1a. Name	1b. Student number	2. Responsibilities
Lim Jun-Wei Ryan Arthur Douglas	A0206028N	Creation of website and video demonstration, some SQL code for tables, some SQL queries for search functionality on the website, wrote the portion of the report regarding website.
Tian Jing Jie	A0190063M	ER Diagram, Most SQL code for tables, some initial website related linkage and clarifications, SQL code for triggers, wrote some parts of the report.
Li Xinze	A0172969J	Data generation, triggers, some code from tables, report writing difficulty part
Yong Cui Ying, Clarissa	A0157661Y	Report writing & formatting, ER diagram, some SQL code for tables, triggers

3. Description of application's data and functionalities

The application allows `User` accounts to be created. These `User` accounts will be categorised into `PCS administrators` and `non-admins`. Among the non-admin users, there will be a further classification whereby the `User` can be a `Pet Owner`, `Care Taker` or both. `Non-admins` are able to input their personal information such as their phone number, address, and preferred transfer method. This information will be used upon a successful bid to facilitate communication between the `Pet Owner` and `Care Taker`.

Based on their `User` type, i.e. `PCS Administrator`, `Pet Owner` or `Care Taker`, they will be able to access different the functionalities and information that they require. For instance, a `Pet Owner` will be able to search for and select `Care Takers` based on criteria such as dates, the `Care Taker`'s rating and price of the `Care Taker`'s services. They will also be able to input their `Pet`'s information such as the `Pet`'s name, type, and any special requirements the `Pet` might need from the `Care Taker`.

On the other hand, `Care Takers` are able to view information regarding their upcoming, current, and past jobs, as well as their `salary` and number of pet-days worked. They are also able to filter the results based on certain criteria, such as the month and year that they require the information for. `Care Takers` are also able to apply for leave through the application, which will be expanded on in the next section. In addition, `Care Takers` can set their own price list, which includes the types of `Pets` they are able to take care of as well as the price for each `Pet` type.

`PCS administrators` are able to set the minimum price for each `Pet` type.

The application matches available `Care Takers` with `Pet Owners` who require their services, and this is done through a bidding system.

Interesting/non-trivial aspects of application's functionalities/implementation

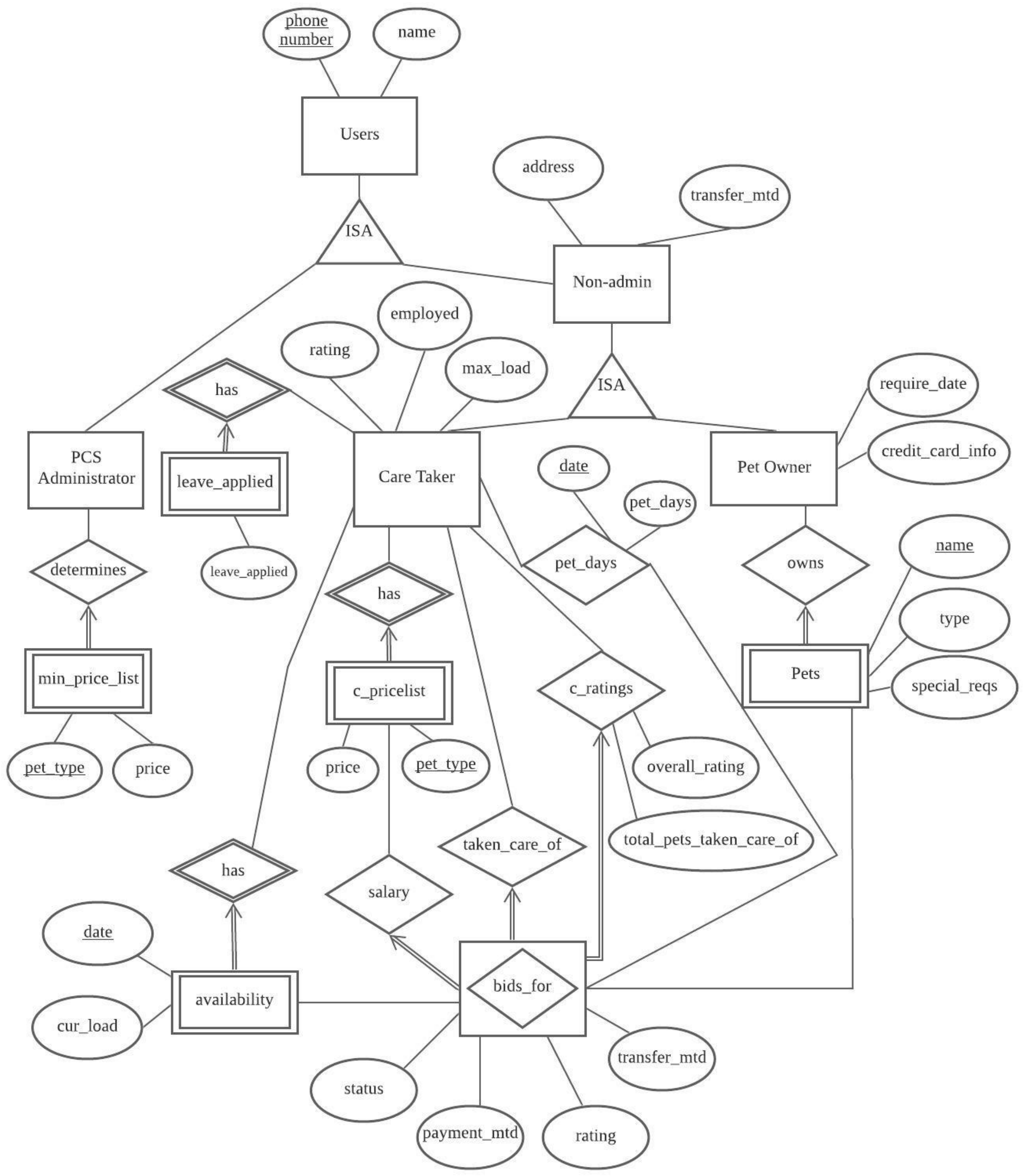
The application supports a bidding system where `Pet Owners` can bid for a `Care Taker`'s services, provided that the `Care Taker` is available on the required date and is able to care for the particular type of `Pet`. Upon winning a bid, i.e. a successful match between `Care Taker` and `Pet Owner`, the status of the transaction will be shown as 'pending'. It will change to 'in progress' and 'completed' accordingly. After the transaction has been completed, `Pet Owners` will then have to leave a rating for the `Care Taker`. An overall rating for `Care Takers` will be calculated using all their ratings they receive. This rating, along with the number of pet-days worked, will be used to calculate a performance indicator that is used to identify underperforming `Care Takers`.

