

# CS2102 Final Report - Team #14 (YogaPets)

Deployed website: <https://cs2102-ay2021-s1-team14.herokuapp.com/>

Github Repo: [https://github.com/CS2102-AY2021-Team14/cs2102\\_2021\\_s1\\_team14\\_project](https://github.com/CS2102-AY2021-Team14/cs2102_2021_s1_team14_project)

## Login credentials to test out our website with:

Pet Owner - Username: **jsmullen6**, Password: **pw**

Full Time Caretaker - Username: **ntanti0**, Password: **pw**

Part Time Caretaker - Username: **ashyram0**, Password: **pw**

PCS Admin - Username: **a1**, Password: **pw**

## 1. Team Members

- Chua Jia Cheng, Jon A0135440R
- Roby Tanama A0187563R
- See Zi Yang A0183729R
- Koh Wee Lun, Clarence A0187535U
- Low Jun Kai, Sean A0183823B



## 2. Roles and Responsibilities

### Chua Jia Cheng, Jon

- Frontend authentication pages and frontend routing, sign in and register pages prototyping
- PCS Admin Base Daily Prices and Employee of the Month Features
- Ideation of Interesting queries and SQL Triggers, Generation of Seed data

### Roby Tanama

- Caretaker Front-End and Back-End Features
- Caretaker API integration
- Prototyping for Front-End

### See Zi Yang

- Setting Up for Backend on Heroku
- Setting Up for SQL Tables and Relations
- Pet Owner Back-End and Front-End Features

### Koh Wee Lun, Clarence

- User Sign-In and Registration
- PCS Admin Back-End and Front-End Features
- Bidding System

### Sean Low Jun Kai

- Handling SQL Queries and Triggers
- Ideation of Interesting queries and SQL Triggers
- Pet Owner Back-End and Front-End Features

### 3. Data Requirements and Functionalities, Data Constraints

#### Data Requirements and Functionalities

Interesting aspects of YogaPets' functionalities and implementation are in blue.

##### General Users (All Users)

- Username (varchar)
- Password (varchar)
- Name (varchar)
- Email (varchar)
- Address (varchar)
- Country (varchar)
- Can login with username and password

##### Caretakers (All Caretakers)

- isPartTime (boolean)
- Introduction (varchar)
- Can specify their availability, which includes the days they are available, the types of pets they can care for
- Can take leaves, which means they are not available to care for pets on the days they are on leave
- Will receive a salary each month

##### Full-Time Caretakers

- Automatically assigned bids
- Daily price for each pet type set by PCS Admin and is proportional to their rating

##### Part-Time Caretakers

- Can accept bids
- Can specify their daily price for each pet type

##### Pet Owners

- Can have pets
- Can browse for caretakers
- Can bid for caretakers to take care of their pets
- Can specify payment type and transfer method for successful bids
- Can review caretakers' service after successfully bidding and having caretaker take of their pet
- Can look at the caretaker they are most obsessed with (the unique caretaker you always engage the service with)

##### Pets

- Name (varchar)
- Type (enum)
- Can have special requirements
- Can belong to categories

## PCS Admins

- Can set base daily prices of each type of pet
- Can look at the number of pets of each pet type taken care of each month
- Can look at the employee of the month
- Can look at underperforming employees

## Bids

- Pet name (enum)
- Pet type (enum)
- Owner (varchar)
- Caretaker (varchar)
- Start date (date)
- End date (date)
- Price (numeric)
- isActive (boolean)
- isSuccessful (boolean)
- Payment type (varchar)
- Transfer method (varchar)
- Rating (int)
- Review text (varchar)

## Data Constraints

### General User Constraints

- username cannot be NULL
- password cannot be NULL

### General Caretaker Constraints

- isPartTime cannot be NULL
- Can care for  $\geq 1$  Pet category

### Full-Time Caretaker Constraints

- Full-time Caretaker must work for a minimum of 2 x 150 consecutive days a year (since it includes weekends, so there are plenty of leave days available)
- Full-time Caretaker are treated as available until they apply for leave
- Full-Time Caretaker cannot apply for leave if there is at least one Pet under their care.
- Full-time Caretaker have a limit of up to 5 Pet at any one time
- Daily price cannot be lower than base price determined by PCS Admin, and is proportional to their rating
- Full-Time Caretaker will receive a salary of \$3000 per month for up to 60 pet-days. For any excess pet-day, they will receive 80% of their price as bonus.

### Part-Time Caretaker Constraints

- Part-time Caretaker, should have their current year's and the next year's availability specified, both cannot be NULL

- 75% is paid to Part-time Caretaker (the other 25% is taken away by YogaPets PCS)
- Part-time Caretaker cannot take care of more than 2 Pet unless they have a good rating ( $\geq 4.0$ )
- Cannot have more than 5 Pet regardless of rating

### Pet Owner Constraints

- Pet Owner can own 0 or more pets
- Pet Owner can bid for multiple Caretakers for each pet
- Pet Owner cannot have 2 pets with the same name

### Pet Constraints

- Pet must be owned by a Pet Owner
- Pet must have only 1 type (either cat, dog, bird, rabbit, rodent, fish, insect, or turtle)
- Pet can have 0 or more Special Requirements
- Pet can belong to 0 or more categories (e.g. things like the size of pet and pet breed)

