UE21CS351A: Database Management System

Car Rental System

Mini Project Report

By,
Harikrishnan G (PES1UG21CS219)
Hitesh Singh (PES1UG21CS233)

Table of Contents:

1. Introduction

A: Need for Car Rental System:
Car Rental services offer cars for people to rent. It is more convenient than buying a car. A car rental company lends vehicles for a price for a stipulated time frame, at a pre-determined rate. Managing the various aspects of the cars, customers and their rentals can become a taxing task and hence a database system would come in handy.

B: Purpose:
The project's goal is to automate car rental and reservation so that customers do not have to waste time calling and waiting for a vehicle. The project aims to develop a system which can save both, time and effort of the user as well as the employees during the procedure of renting a car.

C: Scope:
This project will provide a user friendly yet powerful way of facilitating the car rentals as well as storing data such as past transactions, customer details, employee details and payments.

2. Project Description

A: Overview:
This project aims to develop a web application for car rental management using nextJS.

B: Major Project Functionalities
Once a customer provides their details and creates an account, they can login to the service to view and book cars for rental.
They can choose a car from a branch of their choice and make a booking.
An employee of the branch will be available to contact and brief them about the process and the vehicle.
Payment can be facilitated through the application.
Admins can log into the service to run sales diagnostics across branches.

3. System Features and Function Requirements

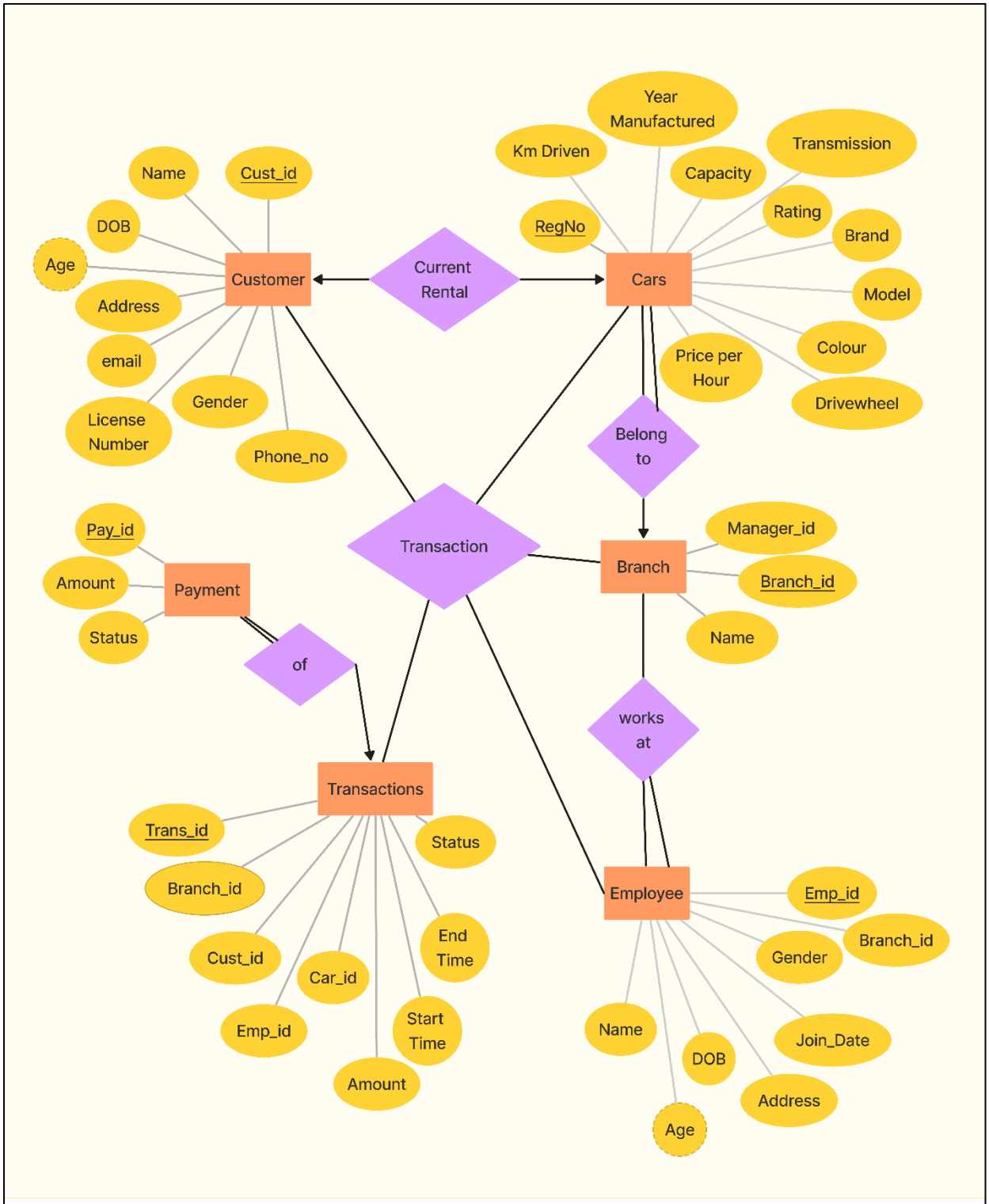
SF-1: Login/SignUp:
Customers need to provide their personal details and signUp with the platform to be able to rent cars.
Employees and the Admin can login using their predefined credentials.
Functional Requirements include a username and password for each customer and employee.

SF-2: Browse Cars:
Customers can choose which branch they wish to rent from and view a list of cars available at the branch as well as their specifications.
Functional Requirements include displaying of only those cars that are currently available to rent.

