CS2102 AY20/21 Team 1 Project Report

<!-- For reference
MARKING SCHEME

- ER Data Model
- Relational Schema
- Interesting queries (3 most interesting to how application can improve business decision)
- Triggers for complex constraints
- User interface design
  -->

Table of Contents

Team <a name = "info"></a>

Name	Student Number	Responsibilities
Matthew Nathanael Sugiri	A0183805B	Triggers, Integration, API Development, Deployment
Joshua Tam	A0190309H	Frontend, CareTaker Features, API Development
Tan Guan Yew	A0183464Y	Frontend, Petowner Features, API Development
Sean Lim	A0187123H	Admin features,
Glen Wong	A0188100N	Frontend, Backend Salary calculation

Application's functionalities <a name = "application_description"></a>

Our web application, Pet Society, allows pet owners to search for caretakers who are able to take care of their pets, and apply for their services.
It can be viewed as a 'marketplace' for pet caring services. Here are the functions that different users have access to!

Functions

Pet owners:

- Sign up for an account and register the pets that they want to find a caretaker to care for.
- Bid for a caretaker's services when he/she browses through the caretaker search page for their registered pets
- Search through all the available services offered. They can also the filter the results of their search using a few different parameters (average rating, availability date, area of the caretaker .etc)
- Edit the details of their profile on the profile page after they have signed up (e.g update their address and area, the pets they have .etc)
- Edit pet information, or delete a registered pet
- Submit a review for the caretaker on any completed service/transaction that they were involved in
- View all the past reviews and average rating of the caretaker when they are browsing the services offered
- Submit an enquiry to the administrators of thr website in the case of any doubts or disputes
- View the answers to the enquiries that they have previously submitted to the administrator
- View their past and current transactions. They can also filter their transactions based on its status (e.g completed, rejected, .etc)

Caretakers:

- Sign up for an account as either a full timer or part timer (but not both).
- Part timers are free to indicate their availability, there is no minimum number of days they are to be available every month
- Part timers can indicate the daily price for their periods of availability, and choose which bids they want to accept
- Full timers can take leave by indicating the period they want apply for, and the system will check if they are able to do so
- Full timers are entitled to an increase the price of their services if they perform well and achieve good ratings from pet owners
- Accept or reject bids offered by the pet owners for their services
- View the ratings and reviews that the caretakers have given them for the transactions they have been involved in
- View their expected salary for the current month
- View their past and current transactions. They can also filter their transactions based on its status (e.g accepted, rejected, completed .etc)

Administrators:

- View caretakers who are underperforming
- View the total number of jobs that have been completed/in progress for the month/year
- See the total earnings that the website has made through the commissions that it takes
- View the category of pets for which more caretakers are needed
- See the distribution of their users islandwide (split up by the five areas, Central, North, Northeast, East and West)
- View the total salary that needs to be paid to the caretakers every month
- See which caretakers they are managing (every caretaker is assigned to an administrator)
- See other relevant statistics in the admin dashboard
- Change the base price of the full time caretakers under their management

Application's Data Constraints

1. Pet types are classified into categories (Dog, Cat, Fish, Rabbit, Bird, Reptile).
2. A User **can** be either a Pet Owner, Care Taker, both a Pet Owner and a Care Taker, or a PCSAdmin.
3. A Pet Owner **can** own more that one Pet.
4. A Pet Owner **can opt** to make their payment using Cash or Credit-Cards.
5. A Pet Owner **can** decide on how to transfer their pet from 3 methods (Pet owner delivery, Care Taker pick up, Transfer through the physical building of PCS).
6. A Pet Owner **cannot** request for a service if their pet does not match the Type Preference of that service offered.
7. A Pet Owner **can** submit multiple review/rating for a Care Taker if the Care Taker has taken care of the Pet Owner's Pet multiple times, including for the same pet.
8. A Pet Owner **can only** submit a review/rating after the care period has ended, for a specific transaction.
9. A Care Taker **is required** to have the pet under their care for the Entire Day (*24 Hrs*), for all Pet Days in the Transaction in which they accepted.
10. Care Takers employment types are classified into Full-Time or Part-Time.
11. A full time care taker **can opt** to take up to 65 days of leave, if they satisfy a minimum of 2 x 150 consecutive working days a year.
12. A full time care taker **cannot** take leave if they have at least 1 Pet under their care.
13. A PCSAdmin **must** manage at least one Caretaker.
14. A Care Taker **must** be managed by exactly one PCSAdmin, and is **randomly** assigned to a PCSAdmin upon registration.
15. A Care Taker's base daily price (**$50**) is determined by a PCSAdmin for each Pet Type upon registration.
16. A Care Taker **can** only take care of up to 5 Pets at any single point of time.
17. A Care Taker **must** manually *accept* or *reject* each bid, regardless if they are full time or part time.
18. A Care Taker's daily price will increase with their rating:
    - Rating between **4.0 and 4.2**, Daily Price is **$52**
    - Rating between **4.2 and 4.4**, Daily Price is **$55**
    - Rating between **4.4 and 4.6**, Daily Price is **$59**
    - Rating between **4.6 and 4.8**, Daily Price is **$64**
    - Rating between **4.8 and 5.0**, Daily Price is **$70**
19. A full time Care Taker **will** receive a salary of $3000 per month for up to 60 Pet-Day, and receive 80% of their price as bonus for any excess Pet-Day.
20. A part time Care Taker **can** specify their availability for the current year and the next year.
21. A part time Care Taker **cannot** take more than 2 Pets at any single point of time, if their rating is below 4.
22. A part time Care Taker **will** receive 75% of their price as payment.
23. All transactions history **will** be stored, regardless of whether a bid is rejected or accepted, and each transaction will have a status.

