CS2102-2021-S1-33 Project Report

Table of contents:

Project Responsibilities

NADIA ATIQAH BINTE MOHD YAHYA (E0318573)
Role: Frontend Developer
Project Contributions:

- Created the preliminary ER diagram for the application
- Created the mock-up for the application pages
- Wrote the HTML and CSS for most of the application pages
- Made justifications for whether database is in BCNF or 3NF

LIM LI QUAN (E0407432)
Role: Backend Developer
Project Contributions:

- Created and update the ER diagram for the application
- Modified SQL schema, procedures and triggers to better translate the ER diagram and constraints
- Help to implement some backend APIs using ExpressJS
- Ensure correctness of backend APIs, SQL schema, procedures and triggers

KOH RAYSON (E0014788)
Role: Backend and DevOps Lead
Project Contributions:

- In charge of deploying the application using Heroku PAAS
- Set up the backend service and APIs using ExpressJS
- Set up API documentation website using Swagger
- Implemented initial SQL schema based on preliminary ER diagram and constraints
- Made justifications for whether database is in BCNF or 3NF
- Integrated code changes to the backend service
- Wrote the preliminary constraints for the application

CLARISSA LAURENT YEE QI XUAN (E0324450)
Role: Frontend Developer
Project Contributions:

- Wrote HTML and CSS for the landing page, PCS Admin page and Pet Owner profile page
- Wrote preliminary constraints for the application
- Wrote about project functionalities for the report

BALAM SAI ROHIT REDDY (E0376617)
Role: Frontend Lead
Project Contributions:

- Wrote preliminary constraints for the application, set up the git org and repo
- Set up the frontend service using Angular and wrote the HTML and CSS for some of the pages which include login and signup
- Wrote the TypeScript code for all the frontend files, including API calls and auth services
- In charge of making the video for presentation

App Functionalities

- Pet owners can search for caretakers and see their reviews and ratings.
- Pet owners can bid for the service of this caretaker by filling in a form denoting their pet for caretaking, start date and end date, phone number and bid price.
- Users can view their profiles and edit their personal information. From the profile page, pet owners can view their pet information and average caretaking cost of each pet.
- Caretakers can also view their upcoming caretaking jobs, caretaking jobs history and bids pending action. From this page, they can view the total number of pet-days for the current month.
- PCS Admin can verify caretakers, approve full-time caretakers' leave and add a new pet category. They can also view the following summary information: list of underperforming full-time caretakers.

Data Requirements and Constraints

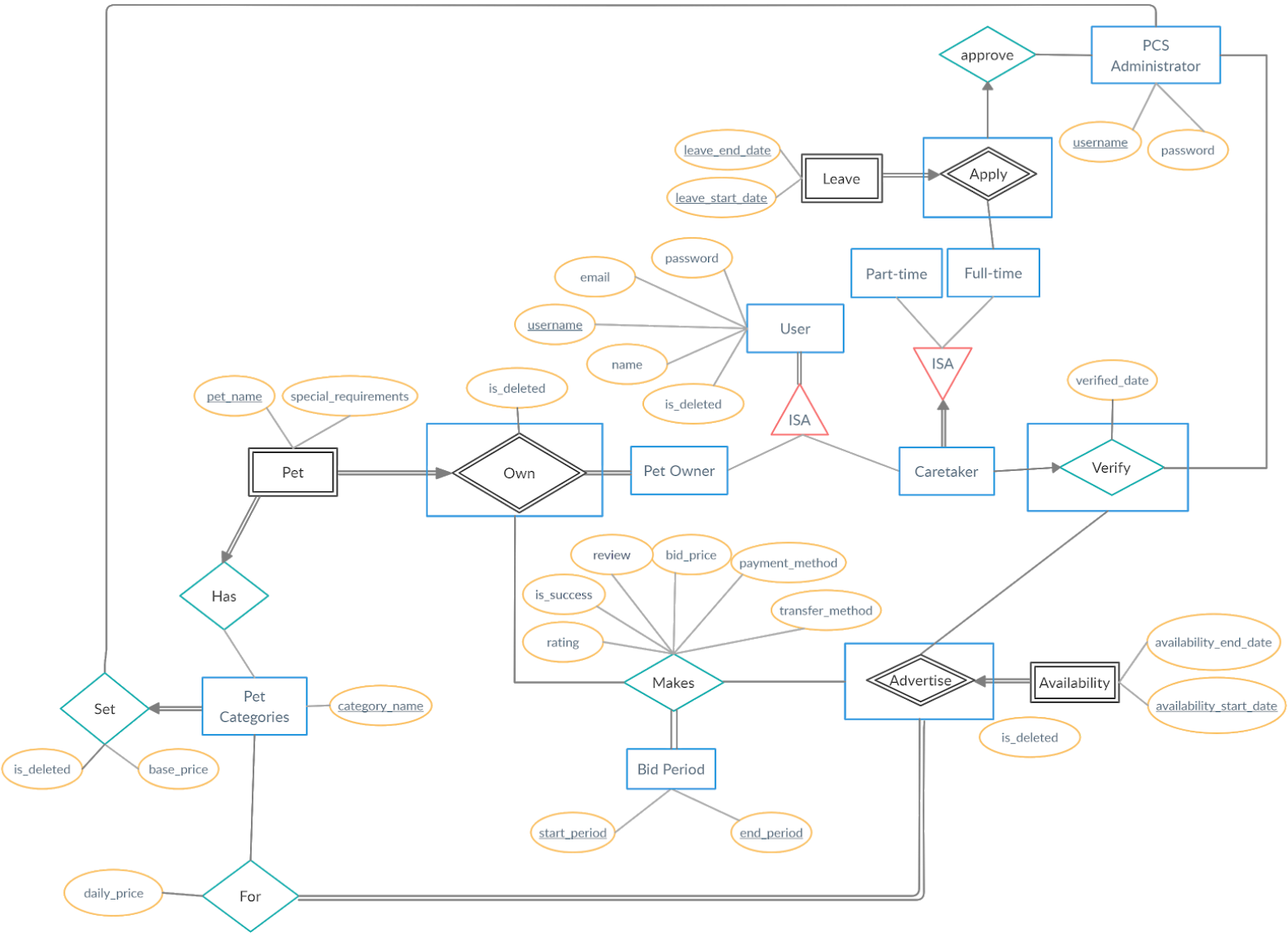
Note: Constraints that were not captured by our ER diagram will be indicated as a note beside the relevant constraints.