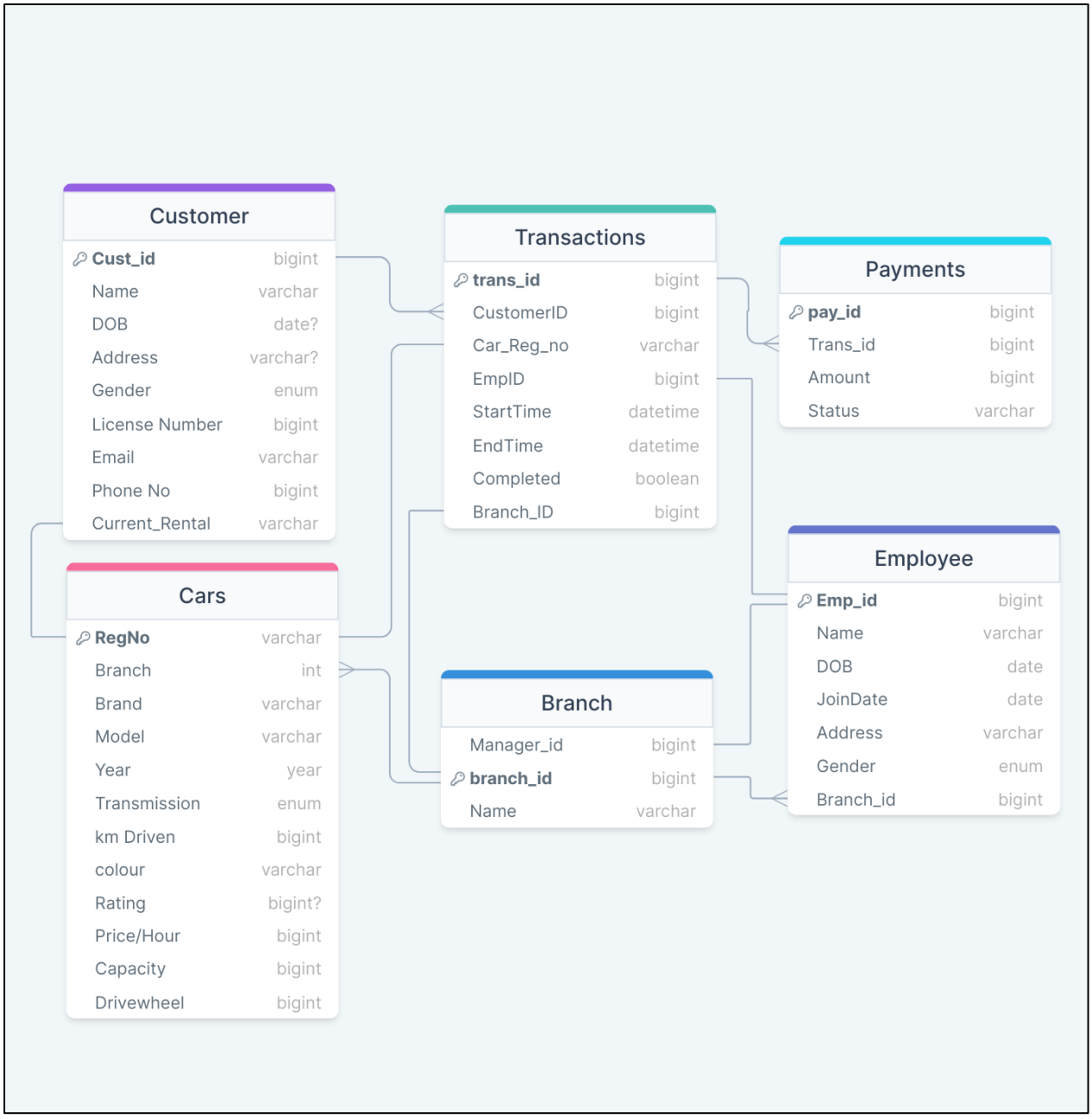
SF-3: Renting a Car:
Customers can make a booking for a car and make a payment for it through the service. Once a booking is confirmed, an employee will be assigned for assistance. Functional Requirements include assigning an employee for the rental and marking the car as reserved in the database so that it does not appear available to other customers.

SF-4: Admin Analytics:
Admins can view consolidated information such as demand for a particular brand and branch wise revenue.

4. ER Diagram


Customer
Name
Cust_id
DOB
Age
Address
email
License Number
Gender
Phone_no
Current Rental
Cars
Year Manufactured
Km Driven
Transmission
Capacity
Rating
RegNo
Brand
Model
Colour
Price per Hour
Drivewheel
Belong to
Transaction
Branch
Manager_id
Branch_id
Name
Payment
Pay_id
Amount
Status
of
works at
Transactions
Trans_id
Status
Branch_id
Cust_id
Car_id
End Time
Emp_id
Start Time
Amount
Employee
Emp_id
Branch_id
Gender
Name
DOB
Join_Date
Address
Age

5. Relational Shema



6. DDL SQL Commands

The following commands can be run to create the tables and users required for the application.

```
--creating database:

CREATE DATABASE car_rental;
use car_rental;

--Creating tables:

create table customer (
    cust_id bigint PRIMARY KEY auto_increment,
    name varchar(30) not null,
    DOB date not null,
    address varchar(80) not null,
    email varchar(30) not null,
    license_no varchar(20) not null,
    gender enum('Male','Female') not null,
    phone_no bigint not null,
    password varchar(20) not null

);

create table cars(
    reg_no varchar(10) PRIMARY KEY,
    km_driven int,
    year_manf YEAR not null,
    branch_id bigint not null,
    capacity int ,
    transmission enum('Manual','Automatic'),
    rating int ,
    brand varchar(20) not null,
    model varchar(40) not null,
    colour varchar(20),
    drivewheel enum('FWD','RWD','AWD'),
    price_per_hour int not null,
    fuel enum('petrol','diesel','electric','CNG'),
    img varchar(40)

);

create table branch(
    manager_id BIGINT,
    branch_id BIGINT primary key,
    name varchar(40),
    address varchar(80)
```

```sql
);

create table Employee(
    emp_id bigint PRIMARY KEY auto_increment,
    name varchar(30) not null,
    dob date not null,
    JoinDate date not null,
    email varchar(30) not null,
    address varchar(80),
    Gender enum('Male','Female') not null,
    branch_id bigint,
    password varchar(20) not null
);

create table Transactions(
    trans_id bigint PRIMARY KEY auto_increment,
    cust_id bigint not null,
    car_reg_no varchar(10) not null,
    startTime DateTime ,
    endTime DateTime,
    Status enum('InProgress','Completed','Cancelled','Initiated'),
    amount decimal(10,2),
    branch_id bigint not null
);

create table Payment(
    pay_id bigint PRIMARY KEY,
    trans_id bigint not null,
    amount decimal(10,2),
    status enum('successful','unsuccessful')
);

-- Adding foreign key constraints:

ALTER TABLE cars
ADD FOREIGN KEY (branch_id) REFERENCES branch(branch_id);

ALTER TABLE branch
ADD FOREIGN KEY (manager_id) REFERENCES Employee(emp_id)
on delete set null;

ALTER TABLE Employee
ADD FOREIGN KEY (branch_id) REFERENCES branch(branch_id);

ALTER TABLE Transactions
ADD FOREIGN KEY (cust_id) REFERENCES customer(cust_id);

ALTER TABLE Transactions
```

```sql
ADD FOREIGN KEY (car_reg_no) REFERENCES cars(reg_no);

ALTER TABLE Transactions
ADD FOREIGN KEY (branch_id) REFERENCES branch(branch_id);

ALTER TABLE Payment
ADD FOREIGN KEY (trans_id) REFERENCES Transactions(trans_id);

--creating roles and thier priviliges:

create role 'Customer';
grant select, update on car_rental.customer to 'Customer';
grant select, update on car_rental.transactions to 'Customer';
grant select on car_rental.cars to 'Customer';

create role 'Employee';
grant select, update on car_rental.employee to 'Employee';
grant select, update, delete on car_rental.transactions to 'Employee';
grant select on car_rental.cars to 'Employee';

create role 'Manager';
grant select, update,delete on car_rental.employee to 'Manager';
grant select, update, delete on car_rental.transactions to 'Manager';
grant select on car_rental.cars to 'Manager';
```

7. CRUD Operations

The following are the queries that are executed through the various Api routes present in the application as well as to initialize the database.

