



Bilkent University
Department of Computer Engineering

CS353 Database Systems

Group 13
Project Final Report
Car Rental System
04.01.2022

Group Members:

Akmuhammet Ashyralyyev - 21801347
Berke Ceran - 21703920
Hakan Gülcü - 21702275
Sila Saraoglu - 21803313

Git Link: <https://berkeceran.github.io/CarRentalSystem/>

Instructor: Özgür Ulusoy
Teaching Assistant: Mustafa Can Çavdar

1. DESCRIPTION OF APPLICATION SYSTEM	3
2. FINAL E-R MODEL.....	4
2.1. FINAL E-R MODEL	4
2.2. CHANGES.....	5
3. LIST OF TABLES – RELATION OF SCHEMAS	6
4. IMPLEMENTATION DETAILS.....	10
5. ADVANCED DB FEATURES.....	11
6. USER’S MANUAL.....	13
6.1 MANAGER’S PAGE	13
6.2 EMPLOYEE’S PAGE	13
6.3 CUSTOMER’S PAGE.....	13
6.4 LOGIN	14

Table of Contents

1. Description of Application System

This project is an Online Car Rental System application that functions over the web and database. It aims to bring both gallerist and customers in the same platforms. The company can provide various options such as cars and vehicles to the customer so they can make a request for them. Also, withing the system, there is an employee and manager type of users that works for company and additionally drivers, but they do not have much functionality over the system yet. Users can sign up to the system after they provide the information that have been asked to them. On the other hand, employees are introduced to the system by manager, meaning that the branch's manager will load the information of the employee so he/she will be able to login to the system. The manager is introduced to the system by administrator. The administration system allows to create branches of company and create company itself which are additional functionalities of him/her.

After logging in to the system, manager will be directed to the main page where he/she will be able to see the cars and employees that belongs to his/her Branch. He/she was provided a 2 button which enables the creation of employee and car there, as well. Also, he/she can see the most successful and least successful employees in the main page in terms of the rental cars that they have sold.