1. Users are identified by their username. For each user, their name, password, address, email address and account status (is_deleted) must be recorded.
2. Instead of removing the record of the user when deleting the user, his/her record will be marked as "deleted" in their account status attribute. **(Not captured in ER diagram)**
3. PCS administrators are identified by their username. For each PCS administrator, their password must be recorded.
4. Each user can only be a caretaker or a pet owner or both a caretaker and pet owner.
5. A caretaker must be verified by a PCS Administrator before advertising their availabilities and taking on any caretaking jobs.
6. Each pet owner must own 1 or more pets.
7. Each pet must be owned by exactly 1 pet owner. The pet name only uniquely identifies a pet from among the pets that are owned by its pet owner.
8. For each pet, their special caretaking requirements must be recorded.
9. Each pet must belong to exactly 1 pet category.
10. Pet Categories are identified by their name.
11. The PCS administrator sets the base daily price for the caring of each pet category. Every pet category must have exactly one base daily price.
12. Each caretaker must be either a full-time caretaker or part-time caretaker.
13. A caretaker can advertise their availability, which consist of the start and end date of their availability period. For a particular caretaker A, the start date of one of their availability periods only uniquely identifies that availability period from among all availability periods of A.
14. A full-time caretaker is treated as available for 1 year from the date of account creation unless they apply for leave and the leave is approved. Part-time caretakers can specify their availability for the current and next year. **(Not captured in ER diagram)**
15. For each availability period, the caretaker can specify exactly one daily price for each pet category that they are able to take care of.

16. If the caretaker is working full-time, then the daily price should not be set lower than the base daily price set by the PCS Admin for that particular pet category. **(Not captured in ER diagram)**
17. A caretaker must specify at least one pet category that they are able to care for, whenever they are available.
18. A caretaker cannot care for pets that do not belong to the set of pet categories that they are able to care for.
19. A pet owner can bid for the services of caretakers for their pets based on the caretakers' advertised availability, the pet categories that the caretakers are able to care for and the daily rate.
20. If a pet owner PO bids for a caretaker C to care for his pet P based on C's advertised availability_start_date A from the period bid_start_period BS to bid_end_period BE, the bid made will be identified by the username of PO, username of C, pet name of P, A, BS and BE. For each bid, the bid price and is_successful field must be recorded.
21. A pet owner cannot bid for the services of a caretaker to care for a pet that they do not own.
22. A pet owner looking for caretaking services for their pet P that has a pet category C cannot make a bid for a caretaker that did not advertise C as a pet category that he/she is able to care for. **(Not captured in ER diagram)**
23. A pet owner cannot bid for the services of a caretaker for a caretaking period that does not fall within the caretaker's availability period. For example, if caretaker A is available from 12 September 2020 to 12 November 2020, pet owner B cannot make a bid for A to care for his pet from 11 September 2020 to 13 November 2020. **(Not captured in ER diagram)**
24. A pet owner can bid for the services of a caretaker for the same availability period, but with a different start and end period of the caretaking service. For example, if Caretaker A is available from 12 September 2020 to 12 November 2020, pet owner B can make separate bids for A to care for his pet from 20 September 2020 to 25 September 2020 and 10 October 2020 to 15 October 2020.
25. The pet owner cannot make a bid for a given pet category and caretaker availability period with a lower bid price than the daily price that was set by the caretaker for that corresponding pet category and availability period. **(Not captured in ER diagram)**
26. The part-time caretaker can choose to accept or not accept bids for their service from pet owners.
27. For full-time caretakers, the system will automatically accept a bid on their behalf if the caretaker is available and he/she has not reached the maximum quota of caring for 5 pets at any one time. **(Not captured in ER diagram)**
28. Once the caretaker has chosen to accept or not accept a particular bid, he/she cannot undo his/her decision. **(Not captured in ER diagram)**
29. Once a bid is accepted, it will be proceeded by exactly one transaction. The price of the transaction is set as the bid price multiplied by the number of caretaking days.
30. Payment method of successful bids can be either credit card or cash. **(Not captured in ER diagram)**
31. Transfer method of the pet can be either the pet owner delivers, or the caretaker picks up, or transfer through the physical PCS building. **(Not captured in ER diagram)**
32. After the end of the care period, the pet owner can submit exactly one review and exactly one rating, where 1 is the worst rating and 5 is the best rating, for the caretaker.

33. Full-time caretakers can take care of more than one pet, up to a maximum of 5 pets, at any given time. **(Not captured in ER diagram)**
34. Part-time caretakers can take care of more than 2 pets, up to a maximum of 5 pets, if their average rating is greater than or equal to 4 out of 5. If their average rating is lower than 4 out of 5, they can take care of at most 2 pets only. **(Not captured in ER diagram)**
35. Each full-time caretaker must work for a minimum of 2 x 150 consecutive days a year. **(Not captured in ER diagram)**
36. A full-time caretaker can apply for leave while a part-time caretaker cannot apply for leave.
37. For a caretaker A, the start and end date of the leave application only uniquely identifies a leave application from among all leave applications of A.
38. Full-time caretakers cannot apply for leave if there is at least one Pet under their care. **(Not captured in ER diagram)**
39. Before the full-time caretaker can go on leave, the leave application must be approved by the PCS administrator.
40. The salary of a full-time caretaker depends on the number of pet-days, which is the sum of number of days per pet, for all pets that the caretaker takes care of in a given month. For up to 60 pet-days, the full-time caretaker will be paid $3000. For any excess pet-days, the full-time caretaker will be paid 80% of their price. **(Not captured in ER diagram)**
41. The part-time caretaker will earn 75% of their stated price from each caretaking job. PCS will take the remaining 25%. **(Not captured in ER diagram)**

