# CS 353 - Database Systems

## Project Design System

## Room / Flat Rental Application


**Group 18**

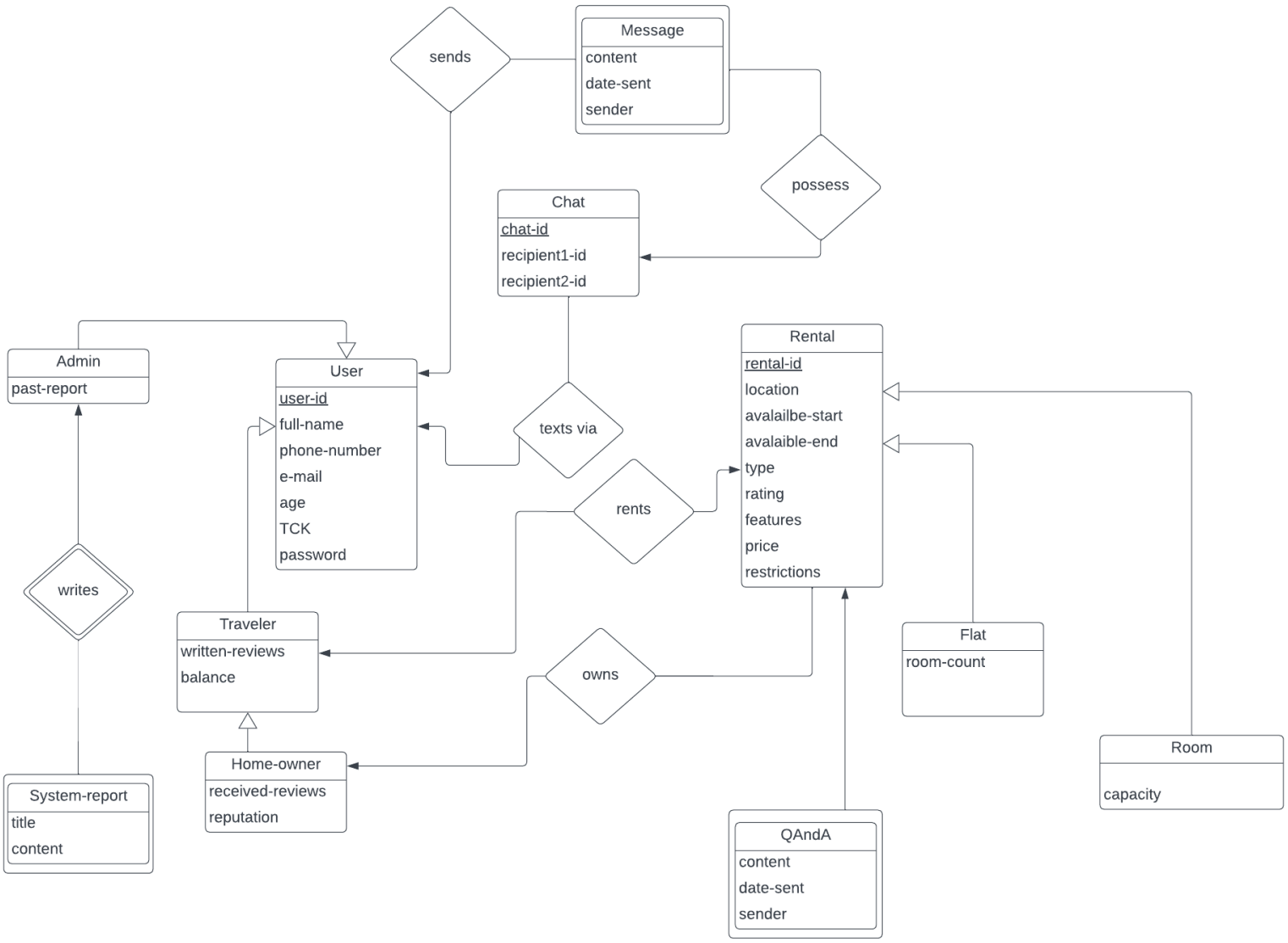
Mehmet Feyyaz Küçük - 22003550
Ender Utlu - 22001983
Ege Ayan - 22002478
Deniz Çelik - 22003271
Parsa Keihan - 22001422

**1. Revised E/R Model** **3**
**2. Relation Schemas** **5**
2.1. User 5
2.2. Traveler 5
2.3. Homeowner 6
2.4. Admin 6
2.5. Message 7
2.6. Chat 7
2.7. Rental 8
2.8. Flat 9
2.9. Room 9
2.10. Q&A 10
2.11. System Report 11
**3. Functional Dependencies and Normalization of Tables** **12**
**4. User Interface Design and Corresponding SQL Statements** **13**
4.1. Login 13
4.2. Register 14
4.3. Reset Password 15
4.4. Profile 16
4.4.1. Homeowner 16
4.4.2. Traveler 17
4.4.3 Admin 18
4.5 Rental 20
4.5.1 Homeowner 20
4.5.2 Traveler 22
4.6 Home Page 24
4.6.1 Homeowner 24
4.6.2 Traveler 25
4.7 Messaging 26
**5. Advanced Database Components** **27**
5.1. Views 27
5.2. Triggers 27
5.3. Constraints 28
**6. Implementation Plans** **29**

# 1. Revised E/R Model

After getting feedback from the teaching assistant, we revised our E/R diagram and made ten changes:

1. We added a messaging system between users, and added the relevant entities and relationships to our E/R diagram. We added a weak entity called “Message”, and a strong entity called “Chat”.
2. All of our entities were strong entities. We turned suitable ones to be weak entities such as “Feature”, “System-report”. We removed the primary keys from the weak entities.
3. We deleted the “1”s and “M”s from the diagram as the arrows already indicated the relationship between them.
4. We renamed some of the relationships because there were duplicate names.
5. Instead of having flats include rooms, we made rooms to be individually rented by inheriting rooms and flats from rental entities.
6. We added rental price and phone number attributes to rental and traveler entities respectively.
7. We split the available-date attribute of the rental entity into two attributes: available-start and available-end.
8. We deleted the feature entity and we added an array of strings named features in the rental entity.
9. We added the Q&A entity for the questions and answers about the rentals.
10. We changed the type of the reputation attribute in Homeowner entity from an int to a float.


sends
Message
content
date-sent
sender
possess
Chat
chat-id
recipient1-id
recipient2-id
Admin
past-report
User
user-id
full-name
phone-number
e-mail
age
TCK
password
texts via
Rental
rental-id
location
avalailbe-start
avalaible-end
type
rating
features
price
restrictions
rents
writes
Traveler
written-reviews
balance
owns
Flat
room-count
Room
capacity
Home-owner
received-reviews
reputation
System-report
title
content
QAndA
content
date-sent
sender

# 2. Relation Schemas

## 2.1. User

**Relational Model:**

user(user-id, full-name, e-mail, dob, TCK, password, phone-number)

**Functional Dependencies:**

user-id → name, password, e-mail, dob, TCK, phone-number

**Candidate Keys:** { (user-id) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE user (
        user-id         uuid PRIMARY KEY,
        full-name       varchar(50) NOT NULL,
        e-mail          varchar(50) NOT NULL,
        dob             datetime,
        TCK             varchar(11),
        password        varchar(15) NOT NULL,
        phone-number        varchar(15) NOT NULL
);
```

## 2.2. Traveler

**Relational Model:**

traveler(user-id, written-reviews, balance)

**Functional Dependencies:**

user-id → written-reviews, balance

**Candidate Keys:** { (user-id) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE user (
        user-id             uuid PRIMARY KEY,
        written-reviews     varchar(1000),
        balance             float
);
```

