CS2102 Project Report AY2021 SEMESTER 1
Team 32

Loh Sze Ying (A0185673R)
Chua Huixian (A0190360M)
Park Junhyuk (A0170638B)
Aloysius Lim Dewen (A0200593M)
Woo Huiren (A0202242B)

1. Project Responsibilities

The project responsibilities are split into different features.
Aloysius is in charge of designing the SQL and responsible for the creation of triggers and SQL tables. Aloysius also helped with the UI.
Sze Ying is in-charge of caretaker features, search, making the login suitable to use with the new SQL table for the project, and deployment to Heroku.
Huixian is in-charge of the pet owner features, displaying of the pet owner's bids, and editing of UI.
Huiren is in-charge of all the admin features and assisting with SQL queries.
Junhyuk is in-charge of the bid features that have the ability to search for available caretakers and submit bids to them accordingly.

2. Data Requirements and Functionalities of Application

The project is split into features, namely Pet Owner, Caretaker, Bidding, Admin, Reviews and SQL.

User

As a user, they are able to update their name and area. They are also able to update their password and credit card information.

Users are able to search for other users via their usernames. Each user has their own profile with their name, area and list of pets, if any. If the user is a caretaker, their reviews/rating will be shown as well.

Pet Owner

A user is automatically considered as a pet owner once they register an account. Hence, all users are pet owners. As a pet owner, users can add their pets by indicating their pet's name and type. After adding a pet, users can edit the pet's name, as well as add any special requirements (which have been predefined by the system) they may have. Special requirements can also be removed. Users cannot delete their pets, but can enable and disable their statuses.

Pet owners can also view a list of their bids, organised according to accepted, pending, rejected and completed.

Caretaker

A registered user can apply to be a caretaker via updating the user's own profile as caretaker. Once applied as caretaker, the user cannot undo its action, and is considered a part-time caretaker. As a part-time caretaker, the user can add his/her own free dates to take care of pets. The date input must be in dd/mm/yyyy format. Free dates known as availability. After adding free dates, there is a table that will display a list of available dates. Caretaker, whether part-timer or full-timer, can also apply for leave by indicating the date that wants to be removed from the available dates list. The system will then check with the SQL if it is valid before allowing for the availability to be deleted. Only a PCS admin can make a part-time caretaker into a full-time caretaker with the admin dashboard.

A caretaker can also see his ratings which is calculated by the ratings he got and his history of monthly salaries.

Both part-time and full-time caretakers can add the type of pet they can care for. This means that if a caretaker can take care of a pet, the caretaker can add the pet type into the “Pet type” table via a dropdown list. This is so that when bidding occurs, the pet owner who has a cat as pet type will be matched with the suitable caretaker that can take care of the pet. Additionally, only part-time caretakers can update the price of each type of pet. To update the price of each pet type, a dropdown list will be listed that shows the specific’s caretaker pet that he/she can take care of. Then, there is a text input to edit the price. Full-time caretakers can only add the pet type via dropdown list.

Once a pet owner bidded for a caretaker, any suitable full-time caretaker will automatically accept pending bids. Only a part-time caretaker can choose to accept or reject a pending bid via a button in the pending bids table. After a bid is accepted, accepted bids will be listed in the “All accepted bids” table. Furthermore, this would cause all invalid pending bids to be rejected automatically. Both full-time and part-time caretakers can mark an accepted bid as complete by pressing the button. After marking it as complete, pet owner who bidded for the caretaker can rate and review. All completed bids that the caretaker had taken care of are displayed in a table form, with details such as start and end date, reviews and ratings.

Bid

A pet owner can make a bid with information about the start date, end date, pet name , caretakers, transfer methods and payment types.

Available caretakers are first searched with the start date and end date for the period of the pet care service they want and the pet name that they saved previously in “Manage your pets”. Date inputs for start date and end date must follow the format “dd/mm/yyyy” and the pet owner can either type it in or select using the on-screen calendar click. A pet name can be selected from the drop-down. After the selection, click on the “Search” button to search for the available caretakers.

Available caretakers are filtered and shown in a drop-down in descending order of ratings and ascending order of names. Ratings are shown together with the caretaker name in a bracket.

Select a caretaker from the list shown in the drop-down together with the transfer methods and payment types. Transfer methods (Caretaker_Pickup, Owner_Deliver and PCS building) are decided by the pet owner as they can choose from the drop-down the method they want. Similarly, payment types (Cash or Credit Card) are chosen by the pet owner in the same manner. Pet owners can then click on the “Submit Bid” button to submit the bid with all the bid information (Start Date, End Date, Pet name, caretaker name, transfer methods and payment types) to the target caretaker. For full-time caretakers, the bid is automatically accepted while for part-time caretakers, the bid would be pending by default.

Admin

An administrator authenticates on the admin login page which then redirects them to the admin dashboard. The dashboard enables the administrator to navigate through the 6 features.

Caretaker Stats page allows the administrator to view the total amount of salary to pay to all caretakers for a specified month and year. The date input must follow the format “dd/mm/yyyy”. The day can be any day of the month.

Pet Stats page allows the administrator to view the total number of pets taken care of in a specified month and year. The date input must follow the format “dd/mm/yyyy”. The day can be any day of the month.

Pet price page allows the administrator to update the base price for all pet types. The data is displayed in a table with an update button for every pet type row. The input data must be a number. Hitting the update button will then update the respect pet type data with the new value.

