CS2102 AY20/21 Semester 1 Team 1 Project Report (*Pet Society*)

Web Link: https://petsocietyof2102.herokuapp.com/
GitHub Repo: https://github.com/sevenmatt7/CS2102_2021_S1_Team1

Done By:

Full Name	Matric
Matthew Nathanael Sugiri	*A0183805B*
Joshua Tam Wei Ian	*A0190309H*
Tan Guan Yew	*A0183464Y*
Lim Hao Xiang Sean	*A0187123H*
Glen Wong Shu Ze	*A0188100N*

Team’s Responsibilities

Matthew Nathanael Sugiri	Triggers, Integration, API Development, Deployment
Joshua Tam Wei Ian	Frontend, CareTaker Features, API Development
Tan Guan Yew	Frontend, Petowner Features, API Development
Lim Hao Xiang Sean	Admin features, Enquiries features
Glen Wong Shu Ze	Frontend, Backend Salary calculation

Application's functionalities and data constraints

Our web application, Pet Society, allows pet owners to search for caretakers who are able to take care of their pets, and apply for their services. It can be viewed as a 'marketplace' for pet caring services. The features and functionalities each user has access to are listed below.

Functionalities

Pet owners:

- Sign up for an account and register the pets that they want to find a caretaker to care for.
- Bid for a caretaker's services when he/she browses through the caretaker search page for their registered pets
- Search through all the available services offered. They can also the filter the results of their search using a few different parameters (average rating, availability date, area of the caretaker .etc)
- Edit the details of their profile on the profile page after they have signed up (e.g update their address and area, the pets they have .etc)
- Edit pet information, or delete a registered pet
- Submit a review for the caretaker on any completed service/transaction that they were involved in
- View all the past reviews and average rating of the caretaker when they are browsing the services offered
- Submit an enquiry to the administrators of the website in the event of any doubts or disputes
- View the answers to the enquiries that they have previously submitted to the administrator
- View their past and current transactions. They can also filter their transactions based on its status (e.g completed, rejected, .etc)
- Can delete their account if they choose to

Caretakers:

- Sign up for an account as either a full timer or part timer (but not both).
- Edit the details of their profile on the profile page after they have signed up (e.g update their address and area .etc)
- Part timers are free to indicate their availability, there is no minimum number of days they are to be available every month
- Part timers can indicate the daily price for their periods of availability, and choose which bids they want to accept

- Full timers can indicate which pet type they want to care for at registration and will be automatically indicated as available to take care for the types of pets indicated for the whole year in the year they signed up in. (for example, if they sign up today, the availability will be marked to be from 01/01/2020 to 31/12/2020)
- Full timers can take leave by indicating the period they want apply for, and the system will check if they are able to do so
- Full timers are entitled to an increase to the price of their services if they perform well and achieve good ratings from pet owners (elaborated more in the data constraints)
- Accept or reject bids offered by the pet owners for their services
- View the ratings and reviews that the caretakers have given them for the transactions they have been involved in
- View their expected salary for the current month
- View their past and current transactions. They can also filter their transactions based on their status (e.g accepted, rejected, completed .etc)
- Can delete their account if they choose to

Administrators:

- View caretakers who are underperforming (have a rating less than 2)
- View the total number of jobs that have been completed/in progress for the month/year
- See their own commission earnings based on the part-time caretakers that they are managing
- View site statistics (number of pet owners (split up by pet type), number of caretakers, (split up by part time and full time)) so they can make business decisions like which caretakers to advertise to join their site to keep up with the demand from pet owners
- See the distribution of their users islandwide (split up by the five areas, Central, North, Northeast, East and West)
- View the total salary that needs to be paid to the caretakers every month
- View which caretakers they are managing (every caretaker is assigned to an administrator)
- See other relevant statistics in the admin dashboard
- Change the base price of the full time caretakers under their management