### Prices Constraints

- Base Daily Price cannot be NULL
- Base Daily Price must be greater than 0

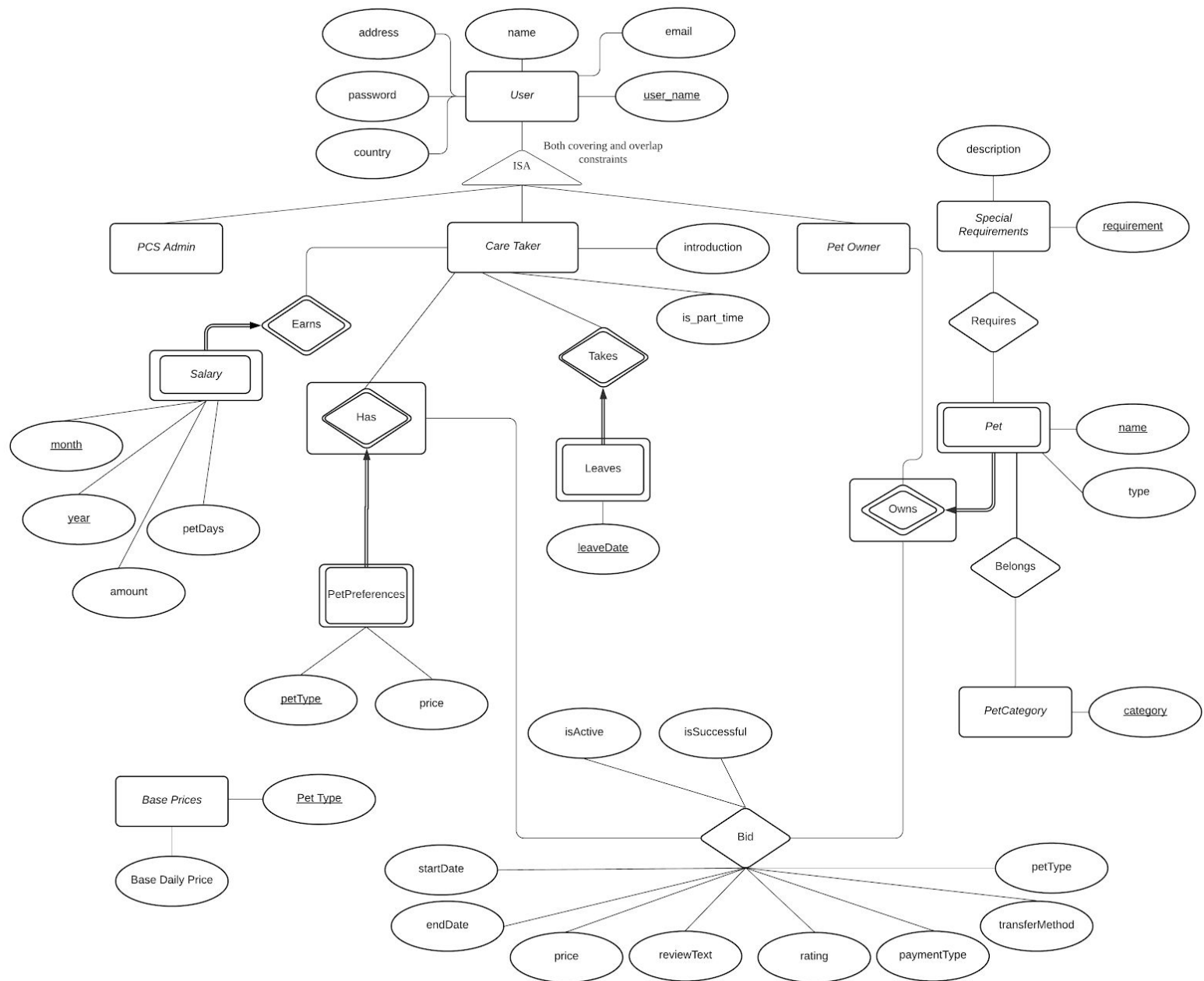
### Salary Constraints

- Month cannot be NULL
- Year cannot be NULL
- Pet days cannot be NULL and must be  $\geq 0$
- Amount cannot be NULL and must be  $\geq 0$

### Bid Constraints

- Start date  $\leq$  End date
- Price  $> 0$
- Successful Bidder chosen by Caretaker, however, Full-Time Caretakers must accept job if available
- isActive cannot be NULL
- isSuccessful cannot be NULL
- Once a bid is accepted, other bids' isActive is changed to false for the same Caretaker in the same period
- Payment Type cannot be NULL once bid is successful
- Transfer Method cannot be NULL once bid is successful
- Pet Owner can only post review after the care-taking period
- $0 \leq \text{Rating} \leq 5$

## 4. ER Data Model



### Constraints not captured by ER model

#### General User Constraints

- password cannot be NULL

#### General Caretaker Constraints

- isPartTime cannot be NULL

### Full-Time Caretaker Constraints

- Full-time Caretaker must work for a minimum of 2 x 150 consecutive days a year (since it includes weekends, so there are plenty of leave days available)
- Full-time Caretaker are treated as available until they apply for leave
- Full-Time Caretaker cannot apply for leave if there is at least one Pet under their care.
- Full-time Caretaker have a limit of up to 5 Pet at any one time
- Daily price cannot be lower than base price determined by PCS Admin, and is proportional to their rating
- Full-Time Caretaker will receive a salary of \$3000 per month for up to 60 pet-days. For any excess pet-day, they will receive 80% of their price as bonus.

