Database Systems

CS2102

Project Report

Chen Caijie	A0166916U
Loh Wei Kiat	A0188708M
Michelle Yong Kai Wen	A0189286J
Phoebe Chan	A0189183R
Tay Sheryl	A0188225Y

Team: 40

1. Project Responsibilities

Team Member	Project Responsibilities
Chen Caijie	Front-End and Back-End Development for Admin Profile and CareTaker Profiles, UI Cleanup, ER Diagram, Project Report
Loh Wei Kiat	Front-End and Back-End Development for Leave Applications and Bid Accept/Delete/Update , ER Diagram, Project Report
Michelle Yong Kai Wen	Front-End and Back-End Development for PetOwner Profiles and Pets, ER Diagram, Project Report
Phoebe Chan	Front-End and Back-End Development for User Authentications and Filtering, Build Schema and Mock Data, ER Diagram, Project Report
Tay Sheryl	Front-End and Back-End Development for Filtering Caretakers and Bid Creation, ER Diagram, Project Report

2. Application's Data Requirements and Functionalities

2.1 Application Functionality

PetLovers is a web application for a pet caring service (PCS). There are four main types of users in PetLovers, namely *Pet Owners*, *FullTime CareTakers*, *PartTime CareTakers* and *Admins*. Users can sign up to be either a *Pet Owner*, *CareTaker* or both, but are not allowed to sign up to be an *Admin*. For this application, we assume that *Admins* will get their accounts by signing up through the backend database. Those who sign up as caretakers will also be prompted to select between a full-time position and a part-time position, enter the pet types they can take care of as well as specify the prices they wish to charge for each pet type.

An *Admin* can view summary information about the users in the database. This information includes the total number of jobs in the current month, the total salary to be paid to all the *FullTime CareTakers* and *PartTime CareTakers*, and all the under-performing *CareTakers*. An under-performing *CareTaker* is defined as having fewer pet days than the current day of the month / 2. For example, if it is the 20th day of the month, an under-performing *CareTaker* would be one that has completed less than 10 pet days in the current month. In addition, we are able to see the list of all *FullTime CareTakers* and *PartTime CareTakers*, and they are sorted according to the number of *Pets* that they have cared for in the current month in descending order, alongside with the salary to be paid to them for the month. An *Admin* will also be able to set the base daily price for *FullTime CareTakers* for their respective pet types, and this base daily price will scale later on during bidding depending on the rating of the *FullTime CareTaker*.

A *Pet Owner* will be able to update information such as their credit card details for payments, and also information about their *Pets* such as the pet name, pet type and any special requirements for that pet. A *Pet Owner* can include multiple *Pets* into their profile. A *Pet Owner* can also browse and bid for a *CareTaker* to care for their *Pet(s)* for designated periods, then review their experience after the *CareTaker* has finished their job.

A *CareTaker* can view and manage all the bids that people have made for them. They can also view basic information of their profile, their current availabilities and advertised prices, reviews for them and current and past jobs. *Full-time CareTakers* are automatically assumed to have an availability period of 2 years on signing up, unless they apply for *Leave*. *Part-time CareTakers* on the other hand, are free to fill in their own availability periods and the type of *Pets* that they want to care for.

The general flow of a bid placed is as follows:

1. *Pet Owner* enters the start date, end date, pet type to be cared for and maximum price they are willing to pay.
2. A list of *CareTakers* that are able to care for the selected pet type during the duration, with a daily price (for *FullTime CareTaker*) or advertised price (for *PartTime CareTaker*) lesser than the bidded price, will then appear.
3. *Pet Owner* can also view reviews and ratings of each *CareTaker* for past successful bids.
4. *Pet Owner* selects a *CareTaker* of his or her choice, indicating the start date, end date, pet type to take care of, the transfer method and the payment method.
5. This bid will then be viewable to the *CareTaker* that the *Pet Owner* had bidded for.
6. Should the *CareTaker* accept the bid, it is presumed that this *CareTaker* will care for the *Pet Owner*, and they will individually make arrangements to carry out this transaction (not supported by the application).
7. When the transaction is finished, which is defined as the current date being after the end date of a successful bid, *Pet Owner* will then be able to leave a rating and review for this particular *CareTaker*, which then can be seen on the *CareTaker* profile.

2.2 Data Requirements & Constraints

2.2.1 User Type

A user account can only belong to one of the four types: *Pet Owners, FullTime CareTakers, PartTime CareTakers and Admins.* A user is identified by their username which is unique, and users are notified if a username they want is already taken. A user's username and password also cannot be null.

2.2.2 Bids

A bid has petowner_username, pet_name, caretaker_username, start_date, end_date, price, transfer_method, payment_method, review, rating and isSuccessful attributes. A bid is identified by the petowner_username, pet_name, caretaker_username, start_date and end_date.

A *Pet Owner* is able to bid for as many *CareTakers* as they like, as long as they have a pet that fulfills the pet_type that they are bidding for. The *Pet Owner* will have to indicate the maximum price that they are willing to pay for. The filtered results include:

1. All *PartTime CareTakers* who have set their availabilities during this period with an asking price less than the indicated price, and have less than 2 *Pets* under their care during this period if they have a rating of less than 4, or have less than 5 *Pets* if they have a rating of 4 or more.
2. All *FullTime CareTakers* who have not indicated *Leave* during this period with a (base daily price multiplied by average ratings) price less than the indicated price, and have less than 5 *Pets* under their care during this period.

