Car Rental System Project Design Report

CS 353 Database Systems

Project Design

Car Rental System

Group No 17

Emre Caniklioğlu - 21803577

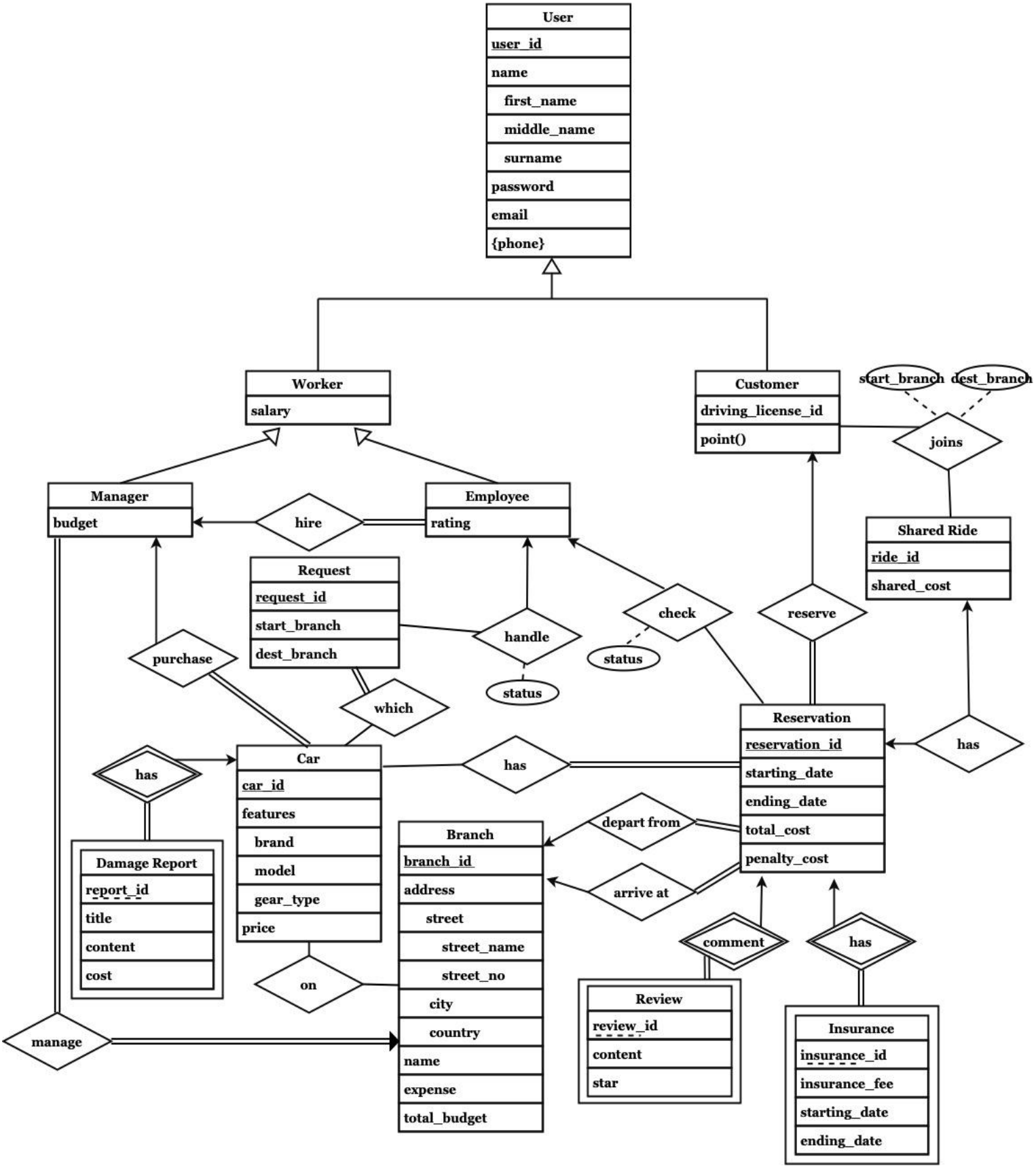
Celal Berke Can - 21702886

Ege Demirkırkan - 21802482

Berk Saltuk Yılmaz - 21903419

Table of Contents

1. REVISED ER MODEL



2. RELATION SCHEMAS

2.1 Users

Relational Model:

users(<u>user_id</u>, first_name, middle_name, surname, password, email)

Candidate Keys and Primary Key:

Candidate Keys: user_id, email
Primary Key: user_id

Functional Dependencies:

user_id -> first_name, middle_name, surname, password, email
email-> first_name, middle_name, surname, password, user_id

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS users CASCADE;
CREATE TABLE users(
   user_id SERIAL NOT NULL PRIMARY KEY,
   first_name VARCHAR(20) NOT NULL,
   middle_name VARCHAR(20),
   surname VARCHAR(20) NOT NULL,
   password VARCHAR(40) NOT NULL,
   email VARCHAR(50) NOT NULL UNIQUE
);
```

2.2 Worker

Relational Model:

Worker(<u>user_id</u>, salary)
FK: user_id references User(user_id)

Candidate Keys and Primary Key:

Candidate Key: user_id
Primary Key: user_id

Functional Dependencies:

user_id -> salary

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS worker CASCADE;
CREATE TABLE worker(
user_id INT PRIMARY KEY,
salary FLOAT,
FOREIGN KEY(user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

2.2.1 Manager

Relational Model:

Manager(user_id, budget, branch_id)
FK: user_id references User(user_id)
FK: manages references Branch(branch_id)

Candidate Keys and Primary Key:

Candidate Keys: user_id, branch_id
Primary Key: user_id

Functional Dependencies:

user_id -> manages, budget
branch_id -> user_id, budget

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS manager CASCADE;
CREATE TABLE manager(
 user_id INT PRIMARY KEY,
 budget FLOAT,
 branch_id INT,
 FOREIGN KEY(user_id) REFERENCES users(user_id) ON DELETE CASCADE,
 FOREIGN KEY(branch_id) REFERENCES branch(branch_id) ON DELETE CASCADE
);
```

2.2.2 Employee

Relational Model:

Employee(user_id, rating, manager_id)
FK: user_id references User(user_id)
FK: manager_id references Manager(user_id)

Candidate Keys and Primary Key:

Candidate Keys: user_id
Primary Key: user_id

Functional Dependencies:

user_id -> rating, manager_id

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS employee CASCADE;
CREATE TABLE employee(
 user_id INT PRIMARY KEY,
 rating FLOAT,
 manager_id INT,
 FOREIGN KEY(user_id) REFERENCES users(user_id) ON DELETE CASCADE,
 FOREIGN KEY(manager_id) REFERENCES manager(user_id) ON DELETE CASCADE
);
```

2.3 Customer

Relational Model:

Customer(<u>user_id</u>, driving_license_id, point)

Candidate Keys and Primary Key:

Candidate Keys: user_id, driving_license_id
Primary Key: user_id

Functional Dependencies:

user_id -> driving_license_id, point
driving_license_id -> user_id, point

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS customer CASCADE;
CREATE TABLE customer(
 user_id INT PRIMARY KEY,
 driving_license_id INT NOT NULL UNIQUE,
 point FLOAT,
 FOREIGN KEY(user_id) REFERENCES users(user_id) ON DELETE CASCADE
);
```

2.4 Reservation

Relational Model:

Reservation(reservation_id, starting_date, ending_date, total_cost, penalty_cost, status, customer_id, employee_id, depart_from, arrive_at)

FK: customer_id references Customer(user_id)
FK: employee_id references Employee(user_id)
FK: depart_from references Branch(branch_id)
FK: arrive_at references Branch(branch_id)

Candidate Keys and Primary Key:

Candidate Key: reservation_id, employee_id, customer_id
Primary Key: reservation_id

Functional Dependencies:

reservation_id -> starting_date, ending_date, total_cost, penalty_cost, status, customer_id, employee_id, depart_from, arrive_at

reserved_by -> starting_date, ending_date, total_cost, penalty_cost, status, reservation_id, employee_id, depart_from, arrive_at

checked_by -> starting_date, ending_date, total_cost, penalty_cost, status, customer_id, reservation_id, depart_from, arrive_at

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
CREATE TABLE reservation(
 reservation_id SERIAL NOT NULL PRIMARY KEY,
 starting_date DATE,
 ending_date DATE,
 total_cost FLOAT,
 penalty_cost FLOAT DEFAULT 0,
 status INT,
 customer_id INT NOT NULL,
```

```sql
 employee_id INT NOT NULL,
 depart_from INT,
 arrive_at INT,
 FOREIGN KEY (customer_id) REFERENCES customer(user_id),
 FOREIGN KEY (employee_id) REFERENCES employee(user_id),
 FOREIGN KEY (depart_from) REFERENCES branch(branch_id),
 FOREIGN KEY (arrive_at) REFERENCES branch(branch_id)
);
```

2.5 Car

Relational Model:

Car(car_id, brand, model, gear_type, price, manager_id)
FK: manager_id references Manager(user_id)

Candidate Keys and Primary Key:

Candidate Keys: car_id, manager_id
Primary Key: car_id

Functional Dependencies:

car_id -> brand, model, gear_type, price, manager_id

manager_id -> car_id, brand, model, gear_type, price,

on_branch -> manager_id, car_id, brand, model, gear_type, price

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS car CASCADE;
CREATE TABLE car(
 car_id SERIAL NOT NULL PRIMARY KEY ,
 brand VARCHAR(30),
 model VARCHAR(30),
 gear_type VARCHAR(10),
 price FLOAT,
 manager_id INT NOT NULL,
 FOREIGN KEY (manager_id) REFERENCES manager(user_id) ON DELETE CASCADE
);
```

2.6 Branch

Relational Model:

Branch(branch_id, branch_name, street_name, street_no, city, country, total_budget, expense)

Candidate Keys and Primary Key:

Candidate Key: branch_id
Primary Key: branch_id

Functional Dependencies:

branch_id -> branch_name, street_name, street_no, city, country, total_budget, expense

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS branch CASCADE;
CREATE TABLE branch(
 branch_id SERIAL NOT NULL PRIMARY KEY,
 branch_name VARCHAR(30),
 street_name VARCHAR(30),
 street_no VARCHAR(10),
 city VARCHAR(30),
 country VARCHAR(30) DEFAULT 'Turkey',
 total_budget INT,
 expense INT
);
```

2.7 Review

Relational Model:

Review(reservation_id, review_id, content, star)
FK: reservation_id references Reservation(reservation_id)

Candidate Keys and Primary Key:

Candidate Key: {reservation_id, review_id}
Primary Key: {reservation_id, review_id}

Functional Dependencies:

reservation_id, review_id -> content, star

Normal Form:

Third Normal Form (3NF)

SQL Declaration:

```
DROP TABLE IF EXISTS review CASCADE;
CREATE TABLE review(
 review_id SERIAL NOT NULL ,
 reservation_id INT NOT NULL ,
 content VARCHAR(255),
 star INT,
 PRIMARY KEY (review_id, reservation_id),
 FOREIGN KEY (reservation_id) REFERENCES reservation(reservation_id) ON DELETE
CASCADE
);
```

2.8 Request

Relational Model:

Request(<u>request_id</u>, start_branch, dest_branch, status, employee_id)
FK: start_branch references Branch(branch_id)
FK: dest_branch references Branch(branch_id)
FK: employee_id references Employee(employe_id))

Candidate Keys and Primary Key:

Candidate Key: request_id, employee_id
Primary Key: request_id

Functional Dependencies:

request_id -> start_branch, dest_branch, status, employee_id

employee_id -> request_id, start_branch, dest_branch, status

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS request CASCADE;
CREATE TABLE request(
   request_id SERIAL NOT NULL PRIMARY KEY,
   start_branch INT,
   dest_branch INT,
   employee_id INT,
   status INT ,
   FOREIGN KEY (start_branch) REFERENCES branch(branch_id) ON DELETE CASCADE,
   FOREIGN KEY (dest_branch) REFERENCES branch(branch_id) ON DELETE CASCADE,
   FOREIGN KEY (employee_id) REFERENCES employee(user_id) ON DELETE CASCADE
);
```

2.9 Insurance

Relational Model:

insurance(reservation_id, insurance_id, starting_date, ending_date, insurance_fee)
FK: reservation_id references Reservation(reservation_id)

Candidate Keys and Primary Key:

Candidate Key: {reservation_id, insurance_id}
Primary Key: {reservation_id, insurance_id}

Functional Dependencies:

reservation_id, insurance_id -> starting_date, ending_date, insurance_fee

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS insurance CASCADE;
CREATE TABLE insurance(
 insurance_id SERIAL NOT NULL,
 reservation_id INT NOT NULL,
 starting_date DATE,
 ending_date DATE,
 insurance_fee INT,
 PRIMARY KEY (insurance_id, reservation_id),
 FOREIGN KEY (reservation_id) REFERENCES reservation(reservation_id) ON DELETE
CASCADE
);
```

2.10 User Phone

Relational Model:

user_phone(<u>user_id, phone</u>)

Candidate Keys and Primary Key:

Candidate Key: {user_id, phone}
Primary Key: {user_id, phone}

Functional Dependencies:

No functional dependencies

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS user_phone CASCADE;
CREATE TABLE user_phone(
 user_id INT NOT NULL PRIMARY KEY,
 phone INT NOT NULL UNIQUE,
 FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE
);
```

2.11 Damage Report

Relational Model:

damage_report(report_id, car_id, title, content, cost)

Candidate Keys and Primary Key:

Candidate Key: {report_id, car_id}
Primary Key: {report_id, car_id}

Functional Dependencies:

report_id, car_id -> content, cost

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS damage_report CASCADE;
CREATE TABLE damage_report(
   report_id SERIAL NOT NULL,
   car_id INT NOT NULL,
   title VARCHAR(50),
   content VARCHAR(200),
   cost INT,
   PRIMARY KEY (report_id, car_id),
   FOREIGN KEY (car_id) REFERENCES car(car_id) ON DELETE CASCADE
);
```

2.12 Shared Ride

Relational Model:

shared_ride(ride_id, shared_cost, reservation_id)
FK: reservation_id references Reservation(reservation_id)

Candidate Keys and Primary Key:

Candidate Key: ride_id, reservation_id
Primary Key: ride_id

Functional Dependencies:

ride_id -> shared_cost, reservation_id
reservation_id -> ride_id, shared_cost, reservation_id

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS shared_ride CASCADE;
CREATE TABLE shared_ride(
   ride_id SERIAL NOT NULL PRIMARY KEY,
   reservation_id INT NOT NULL,
   shared_cost INT,
   FOREIGN KEY (reservation_id) REFERENCES reservation(reservation_id) ON
DELETE CASCADE
);
```

2.13 Shared Ride Customer

Relational Model:

customer_shared_ride(ride_id, passenger_id, start_branch, dest_branch)
FK: passenger_id references customer(user_id)
FK: ride_id references shared_ride(ride_id)
FK: start_branch references Branch(branch_id)
FK: dest_branch references Branch(branch_id)

Candidate Keys and Primary Key:

Candidate Key: {ride_id, passenger_id}
Primary Key: {ride_id, passenger_id}

Functional Dependencies:

ride_id, passenger_id -> start_branch, dest_branch

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS customer_shared_ride CASCADE;
CREATE TABLE customer_shared_ride(
 ride_id INT NOT NULL,
 passenger_id INT NOT NULL,
 start_branch INT,
 dest_branch INT,
 PRIMARY KEY (ride_id, passenger_id),
 FOREIGN KEY(ride_id) REFERENCES shared_ride(ride_id) ON DELETE CASCADE,
 FOREIGN KEY(passenger_id) REFERENCES customer(user_id) ON DELETE CASCADE,
 FOREIGN KEY(start_branch) REFERENCES branch(branch_id) ON DELETE CASCADE,
 FOREIGN KEY(dest_branch) REFERENCES branch(branch_id) ON DELETE CASCADE
);
```

2.14 Request Car

Relational Model:

request_car(car_id, request_id)
FK: car_id references Car(car_id)
FK: request_id references Request(request_id)

Candidate Keys and Primary Key:

Candidate Key: {car_id, request_id}
Primary Key: {car_id, request_id}

Functional Dependencies:

No functional dependencies

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS request_car CASCADE;
CREATE TABLE request_car(
 car_id INT NOT NULL,
 request_id INT NOT NULL,
 PRIMARY KEY (car_id, request_id),
 FOREIGN KEY(car_id) REFERENCES car(car_id) ON DELETE CASCADE,
 FOREIGN KEY(request_id) REFERENCES request(request_id) ON DELETE CASCADE
);
```

2.15 Car Reservation

Relational Model:

customer_shared_ride(car_id, reservation_id)
FK: car_id references Car(car_id)
FK: reservation_id references reservation(reservation_id)

Candidate Keys and Primary Key:

Candidate Key: {car_id, reservation_id}
Primary Key: {car_id, reservation_id}

Functional Dependencies:

No functional dependencies

Normal Form:

Boyce-Codd Normal Form (BCNF)

SQL Declaration:

```
DROP TABLE IF EXISTS car_reservation CASCADE;
CREATE TABLE car_reservation(
 car_id INT,
 reservation_id INT,
 PRIMARY KEY (car_id, reservation_id),
 FOREIGN KEY(car_id) REFERENCES car(car_id) ON DELETE CASCADE,
 FOREIGN KEY(reservation_id) REFERENCES reservation(reservation_id) ON DELETE
CASCADE
);
```

2.16 Car Branch

Relational Model:

car_branch(<u>car_id, branch_id</u>)
FK: car_id references Car(car_id)
FK: branch_id references branch(branch_id)

Candidate Keys and Primary Key:

Candidate Key: {car_id, branch_id}
Primary Key: {car_id, branch_id}

Functional Dependencies:

No functional dependencies

Normal Form:

Boyce-Codd Normal Form (BCNF)

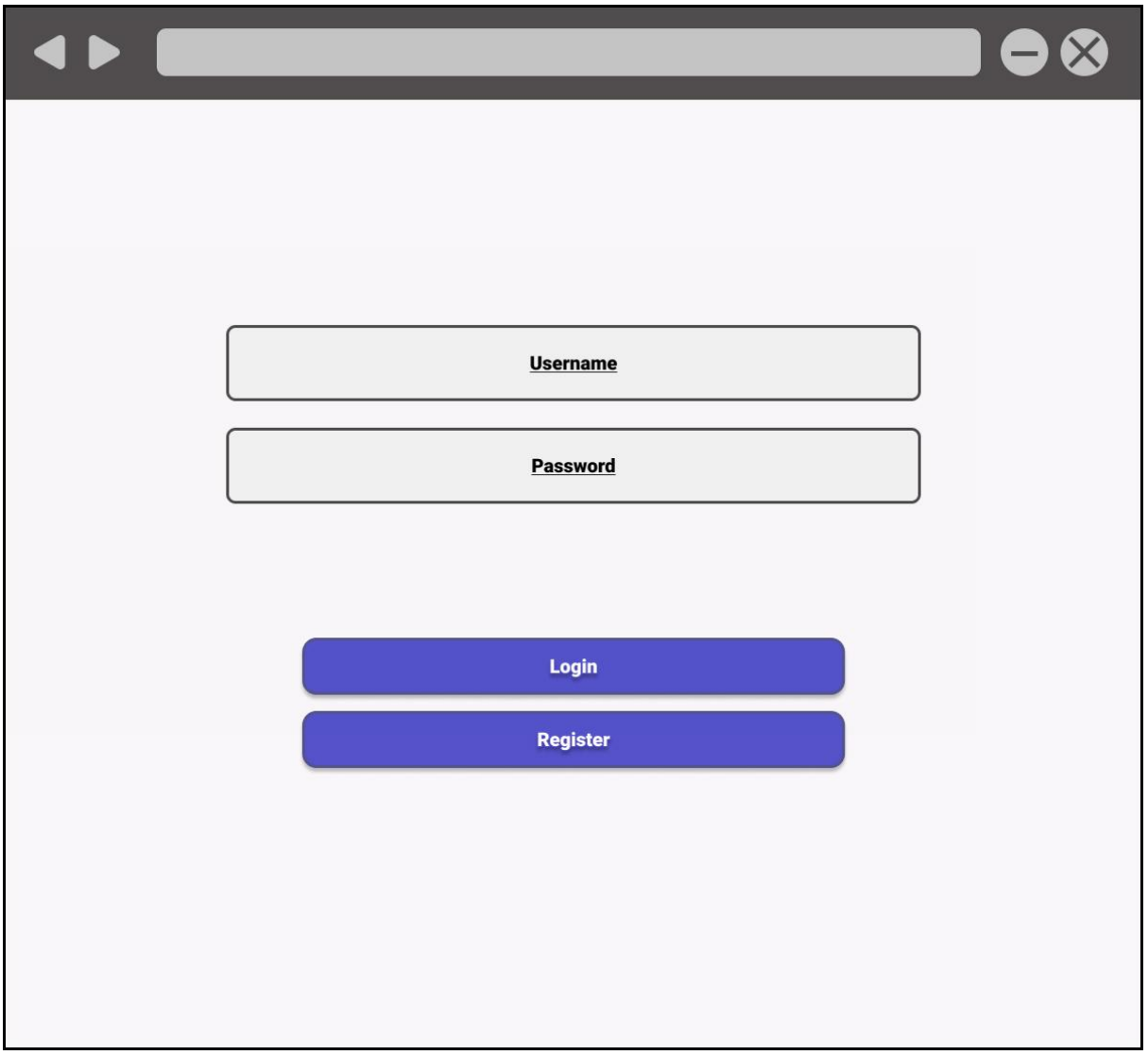
SQL Declaration:

```
DROP TABLE IF EXISTS car_branch CASCADE;
CREATE TABLE car_branch (
   car_id INT NOT NULL,
   branch_id INT NOT NULL,
   PRIMARY KEY (car_id, branch_id),
   FOREIGN KEY(car_id) REFERENCES car(car_id) ON DELETE CASCADE,
   FOREIGN KEY(branch_id) REFERENCES branch(branch_id) ON DELETE CASCADE
);
```