## 2.3. Homeowner

**Relational Model:**

homeowner(user-id, received-reviews, reputation)

**Functional Dependencies:**

user-id → received-reviews, reputation

**Candidate Keys:** { (user-id) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE user (
        user-id             uuid PRIMARY KEY,
        received-reviews    varchar(1000),
        reputation          float
);
```

## 2.4. Admin

**Relational Model:**

admin(user-id, past-reports)

**Functional Dependencies:**

user-id → past-report

**Candidate Keys:** { (user-id) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE user (
        user-id             uuid PRIMARY KEY,
        past-reports        text
);
```

## 2.5. Message

**Relational Model:**

message(content, date-sent, sender, chat-id)
        FK: sender references user.user-id
        FK chat-id references chat

**Functional Dependencies:**

none

**Candidate Keys:** { (content, date-sent, sender, history-id) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE message (
        content         varchar(200),
        date-sent       varchar(10),
        sender          uuid,
        history-id      uuid,
        FOREIGN KEY(sender) references user(user-id),
        FOREIGN KEY(chat-id) references chat(chat-id)
);
```

## 2.6. Chat

**Relational Model:**

chat(chat-id, recipient1-id, recipient2-id)

FK: recipient1-id references user.user-id
FK: recipient2-id references user.user-id

**Functional Dependencies:**

chat-id → recipient1-id, recipient2-id

**Candidate Keys:** { (chat-id) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE chat (
        chat-id         uuid PRIMARY KEY,
        recipient1-id  uuid,
        recipient2-id  uuid,
        FOREIGN KEY(recipient1-id) references user(user-id)
        FOREIGN KEY(recipient2-id) references user(user-id)
);
```

## 2.7. Rental

**Relational Model:**

rental(rental-id, location, available-start, available-end, restrictions, type, rating, features, price, traveler-id, homeowner-id)
FK: homeowner-id references homeowner
FK: traveler-id references traveler

**Functional Dependencies:**

rental-id → location, available-date, restrictions, type, rating, features, price

**Candidate Keys:** { (rental-id) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE flat (
```

```
    rental-id       uuid PRIMARY KEY,
    location        text,
    available-date      int,
    restrictions    int,
    type            varchar(10),
    rating          int,
    features        text[],
    price           float,
    traveler-id     uuid,
    homeowner-id        uuid,
    FOREIGN KEY(traveler-id) references traveler(user-id)
    FOREIGN KEY(homeowner-id) references homeowner(user-id)
);
```

## 2.8. Flat

**Relational Model:**

flat(<u>rental-id</u>, room-count)

**Functional Dependencies:**

rental-id → room-count

**Candidate Keys:** { (rental-id) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE flat(
    rental-id       uuid PRIMARY KEY,
    room-count      int
);
```

## 2.9. Room

**Relational Model:**

room(<u>rental-id</u>, room-type, capacity)

**Functional Dependencies:**

rental-id → room-type, capacity

**Candidate Keys:** { (rental-id) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE room (
        rental-id       uuid PRIMARY KEY,
        room-type       varchar(20),
        capacity        int
);
```

## 2.10. Q&A

**Relational Model:**

```
QandA(ask-id, answer-id, ask-date, answer-date, rental-id, question, answer)
        FK: rental-id references flat(rental-id)
        FK: ask-id references user(user-id)
        FK: answer-id references user(user-id)
```

**Functional Dependencies:**

none

**Candidate Keys:** { (ask-id, answer-id,  rental-id, answer-date, ask-date, answer, question) }

**Normal form:** BCNF

**Table Definition:**

