CS2102 Project Report - Team 66

Members

Li Jiayu A0194567M
Liu Sitong A0194515B
Shaun Ware Yeo Zishen A0189658A
Zhou Yurou A0194544X
Chan Hong Yi, Matthew A0183924X

Responsibilities

Name	ER diagram	Constraint specification	Main features	Project report	Others
Li Jiayu	Design and draw.	Consolidate and organize.	Overall design of project. New pet. New request. Handle transactions by pet owners. View request. Open access care takers view. Care Taker profile for others to view.	`SELECT` queries 1 and 2. `TRIGGER` 1 and 3. Most `CREATE` queries. UI 2. Overall organization.	Setup Git and GitHub. Setup team Heroku database.
Liu Sitong	Check and modify.	Draft all the constraints.	Salary, admin related. View salary, edit salary, view summary info.	Application Description, `TRIGGER 2,` salary related queries. Update the data constraints.	Recruiting members. Suggest and arrange meetings.
Shaun Ware	Check and modify.	Check and modify.	Pet, pets, pet profile.	Normal forms. `SELECT` query 3. Updated tools and framework. Corrected grammar and language structure.	Updated the heroku database when others couldn't access it
Zhou Yurou	Check and modify.	Check and modify.	Index, login, register, dashboards, edit_profile pages, and select takers.	Some Data design and constraints. UI 1.	Host some meetings. Front-end design.
Chan Hong Yi, Matthew	Discuss design and modify.	Check and modify.	View requests by pet owner, edit request, delete request, view all services by care taker, accept/decline transactions.	Tools/Frameworks used.	

Application description

The Pet Caring Service application (PCS) provides services to three kinds of App Users: Pet Owners, Care Takers, and PCS Administrators. Users must register and log in prior to using any services. Users are allowed to register as both Care Takers and Pet Owners. They will be able to access the different user services after selecting Pet Owner or Care Taker after logging in. The Pet Owner is able to register new pets, update their pets' profiles and delete pets. They are also able to browse and bid for the pet care services offered by the Care Takers. Special requirements on transfer types can be addressed and recorded using the PCS app as well. As Care Takers, the user must adhere to the working rules but can opt to be full-time or part-time while taking care of pets. Care Takers are offered functions like viewing salary and accepting pet care requests, among others. Finally PCS Administrators are granted the rights to view, edit or update the administrative details of other users. This includes a summary page of Care Taker salaries and several filter options to determine the performance of the Care Takers.

Tools/Frameworks used

In our project, we used PostgreSQL as our database management system. To implement the webpage, we used NodeJS as our runtime environment. Our application is hosted on the Heroku cloud platform.

The PCS we have created is based on the Express framework for Node JS, using the Express Application Generator to create the overall project skeleton. This allowed the project to be configured to be used with commonly implemented CSS stylesheet engines.

We have used dotenv to avoid leaking database credentials in our source code. This also allows us to collaborate and connect to our database from the string stored in our .env file.

Finally, we used Passport.js to support the authentication and log in/out processes for the application.

Data design and constraints

Features	Solution
Every user of the app is an App User, with a user ID, an email and a password that enables logging in. The user ID is unique (not serial), which users can specify upon registration by themselves. They also need to provide a unique not null email for login purposes. The password should be at least 6 characters.	RS. RS: `PRIMARY KEY`, `UNIQUE`, `CHECK`, `NOT NULL`.
The system would check the uniqueness of the user ID, and if the user ID has already been chosen by other users, this user ID would be invalid for registration.	NJ: E. SQ: If the username exists in the Appusers table, system prompts to change another username.
An App User logs in by providing username and password.	SQ: Only when the password matches with the password in

	the AppUsers table can the user log into the system.
An App User can be a PSC Administrator or other Users.	RS. RS: `FOREIGN KEY`.
A User must be a Pet Owner, a Care Taker or both.	RS. RS: `FOREIGN KEY`.
A User has name, gender, and address as additional attributes. The name must be specified, but the User is free to choose whatever name they want. Gender and address are suggested to be filled but not required.	RS. RS: `NOT NULL`.
A Care Taker can be either full-time or part-time, but not both.	RS. RS: `FOREIGN KEY`. The constraints on ISA relationship can be fulfilled using trigger. However, there are no means for app users to make a Care Taker that cannot fulfil this constraint, hence this trigger is not required.
Every Care Taker should specify the categories of pets that they can take care of (at least one category). Every Care Taker may also specify the categories that they definitely cannot take care.	RS. RS: `FOREIGN KEY`. NJ: E. When registering for Care Takers, the system forces the user to indicate at least one category.
A Care Taker has a daily price (> 0) for each Pet Category that he or she has declared as Can Take Care. The default value is from the standard daily price for the Pet Category. Part-time Care Takers can specify the daily price manually.	RS. RS: `CHECK`, `DEFAULT`.
Every Care Taker should specify the periods they are available for 2 years. For a Care Taker, the period that he or she is available should not overlap, and consecutive periods should be unified into one. Part-time Care Takers can specify their Periods Available by themselves, whereas Periods Available of Full-time Care Takers are dependent on their Leaves.	RS. TG 1. TG: to make the Periods Available of Full-time Care Takers dependent on Leaves.
A Full-time Care Taker can take Leave only when he or she is not taking care of any Pets during the period of Leave, and he or she can fulfill the working commitment requirement of a minimum 2 * 150 consecutive days a year.	NJ: E. SQ. If the Care Taker is taking care of some pets (status is ‘Confirmed’ in Transactions table), or will not fulfill the requirements of consecutive days in

	the year after the Leave, the Care Taker will not be allowed to apply for Leaves.
Each Leave of each Full-time Care Taker has a start and end date. A Full-time Care Taker's Periods Available should cover all days that the Care Taker is not on Leave.	The Periods Available of Full-time Caretakers are initialized from today to 5 years later (for which we assume that a major maintenance is needed for the system for 5 years so that the data can be updated). TG: Update the Periods Available after approving a Leave.
Each Care Taker has an average rating out of 5, which is calculated using all of the ratings of confirmed Transactions of the Care Taker.	TG.
**Instead of using atomic days to manage days, we are using Periods to represent consecutive days. Each Period has a start date and end date, and Periods with the same start date and end date represent the same period. The end date should never precede the start date.	There is no concrete table in the database of Periods, but Period information is integrated into all the tables that reference Period.
A Care Taker will not receive any order of Pet Categories that they have indicated as definitely Cannot Take Care of, but he or she may receive Requests that he or she has not yet indicated as Can Take Care. They will automatically be added to the Can Take Care table upon acceptance of the request.	ND. SQ. Pet Owners can only see (and hence send) Requests of a pet to those Care Takers that may be able to take care of this Pet Category.
Each Pet Category of pet is uniquely identified by its category name (dog, cat, etc). They have a standard daily price attached to them that is specified by a PSC Administrator.	RS. RS: `PRIMARY KEY`.
The name of a Pet Category and the name should be trimmed with only lowercase letters.	ND. NJ: E.
Each Pet Owner can own multiple Pets, but each Pet must have only one owner.	RS: `FOREIGN KEY`.
Every Pet has a unique ID, a name and special requirements. ID and name are necessary. (It is possible for a person to have pets of the same name, so we use Pet IDs to uniquely identify a Pet.)	RS, RS: `PRIMARY KEY`, `NOT NULL`.
Each Pet is classified under exactly one Pet Category.	RS, RS: `FOREIGN KEY`, `NOT NULL`.