CareTakers must accept their bids to be considered as successful. At the time of accepting the bid, a *PartTime CareTaker* must have less than 2 *Pets* under their care during this period if they have a rating of less than 4, or have less than 5 *Pets* if they have a rating of 4 or more. On the other hand, a *FullTime CareTaker* must have less than 5 *Pets*. Once accepted, these bids will be reflected in the *CareTaker* profile and *Pet Owner* profile respectively.

As *Pet Owners* may have bidded for multiple *CareTakers* during the same period for the same *Pet*, the first *CareTaker* that accepts their bid will be the successful one. If any of the other bids are later on accepted, it will be deemed as a failure as *Pet Owners* are not allowed to let multiple *CareTakers* care for the same *Pet* during the same period of time.

When viewing the current bids, *Pet Owners* will be able to see their bids for *CareTakers* on one page, and *CareTakers* will be able to see bids for them by *Pet Owners* on another. The bids that are shown will only be bids that are on or after the current's date. Any bids that have not been accepted prior will be assumed as not successful.

2.2.3 Leaves

A *Leave* application has ftct_username, start_date and end_date attributes. *FullTime CareTakers* are able to apply for as many *Leaves* as they want. They are also able to update and delete their *Leaves* should they choose to not take them anymore. However, they will not be able to add any *Leaves* that are before the current date, and they will also not be able to update or delete any of the *Leaves* that have already started, as we presume that they have already or are currently taking the *Leave*. Consequently, they will also not be able to update their *Leaves* to a date before the current date.

2.2.4 Availabilities

An availability consists of username, pet_type, advertised_price, start_date, end_date. Each row in the availabilities table is identified by all its attributes. Only *PartTime CareTakers* indicate their availability, since *FullTime CareTakers* derive their availability from their *Leave* application. The start_date and end_date determine the availability's duration, and the constraint applies that every availability's duration should not overlap with any other availability.

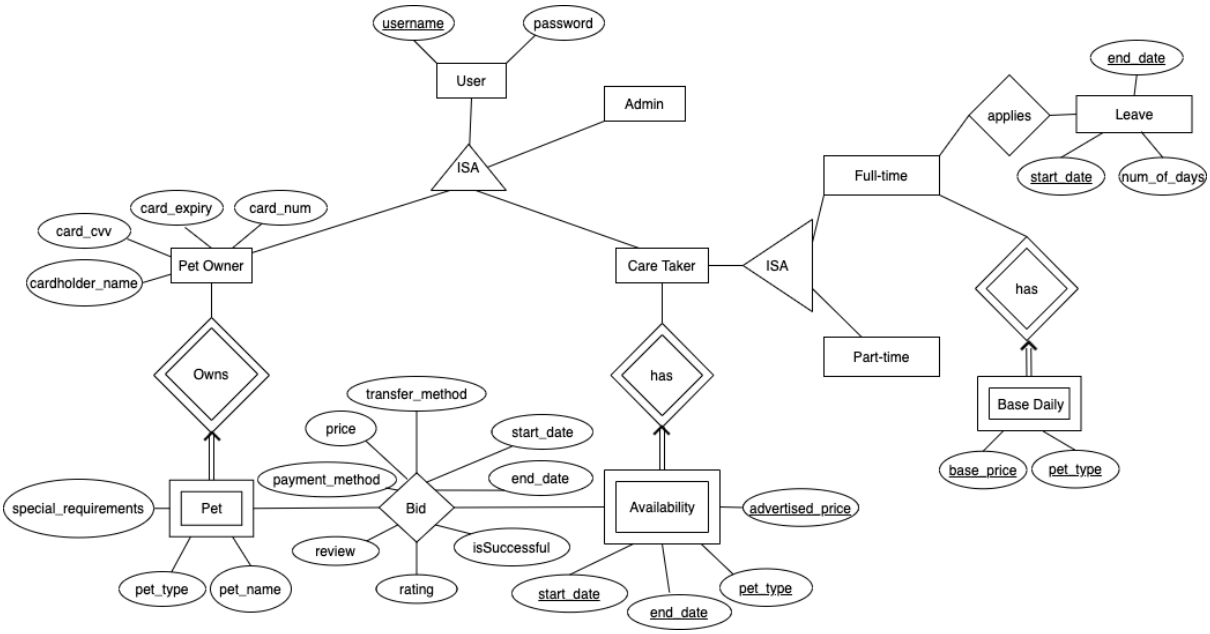
2.2.5 Caretakers Profile

All summary information about a *CareTaker* (salary, pet days, job type for current month), is done in one single query. The single query enforced the following constraints:

1. *FullTime CareTaker* will receive salary of $3000 per month up to 60 pet-day
2. *FullTime CareTaker* will receive 80% of price as bonus for excess pet-day > 60
3. *PartTime CareTaker* receives 75% of their stated price
4. Pet days are calculated by counting the number of successful bids for that *CareTaker* that are completed (both start_date and end_date IS NOT NULL and end_date falls in the current month) multiplied by the number of days the *Pet* was taken care of (end_date - start_date).

3. ER Model & Constraints

3.1 ER Diagram



3.1.1 Constraints shown in the ER diagram

1. Each User is a *Pet Owner, CareTaker* or a *PCS Administrator*. This constraint is covering and overlapping as a *Pet Owners* can also be a *CareTaker*.
2. Each User can be identified by their username and has a password attribute.
3. Each *Pet Owner* has a card_number, card_expiry, card_cvv, and cardholder_name attributes.
4. Each *Pet* has pet_name, special_requirements, and pet_type attributes.
5. *Pet* is a weak entity set with identity dependency relationship with *Pet Owner* since the pet owner owns the *Pet*.
6. Every *Pet* must be owned by one *Pet Owner.*
9. Availability is a weak entity set with identity dependency relationship with *CareTaker*.
10. Each *CareTaker's* Availability slot can be identified by the start date, end date, pet type and advertised price.

