CS 353 - Database Systems Term Project

Project Name: WeRent

Group No: 16

Design Report

20.04.2023

Instructor: Prof. Özgür Ulusoy **TA:** Mousa Farshkar Azari

Group Members:

Mustafa Burak Erkoçak - 22003609

Emre Karataş - 22001641

Selin Bahar Gündoğar - 22001514

Zeynep Hanife Akgül - 22003356

Zeynep Selcen Öztunç- 21902941

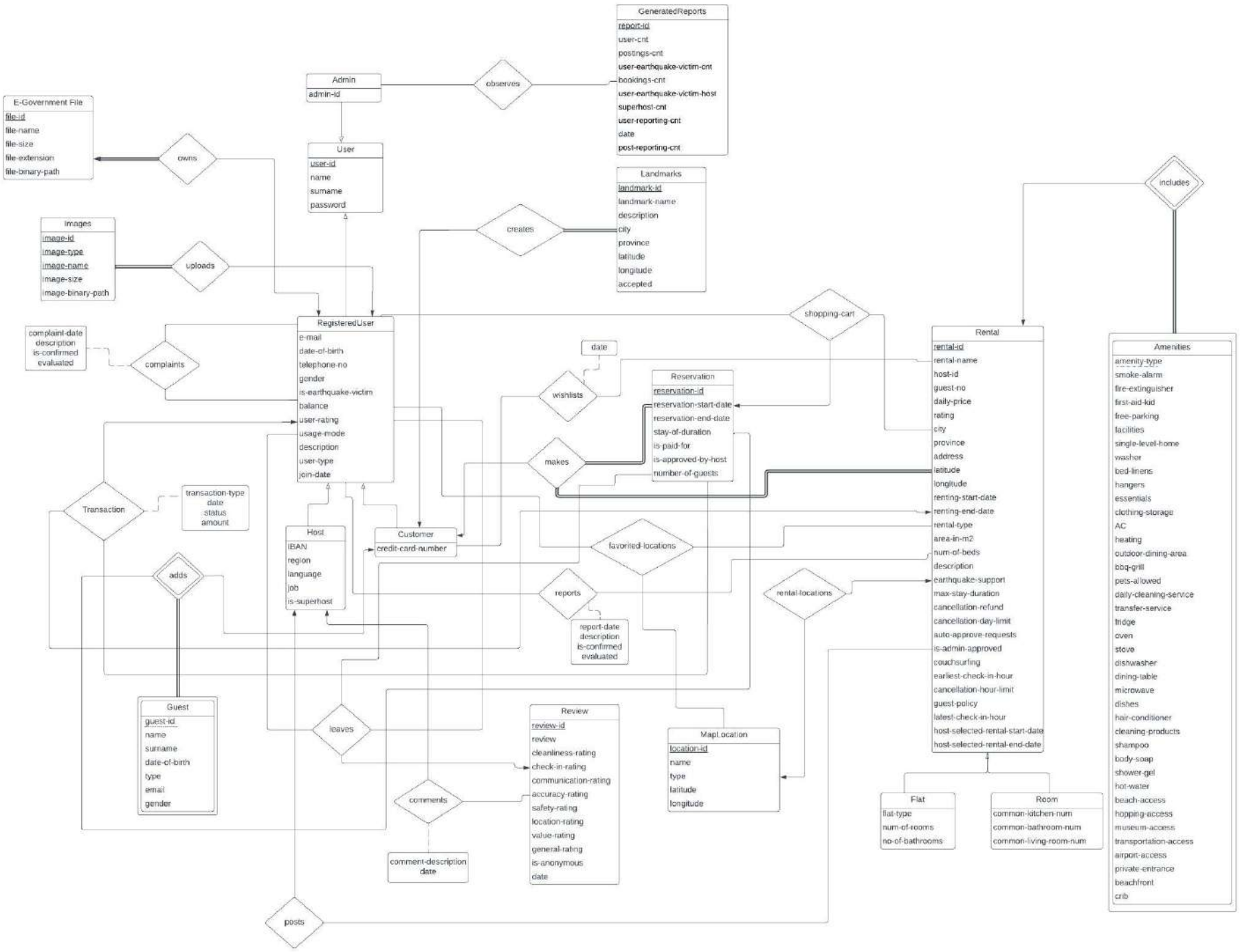
Table of Contents

1.Design of the Database

1.1 Revised E/R Diagram

Editor Link:
https://lucid.app/documents/view/f86d6e47-a6af-4e51-8682-442e388fe89c

Drive Link:
https://drive.google.com/drive/folders/15QGleby7Z6BOlq7KiwhxSz2x4vSqHY1P


GeneratedReports
report-id
user-cnt
postings-cnt
user-earthquake-victim-cnt
bookings-cnt
user-earthquake-victim-host
superhost-cnt
user-reporting-cnt
date
post-reporting-cnt
observes
Admin
admin-id
E-Government File
file-id
file-name
file-size
file-extension
file-binary-path
owns
User
user-id
name
surname
password
Landmarks
landmark-id
landmark-name
description
city
province
latitude
longitude
accepted
creates
includes
Images
image-id
image-type
image-name
image-size
image-binary-path
uploads
shopping-cart
RegisteredUser
e-mail
date-of-birth
telephone-no
gender
is-earthquake-victim
balance
user-rating
usage-mode
description
user-type
join-date
complaint-date
description
is-confirmed
evaluated
complaints
date
wishlists
Reservation
reservation-id
reservation-start-date
reservation-end-date
stay-of-duration
is-paid-for
is-approved-by-host
number-of-guests
makes
Rental
rental-id
rental-name
host-id
guest-no
daily-price
rating
city
province
address
latitude
longitude
renting-start-date
renting-end-date
rental-type
area-in-m2
num-of-beds
description
earthquake-support
max-stay-duration
cancellation-refund
cancellation-day-limit
auto-approve-requests
is-admin-approved
couchsurfing
earliest-check-in-hour
cancellation-hour-limit
guest-policy
latest-check-in-hour
host-selected-rental-start-date
host-selected-rental-end-date
Amenities
amenity-type
smoke-alarm
fire-extinguisher
first-aid-kid
free-parking
facilities
single-level-home
washer
bed-linens
hangers
essentials
clothing-storage
AC
heating
outdoor-dining-area
bbq-grill
pets-allowed
daily-cleaning-service
transfer-service
fridge
oven
stove
dishwasher
dining-table
microwave
dishes
hair-conditioner
cleaning-products
shampoo
body-soap
shower-gel
hot-water
beach-access
hopping-access
museum-access
transportation-access
airport-access
private-entrance
beachfront
crib
Transaction
transaction-type
date
status
amount
Host
IBAN
region
language
job
is-superhost
Customer
credit-card-number
favorited-locations
adds
reports
report-date
description
is-confirmed
evaluated
rental-locations
Guest
guest-id
name
surname
date-of-birth
type
email
gender
leaves
Review
review-id
review
cleanliness-rating
check-in-rating
communication-rating
accuracy-rating
safety-rating
location-rating
value-rating
general-rating
is-anonymous
date
MapLocation
location-id
name
type
latitude
longitude
comments
comment-description
date
Flat
flat-type
num-of-rooms
no-of-bathrooms
Room
common-kitchen-num
common-bathroom-num
common-living-room-num
posts

1.2 Table Schemas

User

Relational Model

User(user-id, name, surname, password)

Functional Dependencies

user-id -> name, surname, password

Candidate Keys

{user-id}

Normal Form

The table is in BCNF and therefore in 3NF. Attribute user-id is a superkey.

Table Definition

```
create table User(
    user-id int not null auto_increment,
    name varchar(30) not null,
    surname varchar(40) not null,
    password varchar(40) not null,
    PRIMARY KEY (user-id)
);
```

Admin

Relational Model

Admin(user-id, admin-id)

user-id: FK to User

Functional Dependencies

user-id -> admin-id

Candidate Keys

{user-id}

Normal Form

The table is in BCNF and therefore in 3NF. Attribute user-id is a superkey.

Table Definition

```
create table Admin(
    user-id int not null,
    admin-id int not null,
    FOREIGN KEY user-id REFERENCES User(user-id)
        ON DELETE CASCADE,
    PRIMARY KEY (user-id)
);
```

RegisteredUser

Relational Model

RegisteredUser(<u>user-id</u>, e-mail, date-of-birth, telephone-no, gender, is-earthquake-victim, balance, user-rating, usage-mode, description, user-type, join-date)

user-id: FK to User

Functional Dependencies

user-id -> e-mail, date-of-birth, telephone-no, gender, is-earthquake-victim, balance, user-rating, usage-mode, description, user-type, join-date

e-mail -> e-mail, date-of-birth, telephone-no, gender, is-earthquake-victim, balance, user-rating, usage-mode, description, user-type, join-date

telephone-no -> e-mail, date-of-birth, telephone-no, gender, is-earthquake-victim, balance, user-rating, usage-mode, description, user-type, join-date

Candidate Keys

{user-id}

{e-mail}

{telephone-no}

Normal Form

The table is in BCNF and therefore in 3NF. Each functional dependency contains a superkey on the left-hand side (user-id, e-mail, telephone-no).

Table Definition

```
create table RegisteredUser(
    user-id int not null,
    e-mail varchar(60) not null,
    date-of-birth DATE not null,
    telephone-no varchar(15) not null,
    gender varchar(20) not null,
    is-earthquake-victim boolean,
    balance FLOAT(10),
    user-rating FLOAT(5),
    usage-mode varchar(20) not null,
    description TEXT,
    user-type varchar(20) not null,
    join-date DATE not null,
    FOREIGN KEY user-id REFERENCES User(user-id)
        ON DELETE CASCADE,
    PRIMARY KEY (user-id)
);
```

Host

Relational Model

Host(<u>user-id</u>, IBAN, region, language, job, is-superhost)

user-id: FK to RegisteredUser

Functional Dependencies

user-id -> IBAN, region, language, job, is-superhost

IBAN -> IBAN, region, language, job, is-superhost

Candidate Keys

{user-id}

{IBAN}

Normal Form

The table is in BCNF and therefore in 3NF. Both attributes user-id and IBAN are superkeys.

Table Definition

```
create table Host(
    user-id int not null,
    IBAN varchar(40) not null,
    region varchar(30),
    language varchar(30) not null,
    job varchar(30) not null,
    is-superhost boolean,
    FOREIGN KEY user-id REFERENCES RegisteredUser(user-id)
        ON DELETE CASCADE,
    PRIMARY KEY (user-id)
);
```

Customer

Relational Model

Customer(<u>user-id</u>, credit-card-number)

user-id: FK to RegisteredUser

Functional Dependencies

user-id -> credit-card-number

credit-card-number -> user-id (trivial due to transitivity)

Candidate Keys

{user-id}

{credit-card-number}

Normal Form

The table is in BCNF and therefore in 3NF. Both attributes user-id and credit-card-number are superkeys.

Table Definition

```
create table Customer(
    user-id int not null,
    credit-card-number varchar(20),
    FOREIGN KEY user-id REFERENCES RegisteredUser(user-id)
        ON DELETE CASCADE,
    PRIMARY KEY (user-id)
);
```

Rental

Relational Model

Rental(<u>rental-id</u>, rental-name, host-id, guest-no, daily-price, rating, city, province, address, latitude, longitude, renting-available-start-date, renting-available-end-date,host-selected-start-date, host-selected-end-date,guest-policy, rental-type, area-in-m2, num-of-beds, description, earthquake-support, max-stay-duration, cancellation-refund,cancellation-day-limit,earliest-check-in-hour,latest-check-in-hour, cancellation-hour-limit, auto-approve-requests, is-admin-approved, couchsurfing)

Functional Dependencies

rental-id -> rental-name, host-id, guest-no, daily-price, rating, city, province, address, latitude, longitude, renting-available-start-date, renting-available-end-date,host-selected-start-date, host-selected-end-date, guest-policy, rental-type, area-in-m2, num-of-beds, description, earthquake-support, max-stay-duration, cancellation-refund,

cancellation-day-limit,earliest-check-in-hour,latest-check-in-hour,cancellation-hour-limit, auto-approve-requests, is-admin-approved, couchsurfing

city, province, address, latitude, longitude -> rental-id

Candidate Keys

{rental-id}

Normal Form

The table is in BCNF and therefore in 3NF. Both functional dependencies include superkeys on their left-hand side (rentaid, {city, province, address, latitude, longitude}).

Table Definition

```
create table Rental(
    rental-id int not null auto_increment,
    rental-name varchar(100) not null,
    host-id int not null,
    guest-no int not null,
    daily-price FLOAT(10) not null,
    rating FLOAT(5),
    city varchar(20) not null,
    province varchar(20) not null,
    address varchar(120) not null,
    latitude FLOAT(20) not null,
    longitude FLOAT(20) not null,
    renting-available-start-date DATE not null,
    renting-available-end-date DATE not null,
    host-selected-rental-start-date DATE not null,
    host-selected-rental-end-date DATE not null,
```

```
        guest-policy varchar(100),
        rental-type varchar(20) not null,
        area-in-m2 FLOAT(5) not null,
        num-of-beds int not null,
        description TEXT,
        earthquake-support boolean,
        max-stay-duration int,
        cancellation-refund int,
        cancellation-day-limit int,
        earliest-check-in-hour TIME,
        latest-check-in-hour TIME,
        cancellation-hour-limit int,
        auto-approve-requests boolean not null,
        is-admin-approved boolean,
        couchsurfing boolean,
        PRIMARY KEY (rental-id)
    );
```

Posts

Relational Model

Posts(<u>rental-id</u>, user-id, added-date)

rental-id: FK to Rental

user-id: FK to Host

Functional Dependencies

rental-id -> user-id, added-date

Candidate Keys

{rental-id}

Normal Form

The table is in BCNF and therefore in 3NF. rental-id is a superkey as well as the only Candidate Key.

Table Definition

```
create table Posts(
    rental-id int not null,
    user-id int not null,
    added-date DATE not null,
    FOREIGN KEY rental-id REFERENCES Rental(rental-id)
        ON DELETE CASCADE,
    FOREIGN KEY user-id REFERENCES Host(user-id)
        ON DELETE CASCADE,
    PRIMARY KEY (rental-id)
);
```

Amenities

Relational Model

Amenities(<u>rental-id, amenity-type</u>, smoke-alarm, fire-extinguisher, first-aid-kit, free-parking, facilities, single-level-home, washer, bed-linens, hangers, essentials, clothing-storage, AC, heating, outdoor-dining-area, bbq-grill, pets-allowed, daily-cleaning-service, transfer-service, fridge, oven, stove, dishwasher, dining-table, microwave, dishes, hair-conditioner, cleaning-products, shampoo, body-soap, shower-gel, hot-water, beach-access, hopping-access, museum-access, transportation-access, airport-access, private-entrance, beachfront, crib)

rental-id: FK to Rental

Functional Dependencies

rental-id, amenity-type -> smoke-alarm, fire-extinguisher, first-aid-kit, free-parking, facilities, single-level-home, washer, bed-linens, hangers, essentials, clothing-storage, AC, heating, outdoor-dining-area, bbq-grill, pets-allowed, daily-cleaning-service, transfer-service, fridge, oven, stove, dishwasher, dining-table, microwave, dishes,

hair-conditioner, cleaning-products, shampoo, body-soap, shower-gel, hot-water, beach-access, hopping-access, museum-access, transportation-access, airport-access, private-entrance, beachfront, crib

Candidate Keys

{rental-id, amenity-type}

Normal Form

The table is in BCNF and therefore in 3NF. rental-id and amenity-type attributes are a superkey together.

Table Definition

```
create table Amenities(
    rental-id int not null,
    amenity-type varchar(20) not null,
    smoke-alarm int,
    fire-extinguisher int,
    first-aid-kit int,
    free-parking int,
    facilities int,
    single-level-home int,
    washer int,
    bed-linens int,
    hangers int,
    essentials int,
    clothing-storage int,
    AC int,
    heating int,
    outdoor-dining-area int,
```

bbq-grill int,

pets-allowed int,

daily-cleaning-service int,

transfer-service int,

fridge int,

oven int,

stove int,

dishwasher int,

dining-table int,

microwave int,

dishes int,

hair-conditioner int,

cleaning-products int,

shampoo int,

body-soap int,

shower-gel int,

hot-water int,

beach-access int,

hopping-access int,

museum-access int,

transportation-access int,

airport-access int,

private-entrance int,

beachfront int,

crib int,

```
        FOREIGN KEY rental-id REFERENCES Rental(rental-id)
            ON DELETE CASCADE,
        PRIMARY KEY (rental-id, amenity-type)
    );
```

Flat

Relational Model

Flat(<u>rental-id</u>, flat-type, num-of-rooms, no-of-bathrooms)

rental-id: FK to Rental

Functional Dependencies

rental-id -> flat-type, num-of-rooms, no-of-bathrooms

Candidate Keys

{rental-id}

Normal Form

The table is in BCNF and therefore in 3NF. rental-id is a superkey for the table.

Table Definition

```
    create table Flat(
        rental-id int not null,
        flat-type varchar(20),
        num-of-rooms int not null,
        no-of-bathrooms int not null,
        FOREIGN KEY rental-id REFERENCES Rental(rental-id)
            ON DELETE CASCADE,
        PRIMARY KEY (rental-id)
    );
```

Room

Relational Model

Room(rental-id, common-kitchen-num, common-bathroom-num, common-living-room-num)

rental-id: FK to Rental

Functional Dependencies

rental-id -> common-kitchen-num, common-bathroom-num, common-living-room-num

Candidate Keys

{rental-id}

Normal Form

The table is in BCNF and therefore in 3NF. rental-id is a superkey for the table.

Table Definition

```
create table Room(
    rental-id int not null,
    common-kitchen-num int not null,
    common-bathroom-num int not null,
    common-living-room-num int not null,
    FOREIGN KEY rental-id REFERENCES Rental(rental-id)
        ON DELETE CASCADE,
    PRIMARY KEY (rental-id)
);
```

E-GovernmentFile

Relational Model

E-GovernmentFile(file-id, file-name, file-size, file-extension, file-binary-path)

Functional Dependencies

file-id -> file-name, file-size, file-extension, file-binary-path

binary-path -> file-id

Candidate Keys

{file-id}

{binary-path}

Normal Form

The table is in BCNF and therefore in 3NF. file-id is a superkey.

Table Definition

```
create table E-GovernmentFile(
    file-id int not null auto_increment,
    file-name varchar(30) not null,
    file-size FLOAT(10) not null,
    file-extension varchar(5) not null,
    file-binary-path TEXT not null,
    PRIMARY KEY (file-id)
);
```

Owns

Relational Model

Owns(<u>user-id, file-id</u>)

user-id: FK to RegisteredUser

file-id: FK to E-GovernmentFile

Functional Dependencies

user-id, file-id -> user-id, file-id (trivial)

Candidate Keys

{user-id, file-id}

Normal Form

The table is in BCNF and therefore in 3NF. user-id and file-id together is trivially the superkey, as well as the Primary Key.

Table Definition

```
create table Owns(
    user-id int not null,
    file-id int not null,
    FOREIGN KEY user-id REFERENCES RegisteredUser(user-id)
        ON DELETE CASCADE,
    FOREIGN KEY file-id REFERENCES E-GovernmentFile(file-id)
        ON DELETE CASCADE,
    PRIMARY KEY (user-id, file-id)
);
```

Images

Relational Model

Images(<u>image-id, image-type, image-name</u>, user-id, image-size, img-binary-path)

user-id: FK to RegisteredUser

Functional Dependencies

image-id, image-type, image-name -> user-id, image-size, img-binary-path

Candidate Keys

{image-id, image-type, image-name}

Normal Form

The table is in BCNF and therefore in 3NF. image-id, image-type and image-name together is the superkey.

Table Definition

```
create table Images(
    image-id int not null auto_increment,
    image-type varchar(15) not null,
    image-name varchar(30) not null,
```

```
        user-id int not null,
        image-size FLOAT(10) not null,
        img-binary-path TEXT not null,
        FOREIGN KEY user-id REFERENCES RegisteredUser(user-id)
            ON DELETE CASCADE,
        PRIMARY KEY (image-id, image-type, image-name)
    );
```

Landmarks

Relational Model

Landmarks(<u>landmark-id</u>, user-id, landmark-name, description, city, province, latitude, longitude, accepted)

Functional Dependencies

landmark-id -> user-id, landmark-name, description, city, province, latitude, longitude, accepted

latitude, longitude -> landmark-id

landmark-name, city, province -> landmark-id

Candidate Keys

{landmark-id}

Normal Form

The table is in BCNF and therefore in 3NF. landmark-id is a superkey as well as the only Candidate Key.

Table Definition

```
create table Landmarks(
        landmark-id int not null auto_increment,
        user-id int not null,
        landmark-name varchar(40) not null,
        description TEXT,
```

```
        city varchar(20) not null,
        province varchar(20) not null,
        latitude FLOAT(20) not null,
        longitude FLOAT(20) not null,
        accepted boolean,
        PRIMARY KEY (landmark-id)
    );
```

Complaints

Relational Model

Complaints(<u>user-id1, user-id2, complaint-date</u>, description, is-confirmed, evaluated)

user-id1: FK to RegisteredUser

user-id2: FK to RegisteredUser

Functional Dependencies

user-id1, user-id2, complaint-date -> description, is-confirmed, evaluated

Candidate Keys

{user-id1, user-id2, complaint-date}

Normal Form

The table is in BCNF and therefore in 3NF. Two separate user-ids (which are user-id1 and userid2) make up for the superkey.

Table Definition

```
    create table Complaints(
        user-id1 int not null,
        user-id2 int not null,
        complaint-date TIMESTAMP not null,
        description TEXT not null,
        is-confirmed boolean,
```

```
        evaluated boolean,
        FOREIGN KEY user-id1 REFERENCES RegisteredUser(user-id)
            ON DELETE CASCADE,
        FOREIGN KEY user-id2 REFERENCES RegisteredUser(user-id)
            ON DELETE CASCADE,
        PRIMARY KEY(user-id1, user-id2, complaint-date)
    );
```

Wishlists

Relational Model

Wishlists(<u>user-id, rental-id</u>, date)

user-id: FK to Customer

rental-id: FK to Rental

Functional Dependencies

user-id, rental-id -> date

Candidate Keys

{user-id, rental-id}

Normal Form

The table is in BCNF and therefore in 3NF. user-id and rental-id together determine the other attributes, thus it is a superkey.

Table Definition

```
create table Wishlists(
    user-id int not null,
    rental-id int not null,
    date DATE not null,
    FOREIGN KEY user-id REFERENCES Customer(user-id)
        ON DELETE CASCADE,
```

```
        FOREIGN KEY rental-id REFERENCES Rental(rental-id)
            ON DELETE CASCADE,
        PRIMARY KEY(user-id, rental-id)
    );
```

Reports

Relational Model

Reports(<u>user-id, rental-id</u>, report-date, description, is-confirmed, evaluated)

user-id: FK to RegisteredUser

rental-id: FK to Rental

Functional Dependencies

user-id, rental-id -> report-date, description, is-confirmed, evaluated

Candidate Keys

{user-id, rental-id}

Normal Form

The table is in BCNF and therefore in 3NF. user-id and rental-id together determine the other attributes, thus it is a superkey.

Table Definition

```
create table Reports(
    user-id int not null,
    rental-id int not null,
    report-date DATE not null,
    description TEXT not null,
    is-confirmed boolean,
    evaluated boolean,
    FOREIGN KEY user-id REFERENCES RegisteredUser(user-id)
        ON DELETE CASCADE,
```

```
        FOREIGN KEY rental-id REFERENCES Rental(rental-id)
            ON DELETE CASCADE,
        PRIMARY KEY(user-id, rental-id)
    );
```

RentalLocations

Relational Model

RentalLocations(rental-id, location-id)

rental-id: FK to Rental

location-id: FK to MapLocation

Functional Dependencies

rental-id, location-id -> rental-id, location-id (trivial)

Candidate Keys

{rental-id, location-id}

Normal Form

The table is in BCNF and therefore in 3NF. {rental-id, location-id} is trivially the superkey.

Table Definition

```
    create table RentalLocations(
        rental-id int not null,
        location-id int not null,
        FOREIGN KEY rental-id REFERENCES Rental(rental-id)
            ON DELETE CASCADE,
        FOREIGN KEY location-id REFERENCES MapLocation(location-id)
            ON DELETE CASCADE,
        PRIMARY KEY(rental-id, location-id)
    );
```

MapLocation
Relational Model

MapLocation(location-id, name, type, latitude, longitude)

Functional Dependencies

location-id -> location-id, name, type, latitude, longitude

Candidate Keys

{location-id}

Normal Form

The table is in BCNF and therefore in 3NF. location-id is a superkey.

Table Definition

```
create table MapLocation(
    location-id int not null auto_increment,
    name varchar(40) not null,
    type varchar(20) not null,
    latitude FLOAT(20) not null,
    longitude FLOAT(20) not null,
    PRIMARY KEY (location-id)
);
```

FavoritedLocations
Relational Model

FavoritedLocations(location-id, user-id, rental-id)

location-id: FK to MapLocation

user-id: FK to RegisteredUser

rental-id: FK to Rental

Functional Dependencies

location-id, user-id -> rental-id

Candidate Keys

{location-id, user-id}

Normal Form

The table is in BCNF and therefore in 3NF. location-id and user-id make up for a superkey together.

Table Definition

```
create table FavoritedLocations(
    location-id int not null,
    user-id int not null,
    rental-id int not null,
    FOREIGN KEY location-id REFERENCES MapLocation(location-id)
        ON DELETE CASCADE,
    FOREIGN KEY user-id REFERENCES RegisteredUser(user-id)
        ON DELETE CASCADE,
    FOREIGN KEY rental-id REFERENCES Rental(rental-id)
        ON DELETE CASCADE,
    PRIMARY KEY(location-id, user-id)
);
```

GeneratedReports

Relational Model

GeneratedReports(<u>report-id, date</u>, user-cnt, host-cnt, postings-cnt, booking-cnt, user-earthquake-victim-cnt, host-earthquake-victim-cnt, superhost-cnt, user-reporting-cnt, post-reporting-cnt)

Functional Dependencies

report-id, date -> user-cnt, host-cnt, postings-cnt, booking-cnt, user-earthquake-victim-cnt, host-earthquake-victim-cnt, superhost-cnt, user-reporting-cnt, post-reporting-cnt

Candidate Keys

{report-id, date}

Normal Form

The table is in BCNF and therefore in 3NF. report-id and date, together is a superkey.

Table Definition

```
create table GeneratedReports(
    report-id int not null auto_increment,
    date TIMESTAMP not null,
    user-cnt int not null,
    host-cnt int not null,
    postings-cnt int not null,
    booking-cnt int not null,
    user-earthquake-victim-cnt int not null,
    host-earthquake-victim-cnt int not null,
    superhost-cnt int not null,
    user-reporting-cnt int not null,
    post-reporting-cnt int not null,
    PRIMARY KEY(report-id, date)
);
```

Observes

Relational Model

Observes(<u>user-id, report-id</u>)

user-id: FK to Admin

report-id: FK to GeneratedReports

Functional Dependencies

user-id, report-id -> user-id, report-id (trivial)

Candidate Keys

{user-id, report-id}

Normal Form

The table is in BCNF and therefore in 3NF. user-id and report-id together is a superkey.

Table Definition

```
create table Observes(
    user-id int not null,
    report-id int not null,
    FOREIGN KEY user-id REFERENCES AAdmin(user-id)
        ON DELETE CASCADE,
    FOREIGN KEY report-id REFERENCES GeneratedReports(report-id)
        ON DELETE CASCADE,
    PRIMARY KEY(user-id, report-id)
);
```

Review

Relational Model

Review(<u>review-id</u>, review, cleanliness-rating, check-in-rating, communication-rating, accuracy-rating, safety-rating, location-rating, value-rating, general-rating, is-anonymous)

Functional Dependencies

review-id -> review, cleanliness-rating, check-in-rating, communication-rating, accuracy-rating, safety-rating, location-rating, value-rating, general-rating, is-anonymous

Candidate Keys

{review-id}

Normal Form

The table is in BCNF and therefore in 3NF. review-id is a superkey.

Table Definition

```
create table Review(
    review-id int not null auto_increment,
    review TEXT,
    date DATETIME not null,
    cleanliness-rating int not null,
    check-in-rating int not null,
    communication-rating int not null,
    accuracy-rating int not null,
    safety-rating int not null,
    location-rating int not null,
    value-rating int not null,
    general-rating int not null,
    is-anonymous boolean not null,
    PRIMARY KEY(review-id)
);
```

Comments

Relational Model

Comments(<u>review-id</u>, user-id, comment-description, date)

review-id: FK to Review

user-id: FK to Host

Functional Dependencies

review-id -> user-id, comment-description, date

Candidate Keys

{review-id}

Normal Form

The table is in BCNF and therefore in 3NF. review-id is a superkey.

Table Definition

```
create table Comments(
    review-id int not null,
    user-id int not null,
    comment-description TEXT not null,
    date DATE not null,
    FOREIGN KEY review-id REFERENCES Review(review-id)
        ON DELETE CASCADE,
    FOREIGN KEY user-id REFERENCES Host(user-id)
        ON DELETE CASCADE,
    PRIMARY KEY(review-id)
);
```

Reservation

Relational Model

Reservation(<u>reservation-id</u>, reservation-start-date, reservation-end-date, stay-of-duration, is-paid-for, is-approved-by-host, number-of-guests)

Functional Dependencies

reservation-id -> reservation-start-date, reservation-end-date, stay-of-duration, is-paid-for, is-approved-by-host, number-of-guests

Candidate Keys

{reservation-id}

Normal Form

The table is in BCNF and therefore in 3NF. reservation-id is a superkey.

Table Definition

```
create table Reservation(
        reservation-id int not null auto_increment,
        reservation-start-date DATE not null,
        reservation-end-date DATE not null,
        stay-of-duration int not null,
        is-paid-for boolean not null,
        is-approved-by-host boolean,
        number-of-guests int not null,
        PRIMARY KEY (reservation-id)
);
```

Leaves

Relational Model

Leaves(<u>reservation-id, user-id1, user-id2</u>, review-id)

reservation-id: FK to Reservation

user-id1: FK to RegisteredUser

user-id2: FK to RegisteredUser

review-id: FK to Review

Functional Dependencies

reservation-id, user-id1, user-id2 -> review-id

Candidate Keys

{reservation-id, user-id1, user-id2}

Normal Form

The table is in BCNF and therefore in 3NF. This relation emerges from the quaternary relationship ‘leaves’. The set of reservation-id, user-id1, user-id2 is a superkey; with the underlying semantics: A given review can be present for a reservation arranged among two users (unique).

