CS2102 Database Systems
AY 2020/2021 Semester 1

Project 1: Pet Caring Web Application

Completed By:

Ayush Kumar E0315961
Ong Wei Cheng E0321478
Sean Iau Yang E0407338
Teng Xin Wei E0426182
Umer Siddiqui E0415669

1. A listing of the project responsibilities of each team member.

Team Member Name	Team Member Responsibilities
Ayush Kumar	- ER Diagram - SQL Queries - Functions - Populating DB
Ong Wei Cheng	- ER Diagram - SQL Queries - Triggers - Functions - Populating DB
Sean Iau Yang	- User Interface - User Authentication/Roles - Schema Creation - SQL Queries - Functions - Deployment
Teng Xin Wei	- User Interface - Schema Creation - SQL Queries - Triggers - Functions - Deployment
Umer Siddiqui	- User Interface - Node server setup - SQL Queries - Triggers - Functions - Deployment

2. A description of your application's **data requirements** and **functionalities**. Highlight any **interesting**/non-trivial aspects of your application's functionalities /implementation. List down all the application's **data constraints**.
   The different data requirements and constraints that we have identified are as follows:

   Entity: User
   - Primary key is email
   - has a password and name, both cannot be null
   - Covering constraint (user must be pet owner, caretaker or PCS administrator)
   - Overlap constraint (user can be both pet owner and caretaker)

Entity: Pet Owner

- Primary key is email
- Pet Owner can choose not to bid for any advertisement of CareTaker or multiple advertisements of CareTaker

Entity: Caretaker

- Primary key is email
- Has an overall rating based on the ratings given for his caretaking services
- Covering constraint (caretakers must be full time or part time)
- No overlap constraint (caretakers cannot be both full time and part time)

Entity: Full time Caretaker

- Primary key is email
- Full time Caretaker posts advertisement stating daily price based on PCS admin’s specified base daily price for each pet category and his overall rating
- Should apply for leaves that satisfy the requirement to work a minimum of 2 x 150 consecutive days a year
- Can only take care of a maximum of 5 pets at one time.
- Earns a minimum of 3000/month for up to 60 pet days, 80% of sales thereafter.

Entity: Part time Caretaker

- Primary key is email
- Part time Caretaker posts advertisement stating daily price based on PCS admin’s specified base daily price for each pet category and his overall rating
- Part time Caretaker can take care of up to 2 pets unless they have a good review, which they can take care of at most 5 pets.
- 75% of the caretaker’s transactions is given as salary.

Entity: PCS Administrator

- Primary key is email
- PCS Administrator specifies a list of base daily prices for different pet categories(existential dependency)

Entity: Pet

- Primary key is email and pet name
- Pet has details stated on the website such as special requirements, pet category and its age
- Has identity dependency on pet owner

Entity: Advertisement

- Primary key is caretaker's email, pet category, start date and end date
- States the daily price for taking care of the pet
- Has identity dependency on caretaker

Relation: Bids

- Primary key is pet owner's email, care taker's email, pet category, pet name, start date and end date of advertisement
- Once a bid meets the daily price specified in a care taker's advertisement, it is marked as successful

The general data constraints that we have imposed are:

- Start date needs to be before end date for all tables requiring dates (advertisements, leaves, available_days)
- User's name and password are not null

The different functions that we have included are as follows:

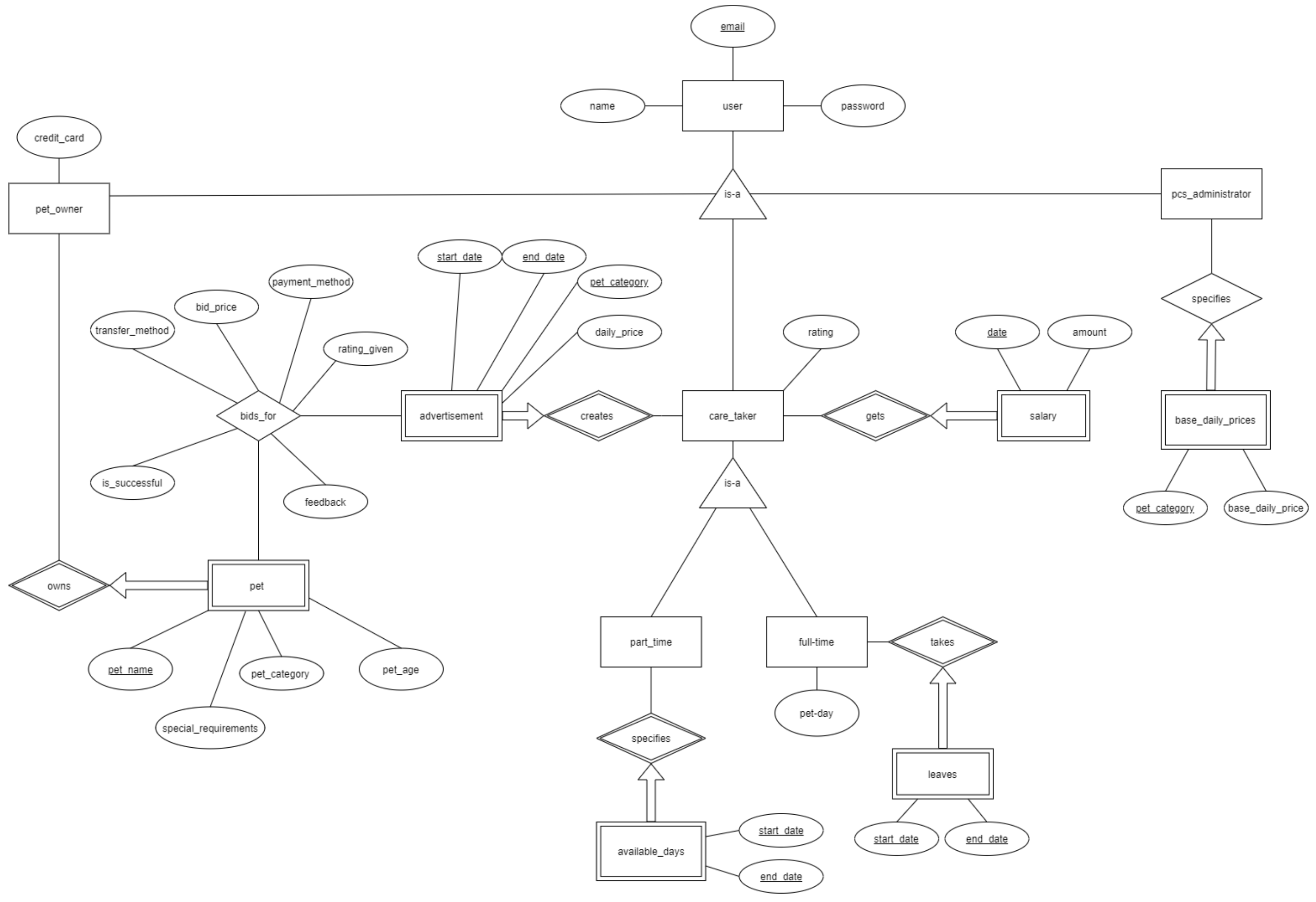
- Check email exists for signup
- Sign up
- Check email exists for sign in
- Verify password
- Sign in
- Create, Retrieve, Update, Delete Profile by all users
- Create, Retrieve, Update, Delete Pet by pet owners
- Create, Retrieve, Update, Delete Advertisements by caretakers
- Create, Retrieve, Update, Delete Availability by part time caretakers
- Create, Retrieve, Update, Delete Leaves by full time caretakers
- Create, Retrieve, Update Bids by pet owners
- PCS Admin specifies full time caretakers' base daily price for different pet categories
- Part time caretaker specifies available days
- Full time caretaker specifies days on leave
- Full time caretaker's pet day increments based on successful bid transactions' start and end dates
- Caretakers get salary for each month
- PCS admin can view all caretaker's past advertisements and successful bids for their services.

Some of the interesting/non-trivial aspects of our application are:

- Automated system to accept bid for advertisement once the bid meets the minimum daily price set by caretaker (base daily price plus bonus based on caretaker's rating)
- Used bcrypt to salt and hash the password before storing into the database. This increases security of users' private information.
- Used JSON web token to initialise each user session upon login for 2 hours before expiry where they are logged out

- Display top 5 performing caretakers for the month (top earners)
- Different users have access to different functionalities based on the type of user they are (pet owner, caretaker, PCS admin)

3. The ER model of your application. If your ER model is too large (i.e., spanning more than a page), you may want to include a single-page simplified ER model (with non-key attributes omitted) before presenting the detailed ER model. Provide justifications for any non-trivial design decisions in your ER model. List down all the application's constraints that are not captured by your ER model.



Application's constraints not captured by ER model:

- User entity satisfies the covering constraint and must be either a pet owner, caretaker or a PCS Admin.
- User satisfies the overlap constraint and any user can assume two different types of roles.
- Caretaker entity satisfies the covering constraint and must be either a part time or full time caretaker.
- Caretaker satisfies no overlap constraint and any caretaker cannot be both a part time and a full time caretaker at the same time.
- Full time caretakers can take care of maximum 5 pets at a time.
- Part time caretakers cannot take care of more than 2 pets unless they have a good rating and they definitely cannot take care of more than 5 pets.