11. Each *CareTaker* is a *FullTime* or a *PartTime CareTaker*. This constraint is covering and not overlapping as a *CareTaker* can either be a full-time or a part-time employee.
12. Base Daily is a weak entity set with identity dependency relationship with *FullTime CareTaker.*
13. Each Base Daily can be identified by the base price and pet type.
14. Each *Leave* has start_date, end_date, and num_of_days attributes.
15. Each *Pet Owner* can bid for an availability slot of a *CareTaker* for his/her *Pet* of his/her choice. Every bid will have a start_date, end date, transfer (how to transfer a *Pet*), as well as an isSuccessful attribute.
16. *Full Time CareTaker* can apply for *Leave*.
17. *Pet Owner* may submit multiple review/rating for a *CareTaker* if the *CareTaker* has taken care of the *Pet Owner's Pet* multiple times.

<u>3.1.2 Constraints not enforced by the ER diagram</u>

18. Base Price multiplied by average Rating for the *FullTime CareTaker* must be lower than the Advertised Price in the Availability.
19. *FullTime CareTaker* has a limit of 5 pets at any time.
20. *PartTime CareTaker* cannot take care of more than 2 *Pets* unless they have good rating and overall has a limit of 5 *Pets* at any time.
21. The Rating and Review attributes of the Bid relation can only be set if the Bid is successful.
22. *CareTaker* can only take care of a *Pet* that they can care for.
23. Successful bidder is chosen by *CareTaker*.
24. *FullTime CareTaker* must work for a minimum of 2 x 150 consecutive days a year.
25. *FullTime CareTaker* cannot apply for *Leave* if there is at least one *Pet* under their care.
26. *FullTime CareTaker* will always accept a bid immediately if possible.
27. *PartTime CareTaker* can specify their availability for the current and next year

4. Relational Schema

```
CREATE TABLE admins (
   username VARCHAR(50) PRIMARY KEY,
   password VARCHAR(256) NOT NULL
);
```

```
CREATE TABLE petowners (
   username VARCHAR(50) PRIMARY KEY,
   password VARCHAR(256) NOT NULL,
   card_num NUMERIC(16),
   card_expiry NUMERIC(4),
   card_cvv NUMERIC(3),
   cardholder_name VARCHAR(256)
);
```

```
CREATE TABLE pets (
   petowner_username VARCHAR(50) REFERENCES
petowners (username) ON DELETE cascade,
   pet_name VARCHAR(50) NOT NULL,
   pet_type VARCHAR(20) NOT NULL,
   special_requirements VARCHAR(256),
   PRIMARY KEY (petowner_username, pet_name)
);
```

```
CREATE TABLE caretakers (
   username VARCHAR(50) PRIMARY KEY,
   password VARCHAR(256) NOT NULL
);
```

```
CREATE TABLE fulltime_caretakers (
   username VARCHAR(50) PRIMARY KEY REFERENCES
caretakers (username) ON DELETE cascade
);
```

```
CREATE TABLE parttime_caretakers (
   username VARCHAR(50) PRIMARY KEY REFERENCES
caretakers (username) ON DELETE cascade
);
```

```
CREATE TABLE bids (
   petowner_username VARCHAR(50),
   pet_name VARCHAR(50) NOT NULL,
   caretaker_username VARCHAR(50),
   start_date DATE,
   end_date DATE,
   price NUMERIC NOT NULL,
   transfer_method VARCHAR(100) NOT NULL,
   payment_method VARCHAR(20) NOT NULL,
   review VARCHAR(200),
   rating INTEGER CHECK ((rating IS NULL) OR (rating >= 0 AND rating <= 5)),
   isSuccessful BOOLEAN DEFAULT NULL,
   FOREIGN KEY (petowner_username, pet_name) REFERENCES pets (petowner_username, pet_name),
   PRIMARY KEY (petowner_username, pet_name, caretaker_username, start_date, end_date),
   CHECK (petowner_username <> caretaker_username)
);
```

```
CREATE TABLE availabilities (
   username VARCHAR(50) REFERENCES caretakers (username) ON DELETE cascade,
   pet_type VARCHAR(20) NOT NULL,
   advertised_price NUMERIC NOT NULL,
   start_date DATE NOT NULL,
   end_date DATE NOT NULL,
   PRIMARY KEY (username, start_date, end_date, advertised_price, pet_type)
);
```

```
CREATE TABLE leaves_applied (
   ftct_username VARCHAR(50) REFERENCES
fulltime_caretakers (username) ON DELETE cascade,
   start_date DATE NOT NULL,
   end_date DATE NOT NULL,
   num_of_days NUMERIC NOT NULL,
   CHECK (num_of_days >= 1),
   PRIMARY KEY(ftct_username, start_date, end_date)
);
```

```
CREATE TABLE base_dailys (
   ftct_username VARCHAR(50) REFERENCES
fulltime_caretakers (username) ON DELETE cascade,
   base_price NUMERIC,
   pet_type VARCHAR(20) NOT NULL,
   PRIMARY KEY(ftct_username, base_price, pet_type)
);
```