Table Definition

```
create table Leaves(
        reservation-id int not null,
        user-id1 int not null,
        user-id2 int not null,
        review-id int not null,
        FOREIGN KEY reservation-id REFERENCES Reservation(reservation-id)
            ON DELETE CASCADE,
        FOREIGN KEY user-id1 REFERENCES RegisteredUser(user-id)
            ON DELETE CASCADE,
        FOREIGN KEY user-id2 REFERENCES RegisteredUser(user-id)
            ON DELETE CASCADE,
        FOREIGN KEY review-id REFERENCES Review(review-id)
            ON DELETE CASCADE,
        PRIMARY KEY(reservation-id, user-id1, user-id2)
);
```

ShoppingCart

Relational Model

ShoppingCart(<u>user-id, rental-id</u>, reservation-id)

user-id: FK to RegisteredUser

rental-id: FK to Rental

reservation-id: FK to Reservation

Functional Dependencies

user-id, rental-id -> reservation-id

Candidate Keys

{reservation-id}

Normal Form

The table is in BCNF and therefore in 3NF. user-id and rental-id together is a superkey since only one reservation can exist (unique) in the shopping cart of the user for a certain rental.

Table Definition

```
create table ShoppingCart(
    user-id int not null,
    rental-id int not null,
    reservation-id int not null,
    FOREIGN KEY user-id REFERENCES RegisteredUser(user-id)
        ON DELETE CASCADE,
    FOREIGN KEY rental-id REFERENCES Rental(rental-id)
        ON DELETE CASCADE,
    FOREIGN KEY reservation-id REFERENCES Reservation(reservation-id)
        ON DELETE CASCADE,
    PRIMARY KEY(user-id, rental-id)
);
```

Transaction

Relational Model

Transaction(<u>reservation-id</u>, user-id, rental-id, transaction-type, date, status, amount)

reservation-id: FK to Reservation

user-id: FK to RegisteredUser

rental-id: FK to Rental

Functional Dependencies

reservation-id -> user-id, rental-id, transaction-type, date, status, amount

Candidate Keys

{reservation-id}

Normal Form

The table is in BCNF and therefore in 3NF. reservation-id is a superkey as it functionally determines the rest of the attributes.

Table Definition

```
create table Transaction(
    reservation-id int not null,
    user-id int not null,
    rental-id int not null,
    transaction-type varchar(20) not null,
    date TIMESTAMP not null,
    status varchar(20) not null,
    amount FLOAT(10) not null,
    FOREIGN KEY reservation-id REFERENCES Reservation(reservation-id)
        ON DELETE CASCADE,
    FOREIGN KEY user-id REFERENCES RegisteredUser(user-id)
        ON DELETE CASCADE,
    FOREIGN KEY rental-id REFERENCES Rental(rental-id)
        ON DELETE CASCADE,
    PRIMARY KEY (reservation-id)
);
```

Guest

Relational Model

Guest(<u>user-id, guest-id</u>, name, surname, date-of-birth, type, email, gender)

user-id: FK to Customer

Functional Dependencies

user-id, guest-id -> name, surname, date-of-birth, type, email, gender

email -> user-id, guest-id

Candidate Keys

{email} is the original candidate key. However, it cannot qualify as a PrimaryKey since it is a NULLABLE value. {user-id, guest-id}, in this case, is the alternative candidate key and should be chosen as the Primary Key.

Normal Form

The table is in BCNF and therefore in 3NF. Both functional dependencies have superkeys ({user-id, guest-id} and email) on the left-hand side.

Table Definition

```
create table Guest(
    user-id int not null,
    guest-id int not null,
    name varchar(30) not null,
    surname varchar(40) not null,
    date-of-birth DATE not null,
    type varchar(20),
    email varchar(60),
    gender varchar(20) not null,
    FOREIGN KEY user-id REFERENCES Customer(user-id)
        ON DELETE CASCADE,
    PRIMARY KEY (user-id, guest-id)
);
```

Makes

Relational Model

Makes(<u>reservation-id, rental-id</u>, user-id)

reservation-id: FK to Reservation

rental-id: FK to Rental

user-id: FK to Customer

Functional Dependencies

reservation-id, rental-id -> user-id

Candidate Keys

{reservation-id, rental-id}

Normal Form

The table is in BCNF and therefore in 3NF. reservation-id, together with rental-id is a superkey since a specific reservation on a rental can only be made by a single customer.

Table Definition

```
create table Makes(
    reservation-id int not null,
    rental-id int not null,
    user-id int not null,
    FOREIGN KEY reservation-id REFERENCES Reservation(reservation-id)
        ON DELETE CASCADE,
    FOREIGN KEY rental-id REFERENCES Rental(rental-id)
        ON DELETE CASCADE,
    FOREIGN KEY user-id REFERENCES Customer(user-id)
        ON DELETE CASCADE,
    PRIMARY KEY (reservation-id, rental-id)
);
```

Adds

Relational Model

Adds(<u>reservation-id, guest-id</u>, user-id)

reservation-id: FK to Reservation

guest-id: FK to Guest

user-id: FK to Customer

Functional Dependencies

reservation-id, guest-id -> user-id

Candidate Keys

{reservation-id, guest-id}

Normal Form

The table is in BCNF and therefore in 3NF. reservation-id and guest-id together is a superkey.

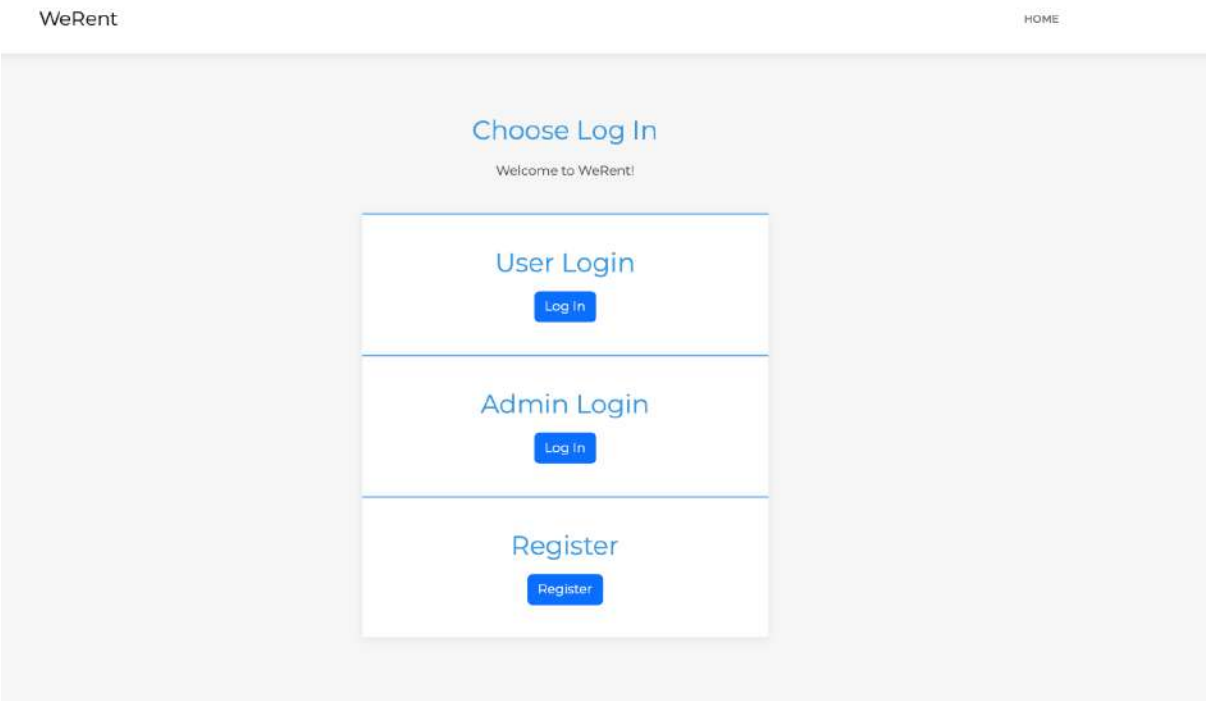
Table Definition

```
create table Adds(
    reservation-id int not null,
    guest-id int not null,
    user-id int not null,
    FOREIGN KEY reservation-id REFERENCES Reservation(reservation-id)
        ON DELETE CASCADE,
    FOREIGN KEY guest-id REFERENCES Guest(guest-id)
        ON DELETE CASCADE,
    FOREIGN KEY user-id REFERENCES Customer(user-id)
        ON DELETE CASCADE,
    PRIMARY KEY (reservation-id, guest-id)
);
```

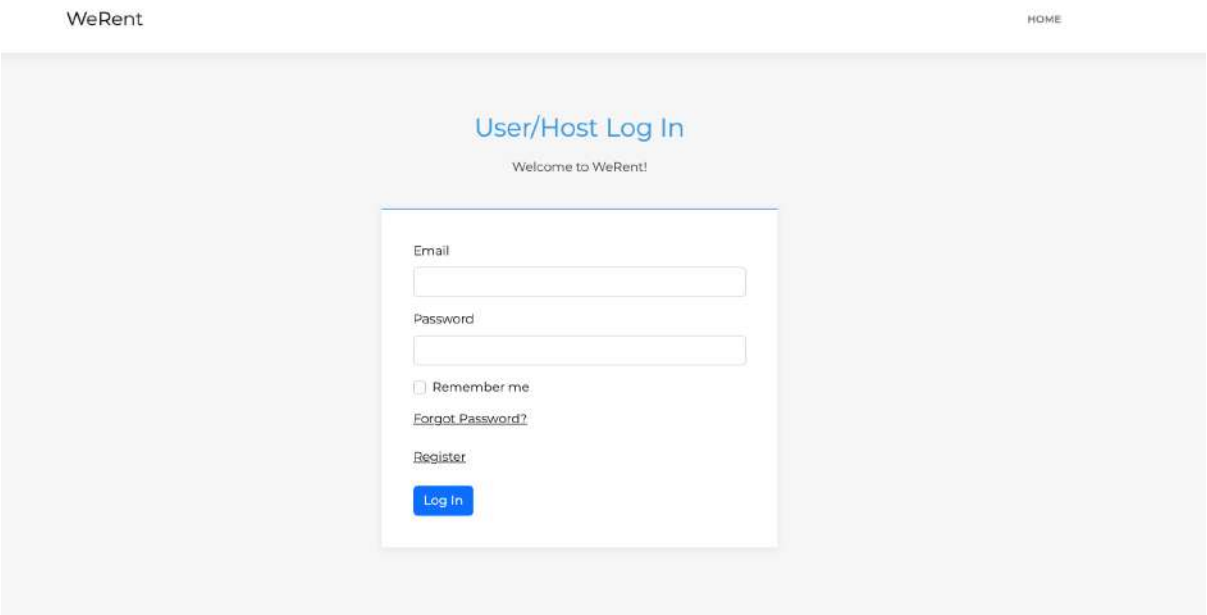
2.User Interface Design and Related SQL Statements

2.1 Common Functionalities

General Login Page



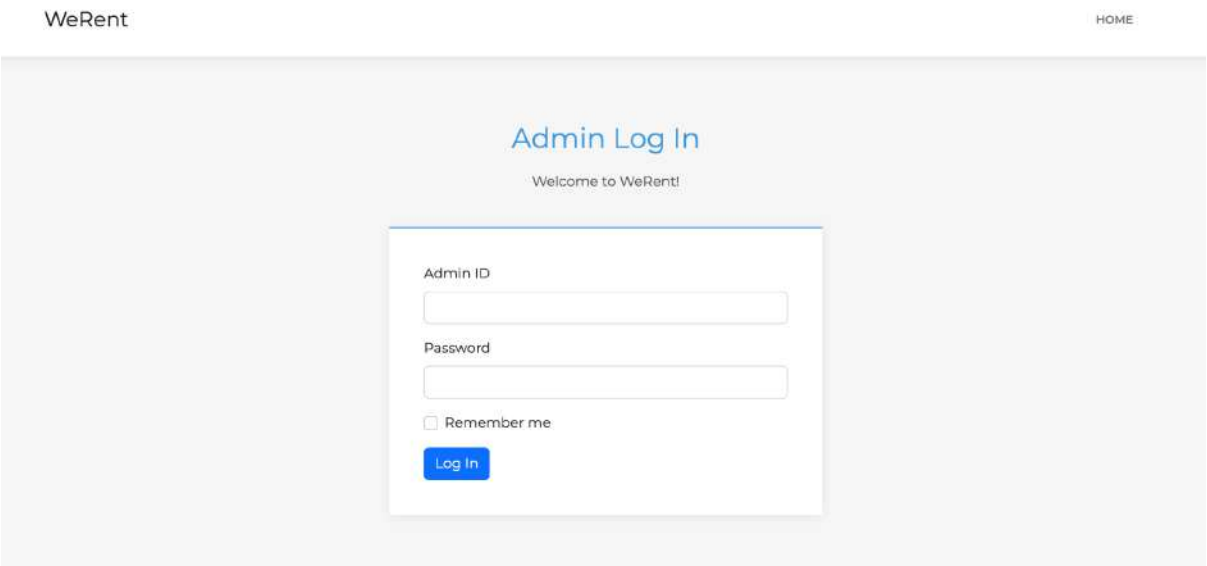
User/Host Login



SQL Statements:

SELECT *
FROM RegisteredUser
WHERE e-mail = @email AND password = @password

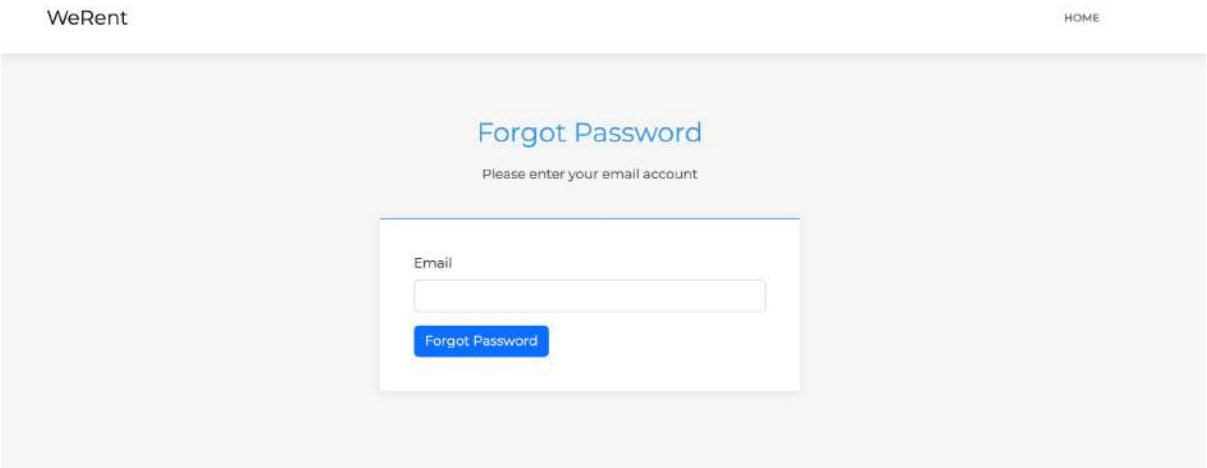
Admin Login

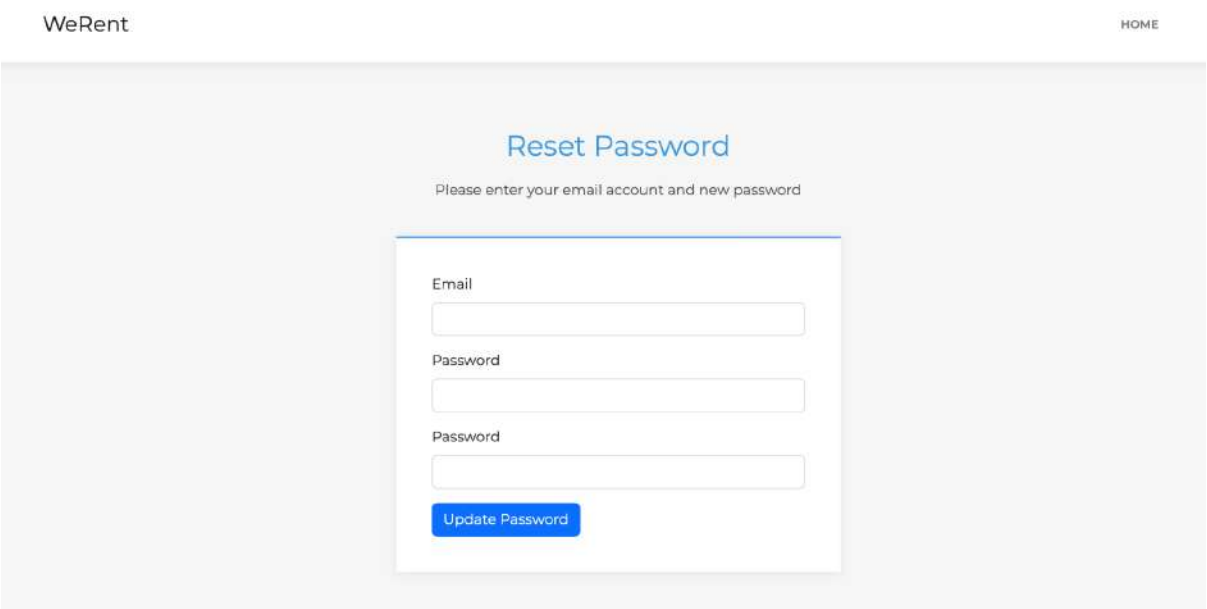


SQL Statements:

SELECT *
FROM Admin
WHERE admin-id = @id AND password = @password;

Forgot Password and Reset Password

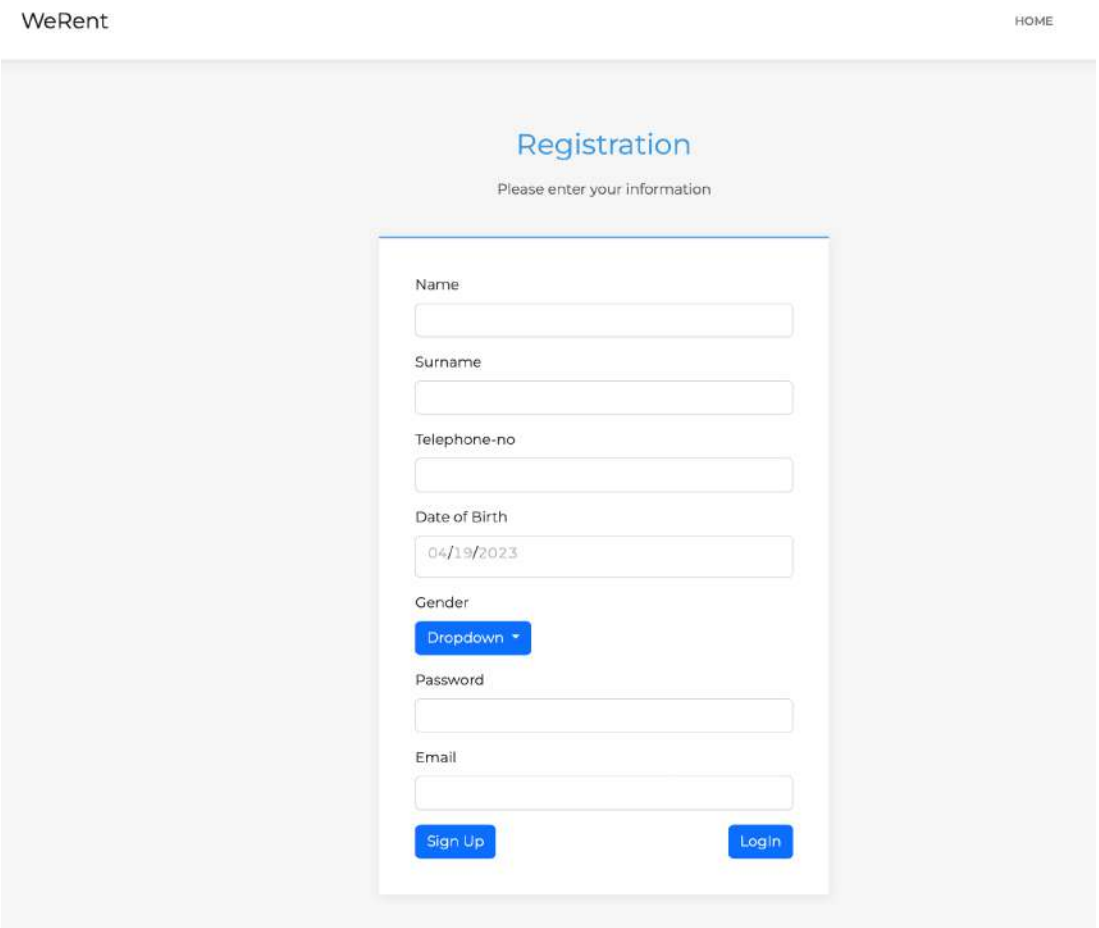




SQL Statements:

UPDATE RegisteredUser
SET RegisteredUser.password = @new_password
WHERE e-mail = @email

Register



SQL Statements:

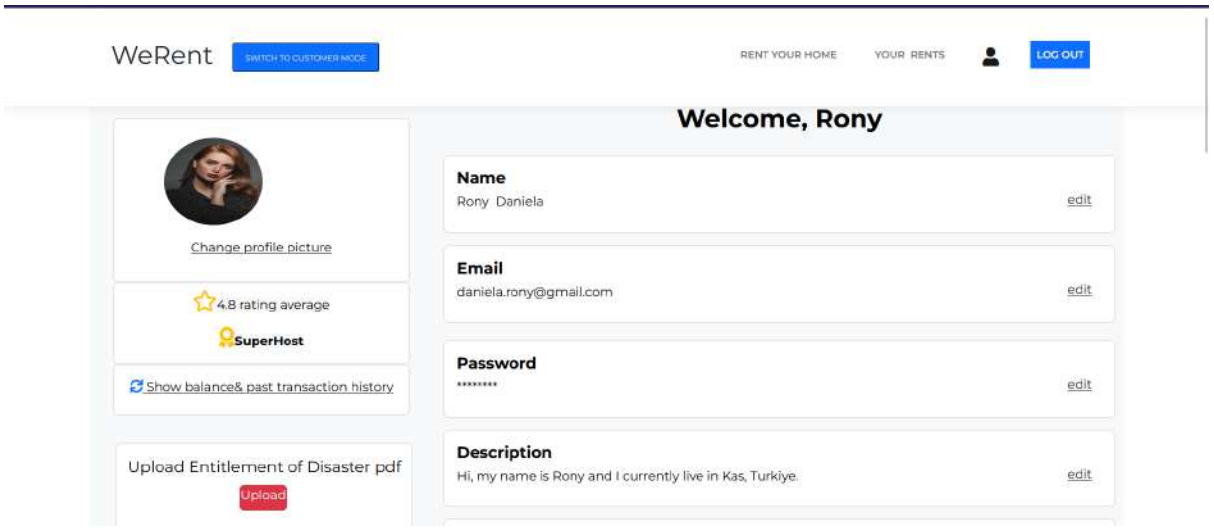
```
INSERT INTO User VALUES(
        @name,
        @surname,
        @password
);

INSERT INTO RegisteredUser(
        @email,
        @date_of_birth,
        @telephone_no,
        @gender,
        FALSE,
        0,
        0,
        "Customer",
        NULL,
```

```
        "Customer",
        @date
);
```

2.2 Topic Specific Functionality

2.2.1 Host Pages

Profile Page



SQL Statements:

To list name,email,password,description,user rating,date of birth, telephone number, gender, IBAN, job, region, languages they speak when the profile page opened (ie. listing all attributes listed in this page)(*):

```
SELECT R.name, R.email, R.password, R.description, R.user-rating,
R.date-of-birth,R.telephone-no,R.gender, H.is-superhost, H.IBAN, H.job, H.region,
H.languages  FROM RegisteredUser R, Host H WHERE user-id = @user-id AND R.user-id
= H.user-id;
```

** This query's is-superhost attribute returns true or false based on the superhost situation. Showing superhost badge and determining from true or false return value will be done by backend and frontend.*

To edit name, surname,password, email or description:

```
UPDATE RegisteredUser
SET name = SUBSTRING_INDEX(@fullname, ' ', 1),
 surname = SUBSTRING_INDEX(@fullname, ' ', -1),
 password = @password,
```

```
 email = @email,
 description = @description
WHERE user_id = @userid;
```

To edit profile photo(*****):

```
UPDATE Images
SET image-id  = @user-id,
          image-type = "profile-photo",
          image-size = @size,
         image-name = @image-name,
         image-binary-path = @image-binary-path
         WHERE user-id = @user-id;
```

******* We made an assumption that every user registered in the system has a default profile photo in the system. Then users change their profile photos, if they wish.*

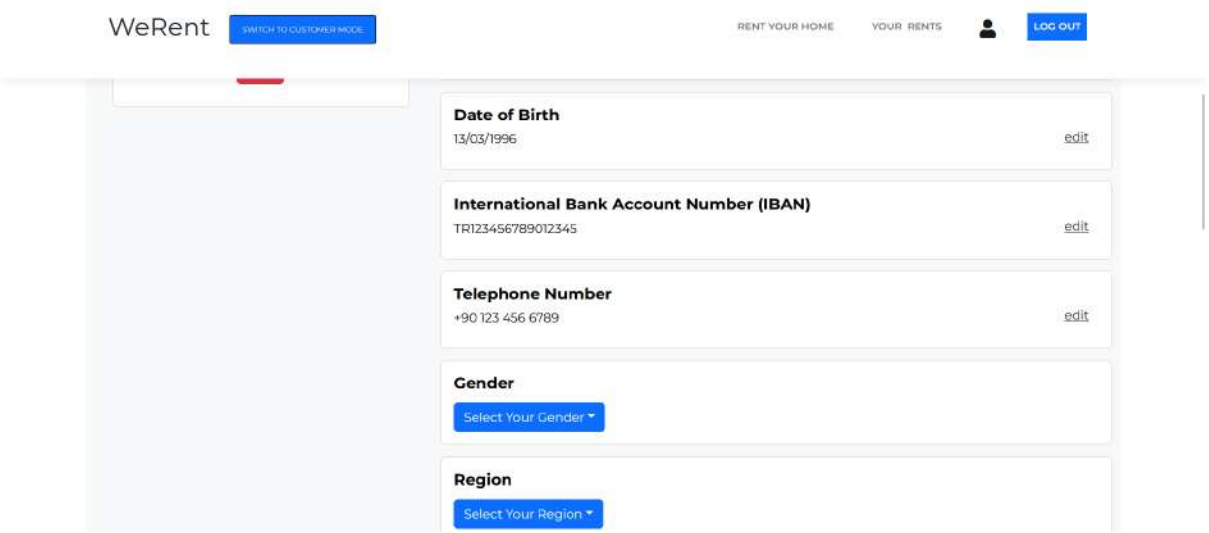
To switch customer mode(**):

```
UPDATE RegisteredUser
SET usage-mode = "customer"
WHERE user-id = @user-id;
```

*** Note that every navigation bar in host pages contains a "Switch to Customer Mode" button. Just one SQL statement about this button shown here. Other buttons also call the same SQL query.*

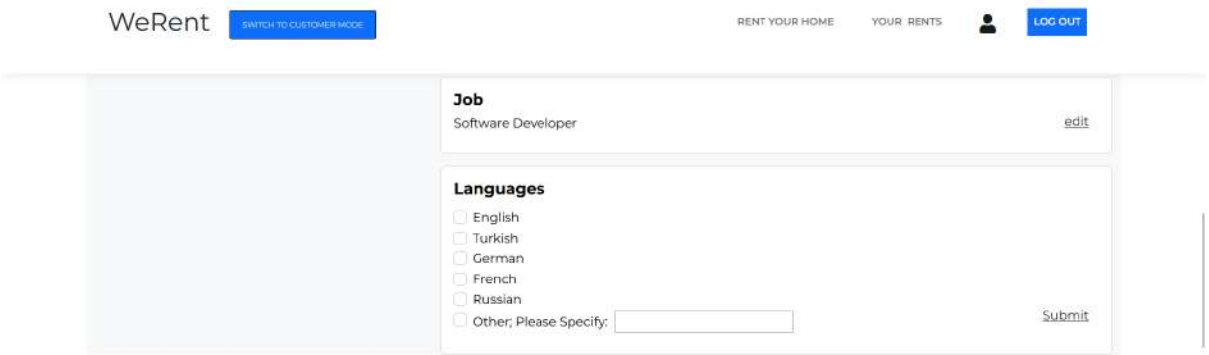
To upload Entitlement of Disaster pdf:

```
INSERT INTO E-Government File(file-name,file-size , file-extension, file-binary-path)
VALUES (@file-name,@file-size , @file-extension, @file-binary-path);
```

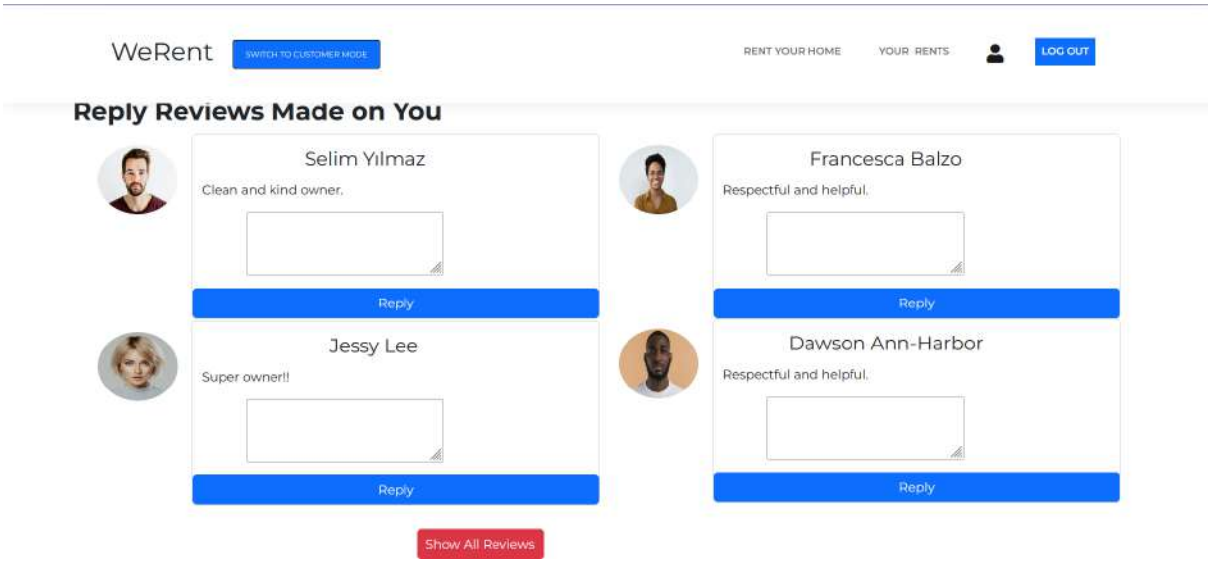


To edit date of birth, telephone-number or gender:
UPDATE RegisteredUser
SET date-of-birth = @date-of-birth, telephone-no= @telephone-no, gender = @selected-gender
WHERE user-id = @userid;