```
CREATE TABLE QandA (
        category        varchar(20),
        rental-id       uuid,
        FOREIGN KEY(rental-id) references flat(rental-id)
);
```

## 2.11. System Report

**Relational Model:**

system-report(<u>title, content, user-id</u>)
FK: user-id references admin.user-id

**Functional Dependencies:**

user-id → title, content

**Candidate Keys:** { (title, content, user-id) }

**Normal form:** BCNF

**Table Definition:**

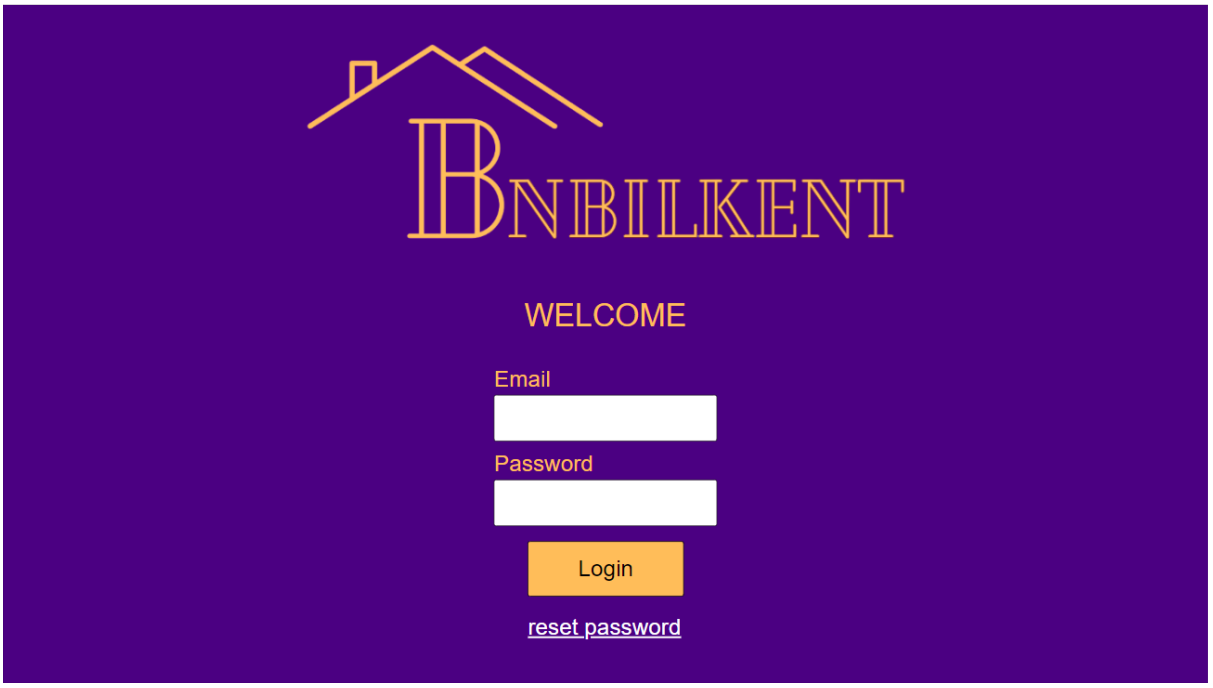
```
CREATE TABLE system-report (
        title           varchar(50),
        content         text,
        admin-id        uuid,
        FOREIGN KEY(admin-id) references admin(user-id)
);
```

# 3. Functional Dependencies and Normalization of Tables

Every normal form and every functional dependency are given in the relation schemas which is in section 2 of this report. Every relation in our design is checked if the relation is in Boyce-Codd Normal Form. Since the left side of the functional dependencies in our schemas are superkeys to the relation, they are in BCNF and do need further decomposition.

# 4. User Interface Design and Corresponding SQL Statements

## 4.1. Login




**Checking if the user exists**

SELECT * FROM user WHERE email = @email AND password = @password

## 4.2. Register


**Checking if the user exists**

SELECT * FROM user WHERE TCK = @TCK OR email = @email