- Pet owners should be able to pay by cash if he/she wishes.
- Full time caretakers need to work for a minimum of 2 x 150 consecutive days.
- Part time caretakers should be able to specify their availability for the current year and the next year.
- Full time caretakers should not be able to apply for leave if they have a pet in their care.
- The salary defined by the PCS Administrator increases with the rating of the caretaker but will never fall below the base price.
- The salary for full time caretakers is $3000 per month for up to 60 pet-days. For any excess pet-day, they will receive 80% of their price as bonus.
- The salary for part time caretaker is 75% of their stated price.

4. The relational schema derived from your ER data model (i.e., show the SQL CREATE TABLE statements and possibly ALTER TABLE if any for all your database tables). List down all the application's constraints that are not enforced by your relational schema (e.g., the constraints that are enforced using triggers, or cannot be enforced at all).

```
CREATE TABLE pet_owners(
 email VARCHAR(255) PRIMARY KEY,
 password VARCHAR (255),
 name VARCHAR(255),
 credit_card VARCHAR(255)
);
CREATE TABLE caretakers(email VARCHAR(255) PRIMARY KEY);
CREATE TABLE pt_caretakers(
 email VARCHAR(255) PRIMARY KEY,
 password VARCHAR (255),
 name VARCHAR(255),
 rating float8 DEFAULT 0,
 FOREIGN KEY(email) REFERENCES caretakers(email) ON DELETE CASCADE
);
CREATE TABLE ft_caretakers(
 email VARCHAR(255) PRIMARY KEY,
 password VARCHAR (255),
 name VARCHAR(255),
 rating float8 DEFAULT 0,
 pet_day integer,
 FOREIGN KEY(email) REFERENCES caretakers(email) ON DELETE CASCADE
);


CREATE TABLE pcs_admins(
 email VARCHAR(255) PRIMARY KEY,
 password VARCHAR (255),
 name VARCHAR(255)
```

```sql
);
CREATE TABLE owns_pets(
 email VARCHAR(255) REFERENCES pet_owners(email) ON DELETE CASCADE,
 pet_name VARCHAR(50),
 special_requirements VARCHAR(255),
 pet_category VARCHAR(255),
 pet_age integer,
 PRIMARY KEY(email, pet_name)
);
CREATE TABLE advertisements(
 pet_category VARCHAR(255),
 start_date date,
 end_date date,
 daily_price NUMERIC,
 email VARCHAR(255) REFERENCES caretakers(email) ON DELETE CASCADE,
 PRIMARY KEY(email, pet_category, start_date, end_date),
 CONSTRAINT "start date needs to be earlier than end date" CHECK
(start_date < end_date)
);
CREATE TABLE specifies_available_days(
 start_date date,
 end_date date,
 email VARCHAR(255) REFERENCES pt_caretakers(email) ON DELETE CASCADE,
 PRIMARY KEY(start_date, end_date, email),
 CONSTRAINT "start date needs to be earlier than end date" CHECK
(start_date < end_date)
);
CREATE TABLE salaries(
 payment_date date,
 payment_amount integer,
 email VARCHAR(255) REFERENCES caretakers(email) ON DELETE CASCADE,
 PRIMARY KEY(payment_date, email)
);
CREATE TABLE takes_leaves(
 start_date date,
 end_date date,
 email VARCHAR(255) REFERENCES ft_caretakers(email) ON DELETE CASCADE,
 PRIMARY KEY(start_date, end_date, email),
 CONSTRAINT "start date needs to be earlier than end date" CHECK
(start_date < end_date)
);
CREATE TABLE specifies(
 pet_category VARCHAR(255),
 base_daily_price NUMERIC,
 pcs_email VARCHAR(255) REFERENCES pcs_admins(email) ON DELETE CASCADE,
 PRIMARY KEY(pet_category)
);
CREATE TABLE bids_for(
 transfer_method VARCHAR(255),
 bid_price NUMERIC,
 payment_method VARCHAR(255),
 rating_given NUMERIC DEFAULT 0,
 is_successful BOOLEAN DEFAULT FALSE,
 feedback VARCHAR(255),
 start_date DATE,
 end_date DATE,
 pet_category VARCHAR(255),
 advertisement_email VARCHAR(255),
 owner_email VARCHAR(255),
 pet_name VARCHAR(255),
 FOREIGN KEY (
   start_date,
   end_date,
   pet_category,
   advertisement_email
 ) REFERENCES advertisements(start_date, end_date, pet_category, email)
ON DELETE CASCADE,
 FOREIGN KEY(owner_email, pet_name) REFERENCES owns_pets(email, pet_name)
ON DELETE CASCADE ON UPDATE CASCADE,
 PRIMARY
KEY(advertisement_email,pet_category,start_date,end_date,owner_email,pet_
name),
 CONSTRAINT "range for rating given must be valid" CHECK ((rating_given
>= 0AND rating_given <= 10))
);
```

