PetSOS: Final Project Report

CS2102

Team 28

A0200739J	Bennett Clement
A0200700H	Christian Drake Martin
A0200765L	Erin May Gunawan
A0200760W	Florencia Martina
A0167754R	Quek Shui Herng

Table of Contents

I. Project Responsibilities

1. Bennett Clement: Frontend (Caretaker Profile) + Backend (users, auth system, API management) + DevOps (Deploy to Heroku, PR review, system setup)
2. Christian Drake Martin: Frontend (Dashboard, Pets, some part of admin page) + Backend (Dashboard, Pets, some part of admin page)
3. Erin May Gunawan: Frontend (Landing, Search Caretaker, Edit Profile, major UI touch up) + Backend (Caretakers)
4. Florencia Martina: Frontend (Signup, Login, Become Caretaker, Search Caretaker, Reviews) + Backend (Caretakers + Reviews)
5. Quek Shui Herng: Frontend + Backend (Jobs/Bids). Report Formatting.

II. Functionalities and Data Constraints

1. Terminologies

We use “jobs” to refer to an accepted bid, and “bid” otherwise.

2. Description

PetSOS is a pet-sitting service where pet owners may engage the help of caretakers to look after their pets for some time. Pet owners can register their pets at the PetSOS portal, and subsequently bid for caretakers’ services, which will subsequently be accepted. Owners can view past ratings and reviews of a caretaker before placing bids, and details of each bid (payment and pet transfer method) are discussed before a bid is accepted.

3. Application Workflow

New users (both pet owners and caretakers) have to register at the PetSOS portal, where they can subsequently add their pets to be taken care of, and/or register as a caretaker. Caretakers have to indicate if they will be part-time or full-time, and indicate dates where they will be available to work. Caretakers will also have to define their capabilities in the form of the types of pets they are capable of looking after.

Pet owners can search for suitable caretakers on the portal, by looking for the dates where they require the pet-sitting service, as well as the capabilities of the caretaker. Pet owners can see the caretaker’s average rating and browse past reviews given by other pet owners for the caretaker’s service in the past. Once a suitable one has been found, they may place a bid, where the payment and transfer method is determined. Caretakers may subsequently accept the bids if it falls within their working restrictions.

In every user’s profile, they can find a list of bids and jobs. Placed bids may be removed by a pet owner if they have not been accepted by a caretaker.

After the pet-sitting period, pet owners can rate and give review to the caretaker’s service.