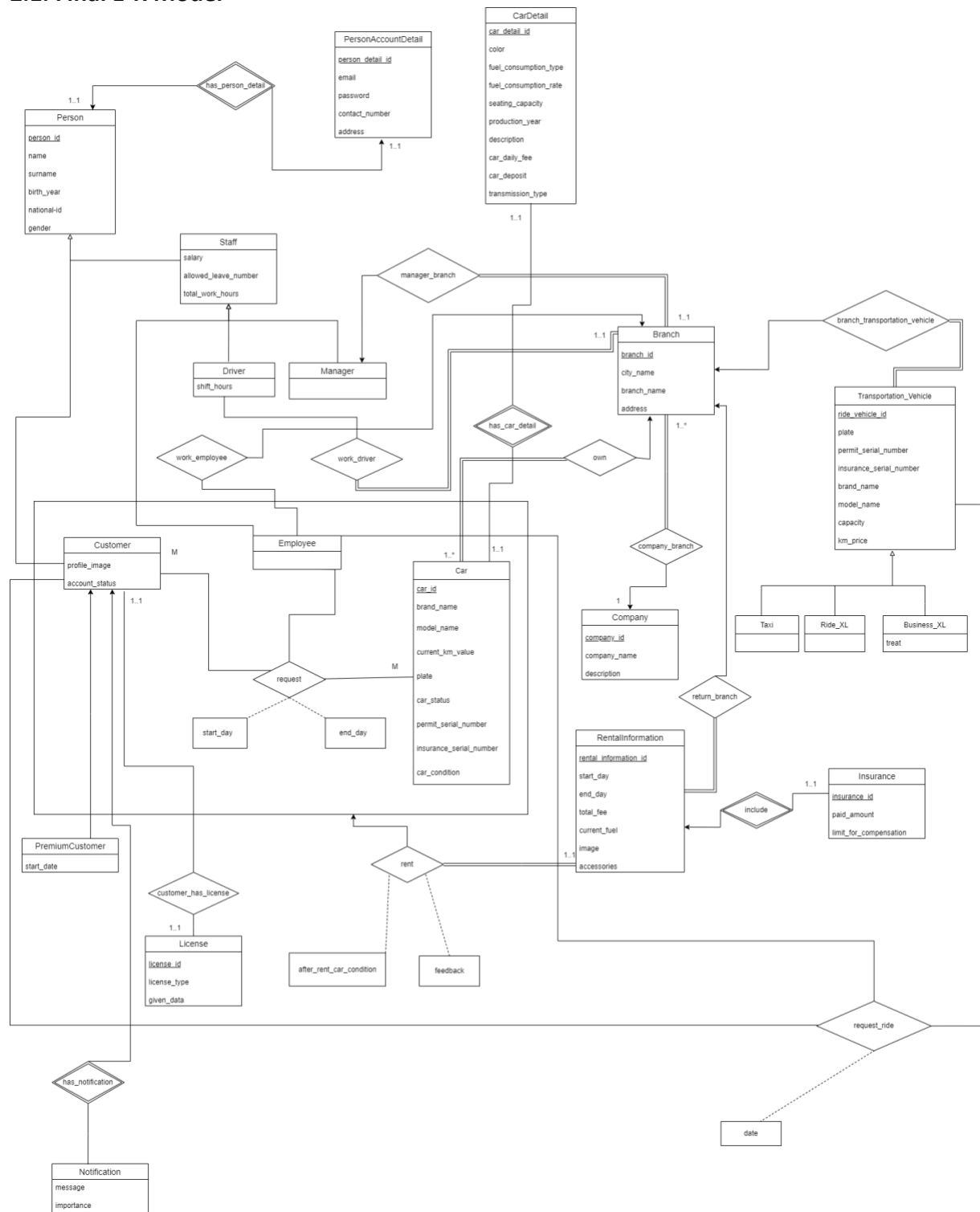
The customer will direct to the search car page where he will be able to search rental cars and transporters. He/she is provided with filtering options which are branch, color, prices both min and max, dates for rental and brands. The dates and branch are must to fill in order to search cars while others are optional and allowed to enter multiple options. According to the items, the cars that satisfies all the categories is displayed to user after searching such car within the system. The filtering items are retrieved from the cars within the system meaning that at least 1 car would be satisficing the selected item (Price is excluded). Also, the cars that are not available at those date that employee enters will not be visible to users. The transportation vehicles can be search according to the branch and dates. After deciding the car and clicking the 'request car' car button, the system automatically creates and request for that car if it is not already rented. The employee that will be processing this request is determent by the random assignment within the same branch. Also, user is informed of the result of each request via Notification whether it is approved or decline by employee. The approved request will be visible in the customer's profile page with old rental information's. In that page, user is also allowed to update its profile.

The last amin actor in the system, which is employee, is directed to his/her own main page where he/she will be able to see the request information that is being automatically assigned by the system. He/she will be able to approve or decline the request after the evaluation and in case of approve, the system creates the history of rental and deleting the request. In case of decline, the request only being deleted from the system. In both cases, uses is being received Notification about the process.

The last functionality about this system is related to the transports which are like the Uber like applications. The request is done for the whole day and driver is assigned automatically to the user if there is any available at that day. If there are no available drivers at that date, then system stops the execution and informs the customer with proper message.

2. Final E-R Model

2.1. Final E-R Model



2.2. Changes

- The foreign key constraints such as Employee(employee_id) table referencing Person(person_id) instead of Staff(person_id) are fixed.
- The foreign key "license_id" is removed from the Customer table and instead customer_id is added as a foreign key in License table for implementation purposes; nevertheless, since the relationship between them is one to one and total participation on both sides, this change does not violate any rules.
- RentalInformation needs to have an attribute indicating the returned branch, thus there is a new relation one to many between RentalInformation and Branch table.
- The entities included in aggregation redrawn in the block of the aggregation. They are no longer outside of the block.
- Drawing the employee entity in request_ride relation is fixed.

3. List of Tables – Relation of Schemas

3.1. Person

Model: Person(person_id, name, surname, birth_date, national_id, gender, email, password, address, contact_number)

Candidate Keys: {(person_id), (national_id), (email)}

Primary Key: (person_id)

3.2. Staff

Model: Staff(person_id, salary, allowed_leave_number, work_hours_per_day)

Foreign Key: person_id references Person(person_id)

Candidate Keys: {(person_id)}

Primary Key: (person_id)

3.3. Employee

Model: Employee(employee_id, branch_id)

Foreign Key: employee_id references Staff(person_id)

Foreign Key: branch_id references Branch(branch_id)

Candidate Keys: {(employee_id)}

Primary Key: (employee_id)