Entity Relationship Model <a name = "er_diagram"></a>

Image of final ER diagram

Constraints not shown in ER diagram:

- **Duration_to** and **duration_from** of transaction_details must be IN BETWEEN the **service_avail_from** and **service_avail_to** attributes.
- All caretakers have a limit of up to 5 Pets at any one time.
    - Full time caretakers and part time caretakers with a rating of 4/5 and higher can only participate in 5 transactions at any given time. In the ER diagram, this means that the number of transactions which have a t_status = 3 at any point in time <= 5. This constraint is enforced by a SQL Trigger.
    - Part time caretakers with a rating lower than a 4/5 can only participate in 2 transactions at any point in time. In the ER diagram, this means that the number of transactions which have a t_status = 4 at any point in time <= 2. This constraint is enforced by a SQL Trigger.
- A full time caretaker must work for a minimum of 2 x 150 consecutive days a year AND a full time caretaker is treated as available until they apply for leave. These constraints are enforced by a check performed by a SQL Function.
- A full time caretakers cannot apply for leave if there is at least one Pet under their care. This constraint is enforced by a check performed by a SQL Function.
- A caretaker should not take care of pets they cannot care for. This constraint is enforced by the foreign key in Transactions_Details onto Offers_Services.
- The daily price for a full time caretaker increases with the rating of the caretaker but will never be below the base price. This constraint is enforced by a SQL Trigger "update_fulltime_price".

Database schema <a name = "schema"></a>

The constraints that cannot be enforced using table constraints/checks are enforced using SQL Triggers.
The primary and foreign keys of the tables are indicated in the creation of each table.

Users ISA PetOwners, Caretakers, PCSAdmins schema

```
CREATE TABLE Users (
    email VARCHAR,
    full_name VARCHAR NOT NULL,
    user_password VARCHAR NOT NULL,
    profile_pic_address VARCHAR,
    user_area VARCHAR,
    user_address VARCHAR,
    is_deleted BOOLEAN DEFAULT FALSE,
    PRIMARY KEY (email)
);

CREATE TABLE PetOwners (
    owner_email VARCHAR
    REFERENCES Users(email)
    ON DELETE cascade,
    PRIMARY KEY (owner_email)
);

CREATE TABLE Caretakers(
    caretaker_email VARCHAR
    REFERENCES Users(email)
    ON DELETE cascade,
    employment_type VARCHAR NOT NULL,
    avg_rating NUMERIC DEFAULT 0,
    no_of_reviews INTEGER,
    PRIMARY KEY (caretaker_email)
);

CREATE TABLE PCSAdmins (
    admin_email VARCHAR
    REFERENCES Users(email)
    ON DELETE cascade,
    PRIMARY KEY (admin_email)
);
```

Manages and Categories schema

```
CREATE TABLE Manages (
    admin_email VARCHAR REFERENCES PCSAdmins(admin_email) ON DELETE cascade,
    caretaker_email VARCHAR REFERENCES Caretakers(caretaker_email) ON DELETE cascade,
    base_price NUMERIC DEFAULT 50,
    PRIMARY KEY (admin_email, caretaker_email)
);

CREATE TABLE Categories (
    pet_type VARCHAR PRIMARY KEY
);
```

