CS353 Project Final Report

CaRent

Group Members

Kaan Ateşel 21703694

Cemal Faruk Güney 21903474

Arda Önal 21903350

Beste Güney 21901631

Instructor: Özgür Ulusoy

TA: Mustafa Can Cavdar

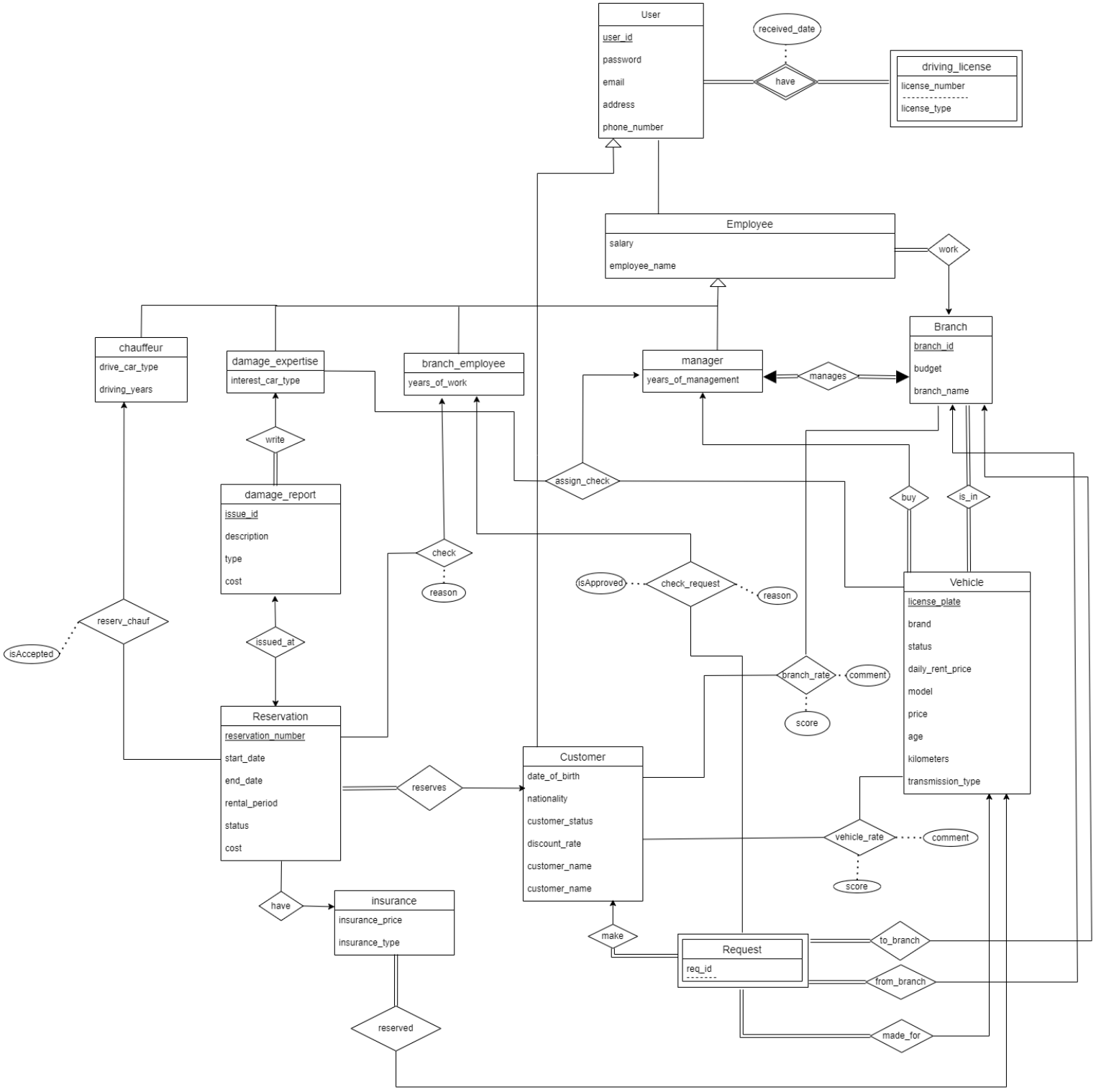
Website: https://kaanatesel.github.io/carent/

Table of Contents

System Description

CaRent is a web based car rental system that makes the lives of people that want to rent vehicles easier. The system can be used by five different types of users which are customers, branch employees, branch managers, chauffeurs and damage experts. Users can sign up to the system by providing user specific type information and can log into the system using their email and passwords. Depending on the user type, they are directed to different types of user interfaces. When a customer logs in, they are directed to a page which displays the list of available vehicles that they can rent. These vehicles are displayed with additional information which consists of its plate, status, daily rental price, age, kilometers, brand name, model and the branch that the vehicle currently is in. By clicking the “rent” button, the user is directed to another page where they enter the rental start date, end date, a chauffeur if they want any and insurance type if they want to have any to create a reservation request which will be processed by the branch employee and chauffeur (if they wanted any). Customers can also see their reservations with their status which displays if it is accepted by employees or not. Customers can create a transfer request which creates a request that will transfer the specified vehicle from a branch to another branch. Customers can also rate vehicles and branches. Lastly, customers can return vehicles that they rented to any branch that they want. When they return a vehicle, they have to pay the rental price. Depending on the customer status type (Gold, Silver, Premium), they receive additional discounts while paying the price. After they return the vehicle, if the car was damaged which is reported by damage expertises, they have to pay the repair cost. If they don’t pay on time, their customer status changes and if they continue to pay their rents on time, their status upgrades. When a branch employee logs in, they are directed to a page which displays the list of all reservations. If a reservation is not approved by another employee before, a button which allows to approve or reject the reservation request appears. They can also approve or reject transfer requests on a different page. Lastly, they can see all the vehicles that belong to their branch. When a manager logs in, they are directed to a page that displays their branches employee’s statistics such as which employee has accepted most reservations, requests etc. They can also see all of the cars in their branch, buy cars to their branch and view all the employees of their branch. When a chauffeur logs in, they are directed to a page where they can see chauffeur requests assigned to them. If the branch employee has not accepted the reservation request yet, chauffeur cannot accept the chauffeur request until the branch employee accepts the reservation. When a damage expert logs in, they can see the list of paid reservations that was not checked by any of the damage experts. They can create a damage report if the car is damaged. Lastly, all users can log out from the system which directs them back to the login page.

Final E/R Diagram



List of Tables

(Primary keys are underlined and F.K. is an abbreviation for foreign key)

User(user_id, password, email, address, phone_number)

driving_license(user_id, license_number, license_type, received_date)
F.K. user_id references User(user_id)

Employee(user_id, salary, employee_name, branch_id)
F.K. branch_id references branch(branch_id)
F.K. user_id references user(user_id)

Customer(user_id, date_of_birth, nationality, customer_status, customer_name)
F.K. user_id references user(user_id),
F.K. customer_status references customer_discount(customer_status)

customer_discount(customer_status, discount_rate)

Branch(branch_id, budget, branch_name, city, manager_id)
F.K. manager_id references manager(user_id)

Manager(user_id, years_of_management)
F.K. user_id references Employee(user_id)

Vehicle(license_plate, status, daily_rent_price, model, price, age, kilometers,
transmission_type, buying_manager_id, branch_id)
F.K. buying_manager_id references manager(user_id)
F.K. branch_id references branch(branch_id)
F.K. model references model_brand(model)

model_brand(model, brand)

car(license_plate, car_type)
F.K. license_plate references vehicle(license_plate)

branch_employee(user_id, years_of_work)
F.K. user_id references Employee(user_id)

damage_expertise(user_id, interest_car_type)
F.K. user_id references Employee(user_id)

damage_report(issue_id, description, type, cost, author_expertise_id, issued_reservation)
F.K. author_expertise_id references damage_expertise(user_id)
F.K. issued_reservation references reservation(reservation_id)

Reservation(reservation_number, start_date, end_date, rental_period, status, cost, reserver, checked_by, isApproved, reason, insurance_type, license_plate, reserved_chauf_id, isChaufAccepted)
F.K. reserver references Customer(user_id)
F.K. checked_by references branch_employee(user_id)
F.K. license_plate references vehicle(license_plate)
F.K. reserved_chauf_id references chauffeur(user_id)
F.K. insurance_type references insurance(insurance_type)

insurance(insurance_type, insurance_price)

chauffeur(user_id, drive_car_type, driving_years)
F.K. user_id references Employee(user_id)

Request(req_id, made_by_customer, from_branch, to_branch, requested_vehicle, checked_by_employee, isApproved, reason)
F.K. made_by_customer references Customer(user_id)
F.K. from_branch references Branch(branch_id)
F.K. to_branch references Branch(branch_id)
F.K. requested_vehicle references Vehicle(license_plate)
F.K. checked_by_employee references

Vehicle_Rate(customer_id, license_plate, comment, score)
F.K. customer_id references Customer(user_id)
F.K. license_plate references Vehicle(license_plate)