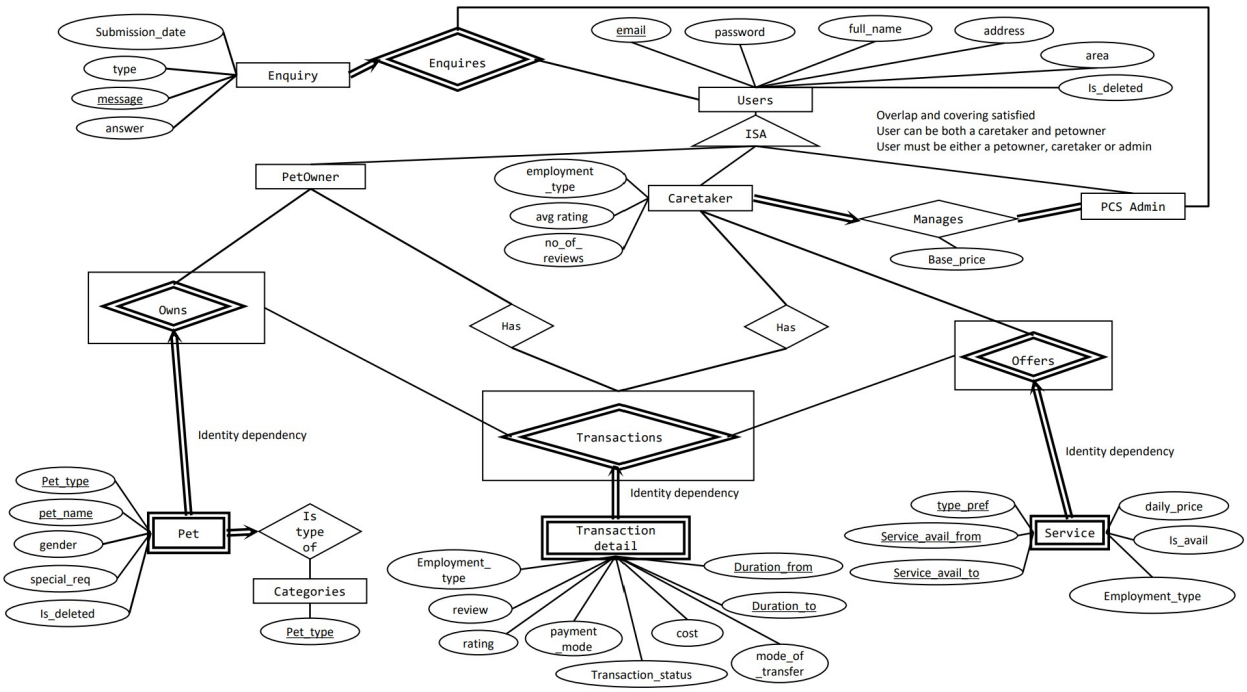
Data Constraints

1. Pet types are classified into categories (Dog, Cat, Fish, Rabbit, Bird, Reptile).
2. A User **can** be either a Pet Owner, Care Taker, both a Pet Owner and a Care Taker, or a PCSAdmin.
3. A Pet Owner **can** own more than one Pet.
4. A Pet Owner **can opt** to make their payment using Cash or Credit-Card.
5. A Pet Owner **can** decide on how to transfer their pet to the Care Taker via 3 methods (Pet owner delivery, Care Taker pick up or Transfer through the physical building of Pet Society).
6. A Pet Owner **cannot** request for a service if their pet does not match the Type Preference of the service offered.
7. A Pet Owner **can only** submit 1 review/rating per transaction.
8. A Pet Owner **can** submit multiple review/rating for a Care Taker if the Care Taker has taken care of the Pet Owner's Pet multiple times, including for the same pet.
9. A Pet Owner **can only** submit a review/rating after the transaction/service has been completed.
10. A Pet Owner’s rating can only give a rating between 0 to 5 and it **must** be an integer.
11. A Care Taker **is required** to have the pet under their care for the Entire Day (*24 Hrs*), for all Pet Days in the Transaction in which they accepted except for the day which the service starts and ends.
12. Care Takers employment types are classified into Full-Time or Part-Time.
13. A full time care taker **can opt** to take up to 65 days of leave, if they satisfy a minimum of 2 x 150 consecutive working days a year.
14. A full time care taker **cannot** take leave if they have at least 1 Pet under their care.

15. A Care Taker **must** be managed by exactly one PCSAdmin, and is **randomly** assigned to a PCSAdmin upon registration.
16. A Care Taker **can** only take care of up to 5 Pets at any single point of time.
17. A Care Taker **must** manually *accept* or *reject* each bid, regardless if they are full time or part time.
18. A Care Taker's daily price will increase with their rating:
    a. Rating between **4.0 and 4.2**, Daily Price is **$52**
    b. Rating between **4.2 and 4.4**, Daily Price is **$55**
    c. Rating between **4.4 and 4.6**, Daily Price is **$59**
    d. Rating between **4.6 and 4.8**, Daily Price is **$64**
    e. Rating between **4.8 and 5.0**, Daily Price is **$70**
19. A full time Care Taker **cannot** indicate their base price. The base price (**$50**) is set by their managing PCSAdmin upon registration and is scaled according to their ratings.
20. A full time Care Taker's monthly salary is calculated based on his **total Pet-Days** in a month. (i.e. If he/she takes care of 3 Pets a Day, that day is considered a 3 Pet-Day).
    a. If he/she works **less that 60 Pet-Days** in a month, they will receive **ONLY**:
    - Base Price (for that transaction) **x** Number of days in the transaction **x** total number of transactions (up to 60 pet-days)
    - E.g. Caretaker accepted two transactions: (1) $50/pet-day for 30 days of service and (2) $50/pet-day for 15 days of service. Both services fall within a single month. The total salary earned will be ($50 x 30) + ($50 x 15) = $2250

    b. If he/she works **more that 60 Pet-Days** in a month, they will receive **BOTH**:
    - Base Price (for that transaction) **x** Number of days in the transaction **x** total number of transactions (up to 60 pet-days)
    - **80%** of Base Price (for that transaction) **x** Number of excess Pet-Days in that month
    - E.g. Caretaker accepted three transactions: (1) $50/pet-day for 30 days of service, (2) $50/pet-day for 30 days of service and (3) $52/pet-day for 15 days of service. Assuming all three services fall within a single month and the third transaction happens last. The total salary earned will be ($50 x 30) + ($50 x 15) + ($52 x 15 x 0.8)= $2874
21. A part time Care Taker **can** specify their availability for the current year and the next year.
22. A part time Care Taker **cannot** take more than 2 Pets at any single point of time, if their rating is below 4.
23. A part time Care Taker **will** receive 75% of their price as payment.
24. A part time Care Taker **can** indicate the base price of the pets they are servicing.
25. All transactions history **will** be stored, regardless of whether a bid is rejected or accepted, and each transaction will have a status.
26. An administrator **can** only change the base price of the full-time Care Takers that they are managing.
27. Administrators **can** view the total salary earned and total pet days of every Care Takers, even those that they are not managing.
28. Administrators **will** earn commission from Care Takers who are part-timers, under their management and have accepted their biddings.

Entity Relationship Model



Constraints not shown in ER diagram:

- **Duration_to** and **duration_from** of transaction_details must be IN BETWEEN the **service_avail_from** and **service_avail_to** attributes. This is enforced in the SQL schema by table constraints.
- All caretakers have a limit of up to 5 Pets at any one time. This constraint is enforced by a SQL Trigger called **check_caretaker_limit**.
    - Full time caretakers and part time caretakers with a rating of 4/5 and higher can only participate in 5 transactions at any given time. In the ER diagram, this means that the number of transactions which have a transaction_status = 3 (which denotes in a transaction in progress) at any point in time <= 5.
    - Part time caretakers with a rating lower than a 4/5 can only participate in 2 transactions at any point in time. In the ER diagram, this means that the number of transactions which have a transaction_status = 3 (which denotes in a transaction in progress) at any point in time <= 2.
- A full time caretaker must work for a minimum of 2 x 150 consecutive days a year AND a full time caretaker is treated as available until they apply for leave. These constraints are enforced by a check performed by a SQL Function called **check_for_leave**.
- A full time caretaker cannot apply for leave if there is at least one Pet under their care. This constraint is enforced by a check performed by a SQL Function. This constraint is also enforced by a check performed by the **check_for_leave** SQL Function.
- A caretaker should not take care of pets they cannot care for. This constraint is enforced in the SQL schema using a foreign key in Transactions_Details and Offers_Services.
- The daily price for a full time caretaker increases with the rating of the caretaker but will never be below the base price. This constraint is enforced by a SQL Trigger **update_fulltime_price**.

Database schema

The constraints that cannot be enforced using table constraints/checks in the schema are enforced using SQL Triggers. The primary and foreign keys of the tables are indicated in the creation of each table.

