CS2102 Database Systems

Project Report

Team 65

ISSUED BY

Professor Prabawa Adi Yoga Sidi

GROUP MEMBERS

JOSIAH EZEKIEL KHOO SHAO QI	A0196634U
TAN HIN KHAI STEPHEN	A0196608R
LAWSON TEO KHAY WEE	A0204928B
CALVIN CHEN XINGZHU	A0190624H
CHEN YU MING	A0196633W

Roles and Responsibilities

No.	Name	Responsibility
1	JOSIAH EZEKIEL KHOO SHAO QI	Front-End and Back-End Development, SQL Expert
2	TAN HIN KHAI STEPHEN	Basic Front-End and Back-End Development, Project Report
3	LAWSON TEO	Front-End and Back-End Development, Admin Dashboard
4	CALVIN CHEN XINGZHU	Front-End and Back-End Development, ER Diagram
5	CHEN YU MING	Front-End and Back-End Development, UI/UX

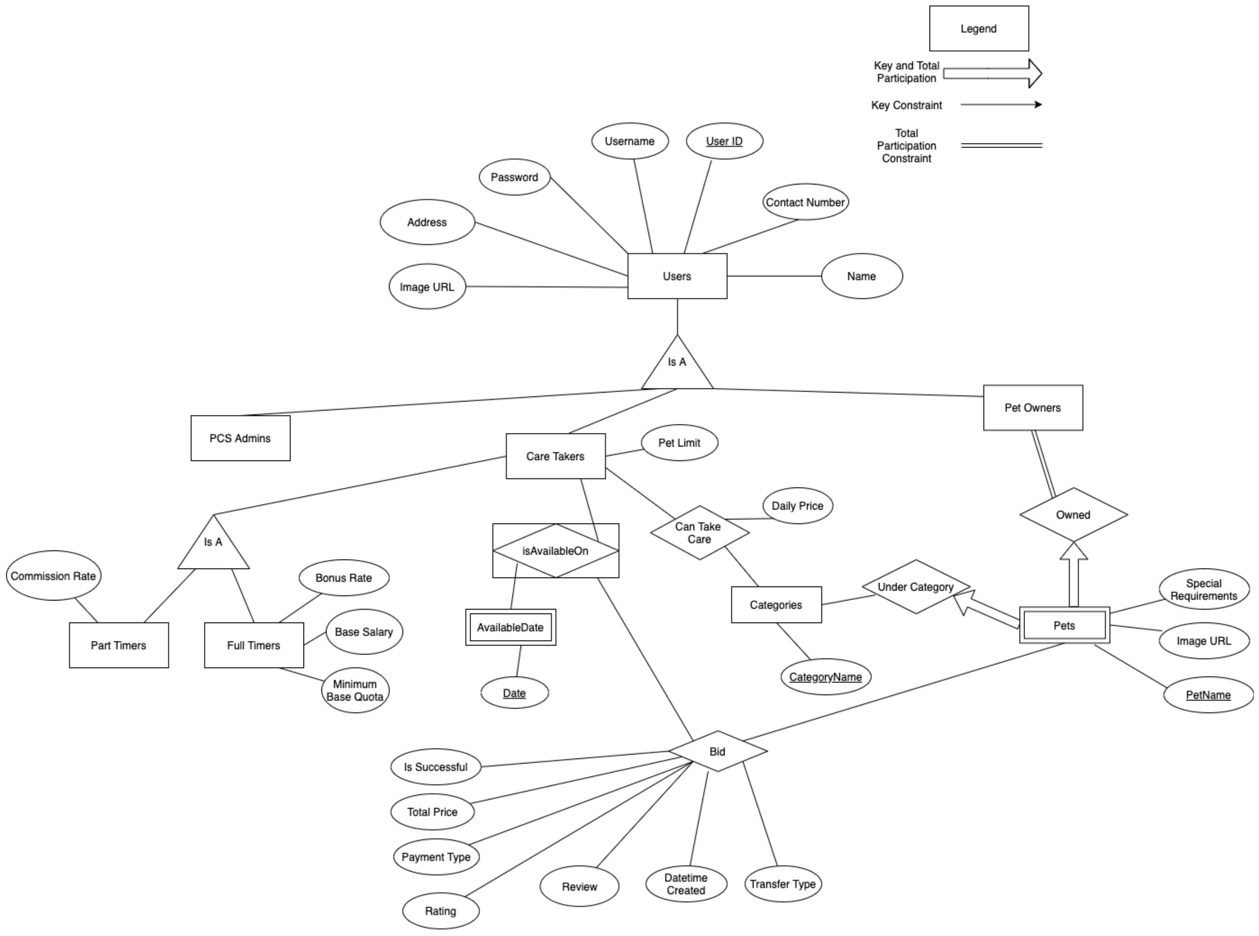
Software Tools/ Frameworks

-	Backend:	ExpressJS + NodeJS
-	Frontend:	ReactJS + React-Bootstrap
-	Database:	PostgreSQL
-	Hosting:	Heroku

ER Model

Application constraints not captured by Relational Schema/ER diagram

- The overlap constraint of allowing a User to be a Pet Owner and a Caretaker.
- The overlap constraint of having both successful and unsuccessful bids in Bids.



Data Requirements and Functionalities

a. Data requirements

1. Authentication

Users need the following additional details to sign up:

- Username (up to 100 characters)
- Password (up to 100 characters)
- Name (up to 100 characters)
- Contact Number (up to 8 characters)
- Address (up to 255 characters)
- Pet Owner Check (Yes or No)
- Caretaker Check (Part-Time, Full-Time, or No)

Users need the following to log in:

- Username (up to 100 characters)
- Password (up to 100 characters)

2. Users

Users can also update their profile by changing the following details:

- Password (up to 100 characters)
- Full Name (up to 100 characters)
- Contact Number (up to 8 characters)
- Address (up to 255 characters)

3. Pet Owners

As a user, they can be a pet owner.
As a pet owner, user can add a new pet to their account with the following details:

- Pet Name (up to 100 characters)
- Pet Category (Handled by UI’s drop-down menu which pulls data from the table ‘Categories’)
- Special Requirements (up to 255 characters)
- Image URL (up to 255 characters)

If there is at least one pet added, pet owners can create a bid to find caretakers to look after their pet(s), with the following details:

- Pet (Handled by UI’s drop-down menu which shows all of user’s pets)
- Time Period (Selected from UI’s datepicker)
- Caretaker (Selected from UI’s drop-down menu of available caretaker names)
- Transfer Type (Select pet transfer type from UI’s drop-down menu)
- Payment Type (Select payment type from UI’s drop-down menu)

4. Caretakers

If users have the role of a caretaker, they will either be a Full-Time Caretaker, or a Part-Time Caretaker.