3.4. Driver

Model: Driver(person_id, shift_hours, branch_id)

Foreign Key: person_id references Staff(person_id)

Foreign Key: branch_id references Branch(branch_id)

Candidate Keys: {(person_id)}

Primary Key: (person_id)

3.5. Manager

Model: Manager(manager_id)

Foreign Key: manager_id references Staff(person_id)

Candidate Keys: {(manager_id)}

Primary Key: (manager_id)

3.6. Customer

Model: Customer(customer_id, profile_image, account_status)

Foreign Key: customer_id references Person(person_id)

Candidate Keys: {(customer_id)}

Primary Key: (customer_id)

3.7. PremiumCustomer

Model: PremiumCustomer(customer_id, start_date)

Foreign Key: person_id references Customer(customer_id)

Candidate Keys: {(customer_id)}

Primary Key: (customer_id)

3.8. Car

Model: Car(car_id, brand_name, model_name, current_km_value, plate, car_status, permit_serial_number, insurance_serial_number, car_condition, branch_id)

Foreign Key: branch_id references Branch(branch_id)

Candidate Keys: {(car_id), (plate), (permit_serial_number), (insurance_serial_number)}

Primary Key: (car_id)

3.9. CarDetail

Model: CarDetail(car_id, car_detail_id, color, fuel_consumption_type, fuel_consumption_rate, seating_capacity, production_year, description, car_daily_fee, car_deposit, transmission_type)

Foreign Key: car_id references Car(car_id)

Candidate Keys: {(car_id, car_detail_id)}

Primary Key: (car_id, car_detail_id)

3.10. Branch

Model: Branch(branch_id, city, branch_name, address, manager_id, company_id)

Foreign Key: manager_id references Manager(person_id)

Foreign Key: company_id references Company(company_id)

Candidate Keys: {(branch_id)}

Primary Key: (branch_id)

3.11. Company

Model: Company(company_id, company_name, description)

Candidate Keys: {(company_id), (company_name)}

Primary Key: (company_id)

3.12. TransportationVehicle

Model: TransportationVehicle(ride_vehicle_id, plate, permit_serial_number, insurance_serial_number, brand_name, model_name, capacity, km_price, branch_id)

Foreign Key: branch_id references Branch(branch_id)

Candidate Keys: {(ride_vehicle_id), (plate), (permit_serial_number), (insurance_serial_number)}

Primary Key: (ride_vehicle_id)

3.13. Taxi

Model: Taxi(ride_vehicle_id)

Foreign Key: ride_vehicle_id references to TransportationVehicle(ride_vehicle_id)
Candidate Keys: {(ride_vehicle_id)}
Primary Key: (ride_vehicle_id)

3.14. Ride_XL

Model: Ride_XL(ride_vehicle_id)
Foreign Key: ride_vehicle_id references to TransportationVehicle(ride_vehicle_id)
Candidate Keys: {(ride_vehicle_id)}
Primary Key: (ride_vehicle_id)

3.15. Business_XL

Model: Business_XL(ride_vehicle_id, treat)
Foreign Key: ride_vehicle_id references to TransportationVehicle(ride_vehicle_id)
Candidate Keys: {(ride_vehicle_id)}
Primary Key: (ride_vehicle_id)

3.16. License

Model: License(license_id, customer_id, license_type, given_date)
Foreign Key: customer_id references to Customer(customer_id)
Candidate Keys: {(license_id)}
Primary Key: (license_id)

3.17. RentalInformation

Model: RentalInformation(rental_information_id, start_day, end_day, total_fee, current_fuel, image, accessories, car_id, customer_id, employee_id, branch_id, after_car_rent_condition, feedback)
Foreign Key: car_id references Car(car_id)
Foreign Key: customer_id references Customer(customer_id)
Foreign Key: employee_id references Employee(employee_id)
Foreign Key: branch_id references Branch(branch_id)
Candidate Keys: {(rental_information_id)}
Primary Key: (rental_information_id)

3.18. Insurance

Model: Insurance(insurance_id, paid_amount, limit_for_compensation, rental_information_id)
Foreign Key: rental_information_id references RentalInformation(rental_information_id)
Candidate Keys: {(insurance_id)}
Primary Key: (insurance_id)

3.19. request

Model: request(employee_id, customer_id, car_id, branch_id, start_day, end_day)
Foreign Key: employee_id references Employee(employee_id)