To edit IBAN or region:
UPDATE Host
SET IBAN = @iban, region = @selected-region WHERE user-id = @user-id;



To edit job or languages:
UPDATE Host
SET job = @job, languages = @selected-languages WHERE user-id = @user-id;



When Profile Page opened, reviews made on the host are listed:

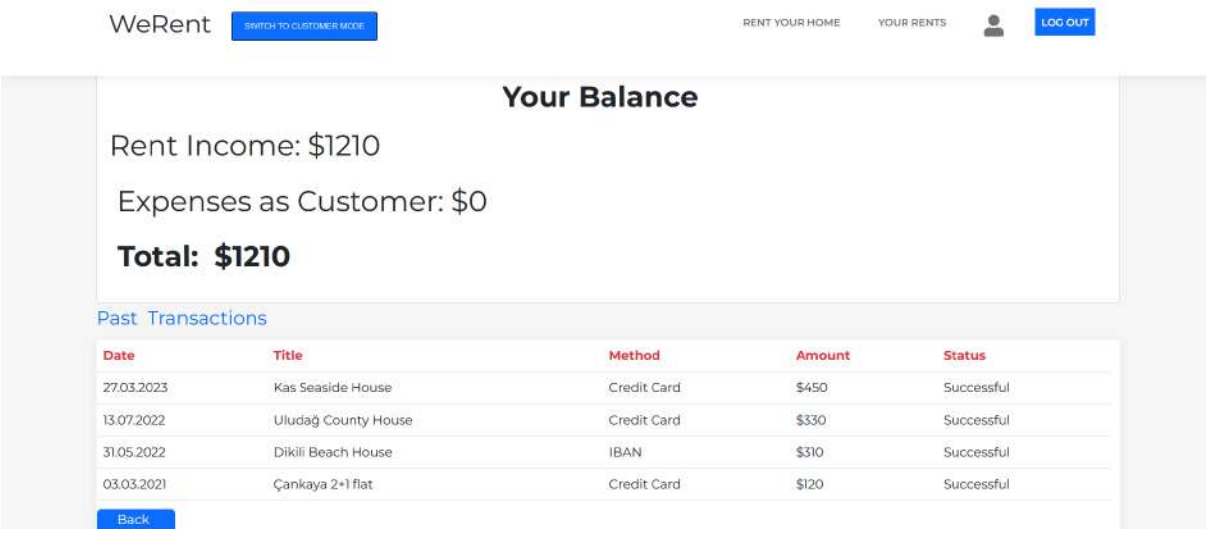
SELECT CONCAT(U.name , " ", U.surname) as full_name, C.comment-description
FROM Review R
JOIN Comments C ON R.review-id = C.review-id
JOIN User U ON C.user-id = U.user-id
WHERE R..user-id = @host_id;

To reply comments given by customers:

SELECT C.review-id, C.user-id FROM Review R JOIN Comments C ON R.review-id = C.review-id WHERE R.user-id = @host_id;

INSERT INTO Comments(review-id, user-id, comment-description, date)
VALUES(@review_id, @user_id, @comment, GETDATE());

Show Balance & Past Transactions Page



SQL Statements:

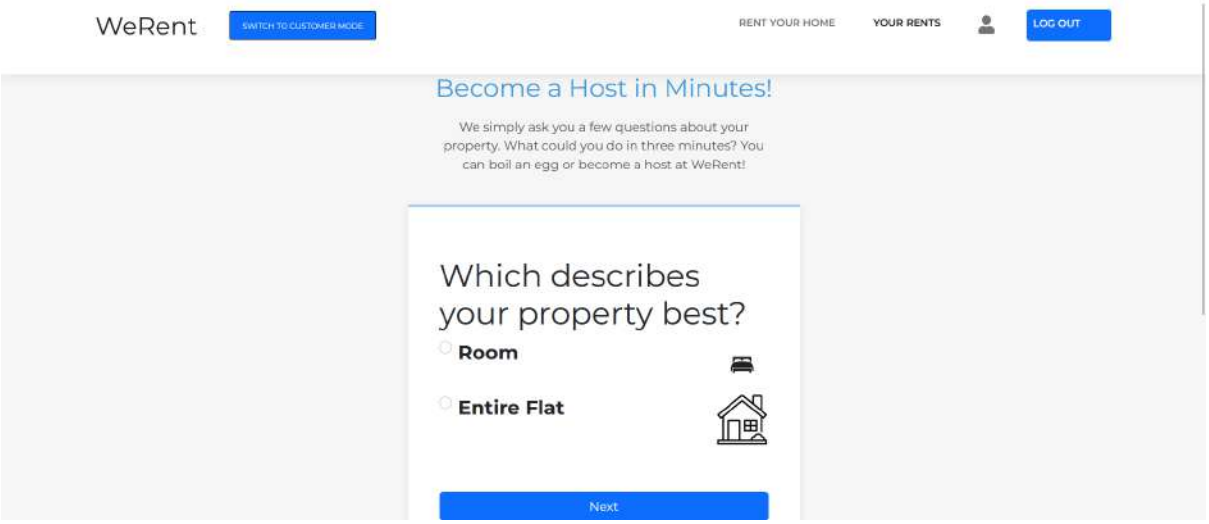
To list rent income and expenses as customer:

```
CREATE VIEW balance AS
SELECT
  CASE
    WHEN R.host_id = @user_id THEN SUM(T.amount)
    ELSE 0
  END AS rent_income,
  CASE
    WHEN C.user_id = @user_id THEN SUM(T.amount)
    ELSE 0
  END AS expenses,
  (CASE
    WHEN R.host_id = @user_id THEN SUM(T.amount)
    ELSE 0
  END) - (CASE
    WHEN C.user_id = @user_id THEN SUM(T.amount)
    ELSE 0
  END) AS total
FROM Rental R
JOIN Transaction T ON R.rental_id = T.rental_id
JOIN Customer C ON T.user_id = C.user_id
WHERE R.host_id = @user_id OR C.user_id = @user_id;
```

To list past transactions of the host:

```
SELECT T.date, R.rental-name, T.transaction-type, T.amount, T.status
FROM Transaction T, Rental R
WHERE T.rental-id = R.rental-id AND T.user-id = @user-id AND T.date < DATE(NOW())
ORDER BY T.date DESC;
```

Rent Type (Flat OR Room) Specification Page

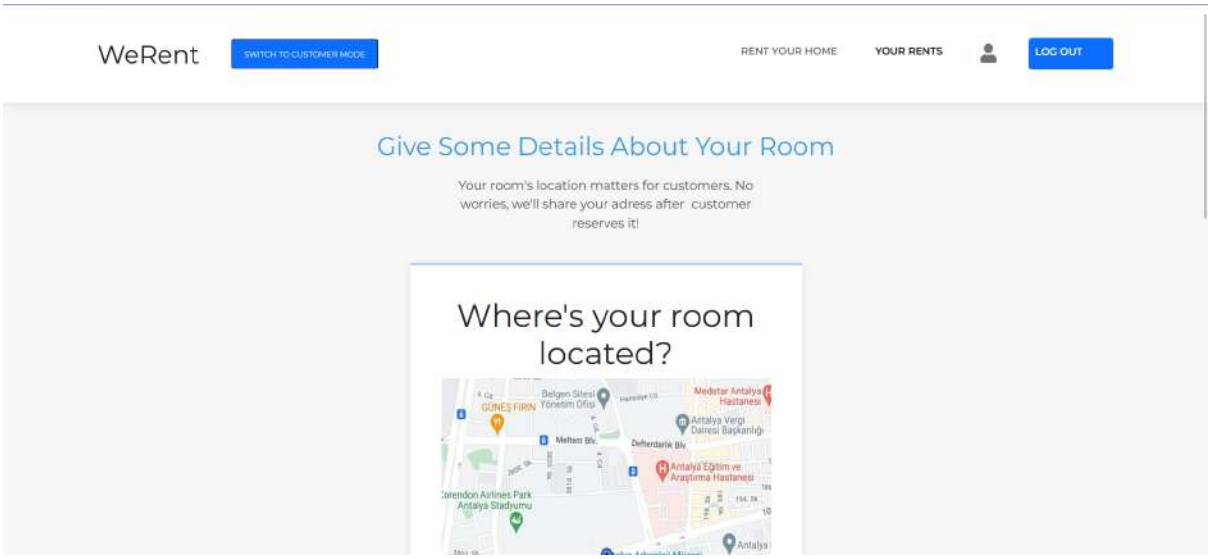


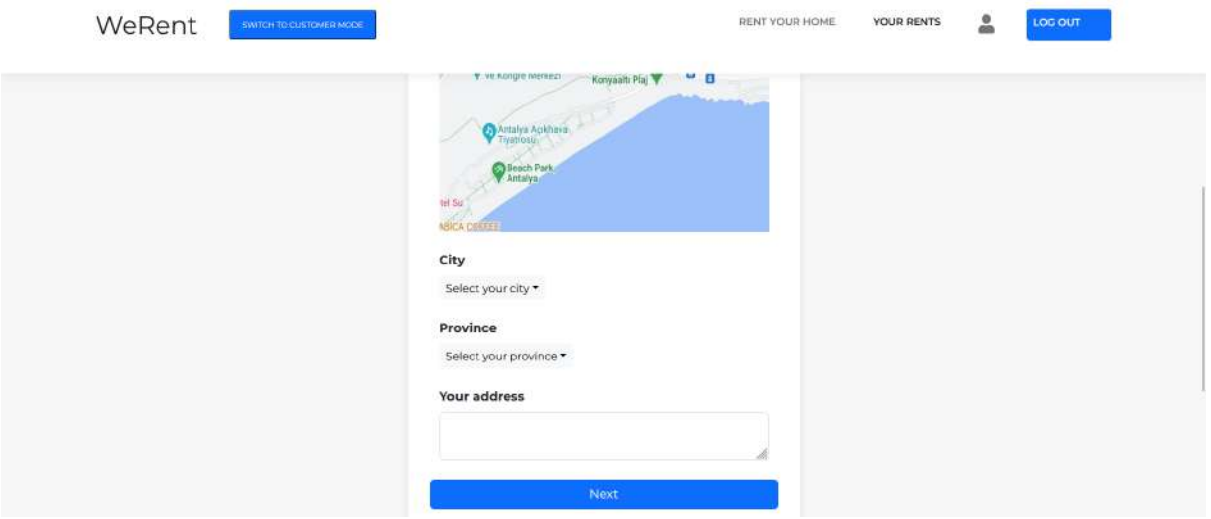
SQL Statements:

To specify property type:

```
IF (@rent_type = 'flat')
BEGIN
        INSERT INTO Flat(rental-id, flat-type, num-of-rooms, no-of-bathrooms)
        VALUES (NULL, 0, 0);
END
ELSE IF (@rent_type = 'room')
BEGIN
        INSERT INTO Room(rental-id, common-kitchen-num, common-bathroom-num,
common-living-room-num)
        VALUES (NULL, NULL, NULL);
END
```

Location Indication Page for Rooms



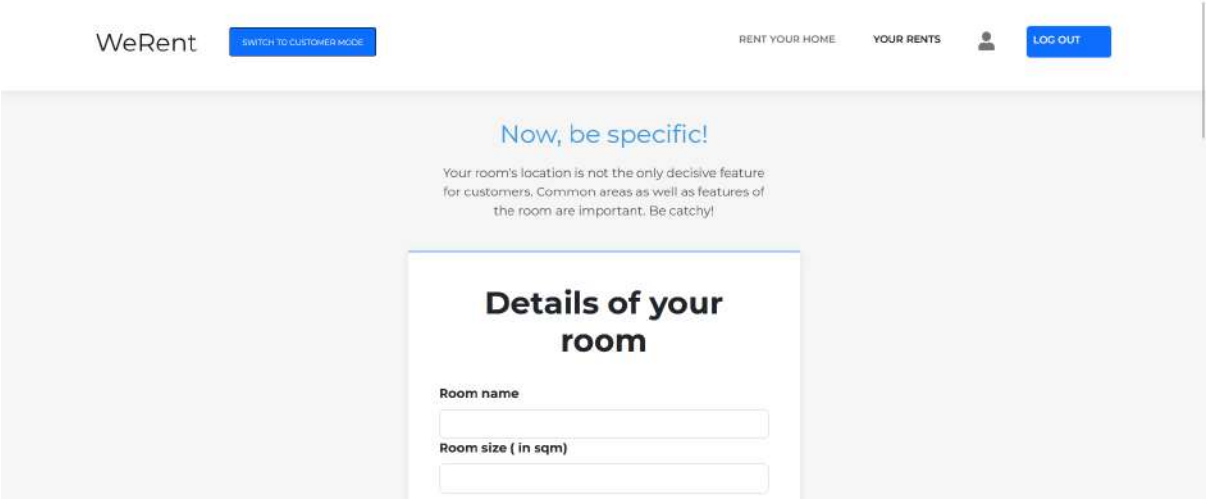


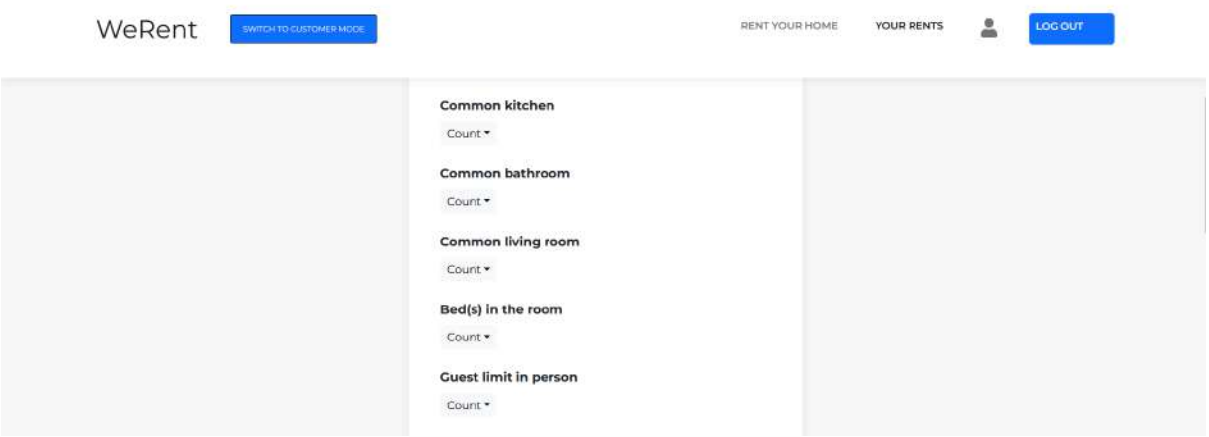
SQL Statements:

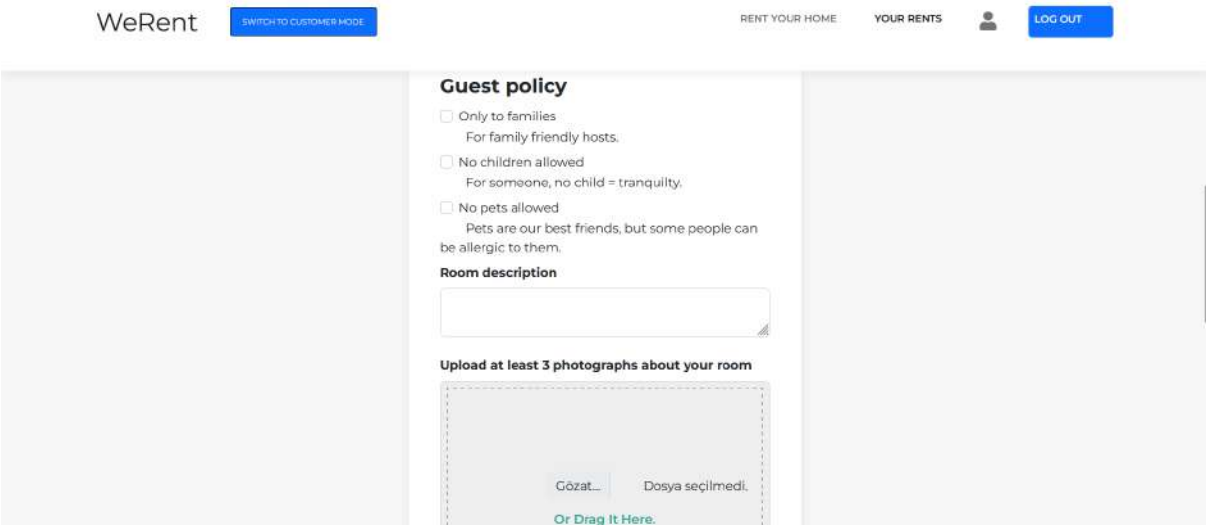
To add location of the room (by address, city and Province)(***):
INSERT INTO Rental (rental-name, host-id, guest-no, daily-price, rating, city, province, address, latitude, longitude, renting-start-date, renting-end-date, rental-type, area-in-m2, num-of-beds, description, earthquake-support, max-stay-duration, cancellation-refund, cancellation-day-limit, auto-approve-requests, is-admin-approved, couchsurfing) VALUES (“empty”,@user-id, 0, 0, 0,@city',@province, @address',@lat,@lon, '1900-01-01,’1900-01-02’, 'Room', 0, ‘0’, '', 0, 0, 0, ‘0’, ‘1’, ‘0’);

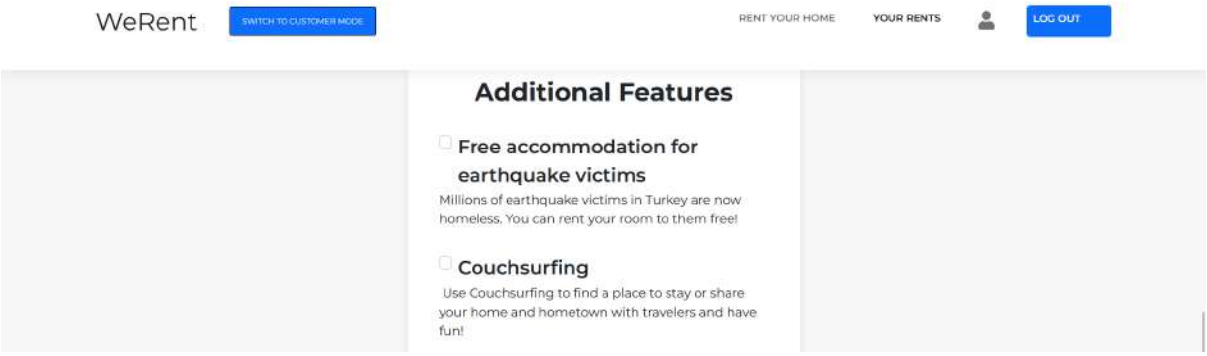
*****We should insert necessary data to the rental table. unnecessary data will be filled by empty values first, when these values are needed; the rental table is updated according to them.*

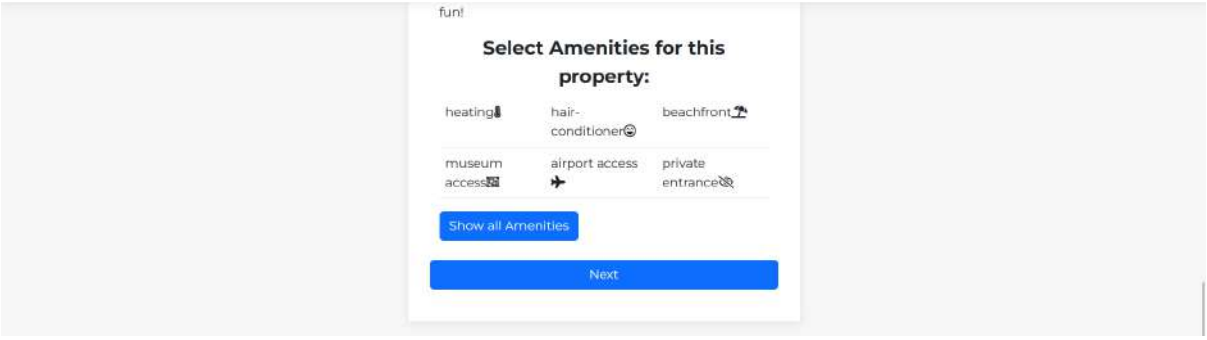
Details Indication Page for Rooms











SQL Statements:

To add input values taken in this page from the user:
UPDATE Rental
SET rental-name = @name, area-in-m2 = @room-size, guest-no = @guest-no, num-of-beds = @bed-no, description = @description, earthquake-support = @earthquake-support-selection, couchsurfing = @couchsurfing-selection WHERE host-id = @user-id AND rental-id = @rental-id;

UPDATE Room
SET common-kitchen-num = @common-kitchen-num, common-bathroom = @common-bathroom, common-living-room-num = @common-living-room-num WHERE rental-id = @rental-id;

To list all amenities:

SELECT * FROM Amenities;

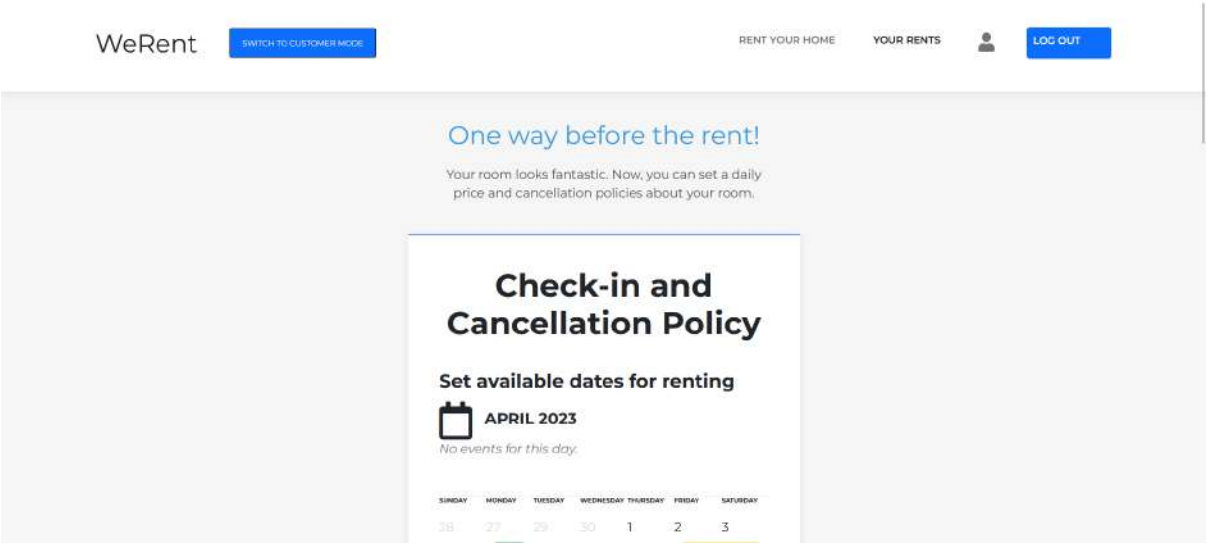
To select amenities and insert them to amenities list:

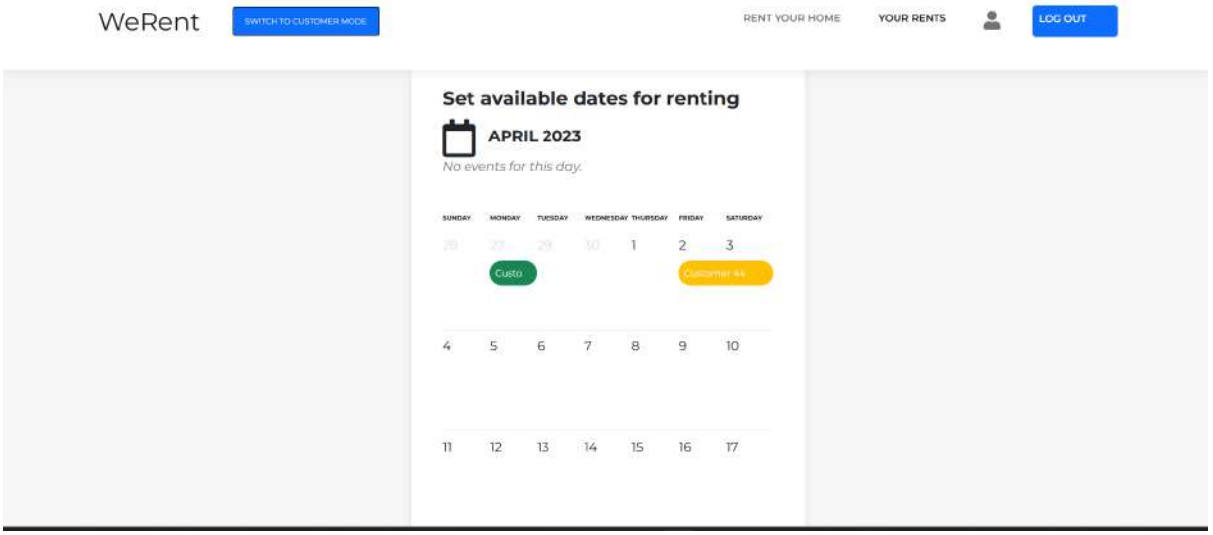
INSERT INTO Amenities (rental-id, amenity-type, smoke-alarm, fire-extinguisher, first-aid-kit, free-parking, facilities, single-level-home, washer, bed-linens, hangers, essentials, clothing-storage, AC, heating, outdoor-dining-area, bbq-grill, pets-allowed, daily-cleaning-service, transfer-service, fridge, oven, stove, dishwasher, dining-table,

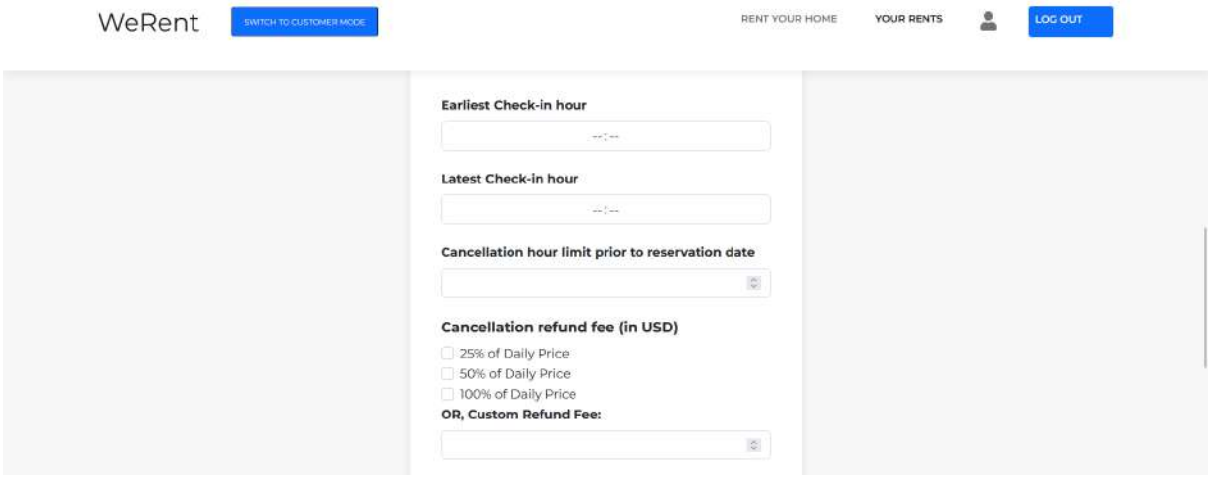
microwave, dishes, hair-conditioner, cleaning-products, shampoo, body-soap, shower-gel, hot-water, beach-access, hopping-access, museum-access, transportation-access, airport-access, private-entrance, beachfront, crib)
VALUES (@rental_id_value,@ amenity_type_value, @smoke_alarm_value, @fire_extinguisher_value, @first_aid_kit_value, @free_parking_value,@ facilities_value, @single_level_home_value, @washer_value, @bed_linens_value, @hangers_value, @essentials_value, @clothing_storage_value, @AC_value, heating_value, @outdoor_dining_area_value, @bbq_grill_value, @pets_allowed_value, @daily_cleaning_service_value, @transfer_service_value, @fridge_value, @oven_value, @stove_value, @dishwasher_value, @dining_table_value, @microwave_value, @dishes_value, @hair_conditioner_value, @cleaning_products_value, @shampoo_value, @body_soap_value, @shower_gel_value, @hot_water_value,@ beach_access_value, @hopping_access_value, @museum_access_value, @transportation_access_value, @airport_access_value, @private_entrance_value, @beachfront_value, @crib_value);

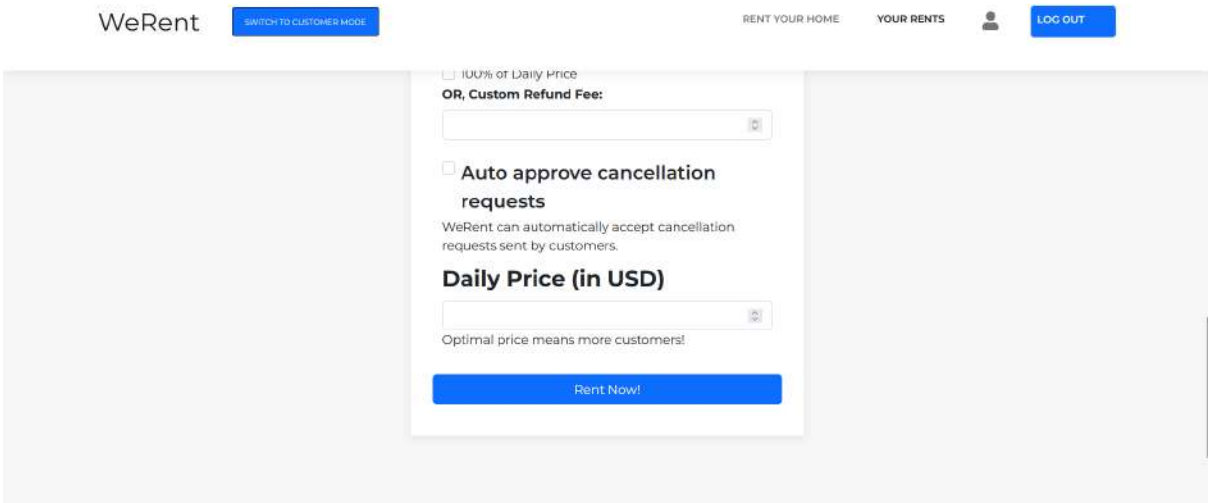
To insert images taken by the user to the images table:
INSERT INTO Images (user-id, image-type, image-name, image-size, img-binary-path)
VALUES (@host-id, “rental-photo”,@photo-name,@photo-size,@photo-path);

Price and Cancellation Policy Setting Page for Rooms




WeRent
SWITCH TO CUSTOMER MODE
RENT YOUR HOME
YOUR RENTS
LOG OUT
Set available dates for renting
APRIL 2023
No events for this day.
SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY
Custo
Customer 44


WeRent
SWITCH TO CUSTOMER MODE
RENT YOUR HOME
YOUR RENTS
LOG OUT
Earliest Check-in hour
Latest Check-in hour
Cancellation hour limit prior to reservation date
Cancellation refund fee (in USD)
25% of Daily Price
50% of Daily Price
100% of Daily Price
OR, Custom Refund Fee:


WeRent
SWITCH TO CUSTOMER MODE
RENT YOUR HOME
YOUR RENTS
LOG OUT
OR, Custom Refund Fee:
Auto approve cancellation requests
WeRent can automatically accept cancellation requests sent by customers.
Daily Price (in USD)
Optimal price means more customers!
Rent Now!