Full-Time Caretakers

- Can submit leave applications with the following information:
  - Date Period (Handled by UI’s datepicker)

Part-Time Caretakers

- Can choose to confirm or decline bids offered to them by selecting ‘Confirm’ or ‘Decline’ buttons on the user interface
- Can specify available dates to work with the following information:
  - Date Period (Handled by UI’s datepicker)

5. Administrator

An Administrator can view all underperforming Full-Time Caretakers in a particular month and year by inputting:

- Month (Handled by UI’s drop-down)
- Year (Handled by UI’s drop-down)

An Administrator can also view the average satisfaction rate for all the pet categories that were looked after in a particular month and year by inputting:

- Month (Handled by UI’s drop-down)
- Year (Handled by UI’s drop-down)

b. Functionalities

1. **Authentication**
    - Register for a new account
    - Login to an existing account
2. **Users**
    - Select to be a Pet Owner
    - Select to be a Full-time/Part-time Caretaker
    - View user profile
3. **Pet Owners**
    - View all owned pets
    - Add a pet
    - Create bid for a pet
    - View all ongoing bids
4. **Caretakers**

    Full-Time Caretaker:
    - View all bids assigned to them

- Submit leave application
- View all leave dates on a Monthly Calendar
- View number of pet days
- View salary

Part-Time Caretaker:

- View all bids offered to them
- Accept or Decline a bid
- View confirmed bids
- Indicate available working dates
- View all available working dates on a Monthly Calendar
- View number of pet days
- View salary

5. **Administrator**
   - View the total number of pets the company has took care of according to the month and year
   - View the month with the highest number of pet days
   - View all *underperforming** caretakers according to a particular month and year
   - View the average satisfaction rate for all pet categories according to all particular month and year
   - View the total pet days any caretaker has, the total earnings from the caretaker, as well as the total salary to pay him/her

**underperforming* refers to Full-Time Caretakers that did not clock at least 60 pet days in a month OR has an average rating less than 2.5.

c. Interesting / Non-trivial aspects of our application's functionalities / implementation

1. When pet owners create a bid and select caretakers from a drop down list, choosing a caretaker will immediately reflect their profile, which shows their image, average rating, as well as the various categories this caretaker is skilled at looking after.
2. As an admin, we are able to view the earnings brought in by all our caretakers as well as the amount we have to pay them. Hence, we can easily view all the profits in a month.
3. We also have an admin table in our database, which stores all the constants required for the company. This allows for easy access and easy updating for changes in the future, since all queries/functions will be updated accordingly. Some examples are:
   a. Salary Multiplier for caretakers with good ratings
   b. Part Time Commission Rate
   c. Full Time Bonus Rate for full-time caretakers with more than 60 pet days
   d. Pet Limit for full-time/part-time caretakers

Data Schema

Justification for using serial id for Users:

We do not want to expose username and login credentials in the user profile url which can be viewed by others, causing cyber-security risks for our clients.

1. **Admin:**
FDs: { all trivial FDs }
This table is in BCNF form.

```
CREATE TABLE Admin (
    good_review_full_time_total_price_multiplier FLOAT NOT NULL DEFAULT 1.2,
    full_time_base_salary INT NOT NULL DEFAULT 3000,
    poor_review_pet_limit INT NOT NULL DEFAULT 2,
    pet_limit INT NOT NULL DEFAULT 5,
    minimum_work_days_in_block INT NOT NULL DEFAULT 150,
    minimum_work_blocks INT NOT NULL DEFAULT 2,
    full_time_bonus_pet_day_threshold INT NOT NULL DEFAULT 60,
    part_time_commission_rate NUMERIC(3, 3) NOT NULL DEFAULT 0.75,
    full_time_bonus_rate NUMERIC(3, 3) NOT NULL DEFAULT 0.80,
    PRIMARY KEY (good_review_full_time_total_price_multiplier,
        full_time_base_salary,
        poor_review_pet_limit,
        pet_limit,
        minimum_work_days_in_block,
        minimum_work_blocks,
        full_time_bonus_pet_day_threshold,
        part_time_commission_rate,
        full_time_bonus_rate),
    CONSTRAINT full_time_base_salary_positive CHECK (full_time_base_salary >= 0),
    CONSTRAINT good_review_full_time_total_price_multiplier_positive CHECK (
        good_review_full_time_total_price_multiplier >= 0),
    CONSTRAINT poor_review_pet_limit_positive CHECK (poor_review_pet_limit >= 0),
    CONSTRAINT positive_pet_limit_positive CHECK (pet_limit >= 0),
    CONSTRAINT minimum_work_days_in_block_positive CHECK (minimum_work_days_in_block >=
0),
    CONSTRAINT minimum_work_blocks_positive CHECK (minimum_work_blocks >= 0),
    CONSTRAINT full_time_bonus_pet_day_threshold_positive CHECK (
        full_time_bonus_pet_day_threshold >= 0),
    CONSTRAINT part_time_commission_rate_positive CHECK (part_time_commission_rate >= 0),
    CONSTRAINT full_time_bonus_rate_positive CHECK (full_time_bonus_rate >= 0));
```

2. Users:
FDs: {user_id->username, user_id->password, user_id->contact_number, user_id->name, user_id->address, user_id->is_pcs_admin, user_id->is_pet_owner, user_id->image_url}
Primary Key: {user_id}
This table is in BCNF form.

```
CREATE TABLE Users (
    user_id SERIAL PRIMARY KEY,
    username VARCHAR(100) NOT NULL,
    password VARCHAR(100) NOT NULL,
    contact_number VARCHAR(20) NOT NULL,
    name VARCHAR(100) NOT NULL,
    address VARCHAR(100) NOT NULL,
    is_pcs_admin BOOLEAN NOT NULL,
    is_pet_owner BOOLEAN NOT NULL,
    image_url VARCHAR(255),
    unique(username));
```

3. Caretakers:
FDs: {user_id} -> {bonus_rate}, {user_id} -> {base_salary} , {user_id} -> {minimum_base_quota}, {user_id}->{commission_rate}, {user_id} -> {is_full_time}
Primary Key: {user_id}
This table is in BCNF form.

```
CREATE TABLE CareTakers (
    user_id INTEGER PRIMARY KEY,
    bonus_rate NUMERIC(3, 3),
    base_salary INTEGER,
    minimum_base_quota INTEGER,
    commission_rate NUMERIC(3, 3),
    is_full_time BOOLEAN NOT NULL,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    CONSTRAINT bonus_rate_positive CHECK (
        bonus_rate >= 0
        OR bonus_rate IS NULL),
    CONSTRAINT bonus_rate_upper_limit CHECK (
        bonus_rate <= 1
        OR bonus_rate IS NULL),
    CONSTRAINT base_salary_postive CHECK (
        base_salary >= 0
        OR base_salary IS NULL),
    CONSTRAINT minimum_base_quota CHECK (
        minimum_base_quota >= 0
        OR minimum_base_quota IS NULL),
    CONSTRAINT is_full_time_contain_bonus_and_salary_and_quota CHECK (
        NOT is_full_time
        OR (bonus_rate IS NOT NULL
            AND base_salary IS NOT NULL
            AND minimum_base_quota IS NOT NULL
            AND commission_rate IS NULL)),
    CONSTRAINT is_not_full_time_contain_commission CHECK (is_full_time
OR (COALESCE(bonus_rate, base_salary, minimum_base_quota) IS NULL
AND commission_rate IS NOT NULL)));
```