Initialization queries:

```
--Branches:

insert into branch(branch_id, name, address)
values
(1001, 'Cox Town', '#302, 7th Street, Assaye Road, Cox Town, Bangalore'),
(1002, 'Ulsoor', '#411, 9th Street, Cambridge Road, Halasuru, Bangalore'),
(1003, 'Indiranagar', '#233, 2nd Main Road, Indiranagar, Bangalore'),
(1004, 'Majestic', '#12, 1st Main Road, Majesic, Bangalore');


--Cars:

insert into cars(brand,model,drivewheel,fuel,capacity,transmission, reg_no,
price_per_hour, branch_id, km_driven, year_manf, rating, colour, img)
values
("Tata" ,           "Nano Genx" ,       "RWD" , "Petrol" ,4
,"Automatic",   "KA01YH3456", 220, 1001, 25000, '2018',4,
'blue',   'tata_nano'),
("Tata" ,           "Nano Genx" ,       'RWD' , "CNG" ,   4
,"Manual",      "KA01HU7543", 200, 1002, 75000, '2014',3,
'blue',   'tata_nano'),
("Datsun" ,         "Redi-Go" ,         "FWD" , "Petrol" ,5
,"Manual",      "KA02GY2345", 200, 1003, 25000, '2018',3,
'red',    'datsun_redigo'),
("Datsun" ,         "Redi-Go" ,         "FWD" , "Petrol" ,5
,"Manual",      "KA02GR7532", 200, 1002, 100000, '2018',3,
'red',   'datsun_redigo'),
("Renault" ,        "Kwid" ,            "FWD" , "Petrol" ,5
,"Manual",      "KA05MX4598", 240, 1003, 25000, '2019',4,
'white',  'renault_kwid'),
("Renault" ,        "Kwid" ,            "FWD" , "Petrol" ,5
,"Manual",      "KA09WQ4918", 240, 1004, 25000, '2018',4,
'white',  'renault_kwid'),
("Maruti Suzuki" ,  "Eeco" ,            "RWD" , "Petrol" ,5
,'Manual',      "KA21AS8734", 300, 1001, 30000, '2018',2, 'silver',
'ms_ecco'),
("Maruti Suzuki" ,  "Alto K10" ,        "FWD" , "Petrol" ,5
,"Manual",      "KA06TY7698", 210, 1001, 25000, '2017',4,
'red',    'ms_altok10'),
("Maruti Suzuki" ,  "Celerio X" ,       "FWD" , "Petrol" ,5
,"Automatic",   "KA08HG3456", 230, 1002, 25000, '2018',4, 'yellow',
'ms_celerio'),
```

```
("Maruti Suzuki" ,  "Dzire" ,           "FWD" , "Diesel" ,5
,"Automatic",   "KA01UH4523", 260, 1002, 25000, '2018',4,
'white',  'ms_dzire'),
("Maruti Suzuki" ,  "Dzire" ,           "FWD" , "Diesel" ,5
,"Manual",      "KA01UH4213", 240, 1001, 25000, '2018',4,
'white',  'ms_dzire'),
("Maruti Suzuki" ,  "Dzire" ,           "FWD" , "Diesel" ,5
,"Manual",      "KA01MG4203", 240, 1003, 40000, '2016',5,
'white',  'ms_dzire'),
("Volkswagen" ,     "Ameo" ,            "FWD" , "Diesel" ,5
,"Manual",      "KA02NK7867", 260, 1003, 25000, '2018',4, 'yellow',
'vw_ameo'),
("Volkswagen" ,     "Ameo" ,            "FWD" , "Diesel" ,5
,"Manual",      "KA06YT9067", 260, 1004, 25000, '2016',4, 'yellow',
'vw_ameo'),
("Audi" ,           "A3" ,              'FWD' , "Diesel" ,5
,"Automatic",   "KA01HL2002", 420, 1003, 25000, '2018',4,
'black',  'audi_a3'),
("Audi" ,           "Q3" ,              "FWD" , "Diesel" ,5
,"Manual",      "KA08BH3002", 430, 1003, 25000, '2018',5,
'black',  'audi_q3'),
("Volvo" ,          "Xc40" ,            "AWD" , "Petrol" ,5
,"Automatic",   "KA23NG4567", 470, 1003, 5000, '2022',5,
'white',   'volvo_xc40'),
("Maruti Suzuki" ,  "Swift" ,           "FWD" , "Petrol" ,5
,"Manual",      "KA08UY6798", 220, 1004, 25000, '2018',4,
'blue',   'ms_swift')
;



--Customers:

insert into customer(cust_id,name,DOB,license_no,gender,phone_no, email,
address, password)
values
(2002, "Morty", "2002-06-
18","PQ798TY65",'Male',0987654321,  'morty@gmail.com','#4, geneva street,
kormanagala', 'mortypwd'),
(2001, "Rick", "2003-04-15","HY786TY65",'Male',9876543210,
'ricksanchez@gmail.com','#3, polo street, jp nagar', 'rickpwd')
;


--Employees:

insert into Employee(emp_id,name,dob,JoinDate,gender,branch_id, email,
address, password)
values
```

```
(3002, "Angela","1998-03-017","2020-03-05",'Female',1003,
'angelamartin@gmail.com', '#23 commerical street', 'angelapwd')
(3001, "Kevin","1995-09-02","2019-01-15",'Male',1003, 'kevin@gmail.com', '#6,
home street, ulsoor', 'keinpwd'),
;

insert into Employee(emp_id,name,dob,JoinDate,gender,branch_id, email,
address, password)
values
(3003, "Jim","1994-06-09","2019-01-15",'Male',1001, 'jim@gmail.com', '#8,
Coner House', 'jimpwd'),
(3004, "Pam","1989-05-07","2020-03-05",'Female',1002, 'pam@gmail.com', '#29 VP
street', 'pampwd'),
(3005, "Dwight","1988-11-17","2020-03-05",'Male',1004, 'dwight@gmail.com',
'#21 Farm Road', 'dwightpwd')
;
```

API queries:

```
JS addCustomer.js ×
car_rental > pages > api > db > JS addCustomer.js > handler
26      insert into customer(name,DOB,license_no,gender,phone_no, email, address, password)
27      values
28      ("${data.name}", "${data.dob}","${data.lis_no}",'${data.gender}',${data.phone_no}, '${data.email}',"${data.address}", '$
        {data.password}');
29      `;
```

```
JS addEmployee.js ×
car_rental > pages > api > db > JS addEmployee.js > handler > db.query() callback
24      `
25      insert into Employee(name, email, address,gender,password, JoinDate,branch_id,dob   )
26      values
27      ("${req.body.name}","${req.body.email}","${req.body.address}",'${req.body.gender}','${req.body.password}',curdate(), $
        {req.body.branch_id},"${req.body.dob}");
28      `;
```

```
JS authenticate_user.js ×
car_rental > pages > api > db > JS authenticate_user.js > handler > db > database
22      select * from customer
23      where (email = '${req.body.email}' or phone_no = '${req.body.phone_no}') and password = '${req.body.password}';
24      `;
```

```
JS bookCar.js ×
car_rental > pages > api > db > JS bookCar.js > handler > db.query() callback
21      insert into Transactions(cust_id, car_reg_no, branch_id, startTime, endTime,status, amount)
22      values
23      (${req.body.cust_id},
24          "${req.body.car_reg_no}",
25          ${req.body.branch_id},
26          '${req.body.startTime}',
27          '${req.body.endTime}',
28          'InProgress',
29          '${req.body.amount}'
30      )
31      `;
```

changeManager.js

car_rental > pages > api > db > changeManager.js > handler

```
const queryString =
  `
  call changeMan(${req.body.man_id},"${req.body.name}","${req.body.email}","${req.body.address}','${req.body.gender}','${req.body.password}',${req.body.branch_id},"${req.body.dob}");
  `;
```

checkUser.js

car_rental > pages > api > db > checkUser.js > handler

```
const queryString =
  `select * from customer
  where email = '${req.body.email}' or phone_no = '${req.body.phone_no}';
  `;
```