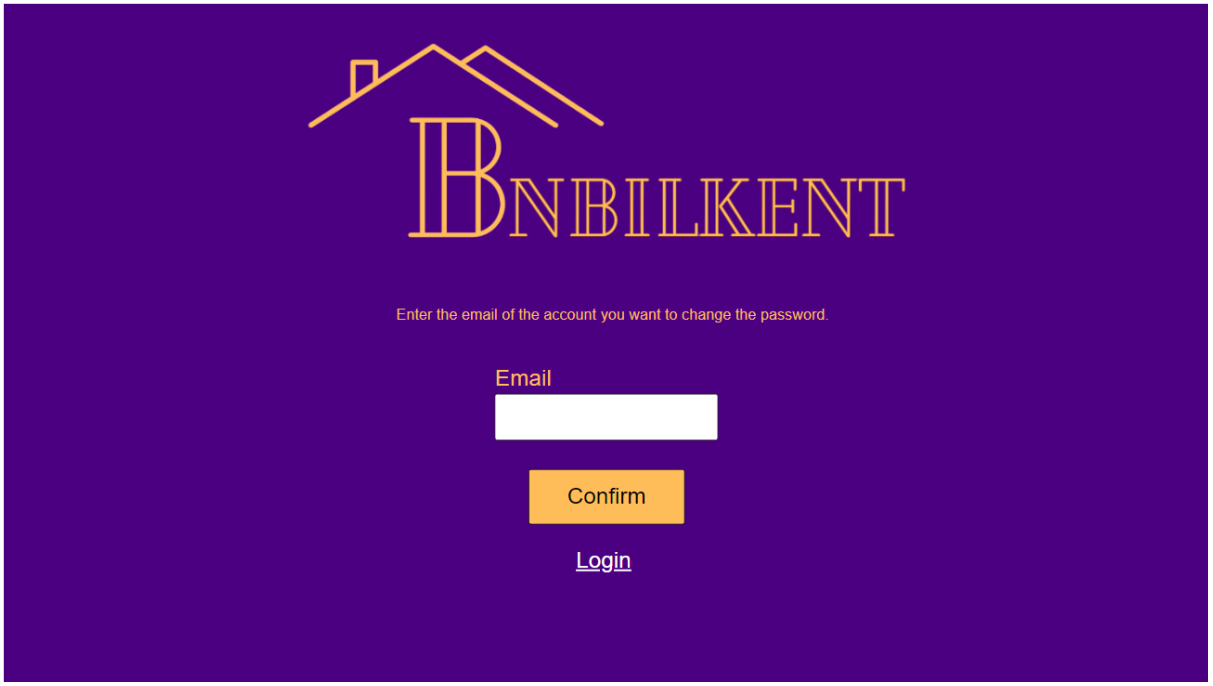
**Inserting the new user into the database (unique uuid will be generated by Spring Boot)**

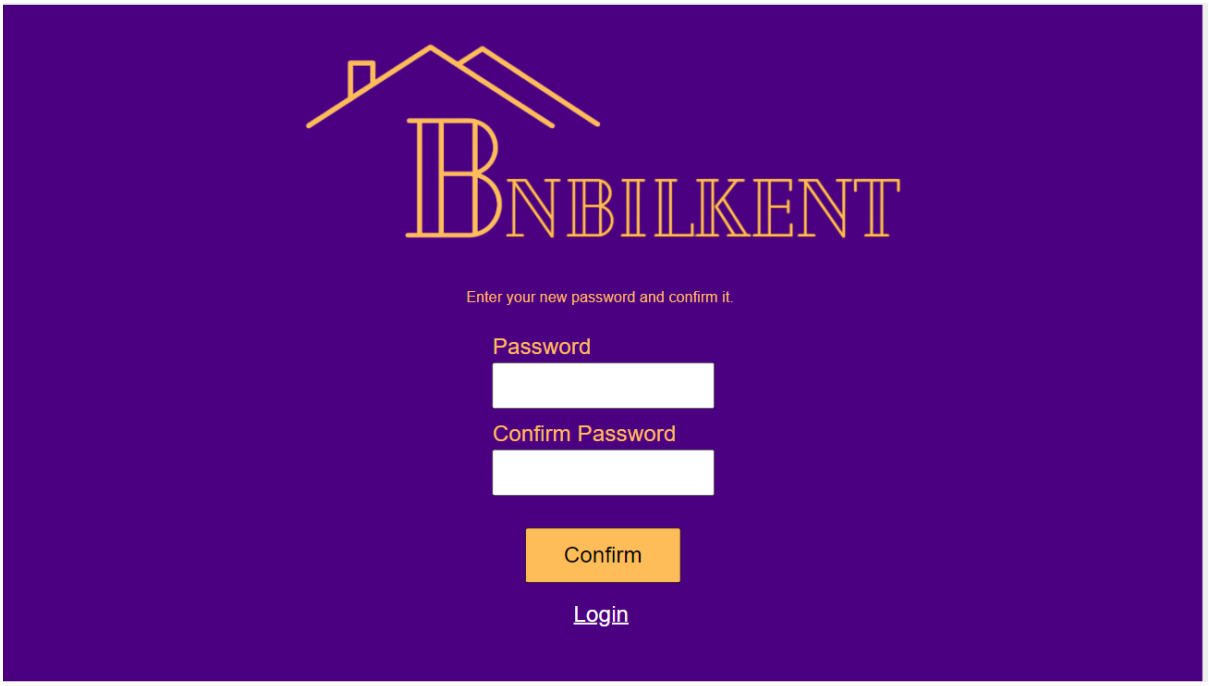
INSERT INTO user VALUES(uuid, @full-name, @email, @dob, @TCK, @password, @phone-number)

INSERT INTO traveler VALUES(uuid, NULL, 0.0)

INSERT INTO homeowner VALUES(uuid, NULL, 0.0)

## 4.3. Reset Password




**Check if the email exists**

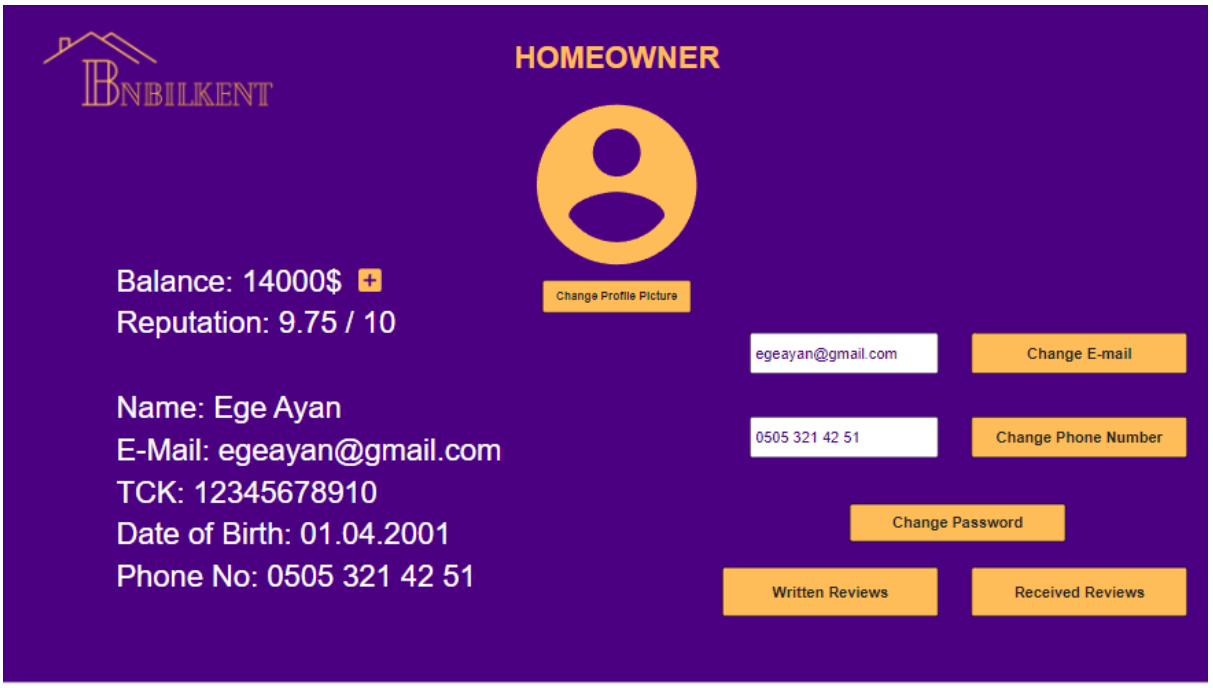
SELECT id FROM user WHERE email = @email

**Update the password of the user**

UPDATE user SET password = @new_password WHERE id = @session_id

## 4.4. Profile

### 4.4.1. Homeowner




**Fetch homeowner information from a view (see section 5.1)**

SELECT * FROM homeowner_profile WHERE id = @session_id

**Change the email of the homeowner**

UPDATE user SET email = @email WHERE id = @session_id

**Change the phone number of the homeowner**

UPDATE user SET phone-number = @phone-number WHERE id = @session_id

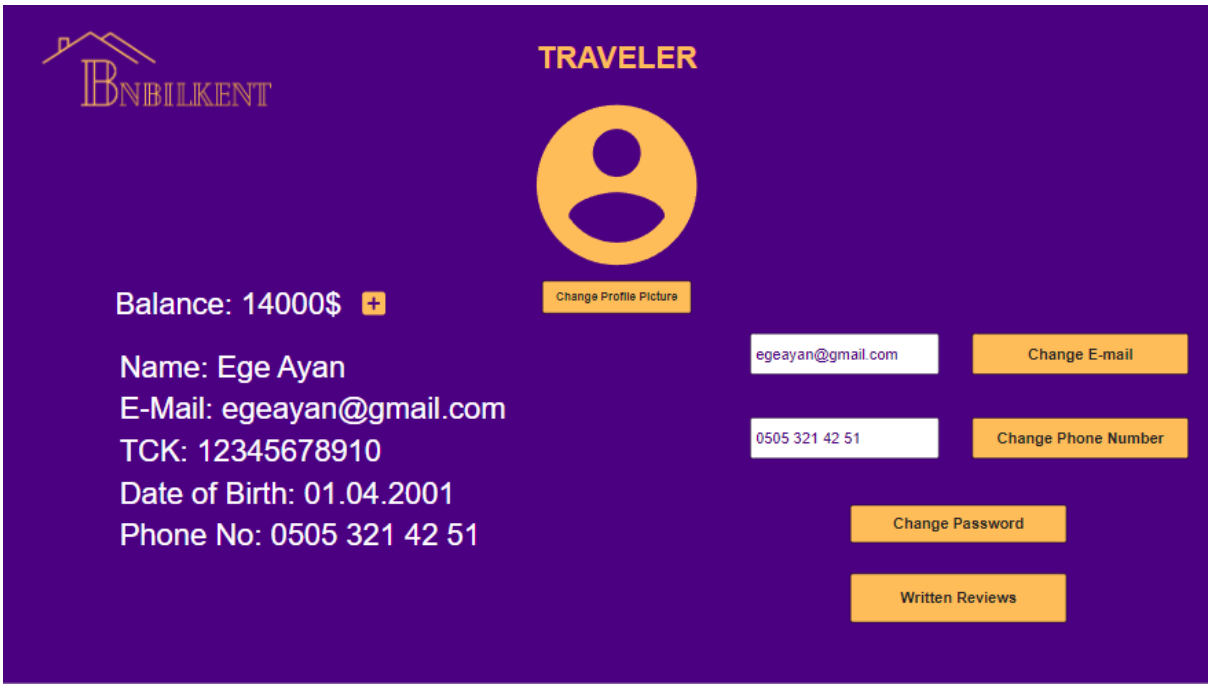
**Fetch the written reviews from the database**

SELECT written-reviews
FROM user U, homeowner H
WHERE U.user-id = H.user-id AND U.user-id = @session_id

**Fetch the received reviews from the database**

SELECT written-reviews
FROM user U, homeowner H
WHERE U.user-id = H.user-id AND T.user-id = U.user-id AND U.user-id = @session_id