The application also allows `Care Taker`s to apply for leave. They can apply for a leave as long as after the leave is applied, there should still exist 2 blocks of 150 consecutive working days, otherwise, the leave application would not be approved and is not reflected in the application.

Data constraints

1. The application does not allow `Care Taker`s choose which bids they want to accept. It is automatically assigned by the system on a first come first served basis, as long as the following conditions are met (these are also constraints):
    a. Bid price ≥ the minimum price set by the `Care Taker`
    b. `Care Taker` must be able to take care of the `Pet`'s pet type.
    c. `Care Taker` has not met maximum workload.
2. Every bid will lead to a `Care Taker` being matched with a `Pet Owner`.
3. Non-admin users can be a `Pet Owner`, a `Care Taker` or both. A user that is a `Pet Owner` or a `Care Taker` cannot be a PCS `administrator`.
4. For each `Pet` type, prices set by the `Care Taker` ≥ minimum price set by PCS `Administrator.`
5. The price for the services of a `Care Taker` increases with their `c_rating` and the demand for their services.
6. `Care Taker`s cannot apply for leave if upon approval of the leave, they would not have worked for a minimum of 2 x 150 consecutive days in a year.
7. `Care Taker`s cannot apply for leave on the days that they have at least 1 `Pet` under their care.
8. The `salary` of full-time `Care Taker` depends on how many `Pet`s they have taken care of in a month.
9. Part-time `Care Taker`s receive 75% of their price as `salary`, and PCS receives the other 25%.
10. `Pet Owner`s have to rate the `Care Taker` after every transaction.
11. `Pet Owner`s cannot have more than 1 `Pet` with the same name.

4. ER Model



Justifications for non-trivial design decisions in the ER Model

1. Weak entity sets

Type of dependency	Between		Justification
Existential	PCS admin	`min_price_list`	The existence of `min_price_list` is dependent on the PCS `Administrator`, since without the PCS `Administrator` creating it, it would not exist.

	Pet Owner	Pets	A Pet cannot exist without a Pet Owner. However, we are able to uniquely identify a Pet based on its attributes.
Identity	Care Taker	availability	Attributes of availability cannot uniquely identify which Care Taker is available on a given day. availability would only make sense if we knew which Care Taker's availability we were referring to.
	Care Taker	c_pricelist	Every Care Taker has their own price list. The attributes of c_pricelist itself are unable to uniquely identify which Care Taker's pricelist it is.
	Care Taker	leave_applied	We cannot uniquely identify whose leave application it was based solely on the attributes of leave_applied. We need to know the Care Taker who applied for it in order to uniquely identify the entity.

2. Key + total participation

Between		Justification
bids_for	salary	Each entry in bids_for is a bid for a Care Taker's services. In our application, a Pet Owner will definitely be allocated a Care Taker whenever they submit a bid. Therefore, the Care Taker will definitely receive a salary for their services. Since each bid is for one session with the Care Taker, the Care Taker will only receive one salary per bid.
	c_ratings	Our application mandates that the Pet Owner give the Care Taker who they bid for a rating after the Care Taker has taken care of their Pet. Therefore, the Care Taker will receive exactly 1 rating per bid, which will be stored in c_ratings.
	taken_care_of	As stated previously, each bid will lead to a Care Taker to be allocated to a Pet Owner to take care of their Pet. Also, each bid only allows the Care Taker to take care of a particular Pet once. Therefore, for each bid, the Care Taker would have taken_care_of the Pet exactly once.

3. ISA

We decided to split the users into PCS Administrators and Non-admins since Care Taker and Pet Owner share certain attributes. Therefore, to avoid redundancy, we grouped them under Non-admin before splitting them into Care Taker and Pet Owner. We needed to split Non-admins into Care Taker and Pet Owner as they differ in attributes and relationships. Non-admins and admins have a covering constraint while Care Taker and Pet Owner have an overlap constraint.

4. Higher-order entity

bids_for is a higher-order entity because Care Takers would not have taken_care_of Pets, clock any pet_days, get a rating or a salary if there were no bids_for their services.

