CS2102 Project Report

Team 47

Ng Shi Xuan, Aerin	A0205054U
Alyssa Nah Xiao Ting	A0202400J
Daniel Seah Jieh Tzen	A0180335L
Ng Chee Heng	A0205454L
Isabella Cheong Xiao Xuan	A0185030R

1. Project Responsibilities

Ng Shi Xuan, Aerin: Back End, Generate Test Data
Alyssa Nah Xiao Ting: Back End, Generate Test Data
Daniel Seah Jieh Tzen: SQL Queries, Triggers, Interesting Queries
Ng Chee Heng: Front End
Isabella Cheong Xiao Xuan: Back End, SQL Queries

2. Description

The Pet Caring Service (PCS) application provides different pet care taking services with a range of functionalities. This portal helps to connect the pet owners and caretakers with both parties' common interests aligned. Searching for the ideal caretaker becomes a lot easier with the PCS, providing a lot of flexibility to meet the needs of the caretaker and pet owner's requirements and busy schedule.

Data Requirements

In order to have a seamless experience with the PCS application, the input data requirements include:

1. Username and password must be at least 6 characters long.
2. Username created has to be unique from other users.
3. Email used to create an account has to be unique from other users.

Functionalities

The functionalities listed below are what the PCS application offers to help pet owners and caretakers alike in searching for the ideal candidate. The PCS administrator also has full control over managing the website, including access rights to manage the price, track the performance of caretakers etc. given to the administrators.

The main functions that this PCS Application offer include:

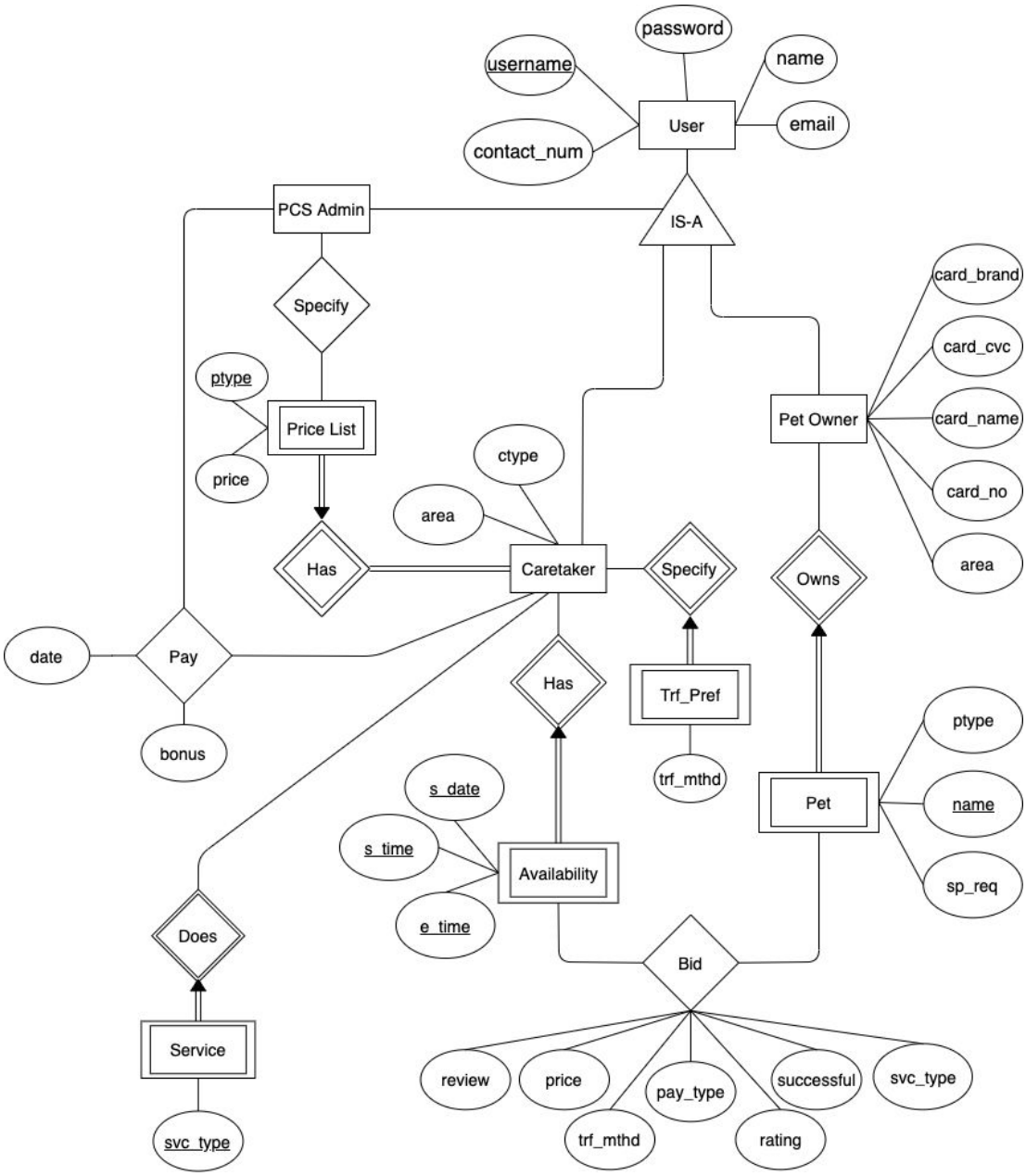
Caretakers / Pet Owners	PCS Administrator
1. Searching for a Caretaker a. Can find caretakers in the pet owner's area 2. Pet owners can bid for Caretakers. 3. The ability for pet owners to become caretakers. 4. Adding user's own information to aid in the searching process. 5. Adding user's own pet information 6. View of "My Bookings" and "My Bids" to keep track of bookings made and bids submitted to secure respective caretakers. 7. For part-time caretakers, they can submit their available dates to work through the platform. 8. Full-time Caretakers can apply for leave if requirements are met. 9. Caretakers can view the current bids made by Pet owners that request for their services and select the jobs they want to accept (up to 5). 10. Pet owners can add their credit cards under their profile. 11. Pet owners can view and leave reviews and ratings for caretakers.	1. View total number of Pets taken care of in the month. 2. View total salary to be paid to all Caretakers for the given month. 3. Filter for specific Caretaker salaries and their bonuses. 4. View underperforming Caretakers a. Underperforming refers to Caretakers with an average rating < 2.4 and number of jobs taken is less than one-third of their availability slots declared. 5. View total number of underperforming Caretakers. 6. View the month with the highest number of jobs taken up by Caretakers. 7. Edit PCS Admin Profile.