Our application's constraints that are not enforced by our relational schema are enforced by triggers:

- Overlap constraint for users (user can be both pet owner and caretaker)
- No overlap constraint for caretakers (caretaker cannot be both part time and full time)
- Full time caretakers need to work 2 x 150 consecutive days.
- A pet owner cannot bid for an advertisement that has already been successfully bidded for.
- When a pet owner gives a rating to the caretaker taking care of his/her pet, the overall rating of the caretaker is updated for all users to see.
- A bid that is met or exceeds by the amount set by the pet owner will automatically be a successful bid.
- When a bid is successful, the pet days of full time caretakers is automatically updated to reflect him/her taking another pet under his/her care for the stated number of days

5. Discussion on whether your database is in 3NF or BCNF. Provide justifications for tables that are not in 3NF or BCNF.
    - For all the tables, every functional dependency in the closure of the set of functional dependencies is either trivial or a superkey. The primary keys

of each table are able to uniquely identify each tuple. Hence our database is in BCNF.

```
pet_owners
email -> password, name, credit_card

caretakers
email -> email (trivial)

pt_caretakers
email -> password, name, rating

ft_caretakers
email -> password, name, rating, pet_day

pcs_admins
email -> password, name

owns_pets
email, pet_name -> special_requirements, pet_category, pet_age

advertisements
email, pet_category, start_date, end_date -> daily_price

specifies_available_days
start_date, end_date, email -> start_date, end_date, email
(trivial)

salaries
payment_date, email->payment_amount

takes_leaves
start_date, end_date, email -> start_date, end_date, email

specifies
pet_category -> base_daily_price, pcs_email

bids_for
advertisement_email, pet_category, start_date, end_date,
owner_email, pet_name -> transfer_method, bid_price,
payment_method, rating_given, is_successful, feedback
```

6. Present the details of three non-trivial/interesting triggers used in your application by providing an English description of the constraints enforced by each trigger and showing the code of the trigger implementation.
    1. **Updating the overall rating of the caretaker when a pet owner gives a rating to the caretaker taking care of his/her pet.**
    Code:

```
CREATE OR REPLACE FUNCTION give_rating_and_update()
```

```
RETURNS TRIGGER AS $$
DECLARE rating INTEGER;
DECLARE is_fulltimer INTEGER;
BEGIN
   SELECT rating_given INTO rating FROM bids_for WHERE
advertisement_email = OLD.advertisement_email
   AND is_successful = true
   AND pet_category = OLD.pet_category
   AND start_date = OLD.start_date
   AND end_date = OLD.end_date
   AND OLD.rating_given = 0;
   SELECT COUNT(*) INTO is_fulltimer FROM ft_caretakers AS f WHERE
OLD.advertisement_email = f.email;
   IF is_fulltimer >= 1 THEN
   UPDATE ft_caretakers SET rating = (SELECT AVG(a.rating_given)
FROM (SELECT * FROM bids_for WHERE advertisement_email =
OLD.advertisement_email AND is_successful = true) AS a) WHERE
ft_caretakers.email = OLD.advertisement_email;
   ELSE
   UPDATE pt_caretakers SET rating = (SELECT AVG(a.rating_given)
FROM (SELECT * FROM bids_for WHERE advertisement_email =
OLD.advertisement_email AND is_successful = true) AS a) WHERE
pt_caretakers.email = OLD.advertisement_email;
   END IF;
   RETURN NEW;
END; $$
 LANGUAGE PLPGSQL;

CREATE TRIGGER update_caretaker_rating
AFTER UPDATE
ON bids_for
FOR EACH ROW WHEN(OLD.is_successful = TRUE AND OLD.rating_given = 0)
EXECUTE PROCEDURE give_rating_and_update();
```

2. **Trigger to ensure no other pet owner can bid for an advertisement that has been successfully bid for and automatic acceptance of a bid once the minimum bid price is met by a pet owner.**