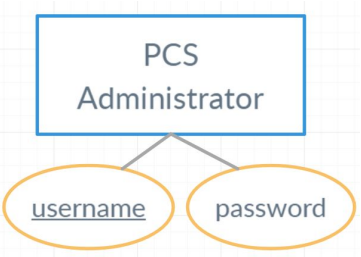
ER Diagram


approve
PCS Administrator
username
password
leave_end_date
leave_start_date
Leave
Apply
password
email
username
name
is_deleted
User
Part-time
Full-time
ISA
ISA
verified_date
pet_name
special_requirements
is_deleted
Pet
Own
Pet Owner
Caretaker
Verify
review
bid_price
payment_method
is_success
transfer_method
rating
Has
Makes
Advertise
Availability
availability_end_date
availability_start_date
is_deleted
Set
Pet Categories
category_name
is_deleted
base_price
Bid Period
start_period
end_period
daily_price
For

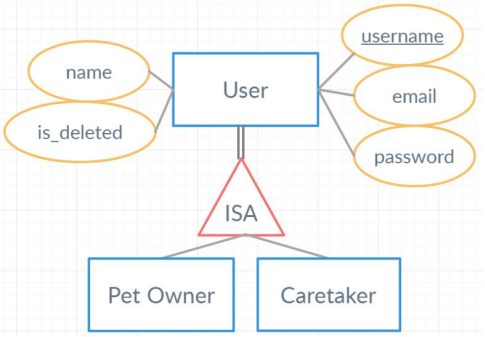
SQL Schema

Note: For each SQL table, we will show the relevant portion in the ER diagram for reference. There may be a short text below certain SQL tables for non-trivial justifications.

```
-- ADMIN
CREATE TABLE pcs_admins (
  username VARCHAR PRIMARY KEY,
  password VARCHAR NOT NULL
);
```




```
-- User
-- ISA is with covering constraint
CREATE TABLE pcs_user (
  username VARCHAR PRIMARY KEY,
  email VARCHAR UNIQUE NOT NULL,
  name VARCHAR NOT NULL,
  password VARCHAR NOT NULL,
  is_deleted BOOLEAN DEFAULT FALSE
);
```



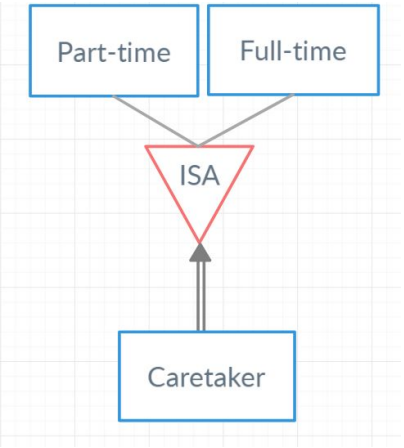

```
CREATE TABLE pet_owners (
  username VARCHAR PRIMARY KEY,
  FOREIGN KEY (username) REFERENCES pcs_user(username)
);
```

```
CREATE TABLE caretakers (
  username VARCHAR PRIMARY KEY,
  FOREIGN KEY (username) REFERENCES pcs_user(username)
);
```

As a user must be a pet owner or a caretaker (covering constraint), our group has decided to adopt a 3 table approach (**pcs_user**, **pet_owners** and **caretakers**) to consolidate the information of the user. In addition, for auditing purposes, deleted user's transactions should be retained in the database. As such, our group has adopted the approach to mark the user as deleted instead of removing the relevant records of the user in the case of user deletion.

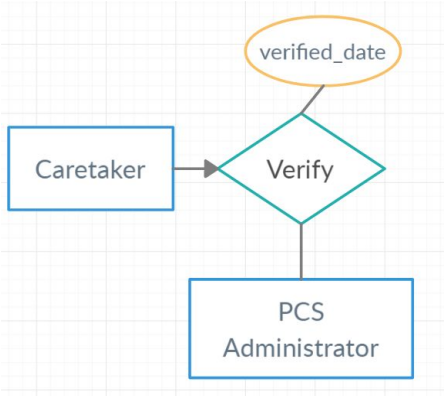
```sql
CREATE TABLE part_time_caretakers (
  username VARCHAR PRIMARY KEY,
  FOREIGN KEY (username)
    REFERENCES caretakers(username)
);

CREATE TABLE full_time_caretakers (
  username VARCHAR PRIMARY KEY,
  FOREIGN KEY (username)
    REFERENCES caretakers(username)
);
```



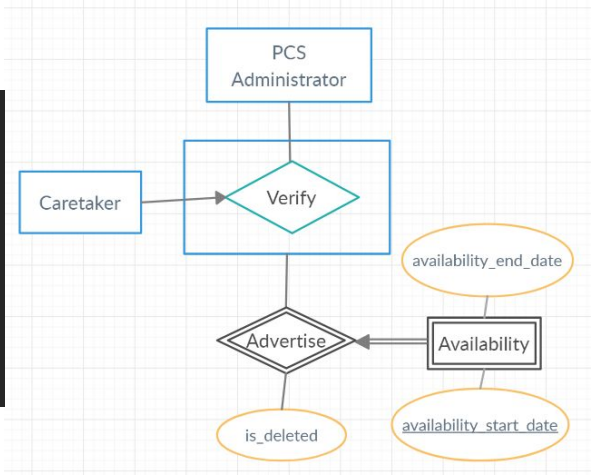
Since there exists a **caretakers** table, the **part_time_caretakers** and **full_time_caretakers** would reference the **caretakers** table instead of the **pcs_user** table. In addition, we have a trigger available to ensure that a caretaker cannot be both a part-time and a full-time caretaker.

```sql
CREATE TABLE verified_caretakers (
  username VARCHAR PRIMARY KEY,
  admin_username VARCHAR NOT NULL,
  verified_date DATE,
  FOREIGN KEY (admin_username)
    REFERENCES pcs_admins(username),
  FOREIGN KEY (username)
     REFERENCES caretakers(username)
);
```