Additional functionalities included in the application include:

1. View under performing Caretakers.
    a. The system allows the PCS admin to view the underperforming Caretakers for the month. This helps to gauge which Caretakers have not been delivering the best service to the customers.
2. Caretakers can offer more than 1 type of service for the pets they care for.
    a. Caretakers can be flexible in the number of services they wish to offer, as long as the dates do not clash.

3. Caretaker and Pet owners can input the area in which they stay, so as to allow Pet Owners to better choose their preferred Caretaker.
    a. This makes it easier to choose caretakers nearby the pet owners.
4. Users are able to input pet type and their desired care taker's average ratings in order to know PCS's statistics of average price for the caretaker.

3.ER Data Model



Entity/Relationship Constraints:

Entity/Relationship	Constraints
User	ER Constraints ● Users are identified by their username. ● Users also have the attributes of name, password, contact number and email. Constraints Not Captured by ER Diagram ● A user cannot be a PCS Admin and Pet Owner / Caretaker at the same time but can be both a Pet Owner and a Caretaker ● User's passwords are hashed and salted for security purposes using Bcrypt. ● User's email should be unique.
Pet Owner	ER Constraints ● Pet Owner is identified by his / her username ● Pet Owner also have an area, card cvc number, card name and card number. Constraints Not Captured by ER Diagram ● A pet owner can be a caretaker at the same time. However, they are not allowed to be a caretaker of their own pet. ● A Pet Owner can be a Caretaker at the same time. ● Pet Owner's card number should be unique.
Caretaker	ER Constraints ● Caretaker is identified by his / her username ● Caretakers also have a caretaker type (either 'Full Time' or 'Part Time') and area. Constraints Not Captured by ER Diagram ● Caretaker is either a Full Time or Part Time Caretaker, having a covering and not overlapping constraint. ● Part-time caretakers will receive 75% of their service prices as their salary. ● Full-time caretakers will receive up to $3000 for 60 pet-days of work, and 80% of their additional service prices beyond that as their bonus.
PCS Admin	ER Constraints ● PCS Admin is identified by his / her username.

	Constraints Not Captured by ER Diagram ● PCS Admin sets the base price for each pet.
Pet	ER Constraints ● Pet is identified by its pet owner username and its own name, they also have a pet type and special requirements. ● Pet's name may also be their nickname, it is up to the Pet Owner's decision.
Service	ER Constraints ● Caretaker's service is identified by the caretaker username and service type. Constraints Not Captured by ER Diagram ● Caretaker's services can be Pet Grooming and/or Full Day Pet Care.
Specify	ER Constraints ● PCS Admin can specify the caretaker's daily price and is identified by the PCS Admin username, caretaker username and the pet type. Constraints Not Captured by ER Diagram ● The default price for each pet is assumed to be $5.00. ● If the PCS Admin specifies a price for a specific Caretaker, the Caretaker's price list will update to reflect the specified price
Price List	ER Constraints ● Caretaker can specify their daily price and is identified by caretaker username and pet type. They also have a price. Constraints Not Captured by ER Diagram ● Caretaker cannot take care of animals they didn't specify. ● If the full-time caretaker did not set the price, they are paid based on the PCS stated price. ● Caretakers are free to set their own daily price, but not below the daily base price stated by the PCS Admin.
Availability	ER Constraints ● Caretaker's availability is identified by the start date, start time, ending time of shift and his / her own username. Constraints Not Captured by ER Diagram ● Full Time Caretakers are assumed to work from 08:00 to 20:00 for every pet-day.