Foreign Key: customer_id references Customer(customer_id)

Foreign Key: car_id references Car(car_id)

Foreign Key: branch_id references Branch(branch_id)

Candidate Keys: {(employee_id, customer_id, car_id, branch_id)}

Primary Key: (employee_id, customer_id, car_id, branch_id)

3.20. request_ride

Model: request_ride(employee_id, customer_id, ride_vehicle_id, date, start_time, end_time)

Foreign Key: employee_id references Employee(employee_id)

Foreign Key: customer_id references Customer(customer_id)

Foreign Key: ride_vehicle_id references TransportationVehicle(ride_vehicle_id)

Candidate Keys: {(employee_id, customer_id, ride_vehicle_id)}

Primary Key: (employee_id, customer_id, ride_vehicle_id)

3.21. Notification

Model: Notification(person_id, notification_id, message, send_date)

Foreign Key: person_id references Person(person_id)

Candidate Keys: {(person_id, notification_id)}

Primary Key: (person_id, notification_id)

4. Implementation Details

During the implementation of our System, we have used React for the front-end and Python for the backend. The system is running on local storage, so we didn't use any other services for hosting and running. On the other hand, we have mainly utilized AWS Lambda, API Gateway and DB instances of AWS services. The database is established with MySQL and the AWS is chosen for the storing since we were already using the other services of the system. For the API, we have used REST and connect to the lambda functions at the database. We have chosen AWS services because of the following reasons: it was free up to 1 million request and allows 5 GB of storage, it contains very well documented documentation, and it contains many functionalities within, so, we didn't have to search for other technologies. For the preference of python, we have chosen it since it is very easy to learn and provides good libraries for the backend services. The main library that we used in python was pymysql, which allows to communicate to the MySQL server and execute codes there. Our plan was also put our code to the AWS servers but since we didn't have enough time, we couldn't accomplish this. Also, we planned to use the Bilkent's database services in Dijkstra but since it requires VPN services, we couldn't redirect the traffic from AWS lambda services to school's database. Hence, we decided to establish our database in AWS servers.

There were 2 main problems we had suffered a lot. One of them was solved after the research that took 1 day. The problem was related to the AWS services that doesn't allow to be communicated through HTTP over cross-region until the enabling of the CORS. Since our database was established in Frankfurt, this issue was important to us. We have enabled the whole CORS in each API and solved the problem. The second problem was related to the Date instance of database system versus Python interpretation. For a long time, we couldn't understand the reason why the objects from database did not pass-through REST. We found that the retrieval from database of Date objects was not in string format and API cannot fit this to the link. We found the solution by converting to the Date objects to string before passing it to the return statements (used `str()` in Python).

The creation of tables was done in local IDE environment, which is Anaconda's Spyder. Berke and Hakan created the python functions that connects to the database and creates tables of our system. For the rest of the project, they were responsible for the front-end implementation while Akmuhammet and Sila was responsible since rest of the backend code. Both teams worked separately from each other at the beginning of the implementation but after the establishment of main lines, both teams gathered and worked as a team, especially during connecting the backend with frontend. Akmuhammet and Sila was quite familiar to each other's code, hence, it allows them to continue to work if one of them was not available. The same collaboration was done between Berke and Hakan. But Hakan was also responsible for the CSS of the pages. The REST implementation as well as procedure, view, secondary indices was also done with collaboration between Akmuhammet and Sila.

5. Advanced DB Features

The report in our project finds the best and worst employee list in terms of their rental count and these insights are shown to the manager.

The SQL statements for the best employee list is the following:

```
SELECT Max(count) AS maxcount
FROM (
    SELECT      employee_id, Count(*) AS count
    FROM        employee natural left JOIN rentalinformation
    WHERE       branch_id = 2
    GROUP BY    employee_id ) AS count_rental_of_employees;

SELECT name, surname
FROM (
    SELECT      employee_id, Count(*) AS count
    FROM        employee natural left JOIN rentalinformation
    WHERE       branch_id = 2
    GROUP BY    employee_id ) AS count_rental_of_employees JOIN person
WHERE person_id = employee_id
AND    count = 38;
```