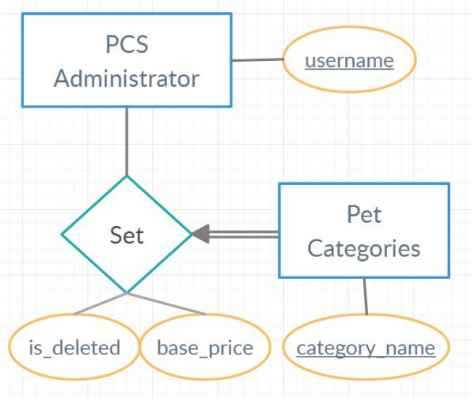
username is a primary key in **verified_caretakers** because of the key constraint with respect to the **Verify** relationship set.

```sql
CREATE TABLE advertise_availabilities (
  username VARCHAR,
  availability_start_date DATE,
  availability_end_date DATE,
  is_deleted BOOLEAN DEFAULT FALSE,
  PRIMARY KEY (username, availability_start_date),
  FOREIGN Key (username)
    REFERENCES verified_caretakers (username)
);
```

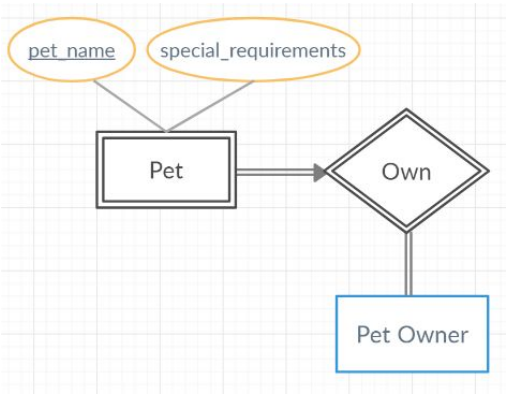


Availability of a verified caretaker is an entity instead of an attribute because a verified caretaker can advertise many availabilities throughout the caretaker's career. However, availability cannot uniquely identify an entity as such it has an identity dependency on verified caretaker. In addition, **verified_caretakers** is evaluated to an aggregation due to the constraint that only **verified_caretakers** can advertise availability.

```sql
CREATE TABLE pet_categories (
  pet_category_name VARCHAR PRIMARY KEY,
  set_by VARCHAR NOT NULL,
  base_price INTEGER NOT NULL,
  is_deleted BOOLEAN DEFAULT FALSE,
  FOREIGN KEY (set_by)
    REFERENCES pcs_admins(username),
  CHECK (base_price > 0)
);
```

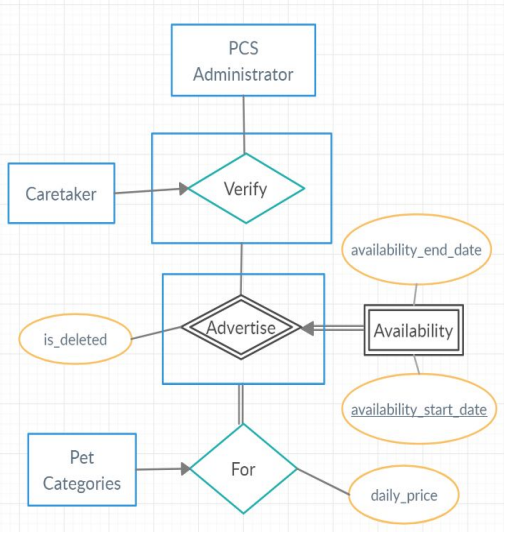



```sql
CREATE TABLE owned_pets (
  username VARCHAR NOT NULL,
  pet_name VARCHAR,
  special_requirements VARCHAR,
  pet_category_name VARCHAR NOT NULL,
  is_deleted BOOLEAN DEFAULT FALSE,
  PRIMARY KEY(username, pet_name),
  FOREIGN KEY (username)
    REFERENCES pet_owners(username),
  FOREIGN KEY (pet_category_name)
    REFERENCES pet_categories (pet_category_name)
);
```



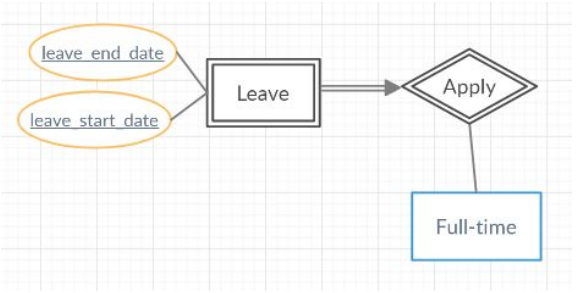
Due to **Pet** having an identity dependency on **Pet Owner,** the primary key of **owned_pets** is a combination of ***pet_name*** and ***username***.

```sql
CREATE TABLE advertise_for_pet_categories (
  username VARCHAR,
  availability_start_date DATE,
  pet_category_name VARCHAR,
  daily_price INTEGER NOT NULL CHECK (daily_price > 0),
  FOREIGN KEY (username, availability_start_date)
    REFERENCES advertise_availabilities(username,
      availability_start_date),
  FOREIGN KEY (pet_category_name)
    REFERENCES pet_categories(pet_category_name),
  PRIMARY KEY (username,
    availability_start_date, pet_category_name)
);
```



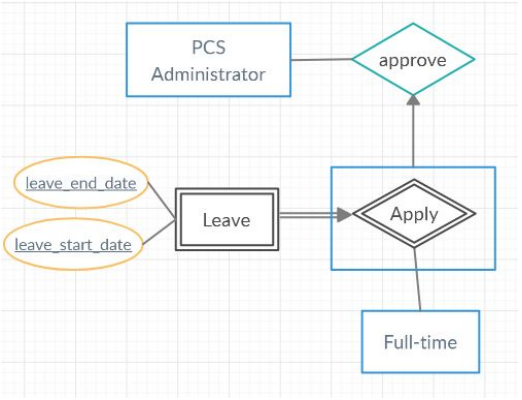
Based on the constraint that pet owners can only bid for a verified caretaker that has advertised for a particular pet category, there is a need to evaluate **advertise_availabilities** to an aggregate entity to have a relation with the **pet_category** entity.

```
CREATE TABLE apply_leaves (
  username VARCHAR,
  leave_start_date DATE,
  leave_end_date DATE,
  FOREIGN KEY (username)
    REFERENCES full_time_caretakers(username),
  PRIMARY KEY (username,
    leave_start_date, leave_end_date)
);
```