Users ISA PetOwners, Caretakers, PCSAdmins schema

```
CREATE TABLE Users (
    email VARCHAR,
    full_name VARCHAR NOT NULL,
    user_password VARCHAR NOT NULL,
    profile_pic_address VARCHAR,
    user_area VARCHAR,
    user_address VARCHAR,
    is_deleted BOOLEAN DEFAULT FALSE,
    PRIMARY KEY (email)
);

CREATE TABLE PetOwners (
    owner_email VARCHAR
    REFERENCES Users(email)
    ON DELETE cascade,
    PRIMARY KEY (owner_email)
);

CREATE TABLE Caretakers(
    caretaker_email VARCHAR
    REFERENCES Users(email)
    ON DELETE cascade,
    employment_type VARCHAR NOT NULL,
    avg_rating NUMERIC DEFAULT 0,
    no_of_reviews INTEGER,
    PRIMARY KEY (caretaker_email)
);

CREATE TABLE PCSAdmins (
    admin_email VARCHAR
    REFERENCES Users(email)
    ON DELETE cascade,
    PRIMARY KEY (admin_email)
);
```

Manages and Categories schema

The **total participation** constraint for the PCSAdmins is enforced by executing an SQL function **assign_to_admin()** which is detailed in the most interesting SQL queries section of this report.

```
CREATE TABLE Manages (
    admin_email VARCHAR REFERENCES PCSAdmins(admin_email) ON DELETE cascade,
    caretaker_email VARCHAR REFERENCES Caretakers(caretaker_email) ON DELETE cascade,
```

```
    base_price NUMERIC DEFAULT 50,
    PRIMARY KEY (admin_email, caretaker_email)
);

CREATE TABLE Categories (
    pet_type VARCHAR PRIMARY KEY
);
```

Owns_Pets schema

```
CREATE TABLE Owns_Pets (
    owner_email VARCHAR REFERENCES PetOwners(owner_email)
    ON DELETE cascade,
    gender CHAR NOT NULL,
    pet_name VARCHAR NOT NULL,
    special_req VARCHAR,
    pet_type VARCHAR REFERENCES Categories(pet_type),
    is_deleted BOOLEAN DEFAULT FALSE,
    PRIMARY KEY (owner_email, pet_name, pet_type)
);
```

Offers_Services schema

The **is_avail** attribute denotes whether the service is valid and can be advertised to the pet owners on the website.

```
CREATE TABLE Offers_Services (
    caretaker_email VARCHAR REFERENCES Caretakers(caretaker_email)
    ON DELETE cascade,
    employment_type VARCHAR NOT NULL,
    service_avail_from DATE NOT NULL,
    service_avail_to DATE NOT NULL,
    type_pref VARCHAR NOT NULL,
    daily_price NUMERIC NOT NULL,
    is_avail BOOLEAN DEFAULT TRUE,
    PRIMARY KEY (caretaker_email, type_pref, service_avail_from, service_avail_to)
);
```

Transactions and transactions_details schema

The **t_status** attribute indicates the status of the transaction using integers.

- a **1** denotes that the transaction has just been SUBMITTED (it is a bid submitted by a pet owner to a caretaker). The caretaker has not taken any action.
- a **2** denotes that the transaction is REJECTED by the caretaker (the bid from the petowner was rejected by the caretaker)
- a **3** denotes that the transaction is IN PROGRESS/ACCEPTED by the caretaker (the bid is accepted so the transaction will be performed)
- a **4** denotes that the transaction has been COMPLETED (the service has been completed by the caretaker and the pet has been returned to the pet owner)

- a **5** denotes that a review for the caretaker has been submitted, written by the petowner, for the transaction after the completion of the transaction

```
CREATE TABLE Transactions_Details (
    caretaker_email VARCHAR,
    employment_type VARCHAR,
    pet_type VARCHAR,
    pet_name VARCHAR,
    owner_email VARCHAR CHECK (caretaker_email != owner_email),
    owner_review VARCHAR,
    owner_rating INTEGER,
    payment_mode VARCHAR NOT NULL,
    cost NUMERIC NOT NULL,
    mode_of_transfer VARCHAR NOT NULL,
    duration_from DATE NOT NULL, --Set by PetOwner
    duration_to DATE NOT NULL, --Set by PetOwner
    service_avail_from DATE NOT NULL,
    service_avail_to DATE NOT NULL,
    t_status INTEGER DEFAULT 1,
    PRIMARY KEY (caretaker_email, pet_name, owner_email, duration_to, duration_from),
    -- the start of the service must be same day or days later than the start of the
availability period
    CHECK (duration_from >= service_avail_from),
    -- the end of the service must be same day or earlier than the end date of the
availability period
    CHECK (duration_to <= service_avail_to),
    CHECK (caretaker_email != owner_email),
    FOREIGN KEY (owner_email, pet_name, pet_type) REFERENCES Owns_Pets(owner_email,
pet_name, pet_type),
    FOREIGN KEY (caretaker_email, pet_type, service_avail_from, service_avail_to)
    REFERENCES Offers_Services(caretaker_email, type_pref, service_avail_from,
service_avail_to)
);
```

Enquiries schema

```
CREATE TABLE Enquiries (
    user_email VARCHAR REFERENCES Users(email),
    enq_type VARCHAR,
    submission DATE,
    enq_message VARCHAR,
    answer VARCHAR,
    admin_email VARCHAR REFERENCES PCSAdmins(admin_email),
    PRIMARY KEY (user_email, enq_message)
);
```

Normalization level of database

All SQL tables have primary keys, which uniquely identify all other attributes within each table. Hence, all our tables are in BCNF format. This helps eliminate data redundancies and anomalies.

Three non-trivial triggers used in the application

1. Trigger to update the average rating of the caretaker and the number of reviews for the caretaker after every new review submission by a pet owner

This trigger is executed every time a new review/rating is submitted by a pet owner for a transaction that is complete. It ensures that the rating shown to the pet owners are as up to date and as accurate as possible.

First, it will count the number of non-NULL ratings that the caretaker has in the Transactions_details table and the no_of_reviews attribute of the caretaker in the Caretakers table will be updated accordingly.

If the number of reviews is not 0, it will compute the average of all the ratings that the caretaker has received from the pet owners and the avg_rating attribute of the caretaker in the Caretakers table will be updated accordingly, else the rating will be set to 0.

```
CREATE TRIGGER update_caretaker_rating
    AFTER UPDATE OF owner_rating ON Transactions_Details
    FOR EACH ROW
    EXECUTE FUNCTION update_caretaker_rating();

CREATE OR REPLACE FUNCTION update_caretaker_rating()
RETURNS TRIGGER AS $$
    DECLARE
           rating NUMERIC := 0;
           reviews_num INTEGER := 0;
    BEGIN
    SELECT COUNT(owner_rating) INTO reviews_num
    FROM Transactions_Details
    WHERE caretaker_email = NEW.caretaker_email;
    IF (reviews_num > 0) THEN
           SELECT AVG(owner_rating) INTO rating
           FROM Transactions_Details
           WHERE caretaker_email = NEW.caretaker_email;
    END IF;

    UPDATE Caretakers
    SET avg_rating = rating,
    no_of_reviews = reviews_num
     WHERE (caretaker_email = NEW.caretaker_email);

    RETURN NULL;
    END;
$$ LANGUAGE plpgsql;
```