Application constraints not captured by the ER Model

1. Each user has an ISA constraint where the user can be a pet owner, a caretaker or both. A user that is a pet owner or a caretaker cannot be a PCS administrator.
2. For each pet_type, the Care Taker's price in their c_pricelist must be greater than or equal to the price determined by the PCS administrator in min_price_list.
3. Care Takers are only available for bidding to Pet Owners whose Pet's pet_type is in their c_pricelist.
4. Care Takers are only available for bidding if their load (cur_load) on the day is less than their max load.
5. Pet Owners can only leave a rating for the Care Taker when bids_for.status = 2 (complete).
6. Each entry in bids_for will lead to a Care Taker being allocated to the Pet Owner who bid for their services. That is to say, as long as a Pet Owner bids for the Care Taker's services, the Care Taker will take care of the Pet on the day that was bid for.
7. Care Takers cannot apply for leave if upon approval of the leave, they would not have worked for a minimum of 2 x 150 consecutive days in the year.
8. Care Takers cannot apply for leave on days that they have Pets under their care, that is, bids_for.status = 1 (in progress).
9. The price for the services of a Care Taker increases with their c_rating and the demand for their services.
10. The salary of full-time Care Taker depends on how many Pets they have taken care of in a month.
11. Part-time Care Takers receive 75% of their price as salary, and PCS receives the other 25%.
12. Pet Owners cannot have more than 1 Pet with the same name.

