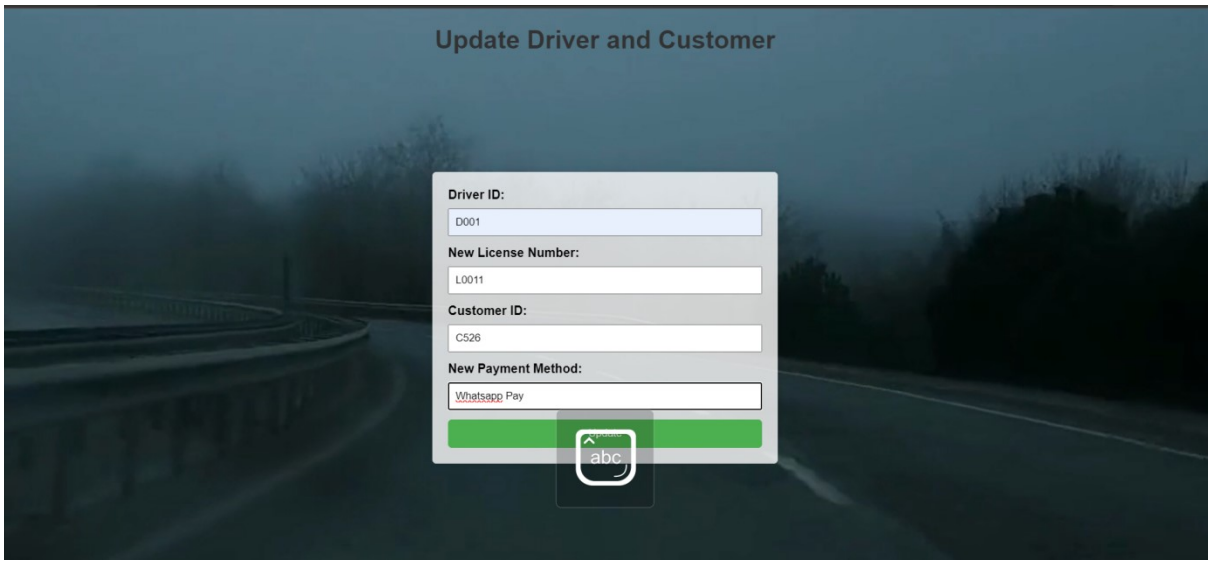
| DRIVER_ID | USER_ID | LICENSE_NUMBER |
|---|---|---|
| D001 | D001 | 1707 |
| D002 | D002 | L200 |
| D003 | D003 | L003 |

Customer table doesn't have the value of C420

Result Grid | Filter Rows:

| CUSTOMER_ID | USER_ID | PAYMENT_METHOD |
|---|---|---|
| 526 | C526 | rupay |
| 531 | C531 | cash |
| C521 | C527 | phonepay |
| C526 | C526 | cash |
| C531 | C531 | gpay |
| C562 | C562 | Phonepay |
| NULL | NULL | NULL |

The admin can even , update the details of the customer as well as driver




The details of the customer and the driver gets updated:

Customer before update:

| CUSTOMER_ID | USER_ID | PAYMENT_METHOD |
|---|---|---|
| 526 | C526 | rupay |

Driver before update:

| DRIVER_ID | USER_ID | LICENSE_NUMBER |
|---|---|---|
| D001 | D001 | 1707 |

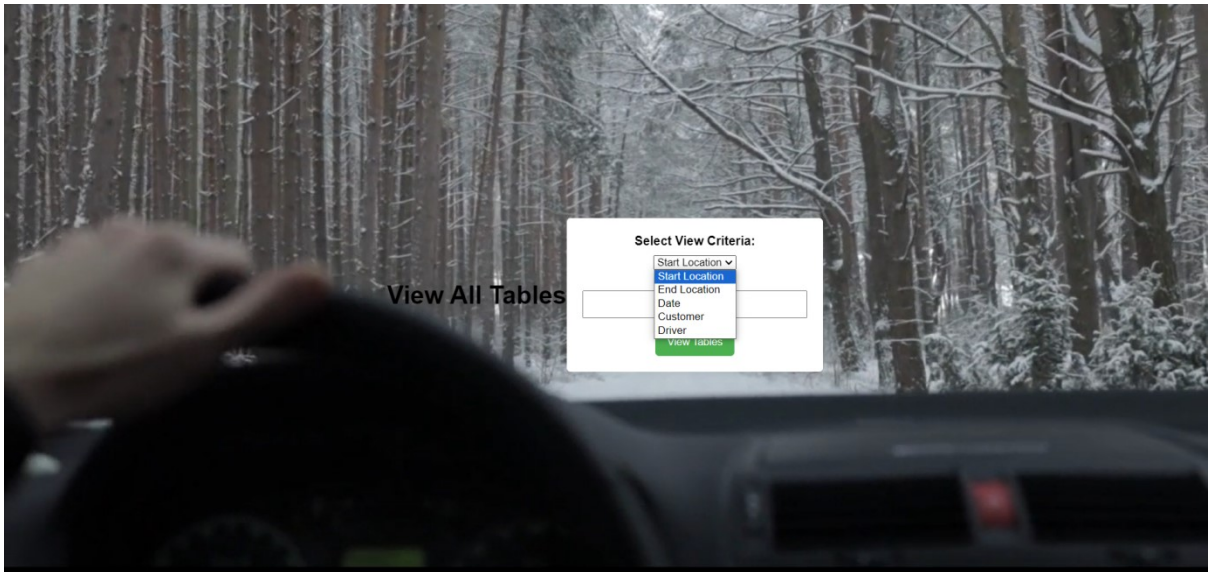
Customer after update:

| CUSTOMER_ID | USER_ID | PAYMENT_METHOD |
|---|---|---|
| C526 | C526 | Whatsapp Pay |

Driver after update:

| | DRIVER_ID | USER_ID | LICENSE_NUMBER |
|---|---|---|---|
| ▶ | D001 | D001 | L0011 |

As the same way, the admin can view all the ride details just as same as the driver and customer by choosing 4th option.