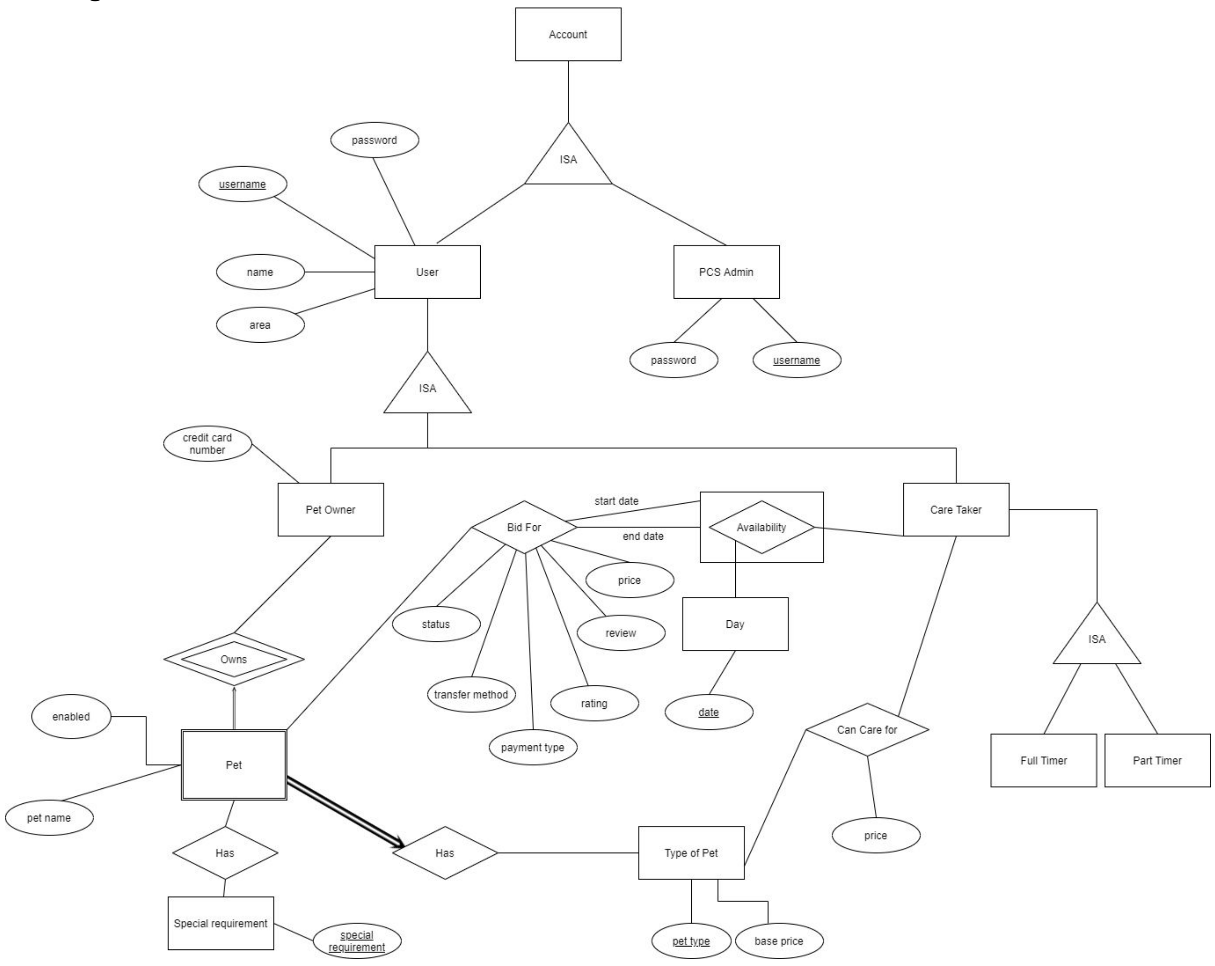
Admin creator page allows the administrator to create new administrators. There are two inputs, the username and password. With unique constraint on the username, the creation will display an error message if the username already exists. Password is then hashed and salted using the bcrypt package.

Job Stats page allows the administrator to query and find out which month has the highest number of jobs for the year. The date input must follow the format “dd/mm/yyyy”. The day and month can be any day or month of the year. It displays the month and job count for the respective month of the year.

Caretaker Status page allows the administrator to adjust the caretaker’s status from full-time to part-time and vice versa. This can be done by inputting the username and selecting the status option. If the username is invalid, the admin will be informed that the caretaker does not exist. If the caretaker is already full-time and the new status option is also full-time, an error will be presented to the admin, informing the admin that the caretaker is already a full-timer.

3. ER Model and constraints

ER Diagram



Constraints

Accounts satisfy the covering constraint but allow for overlaps (same account username can be used for user accounts and admin accounts as User and PCS Admin accounts are stored in different tables).
Account cannot be deleted.

PCS Admin can be identified by an username attribute (primary key constraint), and have a password attribute.

PCS Admin accounts cannot be created by the public.
PCS Admin accounts can be created by other PCS Admins.

Users can be identified by an username attribute (primary key constraint), and have a password attribute.
All Users are Pet Owners.
Users can be both a Pet Owner and Caretaker (i.e. User ISA has covering constraint).
Users must have their name and area recorded.
Users can only be created by the public.
Users can change their name and area.
Users can change their password.

Pet can be identified by a composite key of Pet's name attribute and their Pet Owner's username attribute.
Pet has a foreign key constraint to the Pet Owner's username attribute.
Pet has a dependency constraint to the Pet Owner.

Pets cannot be deleted.
Pets can be enabled or disabled.
Pets' name can be changed.
Each Pet can have multiple special requirements.
Special requirements can be identified by a description attribute (primary key constraint)
Special requirements can be added or removed by the pet's owner.
Each Pet has a pet type. (e.g. cat, dog etc.)
Type of Pet can be identified by a name attribute (primary key constraint).
Type of Pet name can only be deleted or changed if there are no foreign entities referencing it.

Caretakers must be either a Full-time Caretaker or Part-time Caretaker, but not both.(i.e. Care takers ISA has no overlapping constraint, and has a covering constraint).
Caretakers can set their availability which the Pet Owner can bid for.
Full time caretakers will have their availability set to full by default.
Only admins can change part time caretakers to full time and vice versa.
Caretakers can view their own reviews and ratings given by Pet Owners.
Caretakers should not take care of Pets they cannot care for, but can take care of more than one Pet at any given time.
The overall rating of a caretaker will be calculated based on the ratings of his bids and defaults to 3 if it does not exist, it is not an attribute of the caretaker as it can be derived from the bids when needed.

