Birla Institute of Technology and Science

CS F212

Database Systems

CAB BOOKING MANAGEMENT SYSTEM DOCUMENTATION

Submitted to: Dr. Amit Dua & Parth Patel

Submitted by:

TRAYAMBAK SHRIVASTAVA (2021A7PS1629P)

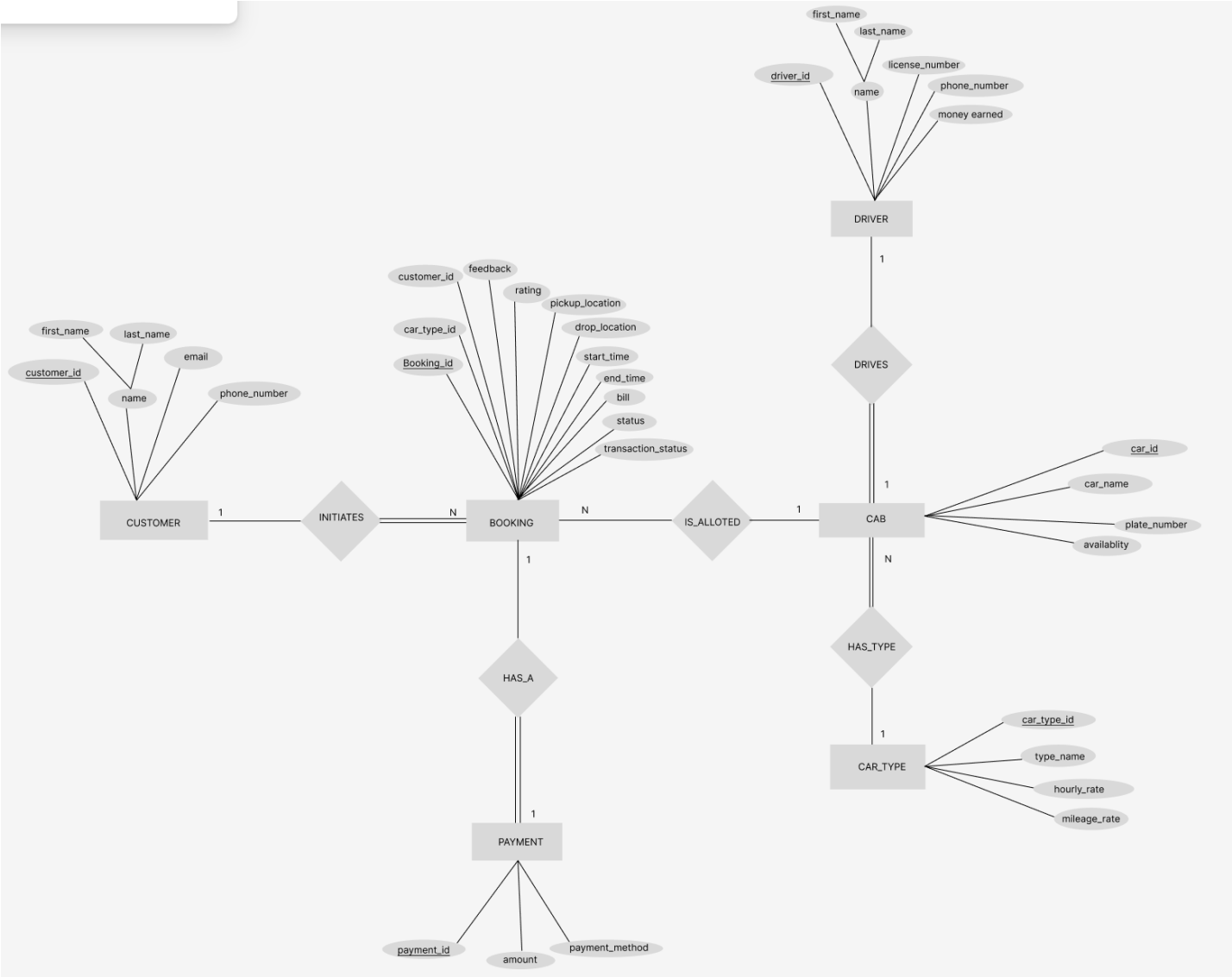
NEK MANCHANDA (2021A7PS0576P)

INTRODUCTION -

This project aims to create a database system with the objective of Cab Management And Booking Services. The schema has several tables. There are various functionalities like booking cabs , managing waiting lists , managing cab drivers , generating customer bills, managing customer data and many more. These functions can be directly performed by executing the SQL queries mainly and for some functionalities the interface is convenient .

The link to ER Diagram are herein attached :
https://www.figma.com/file/x998Mf8adDHXRfqYQ0OCws/DBS-PROJECT-12?node-id=0-1&t=sKsPBgbZU7WxAR8c-0

The snapshots have also been included in this documentation as shown below.



ENTITIES with Attributes -

1. **Booking** - booking_id(primary key ; set to auto-increment) , car_type_id(varchar , this is used to take input for desired type of cab demanded by user , status(set to waiting by default and changed as needed) , bill and transaction_status and other varchar attributes specifying travel details. Booking keeps track of the bookings made by various customers. It is related to other entities like customer , cab and payment.

2. **Driver** - driver_id(int , primary key , auto increment) , driver_name(composite attribute consisting of first and last names) , money_earned(int , updates with every completed ride to increase the amount earned by them)

3. **Car Type** - car_type_id(int , primary key) , this entity maintains a record of types of cabs offered by our vendor(eg. Hatchback or sedan etc.) . Other attributes include mileage and hourly rate for that cab type.

4. **Customer** - customer_id(int , primary key , auto increment) , customer name(composite attribute consisting of firstname and lastname(varchar) along with other attributes like email and phone number.

5. **Cab** - car_id(int , primary key ,auto increment) , this entity maintains a record of all cabs under operation along with driver_id(primary key in Driver entity and foreign key to this table) who drives this cab. Car_name , plate_number(varchar) and current availability(boolean , not null) describing whether at that particular instant that cab is available to take a fresh ride.

6. **Payment** - payment_id(int , primary key , auto increment) , booking_id(primary key in Booking foreign key here) , amount(int) to keep track of the booking for which the payment has been made. Along with this there are attributes containing payment details such as method of payment(eg upi , credit card etc)

RELATIONSHIPS -

1. **INITIATES** : Relationship between customer and booking table. Customer to Booking has a 1:N cardinality indicating that one customer can participate in this relationship with many bookings but each booking can have only one customer. This relation has total participation from Booking side as each customer (registered with us) need not have booked a cab in the past but each booking will have a customer who initiated it.