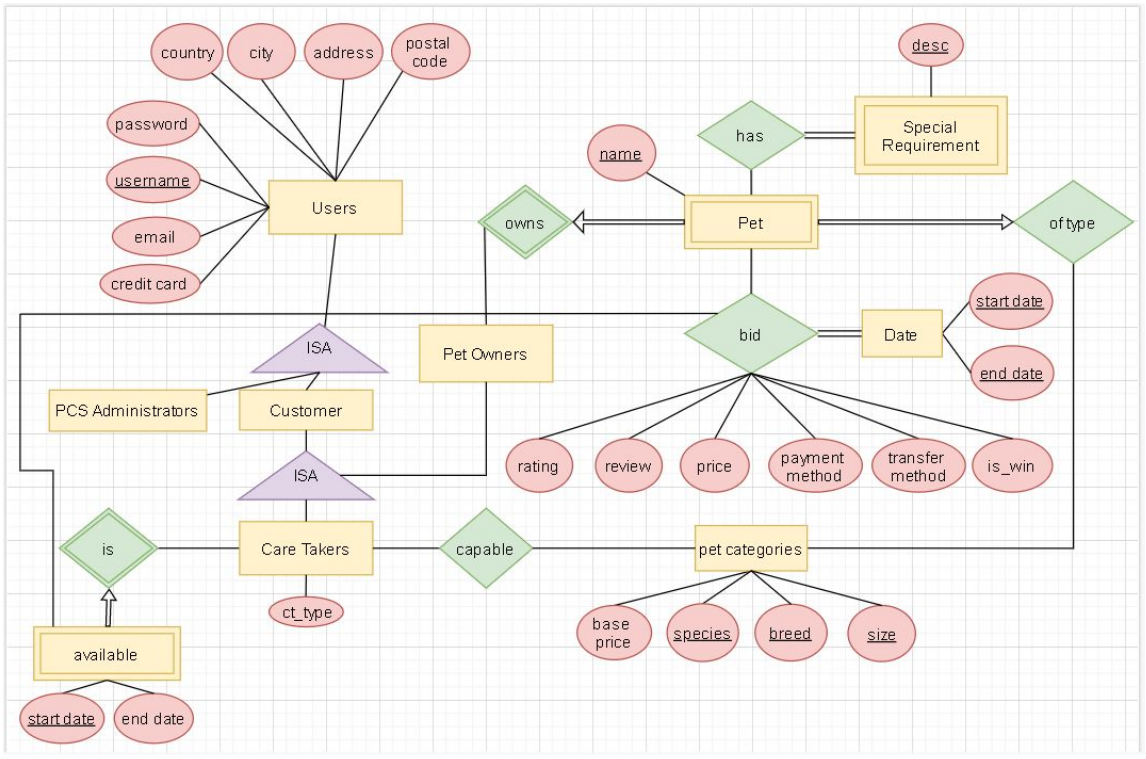
4. Data Constraints

- Every user is a pet owner.
- Usernames uniquely identify a user, and cannot be changed upon registering. User emails must be unique.
- All pets owned by a user must have a different name. Pets owned by different users can have the same name. Each pet may have 0 or more special requirements.
- A caretaker can either be a full time caretaker or a part time caretaker
- All bids and caretaker availability date ranges used in the application comprise a start and end date, and are inclusive of those dates.
- A caretaker may declare their availability for dates up to two years from the current date.
- Multiple sizes of the same breed may exist under the same species. For example, the species ‘dog’ may have breed ‘corgi’s which are of size ‘small’ and ‘medium’.
- Pet size are restricted to ‘small’, ‘medium’, ‘large’
- The base price for each pet category must be positive integer
- Capabilities of a caretaker are defined by their ability to take care of a specific (species, breed, size). In other words, a caretaker may be able to take care of a medium-sized corgi, but be unable to take care of a small one.
- Bids for taking care of a pet must have a start and end date, with start date being before the end date. As discussed above, this date range is inclusive of the start and end.
- Ratings for a caretaker are of a floating point format in the domain [0, 5].
- Review and ratings are per-job. Each job has at most one review and at most one rating. Multiple reviews and ratings may exist for the same pet and caretaker if the caretaker has accepted jobs for the pet on multiple occasions.

5. Other application requirements:

- Full time caretakers receive a salary of $3000 per month for up to 60 pet days. For any excess pet-day they will receive 80% of the price as bonus.
- Part time caretakers receive 75% of the job price for each job. PCS receive the remaining 25% as commission.
- The global base price of a pet category is determined by the pet care system administrator.
- Pet owners pay for the amount required to care for their pet upfront.

III. ER Diagram



Constraints not captured in ER data model:

- A pet may only be under the care of a single caretaker at any time. (ER diagram can only enforce not the same <start_date, end_date>)
- Caretakers cannot take care of pets where their type do not fall under the caretakers capabilities.
- Pet owners who are also caretakers cannot bid for their own service.
- A caretaker may only take on up to five concurrent jobs if their average rating (amongst jobs for which a rating has been issued) exceed 4. Otherwise, they are limited to a maximum of two.
- A caretaker who has no ratings yet (new caretaker, or has completed jobs but has not received a rating) is assumed to have a rating <= 4, and thus are limited to a maximum of two concurrent jobs.
- A caretaker may declare their availability for dates up to two years from the current date.
- A full time caretaker's availability span must be minimally 150 days apart, and should satisfy 2 x 150 days of availability in a year
- Price of a bid depends on the base price of a pet category, as well as the duration of care required and increases with the average rating of the caretaker.

- Pet owners may only post a review and rating after a bid is won and the job is completed.