Code:

```
CREATE OR REPLACE FUNCTION check_bidding_price()
RETURNS TRIGGER AS $$
DECLARE min_bid_price NUMERIC;
DECLARE num_successful INTEGER;
BEGIN
   SELECT COUNT(*) INTO num_successful FROM bids_for WHERE
   is_successful = true AND
   advertisement_email = NEW.advertisement_email AND
   pet_category = NEW.pet_category and
   start_date = NEW.start_date and
   end_date = NEW.end_date;
   SELECT daily_price INTO min_bid_price FROM advertisements WHERE
   pet_category = NEW.pet_category AND
   email = NEW.advertisement_email AND
   start_date = NEW.start_date AND
   end_date =  NEW.end_date;
   IF num_successful > 0 THEN
   RAISE EXCEPTION 'advertisment has been taken. No further bids
allowed';
   ELSIF NEW.bid_price >= min_bid_price THEN
   UPDATE bids_for SET is_successful = true WHERE
   advertisement_email = NEW.advertisement_email AND
   pet_category = NEW.pet_category and
   start_date = NEW.start_date and
   end_date = NEW.end_date and
   owner_email = NEW.owner_email;
   END IF;
   RETURN NEW;
END; $$
LANGUAGE PLPGSQL;

CREATE TRIGGER check_if_successful
AFTER INSERT OR UPDATE
ON bids_for
FOR EACH ROW
EXECUTE PROCEDURE check_bidding_price();
```

3. **Automatically checks if leave applied by full time caretaker is valid (the caretaker must work for a minimum of 2 * 150 consecutive days in one year):**

   Code:

```
CREATE OR REPLACE FUNCTION check_leave()
RETURNS TRIGGER AS $$
DECLARE total_count INTEGER;
BEGIN
SELECT COUNT(*) INTO total_count
FROM (
   SELECT extract(day from (( SELECT MIN(start_date)
           FROM takes_leaves
           WHERE email=NEW.email) - date_trunc('year',
CURRENT_DATE))) AS difference
   UNION ALL
   SELECT table1.start_date-table1.end_date AS difference
       FROM (SELECT leaves.end_date AS end_date,
MIN(leaves2.start_date) AS start_date
           FROM takes_leaves AS leaves, takes_leaves AS leaves2
           WHERE leaves2.start_date >= leaves.end_date
           AND leaves.email = leaves2.email
           AND leaves.email = NEW.email
           GROUP BY leaves.end_date) AS table1
   UNION ALL
   SELECT extract(day from ((date_trunc('year', CURRENT_DATE) +
interval '1 year' - interval '1 day') - (SELECT MAX(end_date) FROM
takes_leaves WHERE email=NEW.email))) AS difference) as bigtable
WHERE bigtable.difference >= 150;
IF total_count < 2 THEN
RAISE EXCEPTION 'invalid leaves';
DELETE FROM takes_leaves WHERE email = NEW.email AND start_date =
NEW.start_date AND end_date = NEW.end_date;
END IF;
RETURN NEW;
END; $$
LANGUAGE plpgsql;

CREATE TRIGGER is_valid_leave
AFTER INSERT
ON takes_leaves
FOR EACH ROW
EXECUTE PROCEDURE check_leave();
```