SQL Statements:

To set available dates for renting, earliest and latest checking hour, cancellation hour limit, auto approving cancellation requests, refund fee and daily price for the room:

```
UPDATE Rental
SET host-selected-rental-start-date =@selected-start-date, host-selected-rental-end-date =
@selected-end-date,
earliest-check-in-hour = @earliest-hour, latest-check-in-hour = @latest-hour,
cancellation-hour-limit = @cancellation-hour-limit, daily-price = @price,
auto-approve-requests = @auto-approve-request-selection, cancellation-refund =
@selected-refund-fee WHERE host-selected-rental-start-date < host-selected-rental-end-date
AND  rental-id = @rental-id;
```

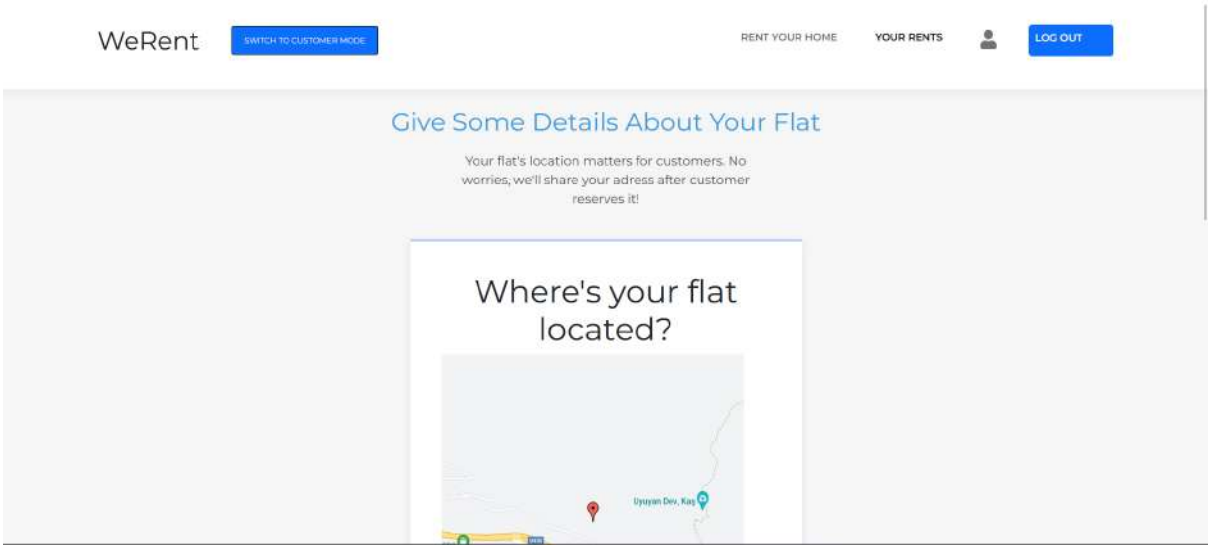
Create assertion to check end date of renting is in the future corresponding to start date of renting :

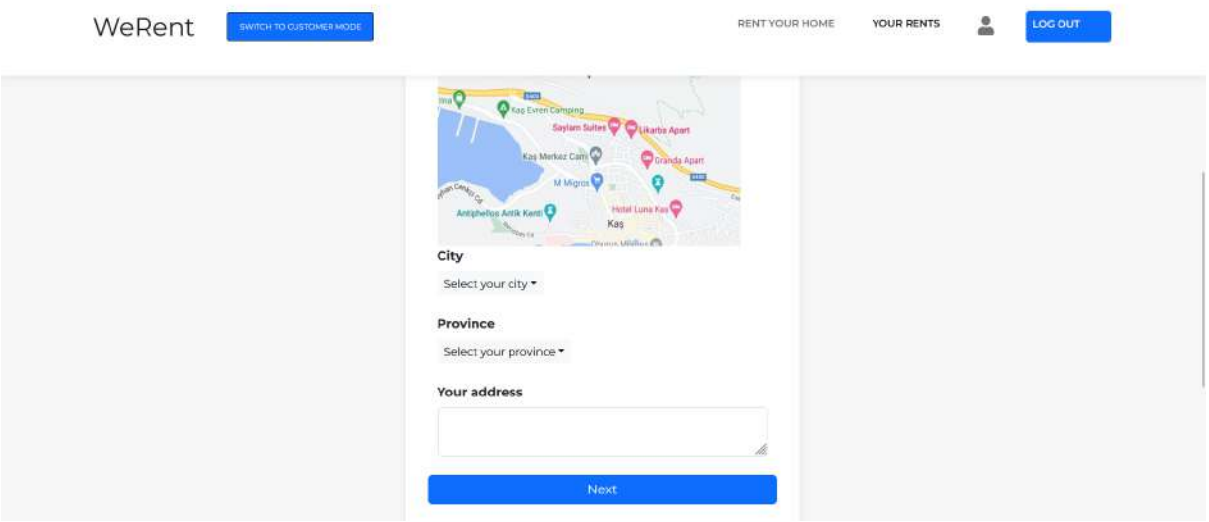
```
CREATE ASSERTION future_date_selection_checking
CHECK (
  NOT EXISTS (
        SELECT * FROM Rental
        WHERE  host-selected-rental-end-date <  host-selected-rental-start-date
  )
);
```

Create an assertion to check the daily price entered by the host is not negative:

```
CREATE ASSERTION check_daily_price_nonnegative
CHECK (
    daily-price > 0
)
```

Location Indication Page for Flats



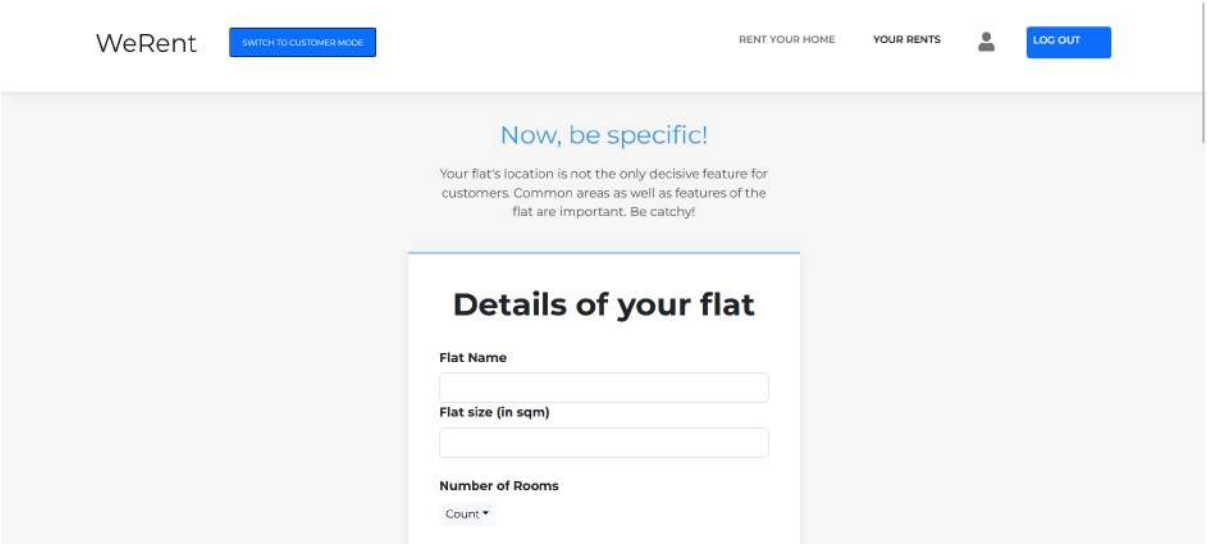


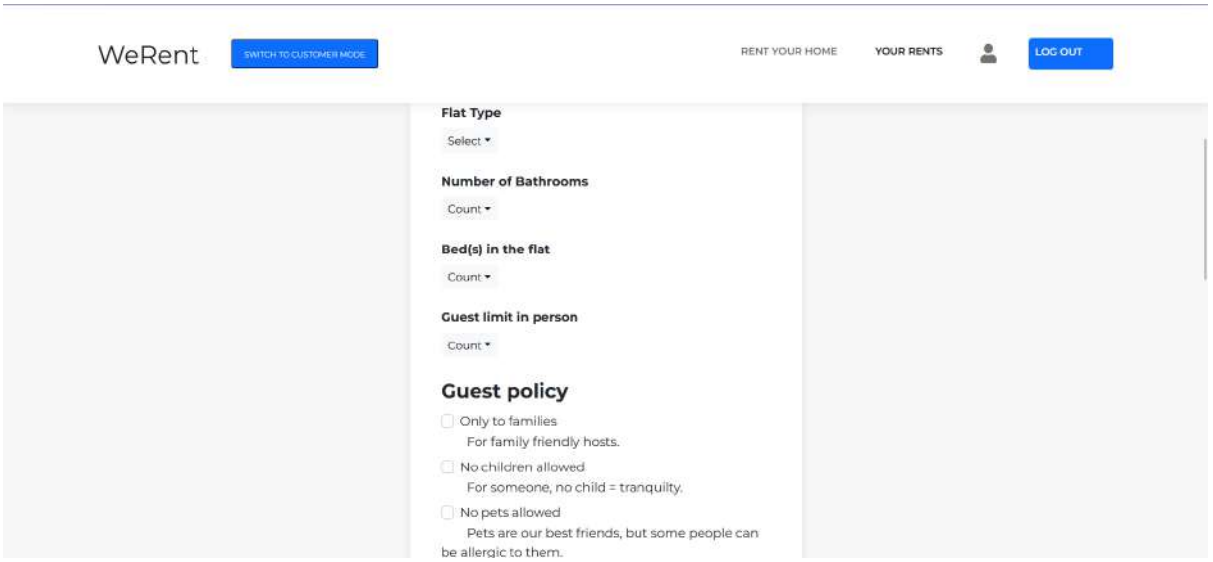
SQL Statements:

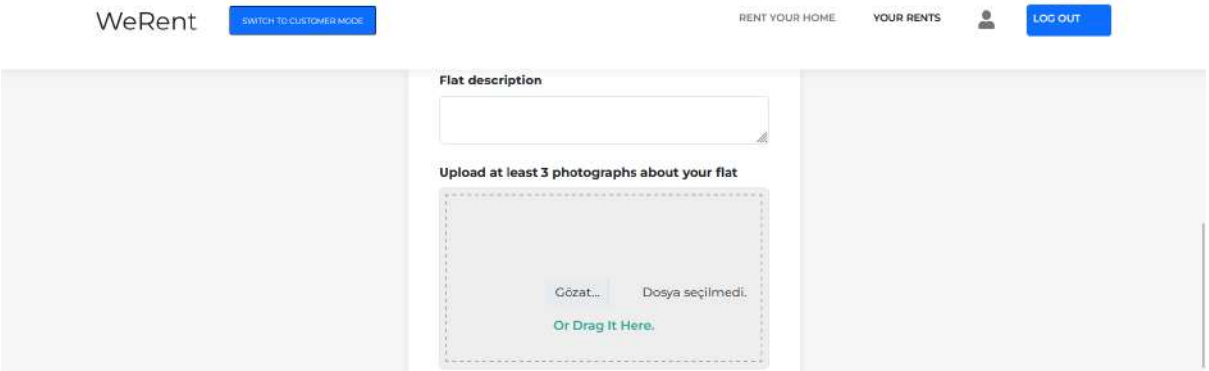
To add location of the room (by address, city and Province)(***):
INSERT INTO Rental (rental-name, host-id, guest-no, daily-price, rating, city, province, address, latitude, longitude, renting-start-date, renting-end-date, rental-type, area-in-m2, num-of-beds, description, earthquake-support, max-stay-duration, cancellation-refund, cancellation-day-limit, auto-approve-requests, is-admin-approved, couchsurfing)
VALUES (“empty”,@user-id, 0, 0, 0,@city',@province, @address',@lat,@lon, '1900-01-01,’1900-01-02’, 'Flat', 0, ‘0’, ' ', 0, 0, 0, ‘0’, ‘1’, ‘0’);

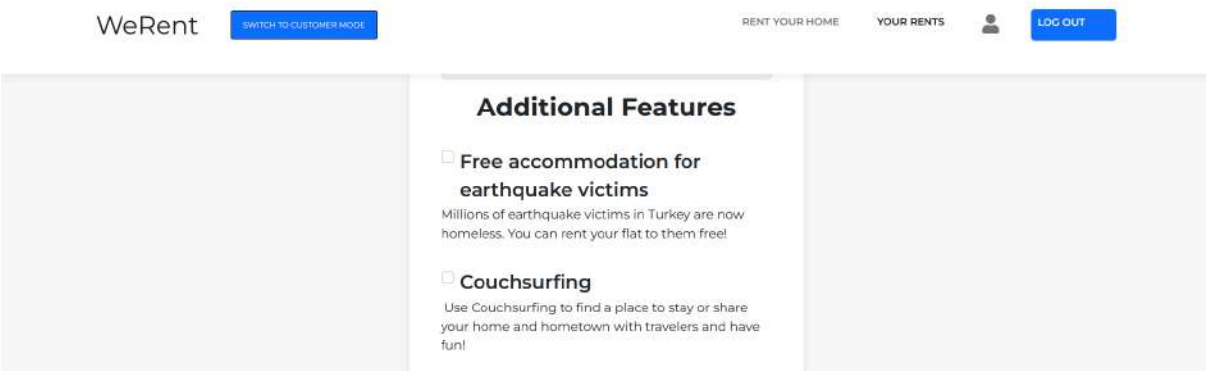
****We should insert necessary data to the rental table. unnecessary data will be filled by empty values first, when these values are needed; the rental table is updated according to them.*

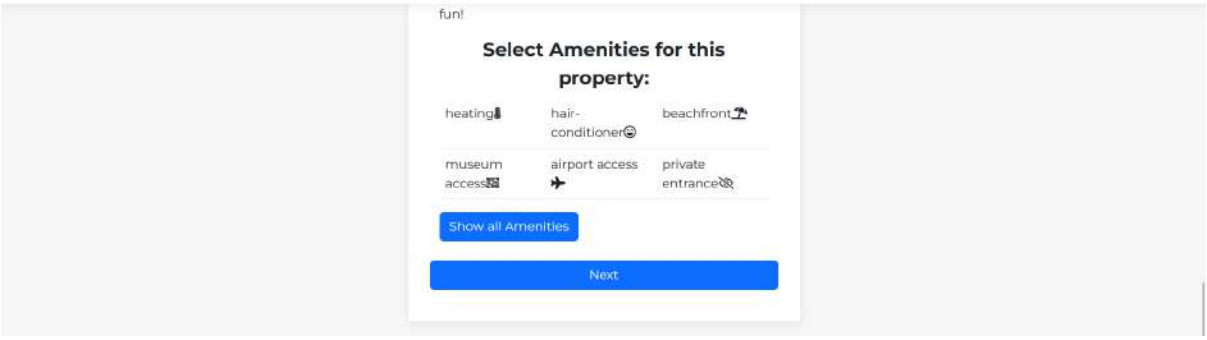
Details Indication Page for Flats











SQL Statements:

To insert input values taken in this page from the user:
UPDATE Rental
SET rental-name = @name, area-in-m2 = @flat-size, guest-no = @guest-no, num-of-beds = @bed-no, description = @description, earthquake-support = @earthquake-support-selection, couchsurfing = @couchsurfing-selection, guest-policy = @guest-policies-selected WHERE host-id = @user-id AND rental-id = @rental-id;

UPDATE Flat
SET flat-type = @flat-type, no-of-bathrooms= @bathroom-num, number-of-rooms = @room-num WHERE rental-id = @rental-id;

To list all amenities:

SELECT * FROM Amenities;

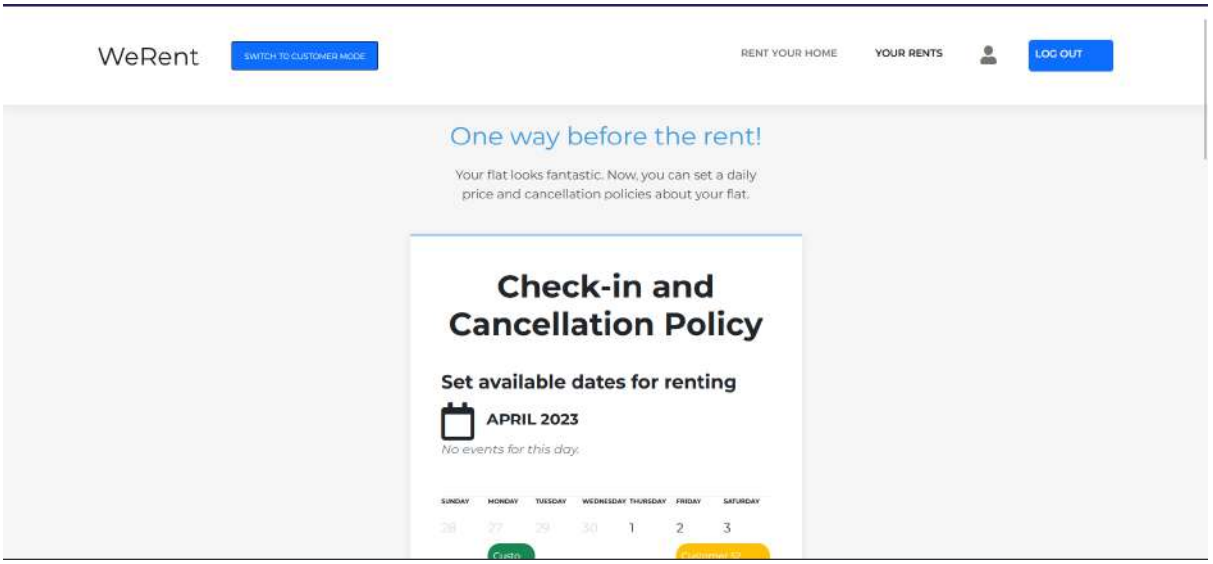
To select amenities and insert them to amenities list:

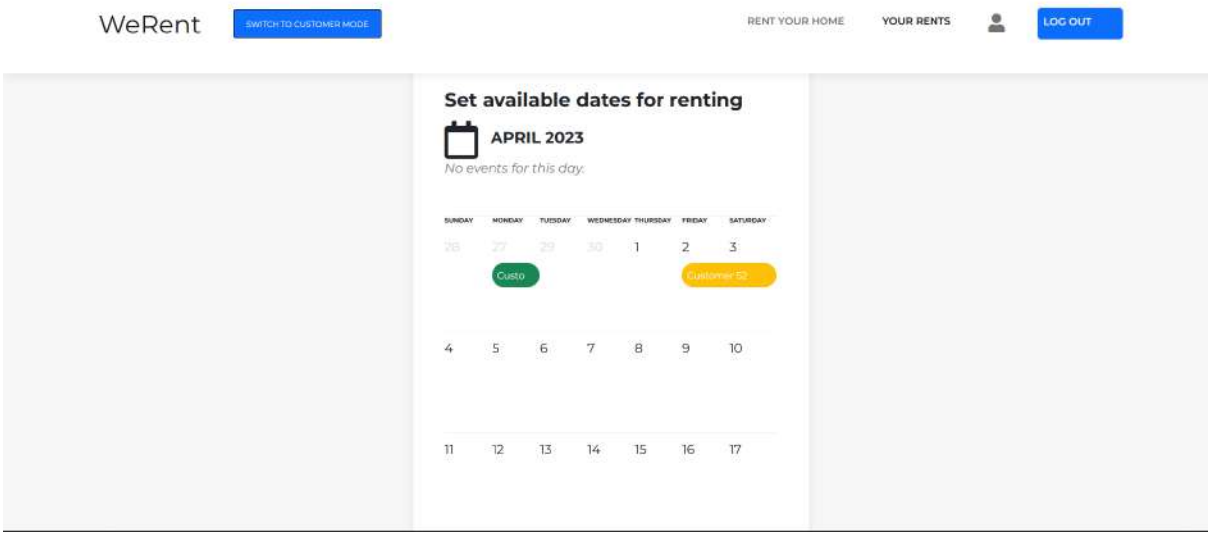
INSERT INTO Amenities (rental-id, amenity-type, smoke-alarm, fire-extinguisher, first-aid-kit, free-parking, facilities, single-level-home, washer, bed-linens, hangers, essentials, clothing-storage, AC, heating, outdoor-dining-area, bbq-grill, pets-allowed, daily-cleaning-service, transfer-service, fridge, oven, stove, dishwasher, dining-table, microwave, dishes, hair-conditioner, cleaning-products, shampoo, body-soap, shower-gel,

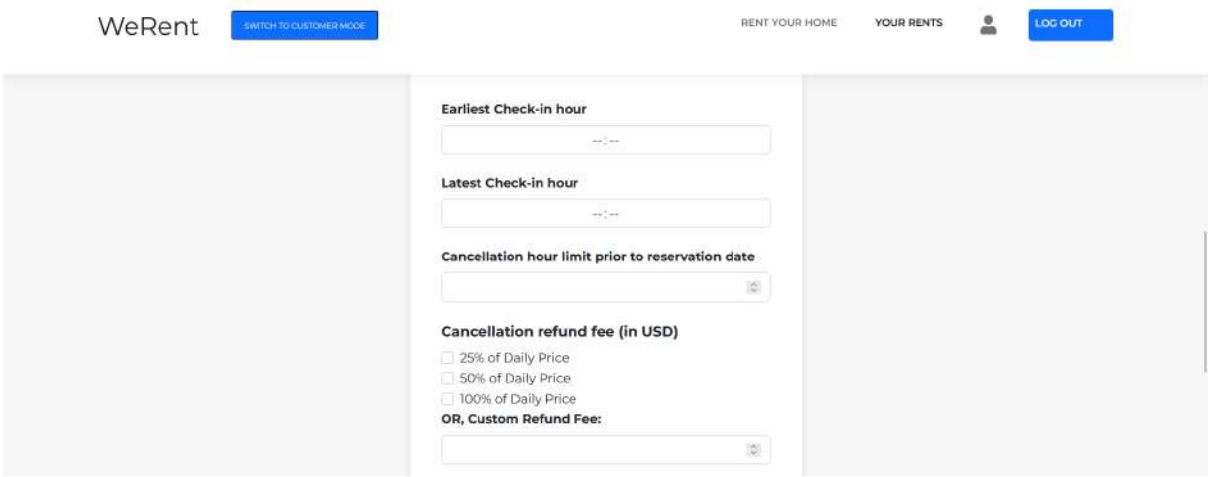
hot-water, beach-access, hopping-access, museum-access, transportation-access, airport-access, private-entrance, beachfront, crib)
VALUES (@rental_id_value,@ amenity_type_value, @smoke_alarm_value, @fire_extinguisher_value, @first_aid_kit_value, @free_parking_value,@ facilities_value, @single_level_home_value, @washer_value, @bed_linens_value, @hangers_value, @essentials_value, @clothing_storage_value, @AC_value, heating_value, @outdoor_dining_area_value, @bbq_grill_value, @pets_allowed_value, @daily_cleaning_service_value, @transfer_service_value, @fridge_value, @oven_value, @stove_value, @dishwasher_value, @dining_table_value, @microwave_value, @dishes_value, @hair_conditioner_value, @cleaning_products_value, @shampoo_value, @body_soap_value, @shower_gel_value, @hot_water_value,@ beach_access_value, @hopping_access_value, @museum_access_value, @transportation_access_value, @airport_access_value, @private_entrance_value, @beachfront_value, @crib_value);

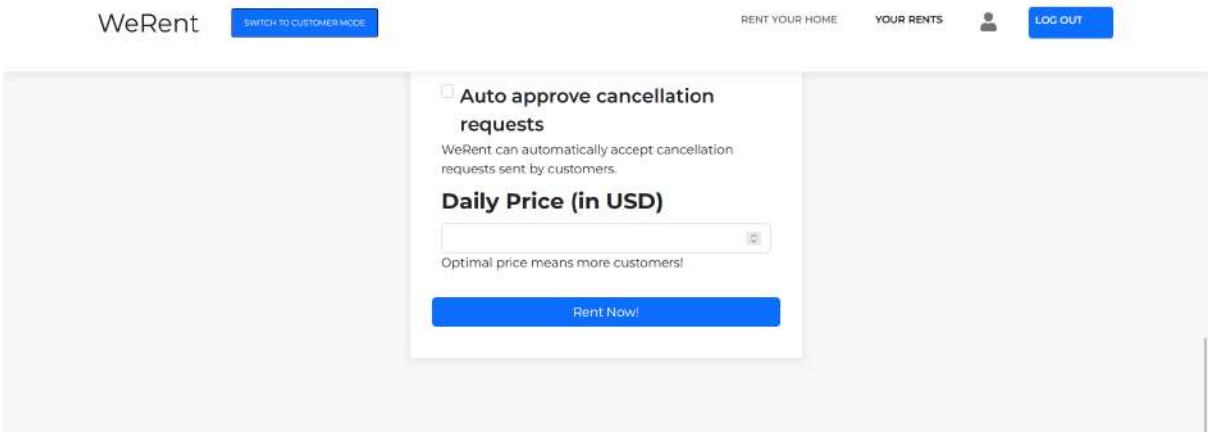
To insert images taken by the user to the images table:
INSERT INTO Images (user-id, image-type, image-name, image-size, img-binary-path)
VALUES (@host-id, “rental-photo”,@photo-name,@photo-size,@photo-path);

Price and Cancellation Policy Setting Page for Flats









SQL Statements:

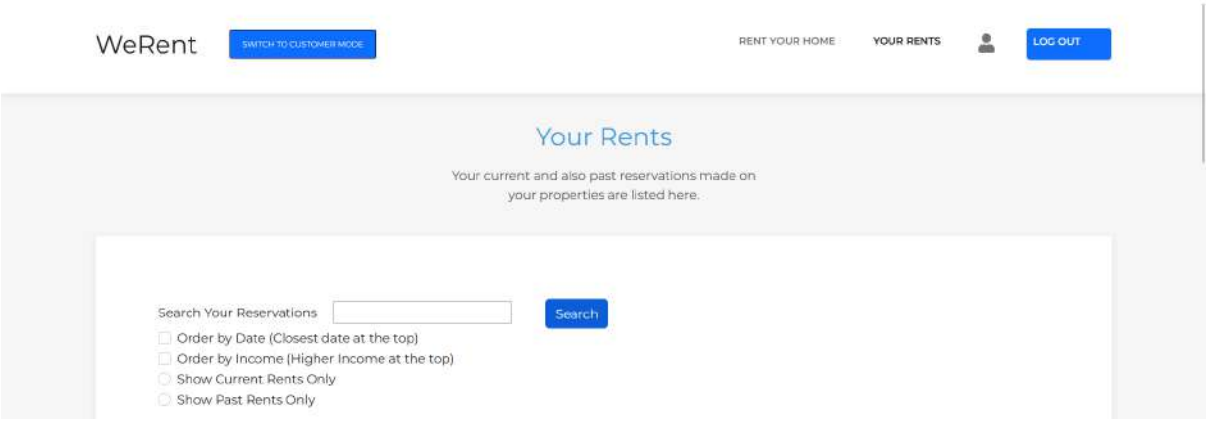
To set available dates for renting, earliest and latest checking hour, cancellation hour limit, auto approving cancellation requests, refund fee and daily price for the flat:
UPDATE Rental

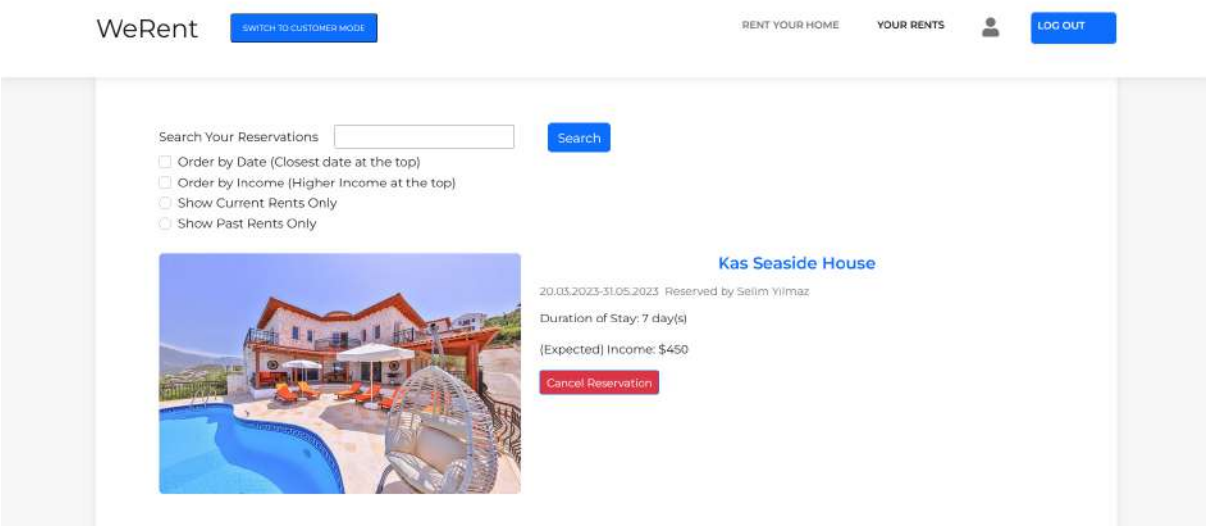
SET host-selected-rental-start-date =@selected-start-date, host-selected-rental-end-date = @selected-end-date,
earliest-check-in-hour = @earliest-hour, latest-check-in-hour = @latest-hour,
cancellation-hour-limit = @cancellation-hour-limit, daily-price = @price,
auto-approve-requests = @auto-approve-request-selection, cancellation-refund = @selected-refund-fee WHERE host-selected-rental-start-date < host-selected-rental-end-date AND rental-id = @rental-id;