3.2.1 Constraints not enforced by the schema

1. Covering constraint of *FullTime* and *PartTime CareTakers*. (Enforced by frontend)
2. *Leaves* are not allowed to be overlapped (Enforced by trigger)
3. Availabilities are not allowed to be overlapped (Enforced by trigger)
4. *CareTakers* may not accept a bid for a pet if another *CareTaker* has already accepted that bid for that particular pet during that period. (Enforced by trigger)
5. Price for *FullTime CareTakers* increases with their rating. (Enforced by query)
6. *FullTime CareTakers* salary is $3000 up to 60 pet-day. They receive 80% of price as bonus for excess pet-day (Enforced by query)
7. *PartTime CareTakers* receive 75% of their price as salary. (Enforced by query)
8. At any single point in time, a *PartTime CareTaker* cannot take care of more than 2 *Pet* unless they have a good rating (e.g., 4 out of 5, or any such threshold you define) and they cannot have more than 5 *Pet* regardless of rating. (Enforced by trigger)
9. *FullTime CareTakers* can take care of maximum 5 *Pets* at any time while *PartTime CareTakers* can take care of maximum 5 *Pets* if their ratings are good. Else, they can take care of maximum 2 *Pets* at any time. (Enforced by trigger)
10. A *PetOwner* may only submit their ratings and reviews after a Bid has been completed. (Enforced by query and frontend)
11. A Bid cannot be accepted, declined or cancelled if its start date is after the current date. (Enforced by query.

5. BCNF or 3NF

Schema	BCNF	3NF
admins	In BCNF	In 3NF
petowners	In BCNF	In 3NF
pets	In BCNF	In 3NF
caretakers	In BCNF	In 3NF
fulltime_caretakers	In BCNF	In 3NF
parttime_caretakers	In BCNF	In 3NF
bids	In BCNF	In 3NF
availabilities	In BCNF	In 3NF
leaves_applied	In BCNF	In 3NF
base_dailys	In BCNF	In 3NF

6. Non-trivial/Interesting Triggers

6.1 Leaves Application - Check for 2 x 150 days

This trigger is used to enforce that *FullTime CareTakers* are not allowed to take any *Leaves* if it stops them from fulfilling 2 x 150 days of work from the current date within the coming year. This is done by fulfilling one of the two conditions below:

1. There are 2 blocks of 2 x 150 days between the current date and the current date + 1 year.
2. There is 1 block of 1 x 300 days between the current date and the current date + 1 year.

```
CREATE FUNCTION func_check_satisfy_2x150days() returns TRIGGER AS
  $$
BEGIN
        IF ((new.start_date = CURRENT_DATE - 1 AND new.end_date = CURRENT_DATE - 1)
                OR (new.start_date = CURRENT_DATE + 366 AND new.end_date = CURRENT_DATE + 366)
        ) THEN
                RETURN new;
        ELSE
                IF (NOT EXISTS ( SELECT 1
                                        FROM    leaves_applied
                                        WHERE   (CURRENT_DATE - 1) <= end_date
                                                AND (CURRENT_DATE - 1) >= start_date
                                )
                ) THEN
                        INSERT INTO leaves_applied VALUES (new.ftct_username, CURRENT_DATE - 1,
                                CURRENT_DATE - 1, 1);

                ENDIF;

                IF (NOT EXISTS ( SELECT 1
                                        FROM    leaves_applied
                                        WHERE   (CURRENT_DATE + 366) <= end_date
                                                AND (CURRENT_DATE + 366) >= start_date
                                )
                ) THEN
                        INSERT INTO leaves_applied VALUES (new.ftct_username,
                        CURRENT_DATE + 366, CURRENT_DATE + 366, 1 );
                ENDIF;

                IF (NOT EXISTS ( SELECT   L1.ftct_username
                                 FROM     leaves_applied L1, leaves_applied L2
                                 WHERE    L1.ftct_username = L2.ftct_username
                                       AND L1.ftct_username = new.ftct_username
                                        AND L1.end_date < L2.start_date
                                        AND NOT EXISTS ( SELECT 1
                                                         FROM   leaves_applied
                                                         WHERE  start_date < L2.start_date
                                                              AND    start_date > L1.end_date )
                                         AND L1.end_date + 150 < L2.start_date
                                 GROUP BY L1.ftct_username
                                 HAVING   Count(L1.ftct_username) >= 2 )
                        AND
                    NOT EXISTS ( SELECT L6.ftct_username
                                 FROM   leaves_applied L6, leaves_applied L7
                                WHERE L6.ftct_username = L7.ftct_username
                                       AND L6.ftct_username = new.ftct_username
                                       AND L6.end_date < L7.start_date
                                       AND NOT EXISTS ( SELECT 1
                                                        FROM   leaves_applied L8
                                                        WHERE  L8.start_date < L7.start_date
                                                              AND L8.start_date > L6.end_date )
```

```
                                                        AND L6.end_date + 300 <
                                                                L7.start_date )
                ) THEN
                        DELETE
                                FROM   leaves_applied
                                WHERE  ftct_username = new.ftct_username
                                       AND    start_date = new.start_date
                                       AND    end_date = new.end_date;
                        DELETE
                                FROM   leaves_applied
                                WHERE  ftct_username = new.ftct_username
                                       AND    start_date = CURRENT_DATE - 1
                                       AND    end_date = CURRENT_DATE   - 1;
                        DELETE
                                FROM   leaves_applied
                                WHERE  ftct_username = new.ftct_username
                                       AND    start_date = CURRENT_DATE + 366
                                       AND    end_date = CURRENT_DATE   + 366
                        RAISE exception 'If you add this leave, you will not have 2 x 150 days
                                of work!';
                ENDIF;
        DELETE
                FROM   leaves_applied
                WHERE  ftct_username = new.ftct_username
                       AND    start_date = CURRENT_DATE - 1
                       AND    end_date = CURRENT_DATE   - 1;
        DELETE
                FROM   leaves_applied
                WHERE  ftct_username = new.ftct_username
                       AND    start_date = CURRENT_DATE + 366
                       AND    end_date = CURRENT_DATE   + 366;RETURN new;
        ENDIF;
        END;
          $$
        language 'plpgsql';

CREATE TRIGGER tr_check_satisfy_2x150days after INSERT OR UPDATE
ON leaves_applied FOR each row EXECUTE PROCEDURE func_check_satisfy_2x150days();
```