3. GUI DESIGN AND SQL QUERIES

3.1 Login Page

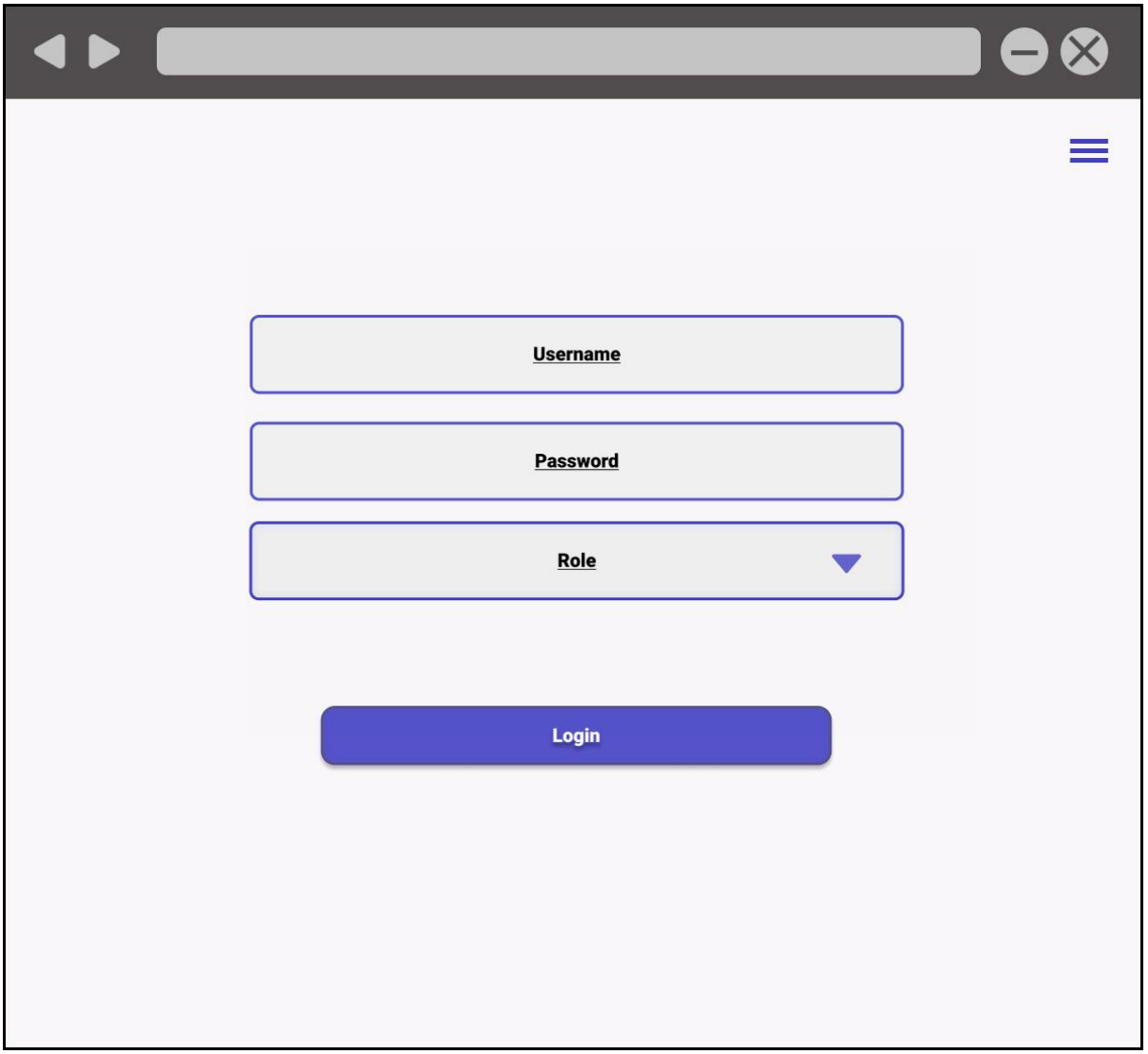
This page will authenticate customers and grant them access to authenticated services such as car reservation or request.




```
SELECT user_id FROM users WHERE email = @email AND password = @password;
```

3.2 Worker Login Page

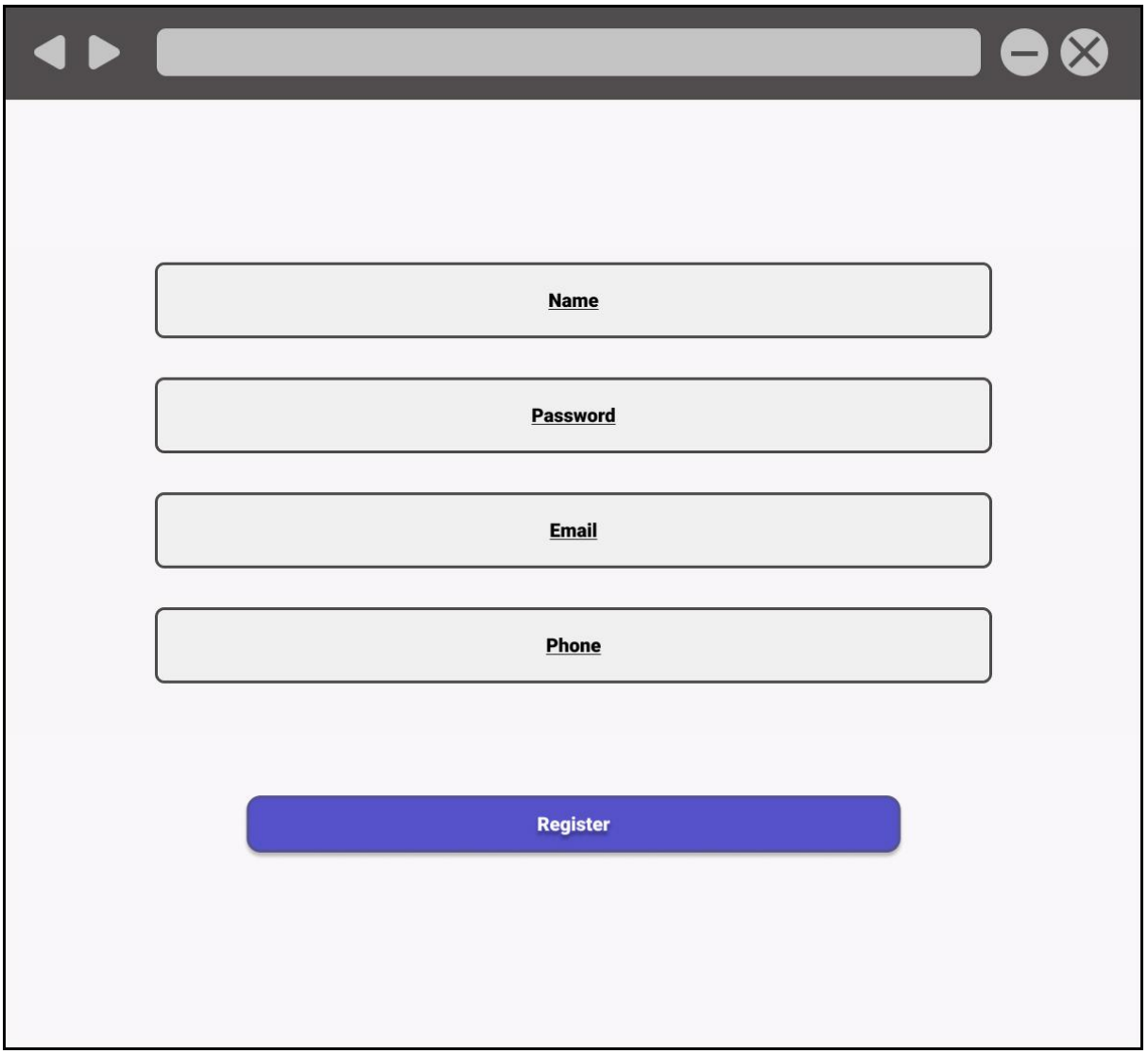
This page will authenticate employees and managers and it will grant them access to the system. Role value will be later used for determining whether a manager or an employee is logging in.




```
SELECT user_id, password FROM users
WHERE email = @email
AND password = @password;

SELECT * FROM employee WHERE user_id = @user_id AND password = @password;;
SELECT * FROM manager  WHERE user_id = @user_id AND password = @password;;
```

3.3 Register Page

This page will register a customer to the records.

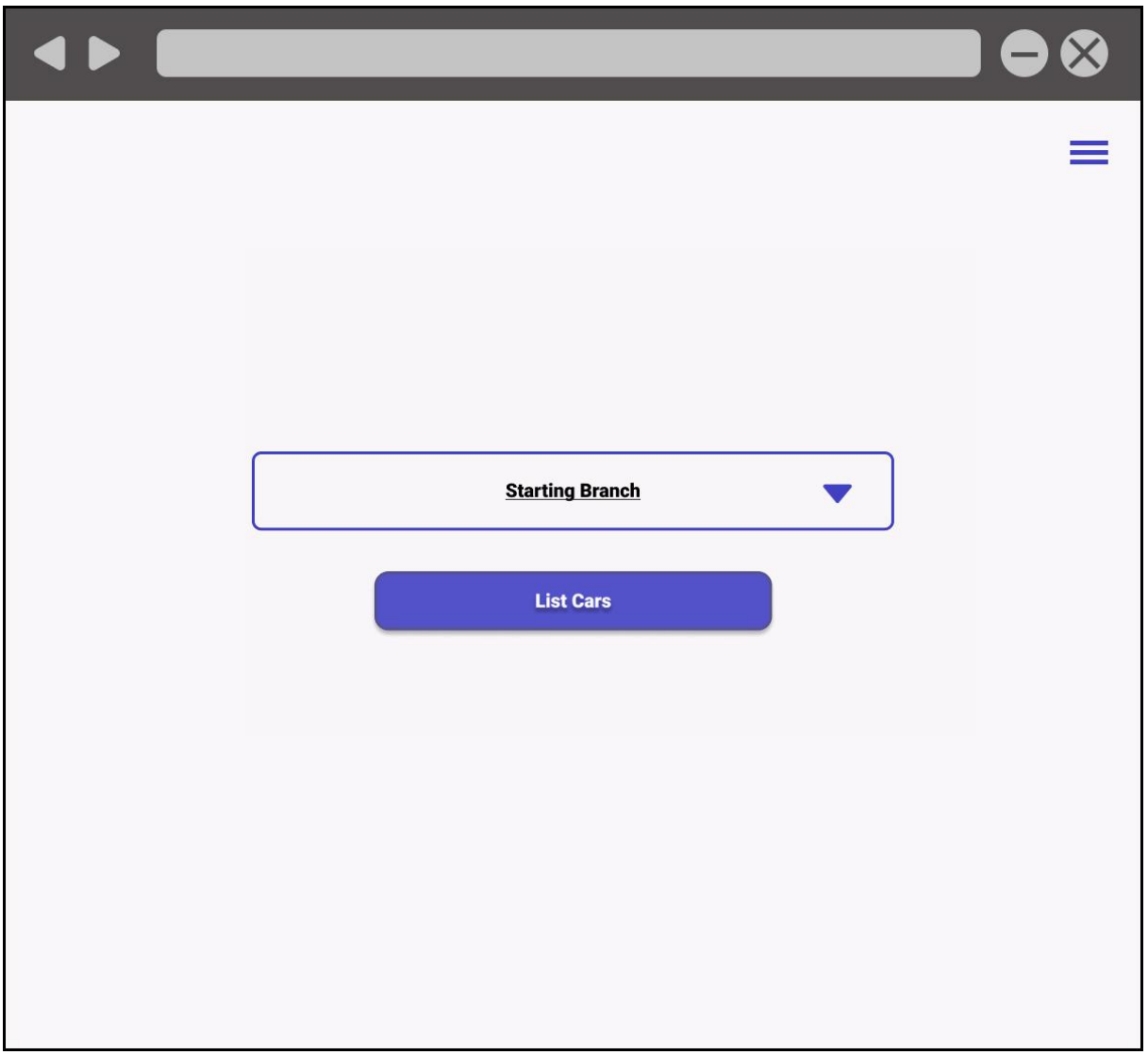



```
INSERT INTO users (first_name, middle_name, surname, password, email) VALUES
(@first_name, @middle_name, @middle_name, @password, @email);
```

```
SELECT user_id FROM users WHERE email = @email;
```