### Part-Time Caretaker Constraints

- Part-time Caretaker, should have their current year's and the next year's availability specified
- 75% is paid to Part-time Caretaker (the other 25% is taken away by YogaPets PCS)
- Part-time Caretaker cannot take care of more than 2 Pet unless they have a good rating ( $\geq 4.0$ )
- Cannot have more than 5 Pet regardless of rating

### Prices Constraints

- Base Daily Price must be greater than 0

### Salary Constraints

- Pet days must be  $\geq 0$
- Amount must be  $\geq 0$

### Bid Constraints

- Start date  $\leq$  End date
- Price  $> 0$
- Successful Bidder chosen by Caretaker, however, Full-Time Caretakers must accept job if available
- isActive cannot be NULL
- isSuccessful cannot be NULL
- Once a bid is accepted, other bids' isActive is changed to false for the same Caretaker in the same period
- Payment Type cannot be NULL once bid is successful
- Transfer Method cannot be NULL once bid is successful
- Pet Owner can only post review after the care-taking period
- $0 \leq \text{Rating} \leq 5$

## 5. Relational Schema

```
CREATE TABLE IF NOT EXISTS users (  
    user_name      VARCHAR(255)    PRIMARY KEY NOT NULL,  
    name           VARCHAR(255),  
    user_email     VARCHAR(255),  
    user_password  VARCHAR(255)    NOT NULL,  
    user_country   VARCHAR(255),  
    user_address   VARCHAR(255)  
);  
  
CREATE TABLE IF NOT EXISTS pet_owners (  
    user_name      VARCHAR(255)    PRIMARY KEY REFERENCES users(user_name)  
);  
  
CREATE TABLE IF NOT EXISTS care_takers (  
    user_name      VARCHAR(255)    PRIMARY KEY REFERENCES users(user_name),  
    is_part_time   BOOLEAN          NOT NULL,  
    introduction   VARCHAR  
);  
  
CREATE TABLE IF NOT EXISTS pcs_admins (  
    user_name      VARCHAR(255)    PRIMARY KEY REFERENCES users(user_name)  
);  
  
CREATE TABLE IF NOT EXISTS pets (  
    name           VARCHAR(255)    NOT NULL,  
    owner          VARCHAR(255)    NOT NULL REFERENCES pet_owners(user_name)  
        ON DELETE CASCADE,  
    type           pet_type        NOT NULL,  
    PRIMARY KEY (name, owner)  
);  
  
CREATE TABLE IF NOT EXISTS pet_category (  
    name           VARCHAR(255),  
    owner          VARCHAR(255),  
    category       VARCHAR(255)    NOT NULL,  
    FOREIGN KEY (name, owner) REFERENCES pets(name, owner) ON DELETE CASCADE,  
    PRIMARY KEY (name, owner, category)  
);  
  
CREATE TABLE IF NOT EXISTS pet_special_requirements (  
    name           VARCHAR(255),  
    owner          VARCHAR(255),  
    requirement     VARCHAR        NOT NULL,  
    description     VARCHAR,  
    FOREIGN KEY (name, owner) REFERENCES pets(name, owner) ON DELETE CASCADE,  
    PRIMARY KEY (name, owner, requirement)  
);  
  
CREATE TABLE IF NOT EXISTS base_prices (  
    pet_type       pet_type        PRIMARY KEY,
```

```

    base_price      NUMERIC(10, 2) NOT NULL CHECK (base_price > 0)
);

CREATE TABLE IF NOT EXISTS care_taker_leaves (
    care_taker      VARCHAR(255) NOT NULL REFERENCES care_takers(user_name)
        ON DELETE CASCADE,
    leave_date      DATE NOT NULL,
    PRIMARY KEY (care_taker, leave_date)
);

CREATE TABLE IF NOT EXISTS care_takers_pet_preferences (
    care_taker      VARCHAR(255) NOT NULL REFERENCES care_takers(user_name)
        ON DELETE CASCADE,
    pet_type        pet_type NOT NULL,
    price           NUMERIC(10, 2) CHECK (price > 0), -- for part-timers
    PRIMARY KEY (care_taker, pet_type)
);

CREATE TABLE IF NOT EXISTS bids (
    pet             VARCHAR(255) NOT NULL,
    owner           VARCHAR(255) NOT NULL,
    care_taker      VARCHAR(255) NOT NULL,
    pet_type        pet_type NOT NULL,
    start_date      DATE NOT NULL,
    end_date        DATE NOT NULL,
    price           NUMERIC(10, 2) NOT NULL CHECK (price > 0),
    is_active       BOOLEAN NOT NULL DEFAULT true,
    is_successful   BOOLEAN NOT NULL DEFAULT false,
    payment_type    VARCHAR(255),
    transfer_method VARCHAR(255),
    rating          INT CHECK (rating >= 0 AND rating <= 5),
    review_text     VARCHAR,
    PRIMARY KEY (pet, care_taker, start_date, end_date),
    FOREIGN KEY (pet, owner) REFERENCES pets(name, owner),
    FOREIGN KEY (care_taker, pet_type)
        REFERENCES care_takers_pet_preferences(care_taker, pet_type),
    CONSTRAINT valid_date_range CHECK (start_date <= end_date),
    CONSTRAINT successful_bid_constraint CHECK
        ((NOT is_successful) OR
        (payment_type IS NOT NULL AND transfer_method IS NOT NULL))
);

CREATE TABLE IF NOT EXISTS salary (
    care_taker      VARCHAR(255) NOT NULL REFERENCES care_takers(user_name)
        ON DELETE CASCADE,
    month           CHAR(3) NOT NULL, -- 3 letter month
    year            CHAR(4) NOT NULL, -- 4 letter year
    pet_days        INT NOT NULL DEFAULT 0 CHECK (pet_days >= 0),
    amount          NUMERIC NOT NULL DEFAULT 0 CHECK (amount >= 0),
    PRIMARY KEY (care_taker, month, year)
);