Admin can view the ride details based on

-start location

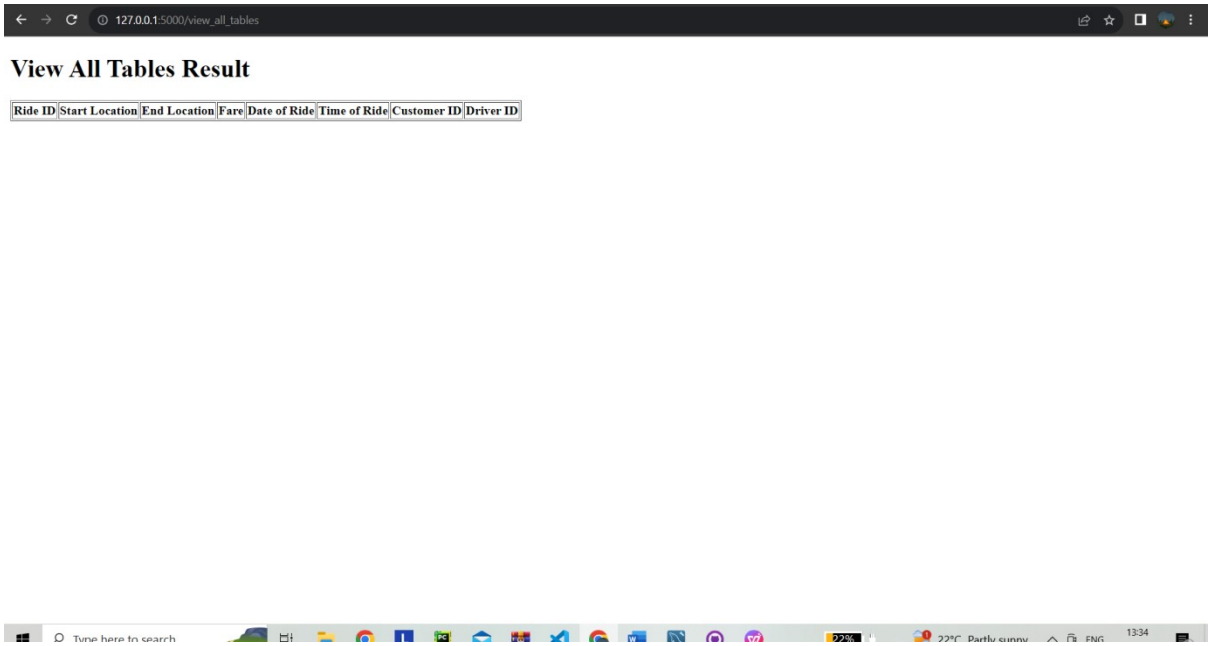
- end location

- date

- customer

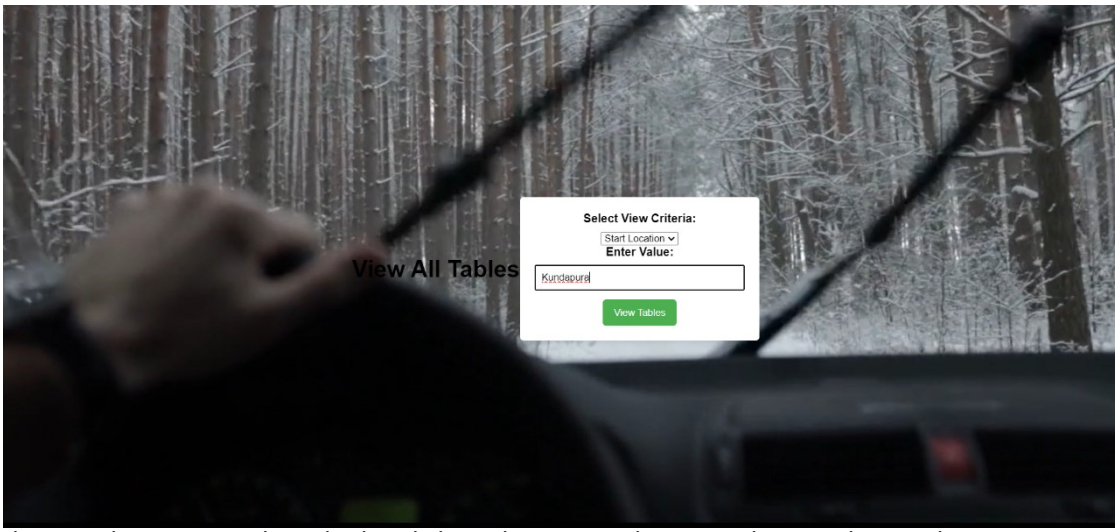
- driver

If there is no ride details found with the entered input the empty tables are displayed




**Ride details based on Start Location:**

If




the admin wishes to view the ride details based on a start location , he can choose the start Location option from the drop down menu and can enter the value of his choice for the Start Location.

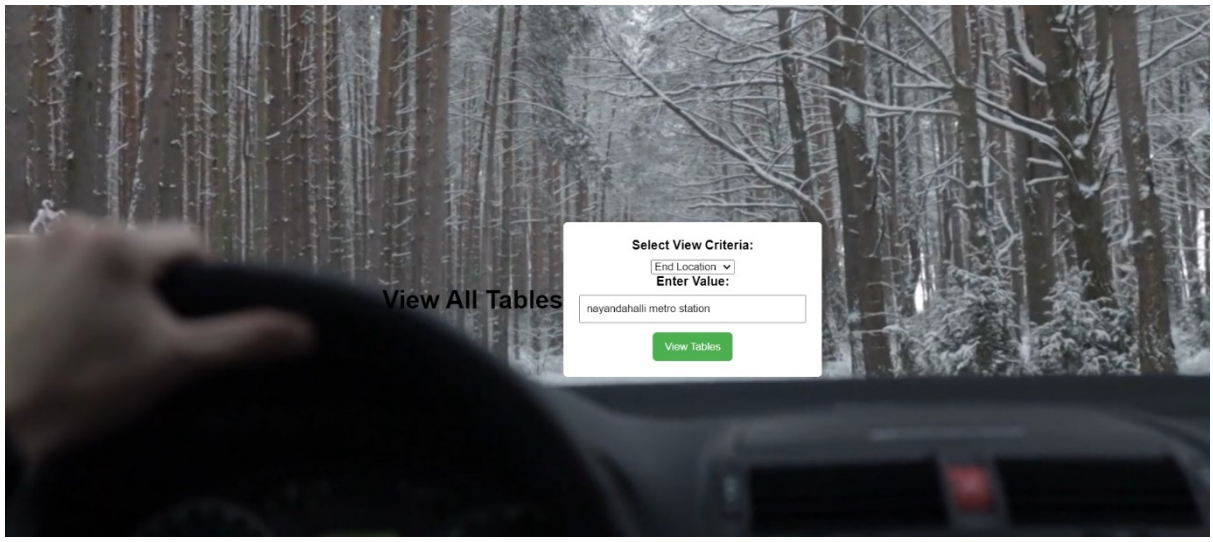
The result of the above input will be:

# View All Tables Result

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R101 | Kundapura | Manipal | 123 | 2023-11-10 | 21:46 | C745 | D002 |
| R2003 | Kundapura | Marvanthe | 50 | 2023-11-23 | 23:00 | C562 | D002 |
| R2101 | Kundapura | Thottam | 192 | 2023-11-10 | 17:55 | C746 | D002 |
| R7477 | Kundapura | Kukki Katte | 139 | 2023-11-23 | 20:07 | C747 | D002 |

The above screenshot deals with the display of information based on particular Start Location “Kundapura”

**Ride details based on End Location:**



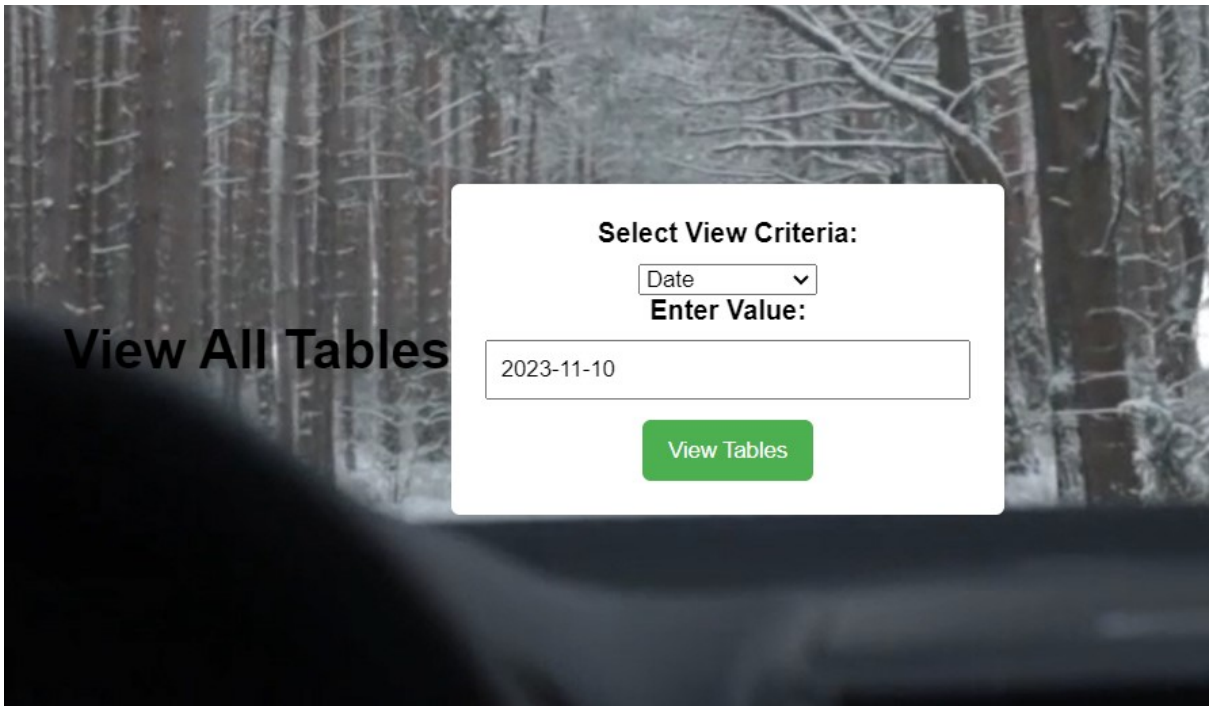

**Same** way if the admin wants to view the ride details based on desired destination he can select the

# View All Tables Result

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R2004 | Girinagar | nayandahalli metro station | 80 | 2023-11-02 | 14:20 | C562 | D003 |
| R202 | pes university banashankari | nayandahalli metro station | 50 | 2023-11-17 | 14:54 | C531 | D002 |
| R456 | Marathalli | nayandahalli metro station | 120 | 2023-11-10 | 17:53 | C531 | D001 |

End Location option from the drop down menu and fill out the desired location in the input field.

The above screenshot deals with the display of information based on particular End Location “nayandahalli metro station”.




**Ride details based on Date:**

**As mentioned earlier,** the admin can view the ride details based on a particular date

# View All Tables Result

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R0044 | Kudla | Karkala | 99 | 2023-11-10 | 16:32 | C526 | D002 |
| R101 | Kundapura | Manipal | 123 | 2023-11-10 | 21:46 | C745 | D002 |
| R2101 | Kundapura | Thottam | 192 | 2023-11-10 | 17:55 | C746 | D002 |
| R456 | Marathalli | nayandahalli metro station | 120 | 2023-11-10 | 17:53 | C531 | D001 |

The above screenshot deals with the display of information based on particular Date i.e "2023-11-10"