2. **HAS_A** : relationship between booking table and payment table. Booking to Payment (1: 1 Cardinality) as each booking has only one payment and each payment made will be corresponding to a certain booking. There is total participation from the Payment table as at any given point in time for a payment to exist, there has to be a booking hence but there can be some bookings which are “Ongoing” and their payment will be made after the ride ends but for a payment to exist, there has to be a booking hence

3. **IS_ALLOTED** : Relationship between cab and booking table. Cab to Booking (1:N Cardinality) as each cab may be allotted to multiple bookings but each booking can have at maximum only one cab. Participation is partial from both ends as there can be cabs that have never done a ride and there can be bookings currently at “Waiting” status that have no cabs assigned to them

4. **DRIVES** : Cab to driver (1: 1 Cardinality) as each driver has an assigned cab and only he can drive it. There is total participation from both tables as each driver has to register himself with a cab only.

5. **HAS_TYPE** : Cab to Cab_Type (N : 1 Cardinality) as there may be multiple hatchbacks or sedans. There is total participation from cab table as each cab must have a type registered with the company while there may be cab types for eg. EV that have no cabs registered.

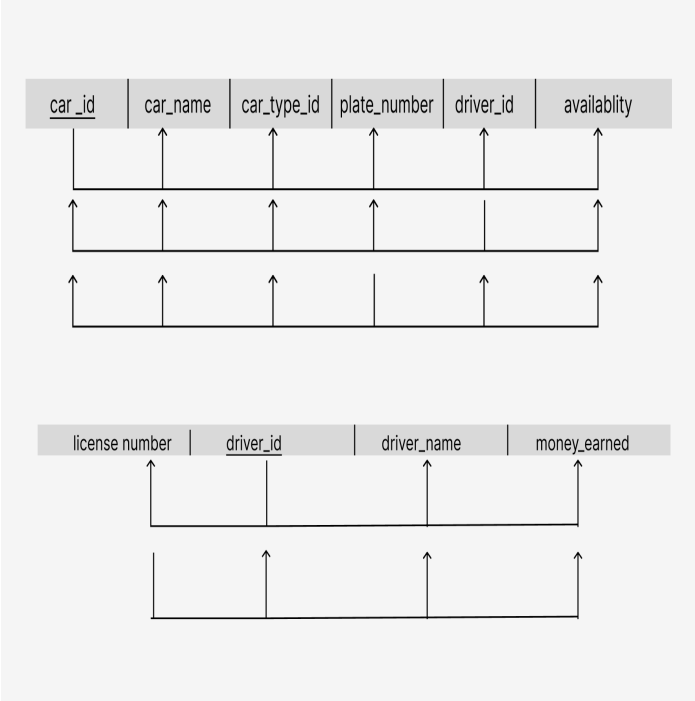
FOREIGN KEYS IN TABLES :

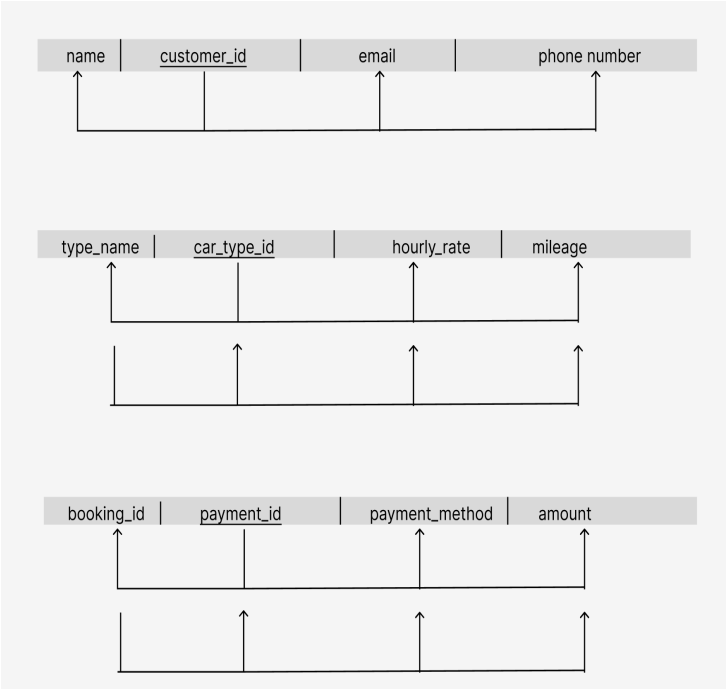
1. Booking - FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
   FOREIGN KEY (car_id) REFERENCES CAB(car_id)

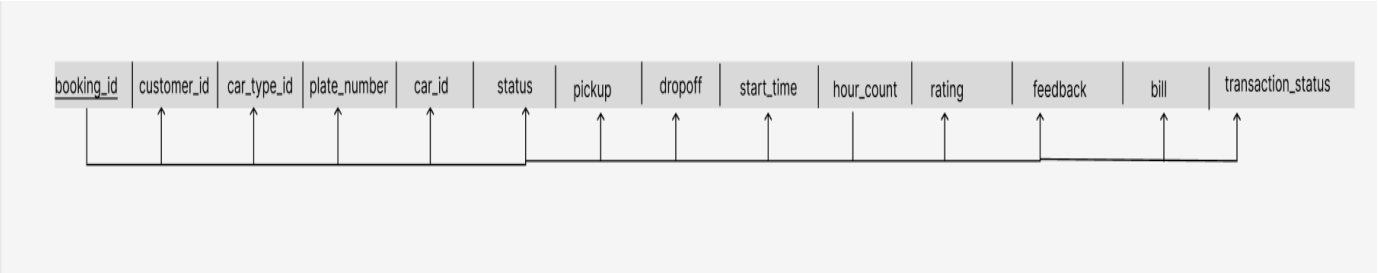
2. Driver - NONE
3. Car Type - NONE
4. Cab - FOREIGN KEY (car_type_id) REFERENCES CAR_TYPE(car_type_id)
   FOREIGN KEY (driver_id) REFERENCES DRIVER(driver_id)

5. Payment - FOREIGN KEY (booking_id) REFERENCES BOOKING(booking_id)
6. Customer - NONE

FUNCTIONAL DEPENDENCIES (3NF FORM) :







NORMALISATIONS :

1. 1NF :

   There are no composite/multi-valued attributes except for the Names of Drivers and Customers .
   Name - first name(varchar) and last name(varchar)
   Solution : Create a new table
   With driver_id/customer_id as foreign key
   As shown in the code snippet next .

```
CREATE TABLE NAME(
    driver_id INT PRIMARY KEY ,
    driver_firstname VARCHAR(20) ,
    driver_lastname VARCHAR(20) ,
    FOREIGN KEY (driver_id) REFERENCES DRIVER(driver_id)
```

But a logically more appealing solution seemed to be merging the name to just one attribute with varchar(50) and hence that is the approach chosen.