	● Part Time Caretakers can indicate their availability by choosing slots of 2 hours time interval between 08:00 to 20:00.
Transfer Preference	ER Constraints ● Caretaker can specify their transfer preference Constraints Not Captured by ER Diagram ● Transfer Preference of the Caretaker can be any mode of transport such as car, bike, cycle or walk.
Pay	ER Constraints ● For each pay, the bonus paid and the date are recorded. Constraints Not Captured by ER Diagram ● Bonus amount appropriate to the average rating of the caretaker is determined by the PCS Admin ● Full-time caretakers are paid up to $3000 per month for up-to 60 pet days. Beyond 60 pet days, the additional days are paid only 80% of the pet price of the caretakers (i.e the number of days x daily pet price stated by the Caretaker.) ● For Part-timers, they are paid 75% of the price all the time, regardless of the number of days worked. Hence, there is no cap. ● Salary is paid on a monthly basis to Part-Timers and Full-Timers.
Bid	ER Constraints ● Each bid is identified by the pet's name, pet owner's username, caretaker username, start date and start time. They also have an end time, review, price, transfer method, payment type, rating, service type and whether the bid is successful. ● Pet owners can only make bids for caretakers whose availability period is within the criteria. ● Caretaker's ratings are from a range of 0 to 5. ● Reviews and Ratings are not necessary. ● Only one mode of transport should be specified for each transaction. Constraints Not Captured by ER Diagram ● A full-time caretaker can only take care of a maximum of 5 pets at any time, and the bid will not be processed for that specific time if that full-time caretaker is taking care of 5 pets for that time. ● A part-time caretaker can only take care of a maximum of 2 pets at any time if their average rating is below 4, and the bid will not be processed for that specific time if that part-time caretaker is taking care of 2 pets for that time.

	• Pet Owners can hire multiple Caretakers, and Caretakers can have multiple Pet Owner clients, up to a maximum of 5 concurrently. • Not every Caretaker has a pet owner client.

Implemented Suggested Improvements:

- Full-time Caretaker can take leave for any period they desire as long as they have worked for consecutive 150 days.
- Part-Time Caretaker will be able to declare their availability 1 week in advance, for the next coming week. This is to reduce the number of no-shows by the Caretaker in the event they declare their schedule too early and have a change of plans without declaring on the PCS.
- When a bid is successful, Pet Owners are assumed to pay via credit card if they have a credit card registered under their account. Else, they will pay via cash. This is to encourage cashless transactions.

Un-implemented Suggested Improvements:

- Pet owners who do not have their bids accepted by one month after the time of the start date will be cleared. This can be done using caching and javascript. However, it is not implemented due to time constraints.

Un-implemented constraints:

- Emergency leaves for full-timer.
- Categorized ratings can be implemented for each transaction.

4.Relational Schema

```
CREATE TABLE users (
 username VARCHAR(255) PRIMARY KEY,
 contact_num CHAR(8) NOT NULL,
 password VARCHAR(255) NOT NULL,
 name VARCHAR(255) NOT NULL,
 email VARCHAR(255) NOT NULL UNIQUE
);

CREATE TABLE pet_owner (
 username VARCHAR(255) PRIMARY KEY REFERENCES users(username) ON DELETE
cascade,
 card_cvc VARCHAR(10),
 card_name VARCHAR(255),
 card_no VARCHAR(255) NOT NULL UNIQUE,
```

```
 card_brand VARCHAR(255),
 area VARCHAR(255)
);

CREATE TABLE care_taker (
 username VARCHAR(255) PRIMARY KEY REFERENCES users(username) ON DELETE
cascade,
 ctype VARCHAR(255),
 area VARCHAR(255)
);

CREATE TABLE pcs_admin (
 username VARCHAR(255) PRIMARY KEY REFERENCES users(username) ON DELETE
cascade
);

CREATE TABLE owns_pet (
 pet_owner_username VARCHAR(255) REFERENCES pet_owner(username) ON DELETE
cascade,
 name VARCHAR(255),
 ptype VARCHAR(255),
 sp_req VARCHAR(255),
 PRIMARY KEY (pet_owner_username, name)
);

CREATE TABLE specify_trf_pref (
 care_taker_username VARCHAR(255) PRIMARY KEY REFERENCES
care_taker(username) ON DELETE cascade,
 trf_mthd VARCHAR(255)
);

CREATE TABLE pay (
 care_taker_username VARCHAR(255) REFERENCES care_taker(username) ON
DELETE cascade,
 pcs_admin_username VARCHAR(255) REFERENCES pcs_admin(username) ON DELETE
cascade,
```

```
 bonus NUMERIC,
 date DATE,
 PRIMARY KEY (care_taker_username, pcs_admin_username, date)
);

CREATE TABLE has_price_list (
 care_taker_username VARCHAR(255) REFERENCES care_taker(username) ON
DELETE cascade,
 ptype VARCHAR(255),
 price NUMERIC DEFAULT 5,
 PRIMARY KEY (care_taker_username, ptype)
);

CREATE TABLE specify (
 pcs_admin_username VARCHAR(255) REFERENCES pcs_admin(username),
 care_taker_username VARCHAR(255),
 ptype VARCHAR(255),
 price NUMERIC,
 PRIMARY KEY (pcs_admin_username, care_taker_username, ptype),
 FOREIGN KEY (care_taker_username, ptype) REFERENCES
has_price_list(care_taker_username, ptype)
);


CREATE TABLE has_availability(
 care_taker_username VARCHAR(255) REFERENCES care_taker(username) ON
DELETE cascade,
 s_date DATE,
 s_time TIME,
 e_time TIME,
 PRIMARY KEY(s_date, s_time, e_time, care_taker_username),
 CHECK(s_time < e_time),
 CHECK(s_date >= CURRENT_DATE)
);
```

```
CREATE TABLE does_service(
 care_taker_username VARCHAR(255) REFERENCES care_taker(username) ON
DELETE cascade,
 svc_type VARCHAR(255),
 PRIMARY KEY(care_taker_username, svc_type)
);

CREATE TABLE bid(
 care_taker_username VARCHAR(255),
 s_date DATE,
 s_time TIME,
 e_time TIME,
 name VARCHAR(255),
 pet_owner_username VARCHAR(255),
 review VARCHAR(255),
 price NUMERIC,
 trf_mthd VARCHAR(255),
 pay_type VARCHAR(255),
 rating NUMERIC,
 successful BOOLEAN,
 svc_type VARCHAR(255),
 PRIMARY KEY (name, pet_owner_username, care_taker_username, s_date,
s_time),
 FOREIGN KEY (s_date, s_time, e_time, care_taker_username) REFERENCES
has_availability(s_date, s_time, e_time, care_taker_username),
 FOREIGN KEY (pet_owner_username, name) REFERENCES
owns_pet(pet_owner_username, name)
);
```