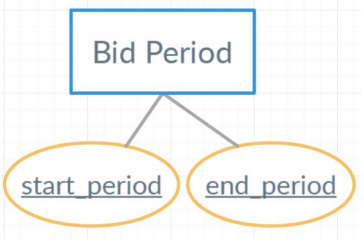
Due to **Leave** having an identity dependency on **Full-time Caretakers,** the primary key of **apply_leaves** is a combination of ***leave_start_date, leave_end_date*** and ***username***.

```
CREATE TABLE approved_apply_leaves (
  username VARCHAR,
  admin_username VARCHAR NOT NULL,
  leave_start_date DATE,
  leave_end_date DATE,
  FOREIGN KEY (username)
    REFERENCES full_time_caretakers(username),
  FOREIGN KEY (admin_username)
    REFERENCES pcs_admins(username),
  PRIMARY KEY (username, leave_start_date,
        leave_end_date)
);
```



Due to the constraint that a leave made by a full time caretaker has to be approved by the admin, the apply relation has to be evaluated to an aggregate entity to have a relation with **pcs_admin**.

```
CREATE TABLE bid_period (
  bid_start_period DATE,
  bid_end_period DATE,
  PRIMARY KEY (bid_start_period,
    bid_end_period)
);
```



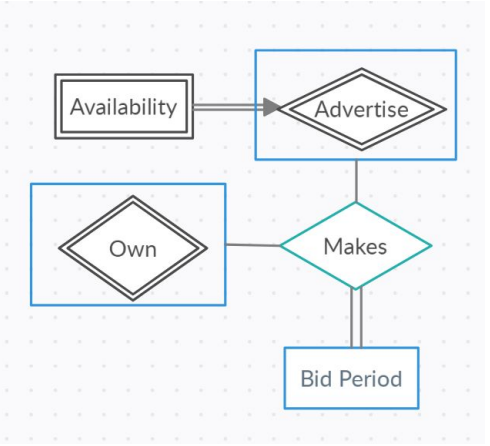
This table is enforcing 2 constraints which are

1) A pet owner can only make a bid for a caretaker for the pet that the pet_owner owns
2) A pet_owner can only make a bid for a caretaker that is available for the period

Due to the 1st constraint, **Own** relation has to be evaluated to an aggregated entity to have a relation with **bid_period**.

As part of the 2nd constraint, the **advertise** relation has to be evaluated to an aggregated entity to have a relation with **bid_period**.

```
CREATE TABLE makes (
  pet_owner_username VARCHAR,
  pet_name VARCHAR,
  bid_start_period DATE,
  bid_end_period DATE,
  caretaker_username VARCHAR,
  availability_start_date DATE,
  bid_price INTEGER NOT NULL,
  is_successful BOOLEAN DEFAULT FALSE,
  payment_method VARCHAR,
  transfer_method VARCHAR,
  rating INTEGER DEFAULT NULL,
  review VARCHAR DEFAULT NULL,
  FOREIGN KEY (pet_owner_username, pet_name)
    REFERENCES owned_pets(username, pet_name),
  FOREIGN KEY (bid_start_period, bid_end_period)
    REFERENCES bid_period(bid_start_period, bid_end_period),
  FOREIGN KEY (caretaker_username, availability_start_date)
    REFERENCES advertise_availabilities(username,
        availability_start_date),
  PRIMARY KEY (pet_owner_username, pet_name, bid_start_period,
    bid_end_period, caretaker_username, availability_start_date)
);
```



Discussion on database normal forms

To prevent naming conflicts in the attribute names, we may add the following prefixes to the relevant attributes.

- “u” - Refers to a user which is either a pet owner or a caretaker.
- “p” - Refers to a pet owner.
- “c” - Refers to a caretaker.
- “a” - Refers to a pcs administrator.

The following is the set of functional dependencies for our application based on the data constraints that we have described in the previous section.

```
{
 u_username -> u_name, u_email, u_password, u_is_deleted;

 u_email -> u_username, u_name, u_password, u_is_deleted;

 a_username -> a_password;

 c_username -> verified_date;

 p_username, pet_name -> special_requirements, pet_category_name;

 pet_category_name -> base_daily_price;

 c_username, availability_start_date -> availability_end_date;

 c_username, availability_start_date, pet_category_name -> daily_price;

 p_username, pet_name, c_username, availability_start_date,
 bid_start_period, bid_end_period -> bid_price, is_successful,
 payment_method, transfer_method, rating, review;

 c_username, leave_start_date, leave_end_date -> is_approved;
}
```

Next, we analyse the set of functional dependencies that hold for each SQL table and justify whether that table is in BCNF or 3NF. For brevity, we exclude tables in which only trivial functional dependencies hold.

TABLE pcs_admins
Set of non-trivial FDs that hold: {`a_username -> a_password`}
Note that <`a_username`> is a superkey in the pcs_admins table as it uniquely identifies a table row. Thus, **pcs_admins** table is in BCNF.

TABLE pcs_user
Set of non-trivial FDs that hold: {`u_username -> u_name, u_email, u_password, u_is_deleted; u_email -> u_username, u_name, u_password, u_is_deleted`}
Note that <`u_username`> and <`u_email`> are both superkeys in the **pcs_user** table as they both uniquely identify a table row. Thus, **pcs_user** table is in BCNF.

TABLE pet_categories
Set of non-trivial FDs that hold: `{pet_category_name -> set_by, base_price, is_deleted}`
Note that `<pet_category_name>` is a superkey in the **pet_categories** table as it uniquely identifies a table row. Thus, **pet_categories** table is in BCNF.

TABLE owned_pets
Set of non-trivial FDs that hold: `{p_username -> pet_name, special_requirements, pet_category_name, is_deleted}`
Note that `<p_username>` is a superkey in the **owned_pets** table as it uniquely identifies a table row. Thus, **owned_pets** table is in BCNF.

TABLE verified_caretakers
Set of non-trivial FDs that hold: `{c_username -> a_username, verified_date}`
Since `<c_username>` is a superkey in the **verified_caretakers** table as it uniquely identifies a table row, **verified_caretakers** table is in BCNF.

TABLE approved_apply_leaves
Set of non-trivial FDs that hold: `{c_username, leave_start_date, leave_end_date -> a_username, is_approved}`
Since `<c_username, leave_start_date, leave_end_date>` is a superkey in the **approved_apply_leaves** table, hence the **approved_apply_leaves** table is in BCNF.