Hence now our schema satisfies 1NF.

2. 2NF :

All the candidate keys in our tables consist of only one attribute. Hence there is *no scope of partial dependency*. There is no functional dependency that is determined by an attribute which is not the primary key, but is a part of the primary key.
Thus, the schema satisfies the 2NF also.

3. 3NF :

There are no transitive dependencies that apply to non-prime applicants. Only super keys determine the candidate keys directly and hence we can say that the 3NF Condition is satisfied.

QUERIES :

1. **Populating the table**

```
-------    WRTING QUERIES    -----------


- Query1 / SIMPLE POPULATION QUERY /TRAYAMBAK
NSERT INTO DRIVER (driver_name,  license_number , phone_number)
ALUES ('Cardinal Erichsen',  'RAJ001' , '65566'),
      ('Nek Man',  'CDG001' , '64566'),
      ('Om Kota',  'RAJ002' , '65266'),
      ('Naman Shethia',  'CHS001' , '68866'),
      ('Sidd Sharma',  'CHS002' , '63336'),
      ('Anurag Bitsh',  'CHS003' , '64446'),
      ('Nach Nalin',  'BR001' , '69996'),
      ('Vikram Bhata',  'DL001' , '65556'),
      ('Tram Shrivatsava', 'BR002' , '60006') ,
      ('Dev Parekh','DL002','69696');


- Query2 / Nek
nsert into CUSTOMER(name,email,phone_number)
ALUES  ('prabhnoor singh','prabhnoor02@gmail.com','9088443275'),
       ('nakul rana','nakul@gmail.com','9955885555'),
       ('isha joshi','isha@gmail.com','9955251222'),
       ('aryan kumar','aryan@gmail.com','9955412111'),
       ('shobhit rathi','shobhit@gmail.com','9955223522'),
       ('pratham bera','pratham@gmail.com','9955321200'),
       ('jatin gupta','jatin@gmail.com','9955824200'),
       ('devansh sharma','devansh@gmail.com','9955220077'),
       ('rahul kumar','rahul@gmail.com','9955145155'),
       ('mohan pandey','mohan@gmail.com','9955647135'),
       ('Ravishankar' ,'ravi.shankar@gmail.com' , '9696966666'),
       ('Vaibhav Singla' , 'vaibha.singla@gmail.com' , '6969696999');
```

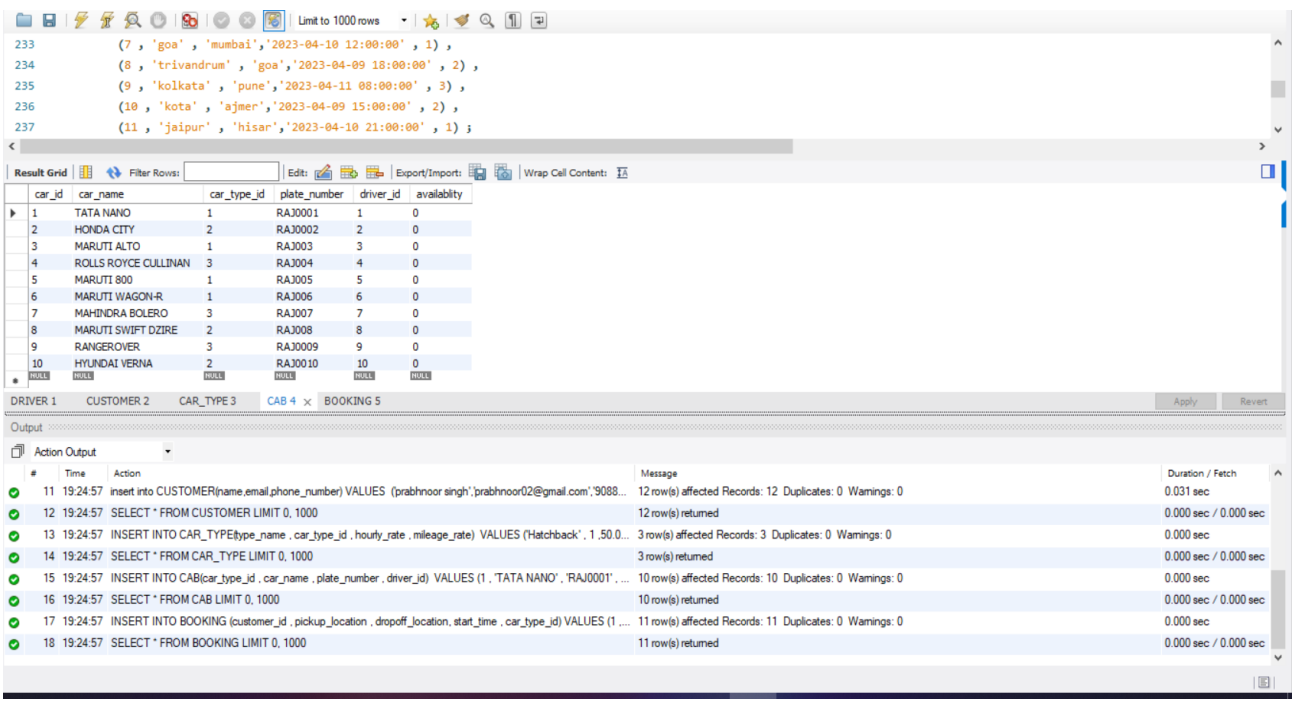
These queries are only for the purpose of populating data.

2. Same as above. Refer figure above

3. **Insertion Of A Cab**

```
-- Query3 /Populating the Cab/ TRAYAMBAK
INSERT INTO CAR_TYPE(type_name , car_type_id , hourly_rate , mileage_rate)
VALUES ('Hatchback' , 1 ,50.00 , 15.50) ,
       ('Sedan' , 2 , 75.00 , 18.75) ,
       ('SUV' , 3 , 100 , 20.50) ;

INSERT INTO CAB(car_type_id , car_name , plate_number , driver_id)
VALUES (1 , 'TATA NANO' , 'RAJ0001' , 1) ,
       (2 , 'HONDA CITY' , 'RAJ0002' , 2) ,
       (1 , 'MARUTI ALTO' , 'RAJ003' , 3) ,
       (3 , 'ROLLS ROYCE CULLINAN' , 'RAJ004' , 4) ,
       (1 , 'MARUTI 800' , 'RAJ005' , 5) ,
       (1 , 'MARUTI WAGON-R' , 'RAJ006' , 6) ,
       (3 , 'MAHINDRA BOLERO' , 'RAJ007' , 7) ,
       (2 , 'MARUTI SWIFT DZIRE' , 'RAJ008' , 8) ,
       (3 , 'RANGEROVER' , 'RAJ0009' , 9) ,
       (2 , 'HYUNDAI VERNA' , 'RAJ0010' , 10) ;
```



These queries are only for the purpose of populating data. A lot of data is already populated.