Branch_Rate(customer_id, branch_id, comment, score)
F.K. customer_id references Customer(user_id)
F.K. branch_id references Branch(branch_id)

assign_check(assigned_expertise_id, assigned_vehicle_license_plate)
F.K. assigned_expertise_id references damage_expertise(user_id)
F.K. assigned_vehicle_license_plate references Vehicle(license_plate)

Implementation Details

For the CaRent project, we chose **MySQL** as the data storage because we had already met with this database in the course. It also comes with the **PhpMyAdmin** user interface tool. Thus we don't have to work with the console every time we need to check and change the database. In our web browsers, we write our queries and check the result with **PhpMyAdmin**. It was the primary tool that we used to interact with the database. **Python Django** is the main framework for both the front-end and back-end sides of the project. We chose Django because it has a fast development process and also the **DTL**(Django Template Language) provides an efficient and easy implementation of the UI. In addition to the DTL, we also used **Bootstrap** to create a fast, reliable and responsive UI. Bootstrap tool kit provided us with all the complement that we needed in the CaRent UI. The components are also responsive so the project also has a good UI in mobile web browsers. To make connections between the Django app and MySQL we need these; **mysqlclient, mysql, mysql-connector** packages. For executing a query first we get the instance of the database connection then create the SQL query as a string, finally we run the query by calling the connections to execute the method. This method returns the query result as a tuple. At the end of the implementation phase, we had a development environment which was running both MySQL and PhpMyAdmin as a database and administration tool. In addition to these, it has Python 3.8.10 and a **python virtual environment** to run the Django application. We used **PIP** as the package manager for python.

The application was crushed if any database exception occurred because we didn't have proper error handling. First, we solve this problem by try and expect blocks in the backend and redirecting the user to another and error page in the front-end. However, by doing this we had lost the chance of giving user messages on the same page. Therefore it was not usable so we decided to implement the messages framework of Django so we can give both user messages at the same page and had proper error handling.

In the HTML form, we didn't have dropdowns so the user should know what to enter in this area. For example, while creating a vehicle if he/she wants a chauffeur full name should be known. It was not practical and user friendly. We solve this problem by using Django's select choices widget for HTML forms.

Kaan Ateşel: He coded the UI as static HTML and these pages changed and were used in the project. He implemented customer dashboard, rate vehicle, rate branch, return vehicle, make reservation and pay vehicle pages.

Cemal Faruk Güney: Implemented the request and penalty functionality for customer, assign vehicle functionality for manager, damage expert account and its functionalities.

Arda Önal: Implemented the chauffeur account and its functionalities, employee's branch and requests pages, logout functionality to all user types and a database initialization program where all the database tables are created/dropped with sample data.

Beste Güney: She has implemented the manager account and its functionalities. She has also coded signup and login pages together with some advanced database concepts at the program.

Advanced Database Features

a) Reports

For Manager:

Managers can access three different reports at their dashboard. These reports include some necessary and intriguing information about their particular branch and their employees.

- Report 1:
  In this report, managers can see their monthly income from reservations by grouping ‘paid’ reservations belonging to that branch monthly. In addition to that, managers can see the employee that made the maximum reservation from price that month.
  SQL Statements:

```
select month(R.start_date) as month,
(select S.employee_name from sum_reserved_employee S
where S.month = month(R.start_date) and
 S.sum_cost=(select max(sum_cost) from sum_reserved_employee
T where T.month = S.month group by T.month)) as employee,
sum(R.cost) as total_income from reservation R ,
vehicle V where R.license_plate = V.license_plate
 and R.status = 'paid' and V.branch_id = @branch_id  group by
month(start_date)
```

  NOTE: Branch_id is a parameter we provide to the sql query
  NOTE: sum_reserved_employee is a view, its definition is in the view section.

- Report 2:
  In addition to the total monthly income, managers can also see the most expensive reservation and who has done it.

  SQL Statement:

```
 select B.user_id, (select employee_name from employee
where employee.user_id = B.user_id)
as name, T.cost, T.start_date from branch_employee B,
(select reservation_number, checked_by, max(cost) as cost, start_date
from reservation group by month(start_date)) as T where T.checked_by =
B.user_id and (select branch_id from employee where employee.user_id =
B.user_id) = @branch_id ;
```

- Report 3:

In this report, managers can see the approval and rejection number of their employees.

SQL Statement:

```
 select count(*) as operation_number, status,
(select employee_name from employee where employee.user_id = B.user_id)
as name from branch_employee B ,
reservation R where B.user_id = R.checked_by
and (select branch_id from employee where employee.user_id = B.user_id)  =
@branch_id group by status having count(*) > 0;
```

For customer:

In their dashboard, customers can see their average expense on each branch together with their rating and number of reservations at their branch.

SQL Statement:

```
SELECT (select branch_name from branch where branch_id = V.branch_id)
as name, count(*) as number_of_reservations, (select score from branch_rate
where branch_id = V.branch_id and customer_id = R.reserver)
as your_score, avg(cost) as cost FROM reservation R, vehicle V where
R.license_plate = V.license_plate and R.reserver = @user_id and R.status <>
'canceled' group by V.branch_id;
```

b) Views

- The first view is used at the manager dashboard page to find the total income of reservations made by each employee monthly.

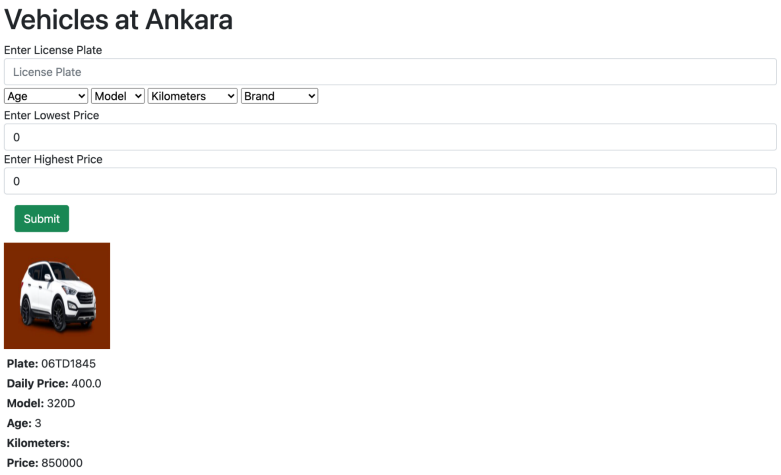
The total income and most harworking employee for month

Month	Employee Name	Total Income
11	Ayse Fatma	207.0

```
create view sum_reserved_employee as
(select employee_name, sum(cost) as sum_cost, month(start_date) as month
from reservation, employee, vehicle where reservation.license_plate =
vehicle.license_plate and reservation.checked_by = employee.user_id and
reservation.status = ‘paid’ group by reservation.checked_by,
month(start_date));
```

- Filtering the vehicles according to their age, model, kilometers and other information, many views are used. Due to the fact that it is not possible to know the fields that the user will request to filter, after input checks, views are created. These view statements can be seen at the manager account branch

vehicle displaying page, car buying page and employee account branch vehicle displaying page.



- create view filter_plate as select * from vehicle where license_plate like @ license%;
- create view filter_age as select * from filter_plate where age between @lower_bound and @upper_bound;
- create view filter_model as select * from filter_age where model = @model;
- create view filter_km as select * from filter_model where kilometers > 40000;
- create view filter_price as select * from filter_km where daily_rent_price > @low;
- create view filter_brand as select * from filter_price where brand= @brand;

c) Triggers, Constraints and Other Procedures

Triggers:

- Whenever a manager buys a new car to his/her branch, the branch budget needs to be updated.
  SQL Statement:
  create trigger update_branch_budget
  after update on vehicle
  for each row
  begin
  if OLD.status = "onsale" then
  update branch set budget = budget - OLD.price
  where branch.branch_id = NEW.branch_id;
  end if;
  end;

- Whenever a transfer request is accepted, vehicle status should change to on_transfer

```
create trigger update_vehicle_status
   after update on request
   for each row
   begin
   if NEW.isApproved = 1 then
   update vehicle set status = "on_transfer"
   where vehicle.license_plate = NEW.requested_vehicle; end if;
   end;
```

- If a user tries to make a reservation at the dates on which he/she already has a reservation, the system should not allow it.

```
create trigger assert_reservation_control before insert on reservation
        for each row
        begin
        if exists(select * from reservation where  NEW.start_date >=
start_date and NEW.end_date <= end_date and NEW.reserver =
reservation.reserver) then
        SIGNAL SQLSTATE '45000'
           SET MESSAGE_TEXT = 'BIG Error';
        end if;
        end;
```

- If a user tries to make a reservation for a vehicle that is already rented, the system should not allow it.

```
create trigger assert_availability_car before insert on reservation
   for each row
   begin
   if exists(select * from reservation where  NEW.start_date >=
start_date and NEW.end_date <= end_date and NEW.license_plate =
reservation.license_plate) then
   SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'BIG Errorrrr';
   end if;
   end;
```

Stored Procedures:

- Insertion of a user to the system is a very common operation. For that, a stored procedure is used.
  SQL Statement:
  ```
  create procedure insert_user( in emailValue varchar(50), in
  passwordValue varchar(50), in addressValue varchar(50), in phone
  varchar(15))
    begin
      insert into user(email, password, address, phone_number) values(
  emailValue, passwordValue, addressValue, phone);
    end;
  ```

- Insertion of an employee is also a needed operation in the system. For this procedure, there is a stored procedure.
  SQL Statement:
  ```
  create procedure insert_employee(in userId int , in sal numeric(8,2), in
  name varchar(20), in branch int)
    begin
      insert into employee(user_id, salary, employee_name, branch_id)
  values(userId, sal, name, branch );
    end
  ```

- Selecting the employees of a branch for an operation is used in the system.
  SQL Statement:
  ```
  create procedure select_employees(in branch int, in manager int)
    begin
      select * from employee where branch_id= branch and user_id <>
  manager;
    end
  ```

Constraints:

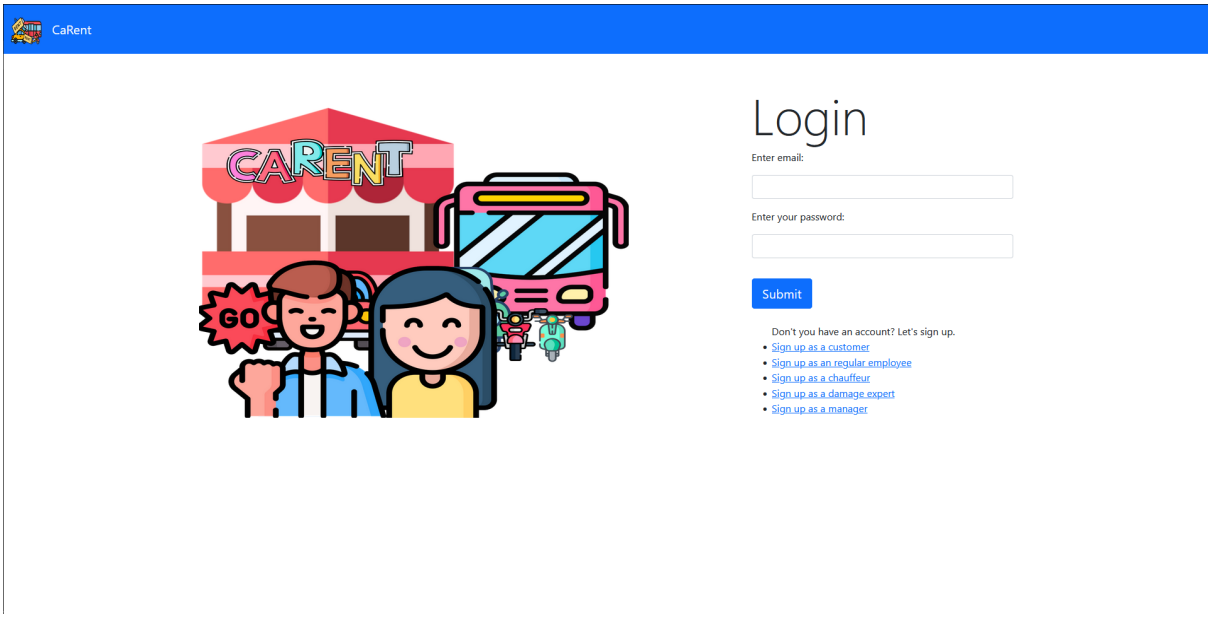
- At table creations, there are many constraints that need to be satisfied for the tables in the system.
  For example;

  At the driving license table;
  ```
  check (license_type in ("A1", "A2", "A", "M", "B1", "B", "BE", "C1",
  "C", "CE"))
  ```

  At vehicle table;

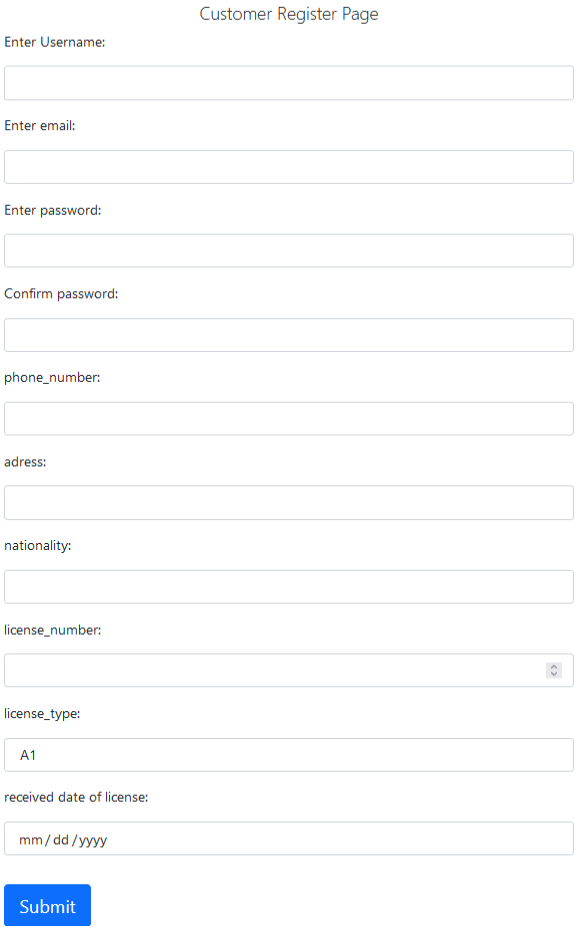
check (status in ("on_rent", "onsale", "available", "on_transfer", "unavailable", "reserved"))

User’s Manual

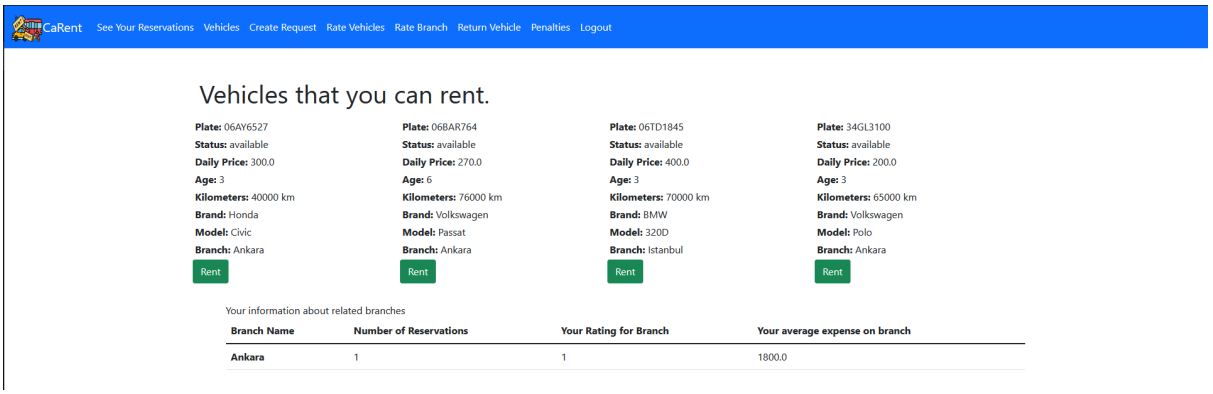


Users of this application can log in to their accounts using the login page. If they do not have an account they can sign-up for one using the sign up links.