**When creating a new Pet, Pet Owners can choose a category from the Pet Category list, but if the pet is not among the existing Pet Categories, the Pet Owner may create a new Pet Category. Categories of the same name but different capitalization are regarded as the same category. A Category name only accepts alphabetical characters or white spaces.	ND. NJ: E.
A Pet Owner can publish Requests to find Care Takers for a specified Period of time of a particular Pet with payment method (card or cash) and transfer type (deliver, pick-up, or transfer at the PSC building) declared.	RS, RS: `FOREIGN KEY`, `NOT NULL`, `CHECK`.
**There should not be conflicting Requests on the same pet (i.e. the Periods overlap). A Request cannot be made if there are existing Transactions of the same Pet during the same time Period overlap. (This allows the Pet ID and start date to uniquely identify any Request.)	SQ. NJ: E. RS: `PRIMARY KEY`.
Pet Owners can send Requests to Care Takers to form Transactions. Each Transaction has a status, which can be pending, confirmed, withdrawn, declined (by Care Taker) and outdated.	RS. RS: `FOREIGN KEY`, `NOT NULL`, `CHECK`.
There should be at least one Transaction formed for each Request.	SQ. The Request will be deleted from the database if the user leaves the page of handle_transaction without creating a corresponding confirmed or pending Transaction.
**There should be at most one confirmed Transaction created for each Request. Once a Transaction is confirmed, the status of other Pending transactions will be changed to outdated (this outdated status can never appear unless there is a confirmed transaction of this request). This action cannot be reverted, once a transaction is confirmed, it cannot be withdrawn or cancelled.	TG 3.
Start and end dates for any Request should be a date within 2 years and not earlier than the date of creation.	NJ. NJ: E.
The end date should be chronologically later than the start date.	NJ: E. RS: `CHECK`.
After the Transaction is completed (the Period of pet-caring is over), the Pet Owner can write a review for this Transaction and give a rating from 1 to 5.	RS. RS: `CHECK`. NJ: E. Can only rate after the service ends.
Only confirmed Transactions can have a review and rating.	RS: `CHECK`. No one can edit the review and rating of withdrawn, pending or outdated Transactions in the app.
A Care Taker may take care of more than one pet at any given time. All Care Takers can only care for up to 5 Pets at the same time (on the	SQ. Pet Owners can only see (and hence

same day). Part-time Care Takers cannot take care of more than 2 Pets unless they have an average rating of more than 4 out of 5, in which case then can take care of up to 4 Pets. We have decided to limit Part-Time Care Takers to a maximum of 4 Pets in order to ensure that the standards of pet care are maintained.	send) Requests of a Pet to those Care Takers that can take care of the pet during the given Period. TG 3. System will automatically decline pending Transactions if the maximum is met.
A Transaction is confirmed with a specific Care Taker to take care of the pet during that Period of time. Each Transaction has a cost calculated from the product of duration and daily price as specified for the particular Care Taker and Category. If there is no declaration, and the standard price is not yet specified, a baseline price of 200 will be used instead.	NJ. TG.

* Notations used in the table:
** = Non-trivial and interesting features
ND = Not enforced in database.
RS = Relational schema.
RS: `XX` = Relational schema `XX` constraints.
NJ = Checked in NodeJS (Completely dependent on Frontend).
NJ: E = Checked in NodeJS because the error message is expected to be shown with details.
SQ = Checked in NodeJS using `SELECT` query.
TG = Enforced using trigger in PostgreSQL. (TG # means that this trigger can be seen in the examples of triggers number #.)

Relational Schema

```
CREATE TABLE AppUsers (
   userid VARCHAR PRIMARY KEY,
   password VARCHAR CHECK ( length(password) >= 6 ),
   email VARCHAR UNIQUE NOT NULL
);

CREATE TABLE PSCAdministrators (
   userid VARCHAR PRIMARY KEY REFERENCES AppUsers(userid)
);

CREATE TABLE Users (
   userid VARCHAR PRIMARY KEY REFERENCES AppUsers(userid),
   name VARCHAR NOT NULL,
   gender VARCHAR CHECK( gender = 'MALE' OR gender = 'FEMALE' ),
   address VARCHAR
)

CREATE TABLE PetOwners (
   userid VARCHAR PRIMARY KEY REFERENCES Users(userid)
);
```

```
CREATE TABLE PetCategories (
   name VARCHAR PRIMARY KEY,
   standard_price NUMERIC
);

CREATE TABLE Pets (
   petid VARCHAR PRIMARY KEY,
   name VARCHAR NOT NULL,
   category VARCHAR NOT NULL REFERENCES PetCategories(name),
   owner VARCHAR NOT NULL REFERENCES PetOwners(userid) ON DELETE CASCADE,
   requirements TEXT
);

CREATE TABLE CareTakers (
   userid VARCHAR PRIMARY KEY REFERENCES Users(userid),
   rating NUMERIC CHECK( rating <= 5 AND rating >= 1 )
);

CREATE TABLE CanTakeCare (
   ct_id VARCHAR REFERENCES CareTakers(userid),
   category VARCHAR REFERENCES PetCategories(name),
   daily_price NUMERIC NOT NULL CHECK ( daily_price > 0 ),
   PRIMARY KEY (ct_id, category)
);

CREATE TABLE CannotTakeCare (
   ct_id VARCHAR REFERENCES CareTakers(userid),
   category VARCHAR REFERENCES PetCategories(name),
   PRIMARY KEY (ct_id, category)
);

CREATE TABLE PeriodsAvailable (
   ct_id VARCHAR REFERENCES CareTakers(userid),
   s_date DATE,
   e_date DATE,
   PRIMARY KEY (ct_id, s_date),
   UNIQUE (ct_id, e_date)
);

CREATE TABLE FullTimeCareTakers (
   userid VARCHAR PRIMARY KEY REFERENCES CareTakers(userid)
);

CREATE TABLE PartTimeCareTakers (
   userid VARCHAR PRIMARY KEY REFERENCES CareTakers(userid)
);

CREATE TABLE Leaves (
   ct_id VARCHAR REFERENCES FullTimeCareTakers(userid),
   s_date DATE,
   e_date DATE,
   PRIMARY KEY (ct_id, s_date),
   UNIQUE (ct_id, e_date)
);

CREATE TABLE Requests (
   pet_id VARCHAR REFERENCES Pets(petid),
   s_date DATE,
   e_date DATE NOT NULL,
   transfer_type VARCHAR NOT NULL CHECK ( transfer_type = 'deliver' OR
transfer_type = 'pick-up' OR transfer_type = 'transfer at building' ) DEFAULT
'deliver',
   payment_type VARCHAR NOT NULL CHECK ( payment_type = 'card' OR payment_type =
'cash' ) DEFAULT 'card',
   PRIMARY KEY ( pet_id, s_date ),
   UNIQUE ( pet_id, e_date ),
   CHECK ( e_date >= s_date )
);

CREATE TABLE Transactions (
   pet_id VARCHAR,
   s_date DATE,
   ct_id VARCHAR REFERENCES CareTakers(userid),
   cost NUMERIC,
   rate NUMERIC CHECK ( rate = 1 OR rate = 2 OR rate = 3 OR rate = 4 OR rate = 5
),
   review TEXT,
   status VARCHAR NOT NULL CHECK ( status = 'Pending' OR status = 'Confirmed' OR
status = 'Withdrawn' OR status = 'Declined' OR status = 'Outdated' ) DEFAULT
'Pending',
   FOREIGN KEY (pet_id, s_date) REFERENCES Requests(pet_id, s_date) ON DELETE
CASCADE,
   PRIMARY KEY (pet_id, s_date, ct_id),
   CHECK ( (rate IS NULL AND review IS NULL) OR status='Confirmed' )
);

CREATE TABLE Salary (
   userid VARCHAR NOT NULL REFERENCES CareTakers(userid),
   amount NUMERIC CHECK (amount >= 0),
   month INTEGER CHECK( month >= 1 AND month <= 12 ),
   year INTEGER CHECK( year >= 2000 AND year <= 2200 ),
   PRIMARY KEY(userid, month, year)
);
```