2. Trigger to check whether the caretaker has already reached the maximum amount of pets he can care for when he accepts a bid by a pet owner

This trigger is implemented to enforce the constraint that a full time caretaker and a part time caretaker with a rating >= 4 cannot take care of more than 5 pets at any one time and a part time caretaker with a rating < 4 cannot take care for more than 2 pets at a time. This trigger will execute when there is an incoming update in the status of the transaction. More details can be found in the comments of the code block below.

```
CREATE TRIGGER check_caretaker_limit
    BEFORE UPDATE OF t_status ON Transactions_Details
    FOR EACH ROW
    EXECUTE PROCEDURE check_caretaker_limit();
CREATE OR REPLACE FUNCTION check_caretaker_limit()
RETURNS TRIGGER AS $$
    DECLARE
           date_start DATE := NEW.duration_from;
           date_end DATE := NEW.duration_to;
           emp_type VARCHAR := NEW.employment_type;
           rating NUMERIC;
           pet_limit INTEGER := 2;
           count BIGINT := 0;
    BEGIN
-- if the status update is to 2 (reject), 4 (complete), 5(review submitted), the checks do
not need to be performed and the update will be carried out.
           IF (NEW.t_status = 2 OR NEW.t_status = 4 OR NEW.t_status = 5) THEN
                  RETURN NEW;
           END IF;

-- code below will execute only when the caretaker is to accept a bid (NEW.t_status = 3)
           -- get rating of caretaker
           SELECT avg_rating INTO rating
           FROM Caretakers
           WHERE caretaker_email = NEW.caretaker_email;

-- for a full time caretaker or a part time caretaker with a rating >= 4, update the pet
limit to 5, else it remains at 2
           IF ((emp_type = 'parttime' AND rating >= 4) OR emp_type = 'fulltime') THEN
                  pet_limit := 5;
           END IF;

-- Loop over the each date of the new bid to be accepted and check if any of the days have
more than 5 transactions in progress, it if it does, raise the exception and the bid cannot
be accepted by the caretaker
           WHILE date_start <= date_end LOOP
-- select all the transactions that are also in the same availability period as the new
transaction to be accepted and check if their count >= 5
                  SELECT COUNT(*) INTO count
                  FROM Transactions_Details
                  WHERE (caretaker_email = NEW.caretaker_email
                  AND service_avail_from = NEW.service_avail_from
                  AND service_avail_to = NEW.service_avail_to AND t_status = 3
                  AND date_start >= duration_from AND date_start <= duration_to);
```

```
                IF (count >= pet_limit AND NEW.t_status = 3) THEN
                        RAISE EXCEPTION 'You have already reached the limit for the number
of pets you can take care of!';
                        RETURN NULL;
                END IF;
                date_start := date_start + 1;
        END LOOP;
-- the update will be performed only if the period of the new transaction accepted has been
looped over and the number of active transactions per day < 5
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
```

3. Trigger to update the daily price of a full time caretaker's services when his rating is updated after every new review submission

This trigger is implemented to enable the feature that a '**full time caretaker's price for his services will increase as his rating increases**', the daily price of a full time caretaker's services will increase according to what is laid out in *Data Constraints 19*. This price increase can only happen once a full time caretaker has at least 10 reviews, then the daily price will be adjusted according to his rating. If he/she has less than 10 reviews, his daily price will remain at $50 regardless of his/her rating.

```
CREATE TRIGGER update_fulltime_price
    AFTER UPDATE OF avg_rating ON Caretakers
    FOR EACH ROW
    EXECUTE PROCEDURE update_fulltime_price();
CREATE OR REPLACE FUNCTION update_fulltime_price()
RETURNS TRIGGER AS $$
    DECLARE
        emp_type VARCHAR := NEW.employment_type;
        rating NUMERIC;
        reviews INTEGER;
        new_price INTEGER := 50;
    BEGIN
        -- get rating of caretaker
        SELECT avg_rating, no_of_reviews INTO rating, reviews
        FROM Caretakers
        WHERE caretaker_email = NEW.caretaker_email;
        IF (emp_type = 'fulltime' AND reviews >= 10) THEN
            IF (rating > 4.2 AND rating < 4.4 ) THEN
                new_price := 52;
            ELSIF (rating > 4.2 AND rating < 4.4 ) THEN
                new_price := 55;
            ELSIF (rating > 4.4 AND rating < 4.6 ) THEN
                new_price := 59;
            ELSIF (rating > 4.6 AND rating < 4.8 ) THEN
                new_price := 64;
            ELSIF (rating > 4.8 ) THEN
                new_price := 70;
            END IF;
        END IF;
```

```
        EXECUTE 'UPDATE Manages SET base_price = $1 WHERE caretaker_email = $2' USING
new_price, NEW.caretaker_email;
        EXECUTE 'UPDATE Offers_Services SET daily_price = $1 WHERE caretaker_email = $2'
USING new_price, NEW.caretaker_email;
        RETURN NEW;
   END;
$$ LANGUAGE plpgsql;
```

Three most complex queries implemented in application

1. Query calculates the total commission earned by an admin

This query will return an admin email accompanied with the total commission earned by this admin. The commissions come from the 25% that the website takes from the completed transactions of part time caretakers who the admin manages. Detailed explanation of the sql query can be found in the comments within the code block.

This query can help to improve business decisions since administrators can infer the performance of the website from their commissions earned. If their commission earnings are decreasing, this might signal a decline in the number of transactions taking place on the site. The business can then investigate why this is happening in order to keep growing.

```
-- query calculates total commission earned by admin inserted in $1
-- returns admin email and the sum of all the commissions earned
SELECT m1.admin_email, SUM(m2.commission)
  FROM pcsadmins AS m1,
(SELECT m.admin_email AS admin_email, td.caretaker_email as caretaker_email,
-- calculates the commission earned from a single transaction
td.cost*SUM(td.duration_to::date-td.duration_from::date+1)*0.25 AS commission
-- commission earned if the caretaker is being managed by the pcsadmin
FROM manages m JOIN transactions_details td ON m.caretaker_email=td.caretaker_email
-- commission earned if employment type is part-time and transaction is accepted
WHERE td.employment_type='parttime'AND td.t_status>= 3
-- grouped to get the aggregate
GROUP BY m.admin_email, td.caretaker_email, td.cost, td.duration_from, td.duration_to) AS
m2
WHERE m1.admin_email=m2.admin_email AND m1.admin_email=$1
-- grouped to get the aggregate
GROUP BY m1.admin_email
```

2. Function checks if full time caretaker can leave

This function is executed when a full time caretaker applies for leave. It enforces the constraint that a full time caretaker must work at least 2 x 150 consecutive days for a year. It returns the two new availabilities (*new_service_avail_from1 and new_service_avail_to1*, *new_service_avail_from2 and new_service_avail_to2* respectively) that will need to be inserted into the **Offers_Services** table when the leave can be taken by the caretaker and also the leave period in days. The detailed step by step explanation of the checks performed can be found in the comments of the code block below.