TABLE advertise_availabilities
Set of non-trivial FDs that hold: `{c_username, availability_start_date -> availability_end_date, is_deleted}`
Since `<c_username, availability_start_date>` is a superkey in the **advertise_availabilities** table, hence the **advertise_availabilities** table is in BCNF.

TABLE advertise_for_pet_categories
Set of non-trivial FDs that hold: `{c_username, availability_start_date, pet_category_name -> daily_price}`
Since `<c_username, availability_start_date, pet_category_name>` is a superkey in the **advertise_for_pet_categories** table, hence the **advertise_for_pet_categories** table is in BCNF.

TABLE makes
Set of non-trivial FDs that hold: `{p_username, pet_name, bid_start_period, bid_end_period, c_username, availability_start_date -> is_successful, payment_mtd, transfer_mtd, rating, review}`
Since `<p_username, pet_name, bid_start_period, bid_end_period, c_username, availability_start_date>` is a superkey in the **makes** table, hence the **makes** table is in BCNF.

Conclusion

The BCNF decompositions are dependency-preserving. Unioning all the FDs that hold on the tables, we have:

```
(F[pcs_admins] U F[pcs_user] U ... U F[makes]) =
{
  a_username -> a_password;

  u_username -> u_name, u_email, u_password, u_is_deleted;

  u_email -> u_username, u_name, u_password, u_is_deleted;

  c_username -> a_username, verified_date;

  c_username, availability_start_date -> availability_end_date,is_deleted;

  c_username, leave_start_date, leave_end_date -> a_username, is_approved;

  c_username, availability_start_date, pet_category_name -> daily_price;

  p_username, pet_name, bid_start_period, bid_end_period, c_username,
  availability_start_date -> bid_price, is_successful, payment_method,
  transfer_method, rating, review;

  pet_category_name -> set_by, base_price, is_deleted;

  p_username, pet_name -> special_requirements, pet_category_name,
  is_deleted;

}
```

Note that some of the original set of FDs can be obtained trivially from the tables:
{a_username -> a_password, u_username -> u_name, u_email, u_password, u_is_deleted; u_email -> u_username, u_name, u_password, u_is_deleted; c_username, availability_start_date, pet_category_name -> daily_price; p_username, pet_name, bid_start_period, bid_end_period, c_username, availability_start_date -> bid_price, is_successful, payment_method, transfer_method, rating, review}.

The other set of FDs can be obtained by performing Armstrong's Axioms:

1) c_username -> a_username, verified_date [Given from table]
   c_username -> verified_date [Decomposition on (1)]

2) p_username, pet_name -> special_requirements, pet_category_name, is_deleted [Given from table]
   p_username, pet_name -> special_requirements [Decomposition on (1)]
   p_username, pet_name -> pet_category_name [Decomposition on (1)]
   p_username, pet_name -> special_requirements, pet_category_name [Union (2) and (3)]

3) `c_username, availability_start_date -> availability_end_date, is_deleted [Given from table]`
   `c_username, availability_start_date -> availability_end_date` `[Decomposition on (1)]`

4) `c_username, leave_start_date, leave_end_date -> a_username, is_approved [Given from table]`
   `c_username, leave_start_date, leave_end_date -> is_approved` `[Decomposition on (1)]`

Hence, `(F[pcs_admins] U F[pcs_user] U ... U F[makes])` ≡ `F`.

SQL Triggers

Note: For each trigger, there will be a brief description of the trigger and what constraints that trigger aims to enforce. After that, the actual SQL code for the trigger will be shown.

makes_insert_trigger is a trigger that enforces the constraint that a full_time_caretaker will immediately accept any bids made as long as the number of pets that the caretaker is attached to is less than 5. In addition, it also checks if the bid price has to be at least higher than the base price set by the admin for the category.

```
CREATE OR REPLACE FUNCTION check_make_bid() RETURNS TRIGGER AS
  $$
  BEGIN
    -- check if caretaker is full time
    -- then if the price is at least above the base price
    -- and num pets caring during that period < 5
    IF (EXISTS (
      SELECT 1
      FROM full_time_caretakers f
      WHERE f.username = NEW.caretaker_username
    ) AND (
      SELECT COUNT(*)
      FROM makes m
      WHERE m.caretaker_username = NEW.caretaker_username AND is_successful = TRUE
        AND m.bid_start_period <= NEW.bid_end_period AND m.bid_end_period <= NEW.bid_start_period) < 5)
      AND NOT EXISTS (
         SELECT 1
        FROM advertise_for_pet_categories NATURAL JOIN pet_categories
          WHERE NEW.bid_price < daily_price AND NEW.bid_price < base_price
    ) THEN NEW.is_successful = TRUE;
    END IF;
    RETURN NEW;

  END;
  $$
LANGUAGE plpgsql;

CREATE TRIGGER makes_insert_trigger BEFORE INSERT ON makes
FOR EACH ROW EXECUTE PROCEDURE check_make_bid();
```

makes_choose_bid_trigger is a trigger that will only be used by part_time_caretaker as they are allowed to choose the bid to accept. In addition, this trigger will also enforce the following constraints,

1) A part_time_caretaker can only care for at most 5 pets for the same period if the caretaker's rating is at least 4.
2) If a part_time_caretaker has a rating less than 4, the part_time_caretaker can only have a most 2 pets for the same period.

```
CREATE OR REPLACE FUNCTION check_choose_bid() RETURNS TRIGGER AS
  $$
  DECLARE rating NUMERIC;
  DECLARE cur_job NUMERIC;
  BEGIN
    SELECT AVG(rating)
      FROM makes m
      WHERE m.caretaker_username = NEW.caretaker_username into rating;

    SELECT COUNT(*)
      FROM makes m
      WHERE m.caretaker_username = NEW.caretaker_username AND is_successful = TRUE
        AND m.bid_start_period <= NEW.bid_end_period
        AND m.bid_end_period <= NEW.bid_start_period into cur_job;

    IF ( rating >= 4)
        THEN IF ( cur_job < 5) THEN RETURN NEW;
      END IF;
    ELSE
      IF (cur_job < 2) THEN RETURN NEW;
      END IF;
    END IF;
    RETURN NULL;
  END;
  $$
LANGUAGE plpgsql;

CREATE TRIGGER makes_choose_bid_update_trigger BEFORE UPDATE OF is_successful ON makes
FOR EACH ROW EXECUTE PROCEDURE check_choose_bid();
```

leave_insert_trigger is a trigger that enforces that the full_time_caretakers must work minimally of 2 x 150 consecutive days a year. Our group has decided to enforce this by allowing the full_time_caretaker to take leave if either conditions has been met.