Normal forms

The database we have designed is in BCNF. The breakdown and justification is as follows:

The table for CannotTakeCare had only trivial functional dependencies, and are therefore BCNF.

The Pets table has a single functional dependency of {petid -> name, category, owner, requirements}. (petid) is a superkey, hence it is in BCNF.

The Appusers table has functional dependencies of {userid -> password, email; email -> userid, password}. Both (userid) and (email) are superkeys, hence the table is in BCNF.

The Salary table has a single functional dependency of {userid, month, year -> amount}. (userid, month, year) is a superkey of the relationship, hence it is in BCNF.

The Leaves and PeriodsAvailable tables both have functional dependencies of {ct_id, s_date -> e_date; ct_id, e_date -> s_date}. (ct_id, s_date) and (ct_id, e_date) are both superkeys for each of the relationships, hence both tables are also in BCNF.

The CanTakeCare table has a single functional dependency of {ct_id, category -> price}. (ct_id, category) is a superkey of the relationship, hence it is also in BCNF.

The Requests table has functional dependencies of {pet_id, s_date -> e_date, transfer_type, payment_type; pet_id, e_date -> s_date, transfer_type, payment_type}. In this relationship, both (pet_id, s_date) and (pet_id,e_date) are superkeys, and hence the table is also in BCNF.

Finally, the Transactions table has a functional dependency of {petid, s_date, ct_id -> cost, rate, review, status}. (pet_id,s_date, ct_id) is a superkey of the relationship, hence this table is also in BCNF.

Examples of triggers

1. Ensuring non-overlapping periods available

```
CREATE OR REPLACE FUNCTION combine_periods() RETURNS TRIGGER AS
$$
   DECLARE
       ct VARCHAR := NEW.ct_id;
       sd DATE := NEW.s_date;
       ed DATE := NEW.e_date;
       sd2 DATE;
       ed2 DATE;
       ctx INTEGER;
   BEGIN
       SELECT PA.s_date, PA.e_date, COUNT(*) INTO sd2, ed2, ctx
       FROM PeriodsAvailable PA
       WHERE PA.ct_id=ct AND sd<>PA.s_date AND ed<>PA.e_date
       AND ((ed<=PA.e_date AND ed>=PA.s_date)
           OR (PA.e_date<=ed AND PA.e_date>=sd))
       GROUP BY (PA.ct_id, PA.s_date, PA.e_date) LIMIT 1;
       IF ctx > 0 THEN
           DELETE FROM PeriodsAvailable
           WHERE ct_id=ct AND s_date=sd2 AND e_date=ed2;
           DELETE FROM PeriodsAvailable
           WHERE ct_id=ct AND s_date=sd AND e_date=ed;
           IF sd > sd2 THEN sd := sd2; END IF;
           IF ed < ed2 THEN ed := ed2; END IF;
           INSERT INTO PeriodsAvailable VALUES (ct, sd, ed);
       END IF;
       SELECT PA.s_date, PA.e_date, COUNT(*) INTO sd2, ed2, ctx
       FROM PeriodsAvailable PA
       WHERE PA.ct_id=ct AND sd-1=PA.e_date
       GROUP BY (PA.ct_id, PA.s_date, PA.e_date) LIMIT 1;
       IF ctx > 0 THEN
           DELETE FROM PeriodsAvailable
           WHERE ct_id=ct AND s_date=sd2 and e_date=ed2;
           DELETE FROM PeriodsAvailable
           WHERE ct_id=ct AND s_date=sd and e_date=ed;
```

```
            INSERT INTO PeriodsAvailable VALUES (ct, sd2, ed);
        END IF;
        SELECT PA.s_date, PA.e_date, COUNT(*) INTO sd2, ed2, ctx
        FROM PeriodsAvailable PA
        WHERE PA.ct_id=ct AND ed+1=PA.s_date
        GROUP BY (PA.ct_id, PA.s_date, PA.e_date) LIMIT 1;
        IF ctx > 0 THEN
            DELETE FROM PeriodsAvailable
            WHERE ct_id=ct AND s_date=sd2 and e_date=ed2;
            DELETE FROM PeriodsAvailable
            WHERE ct_id=ct AND s_date=sd and e_date=ed;
            INSERT INTO PeriodsAvailable VALUES (ct, sd, ed2);
        END IF;
        RETURN NEW;
    END;
$$
LANGUAGE plpgsql;
CREATE TRIGGER combine_periods
    AFTER INSERT OR UPDATE ON PeriodsAvailable
    FOR EACH ROW EXECUTE PROCEDURE combine_periods();
```

Explanation:
The Periods Available of a Care Taker should be non-overlapping. Hence, when inserting a new Period Available, if it is overlapping with existing Periods of this Care Taker, the system would combine the two Periods into one. Moreover, if two Periods are consecutive, the Periods will be combined together. The combination is done by deleting the two conflicting Periods and inserting a combined Period in their place. The trigger will call recursively on itself until there are no more overlaps. The recursion terminates when either there is no overlap at all or the newly inserted rows already exist in the table, after which the trigger will stop changing the data in the table.