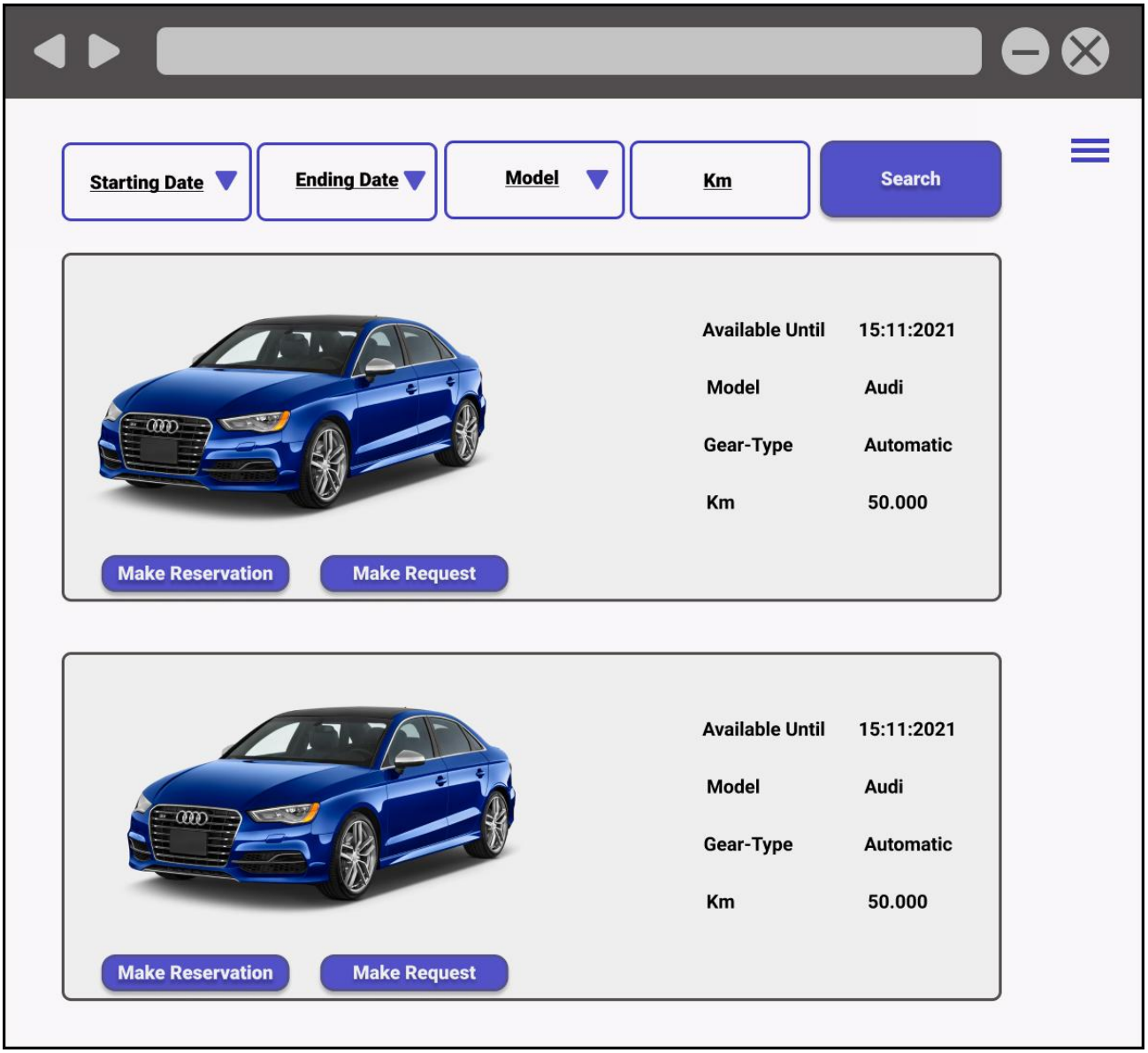
```
INSERT INTO customer (user_id, driving_license_id, point) VALUES (@user_id,
@license_id, @points);
```

3.4 Branch Selection



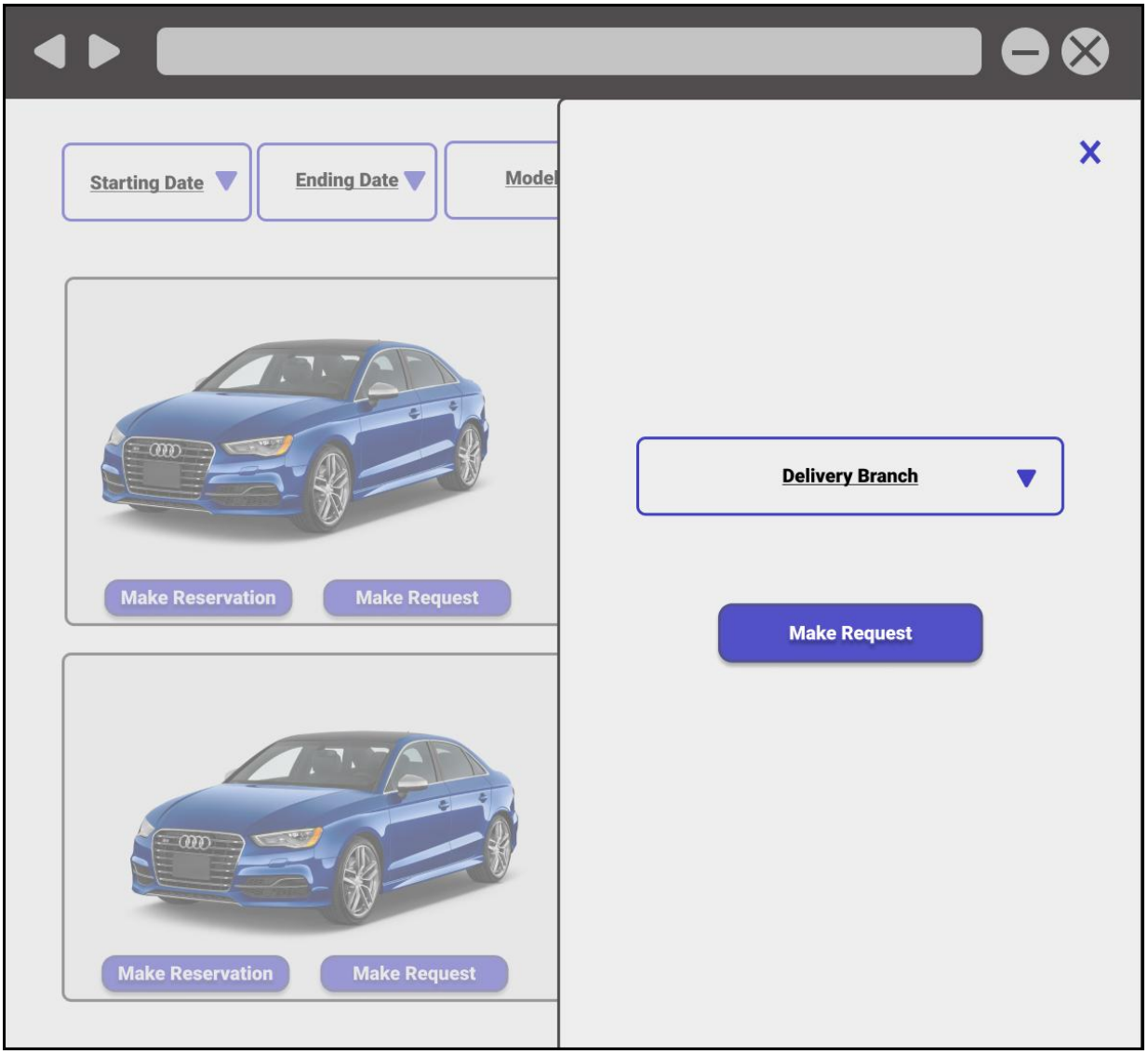

```
SELECT * FROM branch WHERE branch_id = @branch_id;
```

3.5 Car Selection Page




```
SELECT * FROM car
branch_id = @branch_id AND
WHERE model = @model AND  km <= @km;
```

3.6 Car Transfer Request

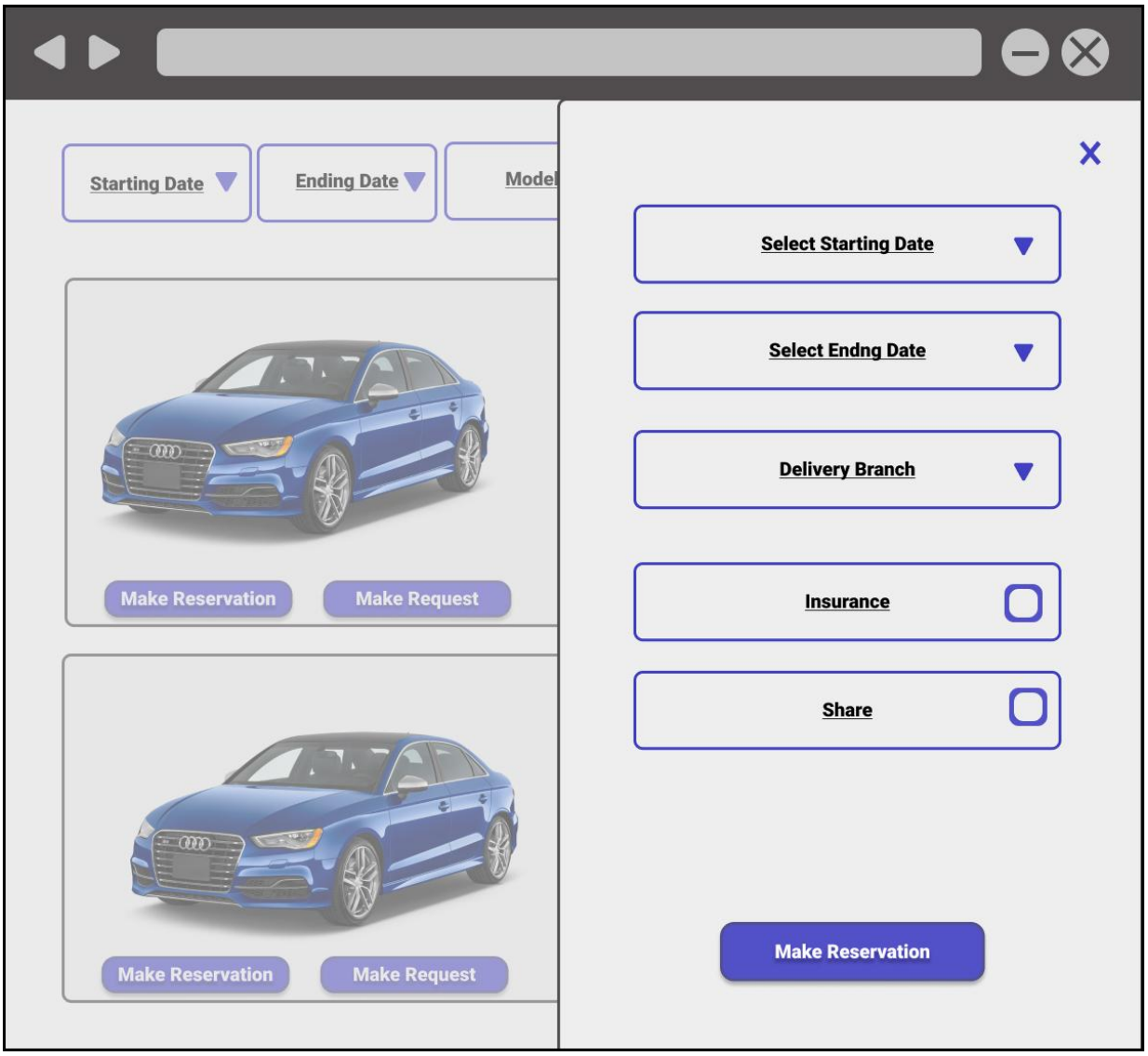



```
SELECT branch_id FROM branch WHERE branch_id = @branch_id;
INSERT INTO request (start_branch, dest_branch, status) VALUES
(@starting_bracnh_id, @ending_branch_id, 0);
```

```
INSERT INTO request_car (car_id, request_id) VALUES (@car_id, @request_id);
```