Constraints enforced by functions:

- To ensure total participation of Caretaker with ‘has_price_list’.
- Full-time caretakers can only apply for leave if they have worked 150 consecutive days prior.

Constraints that can only be enforced by triggers:

- Caretaker pet caring limit: Full-time caretaker can only take care of a maximum of 5 pets at any point of time and a part-time caretaker can only take care of a maximum of 2 pets

at any point of time if their average rating is below 4, if the part-time caretaker has no previous ratings, they will be considered to have an average rating of 0.

- Trigger to maintain IS-A property (for Users): Pet Owner and Caretaker cannot be a PCS Admin.
- Trigger to update Price List when Specify is updated: Price List table is updated to reflect the changes or insertions to the Specify table using a trigger.

5. Whether Database is in BCNF/3NF

1. Care_Taker*: {care_taker_username} -> {password, name, email, contact_num, ctype, area}*
2. Pet_Owner*: {pet_owner_username} -> {password, name, email, contact_num, area, card_cvc, card_name, card_no}*
3. Owns_Pet*: {pet_owner_username, pet_name} -> {ptype, sp_req}*
4. PCS_Admin*: {pcs_admin_username} -> {password, name, email, contact_num}*
5. Has_Price_List: *{care_taker_username, ptype} -> {price}*
6. Specify: *{pcs_admin_username, care_taker_username, ptype} -> {price}*
7. Bid: *{pet_name, pet_owner_username, care_taker_username, s_date, s_time} -> {e_time, review, price, trf_mthd, pay_type, rating, successful, svc_type}*
8. Pay: {*care_taker_username, pcs_admin_username, date} -> {bonus}*
9. Specify_Trf_Pref: *{care_taker_username} -> {trf_mthd}*
10. Does_Service: {*care_taker_username, svc_type} -> {care_taker_username, svc_type}*
11. Has_Availability: *{care_taker_username, s_date, s_time, e_time} -> {care_taker_username, s_date, s_time, e_time}*

For functional dependencies 1- 9 listed above, they satisfy the superkey properties of BCNF based on Zaniolo's definition as all the superkeys are on the left hand side of the functional dependencies.

For functional dependencies 10- 11 listed above, they satisfy the trivial properties of BCNF based on Zaniolo's definition. Every attribute is not transitively dependent on any given candidate keys.

Hence, our database is in BCNF as all functional dependencies satisfy the BCNF properties.

6.Triggers for Complex Constraints

1. Trigger to ensure pet caring limit for Caretaker

```
CREATE OR REPLACE FUNCTION pet_limit_reached()
RETURNS TRIGGER AS
$$
  DECLARE ctx NUMERIC;
```

```
  DECLARE is_fulltime NUMERIC;
  DECLARE avg_rating NUMERIC;
  BEGIN
  SELECT COUNT(*) INTO ctx FROM bid B
  WHERE B.care_taker_username = NEW.care_taker_username
  AND B.successful = TRUE
  AND B.s_date = NEW.s_date
  AND ((B.s_time >= NEW.s_time AND B.s_time < NEW.e_time) OR (B.e_time >
NEW.s_time AND B.e_time < NEW.e_time)); --Check timings
  SELECT COUNT(*) INTO is_fulltime FROM care_taker C WHERE C.username =
NEW.care_taker_username
  AND  C.ctype = 'Full Time';
  SELECT AVG(B2.rating) INTO avg_rating FROM bid B2 WHERE
B2.care_taker_username = NEW.care_taker_username
  AND B2.successful = TRUE;
  IF (is_fulltime > 0 AND ctx >= 5) THEN RETURN NULL; END IF;
  IF (is_fulltime = 0 AND ctx >= 2 AND (avg_rating IS NULL OR avg_rating <
4)) THEN RETURN NULL; END IF;
  RETURN NEW;
  END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER check_pet_limit_reached()
BEFORE INSERT OR UPDATE ON bid
FOR EACH ROW EXECUTE PROCEDURE pet_limit_reached();
```

This trigger is implemented to ensure that the full-time caretaker cannot take care of more than 5 pets at a time, and that a part-time caretaker cannot take care of more than 2 pets at a time if their average rating is less than 4. If the full-time caretaker is currently taking care of 5 pets at any point in time between the bid start time and end time, or in the case of a part-timer, if they are taking care of 2 pets and their average rating is below 4, this trigger will prevent the bid from being added to the bid table and hence prevents the bid from being made for that specific caretaker at that specific stated start and end times.