2. Salary calculating

```
CREATE OR REPLACE FUNCTION salary_calculation() RETURNS TRIGGER AS
    $$
    DECLARE
        cid VARCHAR := NEW.ct_id;
        pet_day INTEGER;
        day INTEGER;
        allcost NUMERIC := 0;
        cost NUMERIC := 0;
        date DATE := NEW.s_date;
        res NUMERIC;
        tr RECORD;
        m INTEGER := EXTRACT(MONTH FROM date);
        y INTEGER := EXTRACT(YEAR FROM date);
    BEGIN
        pet_day := calculate_petday(cid,m,y);
        FOR tr IN
            SELECT daily_price,
            days(s_date,e_date,CAST(y AS VARCHAR),CAST(m AS VARCHAR)) AS days
            FROM ((Requests R NATURAL JOIN Transactions T) INNER JOIN Pets P ON
                (T.pet_id = P.petid)) NATURAL JOIN CanTakeCare C
            WHERE T.status = 'Confirmed' AND T.ct_id = cid
            LOOP
            allcost = allcost + tr.daily_price * tr.days;
            END LOOP;
      IF cid IN(SELECT userid FROM FullTimeCareTakers) THEN
         res := 3000;
         IF pet_day > 60 THEN
            FOR tr IN
            SELECT daily_price,
            days(s_date,e_date,CAST(y AS VARCHAR),CAST(m AS VARCHAR)) AS days
            FROM ((Requests R NATURAL JOIN Transactions T) INNER JOIN Pets P
            ON (T.pet_id = P.petid)) NATURAL JOIN CanTakeCare C
            WHERE T.status = 'Confirmed' AND T.ct_id = cid
            ORDER BY s_date ASC;
            LOOP
            IF(day + tr.days > 60) THEN
                 cost = cost + tr.daily_price * (60-day);
                 EXIT;
            ELSE cost = cost + tr.daily_price * tr.days;
                 day = day + tr.days;
            END IF;
            END LOOP;
            res := res + 0.8 * (allcost-cost);
         END IF;
      ELSE
         res := 0.75 * allcost;
      END IF;
         INSERT INTO Salary (userid,amount,month,year)
         VALUES(cid,res,m,y)
         ON CONFLICT(userid,month,year)
         DO UPDATE SET amount = res;
     RETURN NEW;
     END;
     $$
LANGUAGE  plpgsql;
CREATE TRIGGER calculate_salary
AFTER INSERT OR UPDATE ON Transactions FOR EACH ROW EXECUTE PROCEDURE
salary_calculation();
```

Explanation:
The salary is calculated (and inserted or updated in Salary table) for a Care Taker whenever the table Transactions has been modified (insert or update), which indicates a pet-caring job has been taken. Different calculation policies are adopted, depending on if it is a Part-time or Full-time Care Taker. The Full-time Care Taker should have a salary that includes a $3000 base salary and the 80% of the excess work more than 60 days as bonus. The excess earning is calculated by the total cost from the transaction (calculated in the first loop) minus the cost earned within the first 60 pet-days (calculated in the second loop). The first 60 pet-days are determined by the confirmed Transaction with the earliest start date. The part-time Care Taker’s salary is calculated as 75% of the total cost for the month.
Two helper functions are used:

1. `FUNCTION calculate_petday(cid VARCHAR,year INTEGER,month INTEGER) RETURNS INTEGER` is used to calculate the total pet-days by summing up the working days indicated by the start date and end date in the Requests table and the corresponding “Confirmed” status of transactions done by this Care Taker. The working days within the year and month given is calculated using the `FUNCTION days(s1 DATE, e1 DATE, year VARCHAR, month VARCHAR) RETURNS INTEGER` mentioned below.

2. `FUNCTION` *`days`*`(s1 DATE, e1 DATE, year VARCHAR, month VARCHAR) RETURNS INTEGER` checks how many of the working days in the transaction by comparing the start date and end date with the date of the first and last day of the given month and year.

3. Confirm a Transaction

```
CREATE OR REPLACE FUNCTION confirm_transaction() RETURNS TRIGGER AS
$$
DECLARE
    pet VARCHAR := NEW.pet_id;
    sd DATE := NEW.s_date;
    ct VARCHAR := NEW.ct_id;
    st VARCHAR := NEW.status;
    pct VARCHAR;
    price NUMERIC;
    days INTEGER;
BEGIN
    IF st='Confirmed' THEN
        SELECT category INTO pct FROM Pets WHERE petid=pet;
        UPDATE Transactions SET status='Outdated'
        WHERE (status='Pending' OR status='Confirmed')
        AND pet_id=pet AND s_date=sd AND ct_id<>ct;
        SELECT standard_price INTO price FROM PetCategories WHERE name=pct;
        INSERT INTO CanTakeCare VALUES (ct, pct, COALESCE(price, 200))
        ON CONFLICT DO NOTHING;
        SELECT daily_price INTO price FROM CanTakeCare
        WHERE category=pct AND ct_id=ct;
        SELECT (e_date - s_date) INTO days FROM Requests
        WHERE pet_id=pet AND s_date=sd;
        UPDATE Transactions SET cost = price * days
        WHERE pet_id=pet AND s_date=sd AND ct_id=ct;
        UPDATE Transactions T SET status='Declined'
        WHERE status='Pending' AND ct_id=ct
          AND NOT is_available(ct, T.s_date,
              (SELECT e_date FROM Requests R
              WHERE T.s_date=R.s_date AND T.pet_id=R.pet_id));
    END IF;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;
CREATE TRIGGER confirm_transaction
    AFTER INSERT OR UPDATE OF status ON Transactions FOR EACH ROW EXECUTE
PROCEDURE confirm_transaction();
```

Explanation:
Whenever a Transaction is confirmed, which is implemented by `UPDATE` query, there are a few things to update.

1. All the pending Transactions on this Request are updated to outdated status. In the code above, the status of other confirmed Transactions of the same Request are updated to outdated. This is to ensure that only one Transaction is actually confirmed, especially if the Pet Owner sends multiple Requests simultaneously.

2. If a Care Taker accepts a Transaction of a Pet Category that he or she has not declared as Can Take Care, the Pet Category will automatically be updated in the list of Pet Categories that this Care Taker Can Take Care. The daily price is fetched from the standard daily price of the Pet Category, which the Care Taker can update later by him or herself. If the standard price is not yet declared (as Pet Owners can possibly add new Pet Categories to the system), the system will use a default value of 200.
3. Decline all the pending Transactions that this Care Taker can no longer take care of after accepting this Transaction if at least one day during the Transaction would have the total number of Transactions exceeding the maximum allowed number of pets being taken care of. This is checked by calling upon the `is_available` function as mentioned previously.