This function is able to improve business decisions since it sets restrictions to the number of leaves a caretaker can take, hence ensuring a sufficient supply of caretakers available at any one time.

```
CREATE OR REPLACE FUNCTION check_for_leave(input_email VARCHAR, leave_start DATE, leave_end
DATE)
RETURNS TABLE (new_service_avail_from1 DATE, new_service_avail_to1 DATE,
new_service_avail_from2 DATE, new_service_avail_to2 DATE, leave_duration INTEGER) AS $$
DECLARE
  old_service_avail_from DATE; old_service_avail_to DATE; prev_150_start DATE;
  prev_150_end DATE; new_service_avail_to_1 DATE; new_service_avail_from_2 DATE;
  leave_period INTEGER;
BEGIN
-- Check if the leave period specified is valid, if not just return exception
IF leave_end < leave_start THEN
  RAISE EXCEPTION 'You cannot take leave during this period!';
END IF;
-- First, get the service period of the caretaker that CONTAINS the leave period from the
Offers_services table
SELECT service_avail_from, service_avail_to INTO old_service_avail_from,
old_service_avail_to FROM Offers_Services WHERE caretaker_email = input_email AND
leave_start >= service_avail_from AND leave_end <= service_avail_to AND is_avail = 't';

-- Then, check if there are any transactions accepted within the leave period, if yes
return exception since you cannot take leave when you are taking care of a pet
IF (SELECT COUNT(*) FROM Transactions_Details WHERE caretaker_email = input_email AND
service_avail_from = old_service_avail_from AND service_avail_to = old_service_avail_to AND
leave_start >= duration_from AND leave_end <= duration_to AND t_status = 3) > 0 THEN
  RAISE EXCEPTION 'You cannot take leave during this period!';
END IF;

-- Check whether the caretaker has already had a 150 consecutive day period IN THE SAME
YEAR. If they do, assign these values to the variables to be checked later
SELECT service_avail_from, service_avail_to INTO prev_150_start, prev_150_end FROM
Offers_services WHERE caretaker_email = input_email AND (service_avail_to -
service_avail_from >= 150) AND service_avail_to < old_service_avail_from;

-- check if the previous 150 day shift was completed in the same year. If it is not,
invalidate the variables containing the dates of the 150 day shift so OVERLAPS will always
be true
IF (SELECT extract(year from prev_150_end)) != (SELECT extract(year from
old_service_avail_from)) THEN
  prev_150_start := old_service_avail_from;
  prev_150_end := old_service_avail_to;
END IF;

leave_period := leave_end - leave_start + 1;
new_service_avail_to_1 := leave_start - 1;
new_service_avail_from_2 := leave_end + 1;
-- case when the start of the leave == service_avail_from date (e.g 1/1/2020 start leave
and 1/1/2020 start availability)
IF (leave_start = old_service_avail_from) THEN
  new_service_avail_to_1 := old_service_avail_from;
END IF;
```

```
-- case when end of leave == service_avail_to date (e.g 31/10/2020 end leave and 31/10/2020
end availability)
IF (leave_end = old_service_avail_to) THEN
   new_service_avail_from_2 := old_service_avail_to;
END IF;

-- check whether the previous 150 day shift has an overlap with the current one we are
looking at using the OVERLAP operator, if there is an overlap, we need to check whether the
curr availability period >= 300 days after subtracting the leave period, if not check for
150
IF (prev_150_start, prev_150_end+1) OVERLAPS (old_service_avail_from, old_service_avail_to)
THEN
 -- check if the curr period has at least 300 days since we need to split up into 2
consecutive 150 days
  IF (old_service_avail_to - old_service_avail_from - (leave_end - leave_start) > 300) THEN
  -- this is the case when the leave period is in the middle of the availability, need to
split up into 2 new availabilities
    IF (leave_start - old_service_avail_from > 150 AND old_service_avail_to - leave_end >
150) THEN
    -- this is when the date that the caretaker wants to take leave on is on the same day
the availability starts when he takes a one day leave so need to add 1 day to the date (e.g
availability starts on 1/1/2020 so the new availability should start on 2/1/2020)
      IF (old_service_avail_from = leave_start AND leave_period = 1) THEN
        old_service_avail_from := old_service_avail_from + 1;
        new_service_avail_to_1 := new_service_avail_to_1 + 1;
      END IF;
      IF (old_service_avail_to = leave_end AND leave_period = 1) THEN
        old_service_avail_to := old_service_avail_to + 1;
        new_service_avail_from_2 := new_service_avail_from_2 +1;
      END IF;
      RETURN QUERY SELECT old_service_avail_from AS new_service_avail_from1,
new_service_avail_to_1 AS new_service_avail_to1, new_service_avail_from_2 AS
new_service_avail_from2, old_service_avail_to AS new_service_avail_to2, leave_period AS
leave_duration;
    ELSE -- when the leave near the start or the end of the availability period
      IF (old_service_avail_from = leave_start AND leave_period = 1) THEN
        old_service_avail_from := old_service_avail_from + 1;
        new_service_avail_to_1 := new_service_avail_to_1 + 1;
      END IF;
      IF (old_service_avail_to = leave_end AND leave_period = 1) THEN
        old_service_avail_to := old_service_avail_to + 1;
        new_service_avail_from_2 := new_service_avail_from_2 + 1;
      END IF;
      RETURN QUERY SELECT old_service_avail_from AS new_service_avail_from1,
new_service_avail_to_1 AS new_service_avail_to1, new_service_avail_from_2 AS
new_service_avail_from2, old_service_avail_to AS new_service_avail_to2, leave_period AS
leave_duration;
    END IF;
  ELSE
    RAISE EXCEPTION 'You cannot take leave during this period!';
  END IF;
```

```
-- this ELSE means that there was already a 150 day consecutive period worked in the past
in the same year by the caretaker, so just look for another 150 consecutive days
ELSE
  IF (old_service_avail_to - old_service_avail_from - (leave_end - leave_start) > 150) THEN
  -- if can split up, return true
    IF (old_service_avail_from = leave_start AND leave_period = 1) THEN
      old_service_avail_from := old_service_avail_from + 1;
      new_service_avail_to_1 := new_service_avail_to_1 + 1;
    END IF;
    IF (old_service_avail_to = leave_end AND leave_period = 1) THEN
      old_service_avail_to := old_service_avail_to + 1;
      new_service_avail_from_2 := new_service_avail_from_2 + 1;
    END IF;

    RETURN QUERY SELECT old_service_avail_from AS new_service_avail_from1,
new_service_avail_to_1 AS new_service_avail_to1, new_service_avail_from_2 AS
new_service_avail_from2, old_service_avail_to AS new_service_avail_to2, leave_period AS
leave_duration;
  ELSE -- if less then 150 days, return false
    RAISE EXCEPTION 'You cannot take leave during this period!';
  END IF;
END IF;
END;
$$ LANGUAGE plpgsql;
```