Owns_Pets schema

```
CREATE TABLE Owns_Pets (
    owner_email VARCHAR REFERENCES PetOwners(owner_email)
    ON DELETE cascade,
    gender CHAR NOT NULL,
    pet_name VARCHAR NOT NULL,
    special_req VARCHAR,
    pet_type VARCHAR REFERENCES Categories(pet_type),
    is_deleted BOOLEAN DEFAULT FALSE,
    PRIMARY KEY (owner_email, pet_name, pet_type)
);
```

Offers_Services schema

The **is_avail** attribute denotes whether the service is valid and can be advertised to the pet owners on the website.

```
CREATE TABLE Offers_Services (
    caretaker_email VARCHAR REFERENCES Caretakers(caretaker_email)
    ON DELETE cascade,
    employment_type VARCHAR NOT NULL,
    service_avail_from DATE NOT NULL,
    service_avail_to DATE NOT NULL,
    type_pref VARCHAR NOT NULL,
    daily_price NUMERIC NOT NULL,
    is_avail BOOLEAN DEFAULT TRUE,
    PRIMARY KEY (caretaker_email, type_pref, service_avail_from, service_avail_to)
);
```

Transactions and transactions_details schema

The **t_status** attribute indicates the status of the transaction using integers.

- a **1** denotes that the transaction has just been SUBMITTED (it is a bid submitted by a pet owner to a care taker). The caretaker has not taken any action.
- a **2** denotes that the transaction is REJECTED by the caretaker (the bid from the petowner was rejected by the caretaker)
- a **3** denotes that the transaction is IN PROGRESS/ACCEPTED by the caretaker (the bid is accepted so the transaction will be performed)
- a **4** denotes that the transaction has been COMPLETED (the service has been completed by the caretaker and the pet has been returned to the pet owner)
- a **5** denotes that a review for the caretaker has been submitted, written by the petowner, for the transaction after the completion of the transaction

```
CREATE TABLE Transactions_Details (
    caretaker_email VARCHAR,
    employment_type VARCHAR,
    pet_type VARCHAR,
    pet_name VARCHAR,
    owner_email VARCHAR CHECK (caretaker_email != owner_email),
    owner_review VARCHAR,
    owner_rating INTEGER,
    payment_mode VARCHAR NOT NULL,
    cost NUMERIC NOT NULL,
    mode_of_transfer VARCHAR NOT NULL,
    duration_from DATE NOT NULL, --Set by PetOwner
    duration_to DATE NOT NULL, --Set by PetOwner
    service_avail_from DATE NOT NULL,
    service_avail_to DATE NOT NULL,
    t_status INTEGER DEFAULT 1,
    PRIMARY KEY (caretaker_email, pet_name, owner_email, duration_to, duration_from),
    -- the start of the service must be same day or days later than the start of the availability period
    CHECK (duration_from >= service_avail_from),
    -- the end of the service must be same day or earlier than the end date of the availability period
    CHECK (duration_to <= service_avail_to),
    CHECK (caretaker_email != owner_email),
    FOREIGN KEY (owner_email, pet_name, pet_type) REFERENCES Owns_Pets(owner_email, pet_name, pet_type),
    FOREIGN KEY (caretaker_email, pet_type, service_avail_from, service_avail_to)
    REFERENCES Offers_Services(caretaker_email, type_pref, service_avail_from, service_avail_to)
);
```

Enquiries schema

```
CREATE TABLE Enquiries (
    user_email VARCHAR REFERENCES Users(email),
    enq_type VARCHAR,
    submission DATE,
    enq_message VARCHAR,
    answer VARCHAR,
    admin_email VARCHAR REFERENCES PCSAdmins(admin_email),
    PRIMARY KEY (user_email, enq_message)
);
```

Normalization level of database <a name = "normalization"></a>

<!-- 3NF or BCNF -->

All our tables are in BCNF format to eliminate data redundancies and anomalies.

✍ Three non-trivial triggers used in the application <a name = "triggers"></a>

Trigger to update the average rating of the caretaker and the number of reviews for the caretaker after every new review submission by a pet owner

This trigger is executed every time a new review is submitted by a pet owner for a transaction that is complete.

First, it will count the number of reviews that the caretaker has in the Transactions_details table and the **no_of_reviews** attribute of the caretaker in the Caretakers table will be updated accordingly.