IV. Relational Schema

```
-- We omit DROP TYPE, DROP TABLE, DROP VIEW statements to simplify the
report

CREATE TYPE caretaker_type AS ENUM ('part-time', 'full-time');
CREATE TYPE pet_size AS ENUM ('small', 'medium', 'large');

CREATE TABLE users (
  username VARCHAR PRIMARY KEY,
  email VARCHAR UNIQUE NOT NULL,
  password VARCHAR NOT NULL,
  address VARCHAR,
  city VARCHAR,
  country VARCHAR,
  postal_code INTEGER,
  credit_card VARCHAR(16)
);

CREATE TABLE pcs_admin (
   username VARCHAR PRIMARY KEY REFERENCES users(username)
    ON DELETE CASCADE
);

CREATE TABLE caretakers (
  ctuname VARCHAR PRIMARY KEY REFERENCES users(username)
    ON DELETE CASCADE,
  ct_type caretaker_type NOT NULL
);

CREATE TABLE availability_span (
  ctuname VARCHAR REFERENCES caretakers(ctuname) ON DELETE CASCADE,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  PRIMARY KEY (ctuname, start_date, end_date),
  CHECK (start_date <= end_date AND end_date <= date('now') + interval
'2 years')
);
```

```
CREATE TABLE pet_categories (
  species VARCHAR,
  breed VARCHAR,
  size VARCHAR,
  base_price NUMERIC NOT NULL,
  PRIMARY KEY(species, breed, size),
  CHECK (base_price > 0)
);

CREATE TABLE pets (
  name VARCHAR,
  pouname VARCHAR REFERENCES users(username) ON DELETE CASCADE,
  species VARCHAR NOT NULL,
  breed VARCHAR,
  size VARCHAR NOT NULL,
  FOREIGN KEY (species, breed, size) REFERENCES pet_categories(species,
breed, size),
  PRIMARY KEY (name, pouname)
);

CREATE TABLE special_reqs (
  pouname VARCHAR NOT NULL,
  petname VARCHAR NOT NULL,
  description VARCHAR NOT NULL,
  FOREIGN KEY (petname, pouname)
    REFERENCES pets(name, pouname) ON DELETE CASCADE,
  PRIMARY KEY(pouname, petname, description)
);

CREATE TABLE is_capable (
  species VARCHAR NOT NULL,
  breed VARCHAR NOT NULL,
  size VARCHAR NOT NULL,
  ctuname VARCHAR NOT NULL REFERENCES users(username),
  FOREIGN KEY (species, breed, size)
    REFERENCES pet_categories(species, breed, size),
  PRIMARY KEY (breed, size, species, ctuname)
);

CREATE TABLE bid (
  rating FLOAT(5) CHECK (rating >= 0 AND rating <= 5),
  price NUMERIC(20, 3),
  payment_method payment_method NOT NULL,,
  transfer_method VARCHAR NOT NULL,
  review VARCHAR,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  ctuname VARCHAR NOT NULL REFERENCES users(username),
  pouname VARCHAR NOT NULL,
  petname VARCHAR NOT NULL,
  is_win BOOLEAN NOT NULL DEFAULT false,
  CHECK (ctuname <> pouname),
  CHECK (NOT ((NOT is_win) AND (rating is NOT NULL OR review is NOT
NULL))), -- rating and review can only be inserted if bid is won
  FOREIGN KEY (pouname, petname) REFERENCES pets(pouname, name),
  PRIMARY KEY(pouname, petname, ctuname, start_date, end_date)
);

-- table that stores the multiplier to the base price of a caretaker's
-- service based on avg_rating. avg_rating here indicates the lower
-- bracket to get multiplied by the multiplier
CREATE TABLE multiplier (
      avg_rating FLOAT(5) CHECK (avg_rating >= 0 AND avg_rating <= 5),
      multiplier FLOAT(5),
      PRIMARY KEY (avg_rating, multiplier)
);

CREATE VIEW ratings AS (
    SELECT ctuname, round(CAST(avg(rating) as numeric)) as avg_rating
    FROM bid
    GROUP BY ctuname
);

CREATE VIEW all_ct AS (
      SELECT * FROM
      (caretakers
            NATURAL JOIN (SELECT username AS ctuname, city, country,
postal_code FROM users) AS users
            NATURAL JOIN (SELECT * FROM is_capable NATURAL JOIN
pet_categories) AS cap
            NATURAL JOIN availability_span) AS ct_avail_cap
      NATURAL LEFT JOIN ratings
);
```