Full-time Caretaker must work a minimum of 300 consecutive days a year, including weekends.
Full-time Caretaker is always treated as available to take care of pets until they delete their availability.
Full-time Caretaker cannot take emergency leave and cannot take leave when they have a successful or accepted bid on the day.
Full-time Caretaker can only take care of at most 5 pets at any given time.
Full-time Caretaker will always accept a job when a job is available.

Part-time Caretaker must specify their availability for the current year and next year.
Part-time Caretaker can only take care of at most 5 pets at any given time.
Part-time Caretaker must have a rating of at least 3.5 out of 5 to take care of up to 3 Pets at any given time, 4 out of 5 to care up to 4 pets, 4.5 out of 5 to care up to 5 pets.
Part-time Caretakers can set their own base daily price.
Days can be identified by a date attribute (primary key constraint).

Pet Owners can create Pets.
Pet Owners can view their pet’s information.
Pet Owners can search for Caretakers.
Pet Owners can view reviews and ratings of Caretakers.
Pet Owners can bid for Caretakers’ services for each of their Pets, and the successful bidder can be chosen by either Caretaker or the system.

An Availability is identified by Caretaker and date.

A bid is identified by a composite key of pet owner username, pet name, start date, end date and caretaker username.
A Bid can be rejected, pending, accepted and completed.
Only completed Bid can have a review and rating for the corresponding Caretaker at the end.
There must exist all availabilities for a caretaker between the start availability and end availability of the caretaker.
There must not have any overlap of accepted/pending or completed bids for the same pet with overlapping dates.
The caretaker must be able to take care of a pet that is in the bid.
A bid will have a total price that is auto computed depending on the caretaker prices.

PCS Administrator will specify the minimum base daily price for full-time Caretaker.
Base daily price differs according to the type of Pet that is being taken care of.
As Full Time Caretaker’s rating increased, base daily price increased. The calculation is as followed: If the rating is above 4, the price is 1.25x. If it is above 4.5, it is 1.5x. It is auto updated whenever a new bid with rating is updated for full timers.

The price of a bid will not change once bidded even if the Caretaker’s base daily price increases.
Pet-day depends on how many bids were accepted or completed and how the number of days the bid lasts for during a month. The month of a bid is dependent on its start date.
Once a full-time Caretaker hits 60 pet-days, he will earn an addition of 80% of excess price whose pet day is not counted to the 60 pet days in addition to his base salary of 3000. He will get the base salary of 3000, regardless of whether he reaches 60 pet days or not.
Part-time Caretakers will receive 75% of the total price of accepted/completed bids whose start date is within the month as their salary.

Additional Constraints
Whenever a bid has been accepted (status changed from pending to accepted), all bids that have become invalid (because the caretaker cannot care for so many pets etc) will become rejected automatically.
Whenever a caretaker applies for leave and has an availability deleted, all pending bids that fall within the date will have their status changed to rejected.

Aggregation Assumptions
For the month with the highest number of jobs: Jobs are counted and calculated by the start date of it.