Create assertion to check end date of renting is in the future corresponding to start date of renting :

```
CREATE ASSERTION future_date_selection_checking
CHECK (
  NOT EXISTS (
        SELECT * FROM Rental
        WHERE host-selected-rental-end-date < host-selected-rental-start-date
  )
);
```

Past & Current Rents Made on Host's Properties Page





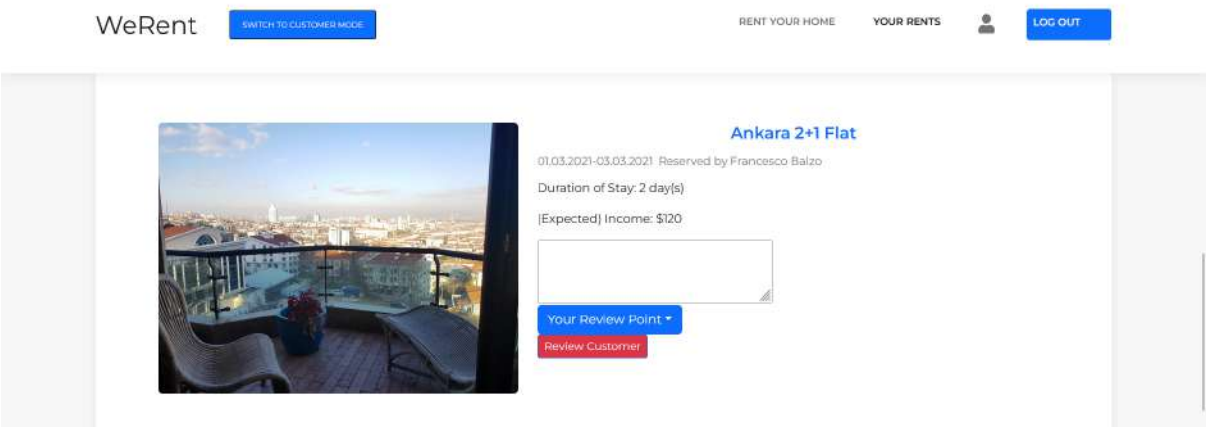
SQL Statements:

To list results from search or filters:

```
SELECT R.rental-name, R.rental-available-start-date, R.rental-available-end-date,
CONCAT(C.name, ' ', C.surname) AS customer-name,
 DATEDIFF(day, R.rental-available-start-date, R.rental-available-end-date) AS
duration-of-stay,
DATEDIFF(day, R.rental-available-start-date, R.rental-available-end-date) * R.daily-price
AS expected_income,
I.image-path
        FROM Rental R INNER JOIN Transaction T ON T.rental-id = R.rental-id
        INNER JOIN Customer C ON C.user-id = T.user-id
        LEFT JOIN Images I ON I.user-id = R.user-id
                WHERE (R.rental-name LIKE '%@search-input%' OR R.description
                LIKE '%@search-input%') AND (@filter-date-past <> 1
                OR R.rental-available-end-date < GETDATE())
                AND (@filter-date-future <> 1 OR R.rental-available-start-date >
                GETDATE())
ORDER BY
        CASE WHEN @filter-by-income = 1
        THEN expected_income
        ELSE rental-available-end-date END
DESC;
```

To cancel reservation (whose end date is in the future):

```
DELETE FROM Reservation
WHERE reservation-id = @reservation_id
        AND reservation-end-date > GETDATE();
```



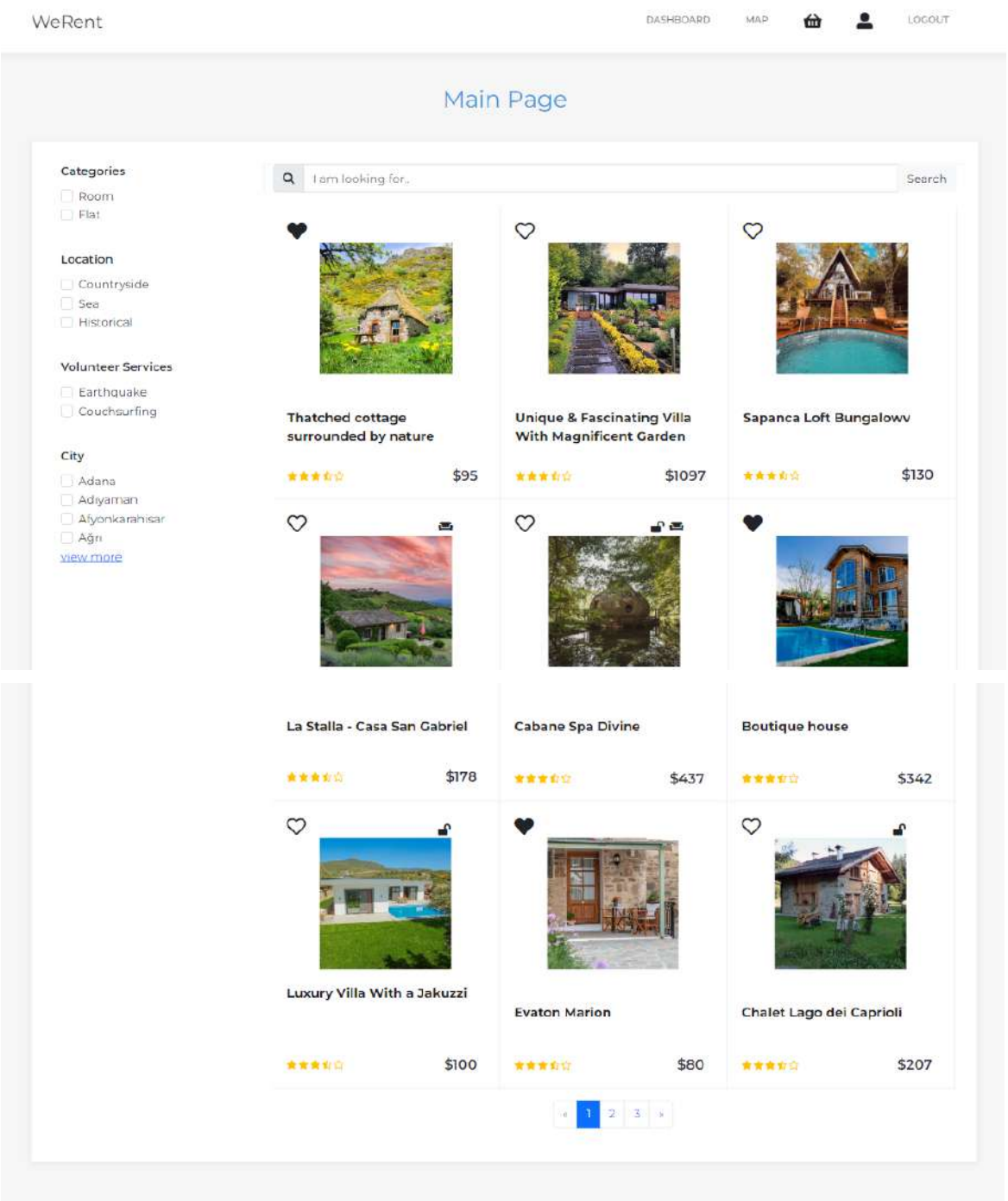
To make a review on customers by the host:

INSERT INTO Review(review, cleanliness-rating, check-in-rating, communication-rating, accuracy-rating, safety-rating, location-rating, value-rating, general-rating, is-anonymous)
OUTPUT inserted.review-id INTO @review_id
VALUES(@comment,-1,1, -1, 1, 1, -1, (@review-point)*7, 0);

INSERT INTO Comments(review-id, user-id, comment-description, date)
VALUES(@review_id, (SELECT user-id2 FROM Leaves WHERE reservation-id = @reservation_id AND user-id1 = @host_id),@comment, GETDATE());

2.2.2 Customer Pages

Main Page



SQL Statements:

To list all rentals and whether they are favorited:

SELECT rental.rental-id, rental.rental-name, rental.daily-price, rental.rating, rental.city, rental.earthquake-support, rental.couch-surfing,

CASE WHEN wishlists.rental-id IS NULL THEN 0 ELSE 1 END AS is-favorited

```
FROM Rental rental
LEFT JOIN (
        SELECT rental-id
        FROM Wishlists wishlists
        WHERE user-id = @registered-user-id
        ) AS wishlists ON rental.rental-id = wishlists.rental-id;
```

To favorite a rental:

```
INSERT INTO Wishlists
VALUES ( @registered-user-id, @clicked-rental-id, @current-date);
```

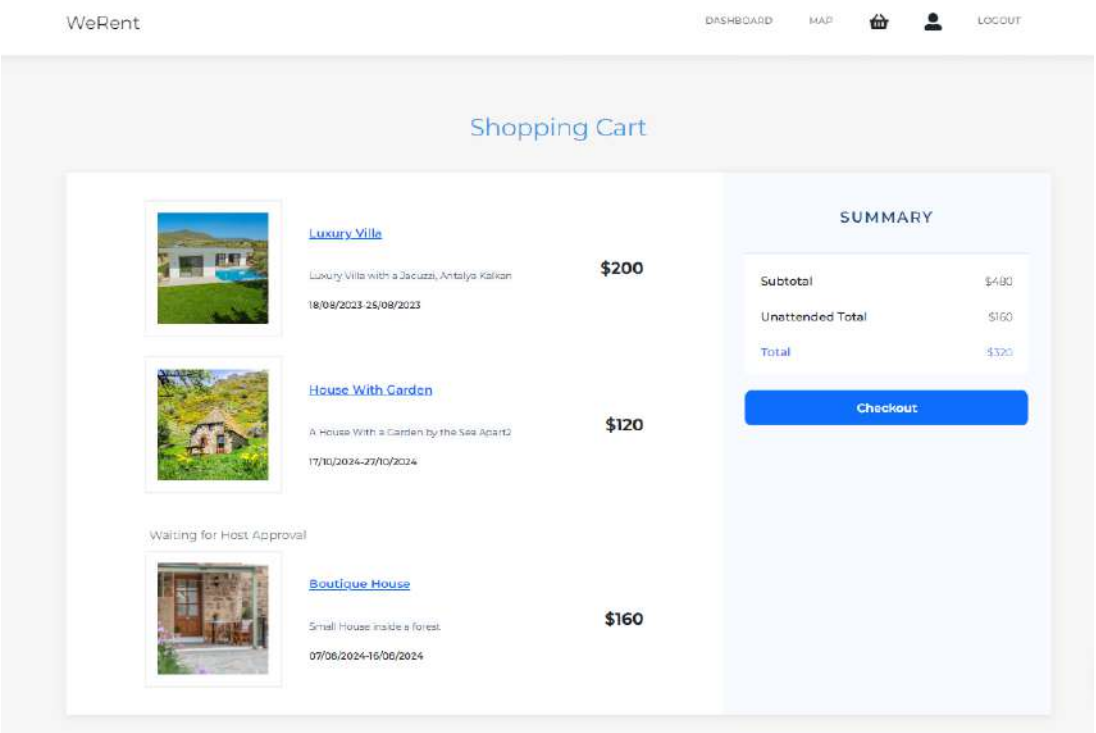
To unfavorite a rental:

```
DELETE FROM Wishlists
WHERE user-id = @registered-user-id AND rental-id = @clicked-rental-id;
```

To list results from search or filters:

```
SELECT rental-id, rental-name, daily-price, rating, city, earthquake-support, couch-surfing
FROM Rental
WHERE ( rental-name LIKE '%@search-input%' OR description LIKE '%@search-input%')
        AND CASE WHEN @filter-city = 1 THEN city END = @chosen-city
        AND CASE WHEN @filter-category = 1 THEN rental-type END =
        @chosen-category;
```

Shopping Cart Page



SQL Statements:

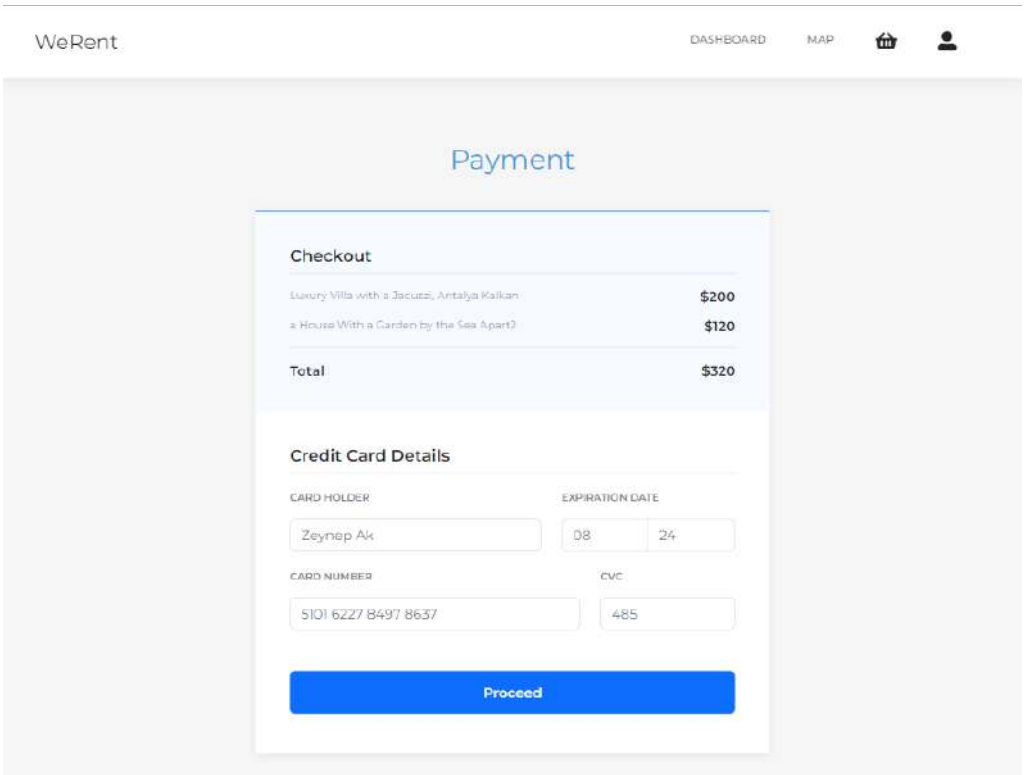
To list the ready to be paid rentals:

SELECT R.rental-name, R.description, RES.reservation-start-date, RES.reservation-end-date, RES.stay-of-duration, R.daily-price * RES.stay-of-duration AS whole-price
FROM Rental R, Reservation RES, ShoppingCart S
WHERE R.rental-id = S.rental-id AND RES.reservation-id = S.reservation-id AND S.user-id = @registered-user-id
AND (R.auto-approve-requests = ‘true’ OR (R.auto-approve-requests = ‘false’ AND RES.is-approved-by-host = ‘true’))

To list the waiting for host approval rentals:

SELECT R.rental-name, R.description, RES.reservation-start-date, RES.reservation-end-date, RES.stay-of-duration, R.daily-price * RES.stay-of-duration AS whole-price
FROM Rental R, Reservation RES, ShoppingCart S
WHERE R.rental-id = S.rental-id AND RES.reservation-id = S.reservation-id AND S.user-id = @registered-user-id
AND (R.auto-approve-requests = ‘false’ AND RES.is-approved-by-host = ‘false’)

Payment Page



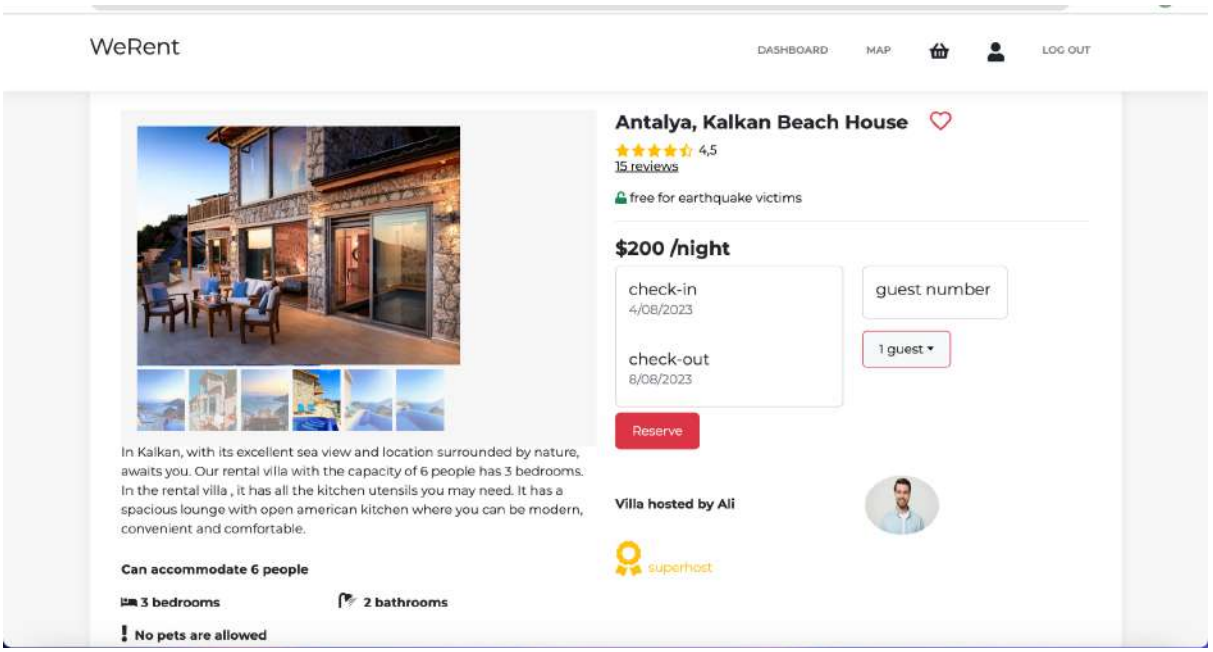
SQL Statements:

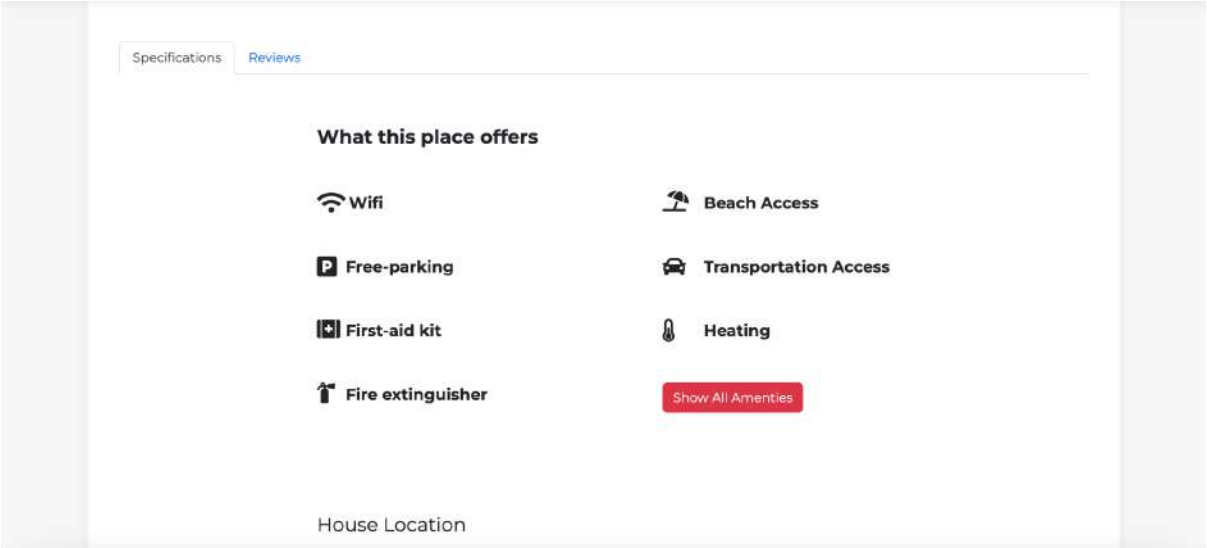
To list the rentals:

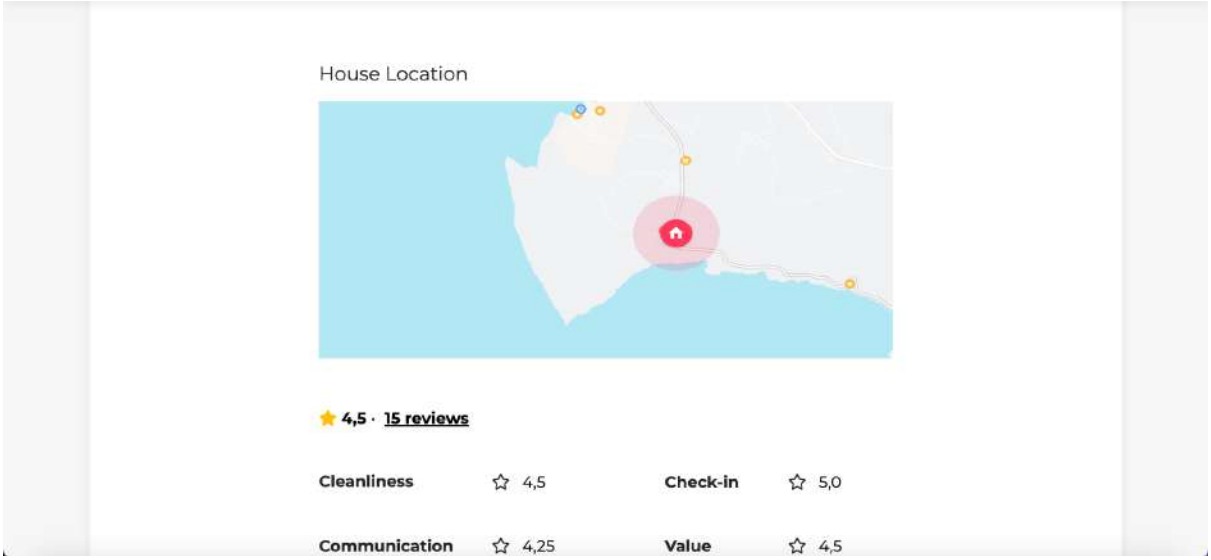
SELECT R.rental-name, R.daily-price * RES.stay-of-duration AS whole-price
FROM Rental R, Reservation RES, ShoppingCart S
WHERE R.rental-id = S.rental-id AND RES.reservation-id = S.reservation-id AND S.user-id = @registered-user-id AND (R.auto-approve-requests = ‘true’ OR (R.auto-approve-requests = ‘false’ AND RES.is-approved-by-host = ‘true’))

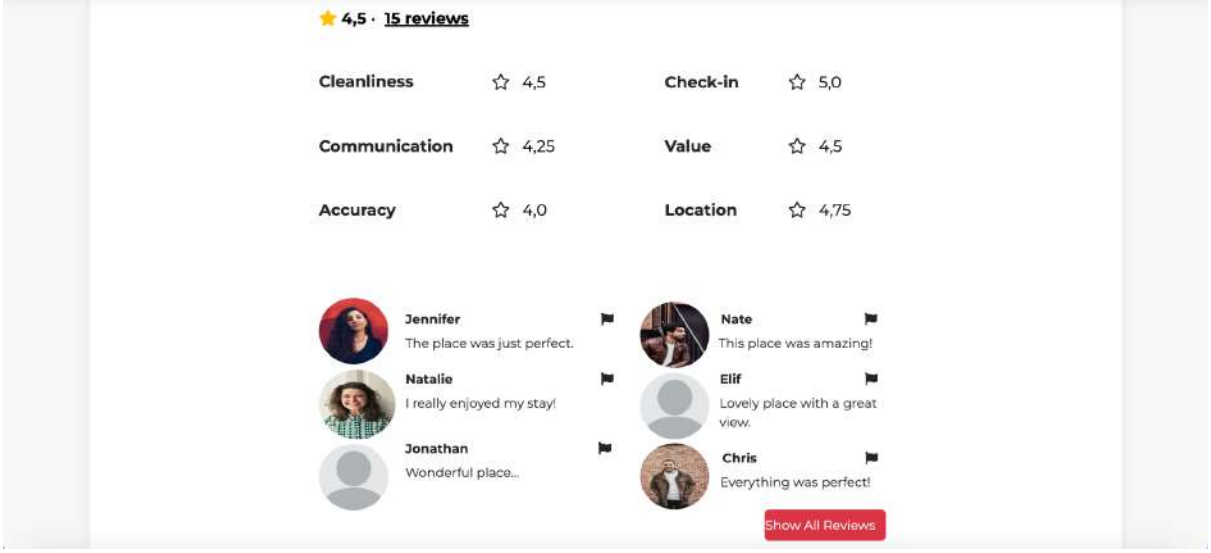
To insert the transaction:
INSERT INTO Transaction
VALUES (@reservation-id, @registered-user-id, ‘Payment’, @current-date, NULL, @amount);

Rental Page




Specifications
Reviews
What this place offers
Wifi
Beach Access
Free-parking
Transportation Access
First-aid kit
Heating
Fire extinguisher
Show All Amenties
House Location


House Location
4,5 · 15 reviews
Cleanliness 4,5
Check-in 5,0
Communication 4,25
Value 4,5


4,5 · 15 reviews
Cleanliness 4,5
Check-in 5,0
Communication 4,25
Value 4,5
Accuracy 4,0
Location 4,75
Jennifer
The place was just perfect.
Nate
This place was amazing!
Natalie
I really enjoyed my stay!
Elif
Lovely place with a great view.
Jonathan
Wonderful place...
Chris
Everything was perfect!
Show All Reviews

SQL Statements:

To list the rental information:

```
SELECT rental-name,daily-price, description,city,province,earthquake-support,couchsurfing
FROM Rental
WHERE rental-id=@rental-id;
```

To display whether the rental is favorited or not:

```
SELECT EXISTS (
  SELECT *
  FROM Wishlists
  WHERE user-id = @registered-user-id   AND rental-id = @rental-id );
```

To favorite a rental:

```
INSERT INTO Wishlists
VALUES(@registered-user-id,@rental-id,@current-date);
```

To unfavorite a rental:

```
DELETE FROM Wishlists
WHERE user-id = @registered-user-id
AND rental-id = @rental-id;
```

To display the general rating average for the rental:

```
SELECT rating
FROM Rental
WHERE  rental-id=@rental-id;
```

To display host information:

```
SELECT name,is-super-host
FROM Host
WHERE user-id=@user-id;
```

To add a reservation to the shopping cart:

```
INSERT INTO
Reservation(rental-id,reservation-start-date,reservation-end-date,stay-of-duration,is-paid-for,
is-approved-by-host,number-of-guests) VALUES (@rental-id,@reservation-start-date ,
@reservation-end-date, @stay-of-duration,FALSE,NULL,@number-of-guests);
```

To display the location of the rental:

```
SELECT latitude,longitude
FROM Rental
```

WHERE rental-id = @rental-id

To display amenities based on amenity types:

```
SELECT smoke-alarm,fire-extinguisher,first-aid-kit
FROM Amenities
WHERE amenity-type = @amenity-type1;

SELECT free-parking,facilities,single-level-home
FROM Amenities
WHERE amenity-type = @amenity-type2;

SELECT washer,bed-linens,hangers,essentials,clothing-storage
FROM Amenities
WHERE amenity-type = @amenity-type3;

SELECT AC, Heating
FROM Amenities
WHERE amenity-type = @amenity-type4;

SELECT outdoor-dining-area, bbq-grill
FROM Amenities
WHERE amenity-type = @amenity-type5;

SELECT pets-allowed,cleaning-service,transfer-service
FROM Amenities
WHERE amenity-type = @amenity-type6;

SELECT fridge,oven,stove,dishwasher,dining-table,microwave,dishes
FROM Amenities
WHERE amenity-type = @amenity-type7;

SELECT hair-conditioner,cleaning-products,shampoo,body-soap,shower-gel,hot-water
FROM Amenities
WHERE amenity-type = @amenity-type8;

SELECT crib
FROM Amenities
WHERE amenity-type = @amenity-type9;

SELECT beach-access,hopping-access,museum-access,airport-acces,transportation-access,
private-entrance,beachfront
FROM Amenities
WHERE amenity-type = @amenity-type10;
```

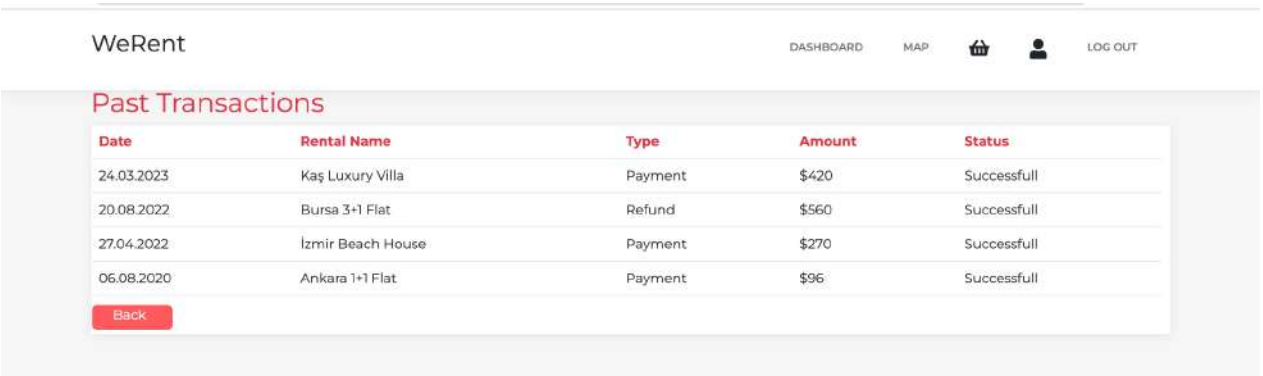
To display average ratings the rental and the host has on each category:

```
 SELECT AVG(Review.cleanliness-rating) AS avg-cleanliness,
AVG(Review.check-in-rating) AS avg-check-in,
AVG(Review.communication-rating) AS avg-communication,
AVG(Review.accuracy-rating) AS avg-accuracy,
AVG(Review.safety-rating) AS avg-safety,
AVG(Review.location-rating) AS avg-location,
AVG(Review.value-rating) AS avg-value

FROM Leaves
JOIN Review ON Leaves.review-id= Review.review.id
JOIN Makes ON Leaves.reservation-id= Makes.reservation-id
JOIN Rental ON Makes.rental-id=Rental.rental-id
WHERE  Leaves.user-id2 = @user-id AND Rental.rental-id=@rental-id;
```