The SQL statements for the worst employee list is the following:

```
SELECT Min(count) AS mincount
FROM (
    SELECT      employee_id, Count(customer_id) AS count
    FROM        employee natural JOIN rentalinformation
    WHERE       branch_id = 2
    GROUP BY    employee_id ) AS count_rental_of_employees;

SELECT name, surname
FROM (
    SELECT      employee_id, Count(customer_id) AS count
    FROM        employee natural left JOIN rentalinformation
    WHERE       branch_id = 2
    GROUP BY    employee_id ) AS count_rental_of_employees
JOIN person
WHERE person_id = employee_id
AND    count = 38;
```

The MySQL query that finds the cars with given parameters is the following (Nested Queries and usage of BETWEEN):

```
SELECT car_id,
       brand_name,
       model_name,
       current_km_value,
       car_status,
       car_condition,
       color,
       fuel_consumption_type,
       fuel_consumption_rate,
       seating_capacity,
       production_year,
       description,
       car_daily_fee,
       car_deposit,
       transmission_type
FROM (SELECT *
      FROM car
      WHERE car_id NOT IN ((SELECT DISTINCT car_id
                           FROM rentalinformation
                           WHERE ( 2021 - 02 - 20 BETWEEN
                                start_day AND end_day )
                                OR ( 2021 - 02 - 25 BETWEEN
                                start_day AND end_day ))
                           UNION
                           (SELECT DISTINCT car_id
```

```

FROM request
WHERE ( 2021 - 02 - 20 BETWEEN
        start_day AND end_day )
        OR ( 2021 - 02 - 25 BETWEEN
        start_day AND end_day )))
AND ( brand_name = 'toyota'
      OR brand_name = 'tesla' )
AND branch_id = (SELECT branch_id
                  FROM branch
                  WHERE branch_id = ''
                  LIMIT 1) AS BrandDateCars
NATURAL JOIN (SELECT *
              FROM cardetail
              WHERE ( color = 'blue'
                     OR color = 'orange'
                     OR color = 'yellow' )
              AND car_daily_fee > 10
              AND car_daily_fee < 1000) AS ColorFeeCars;

```

The view of Car NATURAL JOIN CarDetail. We have implemented this since we were using this operation and extracting the specific parameters many times, but it is very costly. So, we have created view that contains only required columns. It also provides encapsulation of data since license plate, insurance etc. is confidential. This view is used in manager's page where he can see the branch's cars

```

SELECT *
FROM car_detail
WHERE branch_id = 2;

```

The secondary indices are used for the email for the Person, for plate for Car and for customer_id, employee_id, branch_id, car_id for RentalInformation. For the email, we have written our search queries according to the email since it was unique, but since it is not primary key, the searching was not done efficiently. The same thing was for the plate and rentals. So, we decided to create indices on them. Since we did not have large data in storage, it did not affect too much, but it was good practice.

```

CREATE INDEX email_index ON Person(email);
CREATE INDEX plate_index ON Car(plate);
CREATE INDEX rental_index ON RentalInformation(customer_id, employee_id, branch_id, car_id);

```

We have implemented a stored procedure which we use common in our project. We used a stored procedure which gets all the branches. The following is one of the calls to getAllBranches in our project.

```

...
with connection:
    with connection.cursor() as cursor:
        sql_stmt = 'CALL get_all_branches;'
        cursor.execute(sql_stmt)
        connection.commit()
        records = cursor.fetchall()
        for row in records:
            branches.append(row)
        getAllBranchesResponse['branches'] = branches
...

```

6. User's Manual

6.1 Manager's Page

Manager main page is the page that managers who are already registered to the system will see when they log in. The manager can see all the cars registered in his own branch on the left part of the page, all the transportation vehicles registered in the middle part, and all the employees and drivers working in his branch on the right. It can also display the best element and the worst element.

The Manager can see in detail all the cars registered in his branch, all registered transportation vehicles and all employees and drivers working in his branch. It can also display the best element and the worst element. Manager can add a new car, transportation vehicle, driver or employee. Each manager is authorized to see only the information related to his own branch, he cannot view or add other branches to these branches.

6.2 Employee's Page

Employee main page is the page that the employees who are already registered and logged into the system will see. Employee can view, approve or reject car rental requests made in their branch and assigned to it. Employee can create a form and rent a car with the lessor information and information about the car. He can see his personal information and working hours, as well as pre-approved rentals. It does not have access to any information about other employees.