Then, it will compute the average of all the ratings that the caretaker has received from the pet owners and the **avg_rating** attribute of the caretaker in the Caretakers table will be updated accordingly.

```
CREATE TRIGGER update_caretaker_rating
    AFTER UPDATE ON Transactions_Details
    FOR EACH ROW
    EXECUTE FUNCTION update_caretaker_rating();

CREATE OR REPLACE FUNCTION update_caretaker_rating()
RETURNS TRIGGER AS $
    DECLARE
        rating NUMERIC := 0;
        reviews_num INTEGER := 0;
    BEGIN
    SELECT COUNT(owner_rating) INTO reviews_num
    FROM Transactions_Details
    WHERE caretaker_email = NEW.caretaker_email;
    IF (reviews_num > 0) THEN
        SELECT AVG(owner_rating) INTO rating
        FROM Transactions_Details
        WHERE caretaker_email = NEW.caretaker_email;
    END IF;

    UPDATE Caretakers
    SET avg_rating = rating,
    no_of_reviews = reviews_num
    WHERE (caretaker_email = NEW.caretaker_email);

    RETURN NULL;
    END;
$ LANGUAGE plpgsql;
```

Trigger to check whether the caretaker has already reached the maximum amount of pets he can care for when he accepts a bid by a pet owner

This trigger is implemented to ensure that a caretaker does not exceed the maximum number of pets that he can take care for.

```
CREATE TRIGGER check_caretaker_limit
    BEFORE UPDATE ON Transactions_Details
    FOR EACH ROW
    EXECUTE PROCEDURE check_caretaker_limit();

CREATE OR REPLACE FUNCTION check_caretaker_limit()
RETURNS TRIGGER AS $
    DECLARE
        date_start DATE := NEW.duration_from;
        date_end DATE := NEW.duration_to;
        emp_type VARCHAR := NEW.employment_type;
        rating NUMERIC;
        pet_limit INTEGER := 2;
        count BIGINT := 0;
    BEGIN
        -- if the status change is to 2 (reject), 4(complete), 5(review submitted),
        -- the checks do not need to be performed.
        IF (NEW.t_status = 2 OR NEW.t_status = 4 OR NEW.t_status = 5) THEN
            RETURN NEW;
        END IF;

        -- the code below will execute only when the caretaker is about to accept a bid
        -- get rating of caretaker
        SELECT avg_rating INTO rating
        FROM Caretakers
        WHERE caretaker_email = NEW.caretaker_email;

        -- for a full time caretaker or a part time caretaker with a rating >= 4, the limit is 5
        IF ((emp_type = 'parttime' AND rating >= 4) OR emp_type = 'fulltime') THEN
            pet_limit := 5;
        END IF;

        -- Loop over the each date of the new bid to be accepted and check if any of the days have
        -- more than 5 transactions in progress
        WHILE date_start <= date_end LOOP
            -- select all the transactions that are also in the same availability period as the transaction
            -- to be accepted and check if they amount to 5
            SELECT COUNT(*) INTO count
            FROM Transactions_Details
            WHERE (caretaker_email = NEW.caretaker_email
            AND service_avail_from = NEW.service_avail_from
            AND service_avail_to = NEW.service_avail_to AND t_status = 3
            AND date_start >= duration_from AND date_start <= duration_to);

            IF (count >= pet_limit AND NEW.t_status = 3) THEN
                RAISE EXCEPTION 'You have already reached the limit for the number of pets you can take care of!';
                RETURN NULL;
            END IF;
            date_start := date_start + 1;
        END LOOP;

        RETURN NEW;
    END;
$ LANGUAGE plpgsql;
```

Trigger to update the daily price of a full time caretaker's services when his rating is updated after every new review submission

This trigger will execute to enable the feature that a 'full time caretaker's price for his services will increase as his rating increases'

```
CREATE TRIGGER update_fulltime_price
    AFTER UPDATE OF avg_rating ON Caretakers
    FOR EACH ROW
    EXECUTE PROCEDURE update_fulltime_price();

CREATE OR REPLACE FUNCTION update_fulltime_price()
RETURNS TRIGGER AS $
    DECLARE
        emp_type VARCHAR := NEW.employment_type;
        rating NUMERIC;
        reviews INTEGER;
        new_price INTEGER := 50;
    BEGIN
        -- get rating of caretaker
        SELECT avg_rating, no_of_reviews INTO rating, reviews
        FROM Caretakers
        WHERE caretaker_email = NEW.caretaker_email;
        IF (emp_type = 'fulltime' AND reviews >= 10) THEN
            IF (rating > 4.2 AND rating < 4.4 ) THEN
                new_price := 52;
            ELSIF (rating > 4.2 AND rating < 4.4 ) THEN
                new_price := 55;
            ELSIF (rating > 4.4 AND rating < 4.6 ) THEN
                new_price := 59;
            ELSIF (rating > 4.6 AND rating < 4.8 ) THEN
                new_price := 64;
            ELSIF (rating > 4.8 ) THEN
                new_price := 70;
            END IF;
        END IF;
        EXECUTE 'UPDATE Manages SET base_price = $1 WHERE caretaker_email = $2' USING new_price, NEW.caretaker_email;
        EXECUTE 'UPDATE Offers_Services SET daily_price = $1 WHERE caretaker_email = $2' USING new_price, NEW.caretaker_email;
        RETURN NEW;
    END;
$ LANGUAGE plpgsql;
```