4. Relational Schema

```
CREATE TABLE PCSAdmin
(
    username VARCHAR(64) PRIMARY KEY,
    password VARCHAR NOT NULL
```

```
);
CREATE TABLE Users
(
    username VARCHAR(64) PRIMARY KEY,
    password VARCHAR     NOT NULL,
    name     VARCHAR(64) NOT NULL,
    area     VARCHAR(64) NULL
);
CREATE TABLE PetOwner
(
    username VARCHAR(64) PRIMARY KEY,
    credit_card_number NUMERIC(20) DEFAULT NULL,
    FOREIGN KEY (username) REFERENCES Users (username) ON UPDATE CASCADE
);
CREATE TABLE CareTaker
(
    username    VARCHAR(64) PRIMARY KEY,
    is_fulltime BOOLEAN DEFAULT FALSE,
    FOREIGN KEY (username) REFERENCES Users (username) ON UPDATE CASCADE
);
CREATE TABLE PetTypes
(
    pet_type VARCHAR(64) PRIMARY KEY,
    base_price NUMERIC(10,2) NOT NULL
);
CREATE TABLE SpecialRequirements
(
    special_requirement VARCHAR PRIMARY KEY
);
CREATE TABLE Pet
(
    owner_username VARCHAR(64),
    pet_name           VARCHAR(64),
    enabled        BOOLEAN DEFAULT TRUE,
    pet_type       VARCHAR(64) NOT NULL,
    PRIMARY KEY (owner_username, pet_name),
    FOREIGN KEY (owner_username) REFERENCES PetOwner (username) ON UPDATE CASCADE,
    FOREIGN KEY (pet_type) REFERENCES PetTypes (pet_type) ON UPDATE CASCADE
);
CREATE TABLE PetSpecialRequirements
(
    owner_username       VARCHAR(64),
    pet_name                 VARCHAR(64),
    special_requirement VARCHAR,
```

```sql
    PRIMARY KEY (owner_username, pet_name, special_requirement),
    FOREIGN KEY (owner_username, pet_name) REFERENCES Pet (owner_username, pet_name) ON UPDATE
CASCADE ON DELETE CASCADE,
    FOREIGN KEY (special_requirement) REFERENCES SpecialRequirements (special_requirement) ON
UPDATE CASCADE ON DELETE CASCADE
);
CREATE TABLE CareTakerPricing
(
    username VARCHAR(64),
    pet_type VARCHAR(64),
    price    DECIMAL(8, 2),
    PRIMARY KEY (username, pet_type),
    FOREIGN KEY (username) REFERENCES CareTaker (username) ON UPDATE CASCADE,
    FOREIGN KEY (pet_type) REFERENCES PetTypes (pet_type) ON UPDATE CASCADE
);
CREATE TABLE CareTakerAvailability
(
    username VARCHAR(64),
    date     DATE NOT NULL,
    PRIMARY KEY (username, date),
    FOREIGN KEY (username) REFERENCES CareTaker (username) ON UPDATE CASCADE
);
CREATE TYPE transfer_methods AS ENUM ('OWNER_DELIVER', 'CARETAKER_PICKUP', 'PCS_BUILDING');
CREATE TYPE payment_types AS ENUM ('CREDIT_CARD', 'CASH');
--boolean function to check if a certain caretaker can care for a certain pet
CREATE OR REPLACE FUNCTION ABLETOCAREFOR(pname VARCHAR, owner_name VARCHAR, caretaker_username
VARCHAR)
    RETURNS BOOLEAN AS
    $$ BEGIN
        RETURN EXISTS (
            SELECT 1
            FROM pet P, CareTakerPricing C
            WHERE pname = P.pet_name AND owner_name = P.owner_username AND P.pet_type =
C.pet_type AND caretaker_username = C.username
            );
    END;
$$
LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION ONLEAVE(cname VARCHAR, start_date date, end_date date)
    RETURNS BOOLEAN AS
    $$ BEGIN
        RETURN EXISTS (SELECT 1
                            FROM generate_series(start_date, end_date, interval '1 day') AS
t(day)
                            WHERE NOT EXISTS (SELECT 1
                                                FROM CareTakerAvailability
                                                WHERE username = cname and date = t.day)
                        );
        END;
        $$ LANGUAGE plpgsql;

CREATE TYPE bidstatus AS ENUM ('ACCEPTED', 'PENDING', 'REJECTED', 'COMPLETED');

CREATE TABLE Bids
(
    status      bidstatus         DEFAULT 'PENDING' NOT NULL,
    rating          SMALLINT         DEFAULT NULL CHECK (rating >= 0 AND rating <= 5),
    review          VARCHAR          DEFAULT NULL,
    total_price           DECIMAL(8, 2),
    transfer_method transfer_methods DEFAULT NULL,
    payment_type    payment_types    DEFAULT NULL,
    owner_username  VARCHAR(64),
    pet_name        VARCHAR(64),
    caretaker_username    VARCHAR(64),
    start_date      DATE  NOT NULL,
    end_date DATE NOT NULL,

    PRIMARY KEY (owner_username, pet_name, caretaker_username, start_date, end_date),
    FOREIGN KEY (owner_username, pet_name) REFERENCES Pet (owner_username, pet_name) ON UPDATE
CASCADE,
    FOREIGN KEY (caretaker_username) REFERENCES CareTaker (username) ON UPDATE CASCADE,
    CONSTRAINT LEGALTIMEPERIOD CHECK (start_date <= end_date),
    CONSTRAINT ABLETOCAREFOR CHECK (status = 'REJECTED' OR ABLETOCAREFOR(pet_name,
owner_username, caretaker_username)),
    CONSTRAINT NOFALSERATINGS CHECK (status = 'COMPLETED' OR (rating IS NULL AND review IS
NULL)),
    CONSTRAINT NOTONLEAVE CHECK (status = 'REJECTED' OR NOT ONLEAVE(caretaker_username,
start_date, end_date))
);
```

5. Normal Form

PCSAdmin Table
PCSAdmin.username -> PCSAdmin.password
PCSAdmin Table is in BCNF.

Users Table

username -> password, name, area
Users Table is in BCNF.

PetOwner Table
petowner_username -> credit_card_number
PetOwner Table is in BCNF.

CareTaker Table
caretaker_username -> is_fulltime
CareTaker Table is in BCNF.

PetTypes Table
pet_type -> base_price
PetTypes Table is in BCNF.

SpecialRequirements Table
special_requirement -> special_requirement
SpecialRequirements Table is in BCNF.

Pet Table
petowner_username, pet_name -> enabled, pet_type
Pet Table is in BCNF.

PetSpecialRequirements Table
petowner_username, pet_name, special_requirement -> petowner_username, pet_name, special_requirement
Pet Table is in BCNF.

CareTakerPricing Table
caretaker_username, pet_type -> price, pet_type
CareTakerPricing Table is in BCNF.

CareTakerAvailability Table
caretaker_username, date -> caretaker_username, date
CareTakerAvailability Table is in BCNF.

Bids Table
petowner_username, pet_name, caretaker_username, start_date, end_date -> status, rating, review, total_price, transfer_method, payment_type
Bids Table is in BCNF.

Since all fragments (the SQL tables) are in BCNF, hence the database is in BCNF.

6. Triggers

We have quite a number of triggers to ensure the data consistency of our SQL database. Below are the 3 most complex one that we have decided to showcase.

Trigger 1
This trigger is done before a bid is inserted. It does multiple checks before the bid is allowed. Firstly, the pet that is to be taken care of must not have an overlapping accepted or completed bid within the timeframe of the new bid. Next, a check if the caretaker is available on ALL the days of the bid

timeframe. After that, another check will be done to see if the caretaker is currently full on any of the days in the timeframe. This is done by getting his current rating which is a function and then checking the bids for all accepted or completed on the timeframe. If he is busy, the bid status will be rejected automatically. Next, the price of the bid will be calculated based on the timeframe, and the price of the pettype by day. Lastly, if the caretaker is a full-timer, the bid is auto accepted.