Constraints not enforced in SQL schema:

- A pet may only be under the care of a single caretaker at any time. (enforced using trigger)
- A caretaker may only take on up to five concurrent jobs if their average rating (amongst jobs for which a rating has been issued) exceed 4. Otherwise, they are limited to a maximum of two. (enforced using trigger)
- A full time caretaker's availability span must be minimally 150 days apart, and should satisfy 2 x 150 days of availability in a year. (Only the first part can be enforced using trigger)
- Price of a bid depends on the base price of a pet category, as well as the duration of care required and increases with the average rating of the caretaker. (calculated using stored procedure and triggered before insert)

V. Database Properties

The database is in BCNF. Most of the tables have FDs in which the LHS is the primary key and the RHS contains other attributes, and nothing else. The discussion is expanded below.

The table `users` is in BCNF.
The table `pcs_admin` is in BCNF.
The table `caretakers` is in BCNF.
The table `availability_span` is in BCNF.
The table `pet_categories` is in BCNF.
The table `pets` is in BCNF.
The table `special_reqs` is in BCNF.
The table `is_capable` is in BCNF.
The table `bid` is in BCNF. Price depends on the caretaker's average rating, number of days and pet category's base price. `pouname` and `petname` defines `base_ price`, `average_rating` is taken from the `ratings` view, which satisfies `ctuname -> ratings`. Thus the projection of those FDs on `bid` is `{ ctuname, start_date, end_date, pouname, petname } -> { price }`, which satisfies BCNF since the left hand side is a super key.
The table `multiplier` is in BCNF.

VI. Non-Trivial Triggers

```
-- Trigger 1: Ensure caretakers' availabilities are merged to the
largest availability span possible. This ensures that availability spans
stored in the database do not overlap.
```

```
CREATE OR REPLACE FUNCTION merge_availabilities() RETURNS trigger AS
$$
  DECLARE flag INTEGER := 0;
  DECLARE temp_start DATE;
  DECLARE temp_end DATE;
  BEGIN
    DELETE FROM availability_span A
WHERE A.ctuname = NEW.ctuname
  AND A.start_date >= NEW.start_date
  AND A.end_date <= NEW.end_date;
    SELECT SUM(
      CASE
        WHEN A.start_date <= NEW.start_date
    AND A.end_date >= NEW.end_date THEN 4
        WHEN NEW.end_date >= A.start_date
    AND NEW.end_date <= A.end_date THEN 1
        WHEN NEW.start_date <= A.end_date
    AND NEW.start_date >= A.start_date THEN 2
        ELSE 0
      END) INTO flag

      FROM availability_span A
      WHERE A.ctuname = NEW.ctuname;
  -- RAISE NOTICE '%', flag;
    CASE COALESCE(flag, 0)
      WHEN 0 THEN RETURN NEW;
      WHEN 1 THEN
        SELECT end_date INTO temp_end FROM availability_span A
          WHERE NEW.end_date >= A.start_date
AND NEW.end_date <= A.end_date AND NEW.ctuname = A.ctuname;
        DELETE FROM availability_span A
    WHERE end_date = temp_end AND NEW.ctuname = A.ctuname;
        INSERT INTO availability_span(ctuname, start_date, end_date)
    VALUES (NEW.ctuname, NEW.start_date, temp_end);
      WHEN 2 THEN
        SELECT start_date INTO temp_start FROM availability_span A
          WHERE NEW.start_date <= A.end_date
AND NEW.start_date >= A.start_date AND NEW.ctuname = A.ctuname;
        DELETE FROM availability_span A
    WHERE start_date = temp_start AND NEW.ctuname = A.ctuname;
        -- RAISE NOTICE 'deleted start: %', temp_start;
        INSERT INTO availability_span(ctuname, start_date, end_date)
    VALUES (NEW.ctuname, temp_start, NEW.end_date);
      WHEN 3 THEN
        SELECT end_date INTO temp_end FROM availability_span A
          WHERE NEW.end_date >= A.start_date
AND NEW.end_date <= A.end_date AND NEW.ctuname = A.ctuname;
        DELETE FROM availability_span A
    WHERE end_date = temp_end AND NEW.ctuname = A.ctuname;
        -- RAISE NOTICE 'deleted end: %', temp_end;
        SELECT start_date INTO temp_start FROM availability_span A
          WHERE NEW.start_date <= A.end_date
AND NEW.start_date >= A.start_date AND NEW.ctuname = A.ctuname;
        DELETE FROM availability_span A
    WHERE start_date = temp_start AND NEW.ctuname = A.ctuname;
        -- RAISE NOTICE 'deleted start: %', temp_start;
        -- RAISE NOTICE 'deleted end: %', temp_end;
        INSERT INTO availability_span(ctuname, start_date, end_date)
    VALUES (NEW.ctuname, temp_start, temp_end);
      ELSE NULL;
    END CASE;
  RETURN NULL;
  END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER maintain_availability_non_overlapness
  BEFORE INSERT OR UPDATE ON availability_span
  FOR EACH ROW EXECUTE PROCEDURE merge_availabilities();
```