3. Function to get underperforming caretakers

This function will return a table containing all underperforming caretakers with their average rating, number of pet days worked, and the respective number of given ratings for each rating value (0 - 5). An underperforming caretaker is defined as a caretaker with an average rating that is **less than 2**. Only caretakers who have had at least 1 transaction that has been given a rating will be considered. This provides PCS Admins the functionality to view statistics related to underperforming caretakers and take relevant actions accordingly. Detailed explanation of the sql function can be found in the comments within the code block.

This function allows the managing administrators to investigate these caretakers and find out the core reasons for their poor ratings. Doing so enables services provided on the platform to be improved if the reasons for poor performances are fixed, hence allowing the overall business on the platform to improve.

```
-- created custom type to hold desired values in a single row
CREATE TYPE return_type AS
( caretaker VARCHAR, num_pet_days NUMERIC, avg_rating NUMERIC, num_rating_5 NUMERIC,
num_rating_4 NUMERIC, num_rating_3 NUMERIC, num_rating_2 NUMERIC, num_rating_1 NUMERIC,
num_rating_0 NUMERIC );
CREATE OR REPLACE FUNCTION get_underperforming_caretakers()
RETURNS SETOF return_type AS $$
   DECLARE
          val return_type;
          i RECORD;
          j RECORD;
   BEGIN
```

```
        -- get table containing caretaker email, average rating, and total num of pet
days of caretakers
        -- who have at least 1 transaction that has been given a rating
        -- and has an average rating of less than 2
        FOR i IN (SELECT caretakers.caretaker_email, avg_rating,
SUM(duration_to::date-duration_from::date+1)
                FROM caretakers INNER JOIN transactions_details ON
caretakers.caretaker_email = transactions_details.caretaker_email
                WHERE EXISTS (SELECT 1 FROM transactions_details t
                        WHERE t.caretaker_email = caretakers.caretaker_email
                        AND t.owner_rating IS NOT NULL)
                AND avg_rating < 2
                GROUP BY caretakers.caretaker_email) LOOP
            val.caretaker := i.caretaker_email;
            val.num_pet_days := i.sum;
            val.avg_rating := i.avg_rating;
            val.num_rating_5 := 0;
            val.num_rating_4 := 0;
            val.num_rating_3 := 0;
            val.num_rating_2 := 0;
            val.num_rating_1 := 0;
            val.num_rating_0 := 0;
            -- find number of ratings for each value for each caretaker
            FOR j IN (SELECT owner_rating, COUNT(*)
                        FROM transactions_details
                        WHERE caretaker_email = i.caretaker_email
                        GROUP BY owner_rating) LOOP
                IF j.owner_rating = 0 THEN
                        val.num_rating_0 := j.count;
                ELSIF j.owner_rating = 1 THEN
                        val.num_rating_1 := j.count;
                ELSIF j.owner_rating = 2 THEN
                        val.num_rating_2 := j.count;
                ELSIF j.owner_rating = 3 THEN
                        val.num_rating_3 := j.count;
                ELSIF j.owner_rating = 4 THEN
                        val.num_rating_4 := j.count;
                ELSIF j.owner_rating = 5 THEN
                        val.num_rating_5 := j.count;
                END IF;
            END LOOP;
            RETURN NEXT val;
        END LOOP;
        RETURN;
    END;
$$ LANGUAGE plpgsql;
```

4. Function to assign a caretaker to an administrator at registration

This function is executed when a new caretaker (either part time or full time) signs up for an account at Pet Society. The function will randomly assign an admin account that exists in the **PCSAdmins** table to the caretaker

and insert an entry into the **Manages** table. If the caretaker that signs up is a full time caretaker, the **base_price** attribute will be set to $50 as this will be the starting daily price of a full time caretaker at Pet Society.

```
CREATE OR REPLACE FUNCTION assign_to_admin(input_email VARCHAR, emp_type VARCHAR)
RETURNS NUMERIC AS $$
    DECLARE
           assigned_admin VARCHAR;
           daily_price NUMERIC;
    BEGIN
           -- if all admins are already in Manages (total participation achieved)
          IF (SELECT admin_email FROM PCSAdmins EXCEPT SELECT admin_email FROM Manages) = 0
    THEN
                  SELECT admin_email into assigned_admin
                  FROM PCSAdmins
                  ORDER BY RANDOM()
                  LIMIT 1;
           -- select admin by random from admins NOT in the Manages table yet
           ELSE
                  SELECT admin_email into assigned_admin
                  FROM (SELECT admin_email FROM PCSAdmins
                         EXCEPT
                         SELECT admin_email FROM Manages) AS admin_not_in_manages
                  ORDER BY RANDOM()
                  LIMIT 1;
           END IF;

           EXECUTE 'INSERT INTO Manages(admin_email, caretaker_email) VALUES ($1,$2)'
           USING assigned_admin, input_email;
           IF emp_type = 'fulltime' THEN
                  RETURN 50;
           END IF;
           RETURN 0;
    END;
$$ LANGUAGE plpgsql;
```

Tools and Frameworks used

We used the PERN stack to develop our application:

- PostgreSQL - Database
- Express - Server framework
- ReactJS - Frontend framework
- NodeJS - Server runtime environment