4. Updating Cab Availability
   The query was processed with a given car_id value availability of which we want to modify at random.

```
-- Query4

UPDATE CAB
SET availablity=TRUE
WHERE driver_id = 1;
select * from CAB;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

car_id	car_name	car_type_id	plate_number	driver_id	availablity
1	TATA NANO	1	RAJ0001	1	1
2	HONDA CITY	2	RAJ0002	2	0
3	MARUTI ALTO	1	RAJ003	3	0
4	ROLLS ROYCE CULLINAN	3	RAJ004	4	0
5	MARUTI 800	1	RAJ005	5	0
6	MARUTI WAGON-R	1	RAJ006	6	0
7	MAHINDRA BOLERO	3	RAJ007	7	0

The availability of the first cab is changed.

5. Updating user records (phone no/email id) using primary key customer_id

```
-- where cab.availablity=true and booking.status='Waiting'


-- Query5 / Trayambak
UPDATE CUSTOMER
SET email = 'prabhnoor01@gmail.com' , phone_number = '9088443276'
WHERE customer_id = 1 ;
```

```sql
-- Query5

UPDATE CUSTOMER
SET email = 'prabhnoor.singh@gmail.com' , phone_number = '1118443276'
WHERE customer_id = 1 ;
--
select * from customer;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

customer_id	name	email	phone_number
1	prabhnoor singh	prabhnoor.singh@gmail.com	1118443276
2	nakul rana	nakul@gmail.com	9955885555
3	isha joshi	isha@gmail.com	9955251222
4	aryan kumar	aryan@gmail.com	9955412111
5	shobhit rathi	shobhit@gmail.com	9955223522
6	pratham bera	pratham@gmail.com	9955321200
7	jatin gupta	jatin@gmail.com	9955824200

6. Associating a driver with a cab (and its type, cab no.)
   There is no need for such query in our database as there exists a foreign key relationship between the cab_id and driver_id of the tables cab and driver and hence this association is by default .

7. Cancel ride feature by deleting an entry from bookings

```sql
select * from booking;

-- Query7
DELETE from BOOKING
WHERE booking_id = 02 ;
-- cancels the ride for the booking id of 2

select * from booking;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

booking_id	car_type_id	customer_id	car_id	pickup_location	dropoff_location	start_time	hour_count	status	rating	feedback	bill	transaction_status
1	1	1	NULL	delhi	patna	2023-04-09 18:00:00	NULL	Waiting	NULL	NULL	NULL	N/A
3	3	3	NULL	chandigarh	patna	2023-04-10 08:00:00	NULL	Waiting	NULL	NULL	NULL	N/A
4	2	4	NULL	delhi	leh	2023-04-09 14:00:00	NULL	Waiting	NULL	NULL	NULL	N/A
5	1	5	NULL	leh	ladakh	2023-04-09 16:00:00	NULL	Waiting	NULL	NULL	NULL	N/A
6	3	6	NULL	mohali	bilaspur	2023-04-11 18:00:00	NULL	Waiting	NULL	NULL	NULL	N/A
7	1	7	NULL	goa	mumbai	2023-04-10 12:00:00	NULL	Waiting	NULL	NULL	NULL	N/A
8	2	8	NULL	trivandrum	goa	2023-04-09 18:00:00	NULL	Waiting	NULL	NULL	NULL	N/A

8. Generating feedback and bill
   This SQL query updates feedback and rating in the booking table.

```sql

-- Query8
UPDATE BOOKING
inner join CAR_TYPE on booking.car_type_id = CAR_TYPE.car_type_id
SET feedback = 'Extremely Good' , rating = 7, bill = 5 * CAR_TYPE.hourly_rate
WHERE booking_id = 01 ;
-- sets feeback, rating and generates a bill for the booking with id 1
select * from booking;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

9. Creating a trigger of adding surge charge on bill amount
   This SQL query is a trigger that increases the bill amount upon surge in demand.

```
-- Query 9
DELIMITER %%
CREATE TRIGGER apply_surge_charge
before insert ON booking
FOR EACH ROW
BEGIN
    set new.bill = new.bill + 50 ;
END %%
DELIMITER;

INSERT INTO BOOKING (customer_id , pickup_location , dropoff_location, start_time , car_type_id)
VALUES (1 ,'delhi' , 'patna','2023-04-09 18:00:00' , 1) ;
select * from booking;
-- creates a trigger introducing a surge charge in the cab fares by 50rs. for subsequent bookings, as demonstrated
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

booking_id	car_type_id	customer_id	car_id	pickup_location	dropoff_location	start_time	hour_count	status	rating	feedback	bill	transaction_status
9	3	9	NULL	kolkata	pune	2023-04-11 08:00:00	NULL	Waiting	NULL	NULL	0	N/A
10	2	10	NULL	kota	ajmer	2023-04-09 15:00:00	NULL	Waiting	NULL	NULL	0	N/A
11	1	11	NULL	jaipur	hisar	2023-04-10 21:00:00	NULL	Waiting	NULL	NULL	0	N/A
12	1	1	NULL	delhi	patna	2023-04-09 18:00:00	NULL	Waiting	NULL	NULL	50	N/A
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

10. Creating procedures/functions providing admin features including:
    Returns all available cabs of a certain type

```
-- Query 10
drop procedure cars_Available;
DELIMITER &&
CREATE PROCEDURE cars_available(id int)
BEGIN
    select * from CAR_TYPE
    inner join CAB
    ON CAR_TYPE.car_type_id = CAB.car_type_id
    WHERE CAR_TYPE.car_type_id = id AND (SELECT availablity FROM DRIVER WHERE DRIVER.driver_id = CAB.driver_id)=true;
END &&
DELIMITER ;
CAll cars_available(1);
-- admin procedure to display all the cars available of a given type (here we have car_type_id=1)
```

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

car_type_id	type_name	hourly_rate	mileage_rate	car_id	car_name	car_type_id	plate_number	driver_id	availablity
1	Hatchback	50.00	15.50	1	TATA NANO	1	RAJ0001	1	1
1	Hatchback	50.00	15.50	5	MARUTI 800	1	RAJ005	5	1

11. Creating procedures/functions providing admin features including: Total amount generated by specific driver