-- Trigger 2: Ensure full time caretaker's availability span is minimally 150 days apart.

```
CREATE OR REPLACE FUNCTION full_time_must_150() RETURNS trigger AS
$$
  DECLARE flag INTEGER := 0;
  BEGIN
    SELECT (
      CASE
        WHEN ct_type = 'full-time' THEN 1
        ELSE 0
      END) INTO flag
      FROM caretakers
      WHERE ctuname = NEW.ctuname;
    IF FLAG = 1 THEN
      IF date(NEW.start_date) + interval '150 days' > date(NEW.end_date)
THEN
        RAISE NOTICE 'RANGE UNDER 150 DAYS % %', date(NEW.start_date) +
interval '150 days', date(NEW.end_date);
        RETURN NULL;
      END IF;
    END IF;
    RETURN NEW;
  END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER full_time_must_150
  BEFORE INSERT OR UPDATE on availability_span
  FOR EACH ROW EXECUTE PROCEDURE full_time_must_150();
```

-- Trigger 3: Enforce that each pet can only be taken care by 1 ct at a time and cannot bid for the daterange that they are being taken care of

```
CREATE OR REPLACE FUNCTION pet_only_1_caretaker_anytime() RETURNS
trigger AS
$$
      DECLARE counter INTEGER := 0;
      BEGIN
            SELECT COUNT(*) into counter
                  FROM bid b
                  WHERE b.petname = NEW.petname AND b.pouname =
NEW.pouname
                  AND (start_date, end_date) overlaps (NEW.start_date,
NEW.end_date)
                  AND is_win = true;
                  -- allows end_date = next start_date.

            IF counter > 0 THEN
                  RAISE EXCEPTION 'There is an ongoing job for the pet
with the given timeframe!';
                  RETURN NULL;
            END IF;
            RETURN NEW;
      END;
$$
```

```
LANGUAGE plpgsql;

CREATE TRIGGER pet_only_1_caretaker_anytime
      BEFORE INSERT ON bid
      FOR EACH ROW EXECUTE PROCEDURE pet_only_1_caretaker_anytime();
```