1) care_taker has already worked 2 x 150 consecutive days
2) care_taker still can meet the minimum criteria of 2 x 150 consecutive days and not have any jobs during the period of leave.

```
CREATE OR REPLACE FUNCTION check_leave() RETURNS TRIGGER AS
  $$
  DECLARE curYear VARCHAR;
  DECLARE accPeriod NUMERIC;
  DECLARE lastDay DATE;
  BEGIN
    SELECT date_part('year', CURRENT_DATE) into curYear;
    SELECT (date_trunc('MONTH', CURRENT_DATE) + INTERVAL '2 MONTH - 1 day') into lastDay;
    SELECT COUNT(*)
        FROM (
            SELECT COALESCE(availability_end_date,CURRENT_DATE) - availability_start_date as wDay
            FROM advertise_availabilities
            WHERE username = 'micky') as wDayTable
        WHERE wDay >= 150 into accPeriod;

    -- Check if the FT has worked 2 x 150
    if (accPeriod >= 2) THEN
        RETURN NEW;
    ELSE
      -- Check if the FT still has the possibility of working 2 x 150 if leave is made
      IF ( accperiod + ((lastday - new.leave_end_date) % 150) >= 2) THEN
        -- Now we check if the FT has any bid for that period of leave , if have cannot apply leave
        IF NOT EXISTS (
            SELECT 1
            FROM makes
            WHERE is_successful = TRUE
              AND caretaker_username = NEW.username
              AND bid_start_period >= NEW.leave_start_date
              AND bid_end_period <= NEW.leave_end_date
        ) THEN RETURN NEW;
        END IF;
      END IF;
    END IF;
    RETURN NULL;
  END;
  $$
LANGUAGE plpgsql;

CREATE TRIGGER leave_insert_trigger BEFORE INSERT ON apply_leaves
FOR EACH ROW EXECUTE PROCEDURE check_leave();
```

SQL Complex Queries

Gets all underperforming full-time caretakers. A full-time caretaker is said to be underperforming if he/she has an average rating of less than 2 for the past 90 days or he/she has not taken up any caretaking jobs for the past 90 days.

```
-- Admin: Get all underperforming full-time caretaker
(
  SELECT username
  FROM full_time_caretakers
  EXCEPT
  SELECT caretaker_username
  FROM makes
  WHERE is_successful = TRUE AND CURRENT_DATE - bid_start_period <= 90
)
UNION
SELECT m.caretaker_username
FROM makes m INNER JOIN full_time_caretakers f
  ON m.caretaker_username = f.username
    AND m.is_successful = TRUE
    AND CURRENT_DATE - m.bid_start_period <= 60
GROUP BY m.caretaker_username
HAVING AVG(m.rating) < 2
```

Finding caretakers that have not advertised for the most lucrative pet category of a month so that admin can recommend them to advertise for that pet category.

```
WITH
  ctx AS (SELECT pet_category_name , sum(bid_price) AS total
  FROM makes m INNER JOIN owned_pets o
    ON m.pet_name = o.pet_name
  where EXTRACT(MONTH FROM bid_start_period) = $1
    AND is_successful = TRUE
  GROUP BY o.pet_category_name)

  SELECT vc.username
  FROM verified_caretakers vc
  WHERE NOT EXISTS(
    SELECT 1
    FROM advertise_for_pet_categories ap
      NATURAL JOIN advertise_availabilities aa
    WHERE ap.username = vc.username
      AND ap.pet_category_name IN (
      SELECT a.pet_category_name
      FROM ctx a , ctx b
      WHERE a.total >= b.total
        AND a.pet_category_name <> b.pet_category_name
      )
    GROUP BY username
    HAVING EXTRACT(MONTH FROM max(availability_start_date)) = $1);
```

Gets total number of pet-days for the current month for a given caretaker.

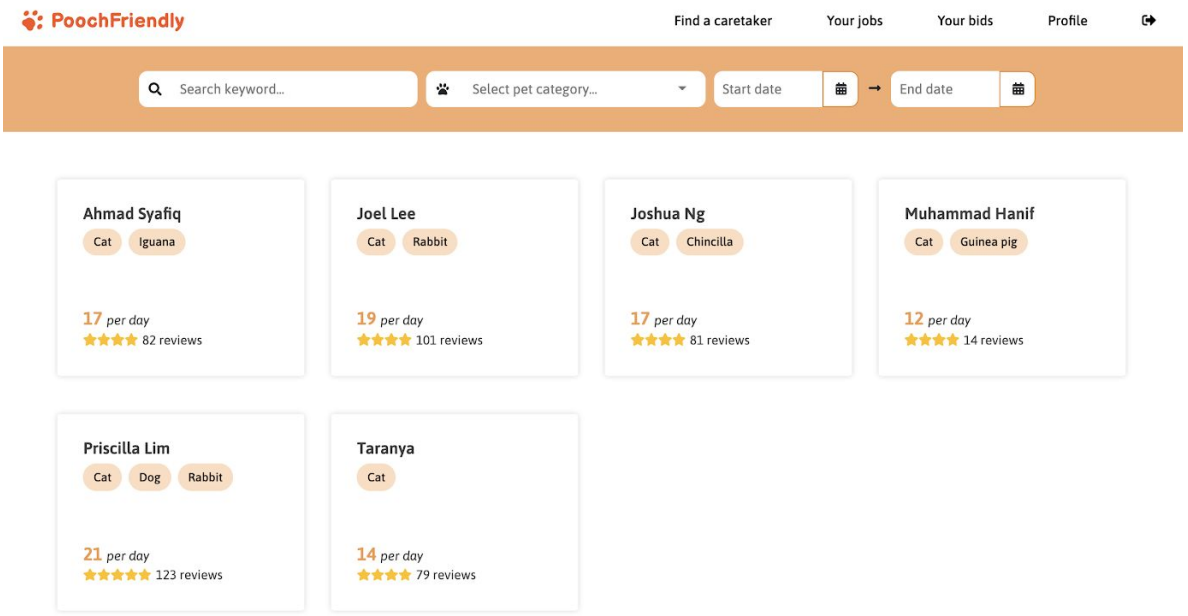
```sql
-- Get total number of pet-days for the current month for a given caretaker
SELECT COALESCE(
  (
    SELECT SUM(m.bid_end_period - m.bid_start_period + 1)
    FROM makes m
    WHERE m.caretaker_username = username AND m.is_successful = TRUE
      AND EXTRACT(MONTH FROM CURRENT_DATE) = EXTRACT(MONTH FROM m.bid_start_period)
    GROUP BY m.caretaker_username
  ), 0) AS num_pet_day
```