2. Trigger to maintain IS-A property (for Users)

```
CREATE OR REPLACE FUNCTION not_admin()
RETURNS TRIGGER AS
$$
  DECLARE ctx NUMERIC;
  BEGIN
```

```
  SELECT COUNT(*) INTO ctx FROM pcs_admin P
  WHERE NEW.username = P.username;
  IF ctx > 0 THEN
    RETURN NULL;
  ELSE
    RETURN NEW;
  END IF;
  END; $$
LANGUAGE plpgsql;

CREATE TRIGGER check_pet_owner()
BEFORE INSERT OR UPDATE ON pet_owner
FOR EACH ROW EXECUTE PROCEDURE not_admin();

CREATE TRIGGER check_care_taker()
BEFORE INSERT OR UPDATE ON care_taker
FOR EACH ROW EXECUTE PROCEDURE not_admin();
```

This trigger is implemented to ensure that any PCS Admin cannot be a Pet Owner and/or Caretaker (although they are all using an IS-A relationship to User). The trigger checks upon inserting or updating the `pet_owner` or `care_taker` table, that the inserting username does not exist in the `pcs_admin` table then it will proceed with the insertion/update, else if the inserting username exists in the `pcs_admin` table, it means that the username is already a PCS Admin and hence the insertion/update will return a NULL.

3. Trigger to update Has_Price_List when Specify is updated

```
CREATE OR REPLACE FUNCTION specify_update_has_price_list()
RETURNS TRIGGER AS
$$
  BEGIN
  INSERT INTO has_price_list VALUES (NEW.care_taker_username, NEW.ptype,
NEW.price)
  ON CONFLICT(care_taker_username, ptype)
  DO UPDATE price = NEW.price
  WHERE care_taker_username = NEW.care_taker_username
  AND ptype = NEW.ptype
  END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER specify_update_price_list()
```

```
AFTER INSERT OR UPDATE ON specify
FOR EACH ROW EXECUTE PROCEDURE specify_update_has_price_list();
```

This trigger is implemented to ensure that when the PCS Admin updates/inserts into the `Specify` table, the appropriate change will be reflected in the price list of the Caretaker. This is because we want the PCS Admin to be able to specify a price for the Caretaker. The trigger will on updating or after inserting into the `specify` table, upsert the relevant price values that had been inserted/updated in the specify table, for the specified caretaker and pet type pair in the `has_price_list` table. This ensures that the prices specified by the PCS Admin will be shown in the price list of the Caretaker.

7.Interesting Queries

1. Query to show all Caretaker Salaries for a given month

```
SELECT P.ct_username as ct_username,
  CASE
    WHEN P.pet_days <= 60 THEN P.pet_days * 50
    WHEN P.pet_days > 60 THEN 3000 + P2.bonus * 0.8
  END as pay
FROM (
  SELECT COUNT(*) as pet_days, C.username as ct_username
  FROM bid B JOIN care_taker C ON B.care_taker_username = C.username
  WHERE C.ctype = 'Full Time'
  AND B.successful = TRUE
  AND date_trunc('month', B.s_date) = date_trunc('month', $1::timestamp)
  GROUP BY C.username
) P LEFT JOIN
(
  SELECT SUM(B2.price) as bonus, C2.username as ct_username
  FROM bid B2 JOIN care_taker C2 ON B2.care_taker_username = C2.username
  WHERE C2.ctype = 'Full Time'
  AND B2.successful = TRUE
  AND date_trunc('month', B2.s_date) = date_trunc('month', $1::timestamp)
  GROUP BY C2.username
  ORDER BY MAX(B2.s_date) ASC
  OFFSET 61 ROWS
) P2 ON P.ct_username = P2.ct_username
UNION
SELECT C.username as ct_username, SUM(B.price) * 0.75 as pay
FROM bid B JOIN care_taker C ON B.care_taker_username = C.username
```

```
WHERE C.ctype = 'Part Time'
AND B.successful = TRUE
AND date_trunc('month', B.s_date) = date_trunc('month', $1::timestamp)
GROUP BY C.username;
```

The purpose of this query is to retrieve all the salary to be paid to each Caretaker for any given month. We first have to split the caretakers into two groups, either being “Full Time” or “Part Time”. We calculated each group separately and then unioned the results to obtain the full list of salaries for each caretaker. For the ‘Full Time’ caretaker, they are paid up to $3000 for 60 pet days of work, and 80% of any additional work they do. Therefore we use the CASE WHEN statement to pay full time caretakers with equal or less than 60 petdays the amount of $50 per petday. To obtain the additional bonus work of full time workers, we Group By the ‘care_taker_username’ and order by the date, we then use `offset` to exclude the first 60 rows and hence we are left with the remaining additional bonus pet days. We then take the sum of the prices paid for these pet days and take 80% of it as the bonus for the full time worker. We use a Left Join to join the total petdays P table and the additional petdays price P2 table, in order to keep the entries whereby the caretaker may have less than or equal to 60 petdays, such that they do not receive any bonus. This will give us the salaries for each full time caretaker. We then union this result with the salary for the part-time caretaker, which is 75% of all the prices of their services.