```
-- Query 11

drop procedure money_Earned_by_Driver;
DELIMITER &&
CREATE PROCEDURE money_earned_by_driver(id INT)
BEGIN
    -- SELECT SUM(BILL) AS money_earned FROM BOOKING
--      join CAB ON BOOKING.car_id = CAB.car_id
--      WHERE cab.driver_id=id;
select money_earned from driver
where driver.driver_id = id;
END &&
DELIMITER ;
call money_earned_by_driver(1);
```

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

money_earned
250

12. Creating procedures/functions providing admin features including:Total amount generated by specific category

```
-- Query 12
DELIMITER &&
CREATE PROCEDURE money_earned_by_car_type(id INT)
BEGIN
    SELECT SUM(BILL) AS money_earned FROM BOOKING
    WHERE BOOKING.car_type_id = id;
END &&
DELIMITER ;
call money_earned_by_car_type(1);

-- admin procedure that displays the money earned by a given cab type =, here 1, net money is 250 + 50 from surge fee
```

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

money_earned
300

13. Creating procedures/functions providing admin features including : View all customers' database

```
-- Query 13
DELIMITER &&
CREATE PROCEDURE view_customer_database()
BEGIN
    select * from CUSTOMER;
END &&
DELIMITER ;
call view_customer_database();
-- admin procedure that displays the whole database of the customers accessing the cab services
```

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

customer_id	c_name	email	phone_number
1	prabhnoor singh	prabhnoor.singh@gmail.com	1118443276
2	nakul rana	nakul@gmail.com	9955885555
3	isha joshi	isha@gmail.com	9955251222
4	aryan kumar	aryan@gmail.com	9955412111
5	shobhit rathi	shobhit@gmail.com	9955223522
6	pratham bera	pratham@gmail.com	9955321200
7	jatin gupta	jatin@gmail.com	9955824200
8	devansh sharma	devansh@gmail.com	9955220077
9	rahul kumar	rahul@gmail.com	9955145155
10	mohan pandey	mohan@gmail.com	9955647135
11	Ravishankar	ravi.shankar@gmail.com	9696966666

14. Creating procedures/functions providing admin features including :

```
-- Query 14
DELIMITER &&
CREATE PROCEDURE view_driver_and_cabs_database()
BEGIN
    select * from DRIVER
    join CAB on DRIVER.driver_id = CAB.driver_id;
END &&
DELIMITER ;
CALL view_driver_and_cabs_database;
-- admin procedure that displays the join of the drivers and the cabs they drive, through a common attribute of the driver id
```

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

driver_id	driver_name	license_number	phone_number	money_earned	car_id	car_name	car_type_id	plate_number	driver_id	availablity
1	Cardinal Erichsen	RAJ001	65566	250	1	TATA NANO	1	RAJ0001	1	1
2	Nek Man	CDG001	64566	0	2	HONDA CITY	2	RAJ0002	2	0
3	Om Kota	RAJ002	65266	0	3	MARUTI ALTO	1	RAJ003	3	0
4	Naman Shethia	CHS001	68866	0	4	ROLLS ROYCE CULLINAN	3	RAJ004	4	0
5	Sidd Sharma	CHS002	63336	0	5	MARUTI 800	1	RAJ005	5	1
6	Anurag Bitsh	CHS003	64446	0	6	MARUTI WAGON-R	1	RAJ006	6	0
7	Nach Nalin	BR001	69996	0	7	MAHINDRA BOLERO	3	RAJ007	7	0
8	Vikram Bhata	DL001	65556	0	8	MARUTI SWIFT DZIRE	2	RAJ008	8	0
9	Tram Shrivatsava	BR002	60006	0	9	RANGEROVER	3	RAJ0009	9	0
10	Dev Parekh	DL002	69696	0	10	HYUNDAI VERNA	2	RAJ0010	10	0

15. Creating a cab allotment i.e. allotting available cabs to waiting customers
(**EXTRA QUERY IMPLEMENTED FOR ADDITIONAL FUNCTIONALITY)**

```
DELIMITER &&
CREATE PROCEDURE ALLOT_CABS()
BEGIN
    update booking join
    (SELECT * FROM CAB WHERE car_id = 7)sub1
    INNER JOIN (SELECT * FROM BOOKING WHERE booking.car_type_id = (SELECT cab.car_type_id FROM CAB WHERE car_id = 7) and BOOKING.status = "Waiting" LIMIT 1 )sub2
    ON sub1.car_type_id = sub2.car_type_id
    set booking.status="Ongoing", booking.car_id=7
    where booking.car_type_id = (SELECT cab.car_type_id FROM CAB WHERE car_id = 7) limit 1;
END &&
DELIMITER ;

call ALLOT_CABS();
select * from booking;

-- procedure which allots a given cab, here cab_id = 7, to a customer with the car type requested same as the cab with id 7, provided that the current booking status is waiting,
-- the cab is alloted and the status of booking is changed to ongoing
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

booking_id	car_type_id	customer_id	car_id	pickup_location	dropoff_location	start_time	hour_count	status	rating	feedback	bill	transaction_status
1	1	1	NULL	delhi	patna	2023-04-09 18:00:00	NULL	Waiting	7	Extremely Good	250	N/A
3	3	3	7	chandigarh	patna	2023-04-10 08:00:00	NULL	Ongoing	NULL	NULL	0	N/A
4	2	4	NULL	delhi	leh	2023-04-09 14:00:00	NULL	Waiting	NULL	NULL	0	N/A
5	1	5	NULL	leh	ladakh	2023-04-09 16:00:00	NULL	Waiting	NULL	NULL	0	N/A
6	3	6	NULL	mohali	bilaspur	2023-04-11 18:00:00	NULL	Waiting	NULL	NULL	0	N/A

TERMINAL FUNCTIONALITIES :

1. Implemented in Python
2. Use of mycursor() functions and mydb() via mysql connector
3. Backend allows terminal running for certain customer and Driver operations such as booking , new customer registration , etc .

USER :
FUNCTIONS –
1. New customer registration
2. Previous Bookings
3. New Booking
4. Status of last booking
5. Generate booking

CAB :
FUNCTIONS –
1. Finish a ride
2. Set your cab is available

s proj > base.py > ...

```python
import mysql.connector
from datetime import datetime

now = datetime.now()

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    passwd="Mysqlnek",
    database="dbs_proj"
)

mycursor = mydb.cursor()
```

```python
if user == 1:
    print("Hello Customer \n For new customer registration, Press 1 \n For previous bookings, Press 2 \n For new booking, Press 3 \n Fo
    use = int(input())



    if use == 1:
        print("Please enter name")
        name = input()

        print("Please enter email")
        email = input()
        print("Please enter phone number")
        mobile = input()

        name_formula = "INSERT into CUSTOMER(name, email, phone_number) VALUES( %s , %s , %s)"
        mycursor.execute(name_formula, ( name, email, mobile))
        mydb.commit()

        show = 'SELECT customer_id FROM CUSTOMER ORDER BY customer_id DESC LIMIT 1'
        print('Your Customer ID ')


        mycursor.execute(show)
        print(mycursor.fetchall())



    elif use == 2:

        print("Please enter Customer ID")
        c_id = int(input())
        opt_formula = 'SELECT * FROM BOOKING WHERE customer_id =' + \
            str(c_id) + ';'
        mycursor.execute(opt_formula)
        print(mycursor.fetchall())
```

FRONTEND :

Tech stack used:

1. React.JS used along with chakra UI components for frontend implementation.
2. Flask framework of Python used for backend integration.