6.3 Customer's Page

When customers log in, the first thing they see is the search car page. On this page, customers can search for cars in terms of colors, models, price ranges, and branches of the cities they want by performing various filtering processes. According to these filtering processes, various cars come to the user. After the cars are filtered, users can request the car they want by clicking the request car button, and this request is displayed on the employee's main page. If the employee approves this request, that is, clicks the accept button, the customer has rented the car, but if he does not approve, that is, if he clicks the decline button, the user cannot rent the car. After the user requests a car, he/she cannot request the same car again, but they can request other cars if the cars are available. When customers click on the switch button in the upper right corner, the transportation vehicle request side is opened, they can rent transportation instead of a car. The filtering feature on this page is by choosing various branches of various cities and appropriate day intervals. When they click the request transportation vehicle button, if there is an available driver employee, users can rent transportation vehicle, but if there is no available driver, they cannot do the rental process. Finally, users can view their profiles by clicking the profile page button in the upper right corner. On this page, the user can change his password or update his address as an example

related to himself. In addition, on this page, the user can see all the car or vehicle rentals he has made before.

6.4 Login





Employee, manager or customer can log in from a single login page. The redirected page changes according to the membership type. Login with email and password information.


6.5 Signup

Signup can only be done by customers. It is not possible to register with a previously registered e-mail. Employees are created by managers and managers are created by admins, so they cannot signup.


7. Additional


Car Rental System









Sign Up







Login


Car Rental System




Login














Birth Date

















Licence Given Date

Sign Up

Branch

RegencySubsea - Istanbul

Brand

☐ FORD
 ☐ TOYOTA
 ☐ MERCEDES
 ☐ AUDI

Start Date

04/01/2022

End Date

11/01/2022

Price

From:

Up To:

Color

☐ GREY
 ☐ ORANGE
 ☐ YELLOW
 ☐ BLACK

Search

Car Brand: **ford**

Car Model: **focus**

Color: **grey**

Fee: 200

Deposit: 100

Seating Capacity: 4

Current Km: 2000

Fuel Consumption Type: **gas**

Fuel Consumption Rate: **better**

Description: **ass**

Request Car

Car Brand: **toyota**

Car Model: **focus**

Color: **orange**

Fee: 200

Deposit: 100

Seating Capacity: 6

Current Km: 57000

Fuel Consumption Type: **gas**

Fuel Consumption Rate: **better**

Description: **ass**

Request Car

Car Brand: **ford**

Car Model: **lola**

Color: **yellow**

Fee: 120

Deposit: 100

Seating Capacity: 4

Current Km: 100

Fuel Consumption Type: **benzin**

Fuel Consumption Rate: **ok**

Description: **landas**

Request Car

Car Brand: **ford**

Car Model: **abc**

Color: **black**

Fee: 100

Deposit: 100

Seating Capacity: 5

Current Km: 100

Fuel Consumption Type: **low**

Request Car

Rent / Transporter

☐

Branch

RegencySubsea - Istanbul

Brand

☒ FORD
 ☐ TOYOTA
 ☒ MERCEDES
 ☐ AUDI

Start Date

04/01/2022

End Date

16/01/2022

Price

From:

1

Up To:

1000

Color

☐ GREY
 ☐ ORANGE
 ☒ YELLOW
 ☐ BLACK

Search

Car Brand: **ford**

Car Model: **lola**

Color: **yellow**

Fee: 120

Deposit: 100

Seating Capacity: 4

Current Km: 100

Fuel Consumption Type: **benzin**

Fuel Consumption Rate: **ok**

Description: **landas**

Request Car

Car Brand: **Mercedes**

Car Model: **A200**

Color: **yellow**

Fee: 180

Deposit: 330

Seating Capacity: 5

Current Km: 50

Fuel Consumption Type: 100

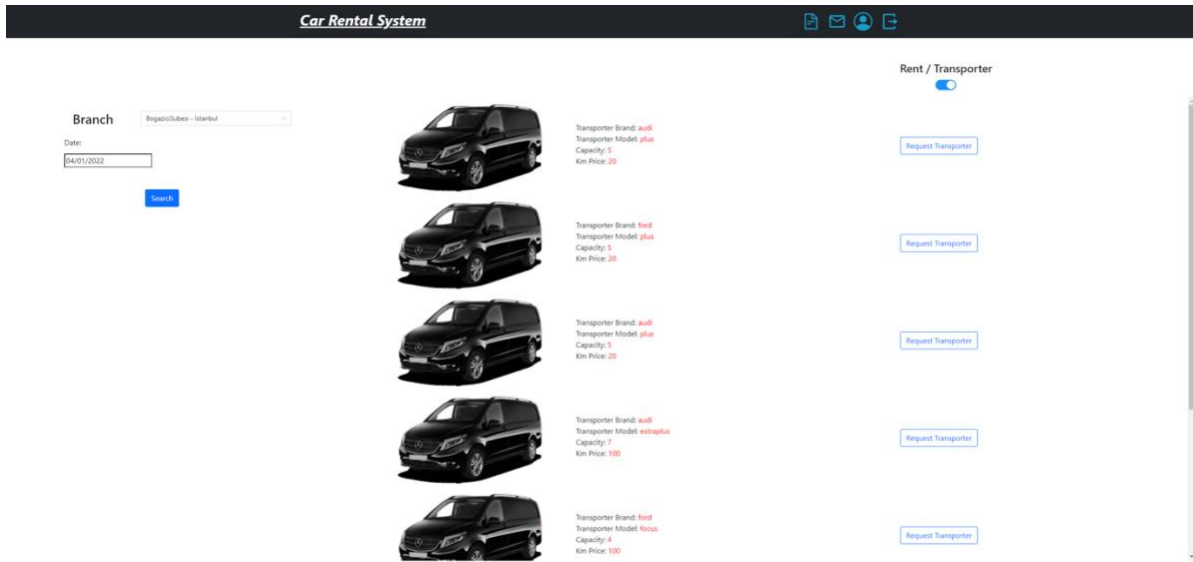
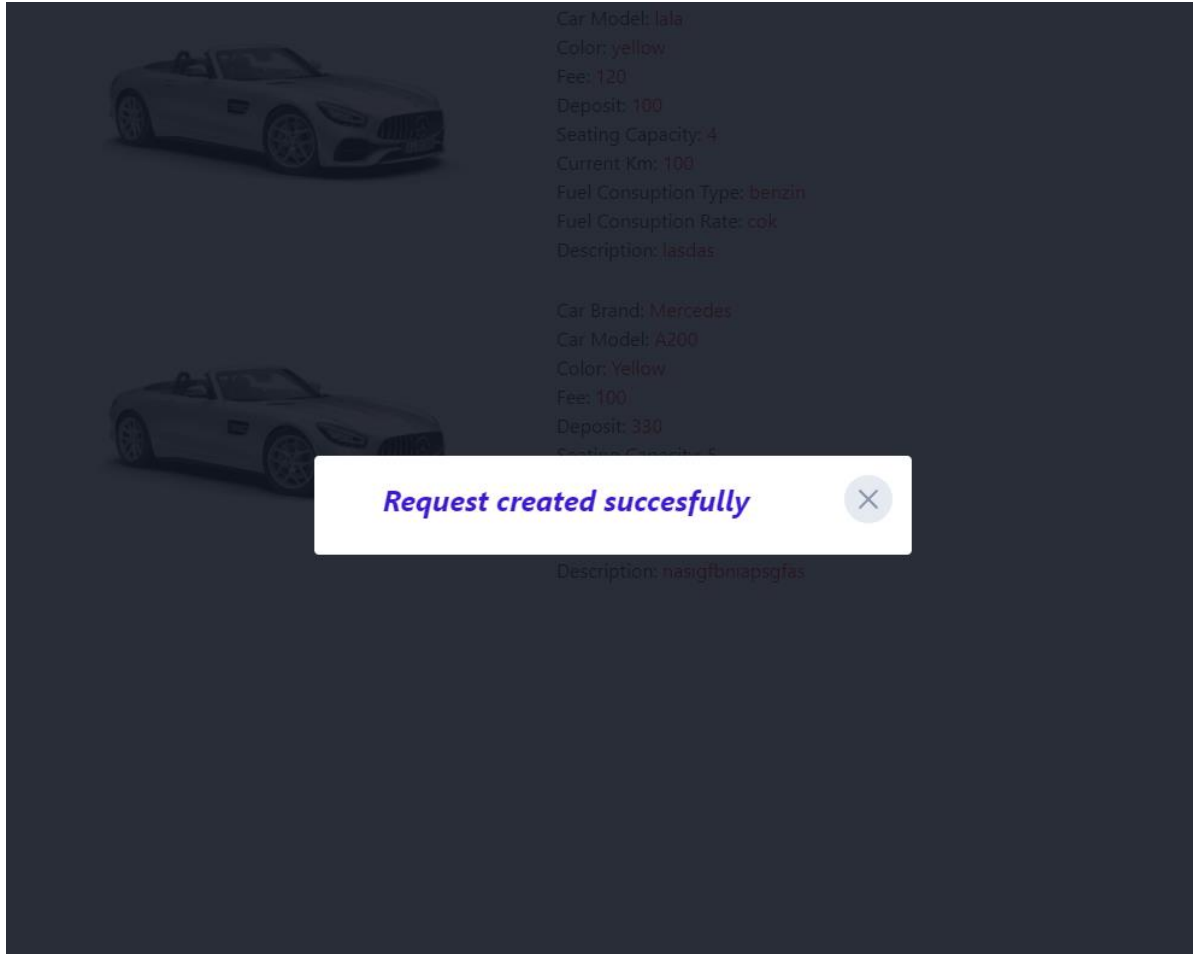
Fuel Consumption Rate: 214