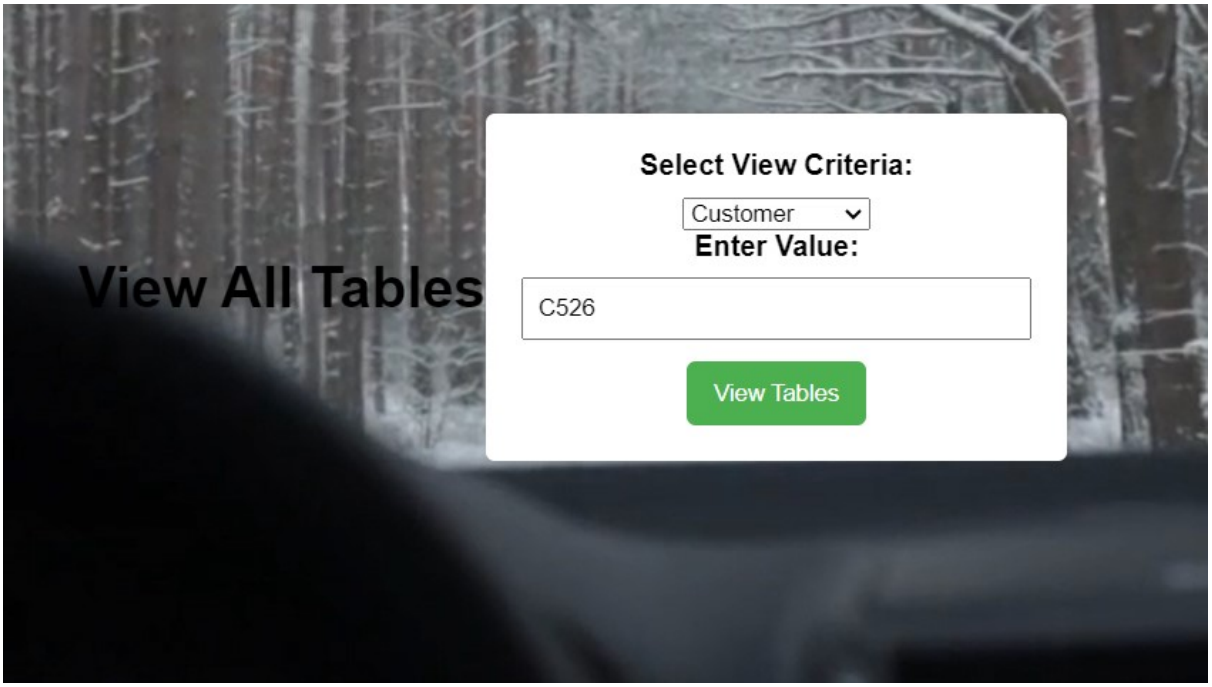
**Ride details based on Driver:**


# View All Tables Result

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R0044 | Kudla | Karkala | 99 | 2023-11-10 | 16:32 | C526 | D002 |
| R101 | Kundapura | Manipal | 123 | 2023-11-10 | 21:46 | C745 | D002 |
| R2003 | Kundapura | Marvanthe | 50 | 2023-11-23 | 23:00 | C562 | D002 |
| R202 | pes university banashankari | nayandahalli metro station | 50 | 2023-11-17 | 14:54 | C531 | D002 |
| R2101 | Kundapura | Thottam | 192 | 2023-11-10 | 17:55 | C746 | D002 |
| R7477 | Kundapura | Kukki Katte | 139 | 2023-11-23 | 20:07 | C747 | D002 |

The above screenshot has the details of all the ride of the driver with driver id “D002”

**Ride details based on customer:**



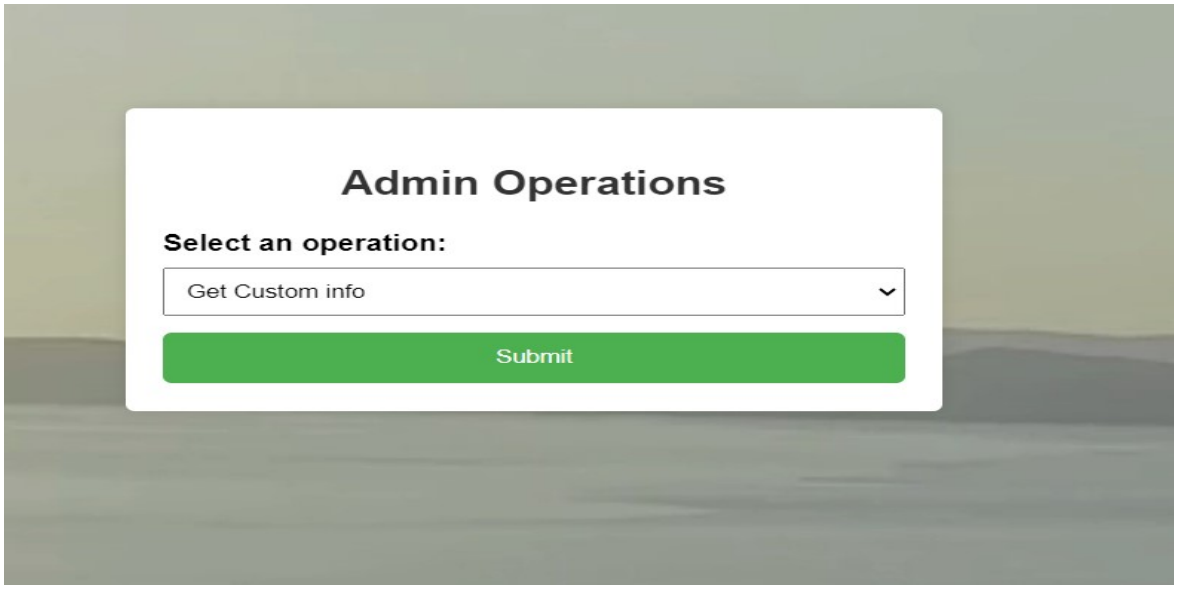

## View All Tables Result

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R001 | Belthangady | Dharmasthala | 120 | 2023-11-08 | 15:48 | C526 | D001 |
| R0044 | Kudla | Karkala | 99 | 2023-11-10 | 16:32 | C526 | D002 |
| R2033 | RUSSIA | BANGALORE | 120 | 2023-11-21 | 12:41 | C526 | D101 |

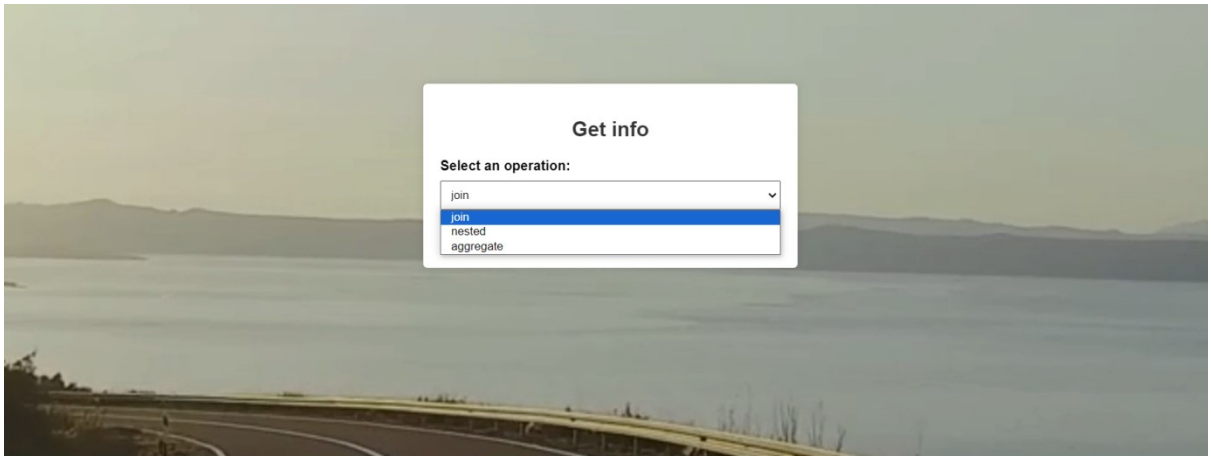
Ride details based on the particular customer with the customer ID “C526”