🎉 Three most complex queries implemented in apllication <a name = "queries"></a>

Show code and write description
A cool SQL query would be to aggregate the number of pets in each category (dog, cat, lizard)
and then compare it to the the number of caretakers that can take care of the different types of pets then the business can see what kind of caretakers they should advertise to join their website
(For example, there are more lizards then lizard caretakers so more lizard caretakers should be recruited)

Advanced SQL Functions used

Function calculates the total commission earned bya pcsadmin

This function will returns the pcs admin email and the total commission earned. Detailed explanation of the sql query can be found in the comments within the code block.

```
-- function calculates total commission earned by admin inserted in $1
-- returns admin email and the sum of all the commissions earned
SELECT m1.admin_email, SUM(m2.commission)
  FROM pcsadmins AS m1, (SELECT m.admin_email AS admin_email, td.caretaker_email as caretaker_email,
                                -- calculates the commission earned from a single transaction
                                -- takes the date difference multipled by cost and 25%
                                td.cost*SUM(td.duration_to::date-td.duration_from::date+1)*0.25 AS commission
                           -- commission earned if the caretaker is being managed by the pcsadmin
                           FROM manages m JOIN transactions_details td ON m.caretaker_email=td.caretaker_email
                           -- commission earned if employment type is part-time
                           WHERE td.employment_type='parttime'
                           -- commission earned if the transaction is accepted, completed or reviewed
                           AND td.t_status>= 3
                        -- grouped to get the aggregate
                        GROUP BY m.admin_email, td.caretaker_email, td.cost, td.duration_from, td.duration_to) AS m2
 WHERE m1.admin_email=m2.admin_email AND m1.admin_email=$1
-- grouped to get the aggregate
GROUP BY m1.admin_email
```