### 4.4.2. Traveler




**Fetch traveler information from a view (see section 5.1)**

SELECT * FROM traveler_profile WHERE id = @session_id

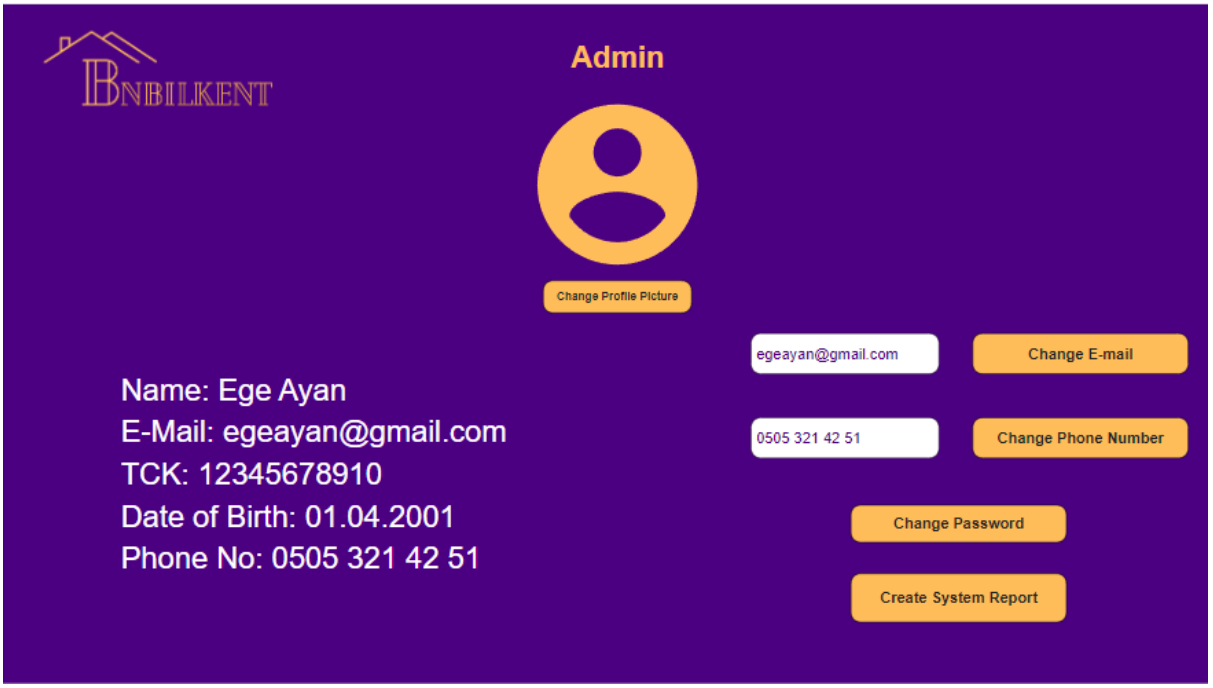
**Change the email of the traveler**

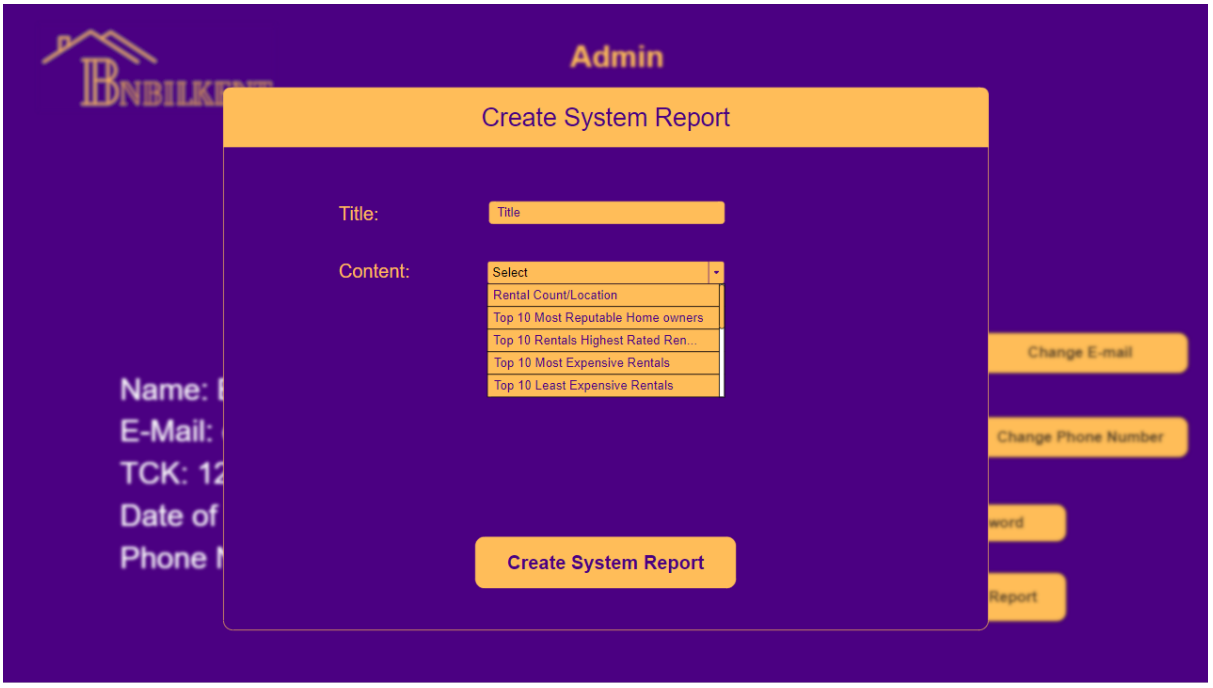
UPDATE user SET email = @email WHERE id = @session_id

**Change the phone number of the traveler**

UPDATE user SET phone-number = @phone-number WHERE id = @session_id

### 4.4.3 Admin




**Fetch admin information from the view (see section 5.1)**

SELECT * FROM admin_profile WHERE id = @session_id

**Add a new system report to the database**

INSERT INTO system_report VALUES(@title, @content, @session_id)

**Create a new system report content**

A. SELECT location, COUNT(*) as count
   FROM rental
   GROUP BY location

B. SELECT full-name, reputation
   FROM homeowner_profile
   ORDER BY reputation DESC
   LIMIT 10

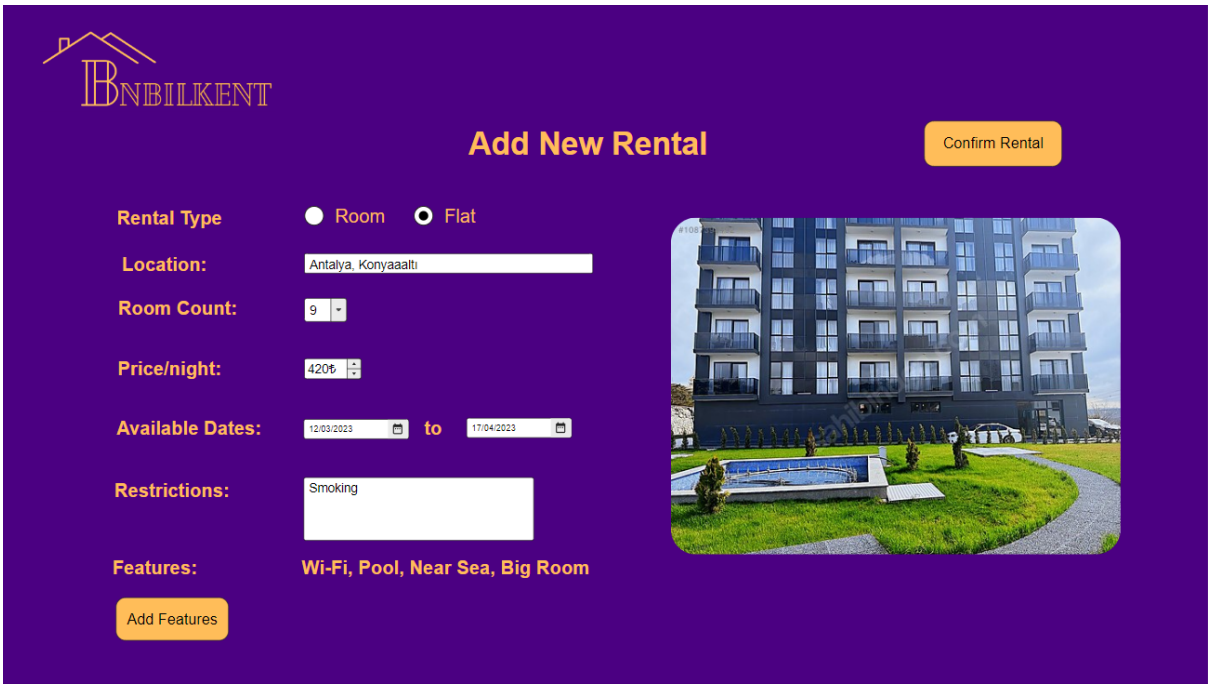
C. SELECT rental-id, location, rating
   FROM rental
   ORDER BY rating DESC
   LIMIT 10

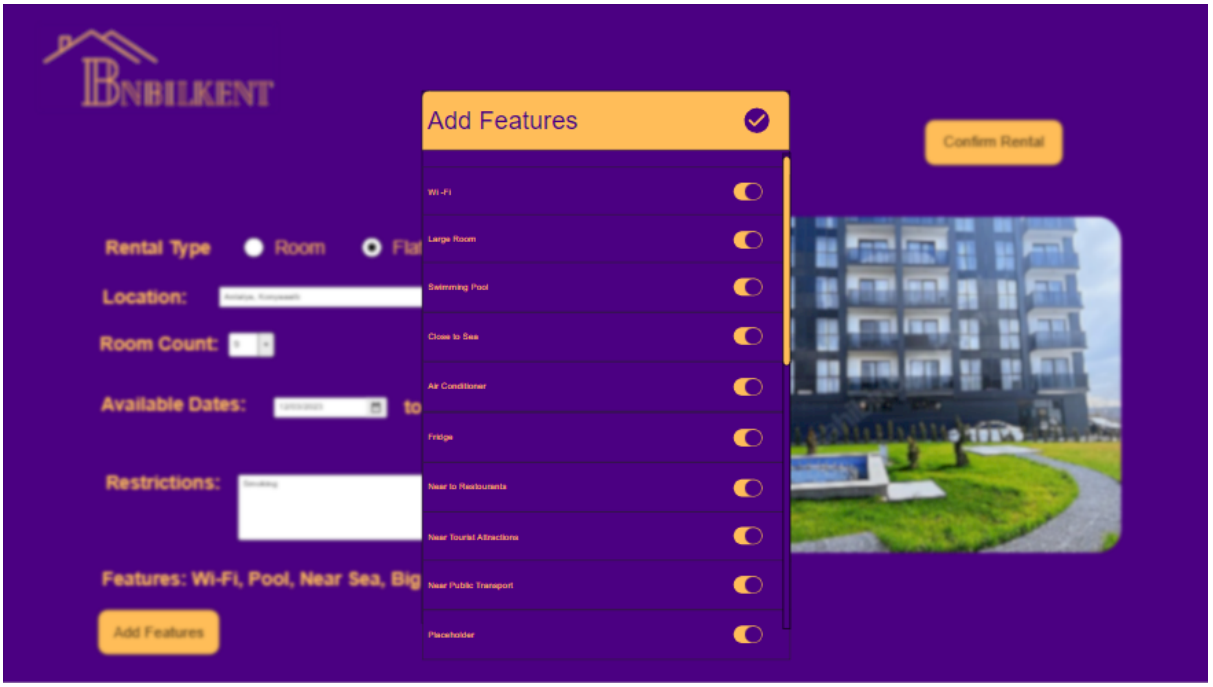
D. SELECT rental-id, location, price
   FROM rental
   ORDER BY price DESC
   LIMIT 10

E. SELECT rental-id, location, price
   FROM rental
   ORDER BY price ASC
   LIMIT 10

## 4.5 Rental

### 4.5.1 Homeowner




**Add the new flat into the database (unique uuid will be generated by Spring Boot)**

INSERT INTO rental VALUES(uuid, @location, @avalaible-start, @avalaible-end, @restrictions, @type, NULL, @features, @price, NULL, @session_id)