custViewInProgressCars.js

car_rental > pages > api > db > custViewInProgressCars.js > handler > db.query() callback

```
  `
  select Cr.brand as brand, Cr.model as model, Cr.reg_no as reg_no, B.name as branch, T.startTime as startTime, T.endTime as endTime, T.amount as amount, T.status as status
  from transactions T
  join customer C on T.cust_id = C.cust_id
  join cars Cr on Cr.reg_no = T.car_reg_no
  join branch B on B.branch_id = T.branch_id
  where Status = 'InProgress' and (C.email = "${req.body.email}" or C.phone_no = "${req.body.email}" or C.cust_id LIKE "${req.body.email}");
  `;
```

custViewPastCars.js

car_rental > pages > api > db > custViewPastCars.js > handler

```
  `
  select Cr.brand as brand, Cr.model as model, Cr.reg_no as reg_no, B.name as branch, T.startTime as startTime, T.endTime as endTime, T.amount as amount, T.status as status
  from transactions T
  join customer C on T.cust_id = C.cust_id
  join cars Cr on Cr.reg_no = T.car_reg_no
  join branch B on B.branch_id = T.branch_id
  where Status <> 'InProgress' and (C.email = "${req.body.email}" or C.phone_no = "${req.body.email}" or C.cust_id LIKE "${req.body.email}");
  `;
```

deleteEmployee.js

car_rental > pages > api > db > deleteEmployee.js > handler

```
  `
  delete from employee where emp_id = ${req.body.emp_id};
  `;
```

empViewInProgressCars.js

car_rental > pages > api > db > empViewInProgressCars.js > handler > db.query() callback

```
  `
  select Cr.brand as brand, Cr.model as model,Cr.reg_no as reg_no, C.name, C.email, C.cust_id, B.name as branch, T.startTime as startTime, T.endTime as endTime, T.amount as amount, T.status as status, T.trans_id as trans_id
  from transactions T
  join customer C on T.cust_id = C.cust_id
  join cars Cr on Cr.reg_no = T.car_reg_no
  join branch B on B.branch_id = T.branch_id
  where T.status = 'InProgress' and (
    T.branch_id = (select branch_id from employee where email = "${req.body.email}")
    or T.branch_id = (select branch_id from branch where name LIKE "${req.body.branch}")
    );
  `;
```

empViewPastCars.js

car_rental > pages > api > db > empViewPastCars.js > handler > queryString

```
select Cr.brand as brand, Cr.model as model,Cr.reg_no as reg_no, C.name, C.email, C.cust_id, B.name as branch, T.
startTime as startTime, T.endTime as endTime, T.amount as amount, T.status as status , T.trans_id as trans_id
from transactions T
join customer C on T.cust_id = C.cust_id
join cars Cr on Cr.reg_no = T.car_reg_no
join branch B on B.branch_id = T.branch_id
where T.status <> 'InProgress' and (
  T.branch_id = (select branch_id from employee where email = "${req.body.email}")
  or T.branch_id = (select branch_id from branch where name LIKE "${req.body.branch}")
  );
`;
```

getAllBranches.js

car_rental > pages > api > db > getAllBranches.js > handler > db.query() callback

```
`
select branch_id, name from branch;
`;
```

getAnalytics.js

car_rental > pages > api > db > getAnalytics.js > handler

```
`
select B.branch_id, B.name as branch_name,E.emp_id as manager_id, E.name as manager_name, sum(T.amount) as revenue,
count(*) as sales
from transactions T
join cars C on C.reg_no = T.car_reg_no
join branch B on T.branch_id = B.branch_id
join employee E on E.emp_id = B.manager_id
GROUP BY B.branch_id, B.name , E.emp_id,E.name;
`;
```

getBranchID.js

car_rental > pages > api > db > getBranchID.js > handler

```
`
select branch_id from employee where email = "${req.body.email}";
`;
```

getEmployees.js

car_rental > pages > api > db > getEmployees.js > handler

```
select * from employee E where E.branch_id = (
    select branch_id from employee M
    where M.email = "${req.body.email}" or M.emp_id LIKE '${req.body.emp_id}'
)
and E.email <> "${req.body.email}" and E.emp_id NOT LIKE '${req.body.emp_id}' ;
`;
```

getManagers.js

car_rental > pages > api > db > getManagers.js > handler > queryString

```
`
select B.branch_id, E.name as name, B.name as branch_name, E.emp_id, E.gender, E.JoinDate from branch B
join employee E on b.manager_id = E.emp_id;
`;
```

markAsCompleted.js

car_rental > pages > api > db > markAsCompleted.js > handler > queryString

```
`
update transactions set status = 'Completed' where trans_id = ${req.body.trans_id};

`;
```

searchCars.js

car_rental > pages > api > db > searchCars.js > handler > db.query() callback

```
`select * from cars C
where C.branch_id = (
    select B.branch_id from branch B where B.name = '${req.body.branch}'
)
and not exists(
    select * from transactions T
    where T.car_reg_no = C.reg_no and(
    startTime BETWEEN "${req.body.startTime}" AND "${req.body.endTime}"
    or endTime BETWEEN "${req.body.startTime}" AND "${req.body.endTime}"
    )
)`;
```

searchFilteredCars.js

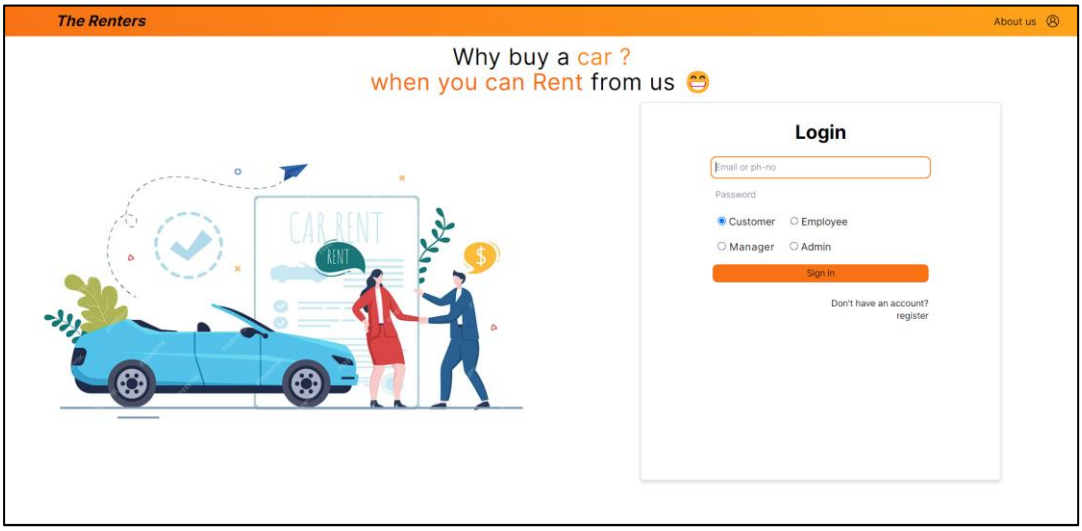
car_rental > pages > api > db > searchFilteredCars.js > handler > db.query() callback