Function checks if full time caretaker can leave

This function will return a table containing all new_service_avail_from1, new_service_avail_to1, new_service_avail_from2 new_service_avail_to2, leave_duration. Detailed explanation of the sql function can be found in the comments within the code block.

```
-- function to check if full time caretaker can take leave
DROP FUNCTION IF EXISTS check_for_leave(input_email VARCHAR, leave_start DATE, leave_end DATE);
CREATE OR REPLACE FUNCTION check_for_leave(input_email VARCHAR, leave_start DATE, leave_end DATE)
RETURNS TABLE (new_service_avail_from1 DATE,
                new_service_avail_to1 DATE,
                new_service_avail_from2 DATE,
                new_service_avail_to2 DATE,
                leave_duration INTEGER) AS $
    DECLARE
        old_service_avail_from DATE;
        old_service_avail_to DATE;
        previous_150_start DATE;
        previous_150_end DATE;
        new_service_avail_to_1 DATE;
        new_service_avail_from_2 DATE;
        leave_period INTEGER;
    BEGIN
        -- Check for valid input
        IF leave_end < leave_start THEN
            RAISE EXCEPTION 'You cannot take leave during this period!';
        END IF;

        -- First, get the service period of the caretaker that contains the leave period from the Offers_services table
        SELECT service_avail_from, service_avail_to INTO old_service_avail_from, old_service_avail_to
        FROM Offers_Services
        WHERE caretaker_email = input_email AND leave_start >= service_avail_from AND leave_end <= service_avail_to AND is_avail = 't';

        -- Then, check if there are any transactions accepted within the leave period, if yes return 0
        IF (SELECT COUNT(*) FROM Transactions_Details WHERE caretaker_email = input_email AND
            service_avail_from = old_service_avail_from AND service_avail_to = old_service_avail_to AND
            leave_start >= duration_from AND leave_end <= duration_to AND t_status = 3) > 0 THEN
            RAISE EXCEPTION 'You cannot take leave during this period!';
        END IF;

        -- proceed to check whether the caretaker has already had a 150 consecutive day period IN THE SAME YEAR
        -- if they do not have a 150 day period served,
        SELECT service_avail_from, service_avail_to INTO previous_150_start, previous_150_end
        FROM Offers_services WHERE caretaker_email = input_email AND (service_avail_to - service_avail_from >= 150)
        AND service_avail_to < old_service_avail_from;

        -- check if the previous 150 day shift was completed in the same year. If not, return false
        IF (SELECT extract(year from previous_150_end)) != (SELECT extract(year from old_service_avail_from)) THEN
            RAISE EXCEPTION 'You cannot take leave during this period!';
        END IF;

        leave_period := leave_end - leave_start + 1;
        new_service_avail_to_1 := leave_start - 1;
        new_service_avail_from_2 := leave_end + 1;

        -- case when the start of the leave == service_avail_from date (e.g 1/1/2020 start leave and 1/1/2020 start availability)
        IF (leave_start = old_service_avail_from) THEN
            new_service_avail_to_1 := old_service_avail_from;
        END IF;

        -- case when end of leave ==  service_avail_to date (e.g 31/10/2020 end leave and 31/10/2020 end availability)
        IF (leave_end = old_service_avail_to) THEN
            new_service_avail_from_2 := old_service_avail_to;
        END IF;

        -- check whether the previous 150 day shift has an overlap with the current one we are looking at
        IF (previous_150_start, previous_150_end) OVERLAPS (old_service_avail_from, old_service_avail_to)  THEN
            RAISE EXCEPTION 'You cannot take leave during this period!';
        END IF;

        -- check if the curr period has at least 300 days since we need to split up into 2 consecutive 150 days
        IF (old_service_avail_to - old_service_avail_from - (leave_end - leave_start) > 300) THEN
            -- if can split up, return true
            IF (leave_start - old_service_avail_from > 150 AND old_service_avail_to - leave_end > 150) THEN
                -- this is when the date that the caretaker wants to take leave on is on the same day the availability starts when he takes a one day leave
                -- so need to add 1 day to the date (e.g availability starts on 1/1/2020 so the new availability should start on 2/1/2020)
                IF (old_service_avail_from = leave_start AND leave_period = 1) THEN
                    old_service_avail_from := old_service_avail_from + 1;
                    new_service_avail_to_1 := new_service_avail_to_1 + 1;
                END IF;

                IF (old_service_avail_to = leave_end AND leave_period = 1) THEN
                    old_service_avail_to := old_service_avail_to + 1;
                    new_service_avail_from_2 := new_service_avail_from_2 + 1;
                END IF;

                RETURN QUERY SELECT old_service_avail_from::DATE AS new_service_avail_from1, new_service_avail_to_1::DATE AS new_service_avail_to1,
                new_service_avail_from_2::DATE AS new_service_avail_from2, old_service_avail_to::DATE AS new_service_avail_to2, leave_period AS leave_duration;


            ELSIF (leave_start - old_service_avail_from > 300 OR old_service_avail_to - leave_end > 300) THEN
                -- this is when the date that the caretaker wants to take leave on is on the same day the availability starts when he takes a one day leave
                -- so need to add 1 day to the date (e.g availability starts on 1/1/2020 so the new availability should start on 2/1/2020)
                IF (old_service_avail_from = leave_start AND leave_period = 1) THEN
                    old_service_avail_from := old_service_avail_from + 1;
                    new_service_avail_to_1 := new_service_avail_to_1 + 1;
                END IF;

                IF (old_service_avail_to = leave_end AND leave_period = 1) THEN
                    old_service_avail_to := old_service_avail_to + 1;
                    new_service_avail_from_2 := new_service_avail_from_2 + 1;
                END IF;

                RETURN QUERY SELECT old_service_avail_from::DATE AS new_service_avail_from1, new_service_avail_to_1::DATE AS new_service_avail_to1,