4. Categories:
FD: {name} -> {full_time_daily_price}
Primary Key: {name}
This table is in BCNF form.

```
CREATE TABLE Categories (
    name VARCHAR(100) PRIMARY KEY,
    full_time_daily_price INTEGER NOT NULL,
    CONSTRAINT full_time_daily_price_positive CHECK (full_time_daily_price >= 0));
```

5. OwnedPets:
FDs:{pet_owner_user_id, pet_name -> category_name, pet_owner_user_id, pet_name -> special_requirements , pet_owner_user_id, pet_name -> image_url}
-> {category_name, special_requirements, image_url}
Primary Key: {pet_owner_user_id, pet_name -> category_name}
This table is in BCNF form.

```
CREATE TABLE OwnedPets (
   pet_owner_user_id INTEGER,
   category_name VARCHAR(100),
   pet_name VARCHAR(100),
   special_requirements VARCHAR(100),
   image_url VARCHAR(255),
   PRIMARY KEY (pet_owner_user_id, pet_name),
   FOREIGN KEY (pet_owner_user_id) REFERENCES Users(user_id),
   FOREIGN KEY (category_name) REFERENCES Categories(name));
```

6. CanTakeCare:
FDs: {care_taker_user_id, category_name} -> {daily_price}
This table is in BCNF form.
Primary Key: {care_taker_user_id, category_name}

```
CREATE TABLE CanTakeCare (
   care_taker_user_id INTEGER,
   category_name VARCHAR(100),
   daily_price INTEGER NOT NULL,
   PRIMARY KEY (care_taker_user_id, category_name),
   FOREIGN KEY (care_taker_user_id) REFERENCES CareTakers(user_id),
   FOREIGN KEY (category_name) REFERENCES Categories(name),
   CONSTRAINT daily_price_positive CHECK (daily_price >= 0));
```

7. IsAvailableOn:
FDs: {all trivial FDs}
This table is in BCNF form.
Primary Key: {care_taker_user_id, available_date}

```
CREATE TABLE IsAvailableOn (
   care_taker_user_id INTEGER,
   available_date DATE,
   PRIMARY KEY (care_taker_user_id, available_date),
   FOREIGN KEY (care_taker_user_id) REFERENCES CareTakers(user_id));
```

8. Bid:
FDs:
{care_taker_user_id, start_date, end_date, pet_owner_user_id, pet_name} -> { is_success},
{care_taker_user_id, start_date, end_date, pet_owner_user_id, pet_name} -> {payment_type},
{care_taker_user_id, start_date, end_date, pet_owner_user_id, pet_name} -> {transfer_type},
{care_taker_user_id, start_date, end_date, pet_owner_user_id, pet_name} -> {review},
{care_taker_user_id, start_date, end_date, pet_owner_user_id, pet_name} -> {total_price},
{care_taker_user_id, start_date, end_date, pet_owner_user_id, pet_name} -> {datetime_created}
Primary Key: {care_taker_user_id, start_date, end_date, pet_owner_user_id, pet_name}
This table is in BCNF form.

```
CREATE TABLE Bid (
   care_taker_user_id INTEGER,
   pet_owner_user_id INTEGER,
   pet_name VARCHAR(100),
   is_success BOOLEAN NOT NULL DEFAULT FALSE,
   payment_type VARCHAR(100) NOT NULL,
   transfer_type VARCHAR(100) NOT NULL,
   total_price INTEGER,
   review VARCHAR(1000),
   rating INTEGER,
   start_date DATE NOT NULL,
   end_date DATE NOT NULL,
   datetime_created TIMESTAMPTZ NOT NULL DEFAULT NOW(),
   PRIMARY KEY (
      care_taker_user_id,
      start_date,
      end_date,
      pet_owner_user_id,
      pet_name),
   FOREIGN KEY (care_taker_user_id) REFERENCES CareTakers(user_id),
   FOREIGN KEY (pet_owner_user_id, pet_name) REFERENCES OwnedPets(pet_owner_user_id,
pet_name),
   CONSTRAINT bid_success CHECK (
      CASE
         WHEN is_success THEN (total_price IS NOT NULL)
         ELSE TRUE
      END),
   CONSTRAINT total_price_positive
   CHECK (total_price >= 0 OR total_price IS NULL),
   CONSTRAINT review_rating_exists_together CHECK (
      (review IS NULL AND rating IS NULL)
      OR (review IS NOT NULL AND rating IS NOT NULL)));
```

All tables are in BCNF.

SQL Triggers

This trigger enforces multiple system requirements, including:

1. Auto-confirming a bid when the pet owner bids for a full time caretaker
2. Sets the price to either
    a. The inflated bonus price if the caretaker is a full-time caretaker and has a rating of more than or equals to 4
    b. The normal price if the caretaker is a full-time caretaker and has a rating less than 4

This trigger is fired automatically when a bid is inserted into the bid table.

```
CREATE OR REPLACE FUNCTION bid_full_time_care_taker_auto_confirm () RETURNS TRIGGER
AS $$
DECLARE is_full_time BOOLEAN;
DECLARE base_price INTEGER;
DECLARE rating FLOAT;
DECLARE multiplier FLOAT;
DECLARE number_days INTEGER;
```

```
BEGIN
SELECT c.is_full_time,
        ctc.daily_price,
        c.rating,
        a.good_review_full_time_total_price_multiplier,
        (NEW.end_date - NEW.start_date + 1) INTO is_full_time,
        base_price,
        rating,
        multiplier,
        number_days
FROM CareTakersWithPetLimitAndRating c,
        CanTakeCare ctc,
        OwnedPets p,
        Admin a
WHERE c.user_id = NEW.care_taker_user_id
        AND p.pet_owner_user_id = NEW.pet_owner_user_id
        AND p.pet_name = NEW.pet_name
        AND p.category_name = ctc.category_name
        AND ctc.care_taker_user_id = NEW.care_taker_user_id;
IF is_full_time = TRUE THEN IF rating >= 4 THEN
UPDATE bid
SET total_price = base_price * multiplier * number_days,
        is_success = TRUE
WHERE bid.care_taker_user_id = NEW.care_taker_user_id
        AND bid.start_date = NEW.start_date
        AND bid.end_date = NEW.end_date
        AND bid.pet_owner_user_id = NEW.pet_owner_user_id
        AND bid.pet_name = NEW.pet_name;
ELSE
UPDATE bid
SET total_price = base_price * number_days,
        is_success = TRUE
WHERE bid.care_taker_user_id = NEW.care_taker_user_id
        AND bid.start_date = NEW.start_date
        AND bid.end_date = NEW.end_date
        AND bid.pet_owner_user_id = NEW.pet_owner_user_id
        AND bid.pet_name = NEW.pet_name;
END IF;
ELSE
UPDATE bid
SET total_price = base_price * number_days,
        is_success = FALSE
WHERE bid.care_taker_user_id = NEW.care_taker_user_id
        AND bid.start_date = NEW.start_date
        AND bid.end_date = NEW.end_date
        AND bid.pet_owner_user_id = NEW.pet_owner_user_id
        AND bid.pet_name = NEW.pet_name;
END IF;
RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;
DROP TRIGGER IF EXISTS bid_full_time_care_taker_auto_confirm_trigger ON Bid CASCADE;
CREATE TRIGGER bid_full_time_care_taker_auto_confirm_trigger
```

```
AFTER
INSERT ON Bid FOR EACH ROW EXECUTE FUNCTION bid_full_time_care_taker_auto_confirm();
```