7. Show the SQL code of three of the most complex queries implemented in your application. Provide an English description of each of these queries.
    1. **PCS admin can view each employee, both part time and full time, their names and emails and their projected salary and revenue earned for the PCS company for the month**
       Code:

```
SELECT email, name, 'Full Time' AS type,
check_ft_salary('ft_caretaker', email) as salary FROM ft_caretakers
   UNION
SELECT email, name, 'Part Time' AS type,
check_pt_salary('pt_caretaker', email) as salary FROM
pt_caretakers;


CREATE OR REPLACE FUNCTION check_ft_salary(
   type VARCHAR,
   input_email VARCHAR)
   RETURNS float8 AS $$
   DECLARE totalDays float8;
   DECLARE totalRevenue float8;
       BEGIN SELECT (SUM(noOfDays)) INTO totalDays
       FROM (SELECT (end_date-start_date) AS noOfDays,
bid_price*(end_date-start_date) AS transactionRevenue FROM bids_for
WHERE is_successful = true
       AND advertisement_email=input_email
       AND (extract(month from start_date) = extract(month from
CURRENT_DATE)
           AND extract(month from end_date) = extract(month from
CURRENT_DATE))
```

```sql
       UNION ALL
       SELECT (extract(day from end_date)) AS noOfDays,
bid_price*(extract(day from end_date)) AS transactionRevenue FROM
bids_for WHERE is_successful = true
       AND advertisement_email=input_email
       AND (extract(month from start_date) < extract(month from
CURRENT_DATE)
           AND extract(month from end_date) = extract(month from
CURRENT_DATE))
       UNION ALL
       SELECT (DATE_PART('days',
               DATE_TRUNC('month', CURRENT_DATE)
               + '1 MONTH'::INTERVAL
               - '1 DAY'::INTERVAL
           )-extract(day from start_date)) AS noOfDays,
bid_price*(DATE_PART('days',
               DATE_TRUNC('month', CURRENT_DATE)
               + '1 MONTH'::INTERVAL
               - '1 DAY'::INTERVAL
           )-extract(day from start_date)) AS transactionRevenue
FROM bids_for WHERE is_successful = true
       AND advertisement_email=input_email
       AND (extract(month from end_date) > extract(month from
CURRENT_DATE)
           AND extract(month from start_date) = extract(month from
CURRENT_DATE))) AS table1;

       SELECT (SUM(transactionRevenue)) INTO totalRevenue FROM
(SELECT (end_date-start_date) AS noOfDays,
bid_price*(end_date-start_date) AS transactionRevenue FROM bids_for
WHERE is_successful = true
       AND advertisement_email=input_email
       AND (extract(month from start_date) = extract(month from
CURRENT_DATE)
           AND extract(month from end_date) = extract(month from
CURRENT_DATE))
       UNION ALL
       SELECT (extract(day from end_date)) AS noOfDays,
bid_price*(extract(day from end_date)) AS transactionRevenue FROM
bids_for WHERE is_successful = true
       AND advertisement_email=input_email
       AND (extract(month from start_date) < extract(month from
CURRENT_DATE)
           AND extract(month from end_date) = extract(month from
CURRENT_DATE))
       UNION ALL
       SELECT (DATE_PART('days',
               DATE_TRUNC('month', CURRENT_DATE)
               + '1 MONTH'::INTERVAL
               - '1 DAY'::INTERVAL
           )-extract(day from start_date)) AS noOfDays,
bid_price*(DATE_PART('days',
               DATE_TRUNC('month', CURRENT_DATE)
               + '1 MONTH'::INTERVAL
               - '1 DAY'::INTERVAL
           )-extract(day from start_date)) AS transactionRevenue
FROM bids_for WHERE is_successful = true
       AND advertisement_email=input_email
       AND (extract(month from end_date) > extract(month from
CURRENT_DATE)
           AND extract(month from start_date) = extract(month from
CURRENT_DATE))) AS table1;

       IF (type = 'ft_caretaker' OR type = 'ft_user') THEN IF
totalDays <= 60 THEN RETURN 3000; END IF;
       RETURN 3000 + 0.8*(totalDays-60)*(totalRevenue/totalDays);
END IF;

       IF totalDays <= 60 THEN RETURN totalRevenue-3000; END IF;
       RETURN totalRevenue - (3000 +
0.8*(totalDays-60)*(totalRevenue/totalDays));
       END;  $$
       LANGUAGE PLPGSQL;
```

2. **PCS can view the number of successful bids (transactions) for each month when the caretaking services start. The PCS can then hire more part time caretakers on months when demand for caretaking services is high.**
   Code:

```
SELECT COUNT(*)
FROM bids_for b
WHERE is_successful = true
GROUP BY extract(month from b.start_date)
```

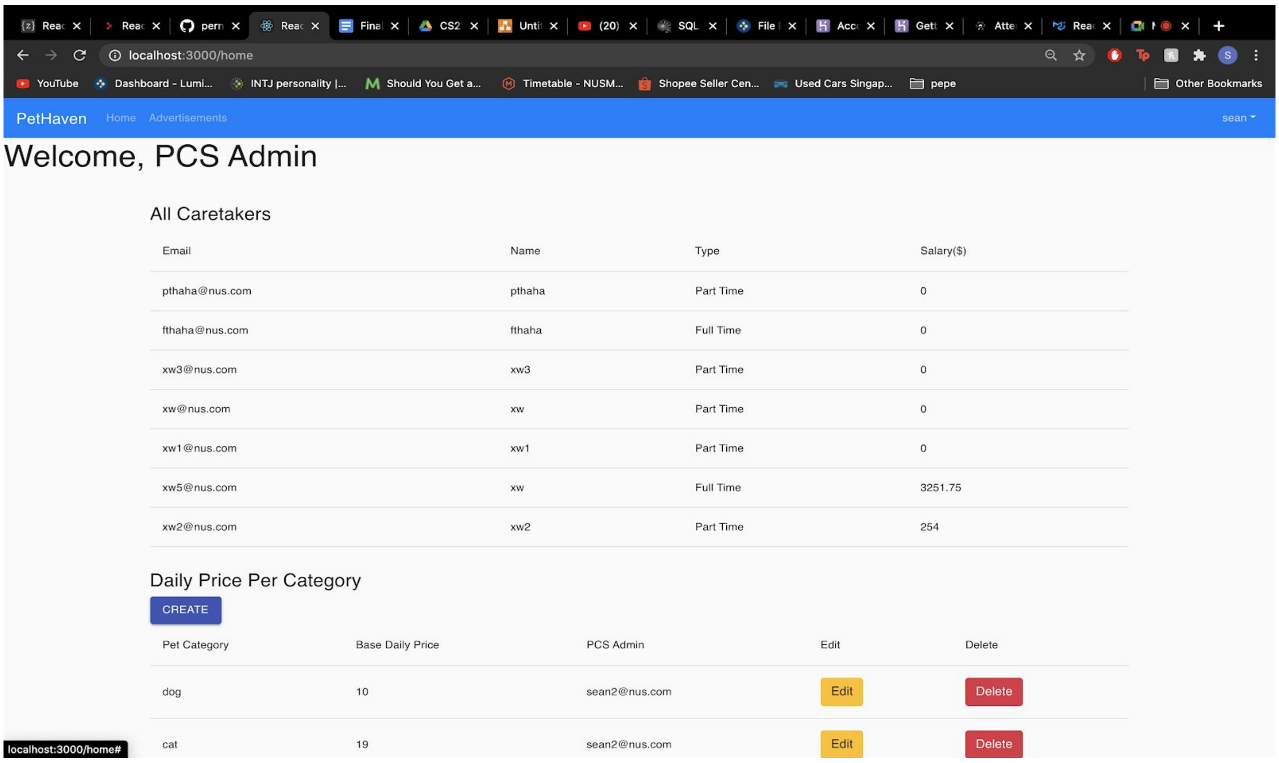
3. **PCS can view the number of successful bids (transactions) for each pet category. The PCS can then hire more employees that can care for specific pet categories where demand for caretaking services for these pet categories is high.**
   Code:

```
SELECT COUNT(*)
FROM bids_for b
WHERE is_successful = true
GROUP BY b.pet_category
```