VII. Complex Queries

-- Query 1: Query for job overlaps depending on the number of concurrent jobs they can take (based on a caretaker's average rating). Returns jobs where an additional insertion at (start date, end date) will cause concurrent job limits to be exceeded for this caretaker. First part of the query is a CTE to lookup all jobs which intersect the given start date and end date ($2, $3) for a caretaker $1. The second part performs an intersection check, with the intersection done on 2 or 5 of the generated table filtered_bid depending on the average rating of the caretaker. The last part of the query enforces a unique constraint on intersection. Note: Performing * on two dateranges returns their intersection, or NULL if they do not intersect.

```
WITH filtered_bid (price, payment_method, transfer_method, start_date,
end_date, ctuname, pouname, petname) AS (
      SELECT b.price, b.payment_method, b.transfer_method,
b.start_date, b.end_date, b.ctuname, b.pouname, b.petname
      FROM bid b
      WHERE b.ctuname = $1 and b.is_win AND not isempty(daterange($2,
$3, '[]') * daterange(b.start_date, b.end_date, '[]')))
      SELECT DISTINCT fb1.price, fb1.payment_method,
fb1.transfer_method, fb1.start_date, fb1.end_date, fb1.ctuname,
fb1.pouname, fb1.petname
      FROM filtered_bid fb1, filtered_bid fb2, filtered_bid fb3,
filtered_bid fb4, filtered_bid fb5\
      WHERE NOT isempty(daterange(fb1.start_date, fb1.end_date, '[]')
      * daterange(fb2.start_date, fb2.end_date, '[]')
      * (CASE WHEN (SELECT AVG(rating) FROM bid WHERE is_win AND
ctuname = $1) IS NULL OR\
            (SELECT AVG(rating) FROM bid WHERE is_win AND ctuname = $1)
< 4\
      THEN daterange('-infinity', 'infinity')
      ELSE daterange(fb3.start_date, fb3.end_date, '[]')
      * daterange(fb4.start_date, fb4.end_date, '[]')
      * daterange(fb5.start_date, fb5.end_date, '[]') END)
      * daterange($2, $3, '[]'))
      AND (CASE WHEN (SELECT AVG(rating) FROM bid WHERE is_win AND
ctuname = 'po1') IS NULL OR (SELECT AVG(rating) FROM bid WHERE is_win
AND ctuname = 'po1') < 4 THEN (fb1.pouname <> fb2.pouname OR
fb1.petname <> fb2.petname OR fb1.start_date <> fb2.start_date OR
fb1.end_date <> fb2.end_date) ELSE (
      (fb1.pouname <> fb2.pouname OR fb1.petname <> fb2.petname OR
fb1.start_date <> fb2.start_date OR fb1.end_date <> fb2.end_date) AND
      (fb1.pouname <> fb3.pouname OR fb1.petname <> fb3.petname OR
fb1.start_date <> fb3.start_date OR fb1.end_date <> fb3.end_date) AND
      (fb1.pouname <> fb4.pouname OR fb1.petname <> fb4.petname OR
fb1.start_date <> fb4.start_date OR fb1.end_date <> fb4.end_date) AND
      (fb1.pouname <> fb5.pouname OR fb1.petname <> fb5.petname OR
fb1.start_date <> fb5.start_date OR fb1.end_date <> fb5.end_date) AND
      (fb2.pouname <> fb3.pouname OR fb2.petname <> fb3.petname OR
fb2.start_date <> fb3.start_date OR fb2.end_date <> fb3.end_date) AND
      (fb2.pouname <> fb4.pouname OR fb2.petname <> fb4.petname OR
fb2.start_date <> fb4.start_date OR fb2.end_date <> fb4.end_date) AND
      (fb2.pouname <> fb5.pouname OR fb2.petname <> fb5.petname OR
fb2.start_date <> fb5.start_date OR fb2.end_date <> fb5.end_date) AND
      (fb3.pouname <> fb4.pouname OR fb3.petname <> fb4.petname OR
fb3.start_date <> fb4.start_date OR fb3.end_date <> fb4.end_date) AND
      (fb3.pouname <> fb5.pouname OR fb3.petname <> fb5.petname OR
fb3.start_date <> fb5.start_date OR fb3.end_date <> fb5.end_date) AND
      (fb4.pouname <> fb5.pouname OR fb4.petname <> fb5.petname OR
fb4.start_date <> fb5.start_date OR fb4.end_date <> fb5.end_date)
      ) END);
```

-- Query 2: Get the payout, number of pet days, and number of (different) jobs taken for the past month for a requested caretaker

```
SELECT *, CASE
      WHEN ct_type = 'full-time' THEN
      (3000 + raw_payout / pet_days * GREATEST(0, pet_days - 60) * 0.8)
      ELSE (raw_payout * 0.75) END AS total_payout
FROM (
      SELECT ctuname, SUM(price) as total_payout, COUNT(*) as num_jobs,
SUM(end_date - start_date + 1) as pet_days, to_char(start_date, 'Mon')
as mon, extract(year from start_date) as yyyy
      FROM bid WHERE is_win = true AND end_date <= date('now') AND
start_date >= date('now') - interval '1 month'
      GROUP BY ctuname, mon, yyyy
      HAVING ctuname = $1
) AS t NATURAL JOIN caretakers;
```