6.2 Leaves Application - Check for dates overlap

This trigger is used to enforce that there are no overlapping *Leaves* within the leaves_applied table for the same ftct_username. If there is an overlapping *Leave*, the update to the leaves_applied table will be rejected and an exception will be raised. This is necessary as a separate trigger because we need to exclude the original dates of the *Leaves* from the leaves_applied table, before we can consider for any other possible overlaps.

```
CREATE FUNCTION func_check_leaves_date_overlap_update() returns TRIGGER
AS
  $$
BEGIN
        IF ( EXISTS (
                SELECT 1
                FROM ( SELECT *
                       FROM   leaves_applied
                       EXCEPT
                       SELECT *
                       FROM   leaves_applied
                       WHERE  ftct_username = old.ftct_username
```

```
                            AND start_date = old.start_date
                            AND end_date = old.end_date ) AS l
                WHERE  new.ftct_username = l.ftct_username
                       AND (new.start_date <= l.end_date AND l.start_date <= new.end_date)
            )
        )
THEN
    raise exception
    'The updated leave must not overlap with any current leaves. OLD start: %, OLD end: %,
NEW start: %, NEW end: %', old.start_date, old.end_date, new.start_date, new.end_date;

END IF;
RETURN new;END;$$ language 'plpgsql';

CREATE TRIGGER tr_check_leaves_date_overlap_update before
UPDATE
ON leaves_applied FOR each row EXECUTE PROCEDURE func_check_leaves_date_overlap_update();
```

6.3 Accepting Bids - Check for pets in multiple successful bids

This trigger is used to enforce that *CareTakers* can only accept bids if they have less than 5 *Pets* under their care at that time if they are a *FullTime CareTaker,* less than 5 *Pets* if they are a *PartTime CareTaker* with ratings of 4 or higher, and less than 2 *Pets* if they are a *PartTime CareTaker* with ratings of less than 4.

```
CREATE FUNCTION func_check_bids_before() returns TRIGGER
AS
  $$
BEGIN
       IF ( EXISTS (
              ( SELECT   1
                FROM     bids b1
                WHERE    new.caretaker_username = b1.caretaker_username
                       AND new.caretaker_username IN ( SELECT username
                                                       FROM   fulltime_caretakers )
                        AND ( new.start_date <= b1.end_date AND new.end_date >= b1.start_date)
                        AND b1.issuccessful = true
                GROUP BY (new.caretaker_username)
                HAVING   count(*) >= 5 )
                UNION
              ( SELECT   1
                FROM     bids b2
                WHERE    new.caretaker_username = b2.caretaker_username
                       AND new.caretaker_username IN ( SELECT username
                                                       FROM   parttime_caretakers )
                       AND (new.start_date <= b2.end_date AND new.end_date >= b2.start_date)
                       AND b2.issuccessful = true
                        GROUP BY (new.caretaker_username)
                HAVING CASE WHEN ( SELECT avg(rating)
                                   FROM   bids b3
                                   WHERE new.caretaker_username = b3.caretaker_username) >= 4
                             THEN count(new.caretaker_username) >= 5
                           ELSE count(new.caretaker_username) >= 2
                END )
            ) AND new.issuccessful = true )
THEN
    raise exception 'You are unable to accepts anymore bids as you have reached the maximum
limit during the period of time.';
  END IF;
RETURN new;
END;
```

```
  $$
language 'plpgsql';

CREATE TRIGGER tr_check_bids_before before UPDATE
ON bids FOR each row EXECUTE PROCEDURE func_check_bids_before();
```

7. Complex Queries

7.1 Getting Required CareTakers via Filtering

This query is to retrieve all *FullTime* and *PartTime CareTakers* that fulfils all requirements set by *Pet Owner*. Such requirements include being able to care for that pet type, having a price lower than the maximum price and having an availability period that coincides with time period stated by P*et Owner*.

We select *PartTime CareTakers* that fulfils all these constraints below:

1. If their average rating is less than 4, then that *PartTime CareTaker* has less than 2 *Pets* in their care during the time period stated by the *Pet Owner*. Else, that *PartTime CareTaker* has less than 5 *Pets* in their care during that time period.
2. There is an entry in the Availabilities table for that *PartTime CareTaker* that fulfils all requirements stated by *Pet Owner.*

We select *FullTime CareTakers* that fulfils all these constraints below:

1. The *CareTaker* is not on *Leave* during the time period stated by the *Pet Owner*.
2. The base price for that pet type multiplied by the average ratings for past successful bids for that *CareTaker* is less than the maximum price stated by the *Pet Owner*.
3. The *CareTaker* has less than 5 *Pets* in their care during the time period stated by the *Pet Owner*.

```
SELECT a.username,
       a.advertised_price,
       a.start_date,
       a.end_date
FROM   availabilities a,
       (
              SELECT username AS caretaker_username
              FROM   parttime_caretakers
              EXCEPT
              SELECT   b.caretaker_username AS caretaker_username
              FROM     bids b
              WHERE    issuccessful
              AND      b.start_date <= '${end_date}'
              AND      b.end_date >= '${start_date}'
              GROUP BY b.caretaker_username
              HAVING
                       CASE
                                WHEN
                                         (
                                                SELECT Avg(rating)
                                                FROM   bids b1
                                                WHERE  b1.caretaker_username =
b.caretaker_username) >= 4 THEN Count(b.caretaker_username) >= 5
                                ELSE Count(b.caretaker_username) >= 2
                       END) canbid
```

```sql
WHERE   a.username IN
        (
                SELECT *
                FROM   parttime_caretakers)
AND     a.advertised_price <= ${maximum_price}
AND     a.pet_type = '${pet_type}'
AND     a.start_date <= '${start_date}'
AND     a.end_date >= '${end_date}'
AND     a.username = canbid.caretaker_username
UNION
SELECT bd.ftct_username AS username,
        bd.base_price *
        (
                SELECT
                        CASE
                                WHEN avg(rating) IS NULL THEN 1
                                ELSE avg(rating)
                        END
                FROM   bids
                WHERE  bd.ftct_username = bids.caretaker_username
                AND    rating IS NOT NULL
                AND    issuccessful = true) AS advertised_price,
        '${start_date}'                     AS start_date,
        '${end_date}'                       AS end_date
FROM    base_dailys bd,
        (
                SELECT username
                FROM   fulltime_caretakers
                EXCEPT
                SELECT ftct_username
                FROM   leaves_applied leave2
                WHERE  leave2.start_date <= '${end_date}'
                AND    leave2.end_date >= '${start_date}') notonleave,
        (
                SELECT username AS caretaker_username
                FROM   fulltime_caretakers
                EXCEPT
                SELECT   b3.caretaker_username
                FROM     bids b3
                WHERE    issuccessful
                AND      b3.start_date <= '${end_date}'
                AND      b3.end_date >= '${start_date}'
                GROUP BY b3.caretaker_username
                HAVING   count(b3.caretaker_username) >= 5) notoverbooked
WHERE   bd.ftct_username = notonleave.username
AND     notoverbooked.caretaker_username = bd.ftct_username
AND     bd.pet_type = '${pet_type}'
AND     bd.base_price *
        (
                SELECT
                        CASE
                                WHEN avg(rating) IS NULL THEN 1
                                ELSE avg(rating)
                        END
                FROM   bids
                WHERE  bd.ftct_username = bids.caretaker_username
                AND    rating IS NOT NULL
                AND    issuccessful = true) <= ${maximum_price}
```

7.2 Get Admin Info

This query retrieves the aggregated *CareTaker* information viewable by the *PCS Admin*. This includes each *CareTaker's* username, job_type, number of pets taken care of in this month and each of their projected salary up till the time the query is made.

```
SELECT cts.username,
       cts.job_type,
       Count(b.pet_name) AS num_pets,
       CASE WHEN job_type = 'Full Time' THEN
              CASE WHEN Sum(b.end_date - b.start_date) > 60 THEN 3000 +
                     CASE WHEN Sum(b.end_date - b.start_date) > 60 THEN
                            (SELECT Sum(b2.price)
                                FROM   bids AS b2
                                WHERE  b2.caretaker_username = cts.username
                                AND    b2.start_date >= Date_trunc('month', CURRENT_DATE)
                                      + interval '60 days'
                                AND    b2.end_date < now()
                                AND    b2.issuccessful )
                            ELSE 0
                     END
                   ELSE 3000
              END
            WHEN job_type = 'Part Time' THEN 0.75 * sum(COALESCE(b.price, 0))
       END AS salary
FROM      (
                SELECT username,
                       'Full Time' AS job_type
                FROM   fulltime_caretakers
                UNION
                SELECT username,
                       'Part Time' AS job_type
                FROM   parttime_caretakers ) AS cts
LEFT JOIN bids b
ON        b.caretaker_username = cts.username
AND       b.issuccessful
AND       b.start_date >= date_trunc('month', CURRENT_DATE)
AND       b.end_date < CURRENT_DATE
GROUP BY  cts.username,
          cts.job_type
ORDER BY  num_pets DESC
```

7.3 Get Caretaker Profile Info

This query retrieves the profile information for one *CareTaker*. This includes their job_type, their pet_days (defined as number of *Pets* taken care of for how many days) and their salary, which is calculated based on the given project requirements for both *FullTime* and *PartTime CareTakers*.

```
SELECT info.job_type,
       info.pet_days,
       COALESCE(
       CASE WHEN job_type = 'Full Time' THEN
            CASE WHEN pet_days > 60 THEN 3000 + excess_price
                 ELSE 3000
            END
            WHEN job_type = 'Part Time' THEN 0.75 * total_price
            ELSE 0
       END, 0) AS salary
```