                new_service_avail_from_2::DATE AS new_service_avail_from2, old_service_avail_to::DATE AS new_service_avail_to2, leave_period AS leave_duration;

            ELSE
                RAISE EXCEPTION 'You cannot take leave during this period!';
            END IF;

        -- this means that there was already a 150 day consecutive period worked in the past
        ELSE
            IF (old_service_avail_to - old_service_avail_from - (leave_end - leave_start) > 150) THEN
                -- if can split up, return true
                IF (leave_start - old_service_avail_from > 150 OR old_service_avail_to - leave_end > 150) THEN
                    -- this is when the date that the caretaker wants to take leave on is on the same day the availability starts when he takes a one day leave
                    -- so need to add 1 day to the date (e.g availability starts on 1/1/2020 so the new availability should start on 2/1/2020)
                    IF (old_service_avail_from = leave_start AND leave_period = 1) THEN
                        old_service_avail_from := old_service_avail_from + 1;
                        new_service_avail_to_1 := new_service_avail_to_1 + 1;
                    END IF;

                    IF (old_service_avail_to = leave_end AND leave_period = 1) THEN
                        old_service_avail_to := old_service_avail_to + 1;
                        new_service_avail_from_2 := new_service_avail_from_2 + 1;
                    END IF;

                    RETURN QUERY SELECT old_service_avail_from::DATE AS new_service_avail_from1, new_service_avail_to_1::DATE AS new_service_avail_to1,
                    new_service_avail_from_2::DATE AS new_service_avail_from2, old_service_avail_to::DATE AS new_service_avail_to2, leave_period AS leave_duration;
                ELSE
                    RAISE EXCEPTION 'You cannot take leave during this period!';
                END IF;
            ELSE-- if cannot split up to 150 days, return false
                RAISE EXCEPTION 'You cannot take leave during this period!';
            END IF;
        END IF;
    END;
$ LANGUAGE plpgsql;

-- function to get underperforming caretakers (rating less than 2)
DROP FUNCTION IF EXISTS get_underperforming_caretakers();
DROP TYPE IF EXISTS return_type;
CREATE TYPE return_type AS
            ( caretaker VARCHAR, num_pet_days NUMERIC, avg_rating NUMERIC, num_rating_5 NUMERIC, num_rating_4 NUMERIC, num_rating_3 NUMERIC, num_rating_2 NUMERIC, num_rating_1 NUMERIC, num_rating_0 NUMERIC );
CREATE OR REPLACE FUNCTION get_underperforming_caretakers()
RETURNS SETOF return_type AS $
    DECLARE
        caretakers_arr VARCHAR [] := '{}';
        caretaker VARCHAR;
        avg_rating_arr NUMERIC [] := '{}';
        transactions_duration_to DATE [] := '{}';
        transactions_duration_from DATE [] := '{}';
        duration NUMERIC;
        num_pet_days NUMERIC;
        num_ratings NUMERIC;
        val return_type;
        rec RECORD;
    BEGIN
        caretakers_arr := ARRAY (SELECT caretaker_email
        FROM caretakers
        WHERE employment_type = 'fulltime'
        AND avg_rating <= 2
        ORDER BY avg_rating ASC);

        avg_rating_arr := ARRAY (SELECT avg_rating
        FROM caretakers
        WHERE employment_type = 'fulltime'
        AND avg_rating <= 2
        ORDER BY avg_rating ASC);

        FOR index IN array_lower(caretakers_arr, 1) .. array_upper(caretakers_arr, 1) LOOP

            transactions_duration_from := ARRAY (SELECT duration_from
                                                    FROM transactions_details
                                                    WHERE caretaker_email = caretakers_arr[index]);
            transactions_duration_to := ARRAY (SELECT duration_to
                                                    FROM transactions_details
                                                    WHERE caretaker_email = caretakers_arr[index]);
            IF array_length(transactions_duration_from, 1) IS NULL OR array_length(transactions_duration_from, 1) = 0 THEN
                CONTINUE;
            END IF;
            num_ratings := (SELECT COUNT(*)
                            FROM transactions_details
                            WHERE caretaker_email = caretakers_arr[index]
                            AND owner_rating IS NOT NULL);
            IF num_ratings = 0 THEN
                CONTINUE;
            END IF;
            num_pet_days := 0;
            FOR i IN array_lower(transactions_duration_from, 1) .. array_upper(transactions_duration_from, 1) LOOP
                duration := transactions_duration_to[i] - transactions_duration_from[i] + 1;
                num_pet_days := num_pet_days + duration;
            END LOOP;
            val.caretaker := caretakers_arr[index];
            val.num_pet_days := num_pet_days;
            val.avg_rating := avg_rating_arr[index];
            val.num_rating_5 := (SELECT COUNT(*)
                                FROM transactions_details
                                WHERE caretaker_email = caretakers_arr[index]
                                AND owner_rating = 5);
            val.num_rating_4 := (SELECT COUNT(*)
                                FROM transactions_details
                                WHERE caretaker_email = caretakers_arr[index]
                                AND owner_rating = 4);
            val.num_rating_3 := (SELECT COUNT(*)
                                FROM transactions_details
                                WHERE caretaker_email = caretakers_arr[index]
                                AND owner_rating = 3);
            val.num_rating_2 := (SELECT COUNT(*)
                                FROM transactions_details
                                WHERE caretaker_email = caretakers_arr[index]
                                AND owner_rating = 2);
            val.num_rating_1 := (SELECT COUNT(*)
                                FROM transactions_details
                                WHERE caretaker_email = caretakers_arr[index]
                                AND owner_rating = 1);
            val.num_rating_0 := (SELECT COUNT(*)
                                FROM transactions_details
                                WHERE caretaker_email = caretakers_arr[index]
                                AND owner_rating = 0);
            RETURN NEXT val;
        END LOOP;

        RETURN;

    END;
$ LANGUAGE plpgsql;
```