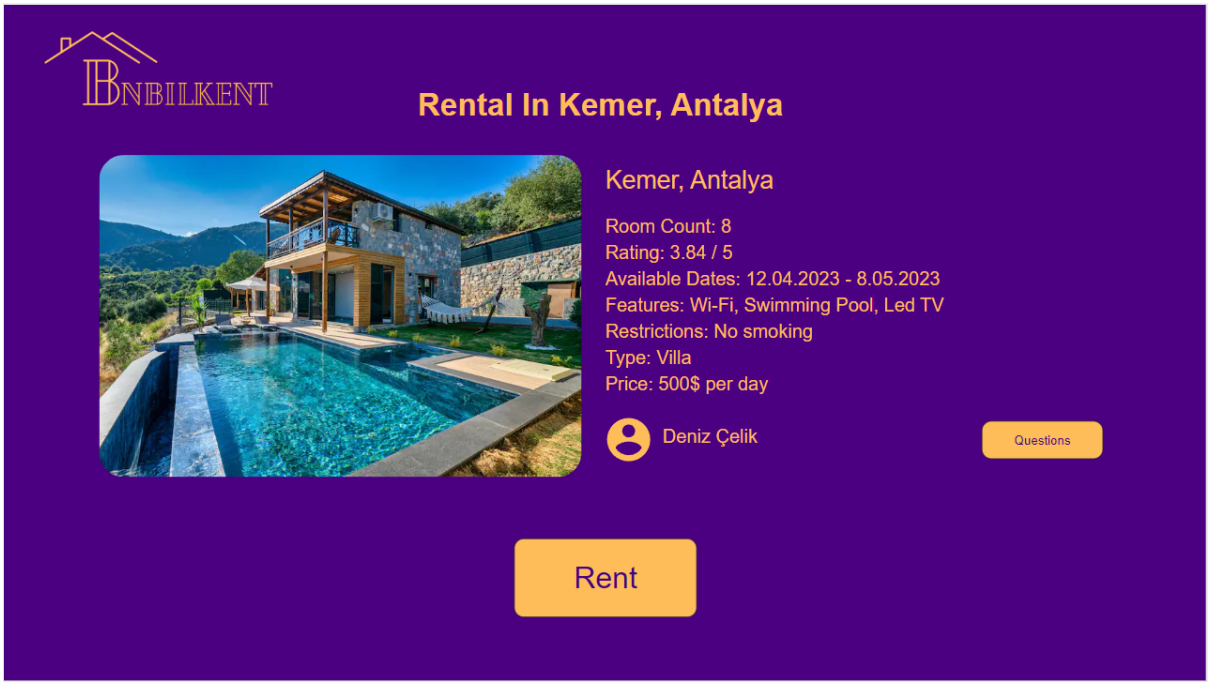
INSERT INTO flat VALUES(uuid, @room-count)

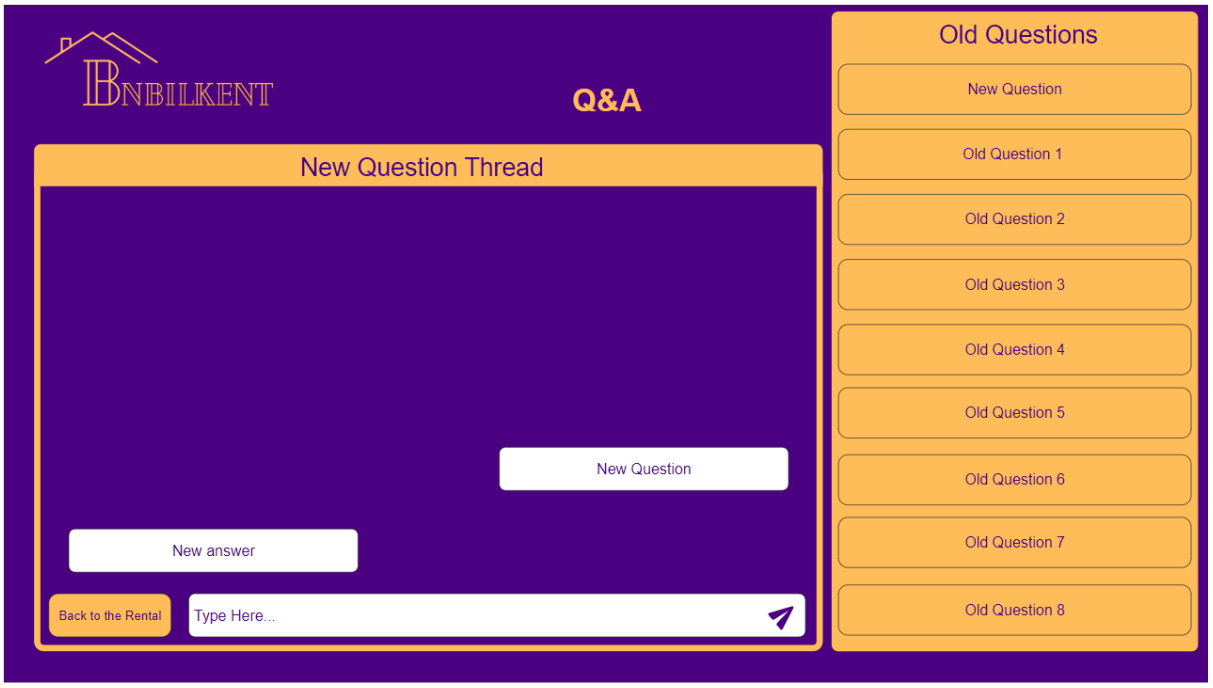
**Add the new room into the database (unique uuid will be generated by Spring Boot)**

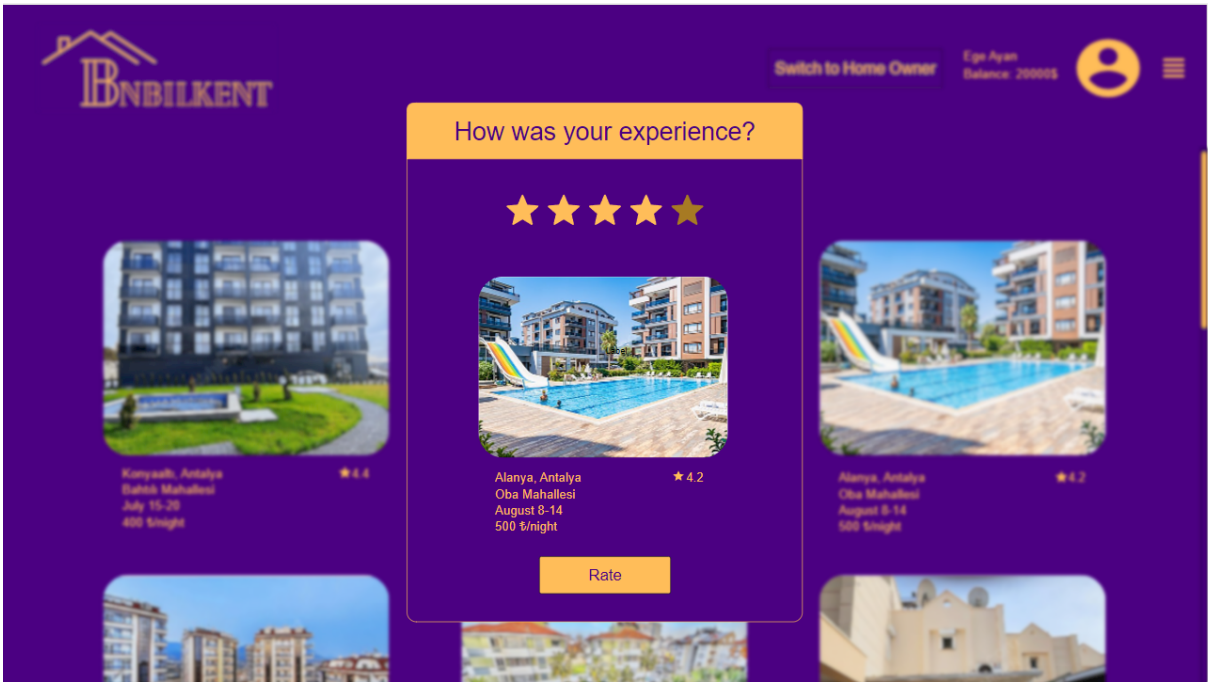
INSERT INTO rental VALUES(uuid, @location, @avalaible-start, @avalaible-end, @restrictions, @type, NULL, @features, @price, NULL, @session_id)

INSERT INTO room VALUES(uuid, @capacity)

### 4.5.2 Traveler






**Fetch rental information**

SELECT *
FROM rental R, homeowner H
WHERE R.homeowner-id = H.user-id AND R.rental-id = @rental-id

**Fetch answers and questions**

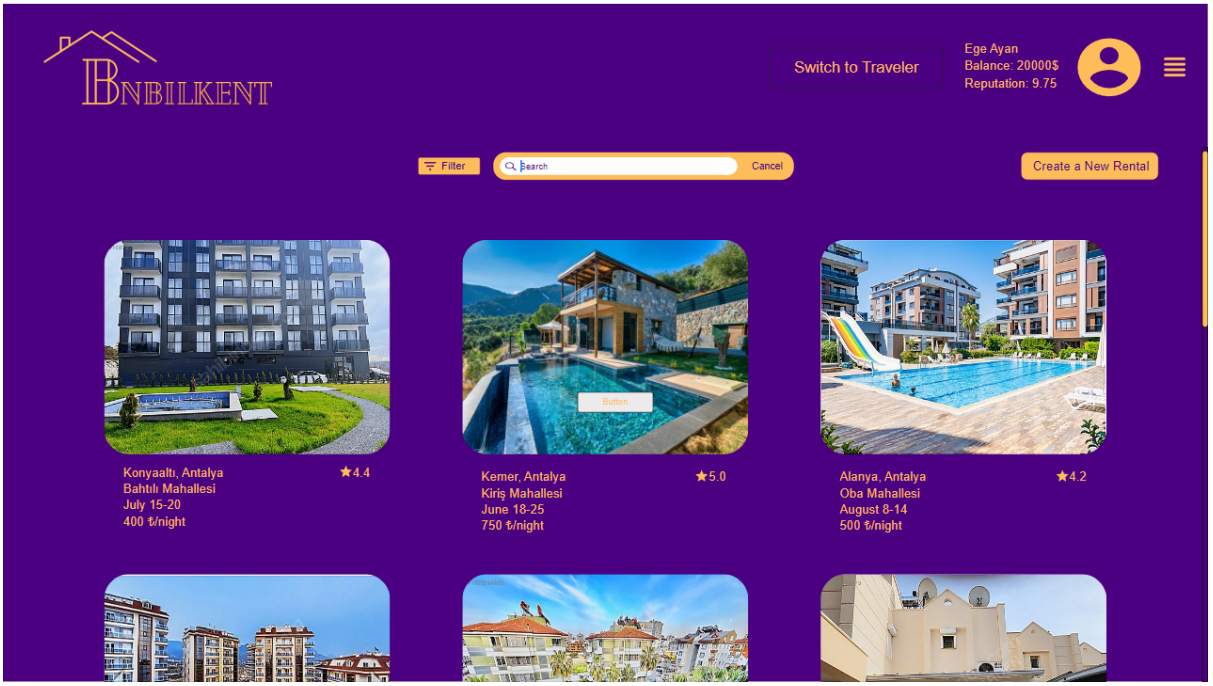
SELECT * FROM QAndA, rental WHERE QAndA.rental-id = rental.rental-id AND rental-id = @rental-id

**Update the rating of a rental**

UPDATE rental SET rating = @rating WHERE rental-id = @rental-id

# 4.6 Home Page

## 4.6.1 Homeowner



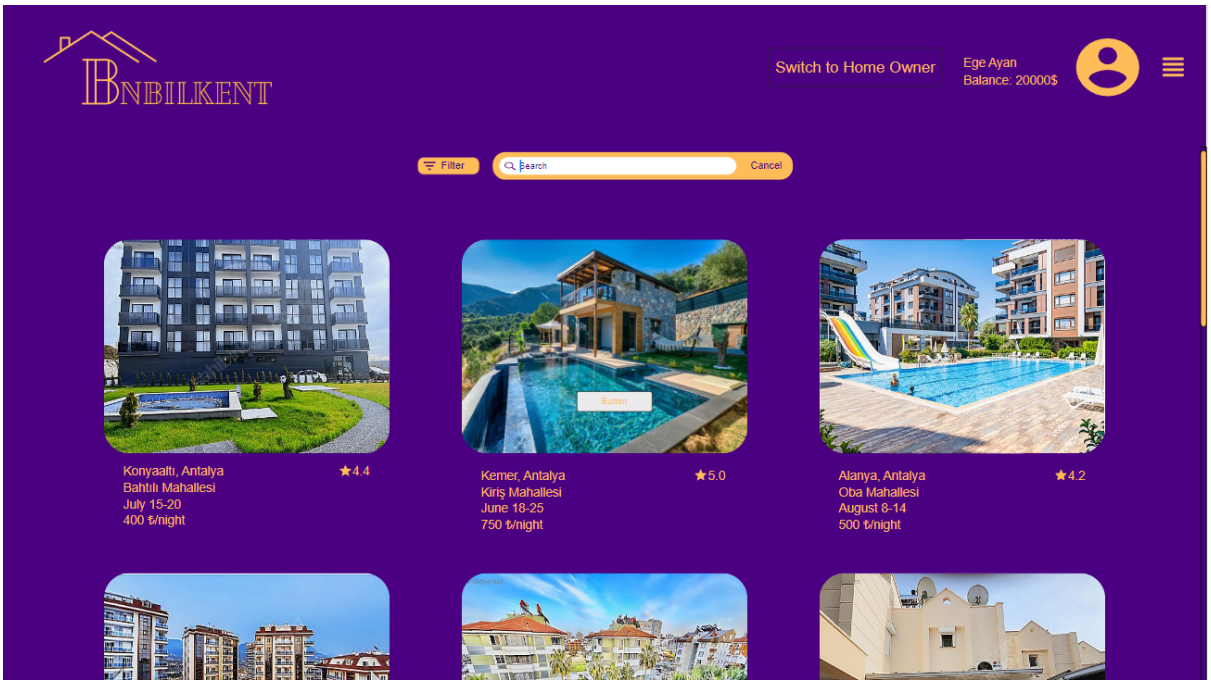

**Fetch all rentals from the database**

SELECT location, available-start, available-end, price, rating FROM rental ORDER BY available-start

**Fetch homeowner information for top-right of the screen**

SELECT full-name, balance, reputation
FROM homeowner_profile
WHERE user-id = @session_id