```



## Constraints not enforced by SQL relational schema

Constraints in blue are enforced by triggers.

### Full-Time Caretaker Constraints

- **Full-time Caretaker must work for a minimum of 2 x 150 consecutive days a year (since it includes weekends, so there are plenty of leave days available)**
- **Full-time Caretaker are treated as available until they apply for leave**
- **Full-Time Caretaker cannot apply for leave if there is at least one Pet under their care.**
- **Full-time Caretaker have a limit of up to 5 Pet at any one time**
- Daily price cannot be lower than base price determined by PCS Admin, and is proportional to their rating (enforced by view and insert statement)
- Full-Time Caretaker will receive a salary of \$3000 per month for up to 60 pet-days. For any excess pet-day, they will receive 80% of their price as bonus.

### Part-Time Caretaker Constraints

- Part-time Caretaker, should have their current year's and the next year's availability specified
- **75% is paid to Part-time Caretaker (the other 25% is taken away by YogaPets PCS)**
- **Part-time Caretaker cannot take care of more than 2 Pet unless they have a good rating ( $\geq 4.0$ )**
- **Cannot have more than 5 Pet regardless of rating**

### Bid Constraints

- **Successful Bidder chosen by Caretaker, however, Full-Time Caretakers must accept job if available**
- **Once a bid is accepted, other bids' isActive is changed to false for the same Caretaker in the same period**
- **Payment Type cannot be NULL once bid is successful**
- **Transfer Method cannot be NULL once bid is successful**
- Pet Owner can only post review after the care-taking period

## 6. Database Normalization

Table	Normal Form (Let F represent the set of functional dependencies of each table)
<b>users</b> ( <u>user_name</u> , name, user_email, user_password, user_country, user_address)	Primary key: user_name F- = { user_name -> name; user_name -> user_email; user_name -> user_password; user_name -> user_country; user_name -> user_address } Since the LHS of every FD in F+ is a superkey, this table is in BCNF.
<b>pet_owners</b> ( <u>user_name</u> )	Primary key: user_name Since every FD in F+ is trivial, this table is in BCNF.
<b>care_takers</b> ( <u>user_name</u> , is_part_time, introduction)	Primary key: user_name F-: { user_name -> is_part_time; user_name -> introduction } Since the LHS of every FD in F+ is a superkey, this table is in BCNF.

<b>pcs_admins</b> ( <u>user_name</u> )	Primary key: user_name Since every FD in F+ is trivial, this table is in BCNF.
<b>pets</b> ( <u>name</u> , <u>owner</u> , type)	Primary key: (name, owner) F- = { name, owner -> type } Since the LHS of every FD in F+ is a superkey, this table is in BCNF.
<b>pet_category</b> ( <u>name</u> , <u>owner</u> , <u>category</u> )	Primary key: (name, owner, category) Since every FD in F+ is trivial, this table is in BCNF.
<b>pet_special_requirements</b> ( <u>name</u> , <u>owner</u> , <u>requirement</u> , description)	Primary key: (name, owner, requirement) F- = { name, owner, requirement -> description } Since the LHS of every FD in F+ is a superkey, this table is in BCNF.
<b>base_prices</b> ( <u>pet_type</u> , base_price)	This table has only 2 attributes. By lemma 2, this table is trivially in BCNF.
<b>care_taker_leaves</b> ( <u>care_taker</u> , <u>leave_date</u> )	This table has only 2 attributes. By lemma 2, this table is trivially in BCNF.
<b>care_takers_pet_preferences</b> ( <u>care_taker</u> , <u>pet_type</u> , price)	Primary key: (care_taker, pet_type) F-: { care_taker, pet_type -> price } Since the LHS of every FD in F+ is a superkey, this table is in BCNF.
<b>bids</b> ( <u>pet</u> , <u>owner</u> , <u>care_taker</u> , <u>pet_type</u> , <u>start_date</u> , <u>end_date</u> , price, is_active, is_successful, payment_type, transfer_method, rating, review_text)	Primary key: (pet, care_taker, start_date, end_date) F- = { pet, care_taker, start_date, end_date -> owner; pet, care_taker, start_date, end_date -> pet_type; pet, care_taker, start_date, end_date -> price; pet, care_taker, start_date, end_date -> is_active; pet, care_taker, start_date, end_date -> is_successful; pet, care_taker, start_date, end_date -> payment_type; pet, care_taker, start_date, end_date -> transfer_method; pet, care_taker, start_date, end_date -> rating; pet, care_taker, start_date, end_date -> review_text } Since the LHS of every FD in F+ is a superkey, this table is in BCNF.
<b>salary</b> ( <u>care_taker</u> , <u>month</u> , <u>year</u> , <u>pet_days</u> , amount)	Primary key: (care_taker, month, year) F-: { care_taker, month, year -> pet_days; care_taker, month; year -> amount } Since the LHS of every FD in F+ is a superkey, this table is in BCNF.