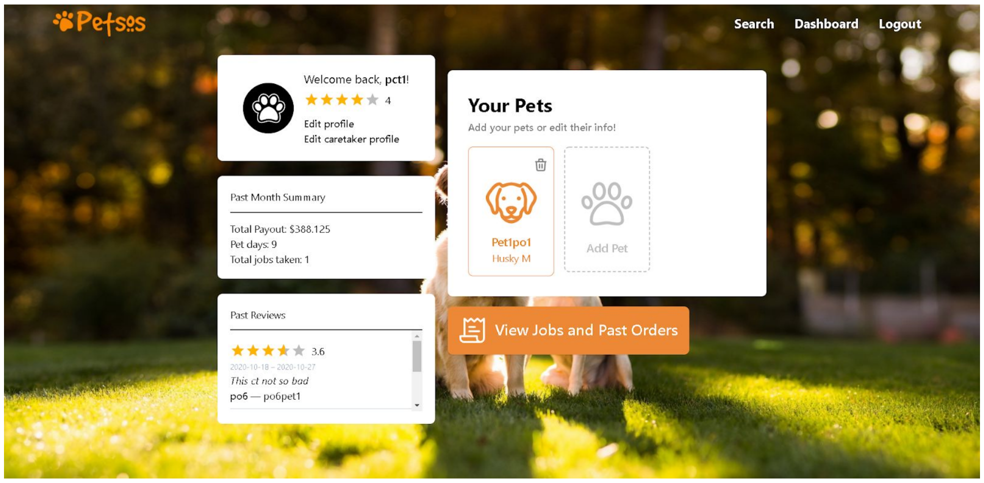
-- Query 3: Search a caretaker based on pet type and availability. The query returns for each matching caretaker: personal details, average rating, and price for the specific pet category requested (which increases with higher average rating)

```
SELECT ctuname, ct_type, city, country, postal_code, avg_rating,
base_price * COALESCE((SELECT multiplier FROM multiplier WHERE
all_ct.avg_rating >= avg_rating ORDER BY multiplier DESC LIMIT 1), 1)
AS price FROM (
      SELECT * FROM (caretakers
            NATURAL JOIN (SELECT username AS ctuname, city, country,
      postal_code FROM users) AS users
            NATURAL JOIN (SELECT * FROM is_capable NATURAL JOIN
      pet_categories) AS cap
            NATURAL JOIN availability_span
      ) AS ct_avail_cap
      NATURAL LEFT JOIN ratings
) AS all_ct
  WHERE start_date <= $1 AND end_date >= $2
  AND species = $3 AND breed = $4 AND size = $5;
```

VIII. Software Tools / Frameworks

- Frontend: React.js + Tailwind
- Backend: Node.js + (node-pg package to access postgresql)
- Database: PostgreSQL

IX. Screenshots



Welcome back, pct1!
★★★★☆ 4
Edit profile
Edit caretaker profile

Past Month Summary

Total Payout: $388.125
Pet days: 9
Total jobs taken: 1

Past Reviews

★★★★☆ 3.6
2020-10-18 – 2020-10-27
This ct not so bad
po6 — po6pet1

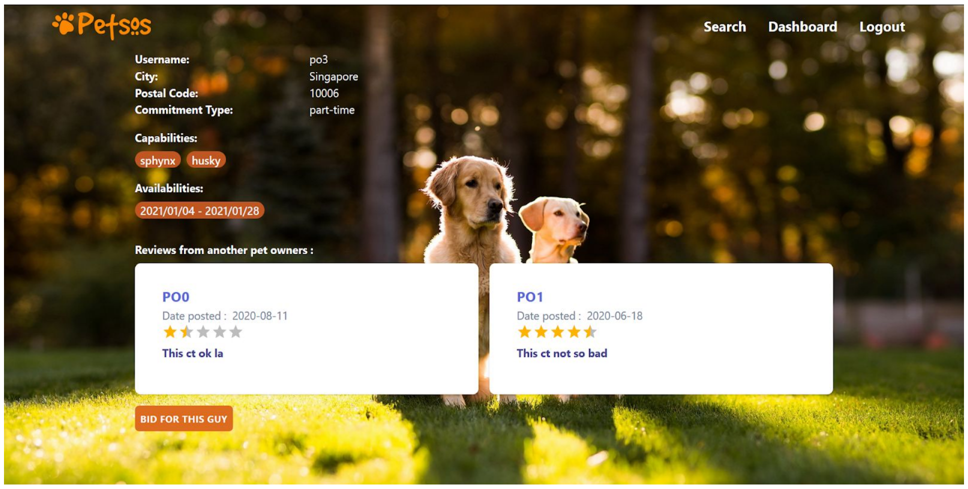
Your Pets
Add your pets or edit their info!