An extension of this query is to obtain the total salary for all the caretakers, which would just be the sum of all the salaries as obtained in the query above. We also have this functionality in our application.

2. Query to show all underperforming caretakers

```
SELECT J1.ct_username as ct_username, J1.num_avail as num_avail,
J2.num_jobs as num_jobs, J2.avg_rating as avg_rating
FROM (
  SELECT H.care_taker_username as ct_username, COUNT(*) as num_avail
  FROM has_availability H
  WHERE date_trunc('month', H.s_date) = date_trunc('month', $1::timestamp)
  GROUP BY H.care_taker_username
) J1 JOIN
(
  SELECT B.care_taker_username as ct_username, COALESCE(COUNT(*), 0) as
num_jobs, COALESCE(AVG(B.rating), 0) as avg_rating
  FROM bid B
  WHERE date_trunc('month', B.s_date) = date_trunc('month', $1::timestamp)
  AND B.successful = TRUE
  GROUP BY B.care_taker_username
  HAVING COALESCE(AVG(B.rating), 0) < 2.5
```

```
) J2 ON J1.ct_username = J2.ct_username
WHERE J2.num_jobs <= (J1.num_avail / 3)
```

This query obtains all the underperforming caretakers for any given month, and displays their username, their number of availabilities their number of jobs performed for that month, and their average rating for the month. Our definition of an ‘underperforming’ caretaker is that, for the month, the number of jobs completed by the caretaker is less than a third of their number available of available days, and their average rating for the month is less than 2.5. The query is split into two subqueries which are then inner joined by the care_taker_username. The first subquery filters the `has_availability` table to only retain the rows for the given month, then groups by the care_taker_username and counts all the rows for that caretaker to obtain the number of available days that caretaker has for the given month. The second subquery filters the `bid` table to only retain the rows for the given month and the bids that were successful, hence they are work that the caretaker did. By using the `having` condition, we can filter the rows whereby the caretaker had an average rating of less than 2.5 for that month. We use coalesce to ensure that a value of 0 is returned in the case that that caretaker has had no work or ratings for that particular month. We then inner join the two subqueries and then filter to find all the caretakers that have worked fewer jobs than a third of their number of available days.

3. Query to find average price per hour of a specific pet type

```
SELECT SUM(J2.price) / SUM(J2.work_hours)
FROM (
  SELECT B1.care_taker_username as ct_username, COALESCE(AVG(B1.rating),
0) as avg_rating
  FROM bid B1
  GROUP BY B1.care_taker_username
  HAVING COALESCE(AVG(B1.rating), 0) >= $2
) J1 JOIN
(
  SELECT B2.care_taker_username as ct_username, B2.price as price,
EXTRACT(HOUR FROM (e_time - s_time)) as work_hours
  FROM bid B2 JOIN owns_pet P
  WHERE P.ptype = $1
  AND B2.successful = TRUE
) J2 ON J1.ct_username = J2.ct_username
```