🔨 Tools and Frameworks used <a name = "tools_used"></a>

We used the PERN stack to develop our application:

- PostgreSQL (https://www.postgresql.org/) - Database
- Express (https://expressjs.com/) - Server framework
- ReactJS (https://reactjs.org/) - Frontend framework
- NodeJS (https://nodejs.org/en/) - Server runtime environment

Deployment was done using Heroku:

- Heroku (https://heroku.com) - Deployment platform

Frontend design was done using Bootstrap:

- Bootstrap (https://getbootstrap.com/docs/4.5/getting-started/introduction/) - Frontend Framework Design

🔍 Screenshots of application <a name = "screenshots"></a>

<table>
<tr>
<th>Admins' dashboard</th>

```
<th>Admins' dashboard (continued)</th>
</tr>
<tr>
<td> <img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/admin_dashboard.jpg?raw=true" alt="Admin dashboard screenshot" ></td>
<td><img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/caretaker_earnings.jpg?raw=true" alt="Admin earnings screenshot" ></td>
</tr>
<tr>
<th>Caretaker profile page</th>
<th>Caretaker homepage</th>
</tr>
<tr>
<td> <img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/caretaker_profile.jpg?raw=true" alt="Admin dashboard screenshot" ></td>
<td><img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/caretaker_homepage.jpg?raw=true" alt="Admin earnings screenshot" ></td>
</tr>
<tr>
<th>Caretakers' earnings page</th>
<th>Browsing for caretakers' services page</th>
</tr>
<tr>
<td> <img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/caretaker_salary.jpg?raw=true" alt="Admin dashboard screenshot" ></td>
<td><img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/caretakers_browse.jpg?raw=true" alt="Admin earnings screenshot" ></td>
</tr>
<tr>
<th>Looking at a caretaker's reviews</th>
<th>Bidding for a caretaker's service</th>
</tr>
<tr>
<td> <img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/browse_reviews.jpg?raw=true" alt="Admin dashboard screenshot" ></td>
<td><img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/requesting_service.jpg?raw=true" alt="Admin earnings screenshot" ></td>
</tr>
<tr>
<th>Pet owner profile page</th>
<th>Pet owner homepage</th>
</tr>
<tr>
<td> <img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/owner_profile.jpg?raw=true" alt="Admin dashboard screenshot" ></td>
<td><img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/owner_home.jpg?raw=true" alt="Admin earnings screenshot" ></td>
</tr>
<tr>
<th>Pet registration page</th>
<th>Pet owners' pets page</th>
</tr>
<tr>
<td> <img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/pet_register.jpg?raw=true" alt="Admin dashboard screenshot" ></td>
<td><img src="https://github.com/sevenmatt7/CS2102_2021_S1_Team1/blob/matt/proj_docs/screenshots/owner_pets.jpg?raw=true" alt="Admin earnings screenshot" ></td>
</tr>
</table>
```

🏴 Summary of difficulties encountered and lessons learned from project <a name = "conclusion"></a>

- Leveraging the power of DBMS to make the application fast and efficient was a problem faced in the early stages of our project. After we realised the importance of normalisation as we saw certain redundancies that appeared in our database, we were able to remove some redundant tables. This showed us the importance of a proper ER diagram.
- With careful planning at the start, we could have avoided some issues due to data parsing. In our initial implementation, we stored durations in our database as a string, with start and end dates concatenation together and delimited by a comma. We faced difficulties parsing the string into dates for calculations, and had to re-implement our schema halfway through the project to store dates as DATE format in the database instead.
- Creating complex queries was challenging with the knowledge and SQL constructs exposed to us from the module and we had to search online for methods to achieve what we want. We were able to find better ways to make our existing queries better and more efficient, such as the usage of LOOP in a PLPGSQL triggers/functions. We were able to make efficient queries with the usage of SQL transactions as well.
- Lack of experience in the field of web developement technologies, such as React and NodeJS posed an initial steep learning curve for some of the group members to keep up with the overall development pace of the team.
- Splitting the workload was a problem faced throughout the project, as progress can be slow without consistent meetings and sprints. Developement could be more efficient with better allocation of task, as well as a issue/task tracker.