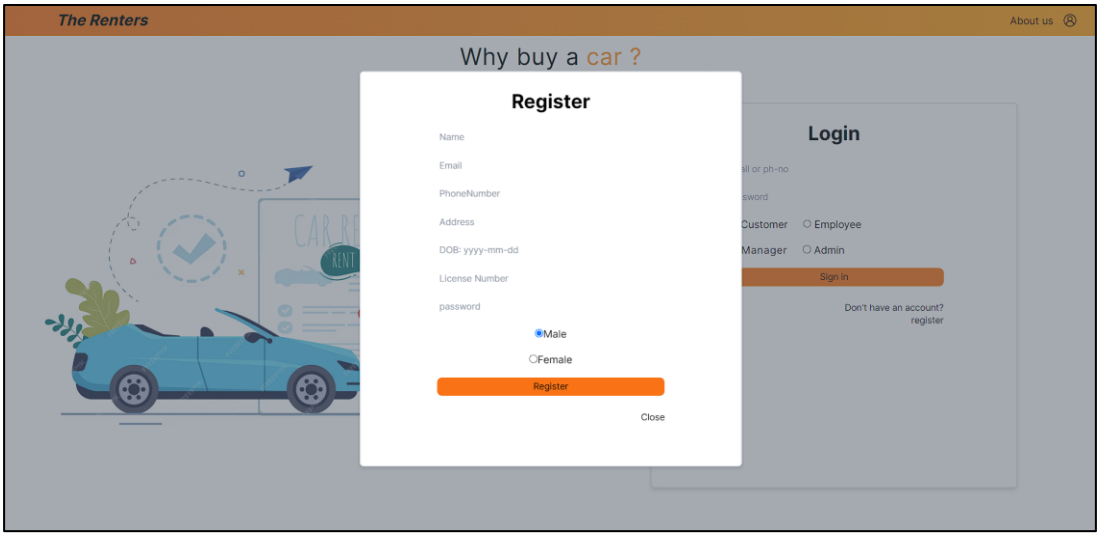
```
`select * from cars C
where C.branch_id = (
    select B.branch_id from branch B where B.name = '${req.body.branch}'
)
and not exists(
    select * from transactions T
    where T.car_reg_no = C.reg_no and(
    startTime BETWEEN "${req.body.startTime}" AND "${req.body.endTime}"
    or endTime BETWEEN "${req.body.startTime}" AND "${req.body.endTime}"
    )
)
and C.fuel LIKE '${req.body.fuel}'
and C.transmission LIKE '${req.body.transmission}'
and C.rating LIKE '${req.body.ratings}'
and C.capacity LIKE '${req.body.capacity}'
ORDER BY C.price_per_hour ${req.body.sort}
;
`;
```

8. Functionalities

Login/Sign-up:

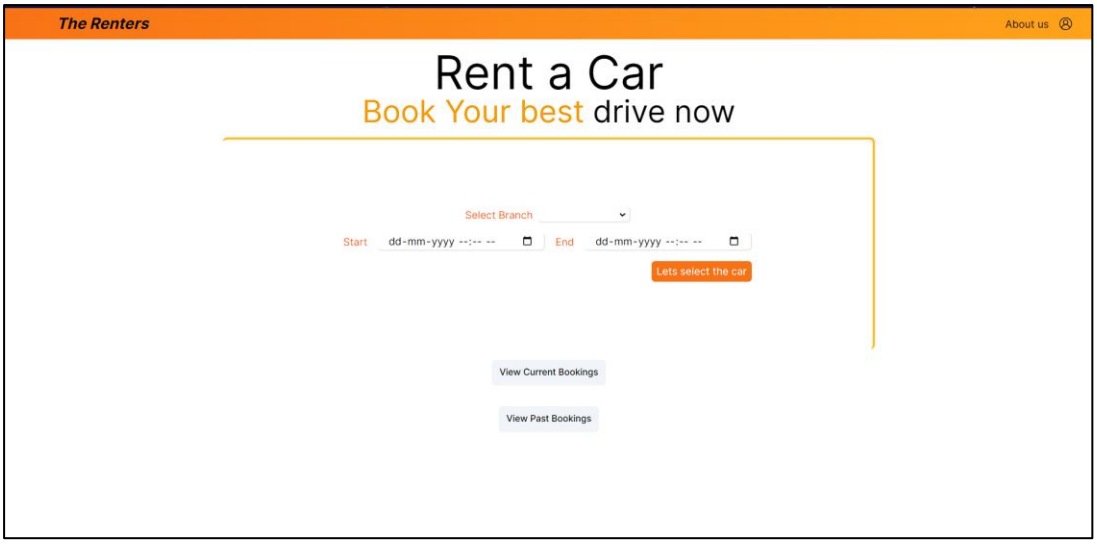
Customers, employees, managers and the admin can all login from the same page. New customers can create a new account for themselves.





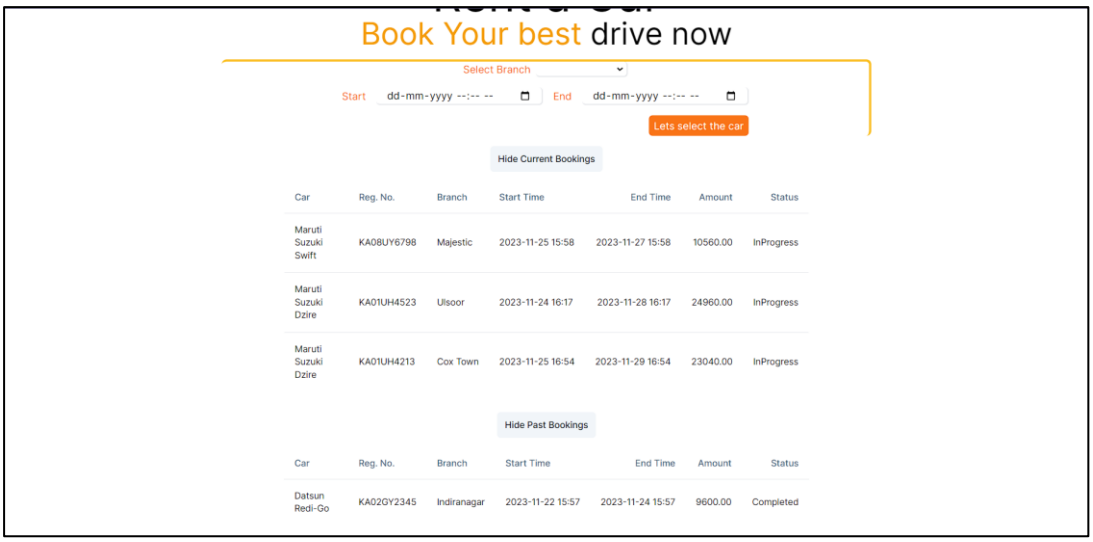
Search Cars:

Customers can search cars by selecting a branch and a date-time range.



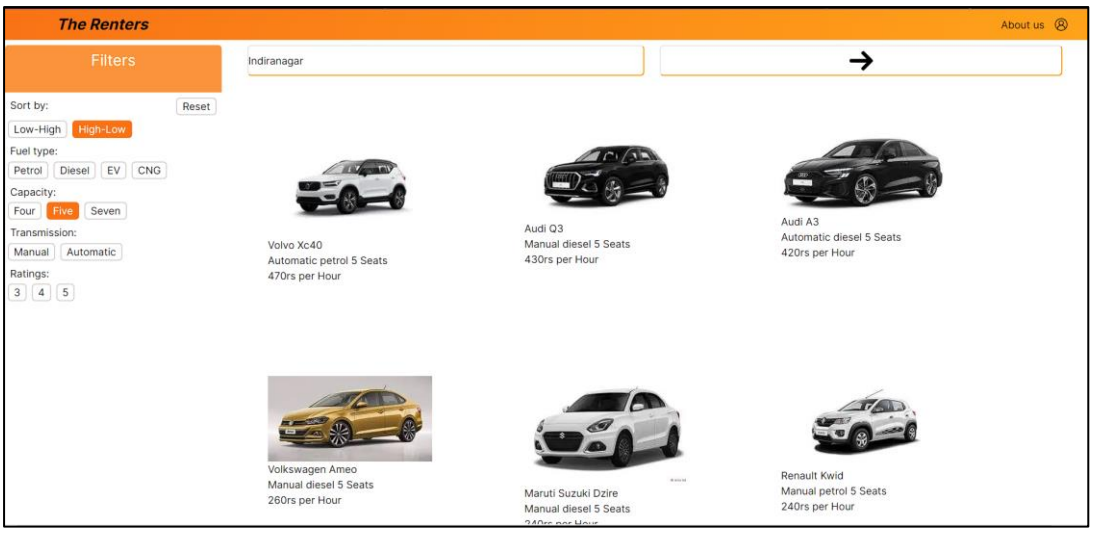
View Bookings:

Customers can view their pasts and ongoing bookings.