3.7 Car Reservation Page

Using this page customers will be able to reserve cars. If the insurance button is checked then it will create a record of insurance and if the share button is checked it will put this reservation to the ride sharing board. Where other customers can join in on the reservation.

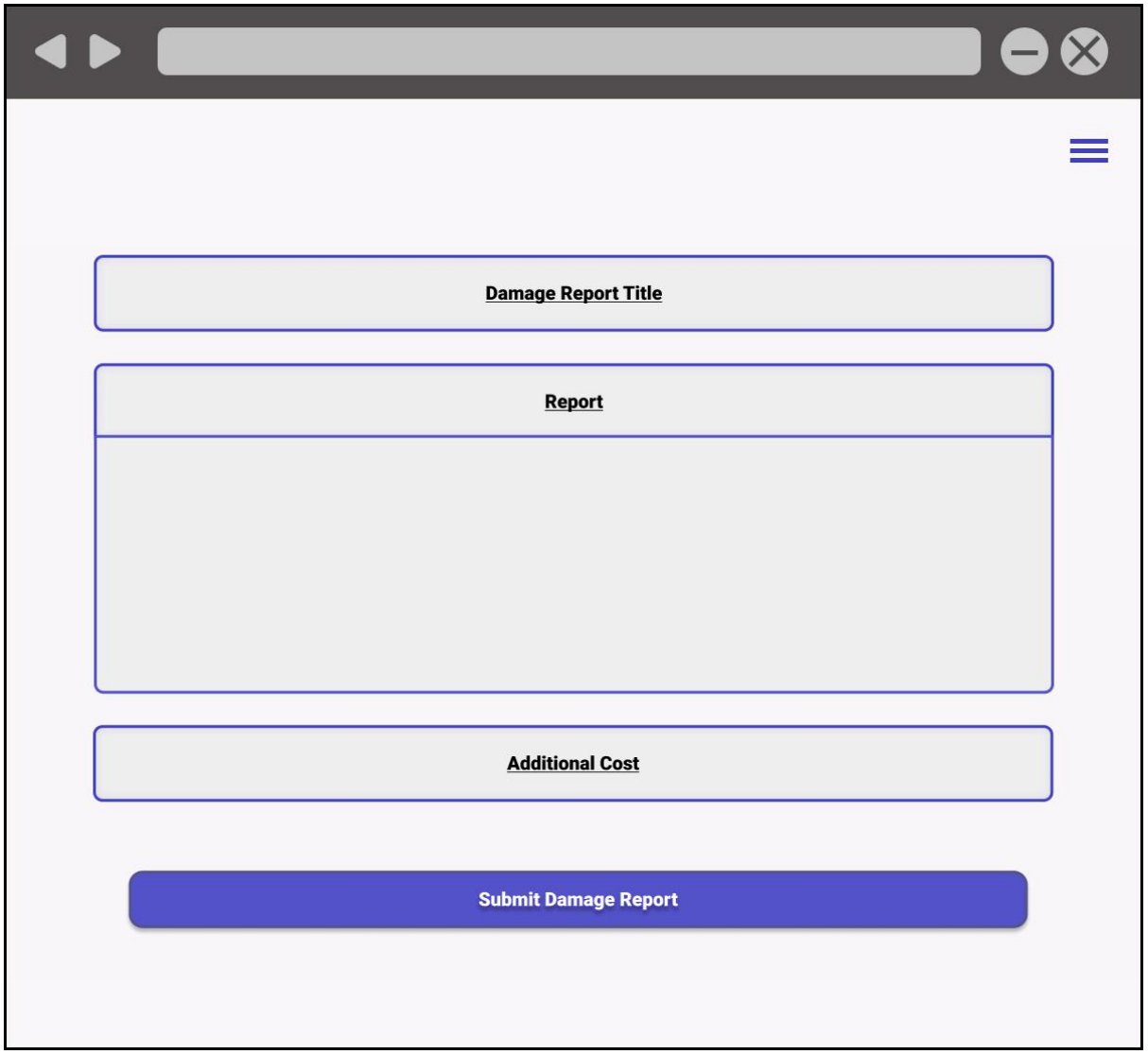



```
SELECT branch_id FROM branch WHERE branch_name = @branch_name;

INSERT INTO reservation (starting_date, ending_date, total_cost,
penalty_cost, status, has_insurance, customer_id, depart_from, arrive_at)
VALUES (@starting_date, @ending_date, @total_cost , 0, 0, @has_insurace,
@user_id, @starting_branch, @ending_branch);
```

```sql
INSERT INTO car_reservation (car_id, reservation_id) VALUES (@car_id,
@reservation_id);

INSERT INTO shared_ride (reservation_id, shared_cost) VALUES
(@reservation_id, @cost);
```

3.8 Car Damage Report Page

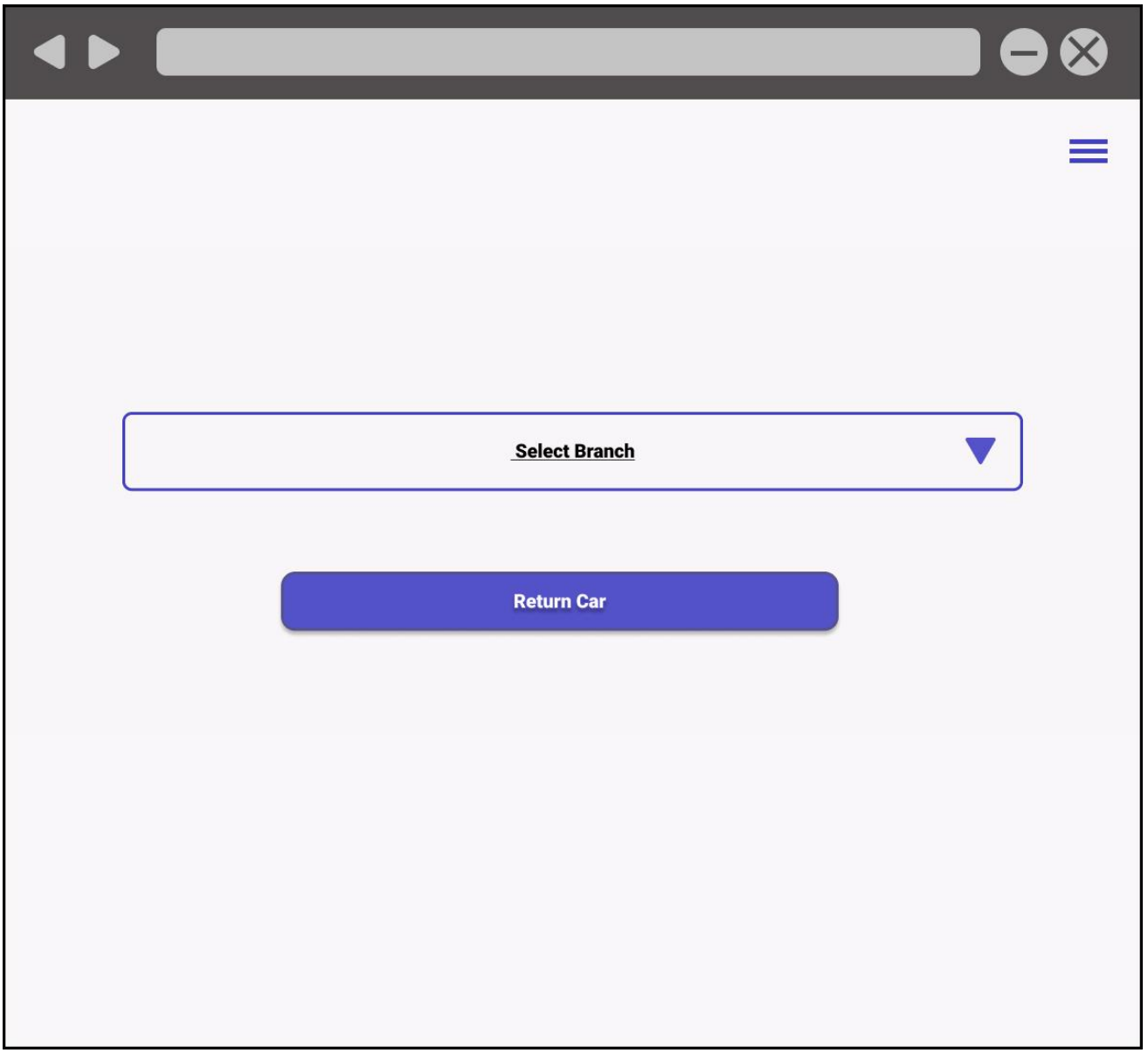
Employee will enter his report about the car's condition using the following page. Using this report additional costs will be calculated.




```
INSERT INTO damage_report (car_id, title, content, cost) VALUES (@car_id, @title,
@content, @cost);
```