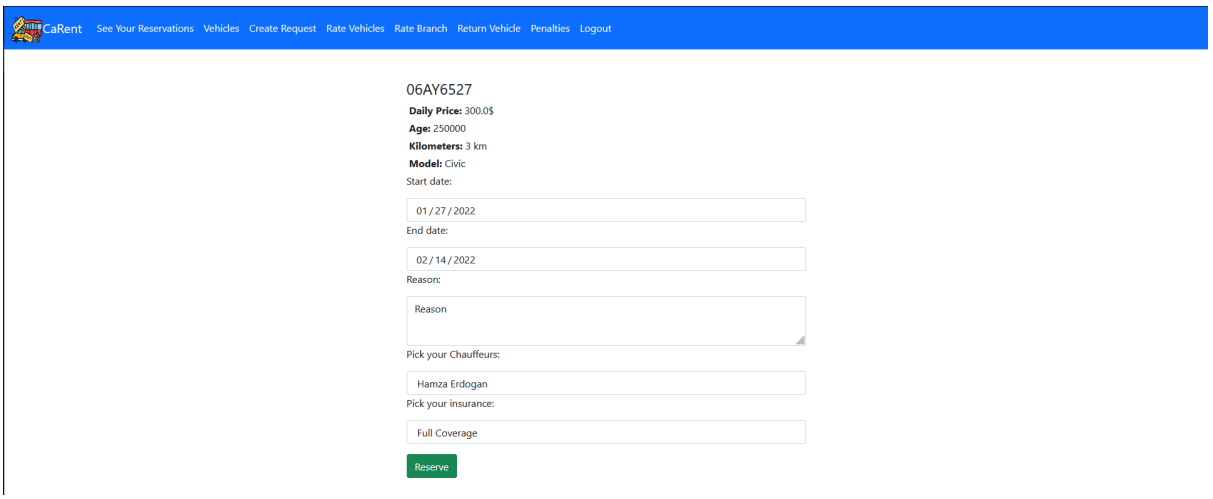
CUSTOMER



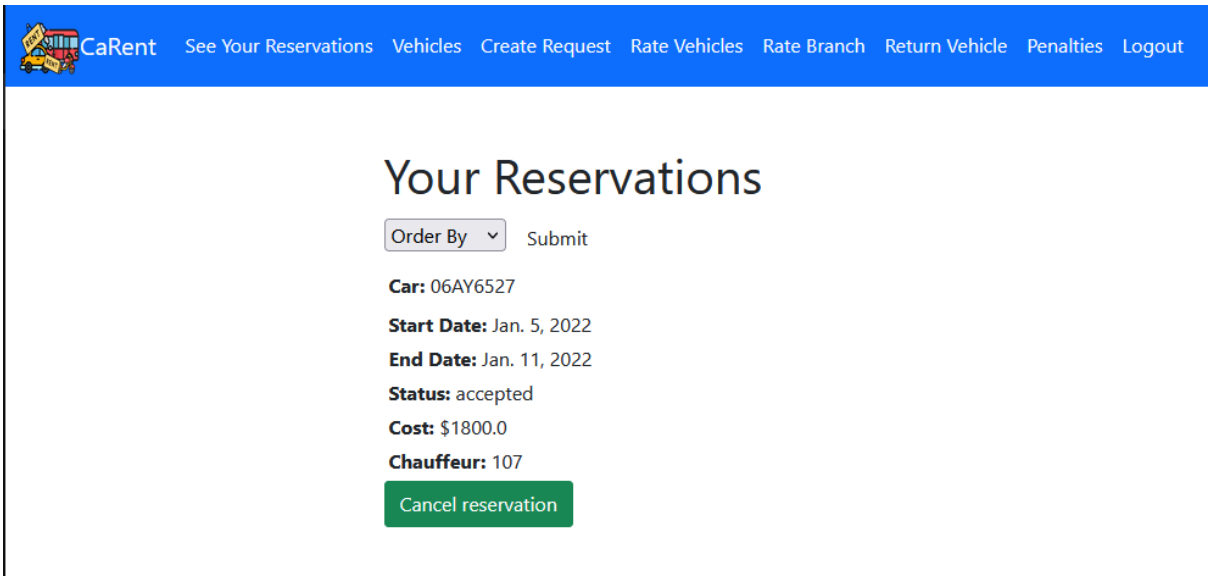
Customers can sign up by entering their credentials and their driver's license information.



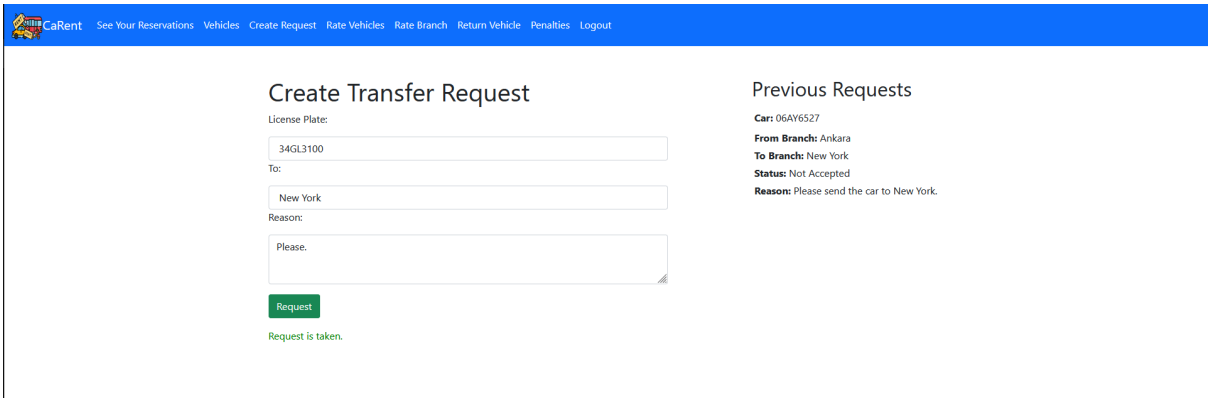
The first page for a customer displays the vehicles that are available for him/her to rent. The customer's rating for branches are also displayed on this page.



If the customer decides on renting a vehicle and clicks the rent button on the previous screen this page is displayed. After entering valid information related to how and when they want to rent the vehicle they can submit their reservation.

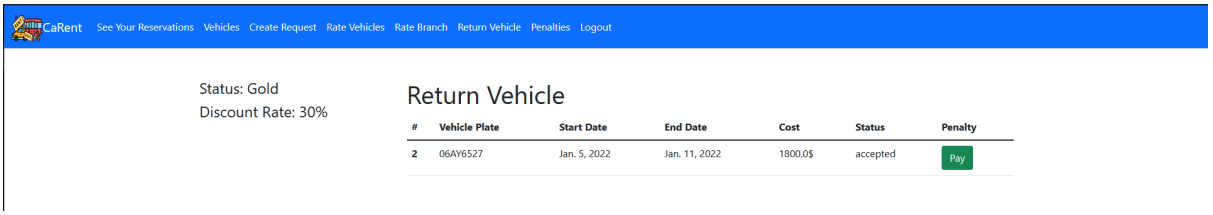


The reservation is then displayed on the See Your Reservations screen. The user can cancel the reservation from this page.

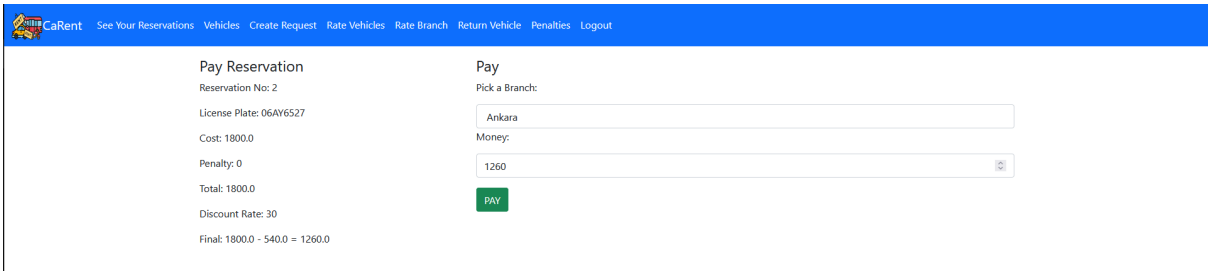


Users can create transfer requests and see their previous requests in the Create Request page. They have to select the license plate of a vehicle in the system and select a branch which will

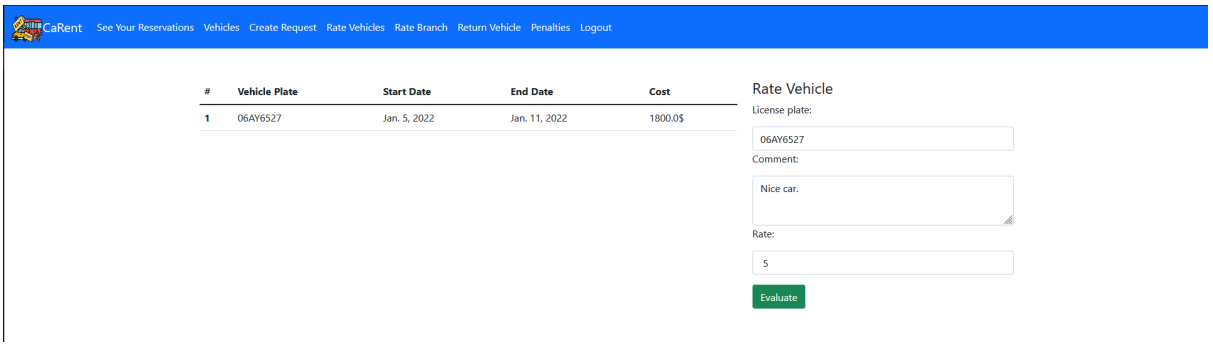
be the next location of the vehicle. The branch should not be the same branch that the car is at.