- **Procedures/Functions/Triggers and their Code snippets for invoking them.**

If the admin chooses get driver count




From the menu he has the options to :



**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**

# Join Query Result

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R001 | Belthangady | Dharmasthala | 120 | 2023-11-08 | 15:48 | C526 | D001 |
| R0044 | Kudla | Karkala | 99 | 2023-11-10 | 16:32 | C526 | D002 |
| R101 | Kundapura | Manipal | 123 | 2023-11-10 | 21:46 | C745 | D002 |
| R2003 | Kundapura | Marvanthe | 50 | 2023-11-23 | 23:00 | C562 | D002 |
| R2004 | Girinagar | nayandahalli metro station | 80 | 2023-11-02 | 14:20 | C562 | D003 |
| R202 | pes university banashankari | nayandahalli metro station | 50 | 2023-11-17 | 14:54 | C531 | D002 |
| R2033 | RUSSIA | BANGALORE | 120 | 2023-11-21 | 12:41 | C526 | D101 |
| R2101 | Kundapura | Thottam | 192 | 2023-11-10 | 17:55 | C746 | D002 |
| R456 | Marathalli | nayandahalli metro station | 120 | 2023-11-10 | 17:53 | C531 | D001 |
| R7477 | Kundapura | Kukki Katte | 139 | 2023-11-23 | 20:07 | C747 | D002 |

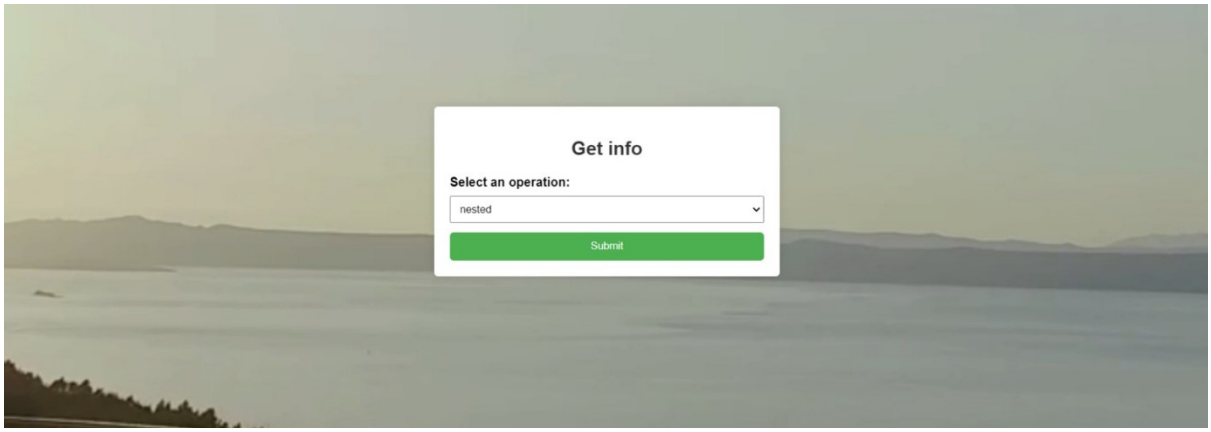
Query:

This SQL query retrieves information about rides from the RIDE table, including details about the start location, end location, fare, date and time of the ride, as well as the user IDs of the associated customer and driver table by joining them

```
@app.route('/join_query', methods=['GET'])
def join_query():
    # Define the base query using text
    base_query = text("""
        SELECT RIDE.RIDE_ID, RIDE.START_LOCATION, RIDE.END_LOCATION, RIDE.FARE, RIDE.DATE_OF_RIDE, RIDE.TIME_OF_RIDE,
               CUSTOMER.USER_ID AS CUSTOMER_ID, DRIVER.USER_ID AS DRIVER_ID
        FROM RIDE
        INNER JOIN CUSTOMER ON RIDE.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
        INNER JOIN DRIVER ON RIDE.DRIVER_ID = DRIVER.DRIVER_ID
    """)

    # Execute the query and fetch results
    result = db.session.execute(base_query).fetchall()

    # Display results
    return render_template('join_query_result.html', result=result)
```

NESTED:

**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**




# Nested Query Example

Vehicle Type: Mini Car Search

# Nested Query Results

| Driver ID | User ID | License Number | Driver Rating | Number of Rides |
|---|---|---|---|---|
| D007 | C527 | L007 | None | 0 |
| D001 | D001 | L0011 | None | 2 |

# Nested Query Example

Vehicle Type: Car Search

# Nested Query Results

| Driver ID | User ID | License Number | Driver Rating | Number of Rides |
|---|---|---|---|---|
| D003 | D003 | L003 | None | 1 |
| D002 | D002 | L200 | None | 6 |

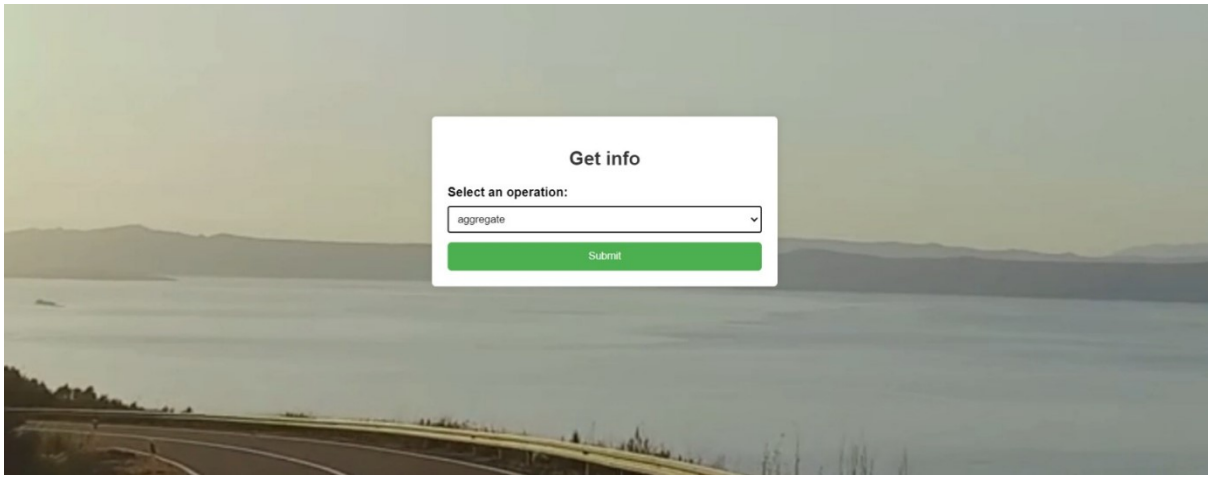
Query:

This query is designed to retrieve information about drivers, their associated rides, and filter the results based on a specified 'vehicle_type'. The result would include driver details along with the count of rides for each driver, limited to those drivers with a specific 'vehicle_type'.

```
@app.route('/nested_query', methods=['GET'])
def nested_query():
    # Retrieve data from the query parameters
    vehicle_type = request.args.get('vehicle_type')

    if vehicle_type is not None and vehicle_type.strip() != "":
        # Define the nested query using text()
        nested_query = text(f"""
            SELECT D.driver_id, D.user_id, D.license_number, D.driver_rating,
                   COUNT(R.ride_id) AS numberOfRides
            FROM DRIVER D
            LEFT JOIN RIDE R ON D.driver_id = R.driver_id
            LEFT JOIN VEHICLE V ON D.driver_id = V.driver_id
            WHERE V.vehicle_type = '{vehicle_type}'
            GROUP BY D.driver_id, D.user_id, D.license_number, D.driver_rating
        """)

        # Execute the nested query and fetch results
        result = db.session.execute(nested_query).fetchall()
```

AGGREGATE:

**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**




# Aggregate Query Result

| Driver ID | Max Fare |
|---|---|
| D001 | 120 |
| D002 | 192 |
| D003 | 80 |
| D101 | 120 |

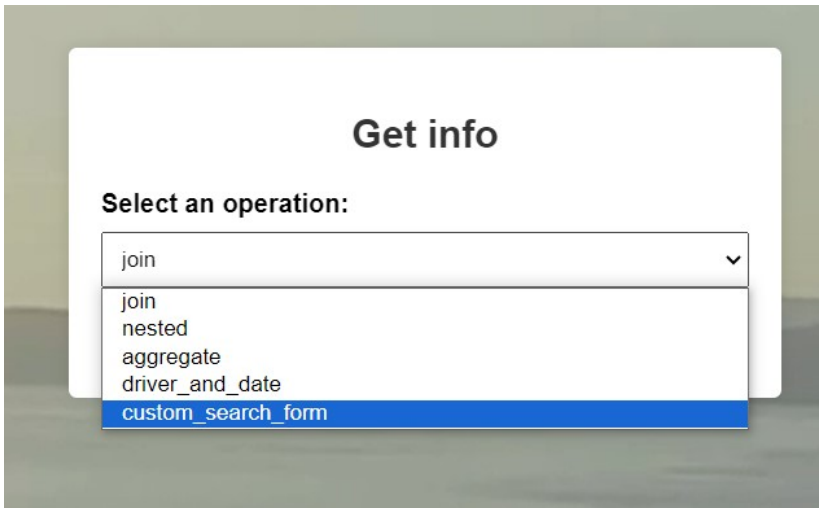
Query:

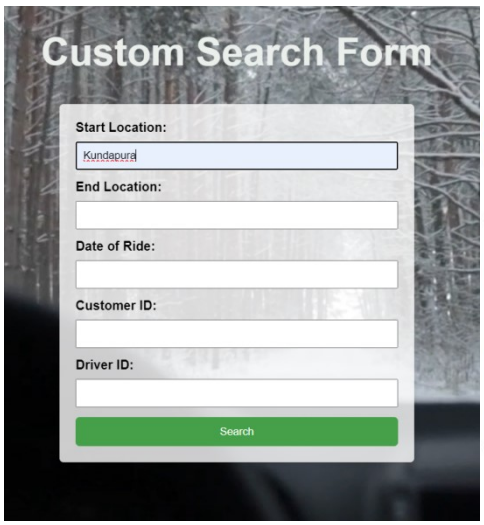
The result of this query will include each unique 'DRIVER_ID' along with the maximum fare associated with that driver.

```
@app.route('/aggregate_query', methods=['GET'])
def aggregate_query():
    try:
        # Define the aggregate query
        aggregate_query = """
            SELECT DRIVER_ID, MAX(FARE) AS MaxFare
            FROM RIDE
            GROUP BY DRIVER_ID
        """

        # Explicitly declare the query as text
        full_query = text(aggregate_query)

        # Execute the query and fetch results
        result = db.session.execute(full_query).fetchall()

        # Display results (modify this based on your frontend requirements
```

# Custom information retrieval

On choosing the custom_search_form option admin can retrieve information customly.



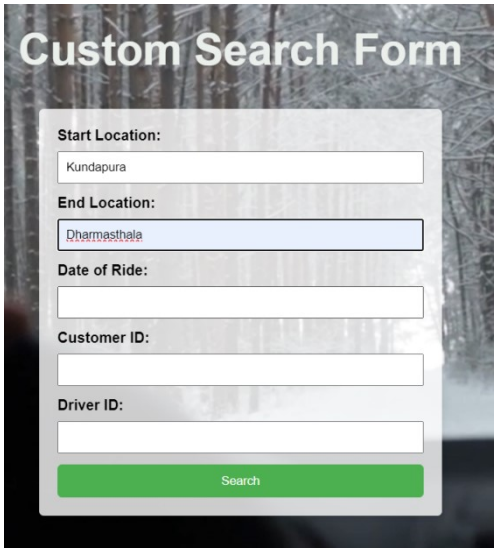




Custom Search Result

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R004 | Kundapura | Dharmasthala | 123 | 2023-11-24 | 02:13 | C526 | D003 |
| R009 | Kundapura | Marvanthe | 123 | 2023-11-23 | 02:17 | C531 | D002 |
| R0105 | Kundapura | Dharmasthala | 80 | 2023-11-23 | 02:22 | C532 | D003 |

On  entering just start location the ride details of the inputted start location can be displayed.