3.9 Return Car Page

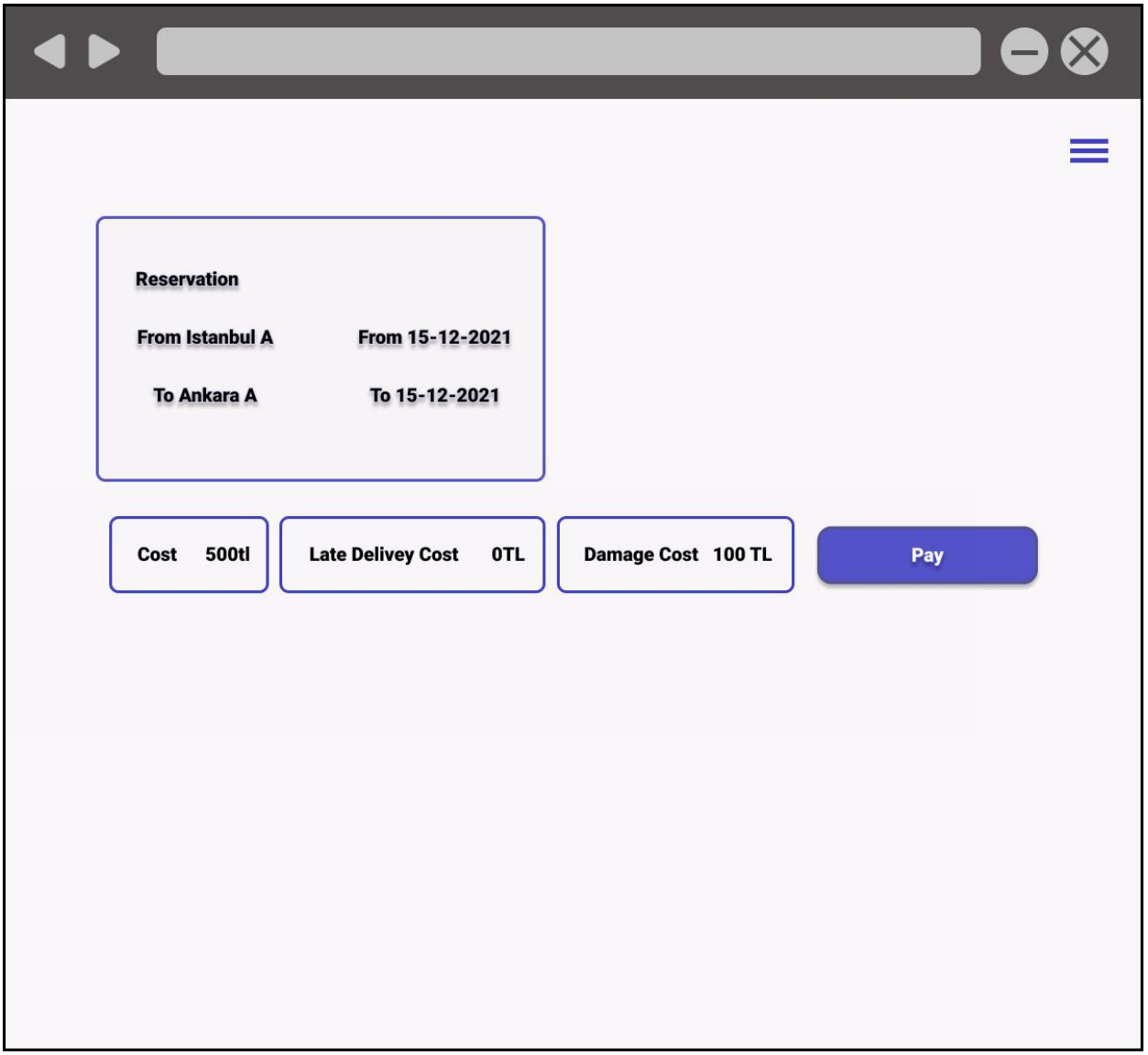
This page will be accessible by employees and managers and it will enable them to return cars into the selected branch.




```
INSERT INTO car_branch (car_id, branch_id) VALUES (@car_id, @branch_id);
```

3.10 Reservation Payment Page

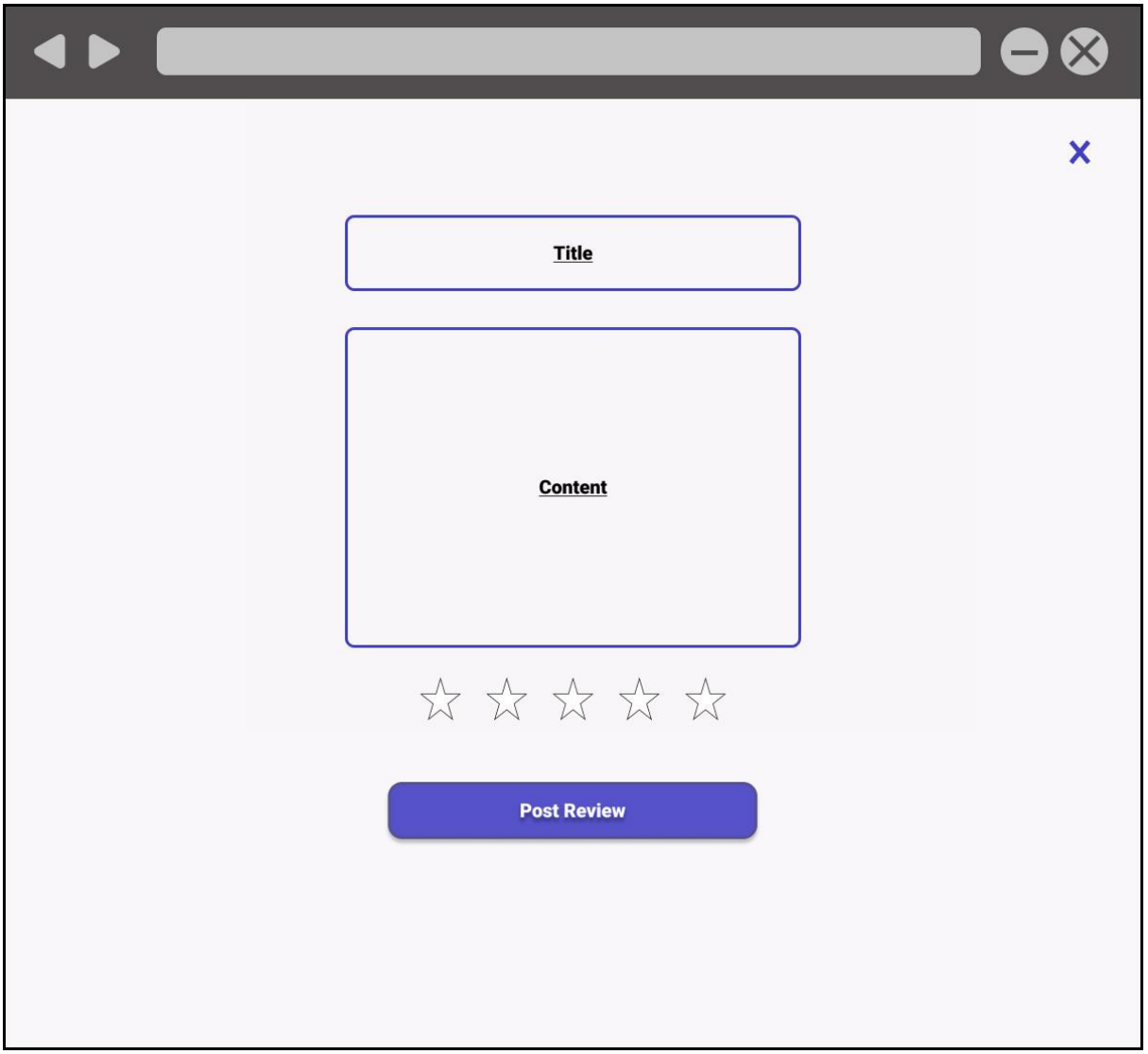
This page will allow customers to pay for the service they received after they deliver the vehicle to the delivery branch.




```
INSERT INTO damage_report (car_id, title, content, cost) VALUES (@car_id, @title,
@content, @cost);

UPDATE reservation SET penalty_cost  = @late_delivery_cost, total_cost =
@late_delivery_cost + @damage_cost WHERE reservation_id = @reservation_id;
```

3.11 Comment Page

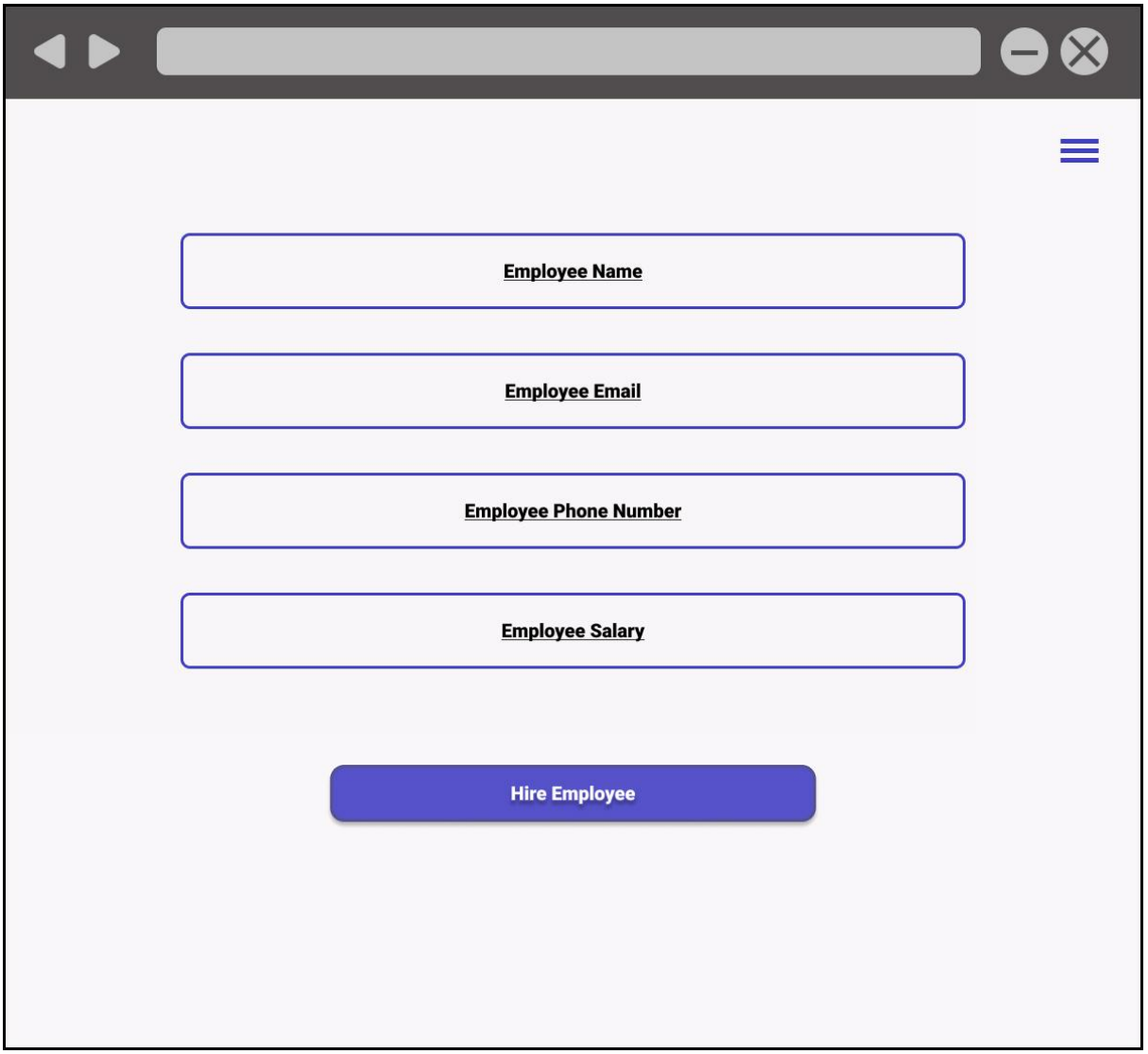
This page will be available to the customers after their payment is processed. They will be able to post reviews about their experiences.




```
INSERT INTO review (reservation_id, content, star) VALUES (@reservation_id,
@content, @star);
```