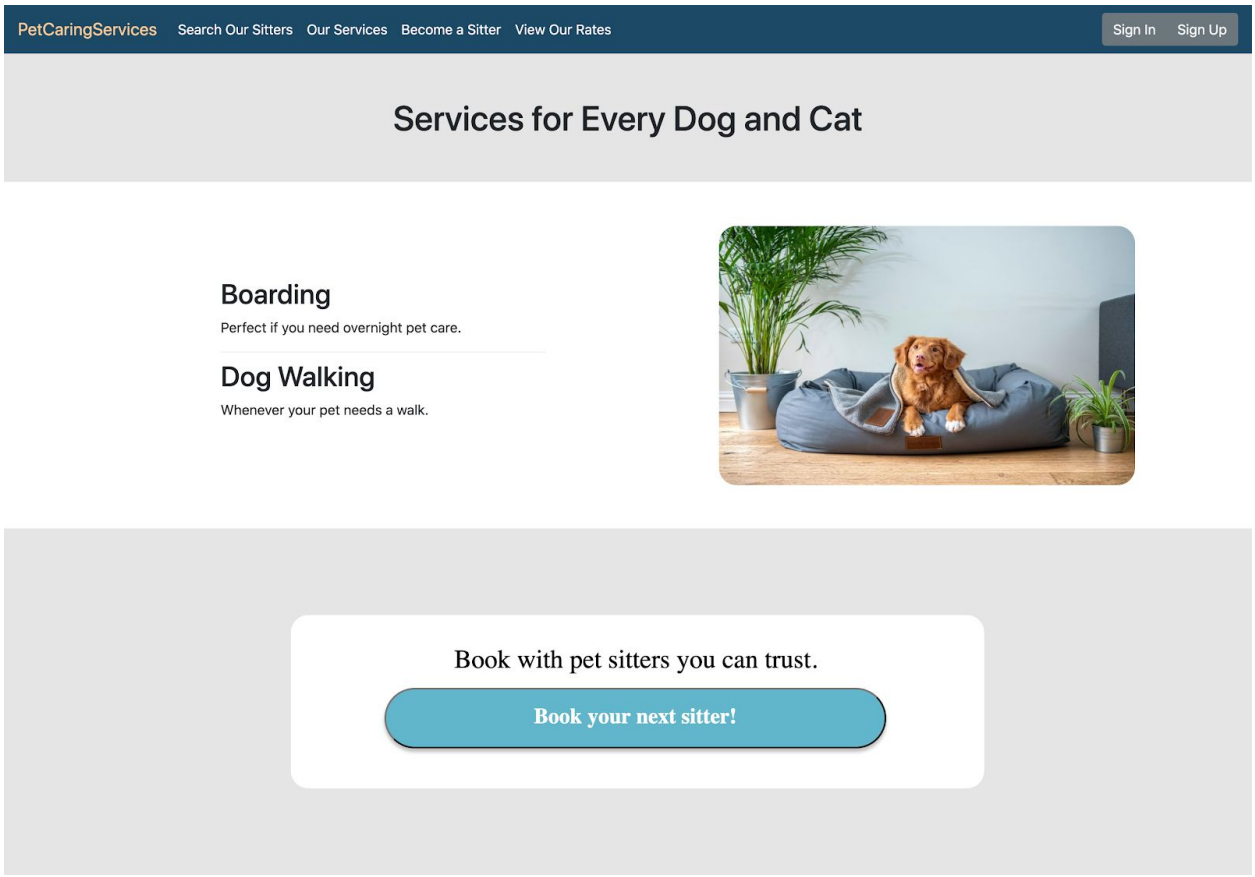
This query obtains the average price per hour for the user specified pet type, and also uses only results from a user specified minimum average rating of caretaker. The query is split into two subqueries which are joined to give the final resulting value. The first subquery is to obtain and filter the average rating of the caretaker, assigning them a value of 0 if they had no ratings. The second subquery is to obtain the price of the successful bid, as well as to calculate the number of hours worked for that bid, we also filter by the specified pet type. We then use an inner join to

join the two subqueries, then we get the average price per hour by taking the total price of all the bids divided by the total hours worked, this value is then returned and displayed for the user to view.

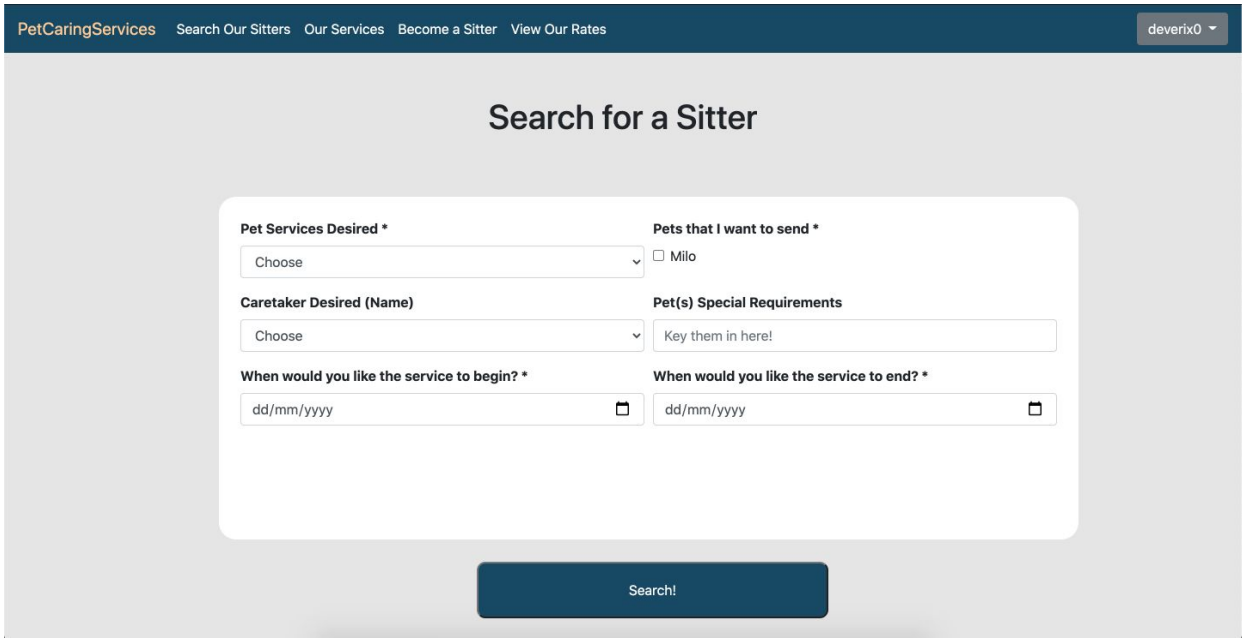
8. Software Framework

- Front End: HTML/CSS
- Back End: Node.js
- Database: PostgreSQL
- Deployment: Heroku