8. Specification of the software tools/frameworks used in your project.
   - PostgreSQL (Database)

- Express.js
- React.js (Frontend)
- Node.js (Backend)
- Bcrypt.js (salt+hash and verification of password)
- JsonWebToken (JWT user authentication)
- Material-UI (React UI templates and components)
- Git (Source Code Management)
- Heroku (Deployment)
- Bootstrap 4 (Frontend)

9. Two or three representative screenshots of your application in action.



Welcome, PCS Admin

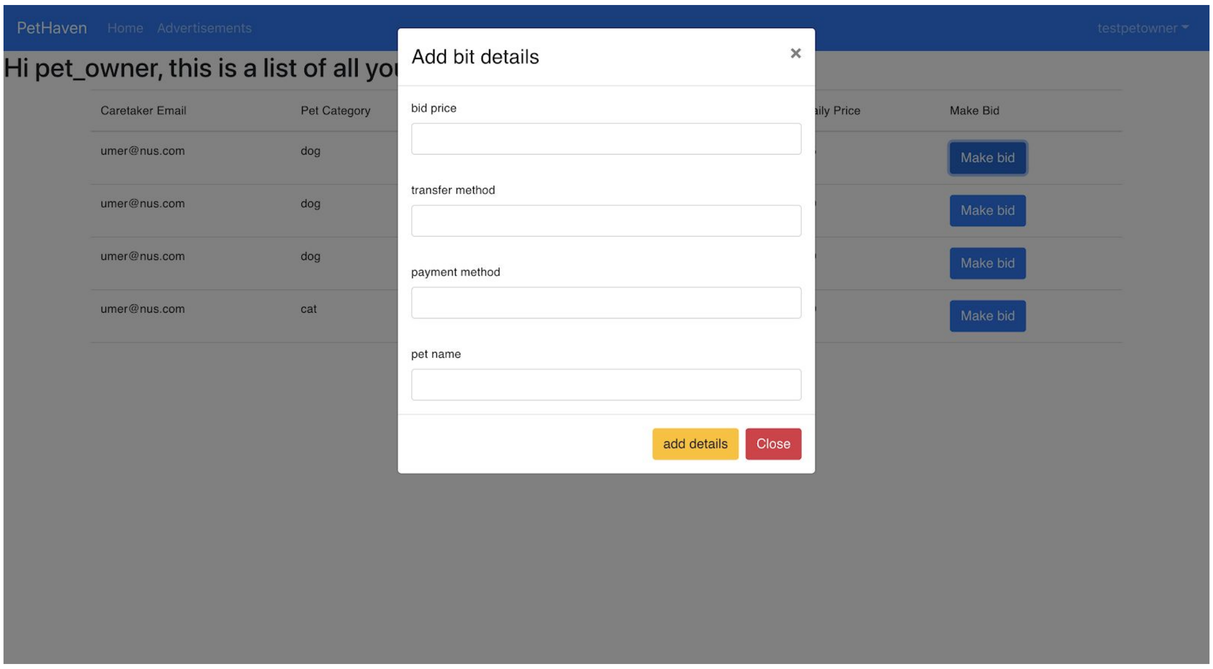
All Caretakers

Email	Name	Type	Salary($)
pthaha@nus.com	pthaha	Part Time	0
fthaha@nus.com	fthaha	Full Time	0
xw3@nus.com	xw3	Part Time	0
xw@nus.com	xw	Part Time	0
xw1@nus.com	xw1	Part Time	0
xw5@nus.com	xw	Full Time	3251.75
xw2@nus.com	xw2	Part Time	254