Examples of SQL queries

* {$n=xxx} are those parts replaced with $1, $2, etc. in the actual query stated in JS and to be filled with actual values of variables when the query is executed in the pool. n is the index used in actual JS file. xxx is the actual value that is used to replace it.
** Ln n (in `WHERE` clause) means the n-th condition that is joined using `AND`. For example, for `WHERE A AND B AND C`, `A` is Ln 0, `B` is Ln 1, and `C` is Ln 2. [n] before each `AND` is the index, which will not appear in the actual query.

1. Find Care Takers of a Pet during a particular period of time with filter conditions

```
SELECT CT.userid AS userid, U.name AS name, CT.rating AS rating,
   CTC.daily_price AS daily_price,
   CASE
       WHEN EXISTS(SELECT 1 FROM FullTimeCareTakers F
           WHERE F.userid=CT.userid) THEN 'Full time'
       ELSE 'Part time' END
   AS category
FROM (Users U INNER JOIN CareTakers CT on U.userid = CT.userid)
   LEFT JOIN CanTakeCare CTC ON CT.userid=CTC.ct_id
WHERE {$1=category of this pet} NOT IN
   (SELECT category FROM CannotTakeCare CN WHERE CN.ct_id=CT.userid)
[1]AND is_available(CT.userid, {$2=s_date}, {$3=e_date})
[2]AND NOT EXISTS(SELECT 1 FROM Transactions T
   WHERE T.ct_id=CT.userid AND T.pet_id={$11=pet ID}
   AND T.s_date={$2=s_date}
   AND (T.status='Pending' OR T.status='Confirmed' OR T.status='Declined'))
[3]AND CT.userid LIKE '%'||{$4=Care Taker’s user ID contains filter}||'%'
[4]AND U.name LIKE '%'||{$5=Care Taker’s name contains filter}||'%'
[5]AND CT.userid IN (SELECT F.userid FROM FullTimeCareTakers F)
[6]AND CT.rating >= {$6=Care Taker’s minimum average rating filter}
[7]AND CTC.daily_price IS NOT NULL
[8]AND CTC.daily_price <= {$7=Maximum daily price filter}
[9]AND (SELECT AVG(rate)
   FROM Transactions T INNER JOIN Pets P ON T.pet_id=P.petid
   WHERE T.ct_id=CT.userid AND P.category={$1=category of this pet}) >=
       {$8=Minimum average rating on this pet category filter}
[10]AND (SELECT AVG(rate)
   FROM Transactions T INNER JOIN Pets P ON T.pet_id=P.petid
   WHERE T.ct_id=CT.userid AND P.owner={$9=user ID of this Pet Owner})
       >= {$10=Minimum average rating given by this user filter}
[11]AND EXISTS(SELECT 1 FROM Transactions T
   WHERE T.ct_id=CT.userid AND T.pet_id={$11=pet ID of this Pet}
   AND T.status='Confirmed');
```

Explanation:

1. This query aims to find information of all Care Takers that a Pet Owner can send a Request to while meeting some of the conditions the User specifies to filter. This query is used in the handle_transaction route.
2. All the information selected is to be displayed on a frontend table. The above is the entire filter query that is used when all the filter fields are filled. In actual execution, the query string is formed using a function based on the input in the filters. Some of the lines of check can be skipped to save time.
3. Information known includes information about the Pet and Request as they are retrieved in the same page before the execution of this query as a precondition to render the page. In particular, primary key information about a Request is included in the route.
4. `is_available(ct VARCHAR, s_date DATE, e_date DATE)` is a predefined function in the database that checks whether a Care Taker `ct` is available to accept new Requests from `s_date` to `e_date` in terms of time. Criteria also includes the Care Taker is available as in the Periods Available table and does not have any day in the Period with the number of Pets being taken care of exceeding the limit. This function is extracted because this piece of code is completely not trivial, it is reused in other functions and procedures, and a `while` loop on all dates within the `s_date` and `e_date`. The implementation of `is_available` is given as follows:

```
CREATE OR REPLACE FUNCTION
is_available(ct VARCHAR, s_date DATE, e_date DATE)
RETURNS BOOLEAN AS
   $$
   DECLARE
       d DATE := s_date;
       res BOOLEAN := TRUE;
   BEGIN
       WHILE d<=e_date AND res LOOP
           IF NOT EXISTS(
               SELECT 1 FROM PeriodsAvailable PA
               WHERE PA.ct_id=ct AND d<=PA.e_date AND d>=PA.s_date
               ) THEN
               res:= FALSE;
           ELSEIF (
               SELECT COUNT(*)
               FROM Transactions T NATURAL JOIN Requests R
               WHERE T.ct_id=ct AND status='Confirmed'
               AND d<=R.e_date AND d>=R.s_date
               ) >= (
               SELECT CASE
                   WHEN EXISTS(
                       SELECT 1 FROM FullTimeCareTakers F
                       WHERE F.userid=ct) THEN 5
                   ELSE CASE WHEN (
                           SELECT (COALESCE(rating, 0))
                           FROM CareTakers C
                           WHERE C.userid=ct
                       ) >= 4
                       THEN 4 ELSE 2
                   END
               END
                   ) THEN res:= FALSE;
```

```
                END IF;
                d:=d+1;
            END LOOP;
            RETURN res;
        END;
        $$
    LANGUAGE plpgsql;
```
5. Ln 0-2 of the `WHERE` clause filters out all the possible Care Takers without filter; Ln 3-4 are filters of strings that must be contained in ID and name; Ln 5 filters the category of Care Taker based on whether they are full-time or part-time, which full-time can be replaced with part-time in this line depending on App User's choice in filter; Ln 6 filters out Care Takers with average rating below the specified value; Ln 7 checks that the Care Takers declares this Pet Category as one that he or she can definitely take care of, and only when Ln 7 is checked will Ln 8 be checked for the maximum daily price.
6. Ln 9-11 of the `WHERE` clause checks the experience and collaboration history of the Care Taker and this Pet Owner. The actual query is slightly different from what is shown in this document because some of the lines may not be executed based on the query construction as mentioned in point 2 of explanation.
7. Ln 9 checks the minimum rating of a Care Taker of this Pet Category. If the user only wants to check that this Care Taker has experience in this Pet Category, the line executed should replace AVG(rate) with anything (to check existence) and add a `WHERE` constraint to check the status is confirmed. Similarly for Ln 10 which checks the average rating given by this Pet Owner and Ln 11 which checks existence of confirmed Transactions on this Pet.