Tools and Frameworks used

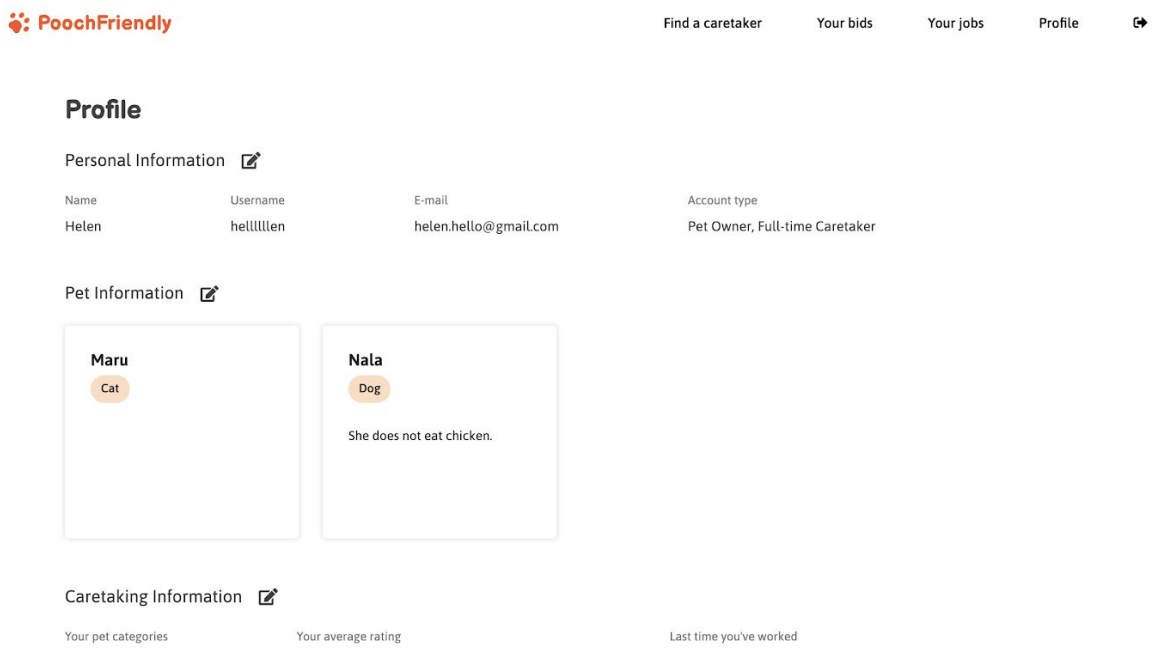
- Typescript is used for both Frontend and Backend
- AngularJS for the Frontend
- NodeJS and ExpressJS for the Backend
- Swagger is used for documenting Backend API
- Postgresql for the Database
- Heroku for hosting the live website and the Database

App Screenshots

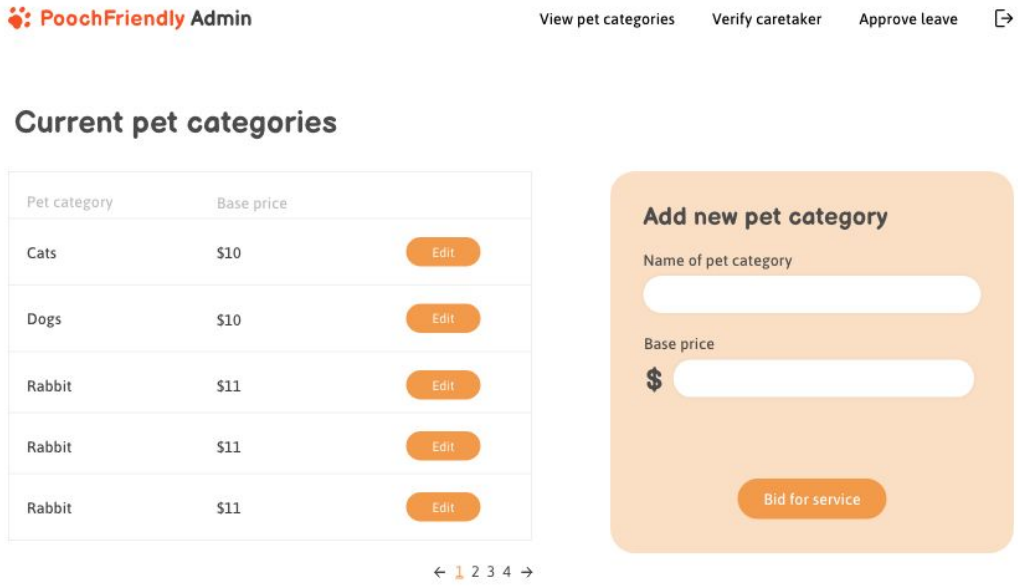
Find a caretaker page



Profile page



Admin page



Difficulties and Lessons Learnt

- Hard to enforce constraints and maintain data-integrity even for a simple pet-caring application.
- There is no “perfect” ER Diagram and SQL Schema as this all depends on the business and systems requirements and which requirements get prioritized.
- Deleting a row from a table is non-trivial as there is a need to consider the state of other tables that are referencing that row.
- Having a good API documentation tool like Swagger is a great time-saver and it has a good UI that helps the frontend developers to know which API to call.