To display the reviews for the rental:

```
SELECT RegisteredUser.name, Review.review
FROM Leaves
JOIN Review ON Leaves.review-id= Review.review.id
JOIN Makes ON Leaves.reservation-id= Makes.reservation-id
JOIN Rental ON Makes.rental-id=Rental.rental-id
JOIN RegisteredUser ON Leaves.user-id1=RegisteredUser.user-id
WHERE  Leaves.user-id2 = @user-id AND Rental.rental-id=@rental-id
```

Past Transactions Page

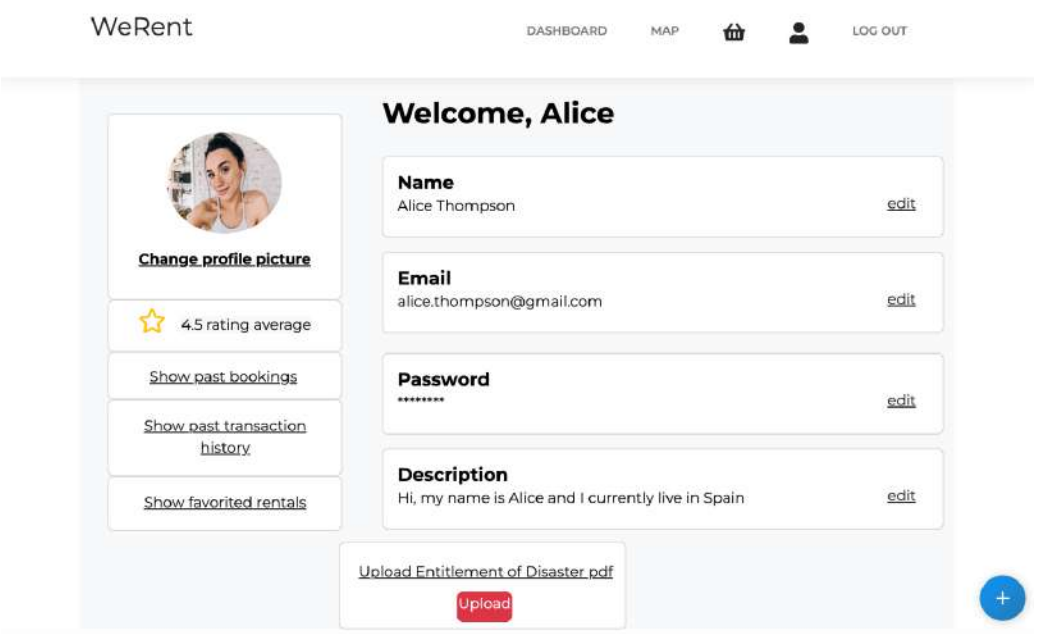


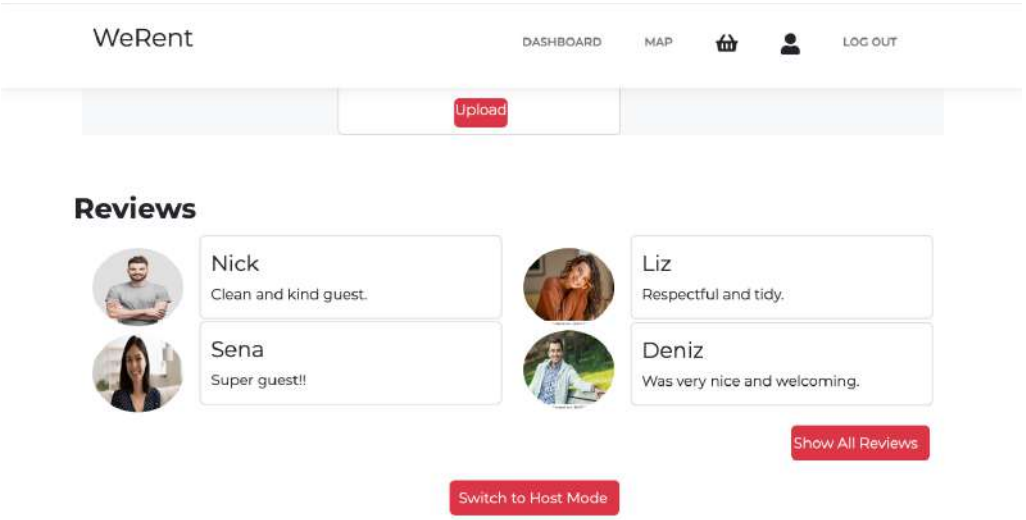
SQL Statements:

To display past transactions ordered by their date:

```
SELECT Transaction.date,Rental.rental-name,Transaction.transaction-type,
Transaction.amount, Transaction.status
FROM Transaction
JOIN Rental ON Transaction.rental-id=Rental.rental-id
ORDER BY Transaction.date DESC;
```

Profile Page





SQL Statements:

To display name, surname, password, email, description and rating-average

```
SELECT name, email, password, description, user-rating FROM RegisteredUser WHERE
user-id = @user-id;
```

To switch host mode:

```
UPDATE RegisteredUser
SET usage-mode = "host"
WHERE user-id = @user-id;
```

To edit name, surname,password, email or description:

```
UPDATE RegisteredUser
SET name = @name, surname = @surname, password = @password, email = @email ,
description= @description
WHERE user-id = @user-id;
```

To upload/change profile picture

```
UPDATE Images
SET image-id  = (SELECT COALESCE(MAX(image-id), 0) + 1 FROM Images),
        image-type = "profile-photo",
        image-size = @size,
        image-name = @image-name,
        image-binary-path = @file-binary-path
WHERE user-id=@user-id
```

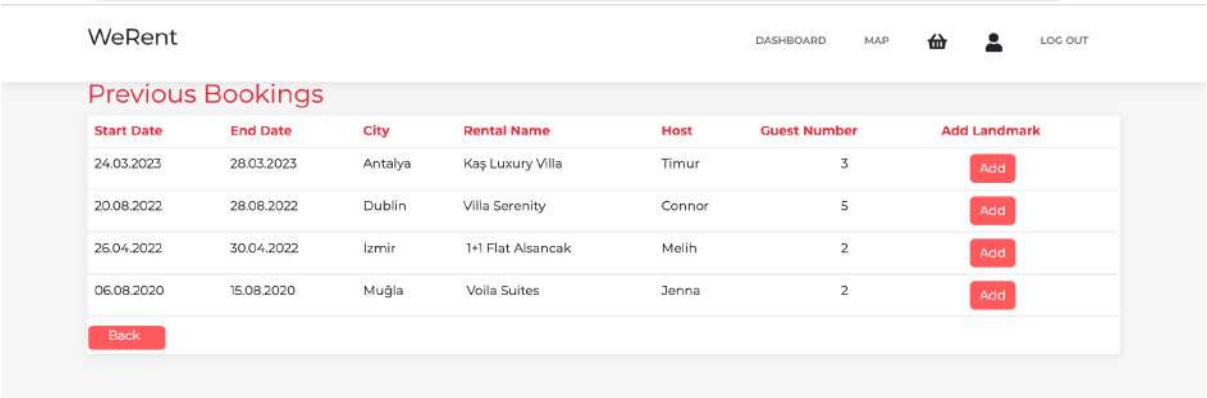
To upload Entitlement of Disaster pdf:

```
INSERT INTO E-Government File(file-name,file-size , file-extension, file-binary-path)
VALUES (@file-name,@file-size , @file-extension, @file-binary-path);
```

To display the reviews that the host has left for user:

```
SELECT RegisteredUser.name, Review.review
FROM Leaves
JOIN Review ON Leaves.review-id= Review.review.id
JOIN RegisteredUser ON Leaves.user-id1=RegisteredUser.user-id
WHERE  Leaves.user-id2 = @user-id
```

Previous Bookings Page

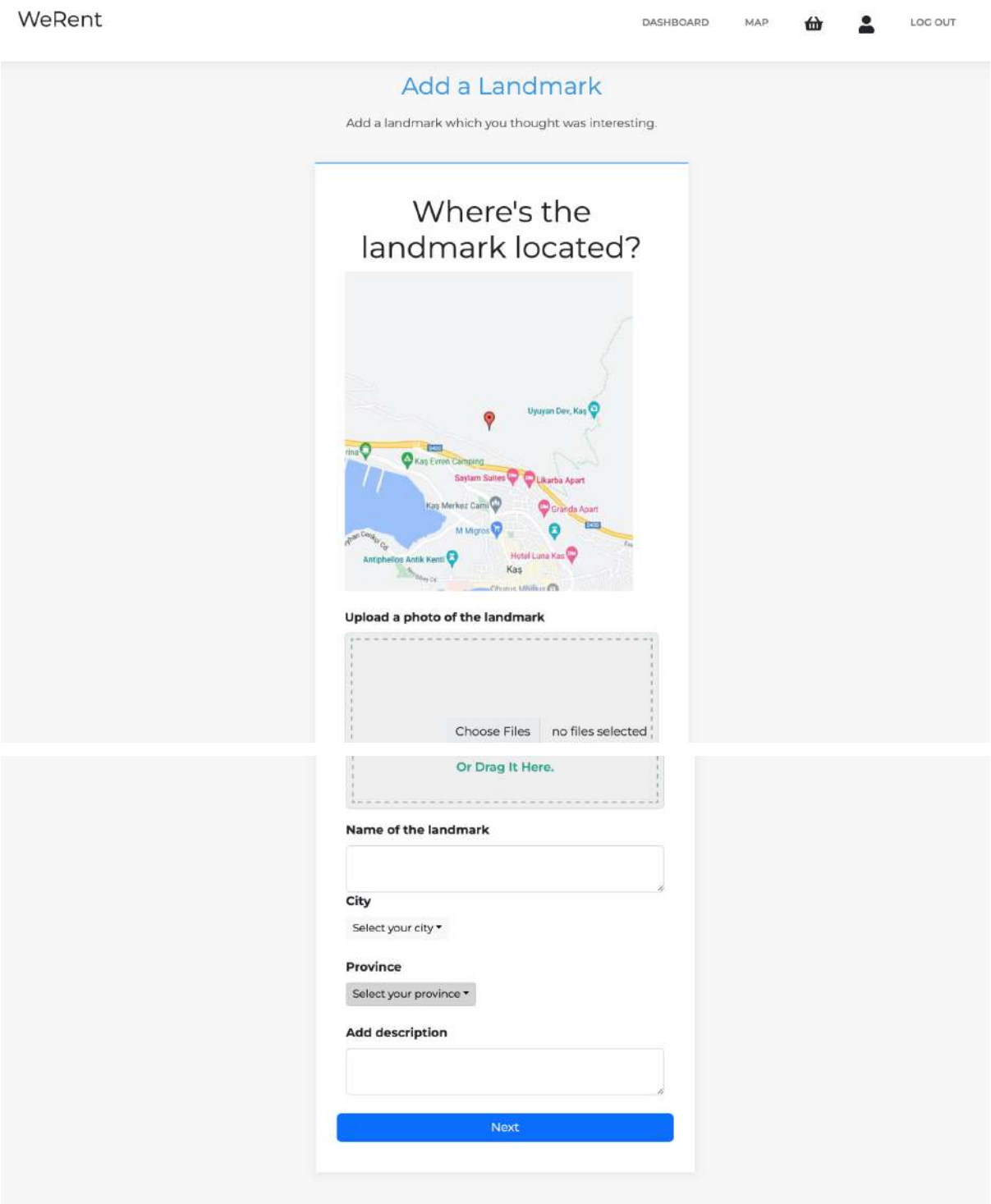


SQL Statements:

To display the previous bookings:

```
SELECT Reservation.reservation-start-date, Reservation.reservation-end-date, Rental.city,
Rental.rental-name, Host.name, Reservation.number-of-guests
FROM Makes
JOIN Rental ON Makes.rental-id = Rental.rental-id
JOIN Host ON Rental.host-id = Host.user-id
JOIN Reservation ON Makes.reservation-id = Reservation.reservation-id
WHERE Makes.user-id = @user-id
ORDER BY Reservation.end-date  DESC;
```

Add a Landmark



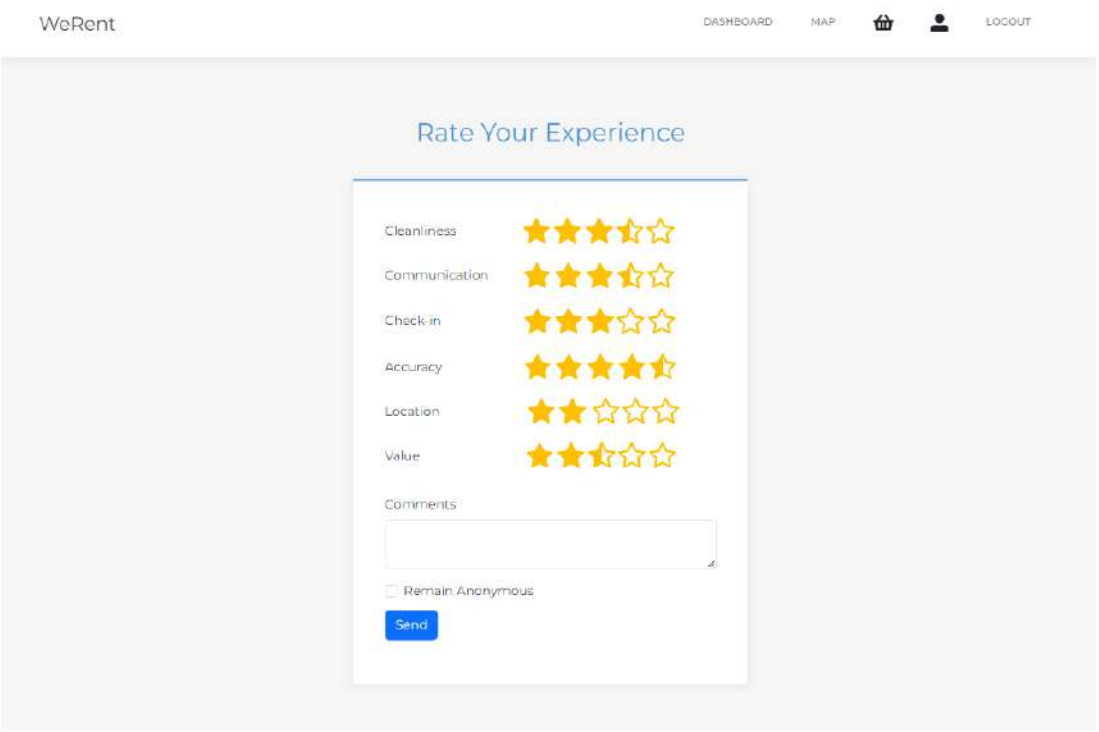
SQL Statements:

To add a new landmark for a previous booking

INSERT INTO Landmark (user-id, landmark-name, description, city, province, latitude, longitude, accepted) VALUES (@user_id, @landmark_name, @description, @city, @province, @latitude, @longitude, NULL);

INSERT INTO Images VALUES (@user_id, “landmark-photo”, @landmark_id, @image_size, @img_binary_path)

Leave Rating Page



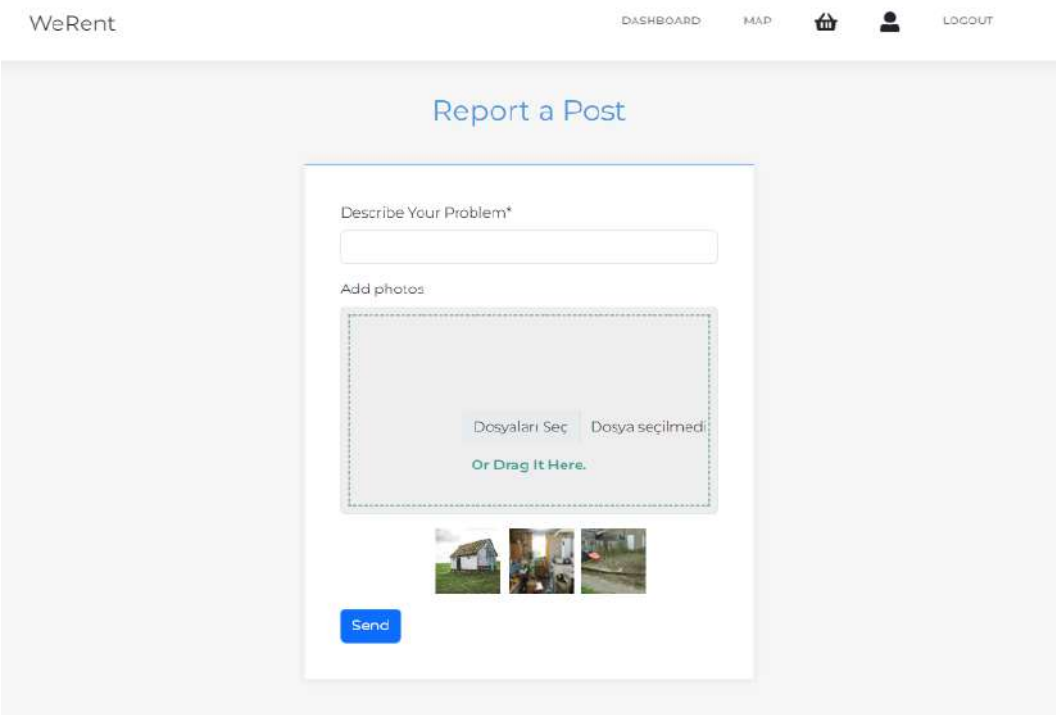
SQL Statements:

To insert into review:

INSERT INTO Review (review, cleanliness-rating, check-in-rating, communication-rating, accuracy-rating, safety-rating, location-rating, value-rating, is-anonymous)

VALUES(@review-text, @cleanliness-rating, @check-in-rating, @communication-rating, @accuracy-rating, @safety-rating, @location-rating, @value-rating, @is-anonymous)

Report a Post Page



SQL Statements:

To insert into reports:

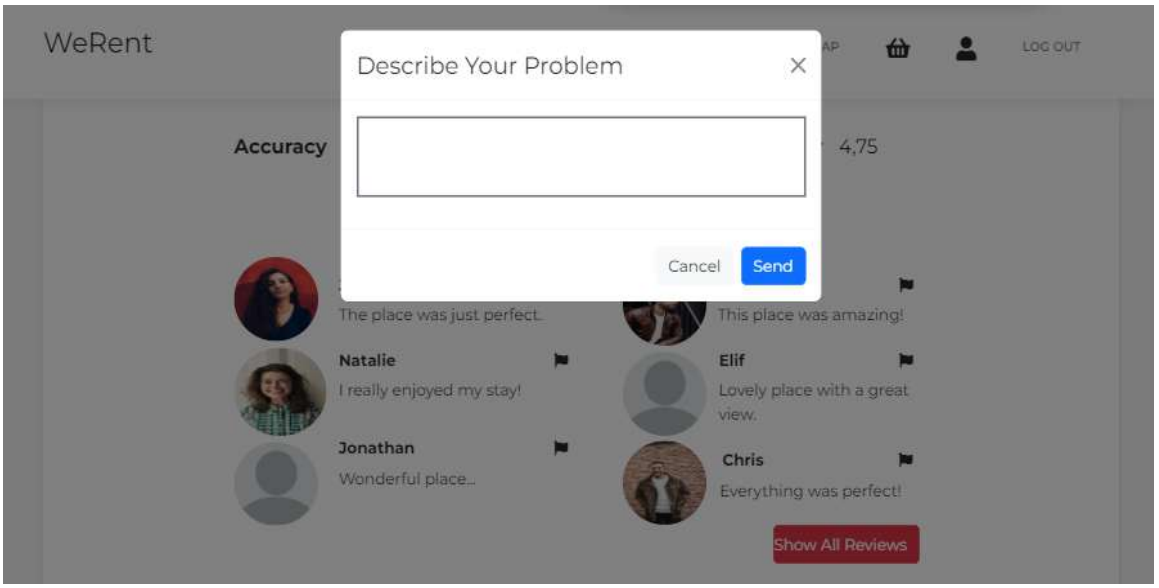
INSERT INTO Reports(user-id, rental-id, report-date, description, is-confirmed, evaluated)

VALUES (@registered-user-id, @clicked-rental-id, GETDATE(), @description, ‘false’, ‘false’);

To insert into images:

INSERT INTO Images (user-id, image-type, image-name, image-size, img-binary-path)
VALUES (@registered-user-id, “report-photo”, @reported-rental-id, @photo-size, @photo-path);

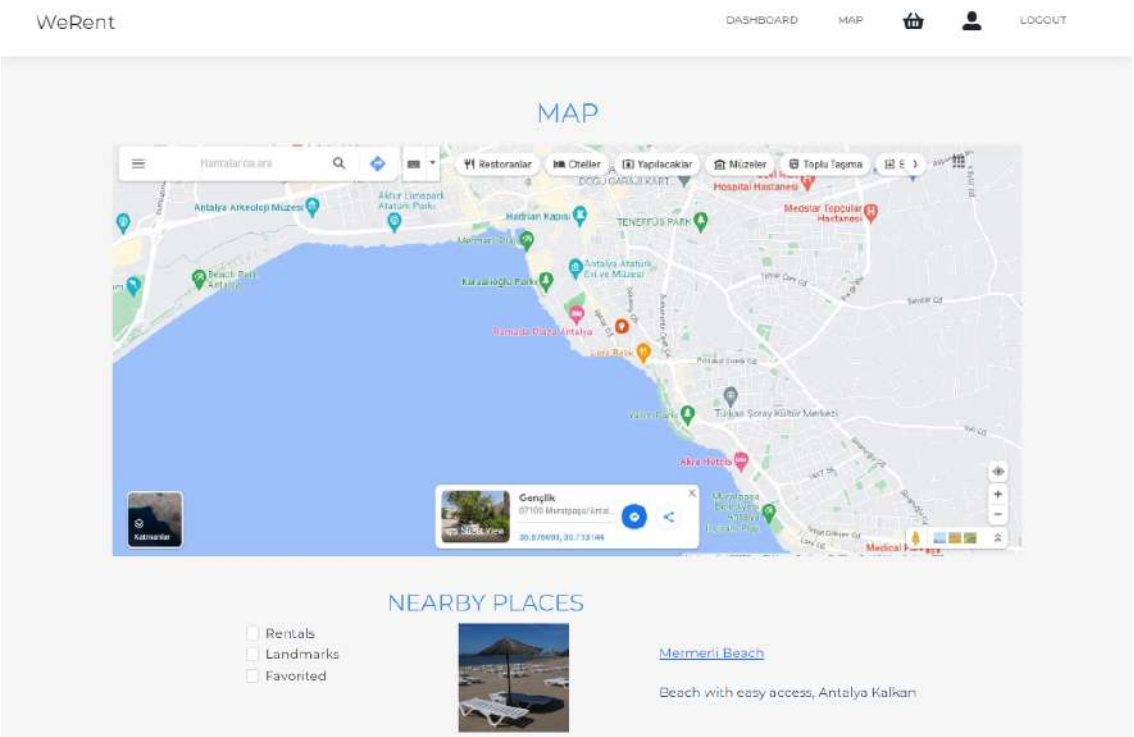
Complain About a User Page

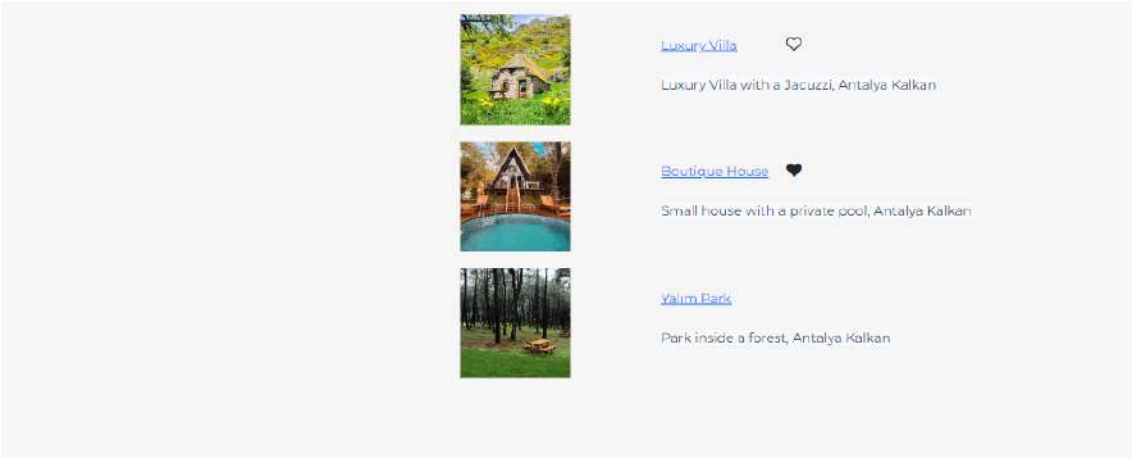


SQL Statements:

To insert into complaints:
INSERT INTO Complaints
VALUES (@registered-user-id, @clicked-user-id, GETDATE(), @description, ‘false’, ‘false’);

Map Page





SQL Statements:

To list nearby places and rentals:

```
SELECT R.rental-name AS place-name, R.description AS description CASE WHEN
wishlists.rental-id IS NULL THEN 0 ELSE 1 END AS is-favorited

FROM Rental R LEFT JOIN (

                    SELECT rental-id

                    FROM wishlists
                    WHERE user-id = @registered-user-id
                    ) AS wishlists ON rental.rental-id = wishlists.rental-id;

WHERE ( ABS( R.latitude - @map-latitude) < 1 AND ABS( R.longitude - @map-longitude)
< 1)

UNION ALL

SELECT L.landmark-name AS place-name, L.description AS description

FROM Landmarks L

WHERE  ( ABS( L.latitude - @map-latitude) < 1 AND ABS( L.longitude - @map-longitude)
< 1);
```

To favorite a rental:

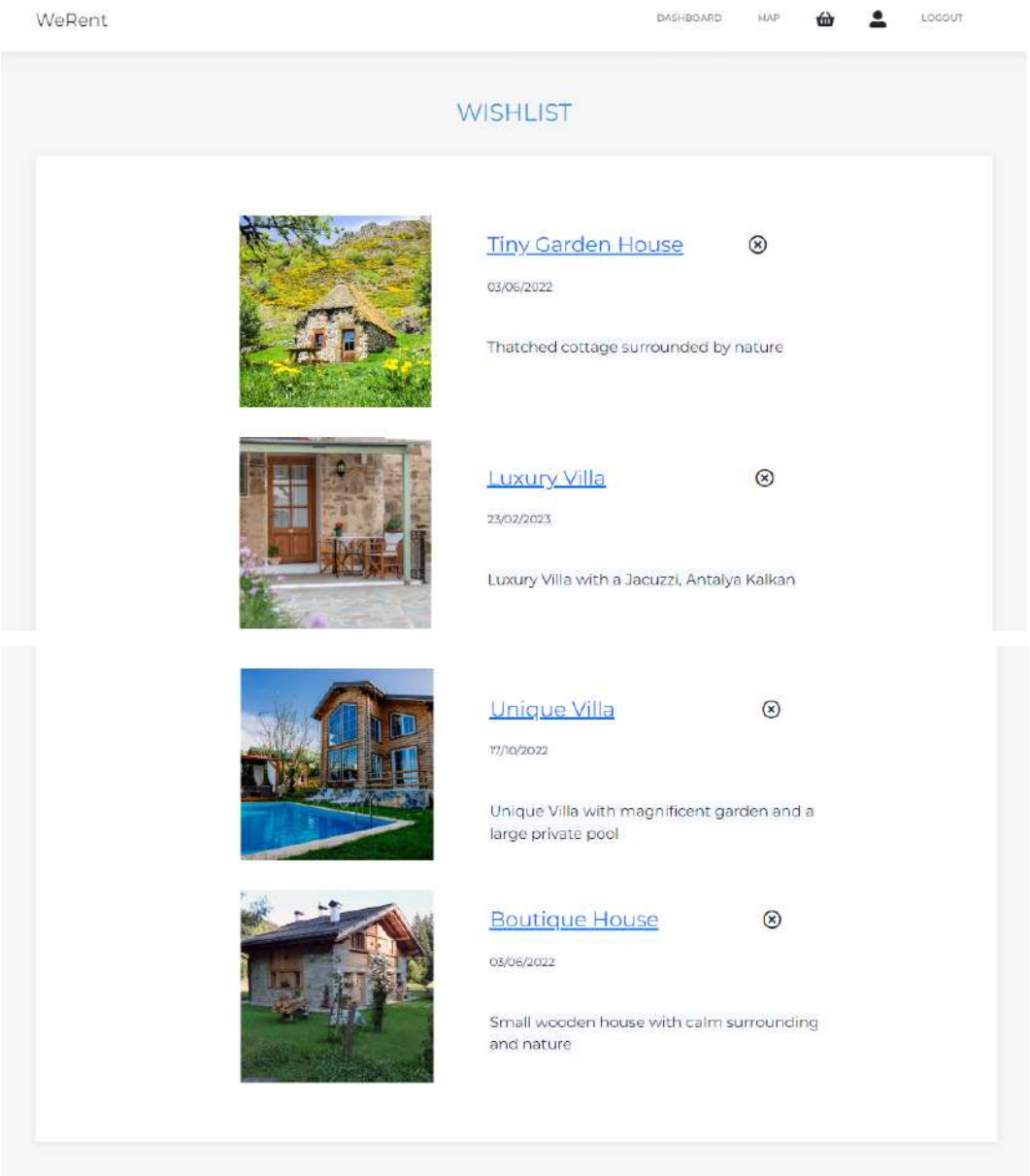
```
INSERT INTO Wishlists
VALUES ( @registered-user-id, @clicked-rental-id, @current-date);
```

To unfavorite a rental:

```
DELETE FROM Wishlists
WHERE user-id = @registered-user-id
```

AND rental-id = @clicked-rental-id;