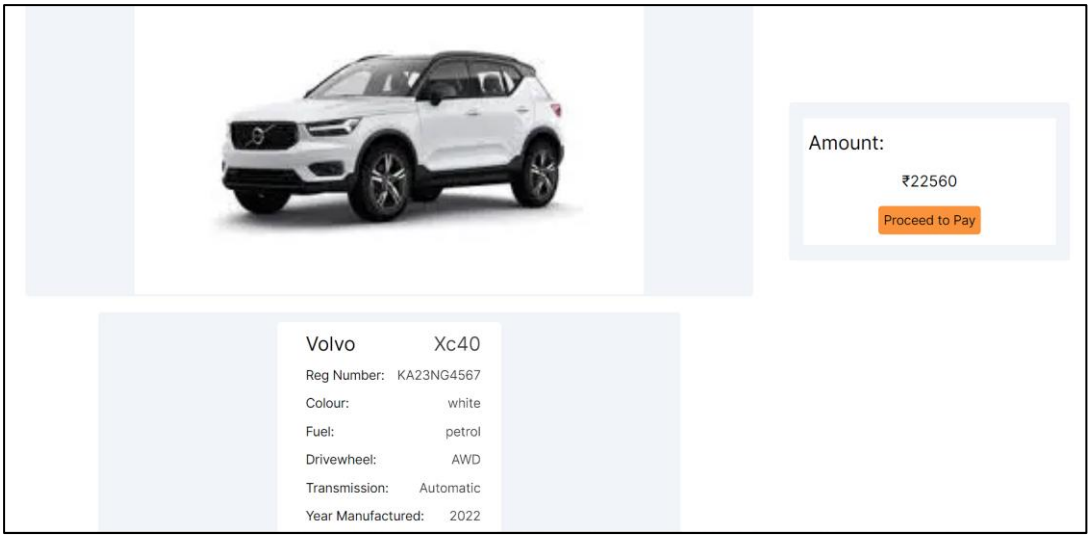
Filter Cars:

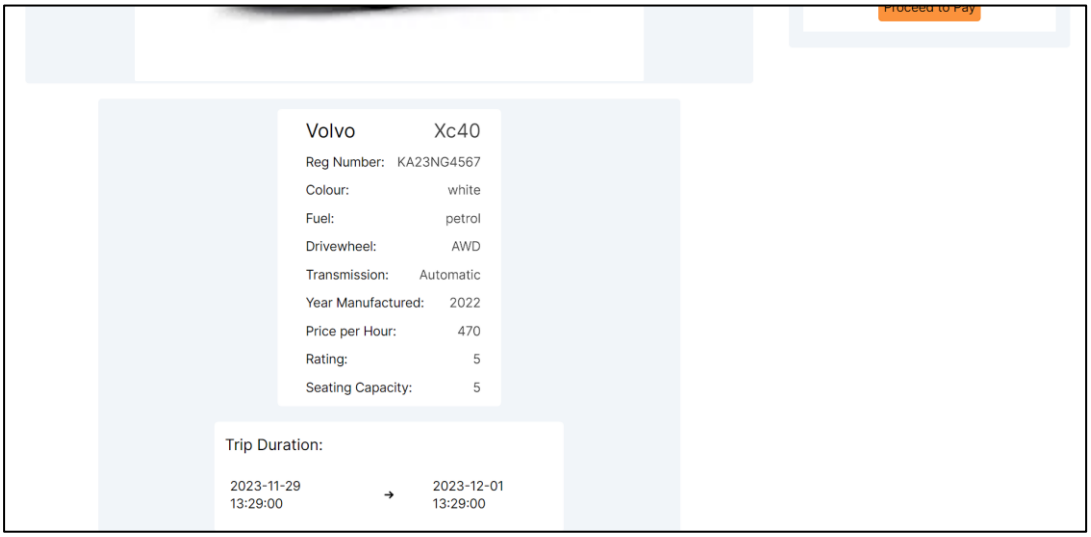
Customers can filter the results of available cars.



Pre-booking Summary:

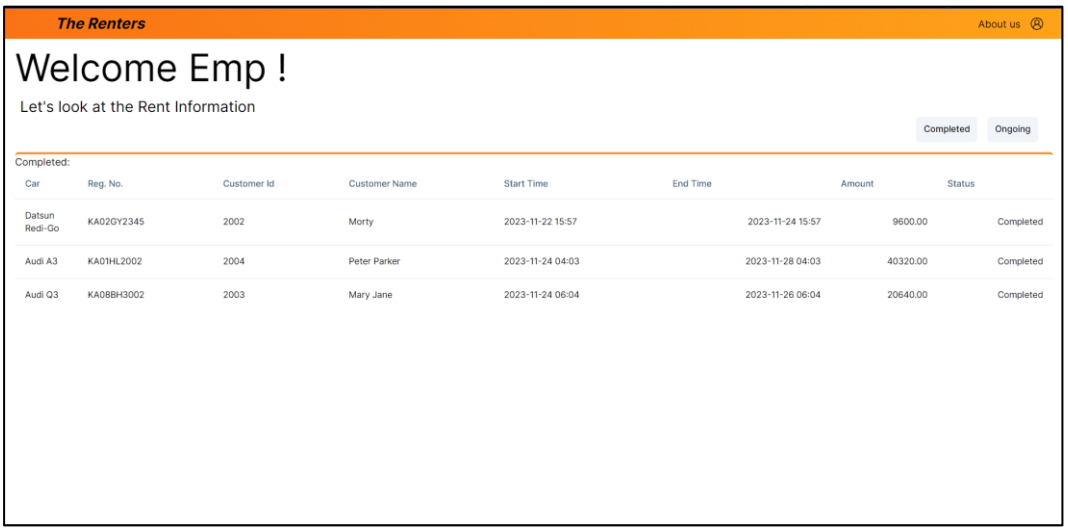
Customers can see all details of their booking before proceeding to pay.