Code for the trigger itself:

```
--checks to be done before the bid is inserted
CREATE OR REPLACE FUNCTION CHECKBEFOREINSERTBID()
    RETURNS TRIGGER LANGUAGE plpgsql
    AS $$
    DECLARE able BOOLEAN;
    DECLARE rating NUMERIC;
    DECLARE fulltime BOOLEAN;
    BEGIN
    -- check for overlapping bids with the same pet that is already accepted
    able := (SELECT NOT EXISTS (SELECT 1
                    FROM Bids B
                    WHERE (B.status = 'ACCEPTED' OR B.status = 'COMPLETED') AND
B.pet_name = NEW.pet_name AND B.owner_username = NEW.owner_username
                        AND (NEW.start_date, NEW.end_date + interval '1 day')
OVERLAPS (B.start_date, B.end_date + interval '1 day')
                    ));
    IF NOT able THEN
        RAISE EXCEPTION 'YOU ALREADY HAVE ACCEPTED OR COMPLETED BID OVERLAPPING!';
    END IF;
    --check for caretaker availability
    able := (SELECT NOT EXISTS (SELECT 1
                                FROM generate_series(NEW.start_date, NEW.end_date,
interval '1 day') AS t(day)
                                WHERE NOT EXISTS ( SELECT 1
                                                   FROM CareTakerAvailability A
                                                   WHERE A.date = t.day AND
A.username = NEW.caretaker_username
                                                   )
                                ));
    IF NOT able THEN
        RAISE EXCEPTION 'Caretaker unavailable on one of those days!';
    END IF;
    --check whether the caretaker is still able to take more pets

    fulltime = (SELECT is_fulltime FROM CareTaker WHERE username =
NEW.caretaker_username);
    rating = GET_RATING(NEW.caretaker_username);
    able := (SELECT (SELECT CASE WHEN fulltime THEN 5
                 WHEN rating > 4.5 THEN 5
                 WHEN rating > 4 THEN 4
                 WHEN rating > 3.5 THEN 3
                ELSE 2
                END ) > ALL(SELECT
GET_PETS_TAKEN_CARE_BY(NEW.caretaker_username,t.day::date)
                        FROM generate_series(NEW.start_date, NEW.end_date,
interval '1 day') AS t(day)));
    IF NOT able THEN
        RAISE NOTICE 'Caretaker already full on one of the days! Auto rejecting
bid';
        NEW.status := 'REJECTED';
    ELSE
        --calculate total price of the bid
        NEW.total_price :=
            (SELECT price * (DATE_PART('day',NEW.end_date::timestamp -
NEW.start_date::timestamp) + 1)
                FROM CareTakerPricing C, Pet P
                WHERE NEW.pet_name = P.pet_name AND NEW.owner_username =
P.owner_username
                    AND P.pet_type = C.pet_type AND C.username =
NEW.caretaker_username);
        IF fulltime THEN
            --autoaccept if fulltime and able
            NEW.status := 'ACCEPTED';
        END IF;
    END IF;
    RETURN NEW;
END; $$;

CREATE TRIGGER CHECKBID BEFORE INSERT ON Bids
FOR EACH ROW EXECUTE PROCEDURE CHECKBEFOREINSERTBID();
```

Code for the Get_rating function:

```
--Get a rating of a particular caretaker
CREATE OR REPLACE FUNCTION GET_RATING(username VARCHAR)
    RETURNS NUMERIC(3,2) LANGUAGE plpgsql AS
    $$ BEGIN
        RETURN (SELECT CASE
                    WHEN COUNT(*) = 0 THEN 3.00
                    ELSE AVG(rating)::numeric(3,2)
                END
            FROM Bids
            WHERE caretaker_username = username AND rating IS NOT NULL);
    END;
```

Trigger 2

This trigger happens whenever the bids has been updated and whenever there is a new rating for a full timer. When there is a new rating given, we would update the price of all the pets by a caretaker based on his new rating. This is assisted with the get_price function. However, as part timers can specify their own prices as compared to full timers whose price is determined by the base price and their rating, they are unaffected. Below are the code

```
--set new prices whenever rating is updated
CREATE OR REPLACE FUNCTION SET_NEW_PRICES()
    RETURNS TRIGGER LANGUAGE plpgsql
    AS $$
    DECLARE NEW_RATING NUMERIC(3,2);
    BEGIN
        IF NEW.rating IS NOT NULL AND (SELECT is_fulltime FROM Caretaker C Where
C.username = NEW.caretaker_username) THEN
            NEW_RATING := GET_RATING(NEW.caretaker_username);
            Update CareTakerPricing
            SET price =
                CASE
                    WHEN username = NEW.caretaker_username THEN
GET_PRICE(pet_type, NEW_RATING)
                ELSE price
                END ;
        END IF;
    RETURN NEW;
    END;
$$;

CREATE TRIGGER UPDATE_PRICE AFTER UPDATE OR INSERT ON Bids
    FOR EACH ROW EXECUTE PROCEDURE SET_NEW_PRICES();
```

This is the code for the get_price function.

```
--get the default price of a pet by a caretaker of a particular rating
CREATE OR REPLACE FUNCTION GET_PRICE(ptype VARCHAR, rating NUMERIC(3,2))
    RETURNS NUMERIC(3,2) LANGUAGE plpgsql AS $$
        BEGIN
            RETURN (SELECT CASE  WHEN rating>=4.5 THEN P.base_price*1.5
                                WHEN rating >= 4 THEN P.base_price*1.25
                                ELSE P.base_price
                                END
                FROM PetTypes P
                WHERE P.pet_type = ptype);
        END;
$$;
```

Trigger 3

This trigger happens before the caretaker applies for leave.