This trigger checks if the caretaker participating in the bid is available to take care of the pet owner's pet from start date to end date, and is fired when a bid is inserted into the table.

```
CREATE OR REPLACE FUNCTION bid_care_taker_is_available_to_insert() RETURNS TRIGGER AS
$$
DECLARE available_days INTEGER;
DECLARE total_days INTEGER;
BEGIN
SELECT COUNT(*),
        NEW.end_date - NEW.start_date + 1 INTO available_days,
        total_days
FROM IsAvailableOnWithPetCareCount a,
        CareTakersWithPetLimitAndRating c
WHERE NEW.care_taker_user_id = a.care_taker_user_id
        AND NEW.care_taker_user_id = c.user_id
        AND a.available_date >= NEW.start_date
        AND a.available_date <= NEW.end_date
        AND a.pet_care_count < c.pet_limit;
IF available_days <> total_days THEN RAISE EXCEPTION '% cannot take care of the pet between %
and %',
        (SELECT username
        FROM Users
        WHERE NEW.care_taker_user_id = Users.user_id),
NEW.start_date,
NEW.end_date;
END IF;
RETURN NEW;
END;
$$ LANGUAGE PLPGSQL;
DROP TRIGGER IF EXISTS bid_care_taker_is_available_to_insert_trigger on Bid CASCADE;
CREATE TRIGGER bid_care_taker_is_available_to_insert_trigger BEFORE
INSERT ON Bid FOR EACH ROW EXECUTE FUNCTION bid_care_taker_is_available_to_insert();
```

This trigger checks if a full time caretaker is allowed to take a leave on a certain day by ensuring he meets his requirements of working 2 x 150 days after the day he is available to work is deleted. This trigger runs when a row in IsAvailableOn is deleted.

```
CREATE OR REPLACE FUNCTION full_time_care_taker_block_leave () RETURNS TRIGGER AS $$
DECLARE number_work_blocks INTEGER := 0;
DECLARE number_consecutive_work_days INTEGER := 0;
DECLARE highest_consecutive_work_days INTEGER := 0;
DECLARE current_day DATE := MAKE_DATE (
        CAST(EXTRACT(YEAR FROM OLD.available_date) AS INTEGER), 1, 1);
DECLARE last_day_of_year DATE := MAKE_DATE (
        CAST(EXTRACT(YEAR FROM OLD.available_date) AS INTEGER), 12, 31);
BEGIN WHILE current_day <= last_day_of_year LOOP IF EXISTS (
            SELECT 1
            FROM IsAvailableOn iao
            WHERE iao.care_taker_user_id = OLD.care_taker_user_id
            AND iao.available_date = current_day
) THEN number_consecutive_work_days := number_consecutive_work_days + 1;
ELSEIF number_consecutive_work_days >= (
            SELECT minimum_work_days_in_block
            FROM Admin LIMIT 1
) THEN number_work_blocks := number_work_blocks + 1;
IF number_consecutive_work_days > highest_consecutive_work_days THEN
highest_consecutive_work_days := number_consecutive_work_days;
END IF;
number_consecutive_work_days := 0;
ELSE number_consecutive_work_days := 0;
END IF;
current_day := current_day + 1;
END LOOP;
IF number_consecutive_work_days >= (
            SELECT minimum_work_days_in_block
            FROM Admin
            LIMIT 1
) THEN number_work_blocks := number_work_blocks + 1;
IF number_consecutive_work_days > highest_consecutive_work_days THEN
highest_consecutive_work_days := number_consecutive_work_days;
END IF;END IF;
IF (SELECT is_full_time
            FROM CareTakers c
            WHERE c.user_id = OLD.care_taker_user_id)
AND number_work_blocks < (
            SELECT minimum_work_blocks
            FROM Admin
            LIMIT 1)
AND highest_consecutive_work_days < (
            SELECT minimum_work_days_in_block
            FROM Admin
            LIMIT 1) *
            (SELECT minimum_work_blocks FROM Admin LIMIT 1
) THEN RAISE EXCEPTION '% cannot take leave on % because it violates the minimum consecutive
days in a year',(
            SELECT username
            FROM Users
            WHERE OLD.care_taker_user_id = Users.user_id),
OLD.available_date;
END IF;
RETURN OLD;
END;
$$ LANGUAGE PLPGSQL;
DROP TRIGGER IF EXISTS full_time_care_taker_block_leave_trigger ON IsAvailableOn CASCADE;
CREATE TRIGGER full_time_care_taker_block_leave_trigger
AFTER DELETE ON IsAvailableOn FOR EACH ROW EXECUTE FUNCTION
full_time_care_taker_block_leave();
```

Interesting and Complex queries and views

Search Availability

Retrieves the available Caretakers, given optional parameters by the user, utilising the COALESCE command to perform input validation on null values input by the user at the server side. This also makes use of an outer join on caretakers with no rating yet given to

```
SELECT DISTINCT x.named, x.contact, x.price, x.category,
    ROUND(AVG(y.avg_rating), 2) as rating
    FROM (SELECT DISTINCT u.user_id as userid,
    u.name as named, u.contact_number as contact, c.daily_price as price,
    c.category_name as category, a.available_date as dated
    FROM isAvailableOn a
    JOIN Users u ON a.care_taker_user_id = u.user_id
    JOIN CanTakeCare c ON a.care_taker_user_id = c.care_taker_user_id
    WHERE c.category_name LIKE COALESCE(CAST($1 as VARCHAR), '')||'%'
    AND a.available_date >= COALESCE(CAST($2 AS DATE), DATE('1970-01-01'))
    AND a.available_date <= COALESCE(CAST($3 AS DATE), '9999-12-31')
    AND u.name LIKE COALESCE(CAST($4 as VARCHAR), '')||'%'
    AND c.daily_price <= COALESCE(CAST($5 as INT), 20000)) as x
    LEFT OUTER JOIN (SELECT u.user_id as userid,
    u.name as named, u.contact_number as contact,
    ROUND(AVG(b.rating), 2) as avg_rating
    FROM Bid b INNER JOIN Users u ON b.care_taker_user_id = u.user_id
    GROUP BY (u.user_id, b.care_taker_user_id, u.name, u.contact_number))
    as y ON x.userid = y.userid
GROUP BY (x.userid, x.named, x.contact, x.price, y.userid, y.named, y.contact, x.category) HAVING
AVG(y.avg_rating) >= COALESCE(CAST($6 AS NUMERIC), 0)
    OR AVG(y.avg_rating) IS NULL
    AND ((CAST($2 AS DATE) IS NULL OR CAST($3 AS DATE) IS NULL)
    OR COUNT(DISTINCT x.dated) = CAST($3 AS DATE)- CAST($2 AS DATE) + 1)
```