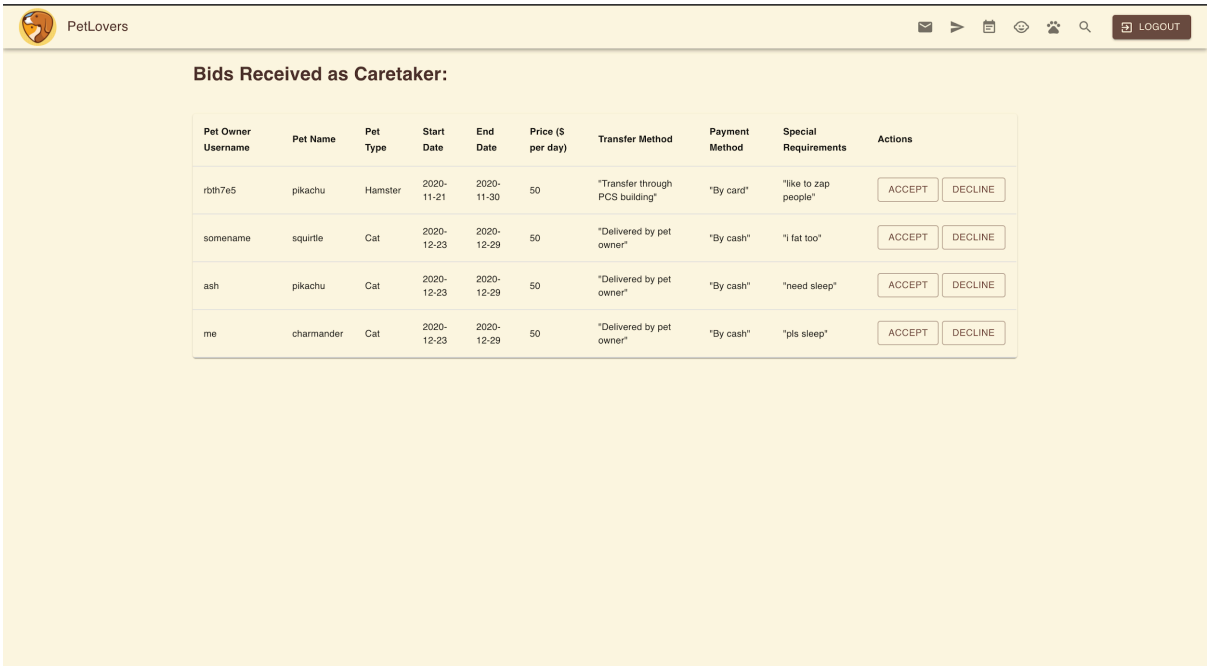
```sql
FROM    (
        SELECT *
        FROM    ( SELECT CASE
                       WHEN '${username}' IN
                              ( SELECT * FROM fulltime_caretakers) THEN 'Full Time'
                       WHEN '${username}' IN
                              ( SELECT * FROM   parttime_caretakers) THEN 'Part Time'
                       END AS job_type ) AS jt,
                ( SELECT COALESCE(Sum(b1.end_date - b1.start_date), 0) AS pet_days,
                      Sum(b1.price) AS total_price,
                      CASE WHEN Sum(b1.end_date - b1.start_date) > 60 THEN
                             ( SELECT Sum(b2.price)
                               FROM   bids AS b2
                               WHERE  b2.caretaker_username = '${username}'
                                      AND b2.start_date >= Date_trunc('month',
                                                   CURRENT_DATE) + interval '60 days'
                                      AND    b2.end_date < now()
                                      AND    b2.issuccessful )
                              ELSE 0
                       END  AS excess_price
                FROM   bids AS b1
                WHERE  b1.caretaker_username = '${username}'
                AND    b1.start_date >= date_trunc('month', CURRENT_DATE)
                AND    b1.end_date < CURRENT_DATE
                AND    b1.issuccessful ) AS oi ) AS info
```

8. Specification of software tools /frameworks used

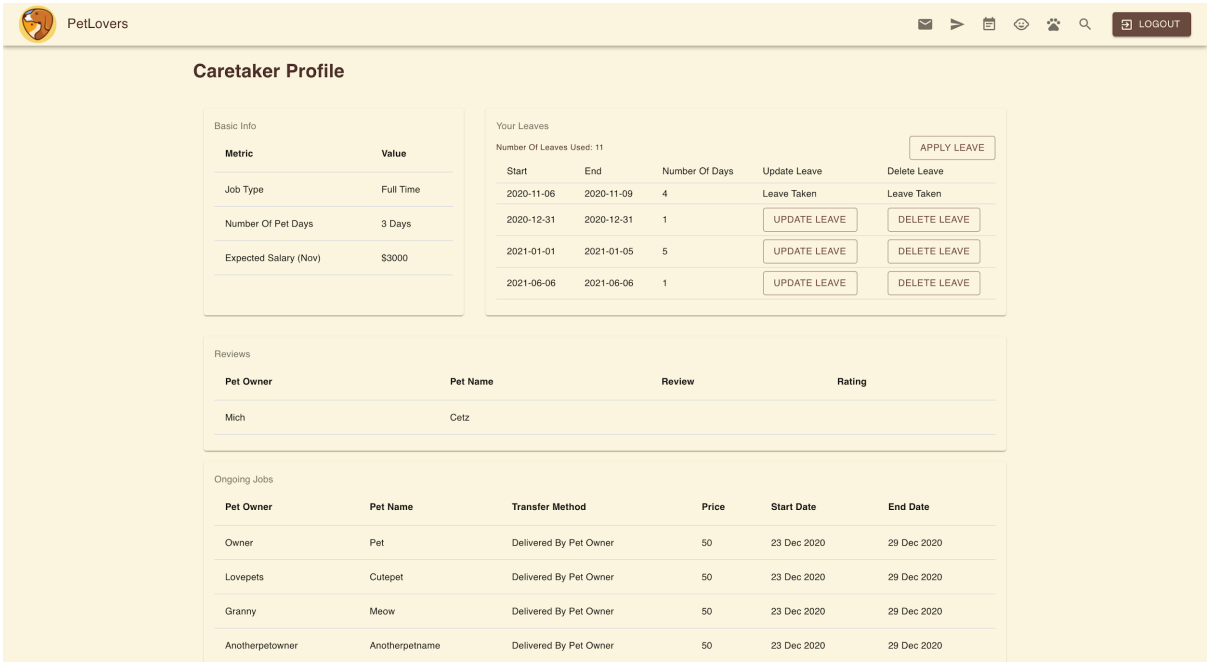
Category	Software Tools/Frameworks
Front-End	ReactJS Redux Bootstrap Material-UI
Back-end	ExpressJS NodeJS
Database	PostgreSQL
CI/CD	Travis