Since all tables are in BCNF, the database is **definitely** in BCNF.

## 7. Triggers

### 1. Checking for Bid Availability

This Trigger simply checks given a new **bid to be inserted**, if any of the caretaker's **leave days fall within the** start and end date of the bid. If the bid **should not be added**.

```
CREATE OR REPLACE FUNCTION checkBidAvailabilityFunction()
RETURNS TRIGGER AS
$$ BEGIN
IF ((SELECT COUNT(*)
      FROM care_taker_leaves
      WHERE care_taker = NEW.care_taker AND leave_date >= NEW.start_date
      AND leave_date <= NEW.end_date)
    = 0) THEN
RETURN NEW;
END IF;
RETURN NULL;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER checkBidAvailability
BEFORE INSERT ON bids
FOR EACH ROW
EXECUTE PROCEDURE checkBidAvailabilityFunction();
```

### 2. Auto-Acceptance of Bids

This Trigger simply checks given a new **bid to be inserted**, if between the start and end date of the **new bid**, there are **less than 5 successful bids belonging to the caretaker**.

This indicates that the caretaker **still has room** to accept more bids. As a result, the caretaker **should immediately accept the bid**, and thus mark it as **a successful bid and** default payment types and transfer methods will be added to the bid.

```
CREATE OR REPLACE FUNCTION autoAcceptFullTimerBidFunction()
RETURNS TRIGGER AS
$$ BEGIN
IF (((SELECT COUNT(*)
      FROM bids
      WHERE care_taker = NEW.care_taker AND start_date >= NEW.start_date
      AND end_date <= NEW.end_date AND is_successful) < 5)
    AND
    NOT (SELECT is_part_time FROM care_takers WHERE user_name = NEW.care_taker))
THEN
NEW.is_successful := TRUE;
NEW.is_active := FALSE;
NEW.payment_type := 'Cash';
NEW.transfer_method := 'Pet Owner Deliver';
END IF;
RETURN NEW;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER autoAcceptFullTimerBid
BEFORE INSERT ON bids
FOR EACH ROW
EXECUTE PROCEDURE autoAcceptFullTimerBidFunction();
```

### **3. Rejecting a Leave Date if there are no 2 blocks of 150 consecutive working days in a year.**

This Trigger simply checks given a **request to insert a new leave day**, if leave day results in such that:

- 1. There is a possibility that there are 2 blocks of 150 consecutive working days in a year.**
- 2. At the leave date, the caretaker is taking care of 0 pets**

**This function returns the working days of the caretaker in the year assuming the leave date is taken.**

```
CREATE OR REPLACE FUNCTION availableDates(the_care_taker VARCHAR(255),
    the_leave_date date)
    RETURNS TABLE ( free_day date )
AS
$$ BEGIN
RETURN QUERY
SELECT *
FROM
(SELECT date_trunc('day', all_dates):: date AS d
    FROM generate_series
        ( (date_trunc('year', now()))::timestamp
        , (date_trunc('year', now()) + interval '1 year' - interval '1 day')::timestamp
        , '1 day'::interval) AS all_dates
EXCEPT
SELECT leave_date
    FROM care_taker_leaves c
    WHERE c.care_taker = the_care_taker
ORDER BY d) AS free_dates
WHERE free_dates.d <> the_leave_date;
END; $$
LANGUAGE plpgsql;
```

**This function returns the number of possibilities where there exists 2 blocks of 150 consecutive working days in that year if and only if the leave day has been taken.**

```
CREATE OR REPLACE FUNCTION getBlocks(the_care_taker VARCHAR(255), the_leave_date date)
    RETURNS BIGINT
AS
$$ BEGIN
RETURN (
SELECT COUNT(*)
    FROM (
SELECT COUNT(a.free_day) AS days_free, b.free_day AS start_day, c.free_day AS end_day
    FROM
        availableDates(the_care_taker, the_leave_date) AS a,
        availableDates(the_care_taker, the_leave_date) AS b,
        availableDates(the_care_taker, the_leave_date) AS c
    WHERE date(c.free_day) - date(b.free_day) = 149
        AND a.free_day >= b.free_day
        AND a.free_day <= c.free_day
    GROUP BY b.free_day, c.free_day
    HAVING COUNT(a.free_day) = 150
    ORDER BY b.free_day, c.free_day
    ) AS X, (
SELECT COUNT(a.free_day) AS days_free, b.free_day AS start_day, c.free_day AS end_day
    FROM
```

```

    availableDates(the_care_taker, the_leave_date) AS a,
    availableDates(the_care_taker, the_leave_date) AS b,
    availableDates(the_care_taker, the_leave_date) AS c
WHERE date(c.free_day) - date(b.free_day) = 149
    AND a.free_day >= b.free_day
    AND a.free_day <= c.free_day
GROUP BY b.free_day, c.free_day
HAVING COUNT(a.free_day) = 150
ORDER BY b.free_day, c.free_day
) AS Y
WHERE date(Y.start_day) >= date(X.end_day));
END; $$
LANGUAGE plpgsql;