Get all Caretakers Salary

Retrieve earning of all caretakers given a start date and end date. The query is recursive, and partitions the pet-days by decomposing them to actual pet days, then enumerating them to partition at 60 Days. Then separating out the post 60 day earnings to be used for the full-time salary computations.

```
with RECURSIVE t AS (
   SELECT care_taker_user_id, start_date, end_date, 1 AS index,
   (end_date - start_date + 1) AS date_count,
   (total_price/A.date_count) AS daily_price from
```

```
    (SELECT care_taker_user_id, start_date, end_date,
    (end_date - start_date + 1) AS date_count, total_price from bid) AS A
     WHERE start_date >= (SELECT COALESCE(CAST($1 AS DATE), DATE('1970-01-01')))
     AND start_date <= (SELECT COALESCE(CAST($2 AS DATE), DATE('9999-12-31') ))
     UNION ALL
     SELECT care_taker_user_id, start_date, end_date, index + 1, date_count,
     daily_price FROM t WHERE index < date_count)
     SELECT care_taker_user_id, count(*) AS pet_day, sum(t.daily_price) AS total_earnings,
     (SELECT sum(daily_price) FROM (SELECT *, rank() OVER (
      PARTITION BY care_taker_user_id
      ORDER BY care_taker_user_id, start_date, end_date, index ASC)
      FROM t) AS t2
      WHERE rank > 60 AND t.care_taker_user_id = t2.care_taker_user_id
     ) AS post_60_days_earnings, caretakers.is_full_time,
    (CASE WHEN caretakers.base_salary IS NULL AND caretakers.is_full_time = false
       THEN sum(t.daily_price) * caretakers.commission_rate
        WHEN count(*) > 60 THEN (SELECT sum(daily_price)
    FROM (SELECT *, rank() OVER (PARTITION BY care_taker_user_id
      ORDER BY care_taker_user_id, start_date, end_date, index ASC)
      FROM t) AS t2
    WHERE rank > 60 AND t.care_taker_user_id = t2.care_taker_user_id
    ) * caretakers.commission_rate + caretakers.base_salary
     ELSE caretakers.base_salary END) as salary
   FROM t JOIN caretakers ON t.care_taker_user_id = caretakers.user_id
   GROUP BY t.care_taker_user_id, caretakers.is_full_time, caretakers.base_salary,
caretakers.commission_rate;
```

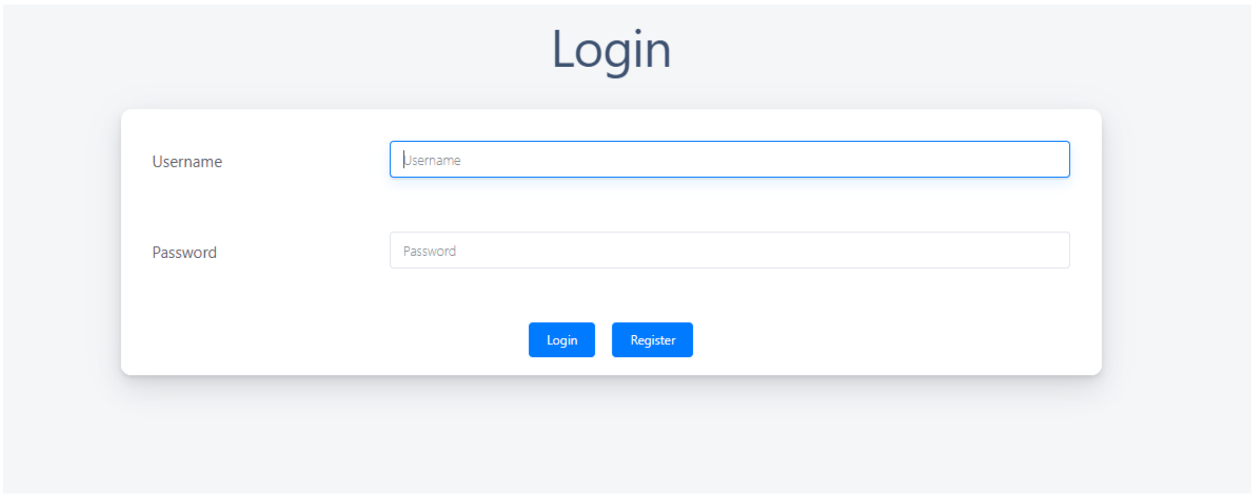
View Caretakers with their pet limits and rating

This view is used in the trigger bit_care_taker_is_available_to_insert to enforce the pet limit constraint in the backend updates of the PCS. The query uses cases on the rating to set the pet_limit for a Part-time Caretaker, as well as a groupby aggregation to project the rating.

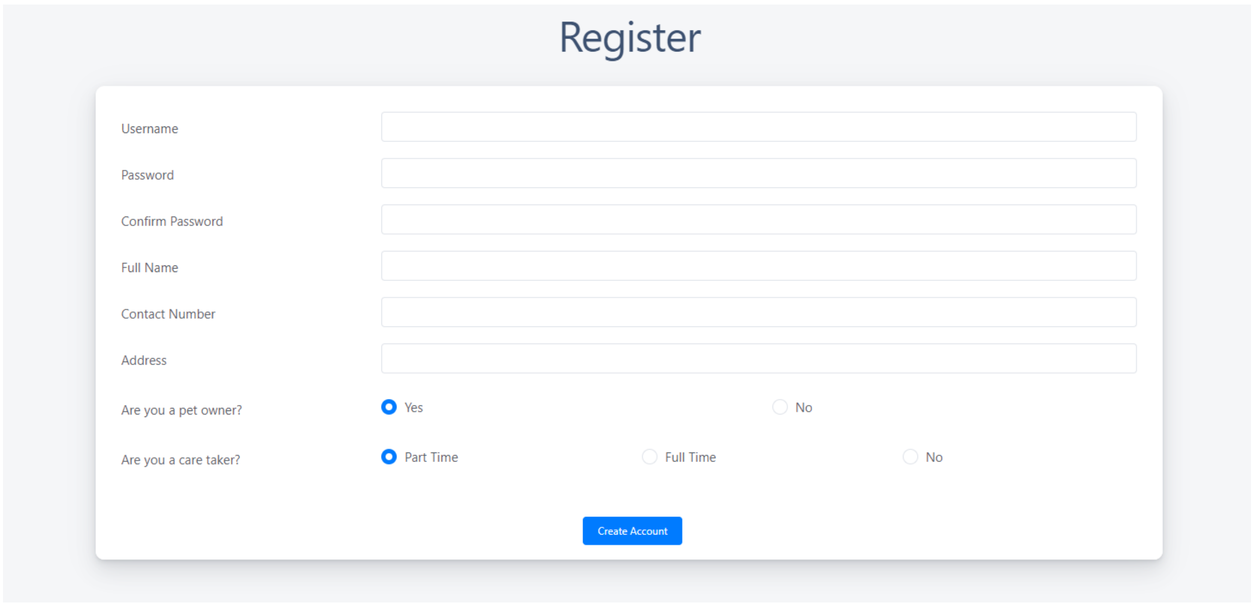
```
CREATE OR REPLACE VIEW CareTakersWithPetLimitAndRating AS (
        SELECT c.user_id AS user_id,
            c.bonus_rate AS bonus_rate,
            c.base_salary AS base_salary,
            c.minimum_base_quota AS minimum_base_quota,
            c.commission_rate AS commission_rate,
            c.is_full_time AS is_full_time,
            AVG(b.rating) AS rating,
            (CASE WHEN (c.is_full_time OR AVG(b.rating) >= 4)
                    THEN a.pet_limit
                    ELSE a.poor_review_pet_limit END) AS pet_limit
        FROM CareTakers c
            LEFT JOIN Bid b ON c.user_id = b.care_taker_user_id, Admin a
        GROUP BY c.user_id, a.pet_limit, a.poor_review_pet_limit);
```