### 4.6.2 Traveler



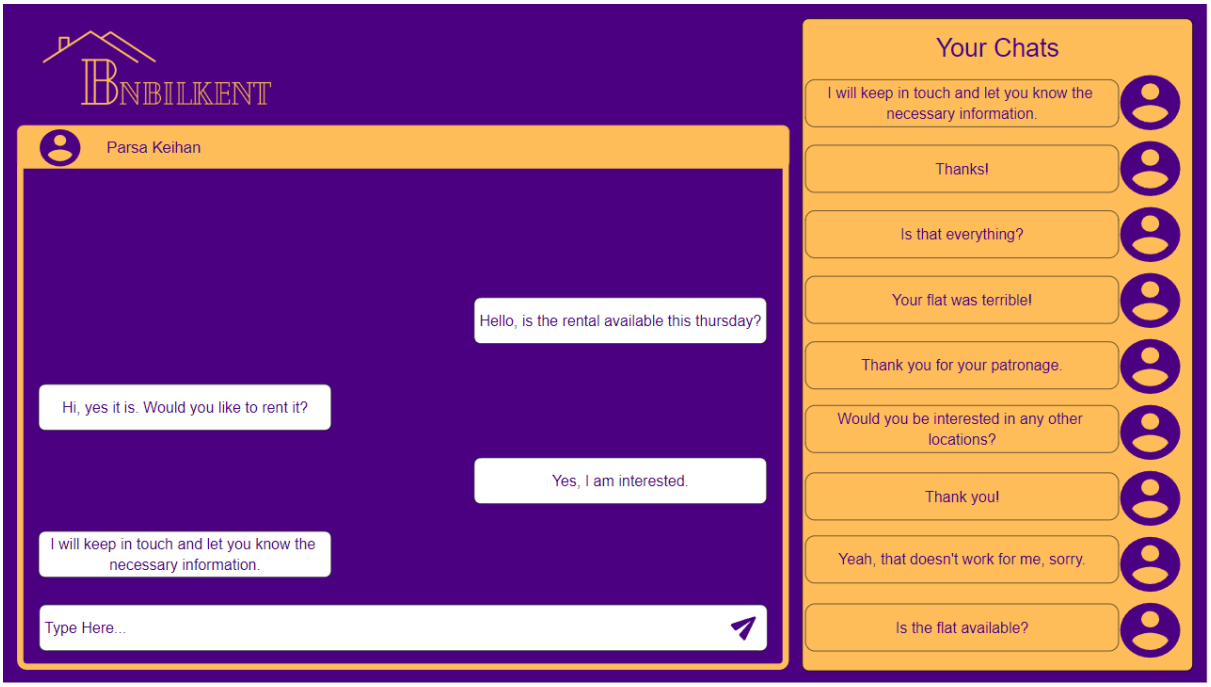

**Fetch all rentals from the database**

SELECT location, available-start, available-end, price, rating FROM rental ORDER BY available-start

**Fetch homeowner information for top-right of the screen**

SELECT full-name, balance
FROM traveler_profile
WHERE user-id = @session_id

## 4.7 Messaging




**Select the chat of user and all of its messages**

SELECT *
FROM user U, chat C, message M
WHERE U.user-id = C.recipient1-id AND C.chat-id = M.chat-id AND (M.sender = C.recipient1-id OR M.sender = C.recipient2-id)

# 5. Advanced Database Components

## 5.1. Views

This view is used to fetch data from a homeowner perspective. This way, password and other sensitive data is abstracted and joining tables is not needed every time data needs to be fetched from a homeowner.

- CREATE VIEW homeowner_profile as
  SELECT full-name, TCK, phone-no, reputation, balance, e-mail, DATEDIFF ( YEAR , dob, GETDATE() ) as age
  FROM user U, homeowner H, Traveler T
  WHERE user-id = @user-id AND U.user-id = H.user-id AND U.user-id = T.user-id

This view is used to fetch data from a traveler perspective. This way, password and other sensitive data is abstracted and joining tables is not needed every time data needs to be fetched from a traveler.

- CREATE VIEW traveler_profile as