5. Relational Schema

```
-- We have included the CREATE TYPE statements here for clarity and ease of
understanding

CREATE TYPE transfer AS ENUM('0','1','2');    -- 0 for pet owner deliver, 1 for
caretaker pick up, 2 for physical transfer
```

```
CREATE TYPE employed_as AS ENUM ('0', '1');    -- 0 for pt and 1 for ft

CREATE TYPE status as ENUM('0', '1', '2');     -- 0 (pending) , 1 (in prog) ,
2(completed)

CREATE TABLE users(
      name         VARCHAR (100),
      phone        INTEGER,
      PRIMARY KEY(phone)
);

CREATE TABLE pcsadmin (
      name         VARCHAR(100),
      phone        INTEGER,
      FOREIGN KEY(name, phone) REFERENCES users(name, phone) ON DELETE cascade ON
        UPDATE cascade,
      PRIMARY KEY(phone)
);

CREATE TABLE nonadmin(
      name         VARCHAR (100),
      phone        INTEGER,
      address      VARCHAR (100) NOT NULL,
      transfer     transfer NOT NULL,
      FOREIGN KEY(name, phone) REFERENCES users(name, phone) ON DELETE cascade ON
        UPDATE cascade,
      PRIMARY KEY(phone)
);

CREATE TABLE petowner (
      name                       VARCHAR (100),
      phone                      INTEGER,
      address                    VARCHAR (100) NOT NULL,
      transfer                   transfer NOT NULL,
      c_card_info                VARCHAR(30) NOT NULL,
      require_date               DATE DEFAULT NULL,
      FOREIGN KEY(name, phone) REFERENCES nonadmin(name, phone) ON DELETE cascade
        ON UPDATE cascade,
      PRIMARY KEY(phone)
);

CREATE TABLE caretakers (
      name         VARCHAR (100),
      phone        INTEGER,
      address      VARCHAR (100) NOT NULL,
      transfer     transfer NOT NULL,
      FOREIGN KEY(name, phone) REFERENCES nonadmin(name, phone),
      PRIMARY KEY(phone)
);

CREATE TABLE c_ratings (
      cname                            VARCHAR,
      cphone                          INTEGER,
      employed                        employed_as NOT NULL,
      total_pets_taken_care_of        INTEGER DEFAULT 0,
      total_sum_ratings               INTEGER DEFAULT 0,
      overall_rating                  NUMERIC(2, 1) DEFAULT 0
                                      CHECK (overall_rating <= 5),
      max_load                        INTEGER DEFAULT 0
                                      CHECK ((max_load <= 5 AND employed='1') OR
                                      (max_load<=2    AND    employed='0')    OR
                                      (max_load<=5    AND    employed='0'    AND
                                      overall_rating>=4)),
      FOREIGN KEY (cname, cphone) REFERENCES caretakers (name, phone),
      PRIMARY KEY (cphone)
);

CREATE TABLE has_availability (
      caretaker    VARCHAR(100),
      phone        INTEGER,
      date         DATE,
      cur_load     INTEGER DEFAULT 0,
      FOREIGN KEY (caretaker, phone) REFERENCES caretakers(name, phone) ON DELETE
        cascade ON UPDATE cascade,
      PRIMARY KEY (phone, date)
);

CREATE TABLE c_pricelist (
      cname        VARCHAR,
      cphone              INTEGER,
      pet_type     VARCHAR,
      price        INTEGER,
      FOREIGN KEY (cname, cphone) REFERENCES caretakers(name, phone) ON DELETE
        cascade ON UPDATE cascade,
      PRIMARY KEY (cphone, pet_type)
);

CREATE TABLE determines_mpl (
      pcsadmin     VARCHAR,
      phone        INTEGER,
      pet_type     VARCHAR,
      price        INTEGER NOT NULL,
      FOREIGN KEY (pcsadmin, phone) REFERENCES pcsadmin(name, phone) ON DELETE
        cascade ON UPDATE cascade,
      PRIMARY KEY (phone, pet_type)
);
CREATE TABLE owns (
      owner               VARCHAR(100),
      phone               INTEGER NOT NULL,
      pet_name            VARCHAR(100),
      pet_type            VARCHAR(100),
      special_reqs VARCHAR(500),
      FOREIGN KEY (owner, phone)  REFERENCES petowner(name, phone) ON DELETE
```

```
        CASCADE ON UPDATE cascade,
      PRIMARY KEY (phone, pet_name)
);

CREATE TABLE bids_for (
      caretaker    VARCHAR(100) ,
      cphone              INTEGER NOT NULL,
      petowner     VARCHAR(100),
      pphone          INTEGER NOT NULL,
      date         DATE,
      pet_name     VARCHAR(100),
      pet_type     VARCHAR(100) NOT NULL,
      special_reqs VARCHAR(100),
      status              status NOT NULL,
      trans_mtd    VARCHAR NOT NULL,
      pay_mtd      VARCHAR NOT NULL,
      rating              INTEGER NOT NULL,
      year         INTEGER DEFAULT 0,
      month        INTEGER DEFAULT 0,
      cost         INTEGER DEFAULT 0,
      FOREIGN KEY (petowner, pet_name, pet_type) REFERENCES owns(owner, pet_name,
        pet_type) ON DELETE CASCADE ON UPDATE cascade,
      FOREIGN KEY (caretaker, cphone, date) REFERENCES has_availability(caretaker,
        phone, date) ON DELETE CASCADE ON UPDATE cascade,
      PRIMARY KEY (cphone, pphone, date, pet_name)
);

CREATE TABLE pet_days(
      caretaker    VARCHAR,
      cphone       INTEGER NOT NULL,
      date         DATE NOT NULL,
      year         INTEGER DEFAULT 0,
      month        INTEGER DEFAULT 0,
      pet_days     INTEGER DEFAULT 0,
      FOREIGN KEY (caretaker, cphone) REFERENCES caretakers (name, phone) ON DELETE
        cascade ON UPDATE cascade,
      PRIMARY KEY (cphone, date)
);

CREATE TABLE leave_applied (
      caretaker    VARCHAR,
      phone        INTEGER,
      leave_applied       DATE,
      successful   BOOLEAN DEFAULT FALSE,
      FOREIGN KEY (caretaker, phone) REFERENCES caretakers(name, phone) ON DELETE
        cascade ON UPDATE cascade,
      PRIMARY KEY (phone, leave_applied)
);

CREATE TABLE salary(
      caretaker    VARCHAR NOT NULL,
      cphone       INTEGER NOT NULL,
      employed     employed_as,
      petowner     VARCHAR NOT NULL,
      pphone       INTEGER NOT NULL,
      pet_name     VARCHAR NOT NULL,
      pet_type     VARCHAR NOT NULL,
      price        INTEGER NOT NULL,
      date         DATE NOT NULL,
      year         INTEGER DEFAULT 0,
      month        INTEGER DEFAULT 0,
      total_cost   INTEGER DEFAULT 0,
      final_salary INTEGER DEFAULT 0,
      FOREIGN    KEY    (caretaker,    petowner,    date,    pet_name)    REFERENCES
        bids_for(caretaker, petowner, date, pet_name),
      FOREIGN    KEY    (pet_type,    price,    caretaker,    cphone)    REFERENCES
        c_pricelist(pet_type, price, cname, cphone),
      PRIMARY KEY (cphone, pet_name, date)
);

CREATE TABLE taken_care_of (
      caretaker VARCHAR,
      cphone INTEGER,
      petowner VARCHAR,
      pphone INTEGER,
      pet_name VARCHAR,
      date DATE,
      FOREIGN    KEY    (caretaker,    petowner,    date,    pet_name)    REFERENCES
        bids_for(caretaker, petowner, date, pet_name),
      FOREIGN KEY (caretaker, cphone) REFERENCES caretakers(name, phone),
      PRIMARY KEY (cphone, pphone, pet_name, date)
);

UPDATE bids_for SET month = EXTRACT (MONTH FROM "date");
UPDATE bids_for SET year = EXTRACT (YEAR FROM "date");
UPDATE  bids_for  SET  cost  =  (SELECT  DISTINCT  price  FROM  c_pricelist  WHERE
cname=caretaker);

UPDATE c_ratings SET total_sum_ratings = (SELECT SUM(rating) FROM bids_for WHERE
caretaker=cname);
UPDATE c_ratings SET total_pets_taken_care_of = (SELECT COUNT(*) FROM bids_for WHERE
caretaker=cname);
UPDATE c_ratings SET overall_rating = total_sum_ratings/total_pets_taken_care_of;
UPDATE c_ratings SET max_load=5 WHERE employed='1';

UPDATE pet_days SET month = EXTRACT (MONTH FROM "date");
UPDATE pet_days SET year = EXTRACT (YEAR FROM "date");
UPDATE  pet_days  SET  pet_days  =  (SELECT  COUNT(*)  FROM  bids_for  WHERE
caretaker=pet_days.caretaker AND year=pet_days.year AND month=pet_days.month);

UPDATE salary SET month = EXTRACT (MONTH FROM "date");
UPDATE salary SET year = EXTRACT (YEAR FROM "date");
UPDATE salary SET total_cost = CASE WHEN (employed='1') THEN price*0.8 ELSE
price*0.75 END;
UPDATE salary SET final_salary = CASE WHEN employed='0' THEN total_cost ELSE (CASE
WHEN (SELECT COUNT(*) FROM salary)<=60 AND employed='1'THEN 3000 ELSE 3000 + (SELECT
SUM(total_cost) FROM salary) END) END;

UPDATE has_availability SET cur_load= (SELECT COUNT(*) FROM bids_for WHERE
caretaker=has_availability.caretaker AND date=has_availability.date);

UPDATE leave_applied SET successful = CASE WHEN (SELECT cur_load FROM
has_availability WHERE date=leave_applied)=0 THEN TRUE ELSE FALSE END;
```