User Interface

- Login Page



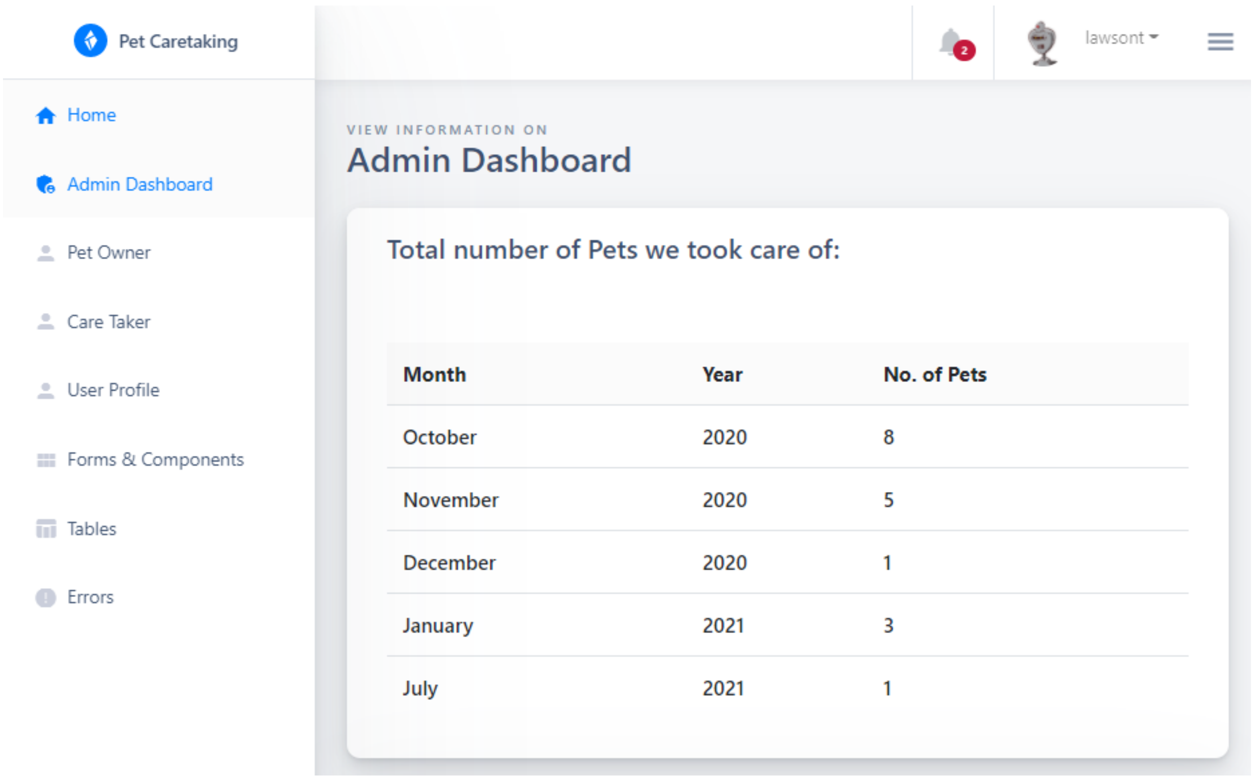
- Register as a new user



Are you a pet owner? [✓ Yes] No

Are you a care taker? [✓ Part Time] Full Time No

- Admin Dashboard View

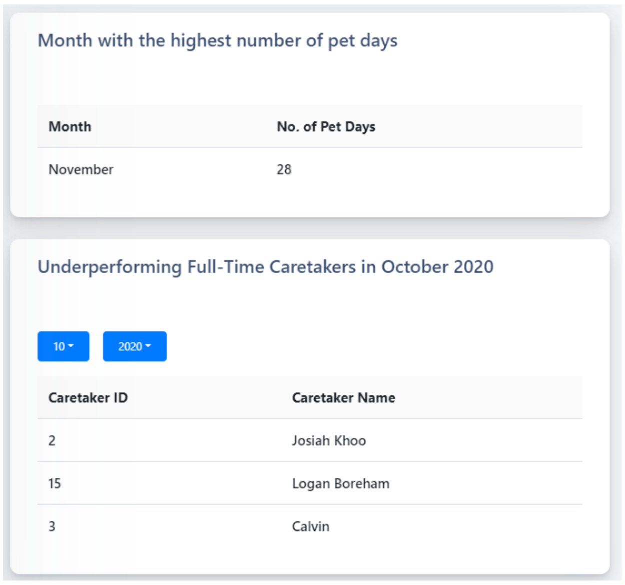


VIEW INFORMATION ON

Admin Dashboard

Total number of Pets we took care of:

Month	Year	No. of Pets
October	2020	8
November	2020	5
December	2020	1
January	2021	3
July	2021	1