```

**This function simply checks if the caretaker can take the leave date. This is simply done by checking if the number of Possibilities of having 2 blocks of 150 consecutive working days is possible**

```

CREATE OR REPLACE FUNCTION canTakeLeave(the_care_taker VARCHAR(255),
    the_leave_date date)
RETURNS BOOLEAN
AS
$$ BEGIN
IF ((SELECT getBlocks(the_care_taker, the_leave_date)) >= 1)
THEN
RETURN TRUE;
END IF;
RETURN FALSE;
END; $$
LANGUAGE plpgsql;

```

**This function simply gets the number of bids where each successful bid happens to clash with the leave date.**

```

CREATE OR REPLACE FUNCTION getPetsPerDay(the_care_taker VARCHAR(255), the_date date)
RETURNS INTEGER
AS
$$ BEGIN
RETURN (
    SELECT COUNT(*)
    FROM bids
    WHERE bids.care_taker = the_care_taker
        AND bids.start_date <= the_date
        AND bids.end_date >= the_date
        AND bids.is_successful = TRUE
);
END; $$
LANGUAGE plpgsql;

```

**This function simply gets the number of bids where each bid happens to clash with the leave date. If there exists a bid which clashes with the leave date, return FALSE. Else, return TRUE.**

```

CREATE OR REPLACE FUNCTION isThereClashWithBids(the_care_taker VARCHAR(255),
the_leave_date date)
RETURNS BOOLEAN

```

```

AS
$$ BEGIN
IF ((SELECT getPetsPerDay(the_care_taker, the_leave_date)) >= 1)
THEN
RETURN TRUE;
END IF;
RETURN FALSE;
END; $$
LANGUAGE plpgsql;

```

This Trigger simply returns the bid if and only if the 2 conditions are satisfied

1. **There is a possibility that there are 2 blocks of 150 consecutive working days in a year.**
2. **At the leave date, the caretaker is taking care of 0 pets. Which indicates that there is no bid which clashes which is successful and clashes with the leave date.**

```

CREATE OR REPLACE FUNCTION checkAbleToTakeLeaveFunction()
RETURNS TRIGGER AS
$$ BEGIN
IF ((SELECT canTakeLeave(NEW.care_taker, NEW.leave_date))
    AND NOT (SELECT isThereClashWithBids(NEW.care_taker, NEW.leave_date))) THEN
RETURN NEW;
END IF;
RETURN NULL;
END; $$ LANGUAGE plpgsql;

CREATE TRIGGER checkAbleToTakeLeave
BEFORE INSERT ON care_taker_leaves
FOR EACH ROW
EXECUTE PROCEDURE checkAbleToTakeLeaveFunction();

```

## 8. Complex and Interesting SQL Queries

### 1. Retrieval of Most Obsessed Caretaker of a Pet Owner

The CTE helps us to retrieve the tuples of (owner, care\_taker, max\_bid\_count) where each tuple in the CTE has the **maximum bid count** in bids, which represents the number of times the pet owner, **\$1**, has bidden for the care\_taker.

Following which, a simple query is made whereby if there is only 1 record with the maximum bid count, the pet owner **is particularly obsessed with that caretaker**. Else, the pet owner is not obsessed with any particular **one caretaker**.

```
WITH count_owner_care_takers AS (
    SELECT $1 AS owner, c.care_taker AS care_taker, c.orders AS max_bid_count
    FROM (
        (
            SELECT b.owner AS owner, b.care_taker AS care_taker, COUNT(b.care_taker) AS orders
            FROM bids b
            GROUP BY (b.owner, b.care_taker)
            HAVING owner = $1
            ORDER BY COUNT(b.care_taker) DESC )
        ) AS c
    WHERE c.orders = (
        SELECT MAX(ct.orders) AS count
        FROM (
            SELECT b.owner AS owner, b.care_taker AS care_taker,
                COUNT(b.care_taker) AS orders
            FROM bids b
            GROUP BY (b.owner, b.care_taker)
            HAVING owner = $1
            ORDER BY COUNT(b.care_taker) DESC
        ) AS ct
    )
) AS ct
SELECT
    CASE
    WHEN COUNT(c.max_bid_count) = 0 THEN 'no_obsession'
    WHEN COUNT(c.max_bid_count) > 1 THEN 'no_obsession'
    ELSE MAX(c.care_taker)
    END AS obsessed_caretaker
FROM count_owner_care_takers c;
```

## **2. Employee Of the Month**

This retrieves the caretaker at the current month where he or she **has**

- 1. the highest number of pet days worked for; 2. the highest average rating**
- 2. and the highest number of owners that he or she has worked for.**

```
SELECT employee.care_taker AS care_taker, users.name AS name
FROM ( SELECT care_taker
      FROM bids
      GROUP BY care_taker, date_part('month', end_date)
      HAVING date_part('month', end_date) = date_part('month', now())
      ORDER BY SUM(date(end_date) - date(start_date) + 1) DESC,
               AVG(rating) DESC, COUNT(owner) DESC
      LIMIT 1 ) AS employee
INNER JOIN
users ON employee.care_taker = users.user_name;
```

## **3. Underperforming Caretakers**

This retrieves the caretakers who are underperforming. The definition of underperforming are as follows:

- 1. He or she has a total average rating of less than 3.**
- 2. He or she has worked for less than an average of 60 pet days per month**
- 3. He or she has a rating of less than 2 for this current month.**