Pet1po1
Husky M

Add Pet

View Jobs and Past Orders

Caretaker Dashboard Page



Username:	po3
City:	Singapore
Postal Code:	10006
Commitment Type:	part-time

Capabilities:
sphynx husky

Availabilities:
2021/01/04 - 2021/01/28

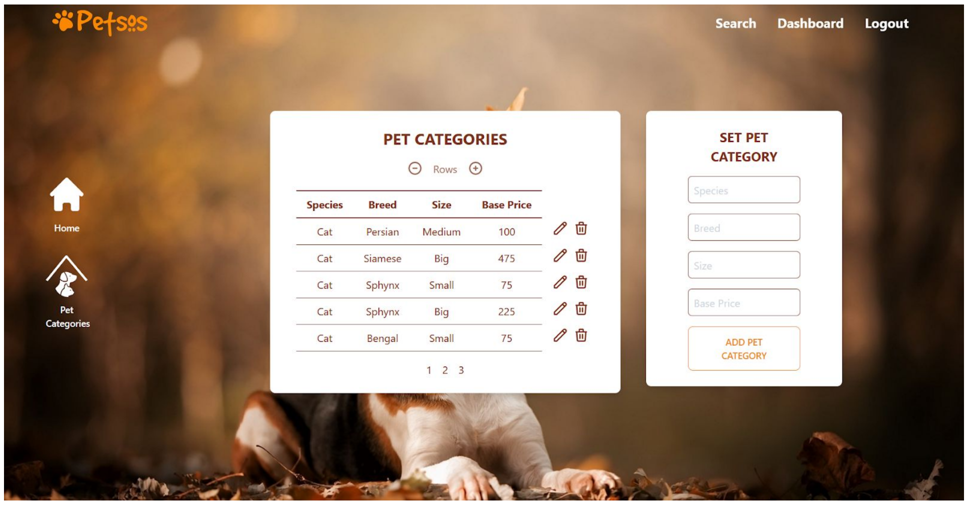
Reviews from another pet owners :

PO0
Date posted : 2020-08-11
This ct ok la

PO1
Date posted : 2020-06-18
This ct not so bad

BID FOR THIS GUY

Search Results page



PET CATEGORIES

Rows

Species	Breed	Size	Base Price
Cat	Persian	Medium	100
Cat	Siamese	Big	475
Cat	Sphynx	Small	75
Cat	Sphynx	Big	225
Cat	Bengal	Small	75

1 2 3

SET PET CATEGORY

Species
Breed
Size
Base Price

ADD PET CATEGORY

Admin page

X. Challenges and Reflections

Almost all members of the team have no experience in web development. We didn't even know how to set up the project. So we had to rely on Google and YouTube to teach us everything. Constructing the schema and ER diagram was challenging since there are many application requirements that need to be fulfilled and despite prof's feedback, we still find it hard to create an ER diagram that is up to this module's standards.

It was also particularly challenging learning SQL, NodeJS, React JS, and CSS in such a short amount of time, even more so putting it into practice. We spent much time debugging and learning how to decipher the error messages. It took us weeks before we were somewhat competent in doing the tasks assigned to us. One of our team members who had experience with web development went out of his way to make our tasks easier by creating functions that make querying and fetching easier. He even took the time to have one-on-one zoom sessions to help us when we were facing a particularly difficult problem.

As one member put it, **"**doing the database is one thing, trying to figure out how to connect the frontend and backend is new and far more confusing".

With that said, this project taught us that it is important to create a robust and well-planned database system because the application relies heavily on how our database is designed. Having us create an ER diagram beforehand admittedly helped us a lot in designing our SQL Schema. Looking back, although the project was really stressful, we gained new skills and experience that will be beneficial for us in the future.