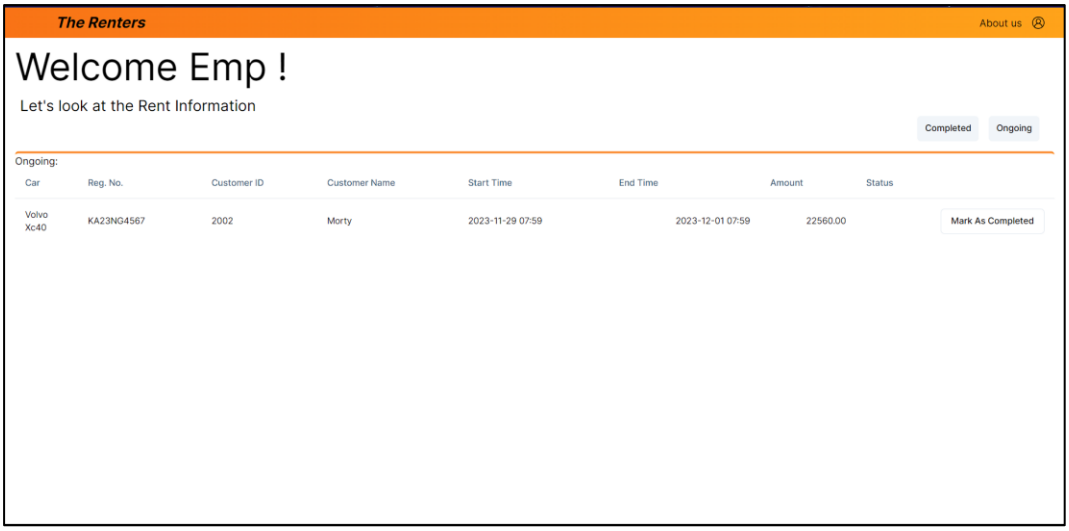




Employee Booking Management:

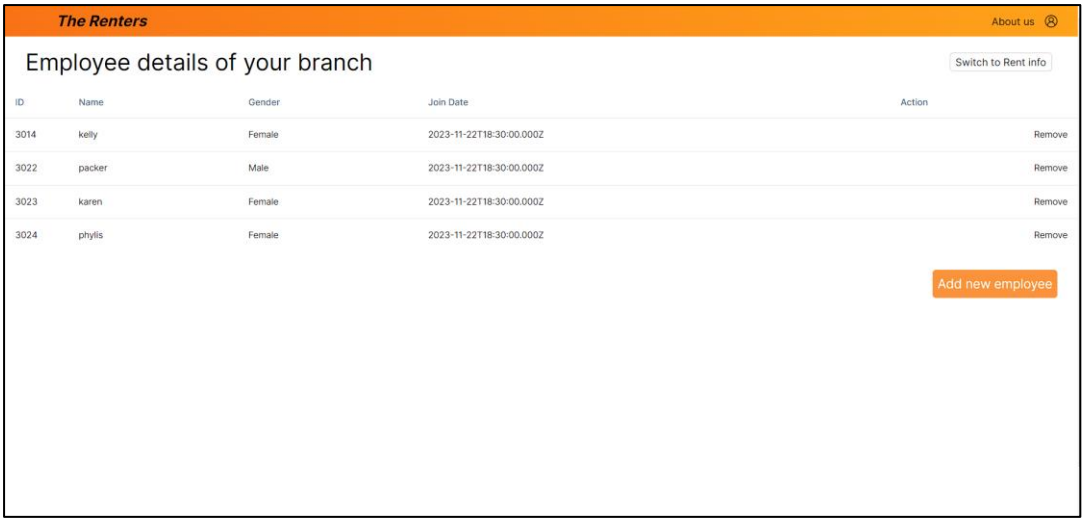
Employees can manage bookings of their branch. They can mark any ongoing booking as completed.

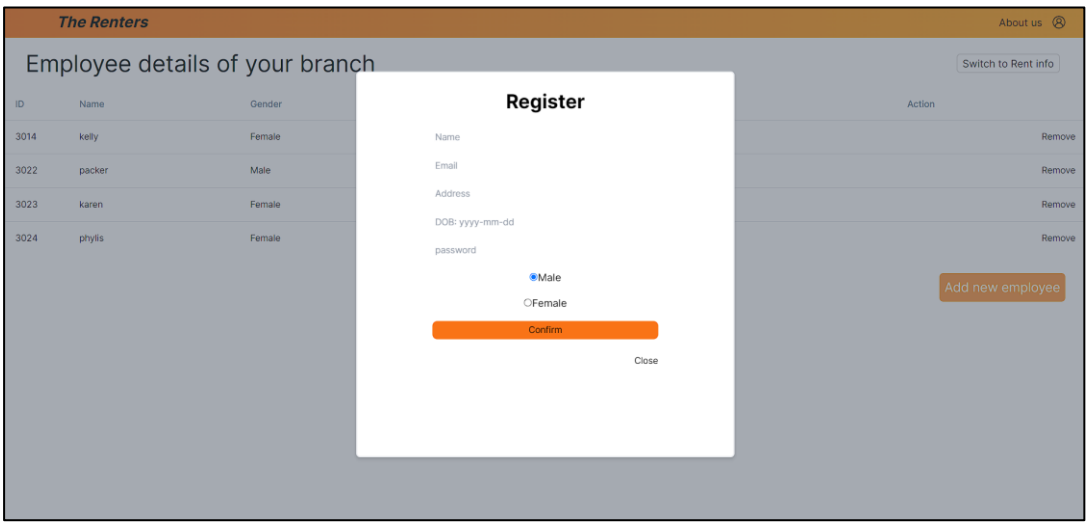




View/Add Employees:

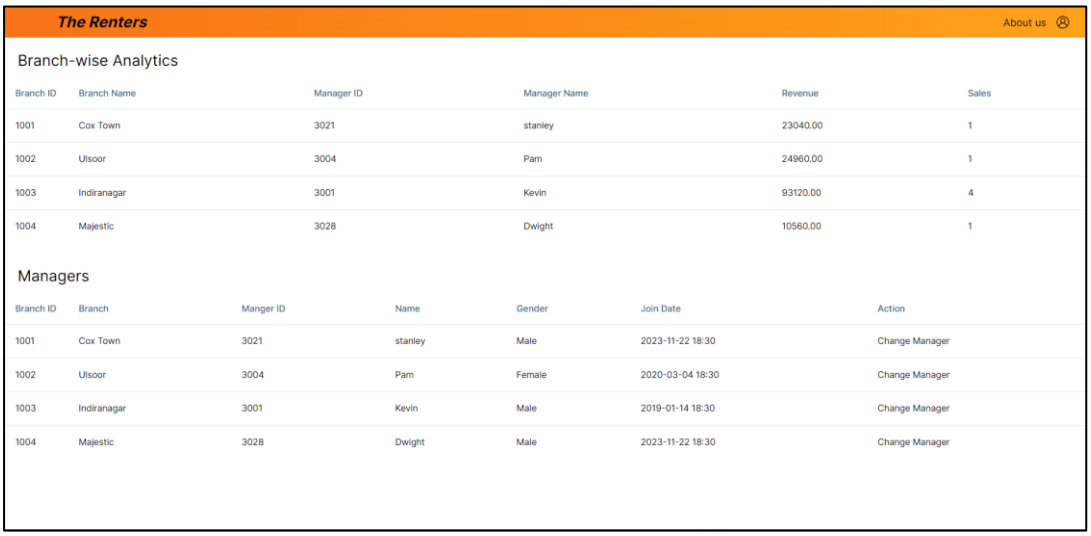
The manager of a branch can view all employees of their branch as well as add a new employee.