Do note that we return the **username, boolean if the caretaker is a part-timer, and an array of error types and error datas. For the error types, do refer to the criteria listed above.** For the error datas, they are as follows: **1, Total Average Rating, 2. Average Pet Days Per Month and 3. Rating for the current month.**

Do note that this is to help the **PCS admin** identify where the caretakers are lacking at.

```
SELECT users.user_name, is_part_time, name, ARRAY_AGG(error_type) AS error_types,
      ARRAY_AGG(error_data) AS error_datas FROM (
SELECT care_taker, 1 AS error_type, avg_rating AS error_data
FROM care_takers_rating
WHERE avg_rating < 3
UNION
SELECT care_taker, 2 AS error_type, (
SUM(date(end_date) - date(start_date) + 1) /
((date_part('year', MAX(end_date)) - date_part('year', MIN(start_date))) * 12
+ DATE_PART('month', MAX(end_date))
- DATE_PART('month', MIN(start_date)) + 1)
) AS error_data
FROM bids
GROUP BY care_taker
HAVING SUM(date(end_date) - date(start_date) + 1) /
((date_part('year', MAX(end_date)) - date_part('year', MIN(start_date))) * 12
+ DATE_PART('month', MAX(end_date))
- DATE_PART('month', MIN(start_date)) + 1) < 60
UNION
SELECT care_taker, 3 AS error_type, AVG(rating) AS error_data
FROM bids
GROUP BY care_taker, date_part('month', end_date)
HAVING AVG(rating) < 2
) AS t INNER JOIN care_takers ON t.care_taker = care_takers.user_name
INNER JOIN users ON t.care_taker = users.user_name
GROUP BY (users.user_name, is_part_time, name);
```