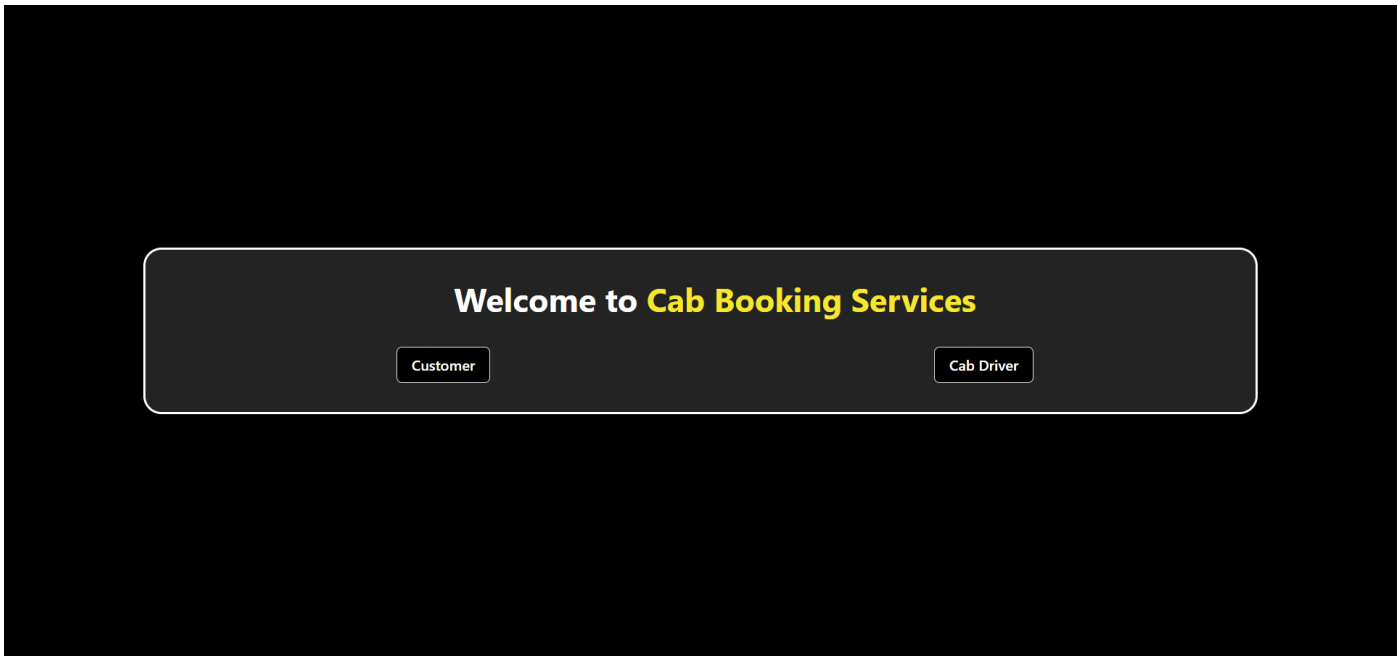
Steps:

1. Install Node and pnpm. For the installation, the steps can be followed from here: https://phoenixnap.com/kb/install-node-js-npm-on-windows and https://pnpm.io/installation
2. Download the zip file and extract the zip folder.
3. Open the terminal inside the zip folder.
4. Open the *DBS Project* folder in the terminal.
5. Run the following commands:
   (a) cd dbs-front
   (b) pnpm i
   (c) pnpm run dev
   Then open the url that is shown in the terminal.
6. Open a new terminal (parallelly) in the same folder.
7. Run the following commands:
   (a) cd server
   (b) pip install flask
   (c) python server.js (python3 server.js)

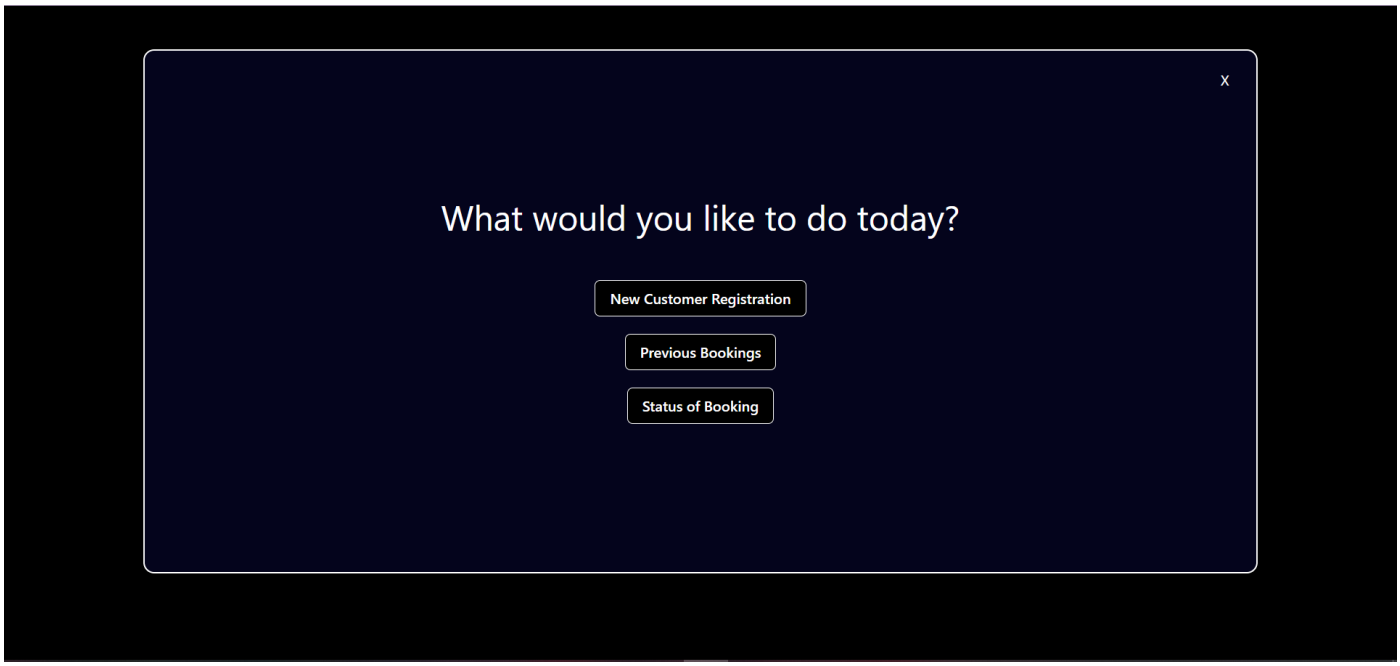
8. In case the request gets blocked due to an unknown CORS origin, use the extension ”CORS Unblock” to run the project on localhost.