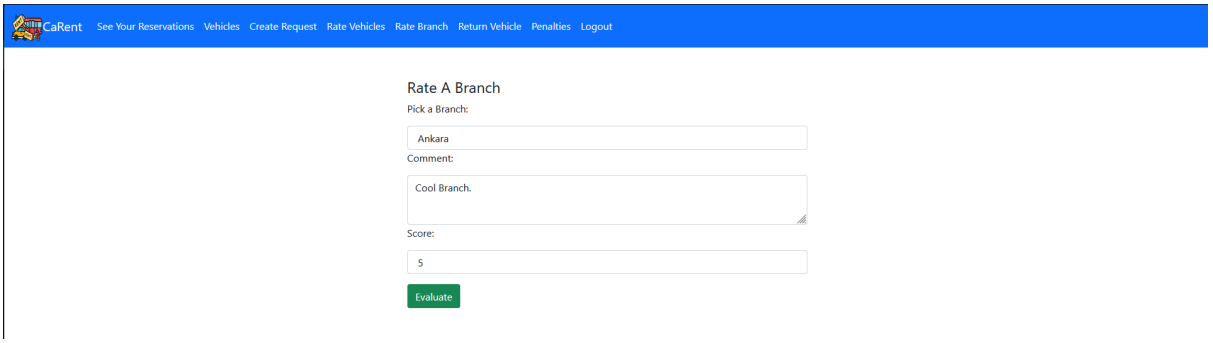
The user can see their account status and return a vehicle they have rented on the Return Vehicle page. They can pay for the rent by clicking the Pay button.



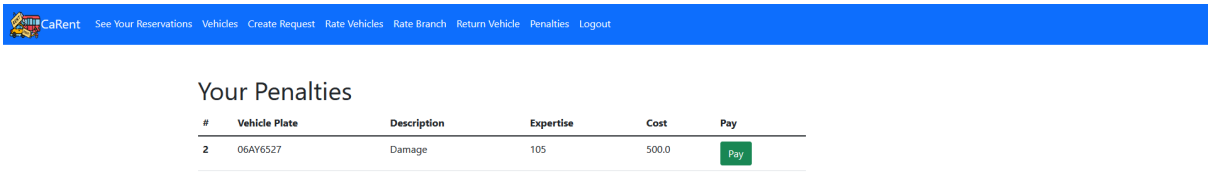
The pay button in the previous page takes the user to this pay page. In this page users can select the branch they are leaving the vehicle and enter the correct amount to pay for the rent.



After returning a vehicle users can rate the vehicle by entering the license plate of the vehicle, a comment and giving it a point out of 5.

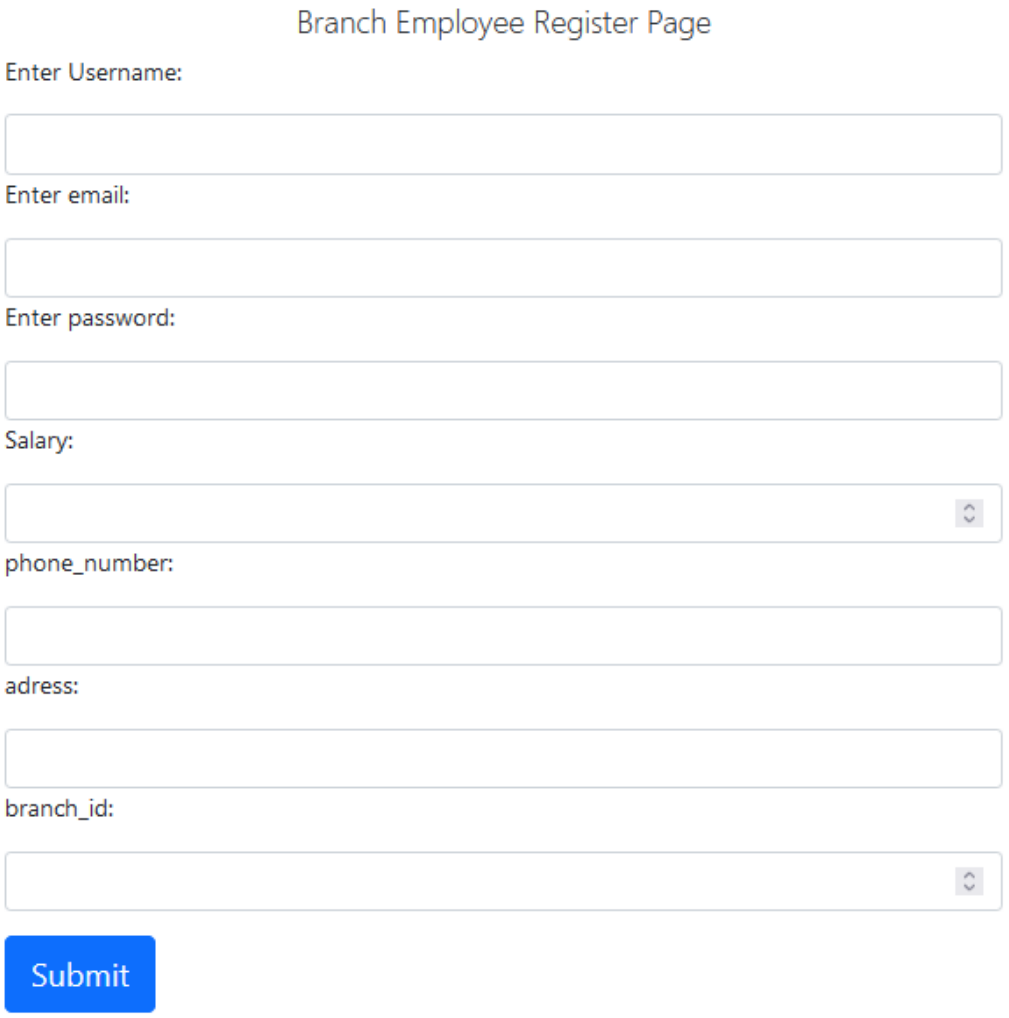


After returning a vehicle users can also rate the branch that they have used. By selecting the branch name, entering a comment and giving it a point out of 5.

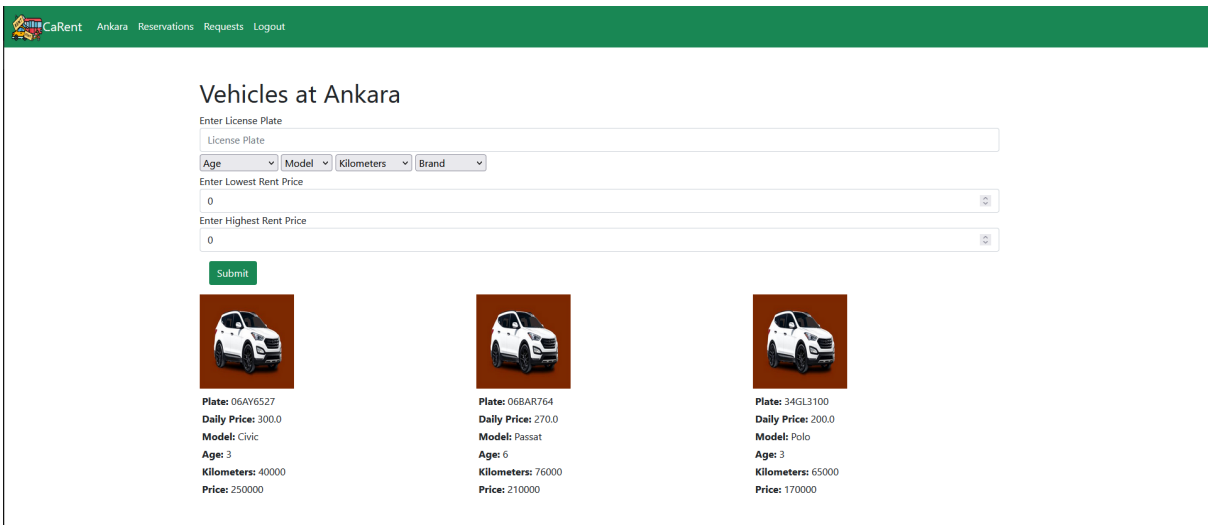


The user can see their penalties related to their previously completed rents on the Penalties page. They can pay these penalties by clicking the Pay button.

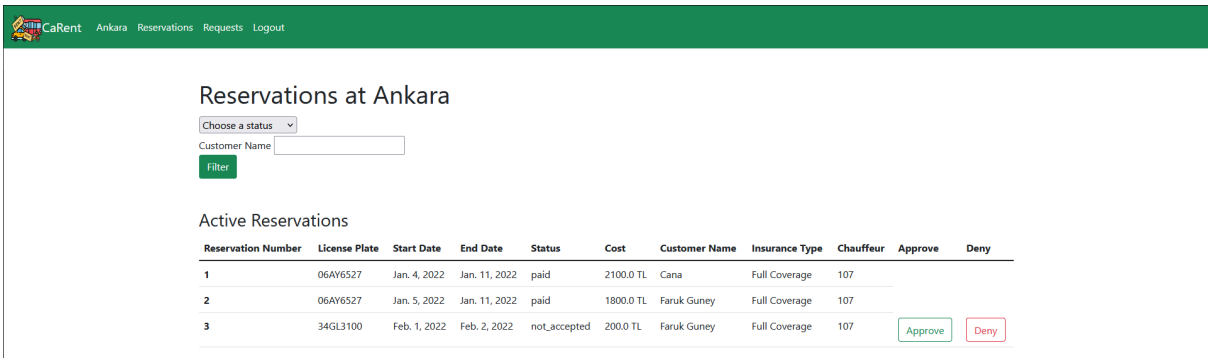
BRANCH EMPLOYEE



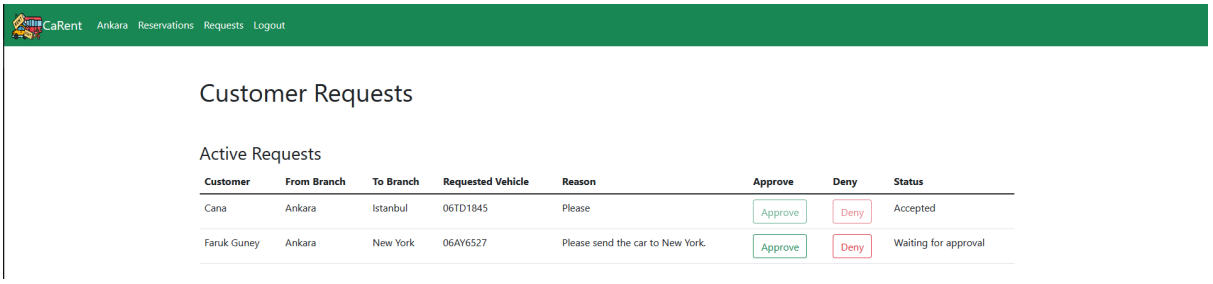
Branch employees can register to the system by entering their credentials and a brand id number to this form.