Application's constraints that are not enforced by relational schema

Constraints enforced by triggers:

1. An existing PCS Administrator cannot be inserted as a Non-admin user.
2. For each `pet_type`, the `Care Taker's` `price` in their `c_pricelist` must be greater than or equal to the `price` determined by the PCS administrator in `min_price_list`.
3. `Care Takers` are only available for bidding to `Pet Owners` whose Pet's `pet_type` is in their `c_pricelist`.
4. `Care Takers` are only available for bidding if their load (`cur_load`) on the day is less than their `max` load.
5. `Pet Owners` can only leave a `rating` for the `Care Taker` when `bids_for.status = 2` (complete).

Constraints not enforced:

1. Minimum price for a `Care Taker` increases with demand and high rating.

6. Discussion on whether your database is in 3NF or BCNF

Table name	3NF/BCNF	Justification
`users`	BCNF	`F = {phone -> name}` By Lemma 2, `users` is in BCNF. phone is the only candidate key. Since there exists some a -> A in F+ where A is `name`, and since `name` is not a prime attribute, `users` is not in 3NF.
`pcsadmin`	BCNF	`F = {phone -> name}` By Lemma 2, `pcsadmin` is in BCNF. phone is the only candidate key. Since there exists some a -> A in F+ where A is `name`, and since `name` is not a prime attribute, `pcsadmin` is not in 3NF.
`nonadmin`	BCNF	`F = {phone -> name, address, transfer}` All FD a -> A in F+ is either trivial or LHS, which will be phone, is superkey. Therefore, nonadmin is in BCNF.

		Candidate keys = {phone; name, address} Since there exists some a -> A in F+ where A is transfer, eg. phone -> transfer, and transfer is not a prime attribute, nonadmin is not in 3NF.
petowner	BCNF	F = {phone -> name, address, transfer, c_card_info, require_date} All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, petowner is in BCNF. Prime attributes = {name, phone, address, c_card_info} Since there exists some a -> A in F+ where A is not a prime attribute, petowner is not in 3NF. For example, phone -> transfer.

caretakers	BCNF	F = {phone -> name, address, transfer}
		All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, caretakers is in BCNF.
		Prime attributes = {phone, name, address}
		Since there exists some a -> A in F+ where A is transfer, and transfer is not a prime attribute, nonadmin is not in 3NF. For example, {name, address -> transfer}.

c_ratings	BCNF	F = {cphone -> cname, employed, total_pets_taken_care_of, total_sum_ratings, overall_rating, max_load} All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, c_ratings is in BCNF. Prime attributes = {cphone} Since there exists some a -> A in F+ where A is not a prime attribute, c_ratings is not in 3NF. For example, cphone -> cname.

has_availability	BCNF	F = {phone -> caretaker; phone, date -> cur_load}
		All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, has_availability is in BCNF.
		Prime attributes = {phone, date}
		Since there exists some a -> A in F+ where A is not a prime attribute, has_availability is not in 3NF. For example, phone, date -> caretaker.

c_pricelist	BCNF	F = {cphone -> cname; cphone, pet_type -> price}
		All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, c_pricelist is in BCNF.
		Prime attributes = {cphone, pet_type}

		Since there exists some a -> A in F+ where A is not a prime attribute, c_pricelist is not in 3NF. For example, cphone, pet_type -> cname.

determines_mpl	BCNF	F = {phone -> pcsadmin; phone, pet_type -> price}
		All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, determines_mpl is in BCNF.
		Prime attributes = {phone, pet_type}
		Since there exists some a -> A in F+ where A is not a prime attribute, determines_mpl is not in 3NF. For example, phone -> pcsadmin.

owns	BCNF	F = {phone -> owner; phone, pet_name -> pet_type, special_reqs}
		All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, owns is in BCNF.
		Prime attributes = {phone, pet_name}
		Since there exists some a -> A in F+ where A is not a prime attribute, owns is not in 3NF. For example, phone, pet_name -> pet_type.

bids_for	BCNF	F = {c_phone -> caretaker; p_phone -> petowner; date -> year, month; p_phone, pet_name -> pet_type, special_reqs; c_phone, p_phone, date, pet_name -> status, trans_mtd, pay_mtd, rating, cost} All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, bids_for is in BCNF. The only candidate key = {c_phone, p_phone, date, pet_name}. Therefore, bids_for is not in 3NF since there exists some a -> A in F+ where A is not a prime attribute. For example c_phone, p_phone, date, pet_name -> status.

pet_days	BCNF	F = {cphone -> caretaker; date -> year, month; cphone, date -> pet_days}
		All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, pet_days is in BCNF.
		Candidate keys = {cphone, date; cphone, year, month}
		Since there exists some a -> A in F+ where A is not a prime attribute, pet_days is not in 3NF. For example, cphone -> caretaker.