3.12 Employee Hire Page

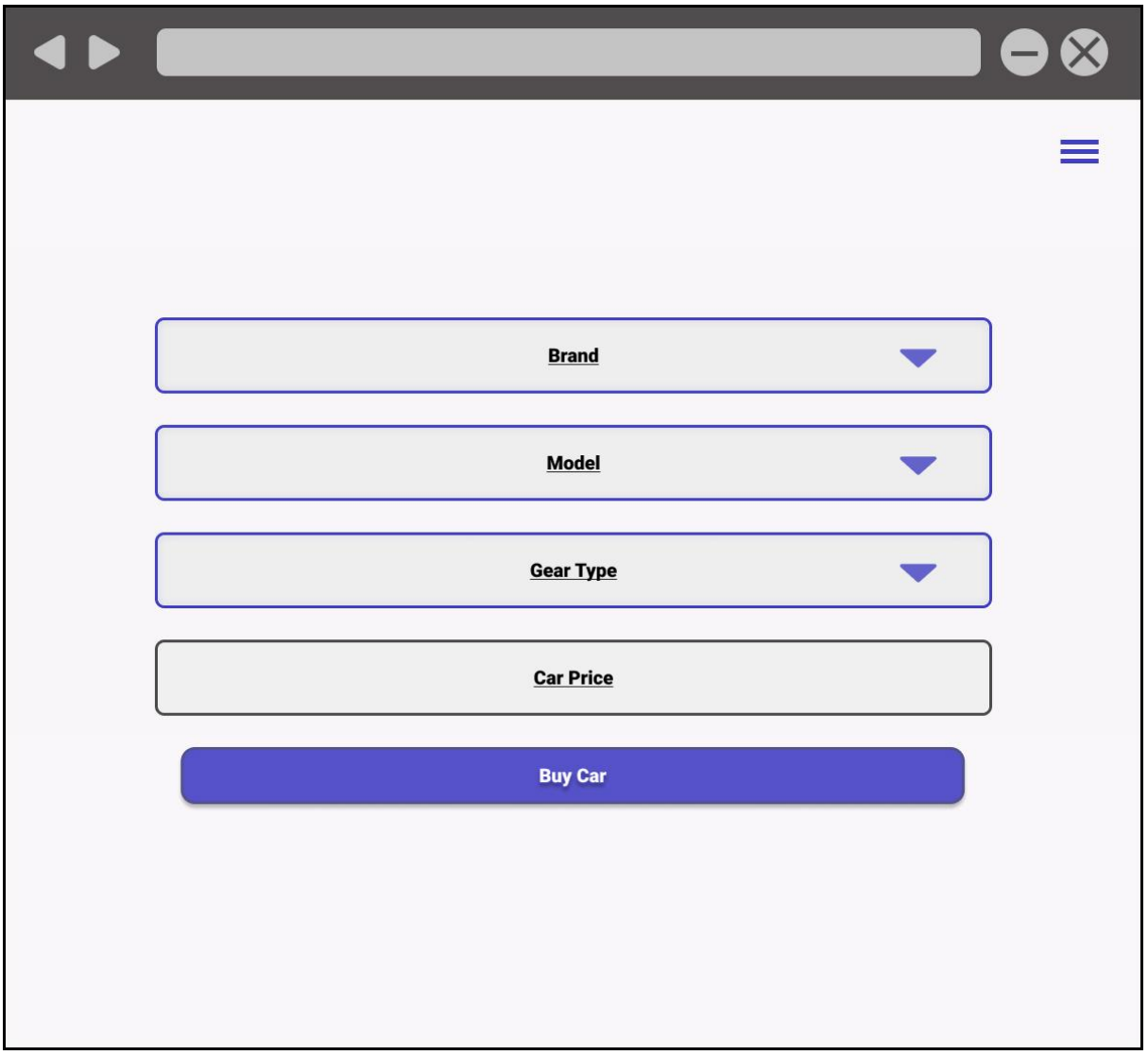
This page will be available to the managers. It will provide managers with a form for hiring new employees to their branches.




```
/* Creates a new user record */
INSERT INTO users (first_name, middle_name, surname, password, email) VALUES
(@first_name, @middle_name, @surname, @password, @email);
/* Retrieves the new records id */
SELECT users.user_id FROM users WHERE users.email = @email;
/* Creates a new worker record */
INSERT INTO worker (@user_id, @salary) VALUES (@user_id, @salary);
/* Creates a new employee record */
INSERT INTO employee (user_id, rating, hired_by) VALUES (@user_id,  0,
@manager_id);
```

3.13 Buy Car Page

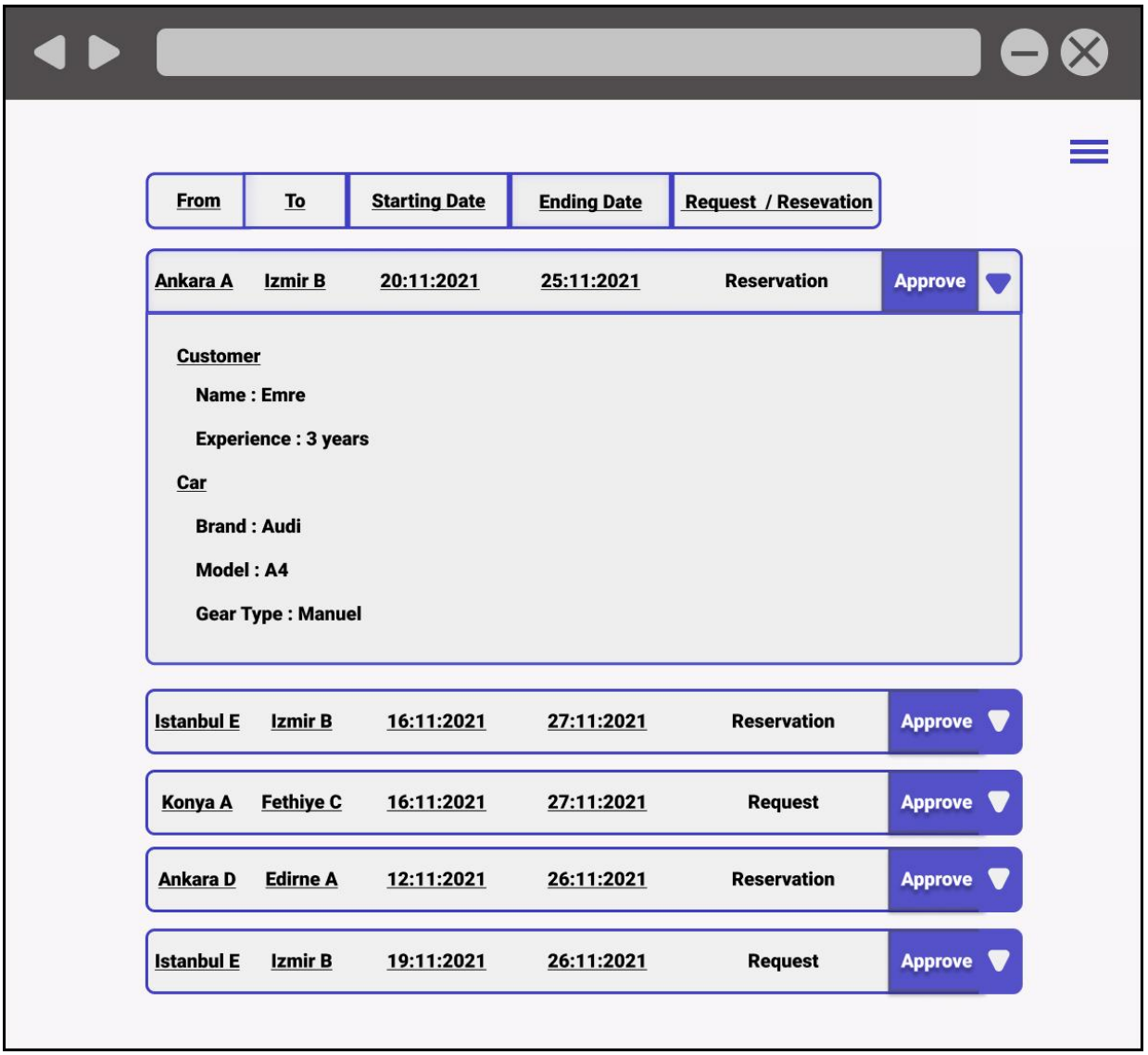
This page will allow managers to buy new cars for their branches.




```
INSERT INTO car (brand, model, gear_type, price, manager_id)
      VALUES (@brand, @model, @gear_type, @price, @manager_id);
INSERT INTO car_branch (@car_id, @branch_id);
```

3.14 Validate Request / Reservation Page

This page will give employees and managers an interface for approving reservations and requests made by the customers.

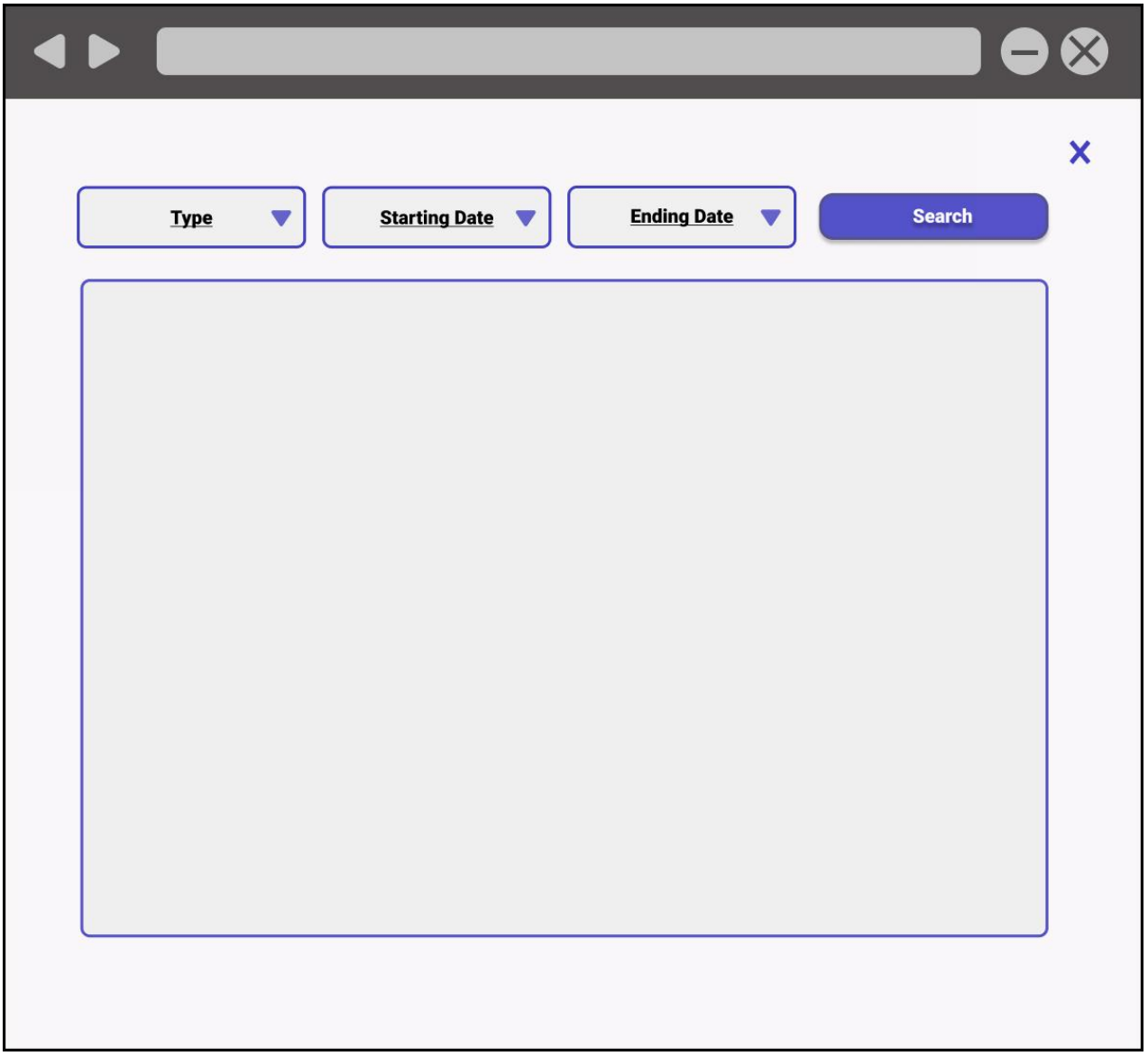



```
-- take the necessary ones..
SELECT * FROM reservation R, car_reservation CR
   JOIN insurance I  ON R.reservation_id = I.reservation_id
   JOIN customer  C  ON C.user_id = R.customer_id
   JOIN branch    B1 ON b1.branch_id = R.depart_from
   JOIN branch    B2 ON b2.branch_id = R.arrive_at
   JOIN car          ON car.car_id = CR.car_id
WHERE CR.reservation_id = R.reservation_id;
```

```sql
-- take the necessary ones.
SELECT * FROM request req
JOIN branch B1          on b1.branch_id = req.dest_branch
JOIN branch B2          on b2.branch_id = req.start_branch
JOIN request_car rcar   on rcar.request_id  = req.request_id
JOIN car                on car.car_id = rcar.car_id
```