Upon entry the branch employees can see the vehicles in the branch they are working in. They can use the input fields to filter the result.

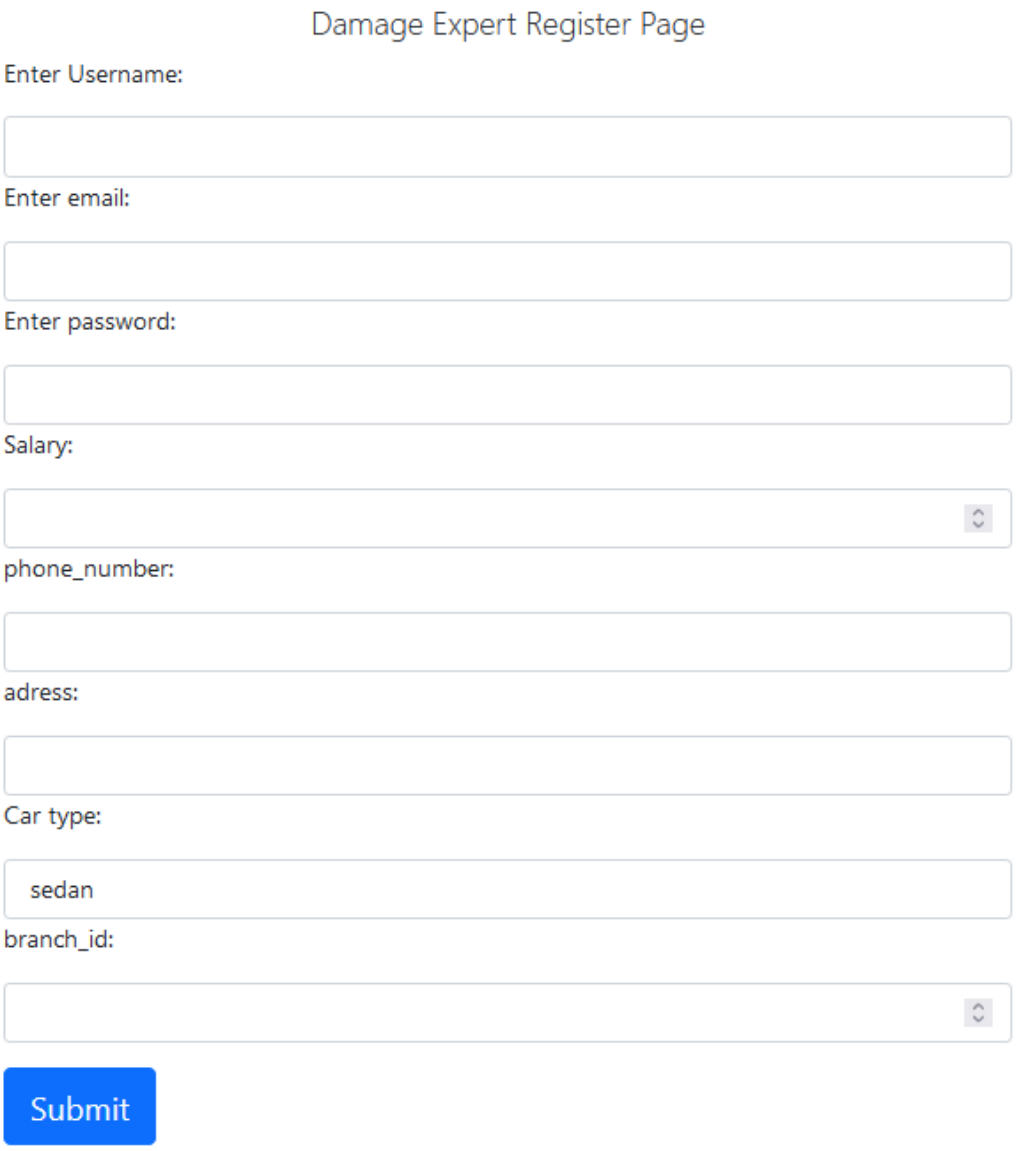


In the reservations page the branch employees can see previous and current rent applications. If the reservation is still not accepted the branch employee has the opportunity to approve or deny the reservation.

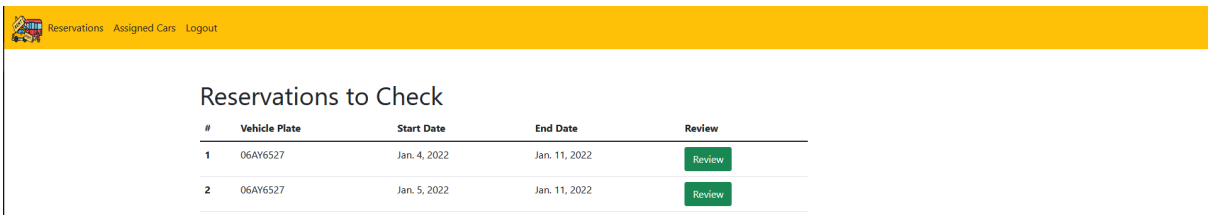


In the requests page branch employees are able to see the requests that are created by customers. The branch employees have the opportunity to approve or deny the request. If the request is accepted the vehicle is moved to the branch that is displayed in the “To Branch” column.

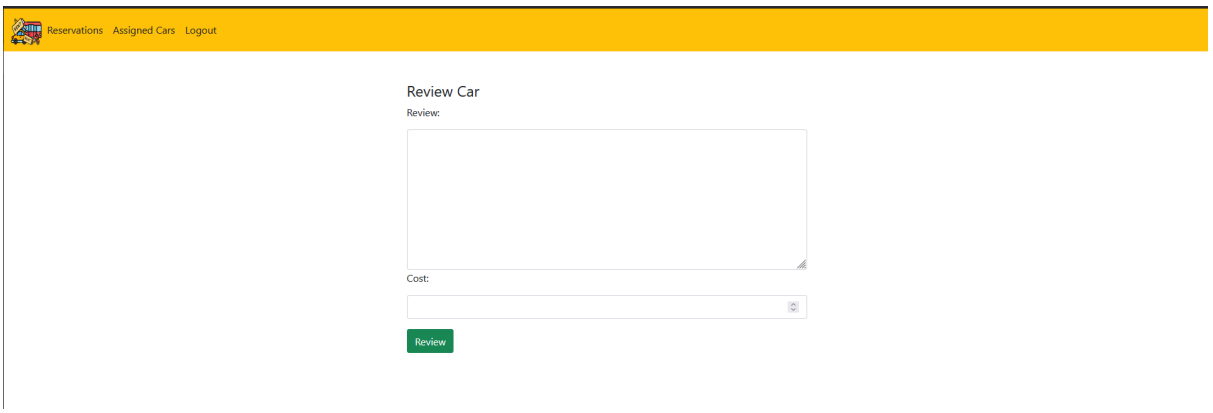
DAMAGE EXPERTISE



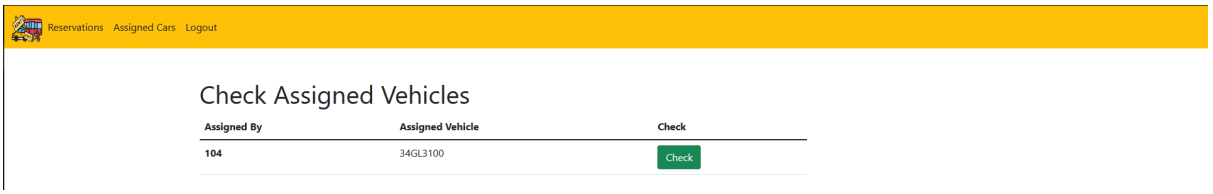
A damage expert can sign up to the system by entering their credentials, the car type they are expert at and the id of the branch they are working at.