Snapshots of the frontend interface :

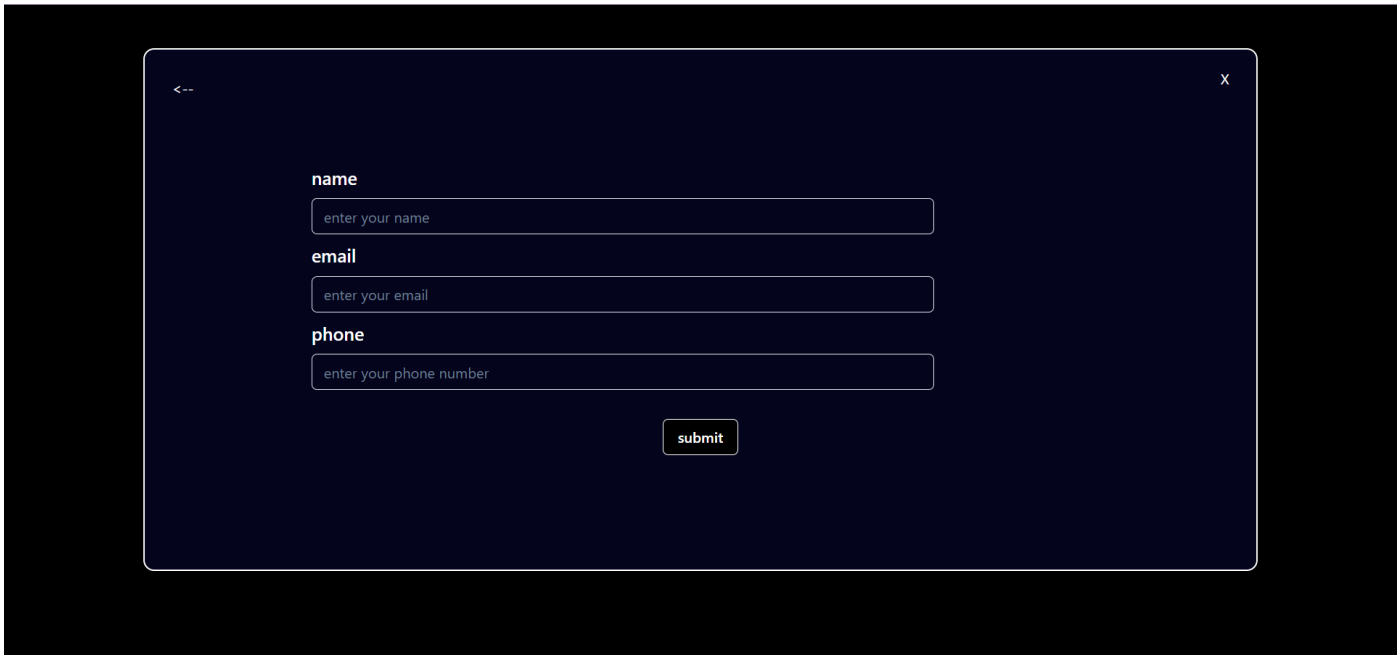
1. Home Page



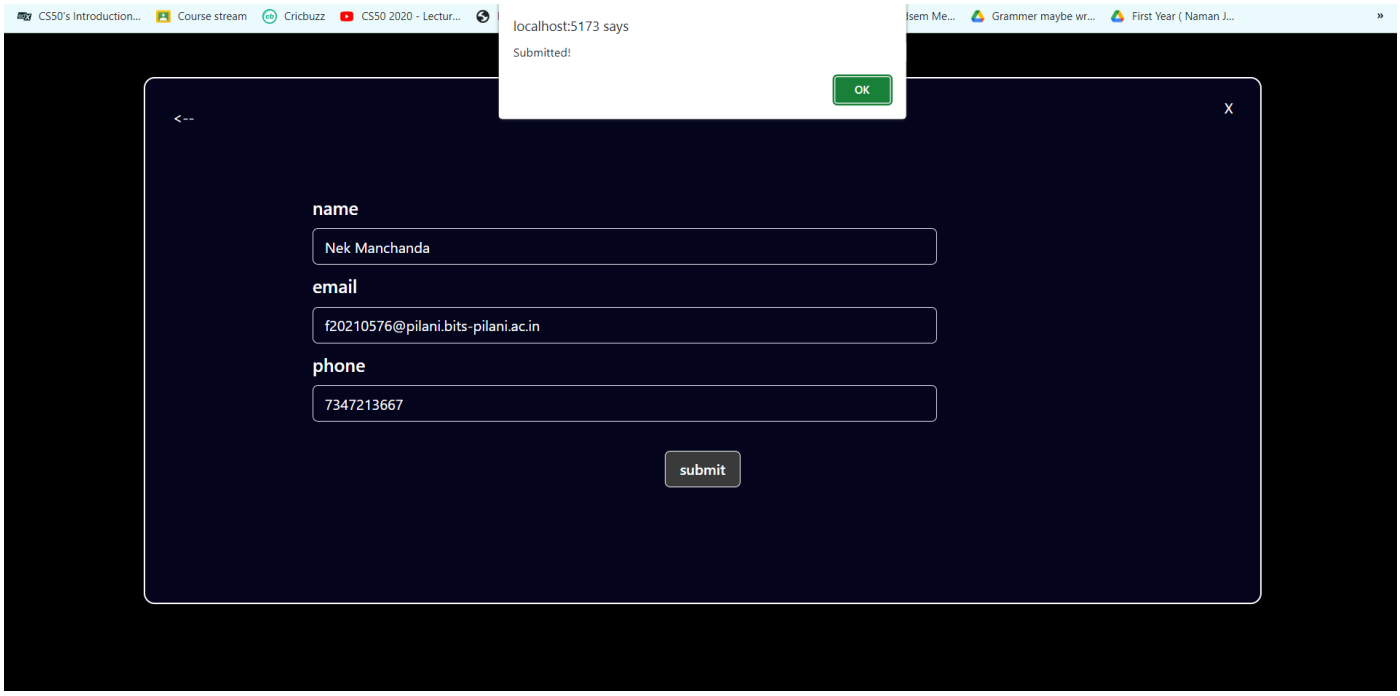
2. Customer Menu



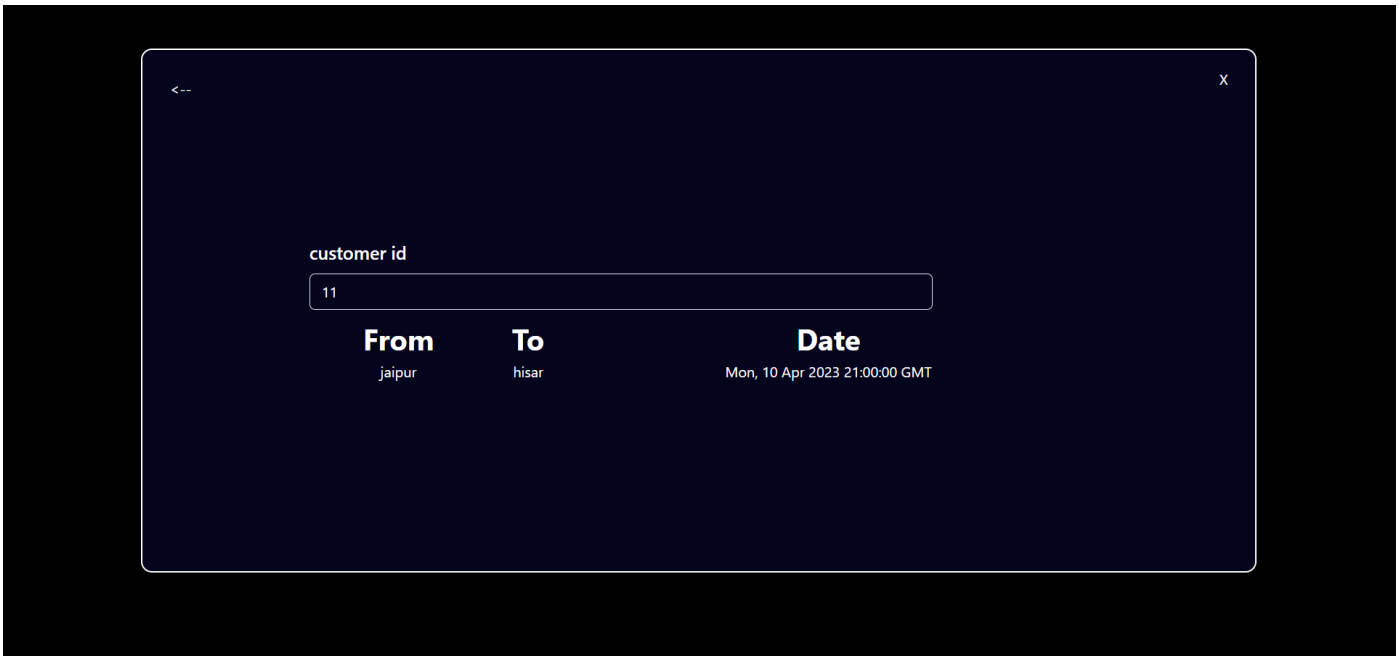
3. Interface for registration of new customer



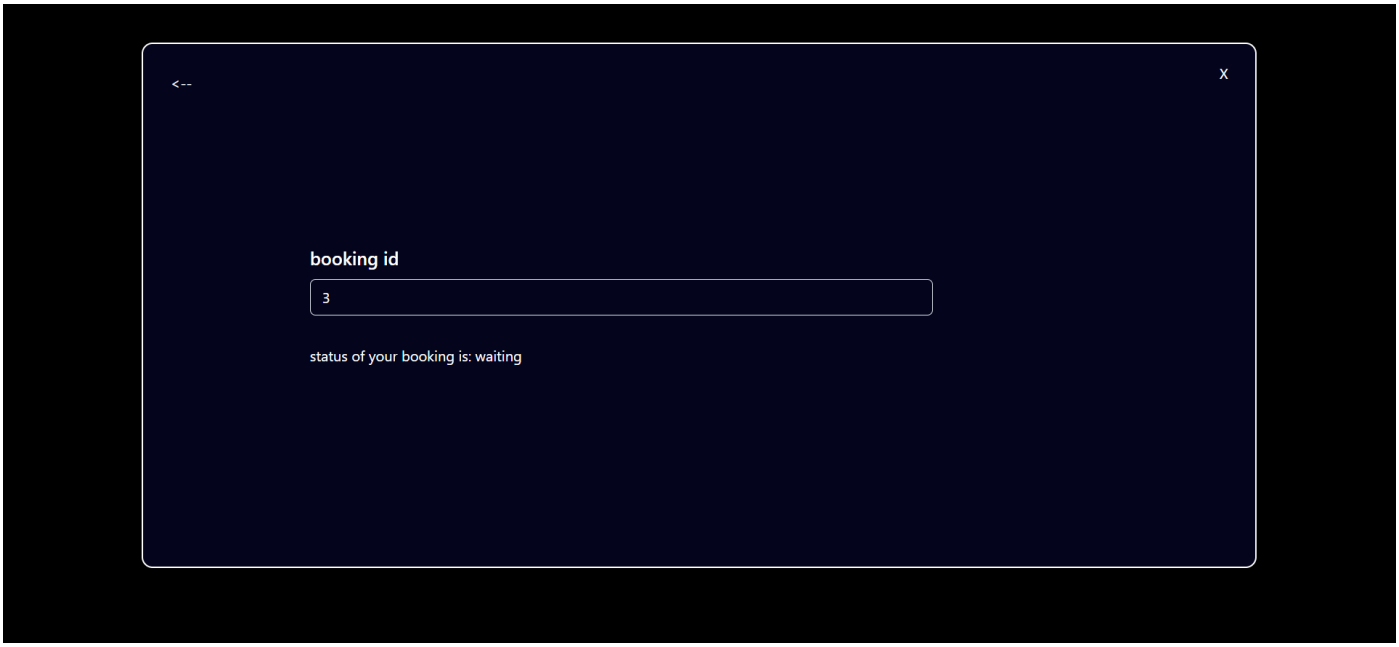
4. Alert Box on successful data entry into the consumer database



5. Shows the list of all bookings done by the customer to-date.



6. Interface of Status for Booking



Project Demonstration Video - Nek Manchanda 2021A7PS0576P :
https://drive.google.com/drive/folders/1xEQwfwHF6yV4AKSqtVZ9FUesdLI6bfZc