We check if he already accepted or completed any bids on the day. If there are any, the leave is disapproved and an exception is thrown. Afterwards, we check if the caretaker is full timer and if he is, whether he can afford to take more leave by checking if the number of available days for the year will drop below 300. If it is, an exception is thrown. Next we auto reject all pending bids and raise a notice on how many bids have been auto rejected. Below is the code:

```
--check if there is any bids before allowing leave
CREATE OR REPLACE FUNCTION CHECKBEFORELEAVE()
    RETURNS TRIGGER LANGUAGE plpgsql
  AS $$
  DECLARE initial INTEGER;
    BEGIN
        -- check if he already have bids on the day
        IF (SELECT EXISTS (SELECT 1
                        FROM Bids
                        WHERE OLD.username = caretaker_username AND (status =
'ACCEPTED' OR status = 'COMPLETED')
                        AND OLD.date >= start_date AND OLD.date <= end_date)) THEN
        RAISE EXCEPTION 'YOU CANNOT APPLY FOR LEAVE! YOU HAVE AN ACCEPTED BID IN
THIS DAY!';
        END IF;
        IF ((SELECT C.is_fulltime FROM CareTaker C WHERE C.username =
OLD.username) AND
            (SELECT COUNT(*) <= 300 AS C FROM CareTakerAvailability A
            WHERE A.username = OLD.username AND DATE_PART('year', OLD.date) =
DATE_PART('year', A.date))) THEN
        RAISE EXCEPTION 'YOU CANNOT APPLY FOR ANYMORE LEAVE!';
        END IF;
        --autoreject bids if you apply for leave
        initial := (SELECT COUNT(*) FROM Bids WHERE status = 'PENDING');
        Update Bids
        SET status = 'REJECTED'
        WHERE status = 'PENDING' AND OLD.username = caretaker_username AND
OLD.date >= start_date AND OLD.date <= end_date;

        RAISE NOTICE 'REJECTING % Bids', initial - (SELECT COUNT(*) FROM Bids
WHERE status = 'PENDING');
        RETURN OLD;
    END; $$;

CREATE TRIGGER CHECKBEFOREDELETINGAVAILABILITY BEFORE DELETE ON
CareTakerAvailability
```

```
    FOR EACH ROW EXECUTE PROCEDURE CHECKBEFORELEAVE();
```

7. Complex Queries

We have many complex queries in our database. Here are the 3 most complex queries.

Query 1

This query finds all the caretaker and its rating that are available in all days from a particular start date and enddate and is able to care for a particular pet.

First, we do a join with all the caretakers username together with its rating and the maximum number of pets he is taking care of within the timeframe. To get the maximum number of pets he is taking care of, we first take all the bids and join all the bids with all the days from the queried start and end date. Afterwards, we do a filtering such that the bids are either accepted or completed and that the start date and end date of the bid falls within the day. Next we group them up by caretaker username and day and then do a count of the number of bids for each day. This would be the number of pets a caretaker is currently going to take care of or already took care of in a given day. Afterwards, we group them up by caretaker username and get the maximum of all those days. Since there are caretakers who have 0 pets taken care of within the timeframe, we have to do a coalesce to finally get the table of all the caretakers and the maximum pet they take care of in those days. The other subquery includes getting the rating of all the caretakers. This is done by grouping up the bids by caretakers and averaging the non null ratings. Finally, as some caretaker may not have ratings, we would give them the default of 3. After we have these 2 subqueries, we join them on the username. Next we can then filter out all the caretakers whose max is already equal to the maximum number of pets they can take care of based on their rating. Finally, we filter out the caretakers who are unable to care for the pet. We then present the final list ordered by their rating and their username. Below is the code for the query.

```
    //inputs: 1. startdate, 2. enddate, 3.petowner, 4. petname
    get_top_available_caretaker: 'SELECT caretaker_username, rate FROM 
ALLAVAILABLE($1, $2, $3, $4)',
```

```
CREATE OR REPLACE FUNCTION ALLAVAILABLE(sdate DATE, edate DATE, puser VARCHAR, 
pname VARCHAR)
RETURNS TABLE(caretaker_username VARCHAR, rate NUMERIC(3,2))
LANGUAGE plpgsql
AS $$
BEGIN
RETURN QUERY
    SELECT pMax.n, R.rat
        FROM (SELECT C2.username AS n, COALESCE(pMAX1.m1, 0) AS m
            FROM CareTaker C2 LEFT OUTER JOIN
            (SELECT CP.cname AS n1, MAX(CP.c) AS m1
                FROM (SELECT B.caretaker_username AS cname, t.day AS d, COUNT(*) 
AS c
                        FROM Bids B, generate_series(sdate, edate, interval '1 
day') AS t(day)
                        WHERE (status = 'ACCEPTED' OR status = 'COMPLETED') AND 
(t.day BETWEEN B.start_date AND B.end_date)
```

```
                        GROUP BY B.caretaker_username, t.day
                ) AS CP
                GROUP BY CP.cname
            ) AS pMAX1 ON C2.username = pMAX1.n1
            ) AS pMAX,
            (SELECT  C4.username AS n, COALESCE(R1.ra, 3.00) AS rat
                    FROM CareTaker C4 LEFT OUTER JOIN
                    (SELECT B2.caretaker_username AS n1,
AVG(B2.rating)::numeric(3,2) AS ra
                    FROM Bids B2
                    WHERE rating IS NOT NULL
                    GROUP BY B2.caretaker_username) AS R1
                    ON R1.n1 = C4.username
                    )
                    AS R
        WHERE pMax.n = R.n AND pMax.m < (SELECT CASE
                    WHEN (SELECT C3.is_fulltime FROM CareTaker C3 WHERE
C3.username = pMAX.n) THEN 5
                    WHEN r.rat > 4.5 THEN 5
                    WHEN r.rat > 4 THEN 4
                    WHEN r.rat > 3.5 THEN 3
                    ELSE 2
                    END) AND NOT EXISTS (SELECT 1
                                FROM generate_series(sdate, edate, interval '1
day') AS t2(day)
                                WHERE NOT EXISTS ( SELECT 1
                                                    FROM CareTakerAvailability A
                                                    WHERE A.date = t2.day AND
A.username = pMAX.n
                                                    )
                                )
                AND EXISTS (
            SELECT 1
            FROM pet P5, CareTakerPricing C5
            WHERE pname = P5.pet_name AND puser = P5.owner_username AND
P5.pet_type = C5.pet_type AND pMAX.n = C5.username
            )
        ORDER BY R.rat DESC, pMAX.n ASC
    ;
 END;$$;
```

Query 2

This query aggregates and finds which month of the year has the highest number of jobs and its respective job count. It takes in a date input with the format YYYY-MM-DD from the user. This date input helps to specify which year the user is querying for. It then truncates the data input and retrieves the year. The reason as to why it did not just take in a year number instead of a date format is because the frontend needed a calendar date input and date validation. The conditional clause for a bid to be considered a job is if its status is 'ACCEPTED'. It first tries to aggregate and find out what is the highest number of jobs for the year. Thereafter, it matches again to find out which month has that count. It then returns the month data in a 'Mon yyyy' format along with the count. An example would be <'August 2018', 500>.