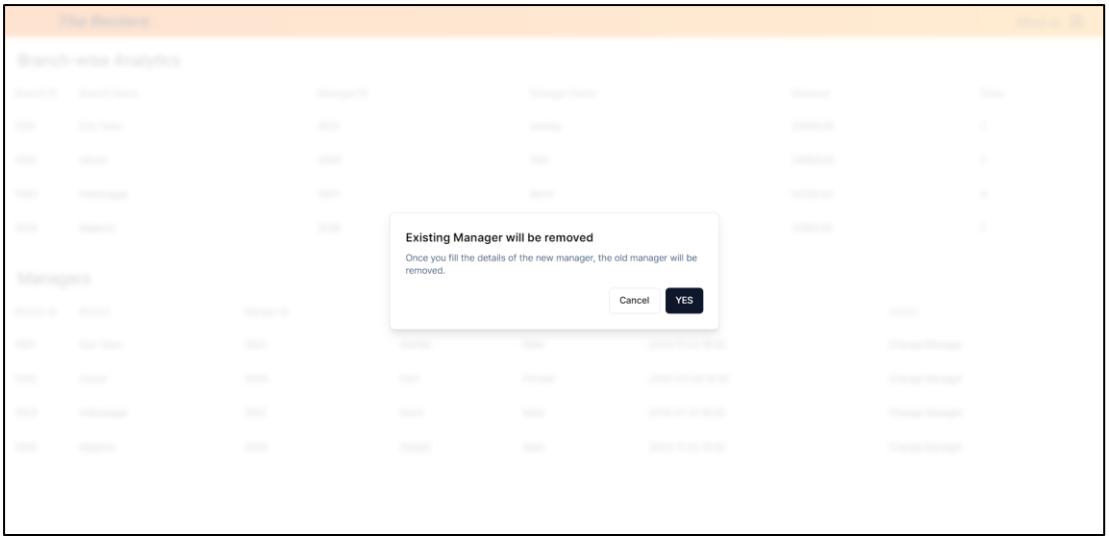


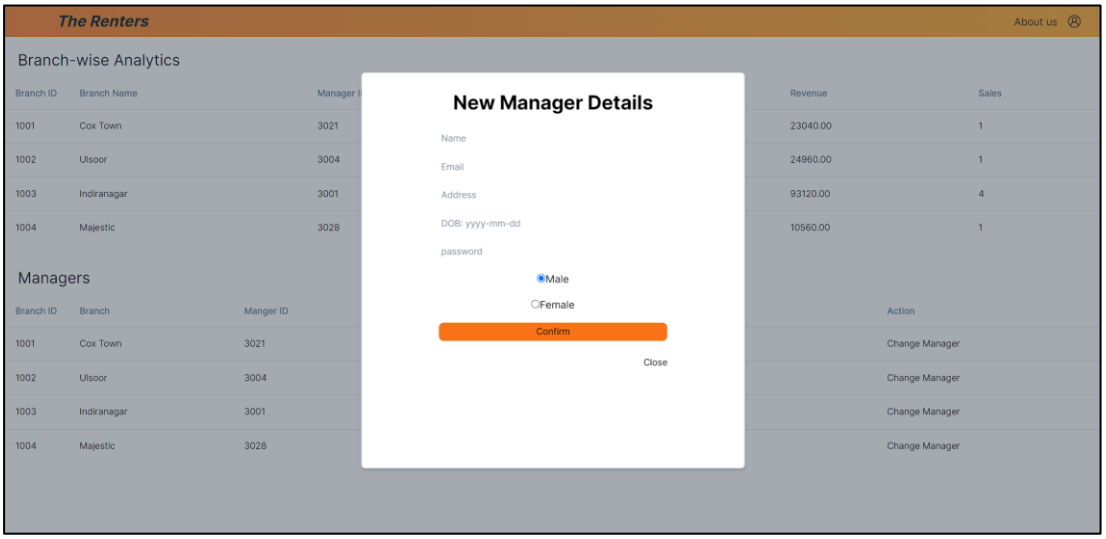


View Managers:

The admin can view all branch managers and even change the manager of a branch.

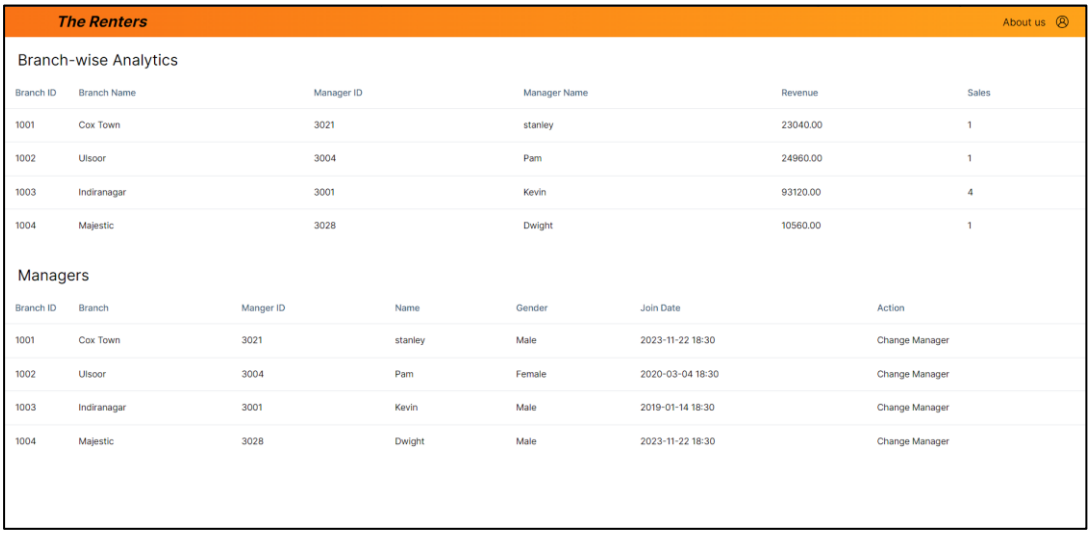






Branch-wise Analytics:

Admin can also see the revenue and sales numbers of each branch.



9. Procedures and Triggers

Procedures:

```
-- changing manager:

drop procedure if exists changeMan;
Delimiter $$
CREATE PROCEDURE changeMan (
    IN old_man_id bigint,
    IN gname VARCHAR(30),
    IN gemail VARCHAR(30),
    IN gaddress VARCHAR(80),
    IN ggender VARCHAR(10),
    IN gpassword VARCHAR(20),
    IN gbranch_id bigint,
    IN gdob date
)
BEGIN

    insert into Employee(name, email, address,gender,password,
JoinDate,branch_id,dob)
    values
    (gname,gemail,gaddress,ggender,gpassword,curdate(), gbranch_id,gdob);

    delete from employee where emp_id = old_man_id;

    update branch set manager_id = (select emp_id from employee where email =
gemail)
    where branch_id = gbranch_id;

end $$
Delimiter ;

---------
```

Triggers:

```
--- checking for duplicate employee:

Delimiter &&
CREATE TRIGGER employee_insert_trigger
    BEFORE INSERT
    ON employee
    FOR EACH ROW
    BEGIN
    IF EXISTS (SELECT 1
```

```
        FROM employee
        WHERE email = NEW.email)
        THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'User Already Exists ';
    END IF;
END&&
Delimiter ;

---  checking for duplicate customer:

Delimiter &&
CREATE TRIGGER customer_insert_trigger
    BEFORE INSERT
    ON customer
    FOR EACH ROW
    BEGIN
    IF EXISTS (SELECT 1
        FROM customer
        WHERE email = NEW.email)
        THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'User Already Exists ';
    END IF;
END&&
Delimiter ;
```