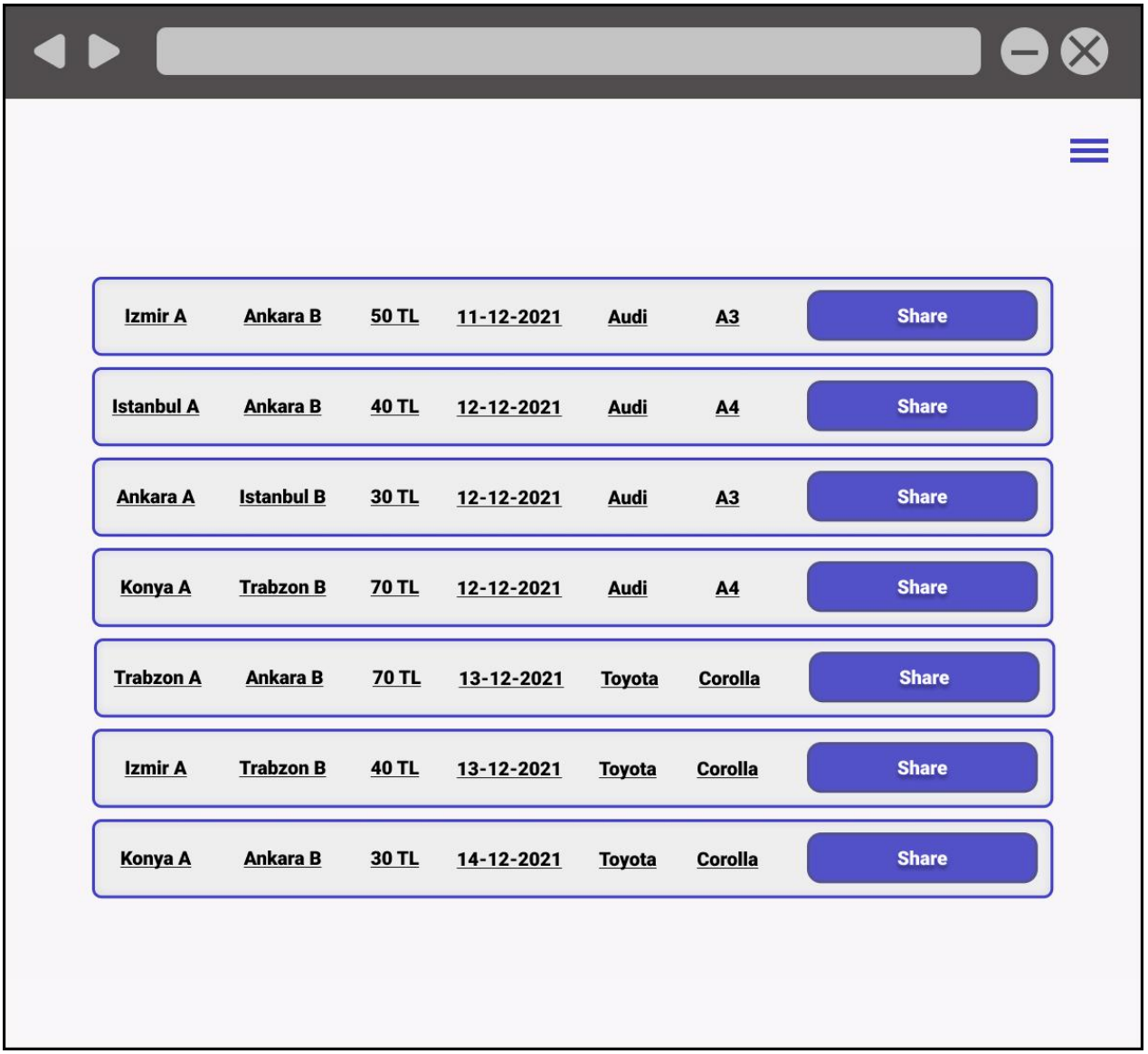
3.15 Query Report Page

This page will give workers an interface for retrieving information about past reservations and requests.



3.16 Ride Sharing Page

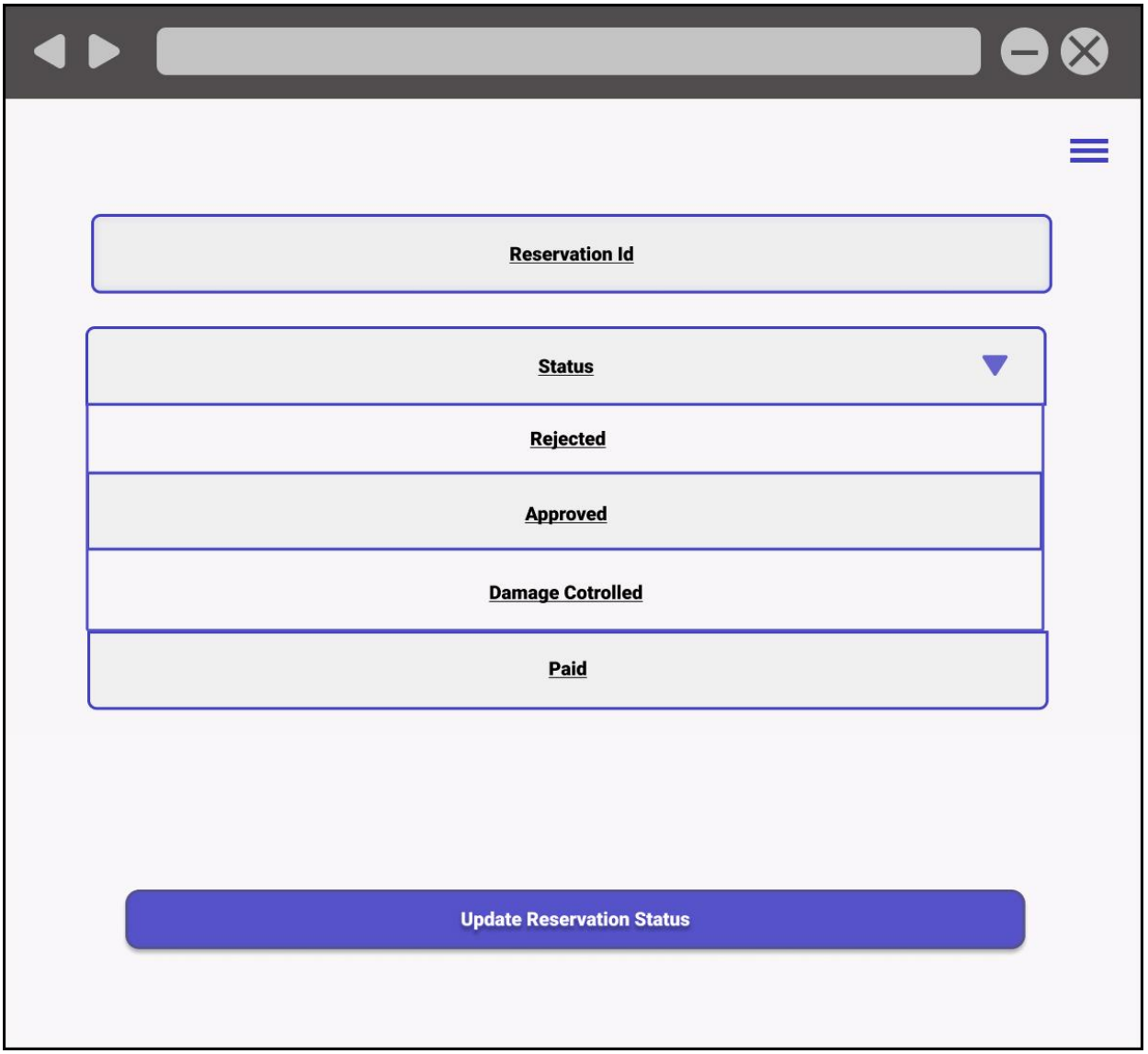
This page allows customers to join other customers' reservations. By clicking the share button.




```
SELECT * FROM shared_ride SR
JOIN reservation R ON R.reservation_id = SR.reservation_id;

INSERT INTO customer_shared_ride (ride_id, passenger_id) VALUES (@ride_id,
@customer_id);
```

3.17 Reservation Status Update Page

This page lets employees and managers update the status of a given reservation.




```
UPDATE reservation SET status = @status WHERE reservation_id =
@reservation_id;
```

4. IMPLEMENTATION PLAN

In the implementation, we are going to start by implementing our database using PostgreSQL.

We are going to use Java's Spring Framework for our web server. Spring JDBC will be used to query the database. Spring Security will be used for authenticating URLs, and Thymleaf will be used as the HTML templating tool. We are going to use a Controller Service Repository design where each controller is responsible for a designated page, and each repository is responsible for its respective tables. Services will be the components where we will encapsulate our business logic and error handling. For serving HTML documents, we will use Spring MVC's model view controller. For live data updates, we will use javascript together with the rest of the controllers in the server. For form validations, we will use javascript together with Thymeleaf. For improving the security and reliability of the system, we will create built-in assertions to our database. For styling, we will use the following tools: CSS, SASS, Bootstrap.

We can say that we are going to lay our project's foundations on PostgreSQL Database and Java Spring Framework.

5. WEBSITE

Website: https://edemirkirkan.github.io/Car-Rental-System/
Repository: https://github.com/edemirkirkan/Car-Rental-System