2. Find 3 Pet Categories a Care Taker is best performing at

```
SELECT P.category
FROM (Pets P NATURAL JOIN CanTakeCare CTC) INNER JOIN
    (SELECT
        CASE WHEN stddev = 0 THEN 0 ELSE (T1.rate - avg) / stddev END AS rate,
        T1.pet_id AS pet_id, T1.ct_id AS ct_id
    FROM (Transactions T1 INNER JOIN Pets P1 ON T1.pet_id=P1.petid) INNER JOIN (
        SELECT AVG(T2.rate) AS avg, STDDEV(T2.rate) AS stddev, P2.owner AS owner
        FROM Transactions T2 INNER JOIN Pets P2 ON T2.pet_id=P2.petid
        GROUP BY P2.owner
        ) PT ON P1.owner=PT.owner
    WHERE T1.status='Confirmed') T
    ON P.petid=T.pet_id AND CTC.ct_id=T.ct_id
WHERE CTC.ct_id={$1=the User ID of this Care Taker}
GROUP BY P.category
HAVING COUNT(*) >= 10
ORDER BY AVG(T.rate) DESC
LIMIT 3;
```
Explanation:

1. This query aims to find the 3 (or less) Pet Categories that this Care Taker is best performing at to show in the Care Taker's profile page.
2. Criteria of the 3 Pet Categories are as follows:
    a. The Care Taker must have declared this Pet Category as a definitely Can Take Care category, and has involved in at least 10 confirmed Transactions on this Pet Category
    b. The average normalized rate for all rated confirmed Transactions of this Care Taker on this Pet Category should be the highest among all Pet Categories that are

described in a. The normalization of rate is done by getting the rate of the Transaction subtracted by the average rate given by this Pet Owner, and then divided by the standard deviation of rate given by this Pet Owner. If the standard deviation is 0, then the normalized rate is simply 0.

3. The average and standard deviation of rate given by one Pet Owner is calculated using aggregate `GROUP BY` owner before other manipulations process the Transaction table to use normalized rating before joining to other tables. The selection of at least 10 confirmed Transactions and top 3 average rated categories is done by aggregate `GROUP BY` category and `ORDER BY` average rate.

3. Find the bottom 5 underperforming Care Takers of the month

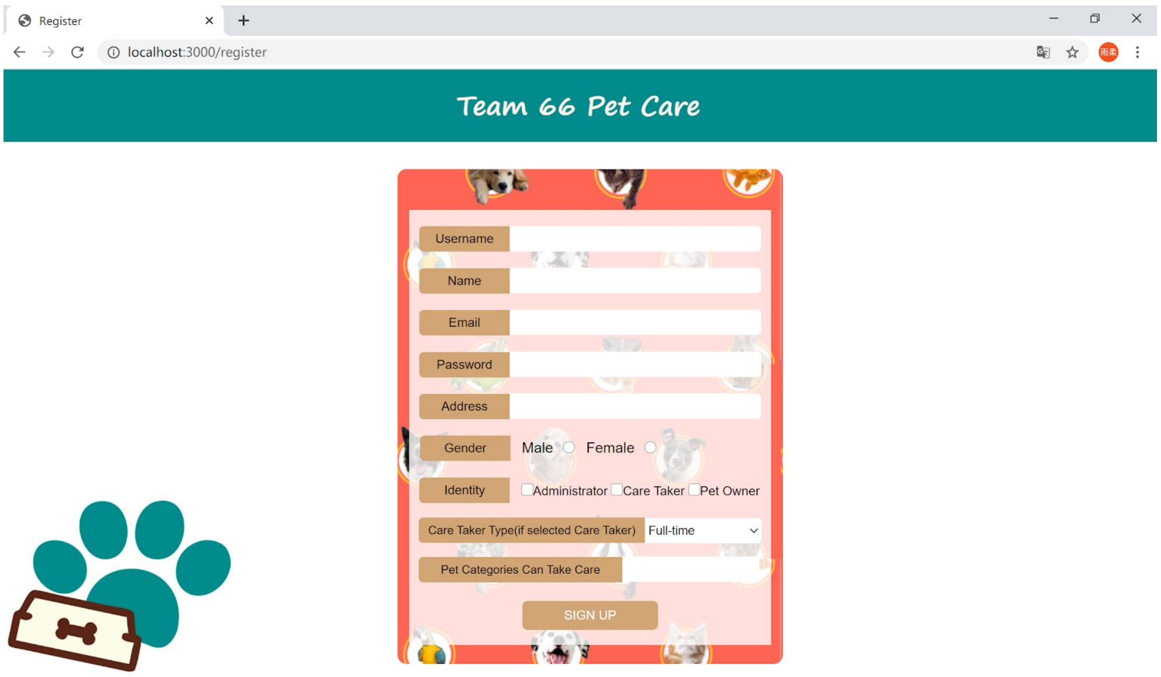
```
WITH monthly
AS
(SELECT
      CASE WHEN e_date <= {$1 = last day of selected month} AND
            s_date >=  {$2 = first day of selected month} THEN e_date - s_date
      WHEN e_date<= {$1 = last day of selected month} AND
            s_date <= {$2 = first day of selected month} THEN e_date -
            {$2 = first day of selected month}
      WHEN e_date >= {$1 = last day of selected month} AND
            s_date >= {$2 = first day of selected month} THEN {$1 = last day of
            selected month} - s_date
      ELSE {$1 = last day of selected month} - {$2 = first day of
            selected month} END AS days, ct_id, rate
FROM transactions NATURAL JOIN requests
      WHERE e_date >= {$2 = first day of selected month} AND
      s_date <= {$1 = last day of selected month} AND status = 'Confirmed' )

SELECT name, ct_id, avg(rate * days / days) AS weightedAverage
      FROM monthly INNER JOIN users ON monthly.ct_id = users.userid
      GROUP BY name, ct_id
      ORDER BY weightedAverage
      LIMIT 5;
```

Explanation:

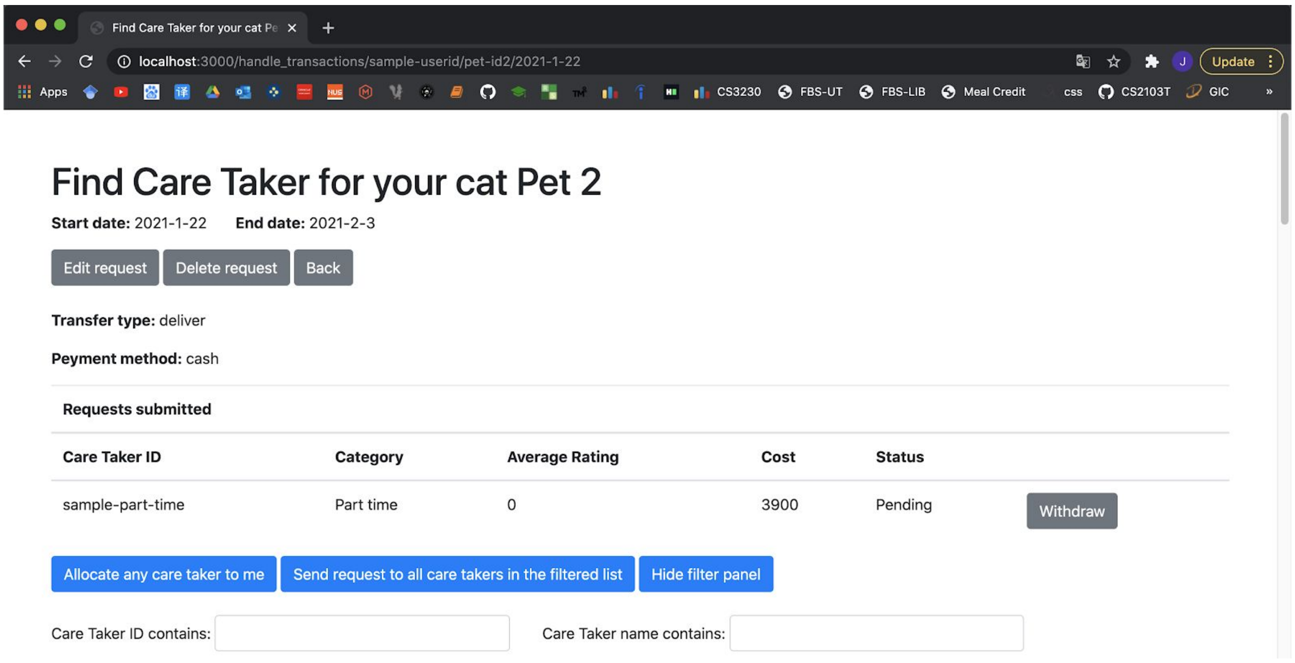
1. This query allows the PSCAdministrators to see who the top 5 underperforming Care Takers were for a particular month, along with their average weighted rating for the month.
2. Only transactions that were 'Confirmed' will be considered.
3. Only transactions that have a rate of not null will be considered.
4. If the start or end date of the service was outside of the month, only service during the month will be used in the calculations.
5. Weighted rating is calculated by taking the sum of the rating received for a service * number of days divided by the total number of days worked for the month.

User Interface

1. Register page (representative of design): user inputs the necessary credentials and the system saves it into the database. The identity is set to multiple choices so that user can be both a Care Taker and a Pet Owner. The Care Taker Type is set to mono choice options so the Care Taker can either be full-time or part-time.



2. Find Care Taker (representative of functionality): for a Pet Owner to find Care Taker for a Pet of him or her. On the top it shows the basic information of this Request, including start date, end date, transfer type, and payment method. There is an “Edit request” button that can toggle an edit panel of the transfer type and payment method of this Request. The User can also choose to delete this Request by clicking on “Delete request” button, which will redirect the user to the Pet page after deletion. Then there is a table showing all the submitted (excluding withdrawn) Requests. The Pet Owner can choose to Withdraw whenever the Transaction is not accepted by a Care Taker. The Pet Owner can choose to allocate anyone to handle this Request, using filter to find some Care Takers and send Requests to all, or send Requests individually.


Find Care Taker for your cat Pe
localhost:3000/handle_transactions/sample-userid/pet-id2/2021-1-22
Edit request
Delete request
Back
Withdraw
Allocate any care taker to me
Send request to all care takers in the filtered list
Hide filter panel

Find Care Taker for your cat Pet 2

Start date: 2021-1-22 **End date:** 2021-2-3

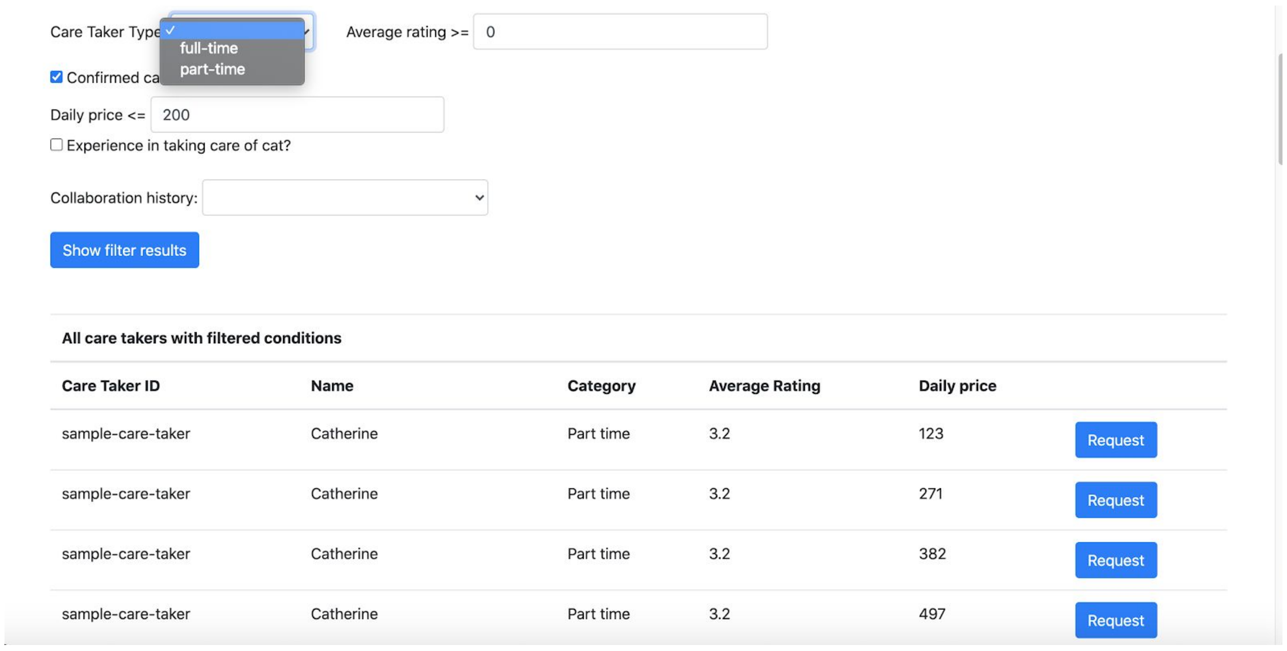
Transfer type: deliver

Peyment method: cash

Requests submitted

Care Taker ID	Category	Average Rating	Cost	Status	
sample-part-time	Part time	0	3900	Pending	Withdraw

Care Taker ID contains: Care Taker name contains:


Care Taker Type ✓ full-time part-time

Average rating >= 0

[✓] Confirmed ca

Daily price <= 200

[ ] Experience in taking care of cat?