leave_applied	BCNF	F = {phone -> caretaker; phone, leave_applied -> successful} All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, leave_applied is in BCNF. The only candidate key is {phone, leave_applied}. Therefore, there will exist some a -> A in F+ where A is not a prime attribute, such as

		phone, leave_applied -> successful. Hence, leave_applied is not in 3NF.
salary	BCNF	F = {cphone -> caretaker, employed; pphone -> petowner, pet_name, pet_type; date -> year, month; cphone, pet_name, date -> price, total_cost; cphone, date -> final_salary} All FD a -> A in F+ is either trivial or LHS is superkey. Therefore, salary is in BCNF. The only candidate key is {cphone, pet_name, date}. Therefore, there will exist some a -> A in F+ where A is not a prime attribute, such as cphone, pet_name, date -> price. Hence, salary is not in 3NF.
taken_care_of	BCNF	The only candidate key is {cphone, pphone, pet_name, date}, and therefore taken_care_of is in BCNF since it has single candidate key. However, this also means that there exists some a -> A in F+ where A is not a prime attribute. For example, cphone, pphone, pet_name, date -> caretaker. This means that taken_care_of is not in 3NF.

7. Triggers

1. This trigger is to ensure that Pet Owners are only able to bid for Care Takers who are available on the required date. We implemented this by checking the availability of each Care Taker on the date that the Pet Owner wants to bid for.

```
DECLARE ctx NUMERIC;
CREATE OR REPLACE FUNCTION has_availability()
RETURNS TRIGGER AS
$$ BEGIN
    PERFORM * FROM has_availability HA
    WHERE NEW.caretaker = HA.caretaker AND NEW.date = HA.date;
    IF NOT FOUND THEN
      RETURN NULL;
    ELSE
      RETURN NEW;
    END IF;
    END; $$
LANGUAGE plpgsql;

CREATE TRIGGER is_available
BEFORE INSERT OR UPDATE ON bids_for
FOR EACH ROW EXECUTE PROCEDURE has_availability();
```

2. This trigger ensures that Pet Owners are only able to bid for Care Takers that are able to care for the type of Pet the Pet Owner owns. We implemented this by

checking if the Pet Owner's Pet type is found in the c_pricelist of each Care Taker as c_pricelist will only contain the price of Pet types that the Care Taker is able to take care of. Care Takers who are unable to take care of the type of Pet the Pet Owner owns will be excluded from the result. The resultant list of Care Takers will consist of those who are able to take care of the type of Pet the Pet Owner owns.

```
CREATE OR REPLACE FUNCTION check_possibility()
RETURNS TRIGGER AS
$$ BEGIN
    PERFORM * FROM c_pricelist CPL
    WHERE NEW.caretaker = CPL.cname AND NEW.pet_type=CPL.pet_type;
    IF NOT FOUND THEN
      RETURN NULL;
    ELSE
      RETURN NEW;
    END IF;
    END; $$
LANGUAGE plpgsql;

CREATE TRIGGER is_possible
BEFORE INSERT OR UPDATE ON bids_for
FOR EACH ROW EXECUTE PROCEDURE check_possibility();
```

3. This trigger ensures that the price set by the Care Taker cannot be lower than the price defined in min_price_list. We implemented this by checking that the price for the care of a particular Pet type defined by each Care Taker is greater than or equal to the price stated in the min_price_list for the corresponding Pet type.

```
CREATE OR REPLACE FUNCTION min_price()
RETURNS TRIGGER AS
$$ BEGIN
    PERFORM * FROM determines_mpl MPL
    WHERE NEW.pet_type = MPL.pet_type AND NEW.price >= MPL.price;
    IF NOT FOUND THEN
        RETURN NULL;
    ELSE
        RETURN NEW;
    END IF;
    END; $$
LANGUAGE plpgsql;

CREATE TRIGGER check_price
BEFORE INSERT OR UPDATE ON c_pricelist
FOR EACH ROW EXECUTE PROCEDURE min_price();
```

8. SQL Queries

1. This query returns the month with the highest number of jobs across all Care Takers.

```
SELECT month
FROM bids_for
GROUP BY month
```

```
HAVING COUNT(month)=(SELECT MAX(count) FROM (SELECT COUNT(date) AS count
FROM bids_for GROUP BY month, year) AS temp);
```

2. This query returns the details of underperforming `Care Takers`. Underperforming `Care Takers` are `Care Takers` that have worked less than 25 pet days and have a rating of less than 2.

```
SELECT *
FROM (pet_days p JOIN c_ratings ON p.caretaker=c.cname)
WHERE p.pet_days <25 AND c.overall_rating<2 AND c.employed='1';
```

3. This query returns the estimated `salary` of `Care Takers` for each month where they had at least 1 job. The difference in `salary` paid for full-time `Care Takers` and part-time `Care Takers` is also accounted for.

```
SELECT DISTINCT caretaker, date, month, year,
CASE
     WHEN employed='0' THEN total_cost
     WHEN (SELECT COUNT(*) FROM salary)<=60 AND employed='1'THEN 3000
     WHEN (SELECT COUNT(*) FROM salary)>60 AND employed='1' THEN 3000 +
(SELECT SUM(total_cost) FROM salary)
END total_paid
FROM salary;
```