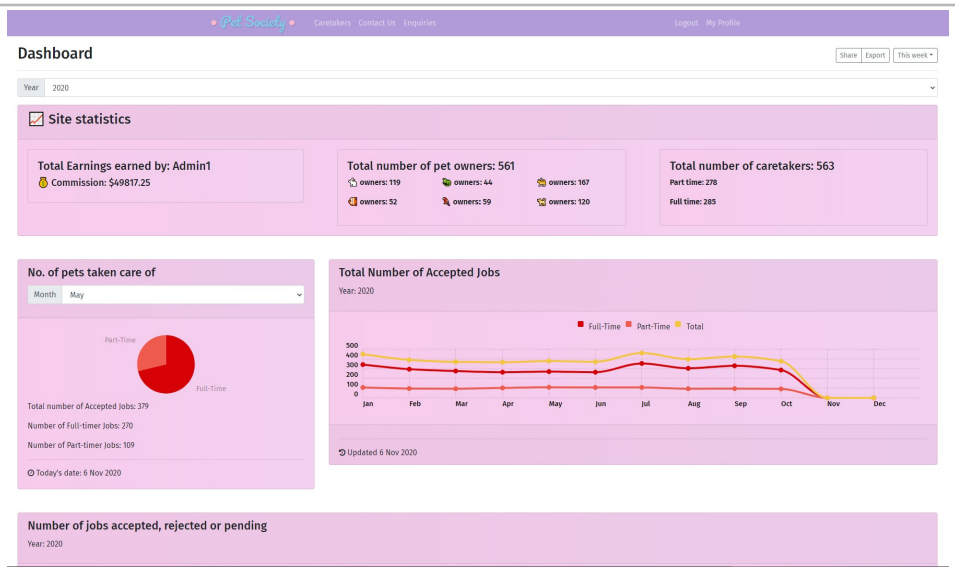
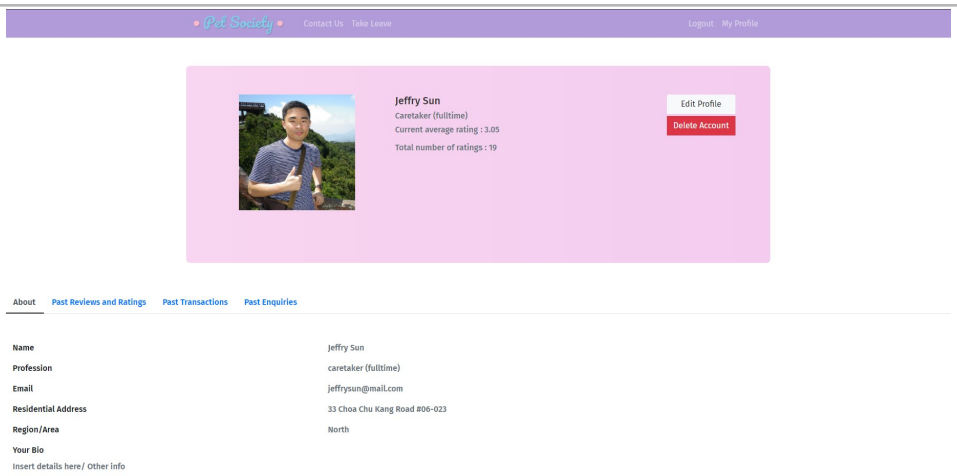
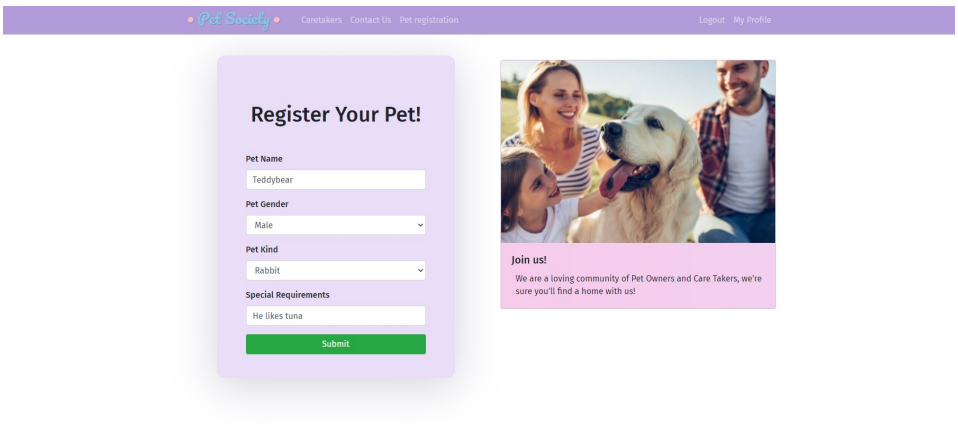
Deployment was done using Heroku:

- Heroku - Deployment platform

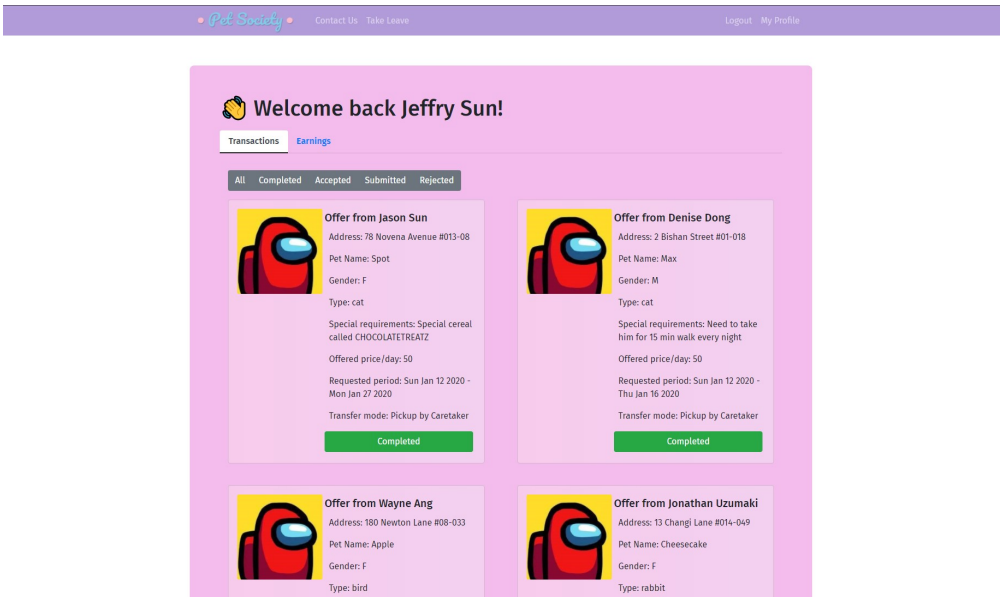
Frontend design was done using Bootstrap:

- Bootstrap - Frontend CSS library

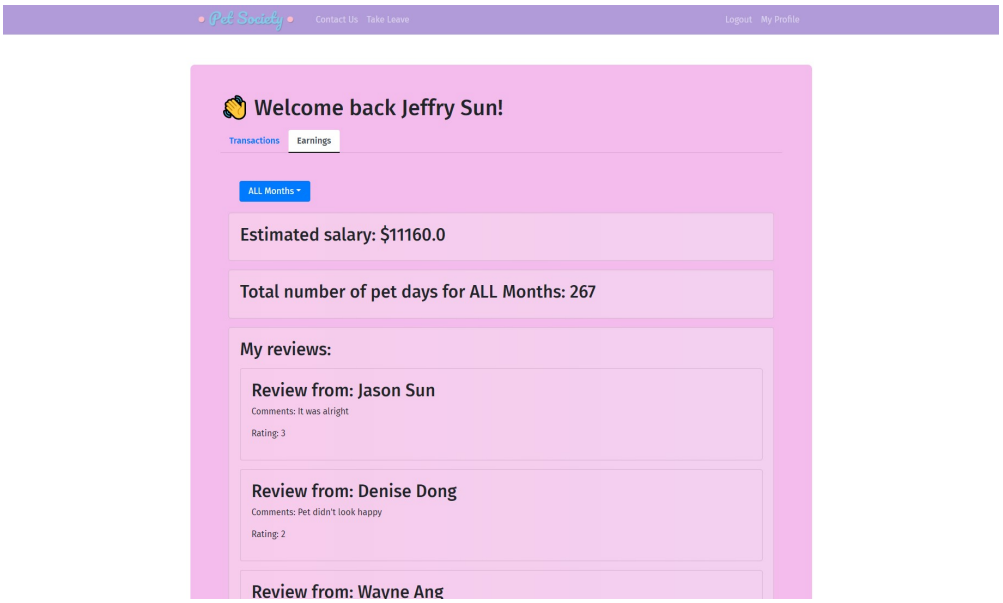
Screenshots of application

Admin Dashboard Page	
Caretaker's Profile page	
Pet registration page for pet owners	

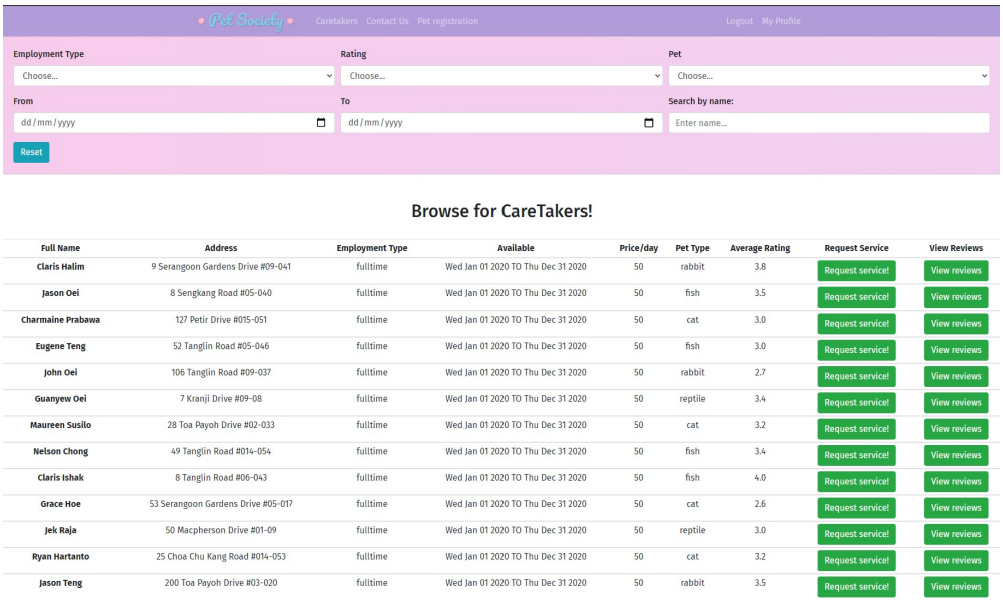
Caretaker's homepage showing his transactions



Caretaker's earnings summary page



Page to browse for caretakers's services



Full Name	Address	Employment Type	Available	Price/day	Pet Type	Average Rating	Request Service	View Reviews
Claris Halim	9 Serangoon Gardens Drive #09-041	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	rabbit	3.8	Request service!	View reviews
Jason Oei	8 Sengkang Road #05-040	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	fish	3.5	Request service!	View reviews
Charmaine Prabawa	127 Petir Drive #015-051	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	cat	3.0	Request service!	View reviews
Eugene Teng	52 Tanglin Road #05-046	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	fish	3.0	Request service!	View reviews
John Oei	106 Tanglin Road #09-037	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	rabbit	2.7	Request service!	View reviews
Guanyew Oei	7 Kranji Drive #09-08	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	reptile	3.4	Request service!	View reviews
Maureen Susilo	28 Toa Payoh Drive #02-033	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	cat	3.2	Request service!	View reviews
Nelson Chong	49 Tanglin Road #014-054	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	fish	3.4	Request service!	View reviews
Claris Ishak	8 Tanglin Road #06-043	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	fish	4.0	Request service!	View reviews
Grace Hoe	53 Serangoon Gardens Drive #05-017	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	cat	2.6	Request service!	View reviews
Jek Raja	50 Macpherson Drive #01-09	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	reptile	3.0	Request service!	View reviews
Ryan Hartanto	25 Choa Chu Kang Road #014-053	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	cat	3.2	Request service!	View reviews
Jason Teng	200 Toa Payoh Drive #03-020	fulltime	Wed Jan 01 2020 TO Thu Dec 31 2020	50	rabbit	3.5	Request service!	View reviews

Summary of difficulties encountered and lessons learned from project

- Leveraging the power of DBMS to make the application fast and efficient was a problem faced in the early stages of our project. After we learnt about normal forms, we saw certain redundancies that appeared in our database that should be removed to prevent any data anomalies. This showed us the importance of using the ER diagram and carefully planning the schema at the beginning.
- With careful planning at the start, we could have avoided some issues due to data parsing. In our initial implementation, we stored durations in our database as a string, with start and end dates concatenated together and delimited by a comma. We faced difficulties parsing the string into dates for calculations, and had to re-implement our schema halfway through the project to store dates as DATE format in the database instead.
- Creating complex queries was challenging with the knowledge and SQL constructs exposed to us from the module. None of us were familiar with pLpgSQL. In the end, we were able to find better ways to make our existing queries better and more efficient by learning and using the available default PostgreSQL functions such as LOOP and OVERLAP. We were able to make efficient queries with the usage of SQL transactions as well.
- Lack of experience in the field of full-stack web development technologies, such as React and NodeJS posed an initial steep learning curve for some of the group members to keep up with the overall development pace of the team.
- Splitting the workload was a problem faced throughout the project as many of the features and components are closely intertwined and have mutual dependencies. This problem makes it difficult to work for us to work independently which resulted in a slow initial start.