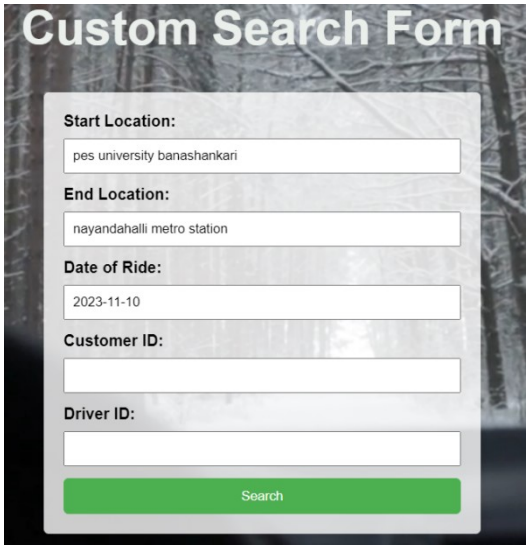
**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**




**Custom Search Result**

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R004 | Kundapura | Dharmasthala | 123 | 2023-11-24 | 02:13 | C526 | D003 |
| R0105 | Kundapura | Dharmasthala | 80 | 2023-11-23 | 02:22 | C532 | D003 |

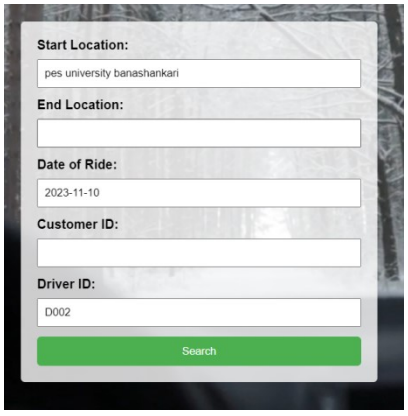
Ride details can be displayed based on both start as well as end location.




**Custom Search Result**

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R002 | pes university banashankari | nayandahalli metro station | 80 | 2023-11-10 | 02:12 | C526 | D002 |
| R0103 | pes university banashankari | nayandahalli metro station | 30 | 2023-11-10 | 23:24 | C532 | D001 |

Ride details can be displayed based on  start as well as end location along with particular date.

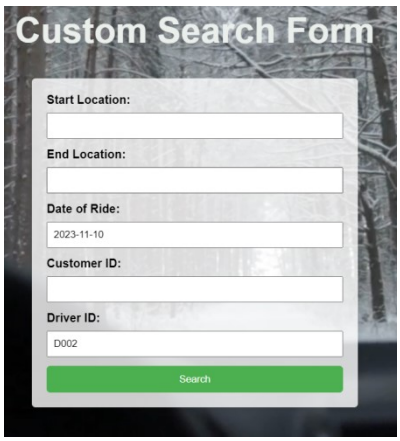



**Custom Search Result**

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R002 | pes university banashankari | nayandahalli metro station | 80 | 2023-11-10 | 02:12 | C526 | D002 |

Ride details can be displayed based on any of location along with particular date and driver id.

**PES UNIVERSITY**
**BENGALURU**
**Department of Computer Science and Engineering**



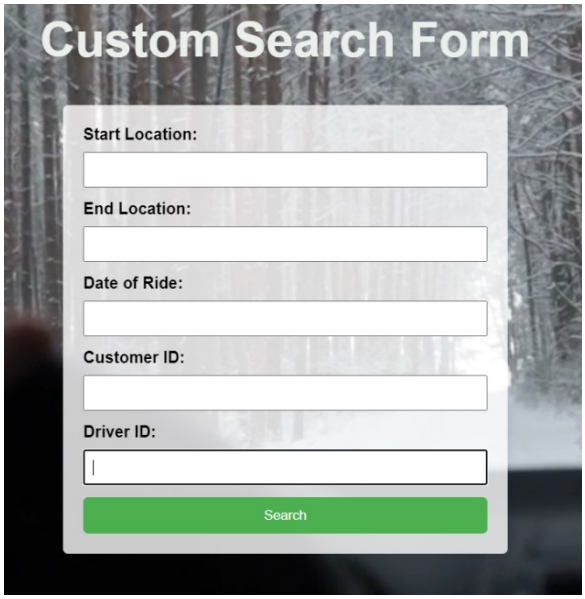

**Custom Search Result**

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R002 | pes university banashankari | nayandahalli metro station | 80 | 2023-11-10 | 02:12 | C526 | D002 |
| R003 | manipal | udupi | 50 | 2023-11-10 | 03:13 | C526 | D002 |
| R0100 | thirthahalli | udupi | 90 | 2023-11-10 | 00:18 | C531 | D002 |

Ride details based on driver and date.

same way for all other combinations admin can retrieve the information.

Without giving any inputs all the ride details as per now booked is displayed.




**Custom Search Result**

| Ride ID | Start Location | End Location | Fare | Date of Ride | Time of Ride | Customer ID | Driver ID |
|---|---|---|---|---|---|---|---|
| R001 | Belthangady | Dharmasthala | 120 | 2023-11-23 | 02:11 | C526 | D002 |
| R002 | pes university banashankari | nayandahalli metro station | 80 | 2023-11-10 | 02:12 | C526 | D002 |
| R003 | manipal | udupi | 50 | 2023-11-10 | 03:13 | C526 | D002 |
| R004 | Kundapura | Dharmasthala | 123 | 2023-11-24 | 02:13 | C526 | D003 |
| R005 | Sulkeri | Dharmasthala | 50 | 2023-11-18 | 02:14 | C526 | D001 |
| R006 | pes university banashankari | Marvanthe | 129 | 2023-11-10 | 02:15 | C531 | D001 |
| R007 | Marathalli | nayandahalli metro station | 500 | 2023-11-23 | 23:20 | C531 | D003 |
| R008 | Belthangady | nayandahalli metro station | 1200 | 2023-11-24 | 23:20 | C531 | D002 |
| R009 | Kundapura | Marvanthe | 123 | 2023-11-23 | 02:17 | C531 | D002 |
| R0100 | thirthahalli | udupi | 90 | 2023-11-10 | 00:18 | C531 | D002 |
| R0101 | pes university banashankari | nayandahalli metro station | 120 | 2023-11-16 | 03:20 | C532 | D002 |
| R0102 | thirthahalli | Marvanthe | 80 | 2023-11-23 | 03:20 | C532 | D003 |
| R0103 | pes university banashankari | nayandahalli metro station | 30 | 2023-11-10 | 23:24 | C532 | D001 |
| R0104 | manipal | udupi | 120 | 2023-11-23 | 23:25 | C532 | D002 |
| R0105 | Kundapura | Dharmasthala | 80 | 2023-11-23 | 02:22 | C532 | D003 |