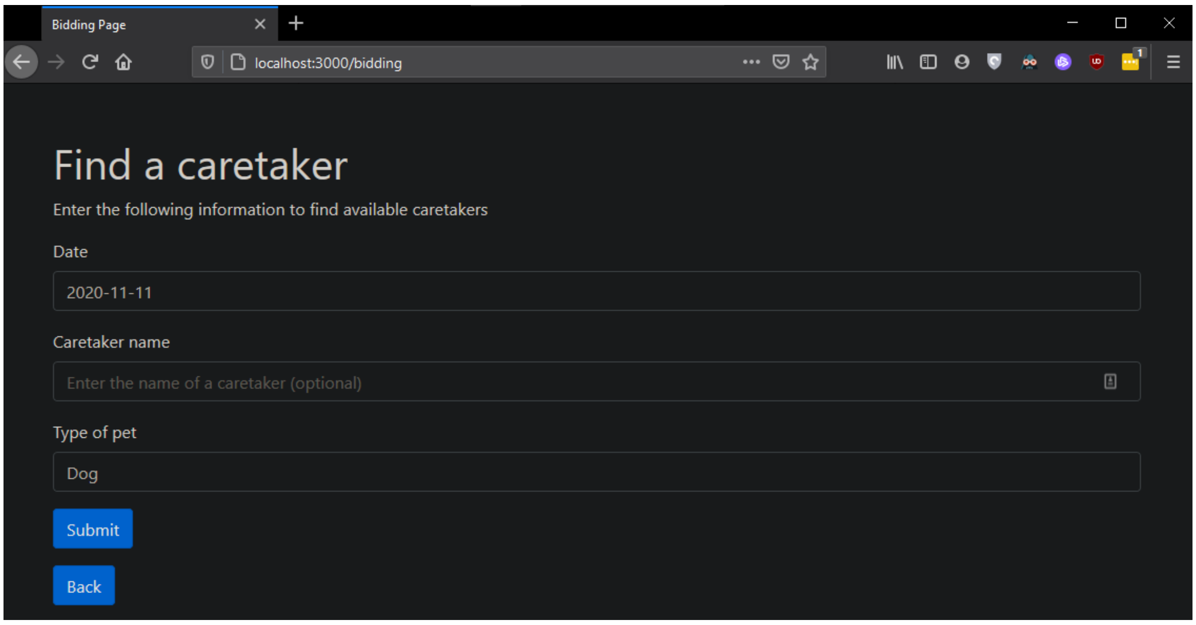
9. Specifications of software tools/frameworks used

– PostgreSQL
– SQL Fiddle
– NodeJS 14.15.0
– HTML

10. Screenshots

These screenshots show a `Pet Owner`'s experience when finding a `Care Taker` to care for their `Pet`.

In this example, the `Pet Owner` is looking for someone who is available to take care of his dog on the 11th of November 2020. The `Pet Owner` will fill in the form shown below and click 'Submit'. If he clicks 'Back', he will return to his profile page.



Find a caretaker

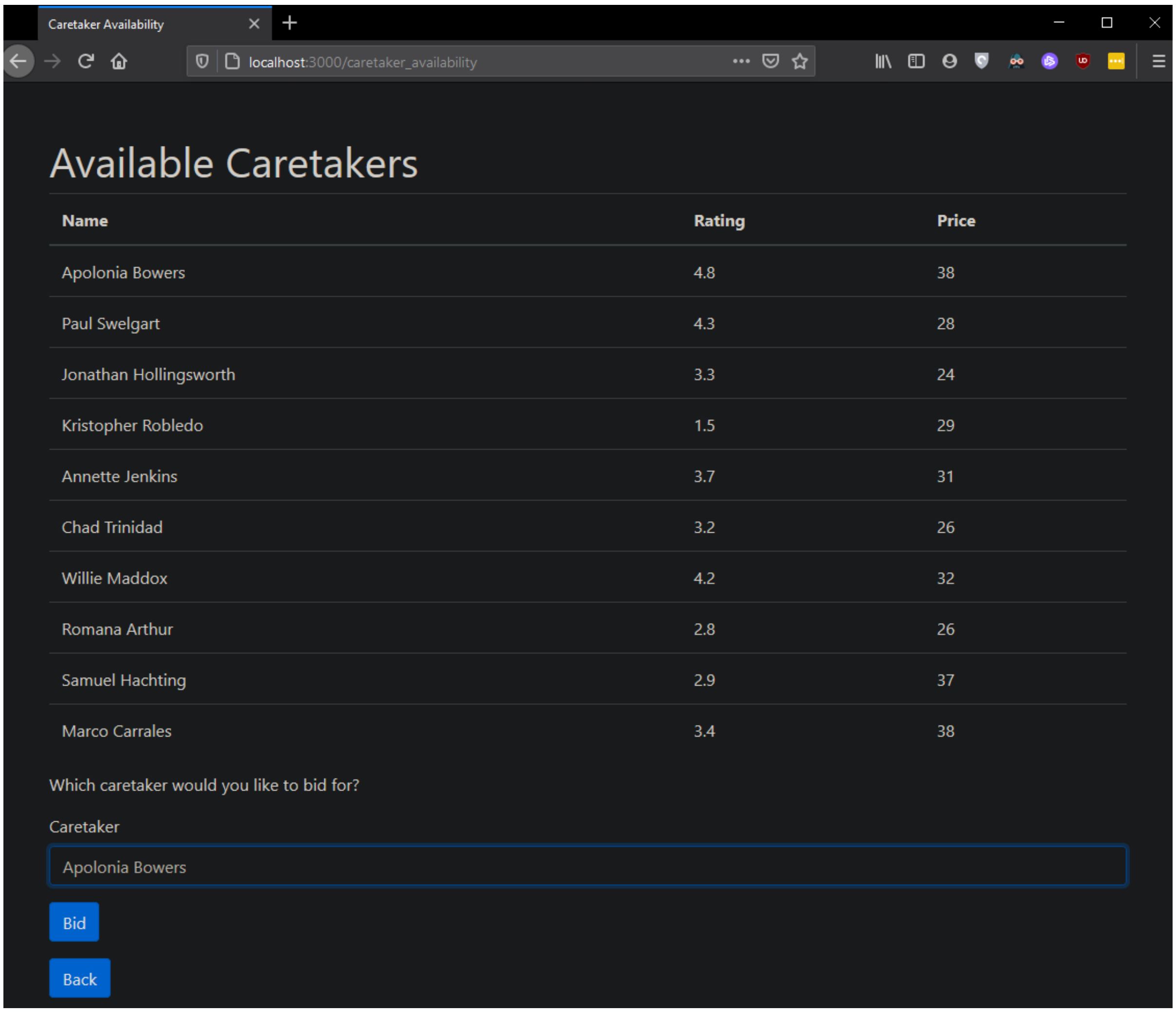
Enter the following information to find available caretakers