Daily Price Per Category

CREATE

Pet Category	Base Daily Price	PCS Admin	Edit	Delete
dog	10	sean2@nus.com	Edit	Delete
cat	19	sean2@nus.com	Edit	Delete



Hi pet_owner, this is a list of all yo

Caretaker Email	Pet Category	ily Price	Make Bid
umer@nus.com	dog		Make bid
umer@nus.com	dog		Make bid
umer@nus.com	dog		Make bid
umer@nus.com	cat		Make bid

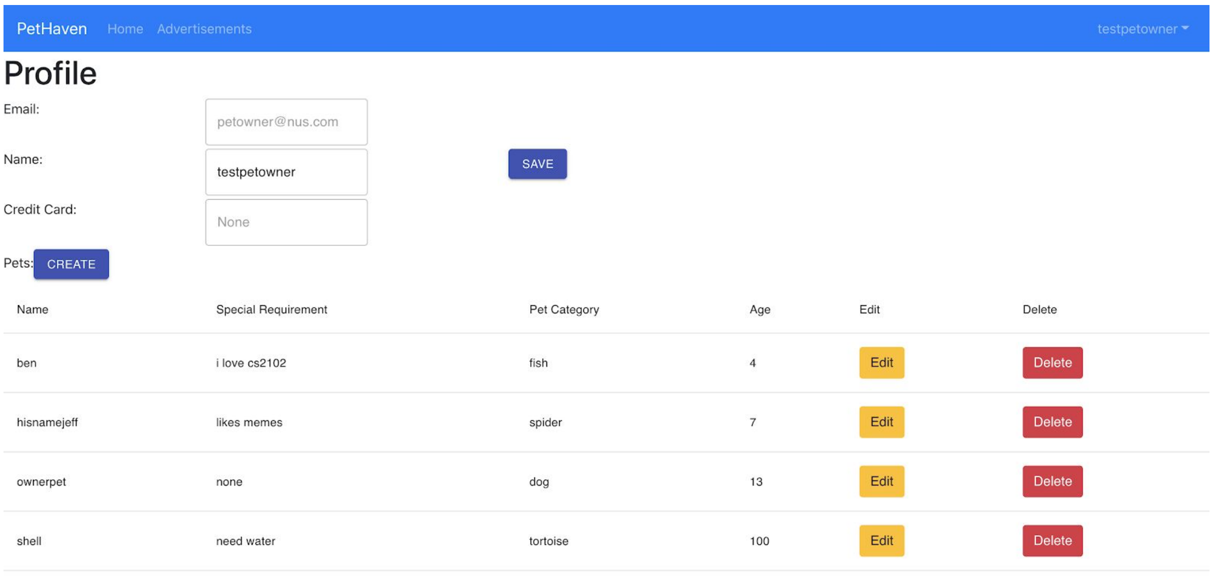
Add bit details

bid price

transfer method

payment method

pet name



Profile

Email:	petowner@nus.com
Name:	testpetowner
Credit Card:	None

Pets:

Name	Special Requirement	Pet Category	Age	Edit	Delete
ben	i love cs2102	fish	4	Edit	Delete
hisnamejeff	likes memes	spider	7	Edit	Delete
ownerpet	none	dog	13	Edit	Delete
shell	need water	tortoise	100	Edit	Delete

10. A summary of any difficulties encountered and lessons learned from the project.
    1. We faced difficulties in assessing the problem with the required complexity. For instance, the ER diagram we initially created was far too complex than required. However, Prof. Adi’s comments and feedback really helped us simplify our ER diagram and remove unnecessary entities and relationship sets.
    2. We also faced major difficulties with the web development component of the project as the code provided by Prof Adi was difficult to comprehend. Thus, we decided to create the web application from scratch using React for frontend and Node.js for backend. However, due to our lack of exposure to these technologies, we spent a lot of time on routing and changing components in the front end. Hence, we think a supplementary lesson on setting up our project would have helped us a lot and we could have dedicated more time working on the database side of the project, which is the core of this web application.