PROCEDURE:

```
DELIMITER //
CREATE PROCEDURE GetMaxFareForDriver(IN driverId INT)
BEGIN
    SELECT MAX(FARE) AS MaxFare
    FROM ride
    WHERE DRIVER_ID = driverId;
END //
DELIMITER ;
```

```
@app.route('/get_max_fare_for_driver', methods=['POST'])
def get_max_fare_for_driver():
    if request.method == 'POST':
        # Retrieve data from the form
        driver_id = request.form['driver_id']

        # Call the stored procedure
        query = text("CALL GetMaxFareForDriver(:driver_id)")
        result = db.session.execute(query, {'driver_id': driver_id}).fetchall()

        # Display results
        return render_template('get_max_fare_for_driver_result.html', result=result)

    return render_template('get_max_fare_for_driver.html')
```

Result Grid | Filter Row

| DRIVER_ID | MaxFare |
|---|---|
| D001 | 120 |
| D002 | 192 |
| D003 | 80 |
| D101 | 120 |

Also, one can see

| Driver ID | Max Fare |
|---|---|
| D001 | 120 |
| D002 | 192 |
| D003 | 80 |
| D101 | 120 |

**TRIGGER 1**

**on inserting a value to PAYMENT_METHOD column of customer table it should reflect in PAYMENT_METHOD column of payment table for a matching CUSTOMER_ID. In order to achieve this we have created a trigger.**

Snapshot of payment table before implementing the trigger:

| PAYMENT_ID | PAYMENT_METHOD | FARE | DATE_OF_RIDE | TIME_OF_RIDE | CUSTOMER_ID |
|---|---|---|---|---|---|
| R2033 | YourPaymentMethod | 120 | 2023-11-21 | 12:41:00 | C526 |

Trigger:

```
DELIMITER //
CREATE TRIGGER update_payment_method
AFTER INSERT ON customer
FOR EACH ROW
BEGIN
    -- Check if the inserted value is not NULL
    IF NEW.PAYMENT_METHOD IS NOT NULL THEN
        -- Check if the new payment method is a specific value
        IF NEW.PAYMENT_METHOD = 'ValueToChange' THEN
            -- Update the payment table with the desired payment method
            UPDATE payment
            SET PAYMENT_METHOD = 'YourPaymentMethod'
            WHERE CUSTOMER_ID = NEW.CUSTOMER_ID;
        END IF;
    END IF;
END;
//
DELIMITER ;
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 1 | 16:22:29 | use hey_taxii | 0 row(s) affected | 0.000 sec |
| 2 | 16:22:50 | CREATE TRIGGER update_payment_method AFTER INSERT ON customer FOR EACH ROW BEGIN -- Che... | 0 row(s) affected | 0.016 sec |
| 3 | 16:23:13 | select * from payment LIMIT 0, 1000 | 6 row(s) returned | 0.000 sec / 0.000 sec |

Successful creation of trigger:

2 16:22:50 CREATE TRIGGER update_payment_method AFTER INSERT ON customer FOR EACH ROW BEGIN -- Che... 0 row(s) affected 0.016 sec

Result of the trigger:

| PAYMENT_ID | PAYMENT_METHOD | FARE | DATE_OF_RIDE | TIME_OF_RIDE | CUSTOMER_ID |
|---|---|---|---|---|---|
| R2033 | Cash | 120 | 2023-11-21 | 12:41:00 | C526 |

**TRIGGER 2:**

**Created a trigger to Setting up the timestamp after users table gets a new value.**

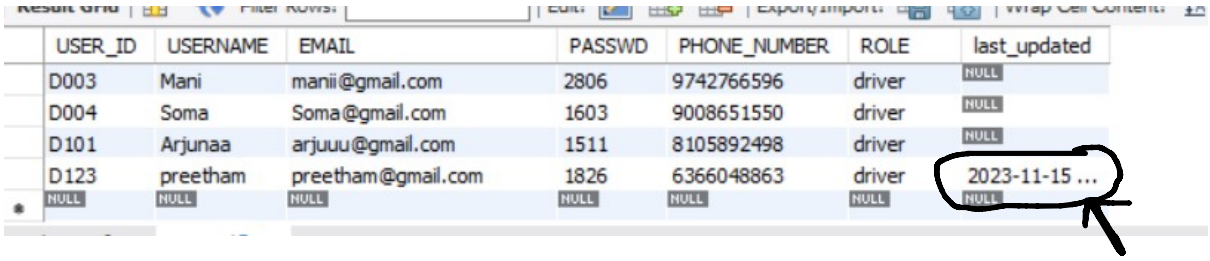
Before adding the trigger:

| USER_ID | USERNAME | EMAIL | PASSWD | PHONE_NUMBER | ROLE | last_updated |
|---|---|---|---|---|---|---|
| 1002 | SAMPU | sampi@gmail.com | sam[ | 9090898989 | customer | NULL |
| C481 | Rashmi | rashmipr9496@gmail.com | 1902 | 9380900636 | admin | NULL |
| C526 | Sameeksha | samijs2003@gmail.com | 2502 | 7899850454 | customer | NULL |
| C527 | Sampreethi | ysampreethi26@gmail.com | 2610 | 6366052151 | admin | NULL |
| C531 | Sanjana | sanjanabt2003@gmail.com | 2309 | 9141071392 | customer | NULL |

TRIGGER:

```
ALTER TABLE users
ADD COLUMN last_updated TIMESTAMP;

-- Step 2: Create the trigger
DELIMITER //
CREATE TRIGGER before_insert_trigger
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
    SET NEW.last_updated = NOW();
END;
//
DELIMITER ;
```

when you insert a new row into your table, the last_updated column will automatically be set to the current timestamp due to the trigger



| USER_ID | USERNAME | EMAIL | PASSWD | PHONE_NUMBER | ROLE | last_updated |
|---|---|---|---|---|---|---|
| D003 | Mani | manii@gmail.com | 2806 | 9742766596 | driver | NULL |
| D004 | Soma | Soma@gmail.com | 1603 | 9008651550 | driver | NULL |
| D101 | Arjunaa | arjuuu@gmail.com | 1511 | 8105892498 | driver | NULL |
| D123 | preetham | preetham@gmail.com | 1826 | 6366048863 | driver | 2023-11-15 ... |
| NULL | NULL | NULL | NULL | NULL | NULL | NULL |

GITHUB repo Link

**https://github.com/SAMPREETHI26/hey_taxi_dbms.git**