Description: **naug/bunapagla**

Request Car

Rent / Transporter

☐



Car Rental System

Remove Image

Change Image

Old Password

New Password

Change Password

hakan

goku

hakan3153@gmail.com

classic

Birth Day

16/01/1999

A

Licence Given Date

22/01/2013

54362673

1605608

Save Changes

Previous Rental Information

Car Brand opel

Car Model corsa

Total Fee

Start Day 2022-01-04

End Day 2022-01-26

Car Rental Condition

Feedback

Branch Name etimesgut subesi

Car Brand mercedes

Car Model c120

Total Fee

Start Day 2025-01-19

End Day 2025-01-28

Car Rental Condition

Feedback

Branch Name etimesgut subesi

Car Brand mercedes

Car Model A200

Total Fee

Start Day 2022-01-04

End Day 2022-01-11

Car Rental Condition

Feedback

Branch Name etimesgut subesi

Car Rental System

Add Car Create Employee

CARS

Car Brand ford

Car Model mustang

Car Color grey

Year 200

Current Rate 2000

Permit Serial Number 2

Insurance Serial Number 2

Fuel Consumption Rate literes

Fuel Consumption Type gas

Car Condition good

Car Value 1000

Car Deprecate 100

Car Description nice

Car Price 100000000

Car Production Year 1990

Car Seating Capacity 1

Transportation Type city

Car Brand toyota

TRANSPORTATION VEHICLES

Plate 980000001

Permit Serial Number 9

Insurance Serial Number 10

Brand Name plus

Model Name plus

Capacity 1

Plate 980000002

Permit Serial Number 1

Insurance Serial Number 2

Brand Name ford

Model Name plus

Capacity 5

EMPLOYEES

Best Employee: hakan abc

Worst Employees: beko cero ,bere bark ,sila tuncay ,

berke

cerah

berke@gmail.com

Employee

sila

tuncay

18

Car Rental System

Remove Image | Change Image

Brand Name

Model

Color

Current Km

Plate

Current Fee

Permit Serial Number

Insurance Serial Numbe

Car Condition

BogaziciSubesi

Fuel Consumption Type

Fuel Consumption Rate

Seating Capacity

Production Year

Description

Deposit

Transmission Type

Create Car

Create Transportation Vehicle

Car Rental System

Remove Image | Change Image

Brand Name

Model

Plate

Permit Serial Number

Insurance Serial Numbe

BogaziciSubesi

Kilometer Price

Seating Capacity

Treat

Business-XL

Create Transportation Vehicle

Create Car

Car Rental System

Remove Image | Change Image

Name

Surname

Email

Contact Number

National Id

Birth Day

01/04/2022

Password

Confirm Password

Female Male

BogaziciSubesi

Address

Salary

Allowed Leave Number

Work Hours Per Day

Employee

Create Employee



Customer Name: beko
Customer Surname : ale
Car Brand : ford
Car Model : focus
Start Date : 2022-01-04
End Date : 2022-01-11
Customer National ID : 12321321321
Customer License : A
Contact Number : 12321321321321321

Accept

Decline

ADD MANAGER

Name
Surname
National Id
Email
Password
Address
Contact Number
Salary
Allowed Leave Number
Work Hours Per Day
Add

ADD BRANCH

City
Branch Name
Address
Company Name
Manager Email
Add

CREATE COMPANY

Company Name
Description
Create