  SELECT full-name, TCK, phone-no, balance, e-mail, DATEDIFF ( YEAR , dob, GETDATE() ) as age
  FROM user U, traveler T
  WHERE user-id = @user-id AND T.user.id = U.user-id

This view is used to fetch data from an admin perspective. This way, password and other sensitive data is abstracted and joining tables is not needed every time data needs to be fetched from an admin.

- CREATE VIEW admin_profile as
  SELECT full-name, TCK, phone-no, e-mail, DATEDIFF ( YEAR , dob, GETDATE() ) as age
  FROM user U, admin A
  WHERE user-id = @user-id AND U.user-id = A.user-id

## 5.2. Triggers

- A trigger will be used to update the rating of a rental whenever a traveler gives a rating to that certain rental.
- A trigger will be used to update the reputation of a homeowner whenever the rating of a rental changes.
- A trigger will be used to update the balance of both homeowners and travelers. Balance will be decreased by the price amount for travelers, it will be increased by the price amount for homeowners.

## 5.3. Constraints

- Only homeowners will be able to put up rentals. Travelers will not be able to put up rentals. However, both types of users can rent them.

# 6. Implementation Plans

In our application, PostgreSQL will be used as a database management system. Spring Boot framework will be used for server-side development and React framework will be used for client-side development.