Date: 2020-11-11

Caretaker name: Enter the name of a caretaker (optional)

Type of pet: Dog

Upon submitting the form, the `Pet Owner` will be able to see the names, average ratings, and prices of the available `Care Takers`. After the `Pet Owner` has decided on which `Care Taker` he would like to bid for, he keys the name of the `Care Taker` in the field below the table. In this case, the `Pet Owner` would like to bid for 'Apolonia Bowers'. He then clicks on 'Bid' to submit the bid.


Caretaker Availability

localhost:3000/caretaker_availability

Available Caretakers

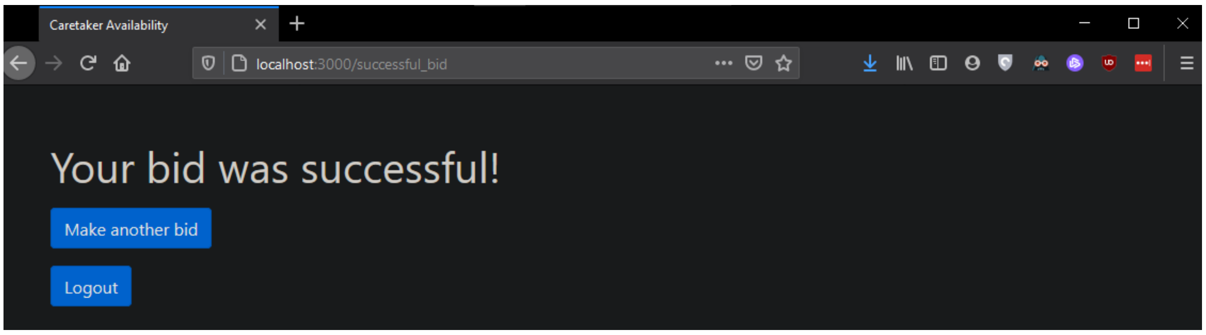
Name	Rating	Price
Apolonia Bowers	4.8	38
Paul Swelgart	4.3	28
Jonathan Hollingsworth	3.3	24
Kristopher Robledo	1.5	29
Annette Jenkins	3.7	31
Chad Trinidad	3.2	26
Willie Maddox	4.2	32
Romana Arthur	2.8	26
Samuel Hachting	2.9	37
Marco Carrales	3.4	38

Which caretaker would you like to bid for?

Caretaker

Apolonia Bowers

Since one of our constraints is that a `Care Taker` who is available will automatically accept any new job, the bid is immediately successful. From here, the `Pet Owner` is able to make another bid for a `Care Taker` or he is able to log out of the system


Caretaker Availability

localhost:3000/successful_bid

Your bid was successful!

11. Summary of difficulties encountered and lessons learnt

One of our group members, Tong Lee, had to withdraw from the module, leaving us short-handed. Thankfully, Prof Adi gave us an extension so that we were able to finish the project.

During the design of the triggers code, the syntax provided in lecture is not accepted by PostgreSQL. Therefore, we needed to try out different syntaxes with the help of online resources to correctly implement the triggers code. SQL queries have slight variations among the different platforms. We hence needed to understand these differences in order to implement our application.

We found it difficult to cater to the various real-world scenarios when designing the database. We had to decide our own logical boundaries and had to reason with one another in order to reach a consensus. For example, we were debating whether there would be a scenario in which a Pet Owner can have more than one Pet with the same name, and if it would be acceptable if the pets were of different types. After intense discussion, we decided that the chances of that happening in the real world is slim to none as it would not make any sense.

Implementing the website was also a challenge as the language needed to setup the website and linking the database, NodeJS, was not taught in lecture. It was a feat having to understand the language and apply it in the building of our website especially due to the lack of prior knowledge and experience as well as time constraints. Getting the website up and running was a challenge in and of itself. Initially, the navigation buttons on the website were not working as intended and were not directing the user to the correct page. This was one of the bigger challenges that we faced but we managed to solve them through many hours of consulting online resources and forums. This portion took up a significant amount of time. From this, we have learnt how to setup a website and connect a database to it as well as how to use JavaScript to manipulate the database.

Throughout the implementation process, we learnt that it was important to not only look at the application a developers' perspective, but also from the users' perspective. It is crucial to know what users want and need in terms of application functionality and design in order to optimise usability. This is an essential application development or system development skill.