In the reservation page damage experts can see the completed reservations and review the returned vehicle to confirm if there is an error.

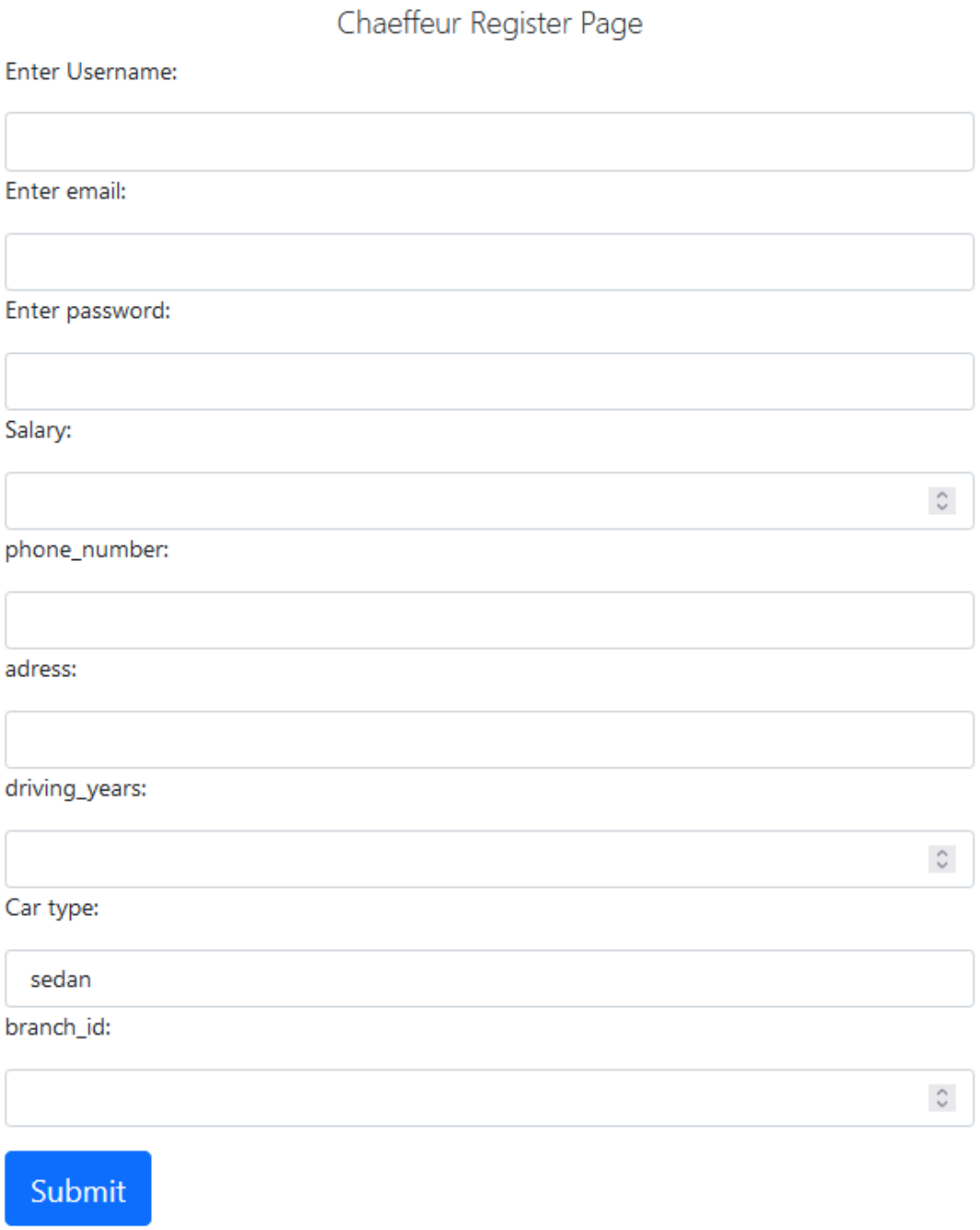


After clicking the review button on the previous page the damage expert is redirected to this page. They can enter a review for the car they have checked and enter the cost that is required to fix the vehicle.

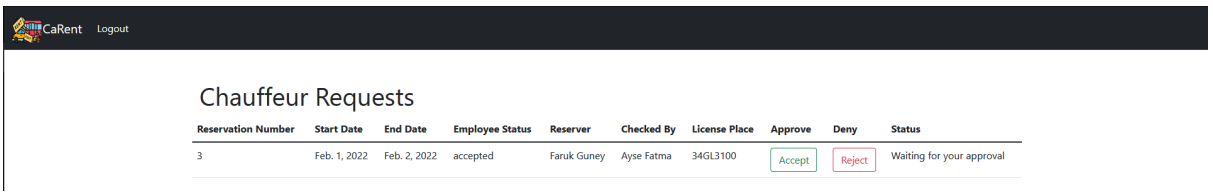


The damage expert can see the list of vehicles that are assigned to him by a manager to check. After they have checked the vehicle they can press the Check button to remove the assignment.

CHAUFFEUR

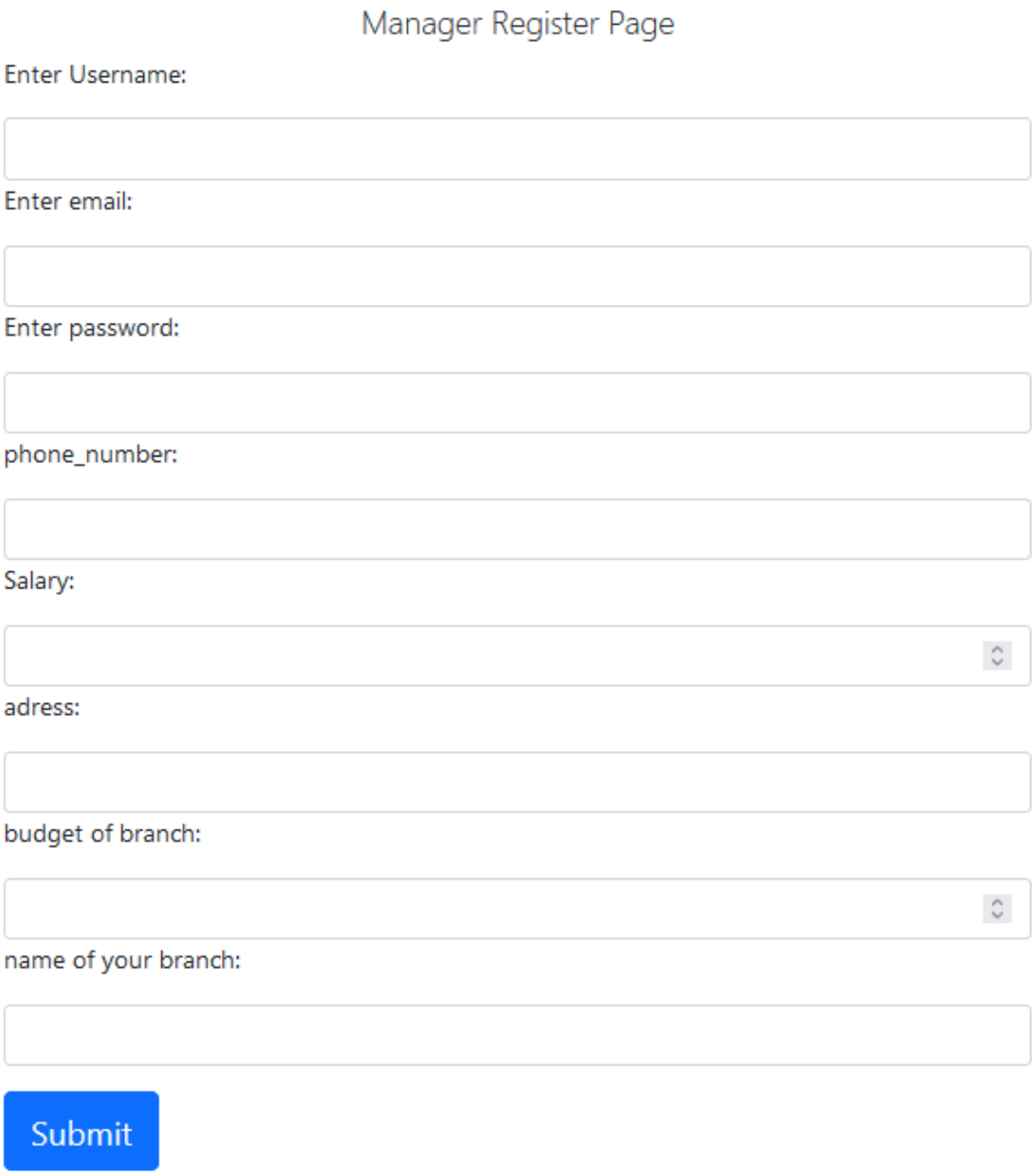


Chauffeurs can sign up to the system by entering their credentials, their driving years of experience, their preferred car type and the id of the branch they are working at.