Wishlist Page



SQL Statements:

To list rentals in the wishlist:

SELECT R.rental-name, R.description, W.date

FROM Rental R, Wishlists W

WHERE R.rental-id = W.rental-id AND W.user-id = @registered-user-id

To remove a rental from wishlist:

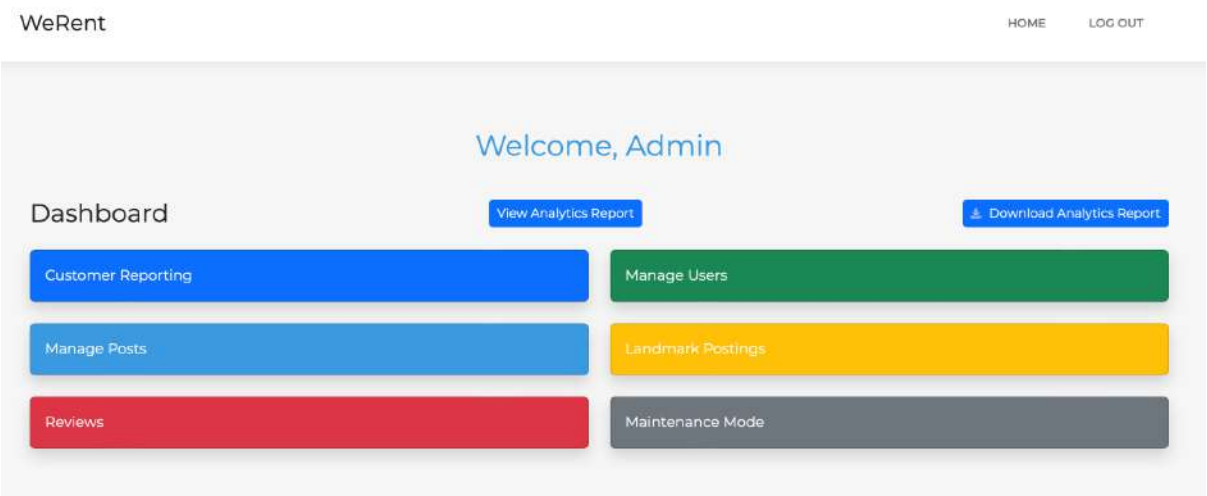
DELETE FROM Wishlists

WHERE user-id = @registered-user-id

AND rental-id = @clicked-rental-id;

2.2.3 Admin Pages

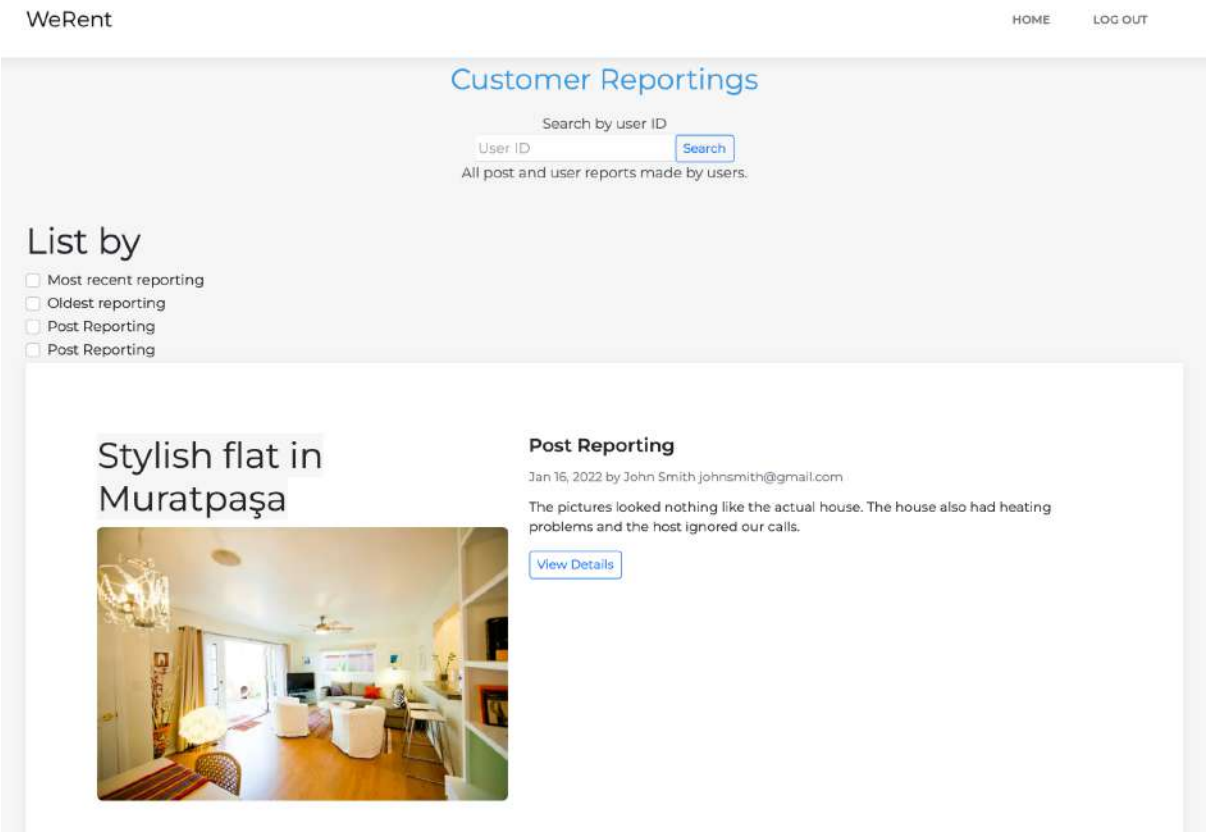
Home Page

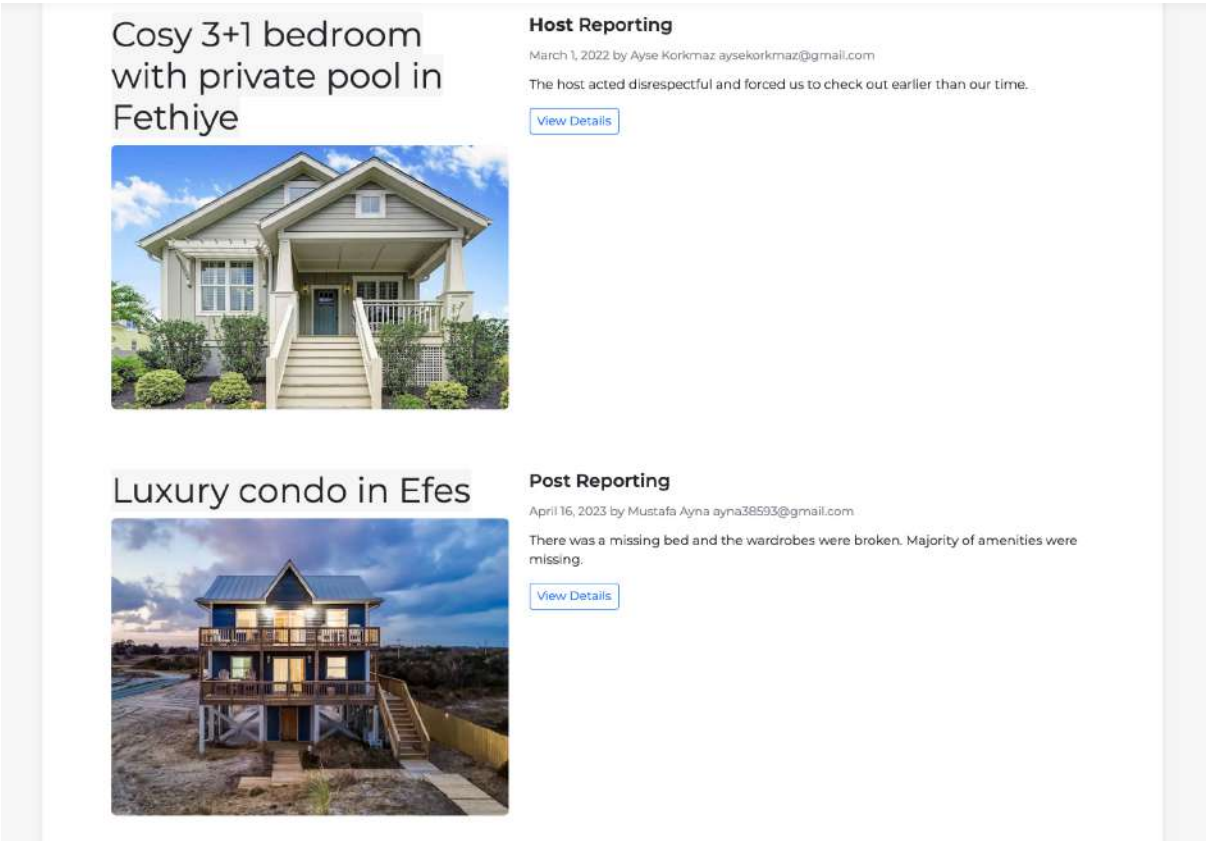


SQL Statements:

To download analysis report when button is clicked:
SELECT *
FROM GeneratedReport
WHERE GETDATE() = (SELECT MAX(date) FROM GeneratedReport)

Customer Reportings





SQL Statements:

To list all reportings:

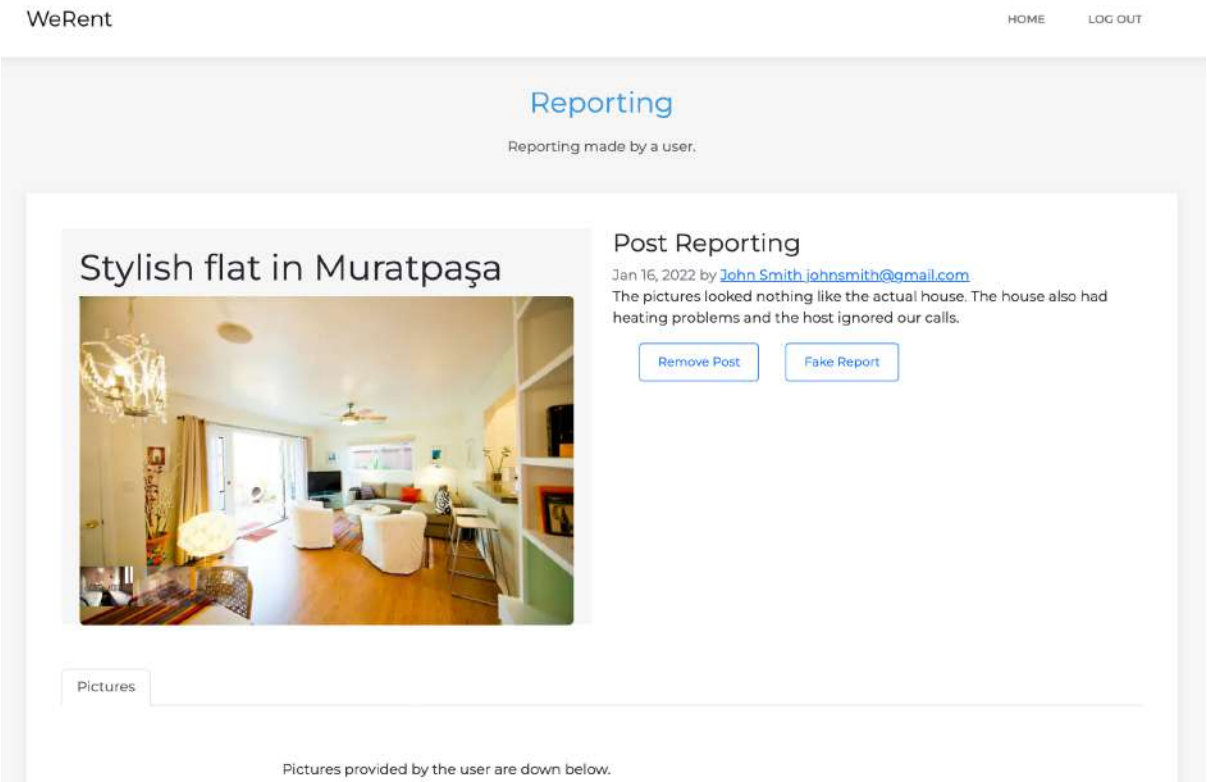
```
SELECT *
FROM Reports R, RegisteredUser U, Rental RE, Complaints C, Images
WHERE R.rental-id = RE.rental-id AND U.user-id = R.user-id AND C.user-id1 = U.user-id
AND I.image-type = 'rental-photo' AND I.image-name = RE.rental-id
```

** The Image entity aspect of the queries are repetitive, thus only one instance was given.*

To list reportings using filters and search engine:

```
SELECT *
FROM Reports R, RegisteredUser U, Rental RE, Complaints C
WHERE R.user-id = U.user-id AND RE.rental-id =R.rental-id AND user-id LIKE
'%@title%' AND C.user-id1 = U.user-id AND ( U.user-type = @host_checked OR
U.user-type = @customer_checked) AND ( R.evaluated = @evaluated OR R.evaluated =
@unevaluated) AND ( C.evaluated = @evaluated OR C.evaluated = @unevaluated)
ORDER BY
CASE WHEN @recent_to_latest = 1 THEN date END DESC
CASE WHEN @latest_to_recent = 1 THEN date END ASC
```

View Individual Reports



Show the previous page

Pictures provided by the user are down below.

SQL Statements:

To list the details of a post report:
SELECT *
FROM Reports R, Rental RE, RegisteredUser U, Images I
WHERE R.rental-id = RE.rental-id AND U.user-id = R.user-id AND U.user-id = @user_id
AND I.image-type = 'report-photo' AND I.image-name = @user_id

** The Image entity aspect of the queries are repetitive, thus only one instance was given.*

To list the details of a user report:
SELECT *
FROM RegisteredUser R, Complaints C
WHERE C.user-id1 = R.user-id AND R.user-id = @user_id

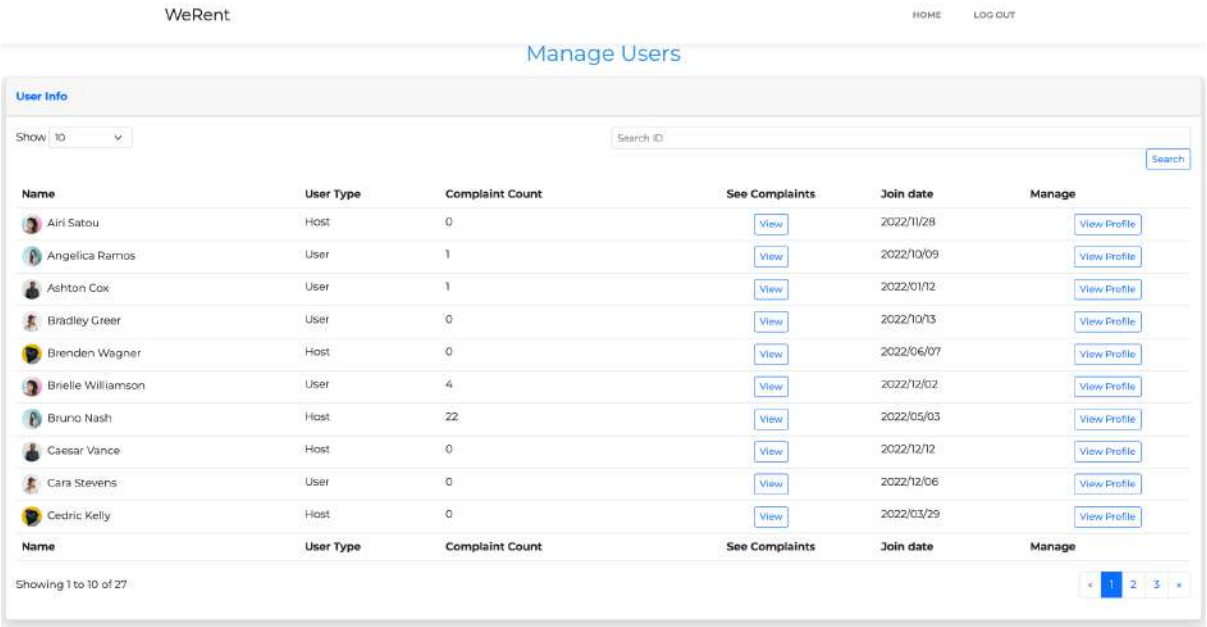
To delete/block reported user:
DELETE FROM User
WHERE user-id = @user_id

To delete/block reported post:
DELETE FROM Rental
WHERE rental-id = @rental_id

To report a fake post:
UPDATE Reports
SET is-confirmed = FALSE AND evaluated = TRUE
WHERE rental-id = @rental_id

To report a fake user complaint:
UPDATE Complaints
SET is-confirmed = FALSE AND evaluated = TRUE
WHERE user-id = @user_id

Manage Users Page



SQL Statements:

To list all users:

SELECT name, surname, user-type, count(*) as complaint-cnt, join-date, img-binary-path
FROM RegisteredUser U, Complaints C, Image I
WHERE U.user-id = C.user-id2 AND I.image-type = “profile-photo” AND I.image-name = U.user-id
GROUP BY user-id;

** The Image entity aspect of the queries are repetitive, thus only one instance was given.*

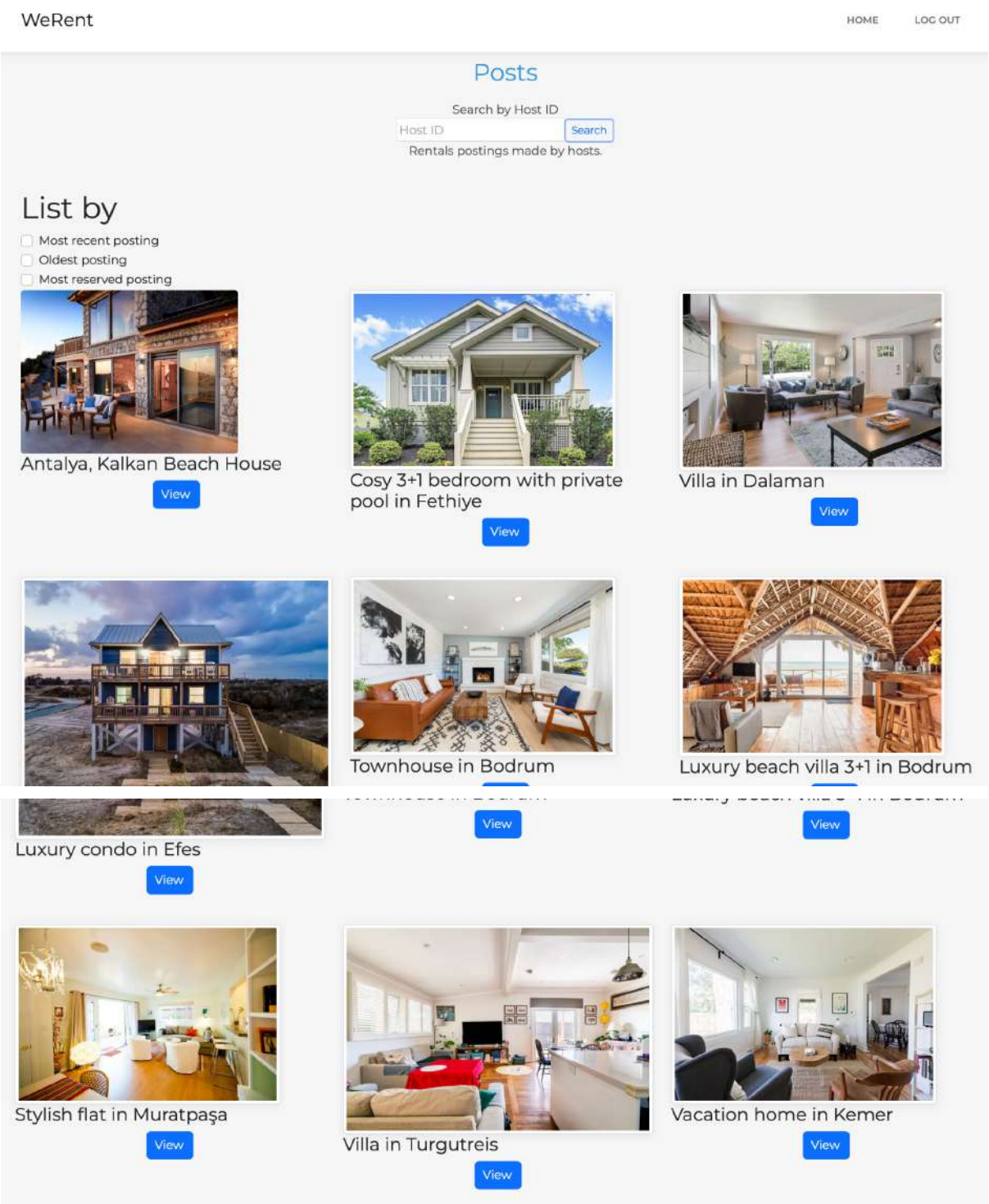
To view a user:

SELECT *
FROM RegisteredUser
WHERE user-id = @user_id;

To see user complaints:

SELECT *
FROM RegisteredUser U, Complaints C
WHERE U.user-id = C.user-id2;

Manage Post Page



SQL Statements:

To list all posts:

SELECT rental-name, img-binary-path
FROM Posts P, Rental R, Images I

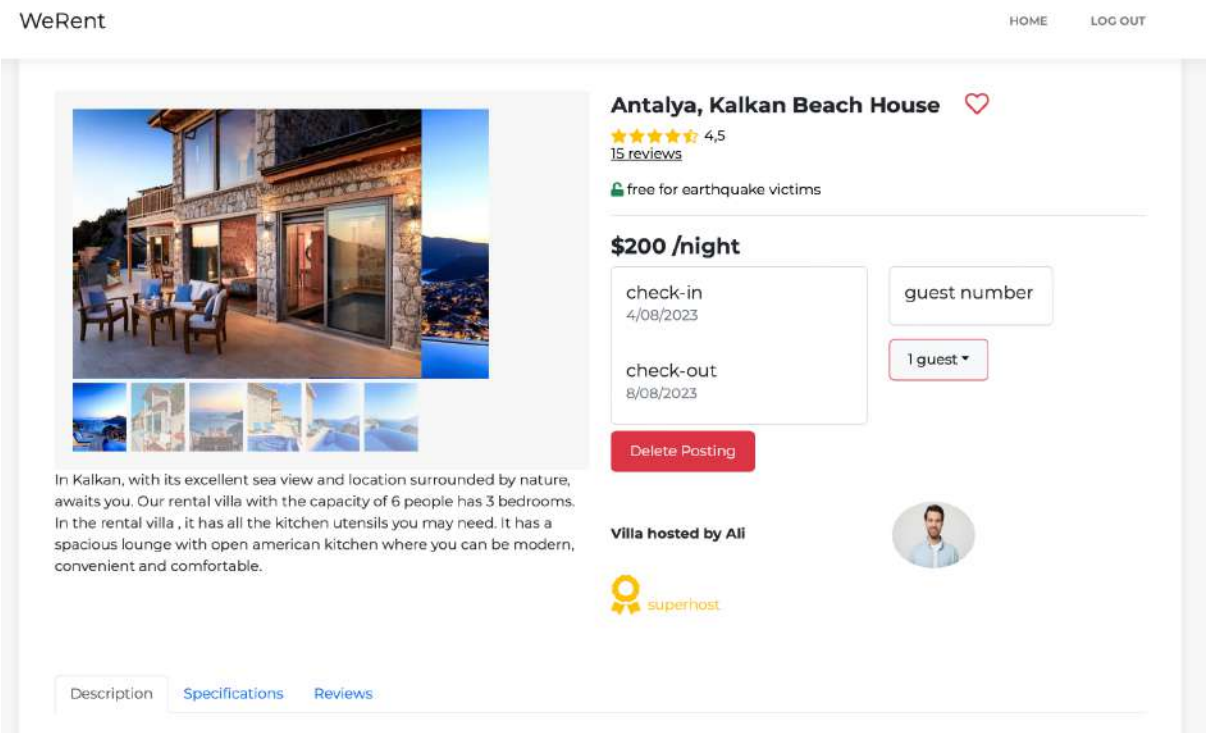
WHERE user-id = @user_id AND P.rental-id = R.rental-id AND I.image-type = ‘rental-photo’ AND I.image-name = R.rental-id

** The Image entity aspect of the queries are repetitive, thus only one instance was given.*

To list all posts using filters and search engine:

```
SELECT *
FROM Posts P, Rental R, RegisteredUser RU
                    WHERE RU.user-id LIKE ‘%@title%’ AND R.rental-id =
                    P.rental-id AND RU.user-id = P.user-id AND R.rental-id =
                    HAVING COUNT(*) = (SELECT MAX(reservation-cnt)
                    FROM (SELECT RES.rental-id, COUNT(*) as reservation-cnt
                          FROM  Reservation RES,
                          GROUP BY rental-id))
ORDER BY
CASE WHEN @recent_to_latest = 1 THEN date END DESC
CASE WHEN @latest_to_recent = 1 THEN date END ASC
```

Manage Single Post Page



SQL Statements:

To create view of the user:

```
CREATE VIEW individual_post
SELECT *
```

FROM Rental R, RegisteredUser RU, Images I
WHERE R.host-id = RU.user-id AND I.image-type = ‘rental-photo’ AND I.image-name = R.rental-id

** The Image entity aspect of the queries are repetitive, thus only one instance was given.*

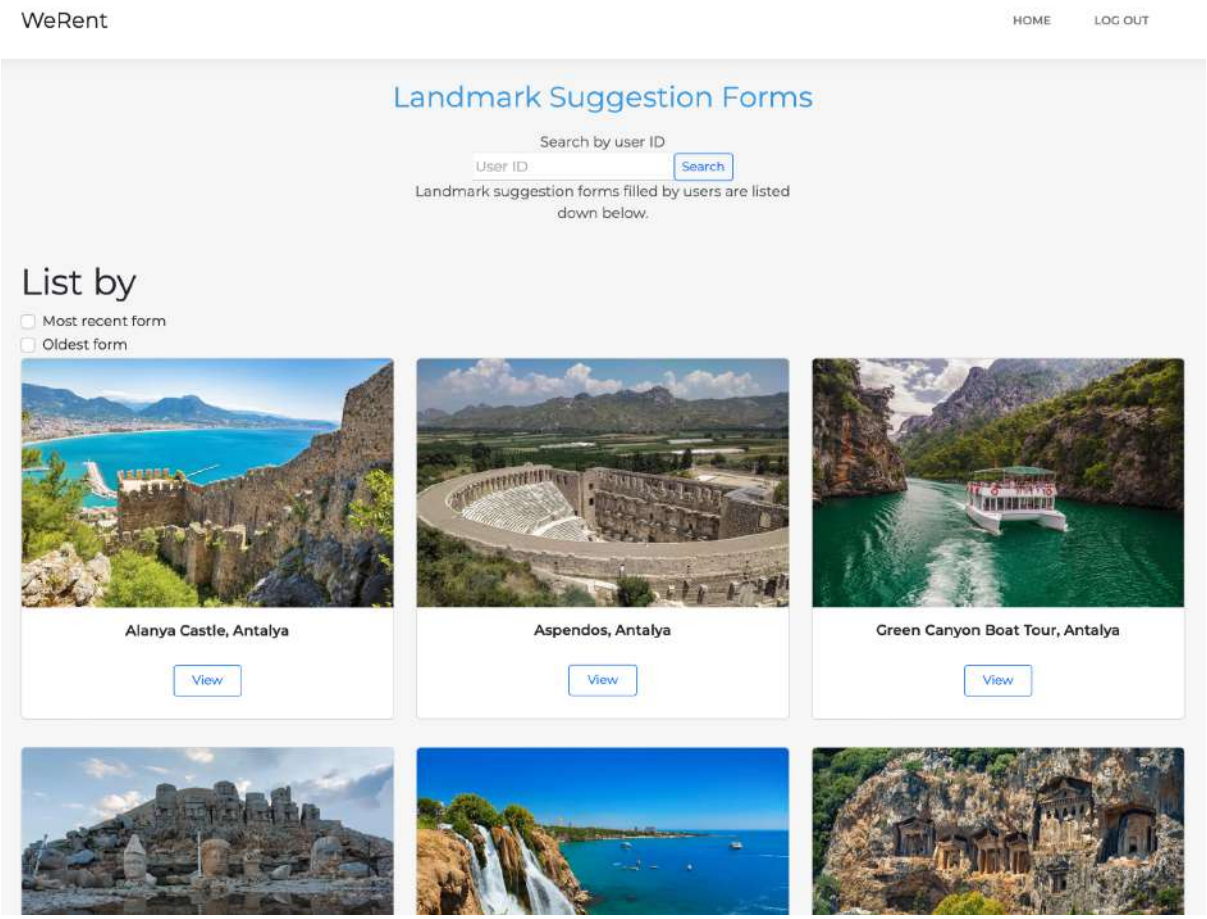
To display a user:

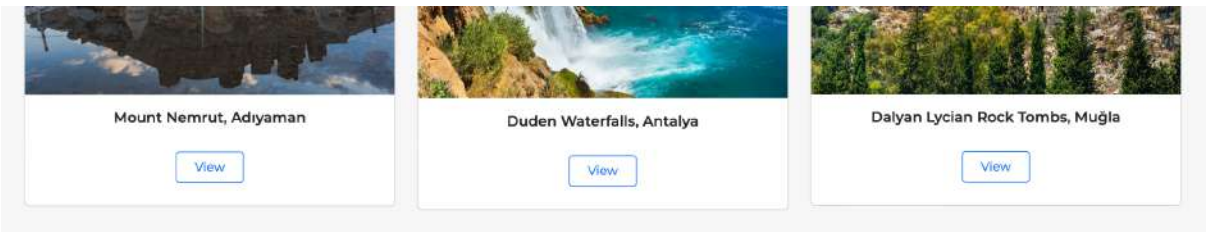
SELECT *
FROM individual_post

To delete a post:

DELETE FROM individual_post
WHERE rental-id = @rental_id

Landmark Suggestion Forms





SQL Statements:

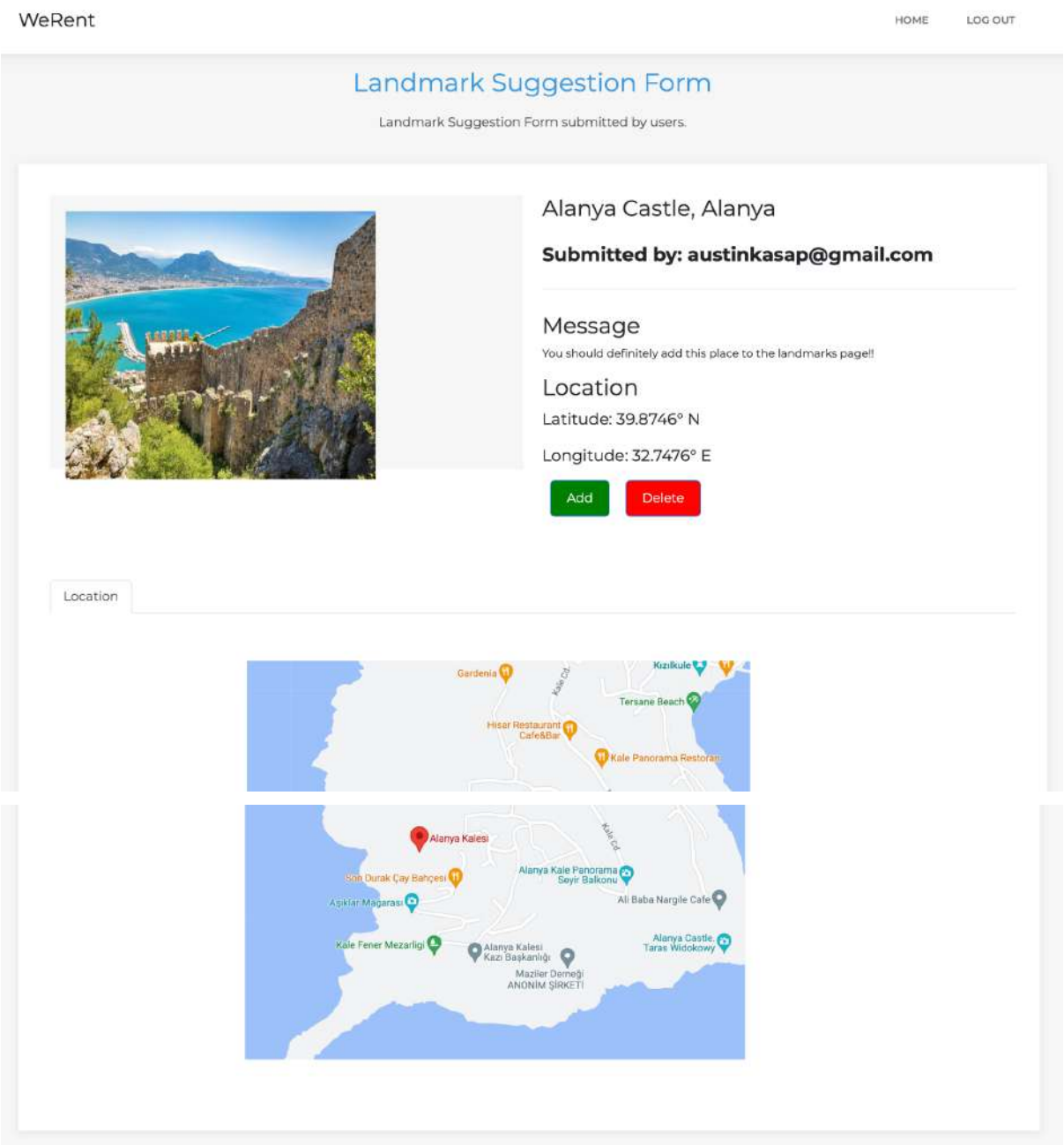
To display all landmark forms:

SELECT landmark-name,
FROM Creates C, Landmark L, Images I
WHERE L.landmark-id = C.landmark-id AND I.image-type = ‘landmark-photo’ AND I.image-name = L.landmark-id

** The Image entity aspect of the queries are repetitive, thus only one instance was given.*

To display all landmark forms based on search and filters:
SELECT *
FROM Creates C, Fills F, Landmark L, Survey S
WHERE S.user-id = C.survey-id AND C.survey-id = F.survey-id AND L.landmark-id = C.landmark-id AND S.user-id LIKE ‘%@title%’
ORDER BY
CASE WHEN @recent_to_latest = 1 THEN date END DESC
CASE WHEN @latest_to_recent = 1 THEN date END ASC

Landmark Suggestion Single Form



SQL Statements:

To display details of a landmark and create view:

CREATE VIEW single_landmark
SELECT landmark-id, landmark-name, description, email, latitude, longitude, city, province
FROM Landmark L, Creates C, Images I
WHERE L.landmark-id = C.landmark-id AND L.landmark-id = @landmark-id AND
I.image-type = ‘landmark-photo’ AND I.image-name = @landmark-id

** The Image entity aspect of the queries are repetitive, thus only one instance was given.*

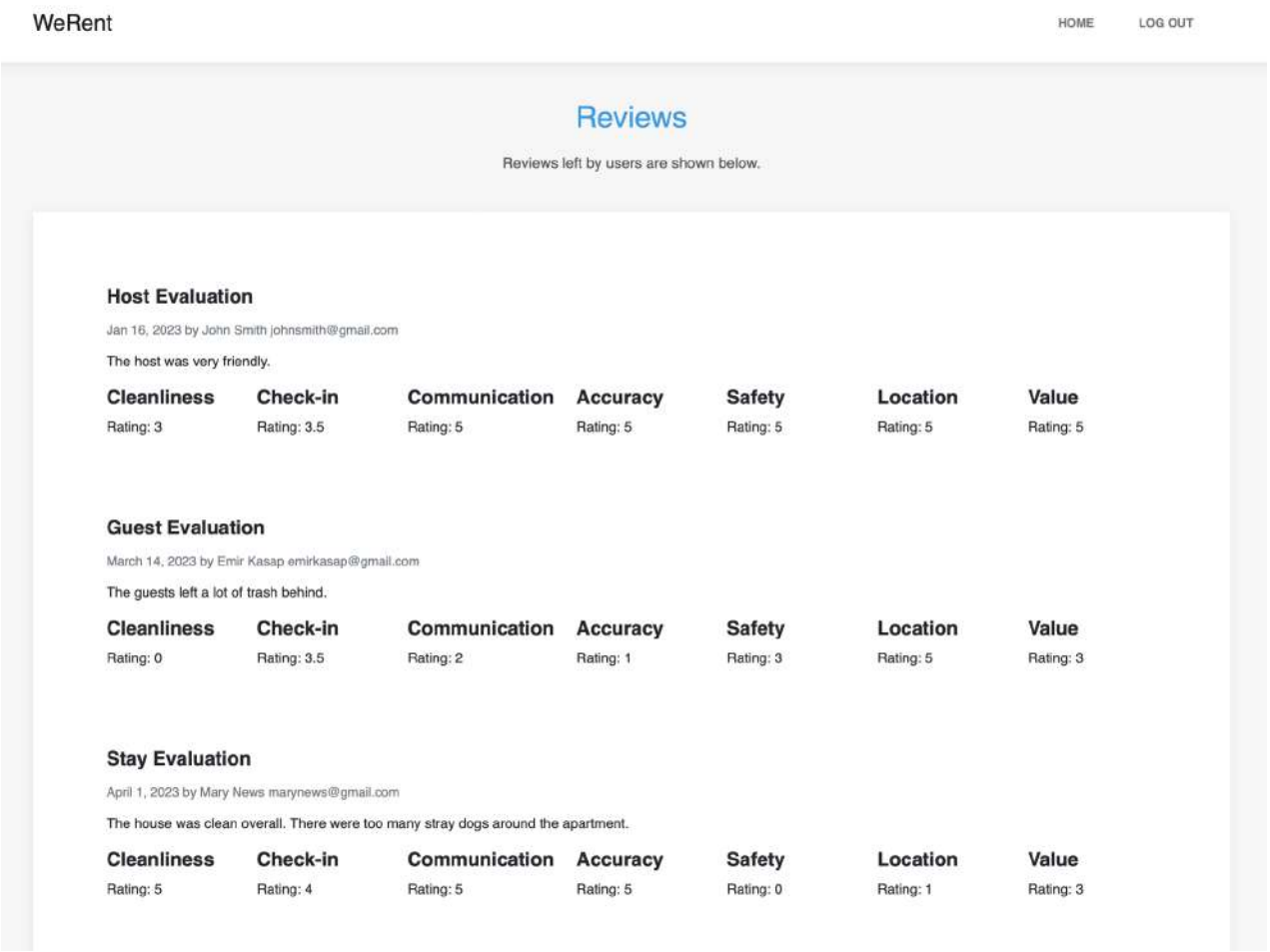
To add a landmark to the official landmarks:

UPDATE Landmark
SET accepted = TRUE
WHERE landmark-id = @landmark-id

To delete a landmark from list of suggestion forms:

DELETE FROM Landmark
WHERE landmark-id = @landmark-id

Reviews



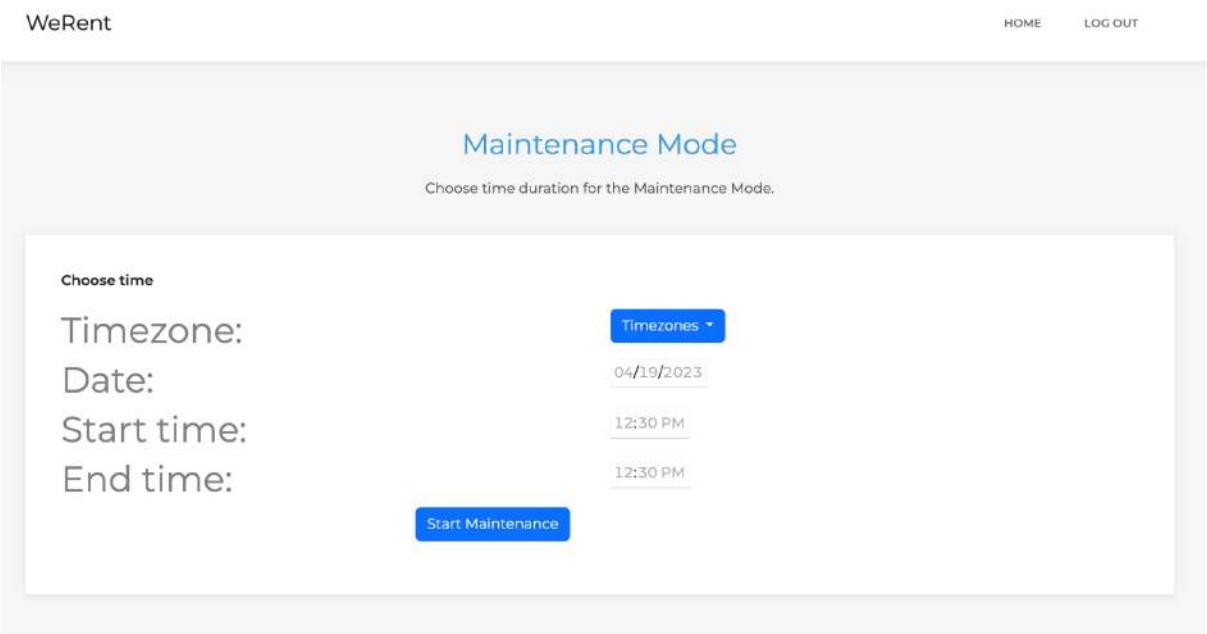
SQL Statements:

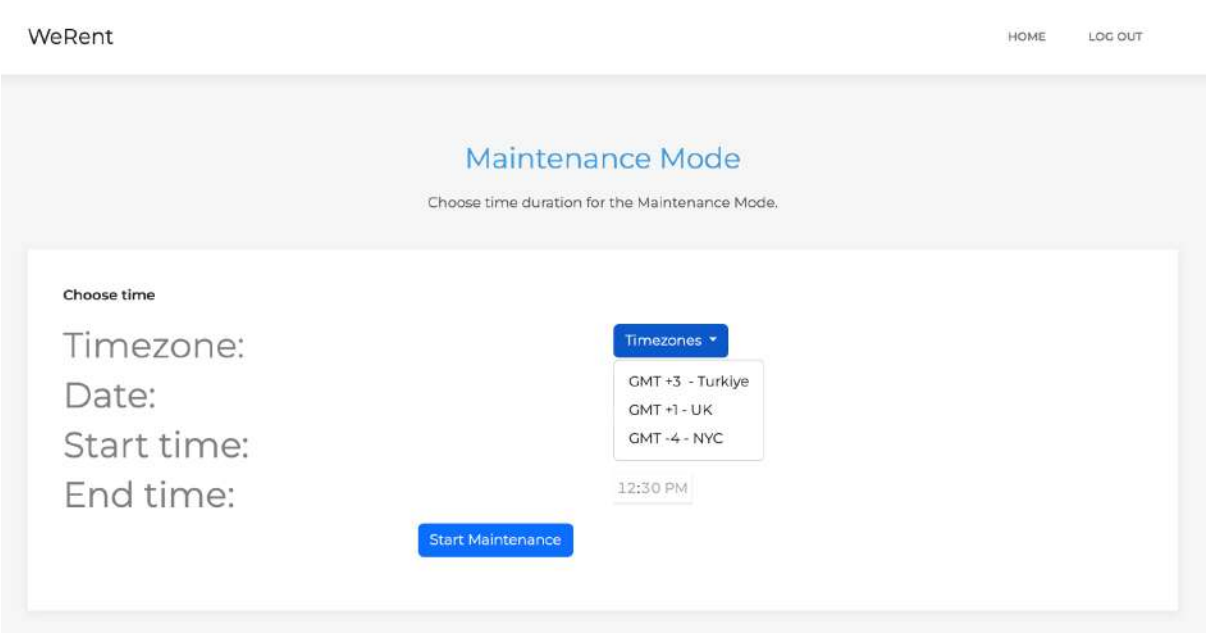
To list evaluations left by guests and hosts

SELECT name, date, e-mail, communication-rating, cleanliness-rating, check-in-rating, accuracy-rating, location-rating, value-rating, safety-rating
FROM Review R, Leaves L, RegisteredUser RU
WHERE L.user-id = RU.user-id AND L.review-id = R.review-id

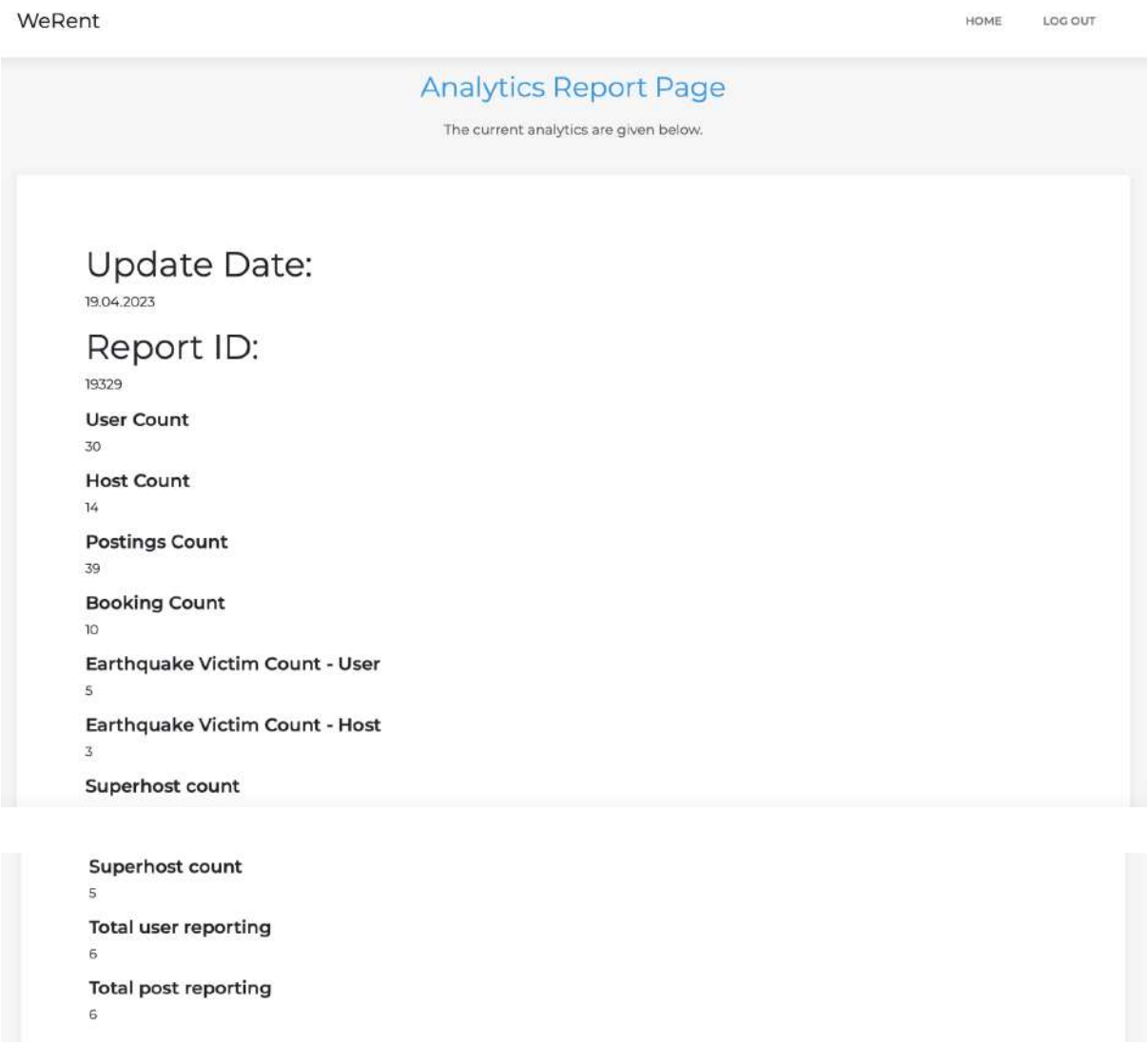
** The Image entity aspect of the queries are repetitive, thus only one instance was given.*

Maintenance Mode





Analytics Page

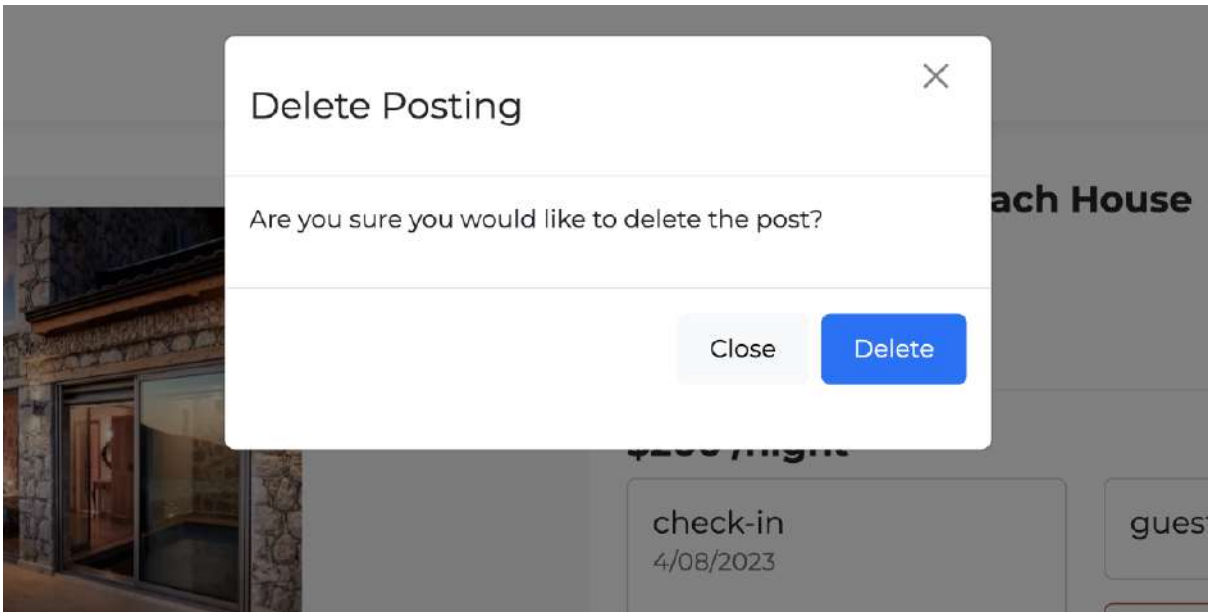


SQL Statements:

To list the analytics:

```
SELECT *
FROM GeneratedReports
WHERE date = (SELECT MAX(date) FROM GeneratedReports);
```

“Are you sure?” pop-up example:



Every button that requires a second-check has a pop-up like the example above.

3.Triggers

UPDATE ANALYTICS REPORT PAGE AFTER CHANGES MADE:

NEW USER ADDED AND DELETED:

```
CREATE TRIGGER analytics_page_for_user
AFTER INSERT ON RegisteredUser
REFERENCING NEW ROW AS nrow
BEGIN atomic
        UPDATE GeneratedReport
        SET user-cnt = user-cnt + 1
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;

CREATE TRIGGER analytics_page_for_user1
AFTER DELETE ON RegisteredUser
REFERENCING NEW ROW AS nrow
BEGIN atomic
        UPDATE GeneratedReport
        SET user-cnt = user-cnt - 1
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;
```

```
CREATE TRIGGER analytics_page_for_host
AFTER INSERT ON RegisteredUser
REFERENCING NEW ROW AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET host-cnt = host-cnt + 1
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;

CREATE TRIGGER analytics_page_for_host1
AFTER DELETE ON RegisteredUser
REFERENCING OLD ROW AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET host-cnt = host-cnt - 1
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;
```

NEW POST ADDED AND DELETED:

```
CREATE TRIGGER analytics_page_for_post
AFTER INSERT ON Posts
REFERENCING NEW ROWS AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET postings-cnt = ( SELECT count(*)
                             FROM Posts)
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;

CREATE TRIGGER analytics_page_for_post1
AFTER DELETE ON Posts
REFERENCING NEW ROWS AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET postings-cnt = ( SELECT count(*)
                             FROM Posts)
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;
```

NEW BOOKING REGISTERED OR CANCELED:

```
CREATE TRIGGER analytics_page_for_booking
AFTER INSERT ON Reservation
REFERENCING NEW ROW AS nrow
```

```
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET booking-cnt = ( SELECT count(*)
                                FROM Reservation)
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;

CREATE TRIGGER analytics_page_for_booking1
AFTER DELETE ON Reservation
REFERENCING NEW ROW AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET booking-cnt = ( SELECT count(*)
                                FROM Reservation)
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;
```

NEW EARTHQUAKE VICTIM REGISTERED-CUSTOMER:

```
CREATE TRIGGER analytics_page_for_earthquake_user
AFTER UPDATE ON RegisteredUser
REFERENCING NEW ROW AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET booking-cnt = ( SELECT count(*)
                                FROM Customer
                                WHERE is-earthquake-victim = TRUE)
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;
```

NEW EARTHQUAKE VICTIM REGISTERED-HOST:

```
CREATE TRIGGER analytics_page_for_earthquake_host
AFTER UPDATE ON  RegisteredUser
REFERENCING NEW ROWS AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET host-earthquake-victim-cnt = ( SELECT count(*)
                                FROM Host WHERE is-earthquake-victim = TRUE)
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;
```

UPDATE REPORTS AFTER A USER IS DELETED:

```
CREATE TRIGGER update_report
AFTER DELETE OF User
REFERENCING OLD  ROW AS orow
BEGIN ATOMIC
UPDATE Complaints
        SET is-confirmed = TRUE AND evaluated = TRUE
        WHERE user-id = orow.user-id
END;
```

UPDATE REPORTS AFTER A RENTAL POST IS DELETED:

```
CREATE TRIGGER update_report
AFTER DELETE of Rental
REFERENCING old row as orow
BEGIN ATOMIC
        UPDATE Reports
        SET is-confirmed = TRUE AND evaluated = TRUE
        WHERE rental-id = orow.rental-id
END;
```

UPDATE REPORTS AFTER A HOST IS UPDATED:

```
CREATE TRIGGER update_host_cnt
AFTER UPDATE OF Host
REFERENCING NEW ROW AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET superhost-cnt = ( SELECT count(*) FROM Host WHERE is-superhost = TRUE)
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;
```

UPDATE REPORTS AFTER A NEW COMPLAINT IS ADDED:

```
CREATE TRIGGER analytics_page_user_reporting
AFTER INSERT ON Complaints
REFERENCING NEW ROW AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET user-reporting-cnt = ( SELECT count(*)
                                   FROM Complaints)
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;
```

UPDATE REPORTS AFTER A NEW REPORT IS ADDED:

```
CREATE TRIGGER analytics_page_post_reporting
AFTER INSERT ON Reports
REFERENCING  NEW ROWS AS nrow
BEGIN ATOMIC
        UPDATE GeneratedReport
        SET post-reporting-cnt = ( SELECT count(*)
                                   FROM Reports)
        WHERE (SELECT MAX(date) FROM GeneratedReports) = GETDATE()
END;
```

UPDATE AVERAGE RATING OF HOST AFTER A NEW RATING IS ADDED:

```
CREATE  TRIGGER new_rating_host
AFTER INSERT ON Leaves
REFERENCING NEW ROW AS nrow
BEGIN ATOMIC
        UPDATE RegisteredUser
        SET RegisteredUser.user-rating = (SELECT AVG(value-rating)
                        FROM Review R, Leaves L
                        WHERE  L.review-id= R.review-id AND L.user-id2 = nrow.user-id2 )
                        WHERE  RegisteredUser.user-id2= nrow.user-id2
END;
```

UPDATE RESERVATION AFTER TRANSACTION STATUS IS SUCCESSFUL:

```
CREATE TRIGGER update_reservation_paid_status
AFTER UPDATE ON Transaction
REFERENCING NEW ROW as nrow
FOR EACH ROW
BEGIN ATOMIC
        IF nrow.status = 'successful' THEN
        UPDATE Reservation
        SET is-paid-for = 'successful'
        WHERE reservation-id = nrow.reservation-id;
        END IF;
END;
```

UPDATE AVERAGE RATING OF CUSTOMER AFTER A NEW RATING IS ADDED:

```
CREATE  TRIGGER new_rating_customer
AFTER INSERT ON Leaves
REFERENCING NEW ROW AS nrow
```

```
BEGIN ATOMIC
        UPDATE RegisteredUser
        SET RegisteredUser.user-rating = (SELECT AVG(general-rating)
                FROM Leaves L, Review R
                WHERE  L.review-id= R.review-id AND L.user-id2 = nrow.user-id2 )
        WHERE  RegisteredUser.user-id2= nrow.user-id2
END;
```

UPDATE AVERAGE RATING OF CUSTOMER AFTER A RATING IS DELETED:

```
CREATE  TRIGGER delete_rating_customer
AFTER DELETE OF Leaves
REFERENCING OLD ROW AS orow
BEGIN ATOMIC
        UPDATE RegisteredUser
        SET RegisteredUser.user-rating = (SELECT AVG(general-rating)
                FROM Leaves L, Review R
                WHERE L.review-id = R.review-id AND L.user-id2= orow.user-id2)
        WHERE RegisteredUser.user-id2 = orow.user-id2
END;
```

UPDATE AVERAGE RATING OF A RENTAL AFTER A RATING IS ADDED:

```
CREATE  TRIGGER new_rating_rental
AFTER INSERT ON Review
REFERENCING NEW ROW AS nrow
BEGIN ATOMIC
        UPDATE Rental
        SET Rental.rating = (SELECT ((R.cleanliness-rating+R.check-in-rating +
R.communication-rating + R.accuracy-rating + R.safety-rating+
R.location-rating+R.value-rating)/7) AS general-average-rating
FROM Review R, Leaves L, Rental REN, Reservation RES
        WHERE  L.review-id = R.review-id AND L.reservation-id = RES.reservation-id
        AND RES.rental-id = REN.rental-id)
END;
```

UPDATE AVERAGE RATING OF A RENTAL AFTER A RATING IS DELETED:

```
CREATE  TRIGGER delete_rating_rental
AFTER DELETE OF Review
REFERENCING OLD ROW AS orow
BEGIN ATOMIC
```

```
        UPDATE Rental
        SET Rental.rating = (SELECT ((R.cleanliness-rating+R.check-in-rating +
        R.communication-rating + R.accuracy-rating + R.safety-rating+
        R.location-rating+R.value-rating)/7) AS general-average-rating
        FROM Review R, Leaves L, Rental REN, Reservation RES
        WHERE  L.review-id = R.review-id AND L.reservation-id = RES.reservation-id
        AND RES.rental-id = REN.rental-id)
END;
```

UPDATE THE LEAVES TABLE WHEN A REVIEW IS ADDED:

```
CREATE TRIGGER add_leaves_table
AFTER INSERT ON Review
REFERENCING NEW ROW AS nrow
BEGIN ATOMIC
        WITH temp AS (
         SELECT R.reservation-id, L.user-id1, L.user-id2
         FROM Leaves L
         JOIN Reservation R ON L.reservation-id = R.reservation-id
         JOIN RegisteredUser U1 ON L.user-id1 = U1.user-id
         JOIN RegisteredUser U2 ON L.user-id2 = U2.user-id
         WHERE U1.user-id <> U2.user-id AND R.reservation-id = nrow.reservation-id )

        INSERT INTO Leaves(reservation-id, user-id1, user-id2, review-id)
        SELECT temp.reservation-id, temp.user-id1, temp.user-id2, nrow.review-id
        FROM temp;
END;
```

UPDATE THE LEAVES TABLE WHEN A REVIEW IS DELETED:

```
CREATE TRIGGER trigger_delete_review
ON Review
AFTER DELETE
REFERENCING deleted_row AS deleted
BEGIN ATOMIC
        DELETE FROM Leaves L
        WHERE L.review-id IN (SELECT deleted.review-id FROM deleted);
END;
```

ADD RENTAL TO SHOPPING CART WHEN A RESERVATION IS MADE:

```
CREATE  TRIGGER add_shopping_cart
AFTER INSERT ON Reservation
REFERENCING NEW ROW AS  nrow
BEGIN ATOMIC
```

```
        WITH temp AS (
        SELECT M.user-id, M.rental-id
        FROM Makes M, Reservation R
        WHERE M.reservation-id = R.reservation-id)

        INSERT INTO ShoppingCart (user-id, rental-id,reservation-id) values
        (temp.user-id,temp.rental-id,Reservation.reservation-id);
END;
```

4.Implementation Plan

The technologies that will be used in the front-end includes React.js, Bootstrap and Openlayers API. We plan on using these technologies as our team members are already knowledgeable in these frameworks and they are widely used in modern websites. Openlayers API is opted to use instead of Google Maps API as it is free for the map component of the project. SpringBoot framework will be used as the backend and PostgreSQL will be used for the database. We chose PostgreSQL due to it supporting modern DBMS functionalities.