Collaboration history:

All care takers with filtered conditions

Care Taker ID	Name	Category	Average Rating	Daily price	
sample-care-taker	Catherine	Part time	3.2	123	Request
sample-care-taker	Catherine	Part time	3.2	271	Request
sample-care-taker	Catherine	Part time	3.2	382	Request
sample-care-taker	Catherine	Part time	3.2	497	Request

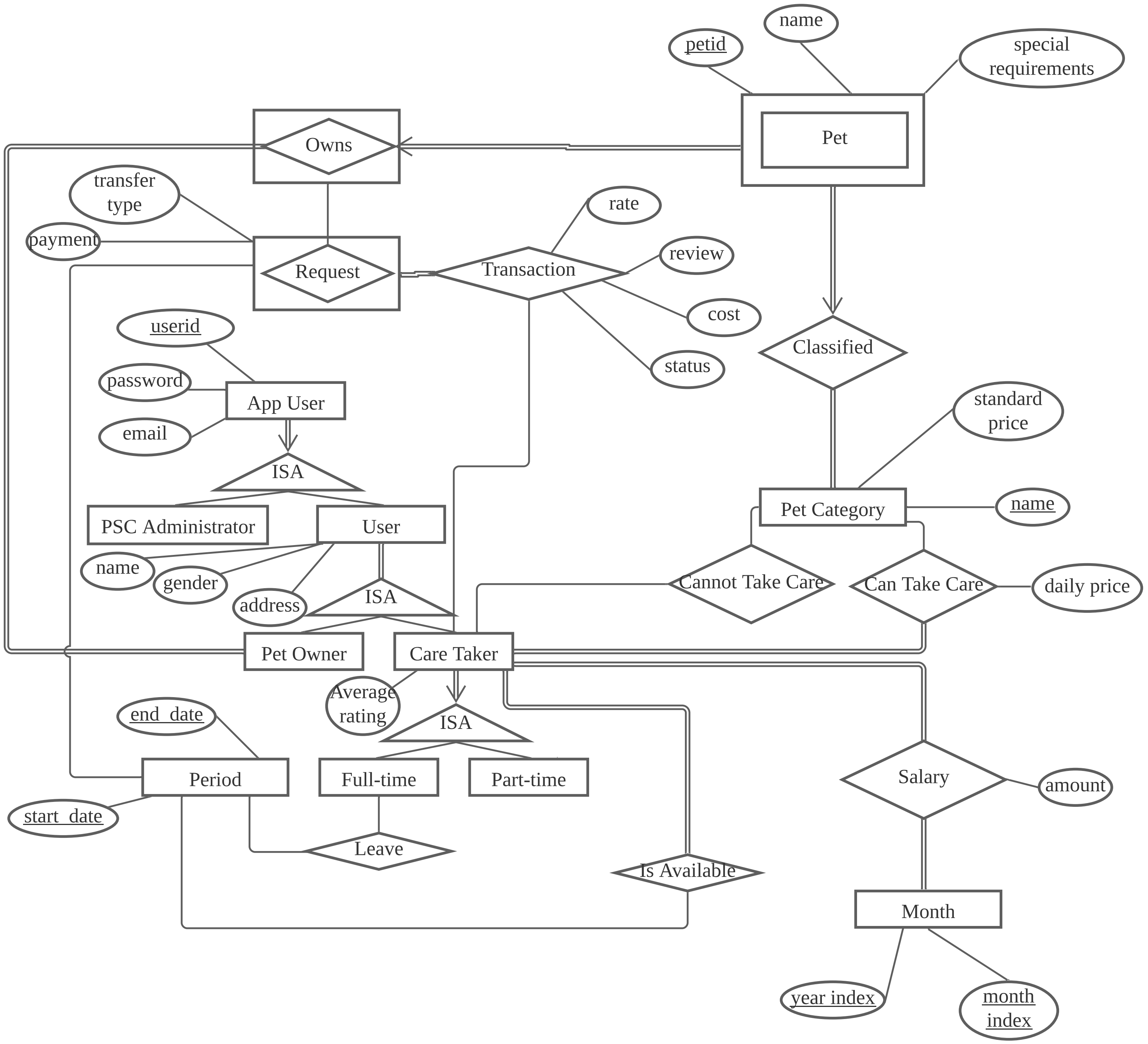
Difficulties encountered

Difficulty	Solution
Some of us cannot connect to the Heroku database correctly.Some of the teammates who are overseas cannot go on Heroku.	Some are using NUS WiFi, which blocks the connection on Heroku. The solution is obtained from CS2102 Coursemology forum.
Most of us do not have the base knowledge to code in JavaScript and HTML language before this module.	We learnt online by myself and also gained help from friends but still not so good at it.
Some of the teammates who are overseas cannot go on Heroku. When the log in can be done via NUSVPN, the real connection still fails(maybe because of the NUS wifi).	The usage of NUS VPN can solve the problem of invalid requests when prompted to log in. However, during the feature implementation period, the local database is used to test but cannot connect to the real heroku database.
The filter query is too long and the user may not fill in all fields to filter, which may make the execution unnecessarily slow. → Skipping some lines in a query will make the `pool.query` method give an error such that the arguments supplied does not match the ones needed in the query.	Use a function to construct the query string for the filter instead of hardcoding it so that every line should be executed. Add a safeguard line that uses all the input at the end of the query such that it will not affect the actual data selected, i.e. will always return true.
There might be multiple submit buttons (that uses the value of fields of input in the front end) on the same view page. In particular, they may use the same inputs. → With multiple POST method path, the length of the URL increases, and if there is a sequence of submission that do not redirect to other pages, the length of URL will keep increasing	Use the same form but different formaction. Write different code to different formaction path in the controller file. Use redirect to direct back to this page.

→ The details of some input fields are lost.	Use a query in the URL to pass the information. Use the ‘url’ to construct the URL.
Want to toggle some panels for Filter based on their value.	Set an onchange event to the selection box and write a script to change the display status of the panel.
Want to remember the state of display of a panel in the frontend that is controlled only by a button.	Use a hidden input to pass the value to the controller. The value of hidden input is updated when an onclick event happens to the button.
When importing packages, some variables are not properly declared and renders not found problems continually.	Check by commenting some syntax related to the packages.
Had many problems attempting to update the heroku database, unable to drop triggers and unable to see updated columns for edited tables.	Required much trial and error to add/drop tables until all members finally had the same information on hand.

Appendix

(see the next page)


petid
name
special requirements
Pet
Owns
transfer type
payment
Request
Transaction
rate
review
cost
status
Classified
userid
password
App User
email
ISA
PSC Administrator
User
name
gender
address
ISA
Pet Owner
Care Taker
standard price
Pet Category
name
Cannot Take Care
Can Take Care
daily price
Average rating
ISA
end_date
Period
Full-time
Part-time
start_date
Leave
Is Available
Salary
amount
Month
year index
month index