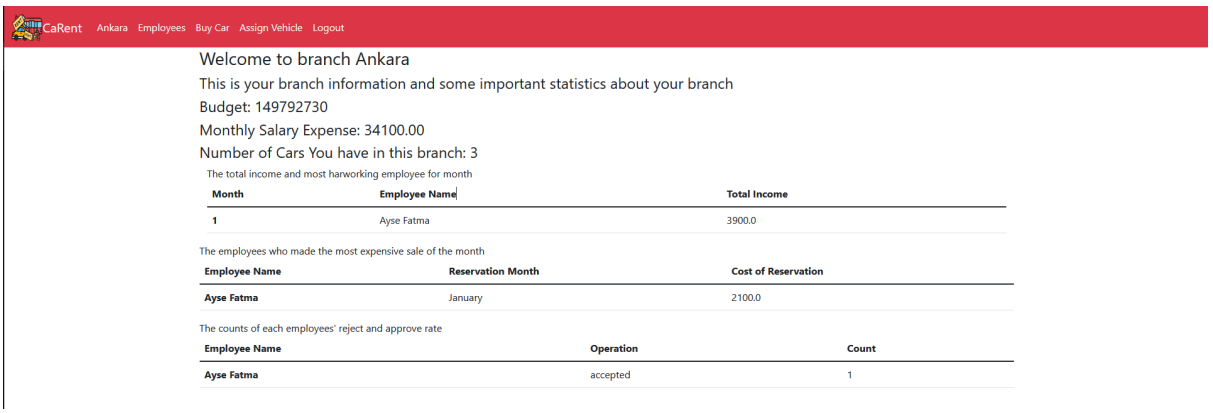


Chauffeurs can see the list of reservations in which the user has selected them as the chauffeur and they can accept or reject this request by clicking the appropriate buttons.

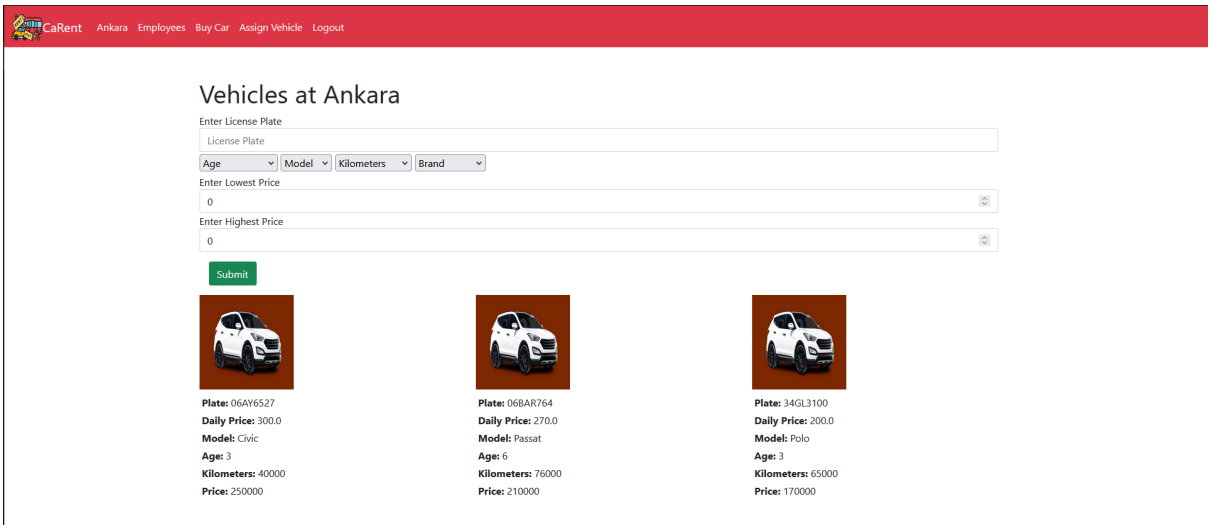
MANAGER



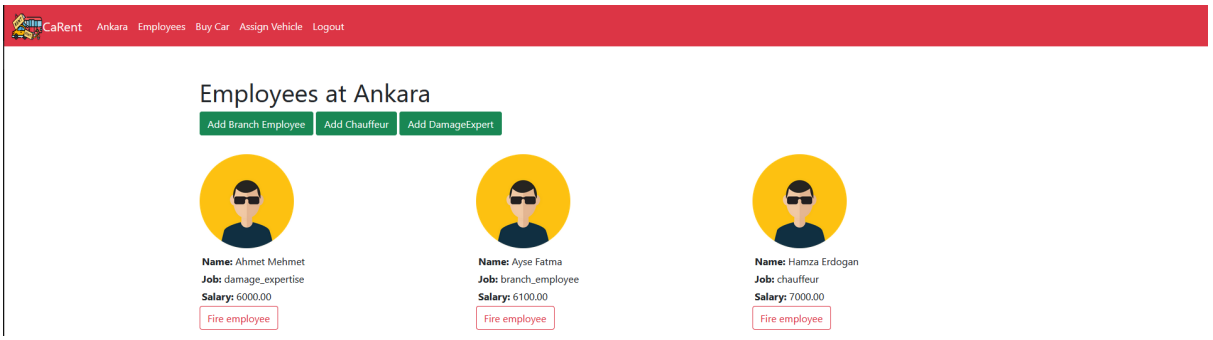
Managers can sign up to the system by entering their credentials, salary and information about the branch they will be managing. In the system every branch has only one manager so, the branch is created at the same time as its manager.



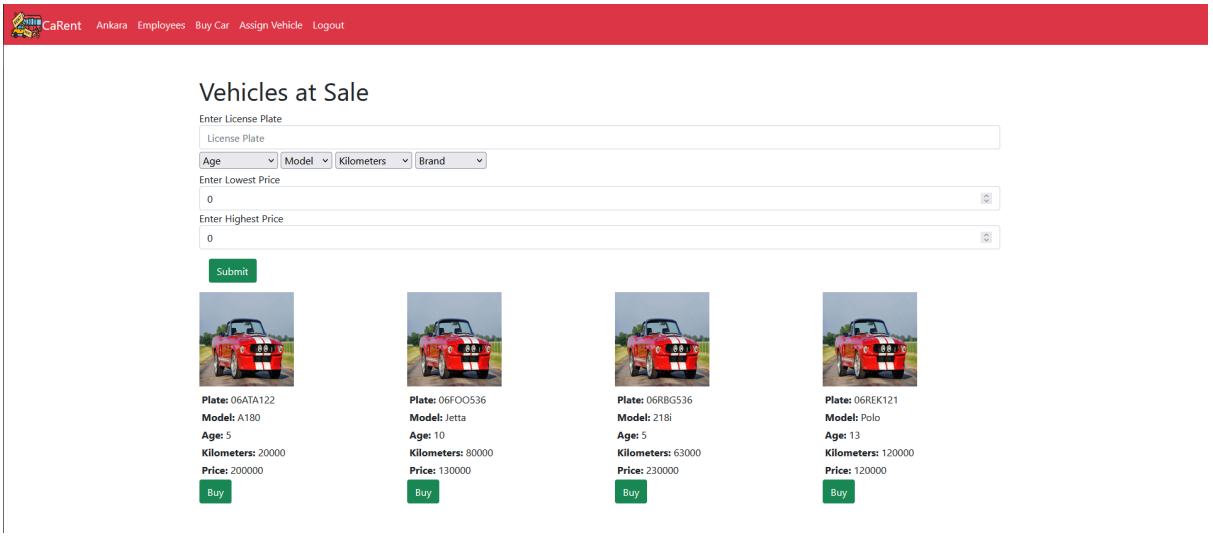
In the home page managers can see statistics and important data that belong to their branch.



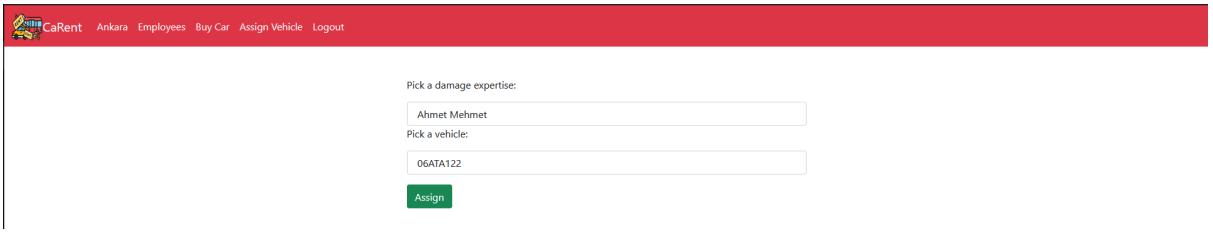
In the branch page the manager can view the cars that the branch has and filter them by filling input forms.



In the employee page managers can see their employees, hire new employees and fire current employees.



The Buy Car page shows vehicles that are on sale. The manager can buy any of these vehicles if their budget allows them to. They can also use the filtering system to specify the results.



In the Assign Vehicle page the manager can select a damage expert and assign them to a vehicle for them to take care of the vehicle.