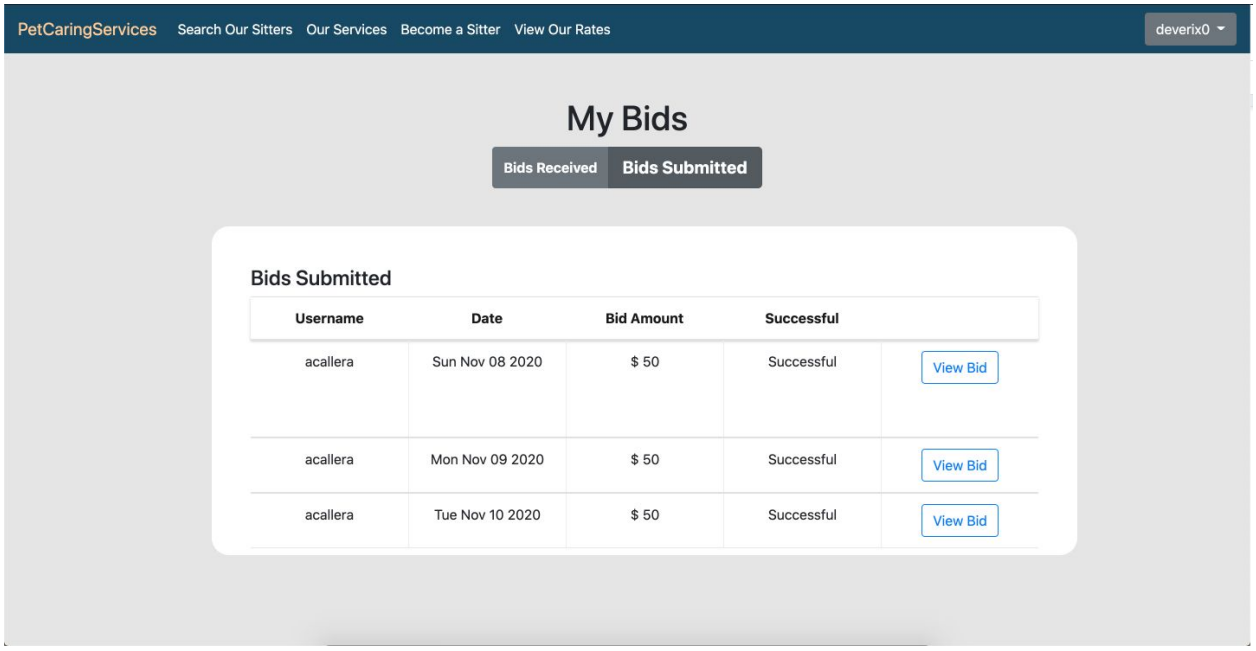
9. User Interface



Home page for Pet Owners and Caretakers, allowing them to Search for Sitters, view Our Services, Become a Sitter (if they are not one), View Our Rates and do user related operations.



Users can input their search criteria entered into the ‘Search Our Sitters’, a list of available caretakers will be displayed for the Pet Owner to choose from and bid for them. Details such as caretakers’s name, username, availability, caretaker type, area, pet transfer preference and ratings will be listed.



The “Bids Submitted” page displays a list of bids that the Pet Owner has submitted.

10. Difficulties Encountered and Lessons Learnt

No.	Difficulties Encountered	Lessons Learnt
1	Getting foreign key error due to database insertions not being in the order as desired	Always check that data has been inserted properly before inserting the next data that may rely on the former
2	Needing intermediate values from database queries	Use Asynchronous functions and Promises to do database queries to make the code more elegant
3	Working with Node.js and Web Development for the first time and the unfamiliarity with it	Learning the proper syntaxes and coding practices as we familiarize ourselves working with Node.js as the core of our backend.
4	Difficulty in tying in the front end and the back in due to lack of experience.	We learnt how to tie SQL in together with Javascript, HTML and CSS in a remotely deployed website. This posed many challenges as the front end, back end and database are more linked than we expected and it involved much collaboration and many changes throughout the entire process. It was a holistic experience and lesson as we learnt different aspects of database development and website development.
5	Developing complex queries to meet the criteria.	The system requirement mainly required simple queries and reading the database to provide information. However, complex SQL queries are more difficult to picture and hence there were challenges in developing them. This taught us how to view complex SQL queries and retrieve specific complex information from the database based on the information required.
6	The separation of tasks when there is a high dependability between them.	The database queries and the website functionalities are very dependent on each other and there is a need to ensure the database queries prepared extracts or returns the information required by the backend (Node.js) calling the queries. Database queries have to be edited along the way whilst implementing the backend in order to make them compatible.
7	Taking into account unexpected input that would crash the webpages.	Doing error handling is very tedious especially since it is very easy for users to enter unexpected input, increasing the amount of constraints that need to be enforced, on top of table constraints. It is best to handle errors along with implementation of the functionalities instead of trying to handle them all at once at the end.

8	Coming up with a realistic website is very tough to envision and realise.	Have to visualize the whole process that a user using the website would have to go through as well as take into account the flexibility of choices that a user expects, as well as the ease of using the website.