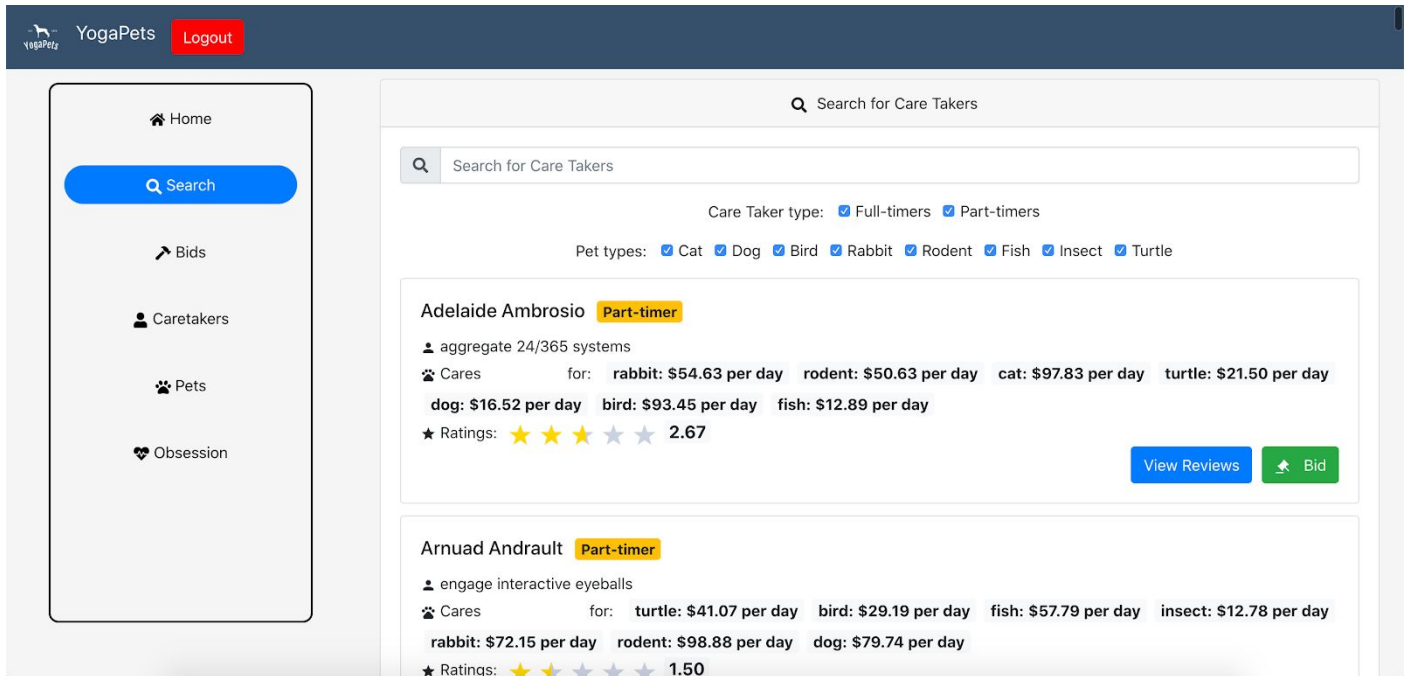
## 9. Software Specification

**Frontend:** React, React Bootstrap

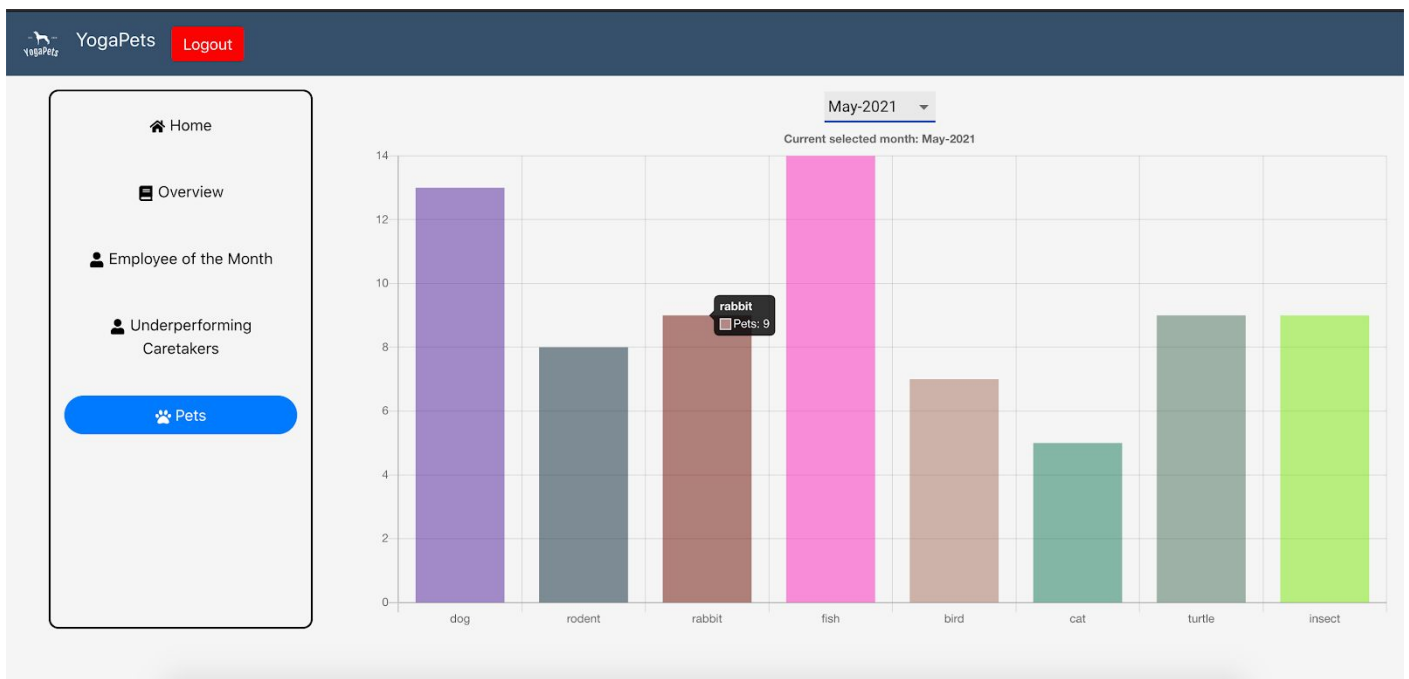
**Backend:** Node.js, Express.js

**Database:** PostgreSQL


## 10. Screenshots



*Screenshot of page for Pet Owners to search and bid for Caretakers*



*Screenshot of page for PCS Admin to view pets caring information to make important business decisions*

 YogaPets

Logout

Home

History

Salary

Offers

Availability

Settings

History

Cared for po1

Pet name: rat1  
Category: rodent  
Period: Oct 02 2020 to Oct 11 2020

10  
pet days

★

★

★

★

★

Read review

Cared for po1

Pet name: rat1  
Category: rodent  
Period: Nov 07 2020 to Nov 07 2020

1  
pet day

★

★


★

★

★

Read review

EST. 2020

 YogaPets

ftct1

Full time  
Joined on Sep 09 2020

EMPLOYED

Screenshot of page for Caretakers to view the pets they have cared for before

18

## 11. Summary

### Difficulties Encountered

Initially, we had no prior experience with regards to developing full-stack web applications. This was our first time using SQL and also PostgreSQL, a declarative language which we were not exposed to.

Moreover, it was not easy to design such a complicated database with so many constraints as we had no experience in database design. We went through many iterations of our ER data model and SQL table schemas. We were grateful for the helpful feedback given for our initial implementation of the ER diagram, and had learnt a lot from the experience.

Not only that, each of us had to pick up a front-end framework and back-end framework to build this amazing full-stack application, which resulted in a steep learning curve. All of us had the opportunity to work on both the frontend and backend for the features we are responsible for. There were interesting things and best practices that we learnt about building full-stack web applications, such as using JSON Web Tokens for authentication with the backend, building and designing RESTful APIs, frontend routing for authorized and unauthorized users, prototyping and UI design with Figma, deploying our application, and using CI/CD. We also needed to fork out some money to purchase a Heroku plan such that our database was seeded properly and users would have a pleasant experience when viewing our website.

Furthermore, we faced difficulties when trying to implement triggers to enforce our constraints. It was difficult to learn how to code the triggers and to make them do what we wanted. Thankfully, through reading many documentations and a lot of trial and error, we managed to implement the triggers to enforce our constraints.

### Lessons Learnt

We have learnt the importance of having simple database design and simple queries. It is important to understand the power of a declarative language such as SQL and not abuse it.

We also learnt how to design a good database schema without having to rely on serial IDs.

As a team, we have had our first taste of full-stack web applications and we hope to pass on this knowledge to our juniors, who might take this fun and amazing module in the future.