Initially, the query was designed to use a CTE. After looking at the possibility of substitution, the CTE was removed and replaced with a longer query.

```
month_highest_jobs: 'SELECT TO_CHAR(b.month :: DATE, \'Mon yyyy\') AS month,
b.count ' +
        'FROM (SELECT COUNT(*), cast(date_trunc(\'month\', Bids.start_date) as
date) AS month ' +
        '      FROM Bids ' +
        '      WHERE status = \'ACCEPTED\' ' +
        '        AND start_date BETWEEN cast( ' +
        '              date_trunc(\'year\', to_date($1, \'YYYY-MM-DD\')) as
date) ' +
        '          AND (cast(date_trunc(\'year\', to_date($1, \'YYYY-MM-DD\'))
+ ' +
        '                  interval \'1 year\' as date)) ' +
        '      GROUP BY month) AS b ' +
        'WHERE b.count = (SELECT max(b2.count) ' +
        '                 FROM (SELECT COUNT(*) ' +
        '                       FROM Bids ' +
        '                       WHERE status = \'ACCEPTED\' ' +
        '                         AND start_date BETWEEN cast( ' +
        '                               date_trunc(\'year\', to_date($1,
\'YYYY-MM-DD\')) as date) ' +
        '                           AND (cast(date_trunc(\'year\', to_date($1,
\'YYYY-MM-DD\')) + ' +
        '                                   interval \'1 year\' as date)) '
+
        '                       GROUP BY cast(date_trunc(\'month\',
Bids.start_date) as date)) AS b2)',
```

Query 3

This query is used to find all the monthly salaries of a given caretaker within the year of 2020

```
    get_salary_record: 'SELECT YR.y, MTH.m, GET_SALARY($1, YR.y, MTH.m) FROM
generate_series(2020, 2020) as YR(y),  generate_series(1,12) as MTH(m) ORDER
BY Mth.m ASC'
```

It calls the get salary function which is shown from below.

```
CREATE OR REPLACE FUNCTION GET_SALARY(caretaker_name VARCHAR, yr INTEGER, mth
INTEGER)
RETURNS NUMERIC(10,2) LANGUAGE plpgsql
AS $$
DECLARE var RECORD;
DECLARE pet_days INTEGER;
DECLARE extra_pay NUMERIC(10,2);
BEGIN
    IF (yr < 2000 OR mth > 12 OR mth < 1) THEN
        RAISE EXCEPTION 'INVALID INPUT';
    END IF;
    IF (SELECT NOT EXISTS(
            SELECT 1
            FROM CareTaker
            WHERE caretaker_name = username)) THEN
        RAISE EXCEPTION 'NO SUCH CARETAKER!';
    END IF;
    IF (SELECT is_fulltime FROM CareTaker WHERE caretaker_name = username)
THEN

        pet_days := 0;
        extra_pay := 0;
        FOR var IN SELECT total_price, (DATE_PART('day', end_date::timestamp -
start_date::timestamp) + 1) AS d
                FROM Bids
                 WHERE caretaker_name = caretaker_username AND (status =
'ACCEPTED' OR status = 'COMPLETED')
                 AND EXTRACT(MONTH FROM start_date) = mth AND EXTRACT(YEAR
FROM start_date) = yr
                 ORDER BY start_date ASC, (DATE_PART('day',
end_date::timestamp - start_date::timestamp) + 1) ASC
        LOOP
            IF (pet_days > 60) THEN
                extra_pay = extra_pay + var.total_price * 0.8;
            ELSEIF ((pet_days + var.d) > 60) THEN
                extra_pay = extra_pay + (var.total_price/var.d) * (pet_days +
var.d - 60);
            ELSE
                extra_pay = extra_pay;
            END IF;
            pet_days = pet_days + var.d;
        END LOOP;

        RETURN 3000 + extra_pay;
    ELSE
        RETURN (SELECT CASE
                        WHEN COUNT(*) = 0 THEN 0
                            ELSE 0.75 * SUM(total_price)
                            END
                        FROM Bids
                         WHERE caretaker_name = caretaker_username AND (status =
'ACCEPTED' OR status = 'COMPLETED')
                         AND EXTRACT(MONTH FROM start_date) = mth AND EXTRACT(YEAR
FROM start_date) = yr);
    END IF;
END; $$;
```