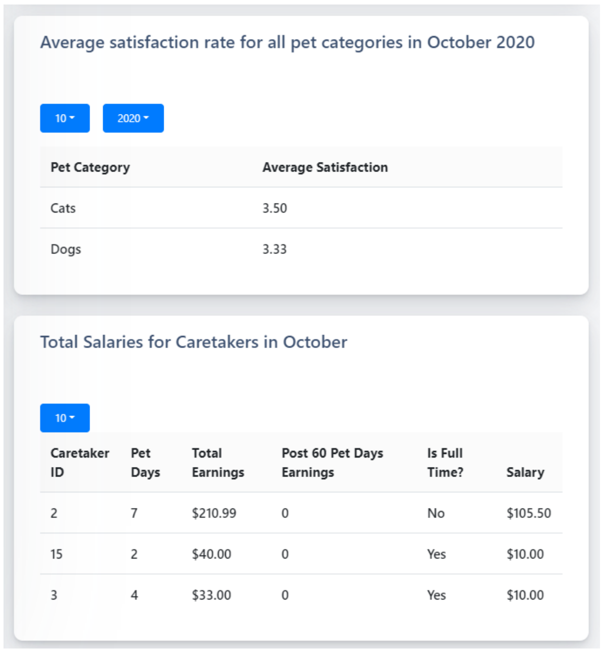


Month with the highest number of pet days

Month	No. of Pet Days
November	28

Underperforming Full-Time Caretakers in October 2020

Caretaker ID	Caretaker Name
2	Josiah Khoo
15	Logan Boreham
3	Calvin



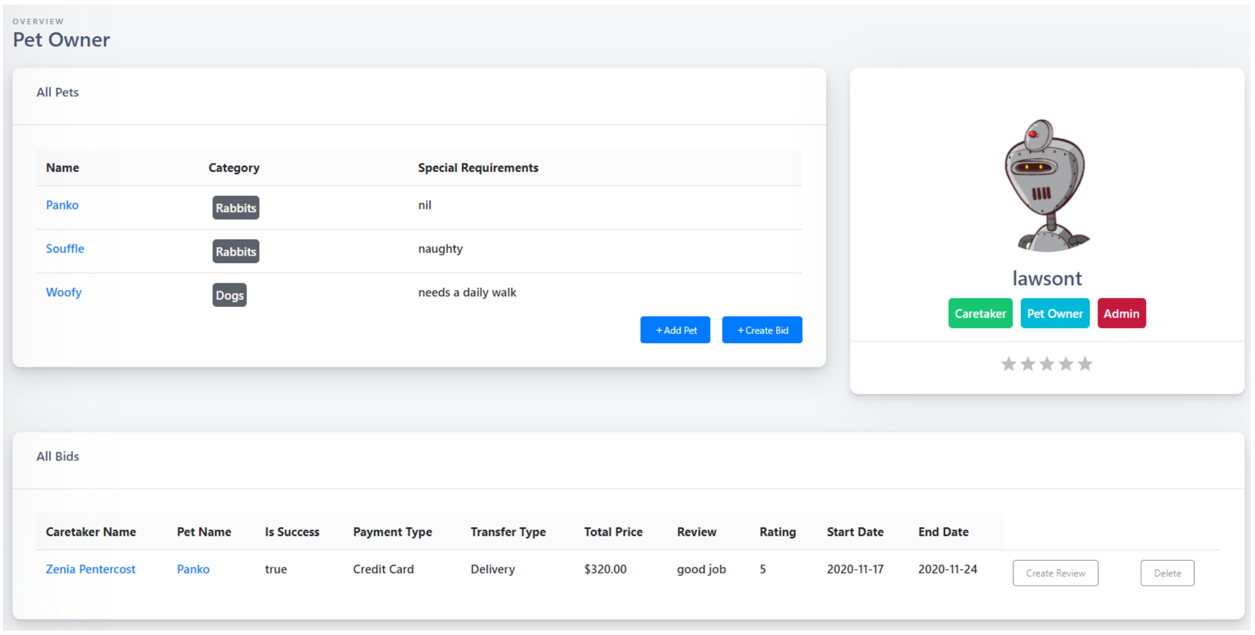
Average satisfaction rate for all pet categories in October 2020

Pet Category	Average Satisfaction
Cats	3.50
Dogs	3.33

Total Salaries for Caretakers in October

Caretaker ID	Pet Days	Total Earnings	Post 60 Pet Days Earnings	Is Full Time?	Salary
2	7	$210.99	0	No	$105.50
15	2	$40.00	0	Yes	$10.00
3	4	$33.00	0	Yes	$10.00

- Pet Owner Views


OVERVIEW

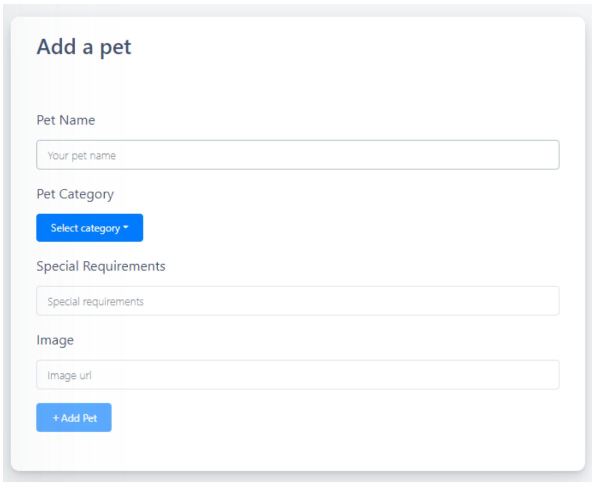
Pet Owner

All Pets

Name	Category	Special Requirements
Panko	Rabbits	nil
Souffle	Rabbits	naughty
Woofy	Dogs	needs a daily walk

All Bids

Caretaker Name	Pet Name	Is Success	Payment Type	Transfer Type	Total Price	Review	Rating	Start Date	End Date		
Zenia Pentercost	Panko	true	Credit Card	Delivery	$320.00	good job	5	2020-11-17	2020-11-24	Create Review	Delete