9. User Interface (UI)

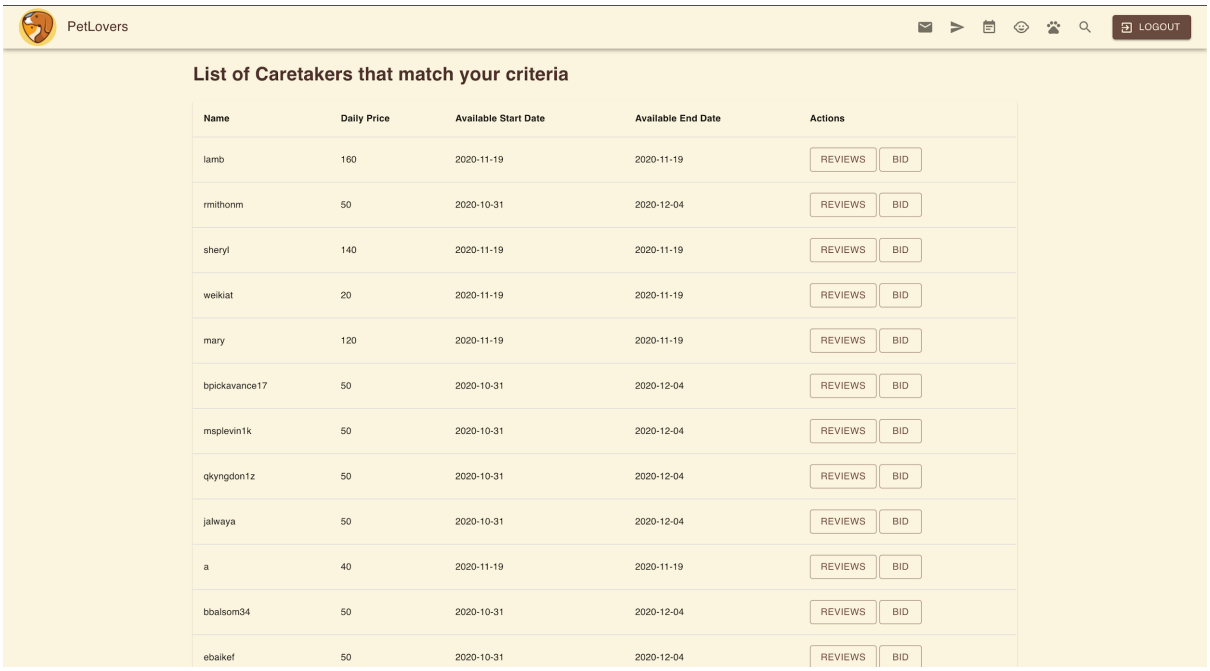
9.1 Viewing Bids Received



9.2 Viewing Caretaker Profile



9.3 Viewing Available Caretakers After Filtering



Name	Daily Price	Available Start Date	Available End Date	Actions
lamb	160	2020-11-19	2020-11-19	REVIEWS BID
rmithonm	50	2020-10-31	2020-12-04	REVIEWS BID
sheryl	140	2020-11-19	2020-11-19	REVIEWS BID
weikiat	20	2020-11-19	2020-11-19	REVIEWS BID
mary	120	2020-11-19	2020-11-19	REVIEWS BID
bpickavance17	50	2020-10-31	2020-12-04	REVIEWS BID
msplevin1k	50	2020-10-31	2020-12-04	REVIEWS BID
qkyngdon1z	50	2020-10-31	2020-12-04	REVIEWS BID
jalwaya	50	2020-10-31	2020-12-04	REVIEWS BID
a	40	2020-11-19	2020-11-19	REVIEWS BID
bbalsom34	50	2020-10-31	2020-12-04	REVIEWS BID
ebaikef	50	2020-10-31	2020-12-04	REVIEWS BID

10. Summary

10.1 Difficulties encountered

1. There were problems in deciding how we wanted our schema to be like, especially for the bids and the availability table. This was mostly due to the fact that we were not sure how to best ensure 3NF and BCNF at that time of building the schema.
2. As there are different components of the application, it was hard for us to assign work as we had to assume that certain other features of the application were working when we were making the queries.
3. Since most of us did not have experience with front-end web development, learning how to do so proved more difficult than the actual SQL queries itself, and took up a lot more time than expected. With all the bugs in the frontend, it gave us little time to refine our queries and the actual database component, which was supposed to be our main focus.

10.2 Lessons learnt

1. We were able to experience a full stack web development process, and learnt many front-end development tools like ReactJS.
2. Planning of the database and the relational schema is paramount before actually embarking on the coding, as it can avoid any unnecessary problems that pop out midway.
3. SQL can take over many of the data processing requirements from Javascript code, making the whole application faster.