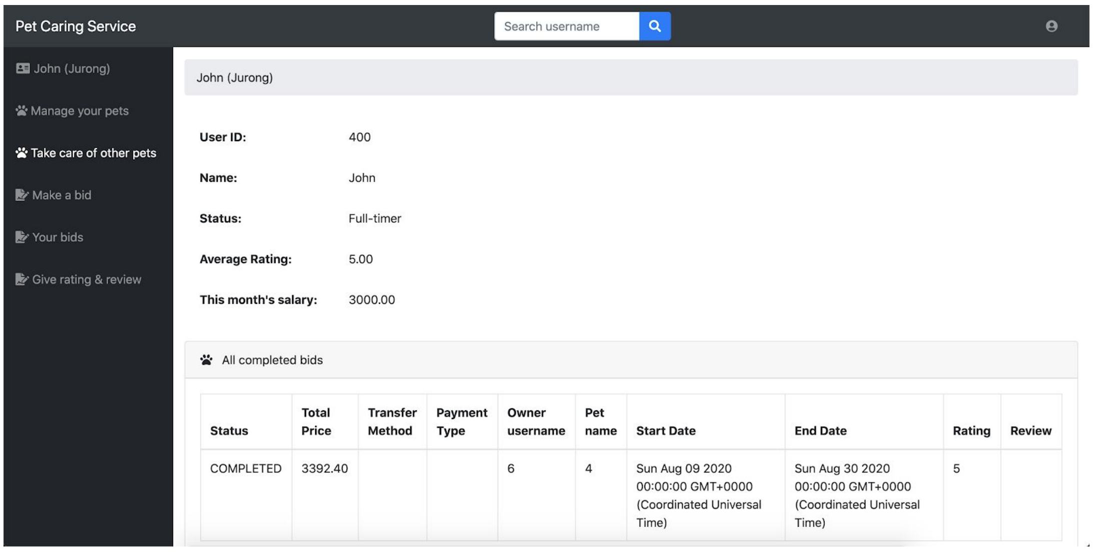
Firstly, we check if the caretaker is a full timer or not. If it is, we then get all the rows that have been accepted or completed whose start date falls within the month sorted by start date and then price per day. Next, we loop through the rows to do the calcalcuation. Upon reaching a pet day of 60, part of the price will be added as extra and the rest of the price will be added to the extra. His monthly salary is thus the base price and the extra price. For the part timer it is done by just taking all his prices and multiplying it by 0.75.

8. Software Specifications

We use Express 4 as our main full stack framework for our application. Passwords are hashed and salted using the bcrypt npm package before being stored into the database. For rendering, we used Embedded JavaScript templates (EJS) to render data passed in from Express's route methods. All of these run on a server-side implementation of JavaScript known as NodeJS. For our frontend, we primarily used Bootstrap and jQuery to provide an elegant user interface. These tools are supported by MomentJS used for handling some of the date time input.

We made use of GitHub as our version control tool to commit and revert changes. These changes are then deployed to Heroku using a Procfile and git push. Heroku automatically detects the buildpack needed for our deployment. This was hosted with Heroku with Heroku Postgres add-on.

9. Screenshots of Application



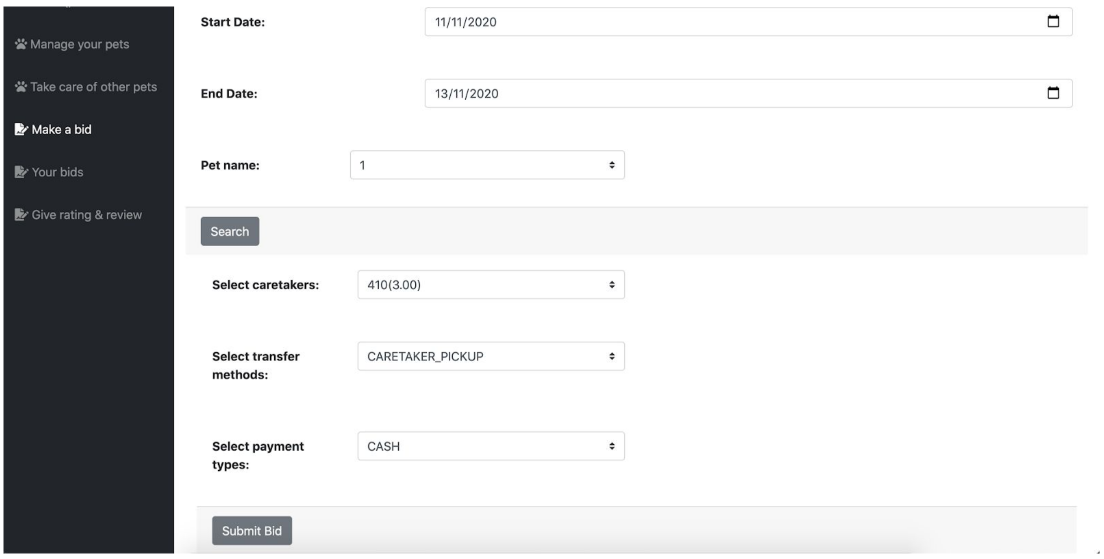
John (Jurong)

User ID:	400
Name:	John
Status:	Full-timer
Average Rating:	5.00
This month's salary:	3000.00

All completed bids

Status	Total Price	Transfer Method	Payment Type	Owner username	Pet name	Start Date	End Date	Rating	Review
COMPLETED	3392.40			6	4	Sun Aug 09 2020 00:00:00 GMT+0000 (Coordinated Universal Time)	Sun Aug 30 2020 00:00:00 GMT+0000 (Coordinated Universal Time)	5	

The above screenshot shows the page of a caretaker.



Start Date:	11/11/2020
End Date:	13/11/2020
Pet name:	1
Select caretakers:	410(3.00)
Select transfer methods:	CARETAKER_PICKUP
Select payment types:	CASH

The above screenshot shows the bidding page.

10. Summary of Lessons Learned

When we first started this project, it was the first time most members experienced web development. Hence, there were a lot of uncertainties on how to build a web from scratch, including integrating with databases. We had no idea how to build a website from scratch. After making use of Professor Adi's code from Github as a base and figuring out how the codebase works, we started to get familiarised with web development as we build our features.

Deploying the codebase to Heroku was also a concern as none of us have experienced deploying a codebase with databases before. We went with working on the features on localhost first, then deployed to Heroku at the end after most of the features on localhost are done.
Furthermore, we had to do research on how to write functions and triggers on SQL to maintain data validity and how to write complex queries to suit the application. We also learnt how to better design our databases.

From this project, we have learned the basics of web development, which includes using git, using node package manager, integrating postgres with web development, making API calls, basics of coding in HTML and JavaScript in EJS templates, and make UI changes with bootstrap. We also learnt to make use of env to handle secret details of the database, so that the password of the database would not be made public.