Add a pet

Pet Name

Your pet name

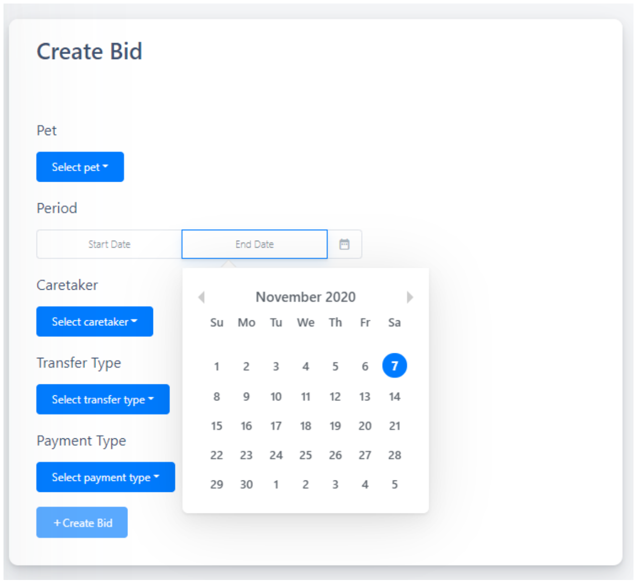
Pet Category

Special Requirements

Special requirements

Image

Image url


Create Bid

Pet

Period

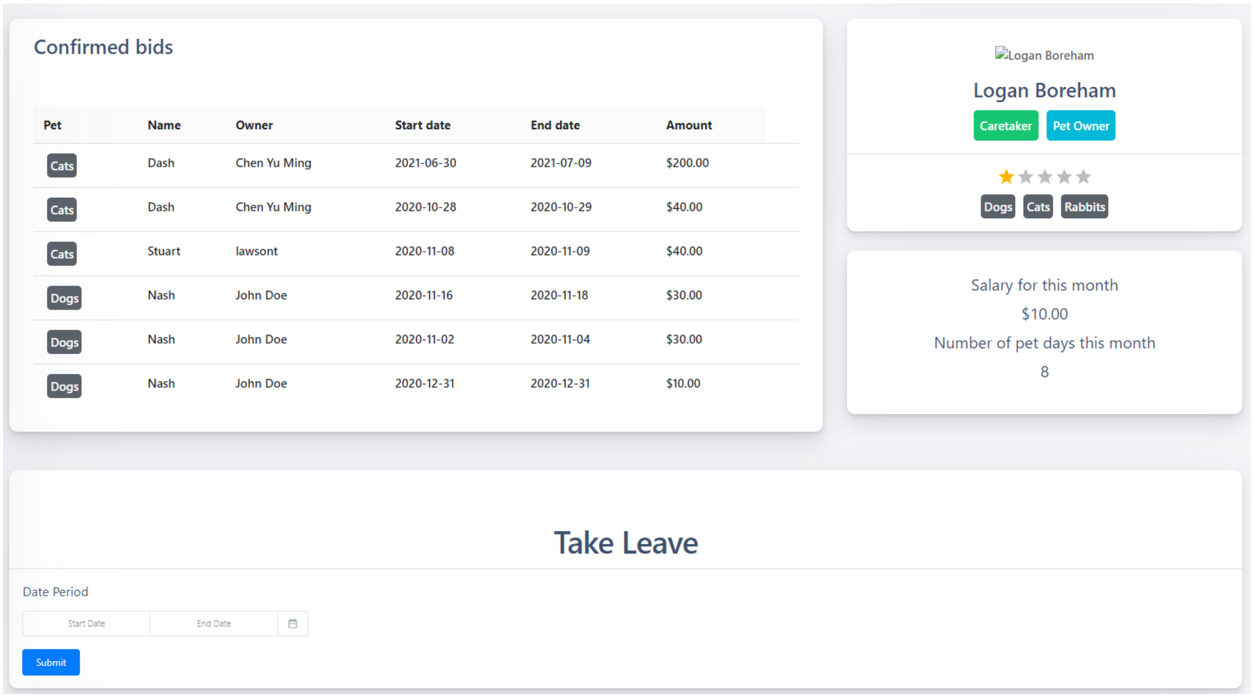
Start Date End Date

Caretaker

Transfer Type

Payment Type

- Caretaker View



Confirmed bids

Pet	Name	Owner	Start date	End date	Amount
Cats	Dash	Chen Yu Ming	2021-06-30	2021-07-09	$200.00
Cats	Dash	Chen Yu Ming	2020-10-28	2020-10-29	$40.00
Cats	Stuart	lawsont	2020-11-08	2020-11-09	$40.00
Dogs	Nash	John Doe	2020-11-16	2020-11-18	$30.00
Dogs	Nash	John Doe	2020-11-02	2020-11-04	$30.00
Dogs	Nash	John Doe	2020-12-31	2020-12-31	$10.00

Logan Boreham

Caretaker Pet Owner

Dogs Cats Rabbits

Salary for this month

$10.00

Number of pet days this month

8

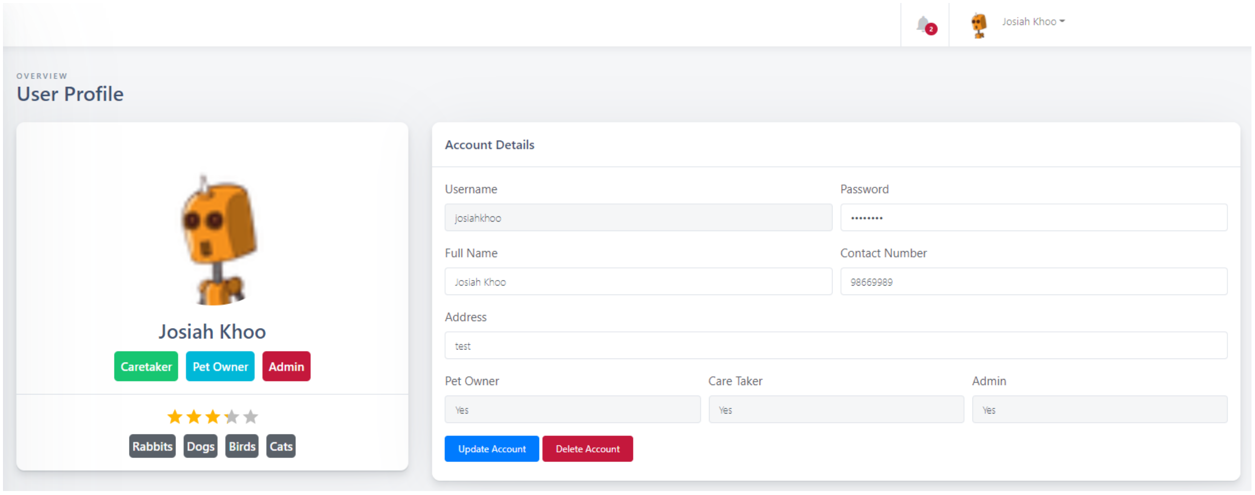
Take Leave

Date Period

Start Date End Date

Submit

- User Profile



Josiah Khoo

OVERVIEW

User Profile

Josiah Khoo

Caretaker Pet Owner Admin

Rabbits Dogs Birds Cats

Account Details

Username	Password
josiahkhoo	·········
Full Name	Contact Number
Josiah Khoo	98669989

Address

test

Pet Owner	Care Taker	Admin
Yes	Yes	Yes

Update Account Delete Account

Challenges and Lessons

During the early phase of the project, we spent the majority of our time working on the ER diagram. We repeatedly made several refinements to the diagram before the consultation, as we kept discovering flaws while discussing. After the consultation, we had to make multiple changes to our diagram as there were still things we had missed. Even during development, minor changes were continuously made due to emergent issues and complications. This was perhaps caused by our inexperience in designing database systems.

The development of this application has given us valuable and insightful first-hand experience in designing database systems and integrating it in a web app with back and front end. For example, we learned to successfully apply the querying techniques taught throughout the module to pull information from the database, and used a JSON API framework to pass data to the front-end.

In addition, for many of the constraints in the ER diagram, it was difficult to translate them to SQL simply using table constraints. We had to use triggers and had 9 total triggers to make sure our system functioned the way we wanted it to. As triggers were not gone through in depth during the lectures, we had to spend time figuring out how to write triggers that applied to our use case and testing the trigger to make sure it worked.

Besides the technical skills that we picked up from working on this project, we managed to practise soft skills such as communication and teamwork. The COVID-19 issue impacted this module with all communication being carried out over virtual communication like Zoom or Discord. This made it less productive when we were discussing, and distributing the workload. Furthermore, each of us are taking different modules and some modules were affected in similar ways, causing scheduling problems. Hence, we had to make some adjustments to accommodate each other’s hectic schedules. As the workload of this project is relatively high, and we are also taking other modules with group projects, we are fortunate we could complete the requirements for this project in time through fortitude and determination.