# CS2102 Project Report

Ge Shuming A0182992N

Goh Yee Loon A0188036Y

Guo Zili A0183440L

Lee Wei Min A0201446R

Wu Xinghao, Alex A0180330W

## Project Responsibilities

A listing of the project responsibilities of each team member.

- Ge Shuming
  - Repo creation
  - Organization of code and setup of express web framework
  - Writing triggers and queries for multiple tables
  - Report writeup
- Goh Yee Loon
  - Base prices
  - Bids
  - Report writeup
- Guo Zili
  - Setting up of web framework.
  - User registration and login frontend and backend.
  - Report writeup
- Lee Wei Min
  - Caretaker summary information - fullstack
  - Refinement of schema
  - Triggers (covering constraint, business logic)
  - Report writeup
- Wu Xinghao, Alex
  - Pets
  - Dashboard
  - Report writeup

## Application data requirements and functionalities

PetCareService is an application that allows pet owners to find caretakers to take care of their pets for a period of time. Pet Owners can make bids for days that a caretaker is available for, and the caretaker can accept the bid.

### General Account Information

1. An account is a PCS Admin or a User
   - Covering constraint on ISA hierarchy
2. An account must consist of username and password
3. A User is either a Pet Owner, or Care Taker, or both
   - Covering + Overlap constraint on ISA hierarchy
4. A User must consist of a name and profile

### Pet

1. A Pet must consist of a name, a profile and a category
2. A Pet may have special requirements
3. A Pet must be owned by a Pet Owner

    - Identity dependency
    - Name as partial key
        - It is probably realistic to assume that a pet owner will not use the same name for two different pets
4. A Pet Owner can own multiple Pets

## Care Taker Availability

1. An Availability must consist of a start date, an end date, the category of the pet that the Care Taker can care for, and the daily price
2. Care Takers can create and advertise their Availability
    - Identity dependency
    - Start date, end date and category as partial keys
3. Availabilities with the same category should not have overlapping start/end dates
    - Cannot be reflected on ER diagram

## Bid

1. A Pet Owner can bid for a Care Taker's Availability for their Pet
2. A Bid consists of a date (single day) that falls within that of the Care Taker's Availability
    - i.e. Pet owner cannot choose a date the Care Taker is unavailable for
3. A Bid is only valid if the category of the Pet matches the category of the Availability
    - Cannot be reflected on ER diagram
4. A Care Taker can choose to accept or reject a Bid. The Bid accepted by the Care Taker will be known as a successful Bid

## Review and Rating

1. A Pet Owner can post a review on a Care Taker based on a successful bid
2. A Pet Owner can give a rating on a Care Taker based on a successful bid

## Care Taker Working Status

1. A Care Taker is either a Full Timer, or Part Timer
    - Covering constraint on ISA hierarchy
2. A Full Timer can choose to specify the date they wish to take Leave
    - Identity dependency
    - Date as partial key
3. A Full Timer cannot create an Availability on days covered by their Leave

## Care Taker Pet Limit

1. A Care Taker cannot take care of more than 5 Pets on any single day
    - For any bid, the database can check if there are 5 or more successful Bids active on that day
        - If there are 5 or more successful Bids, the Care Taker will not be able to accept the bid. Else, the Care Taker can accept the bid
    - Cannot be reflected on ER diagram

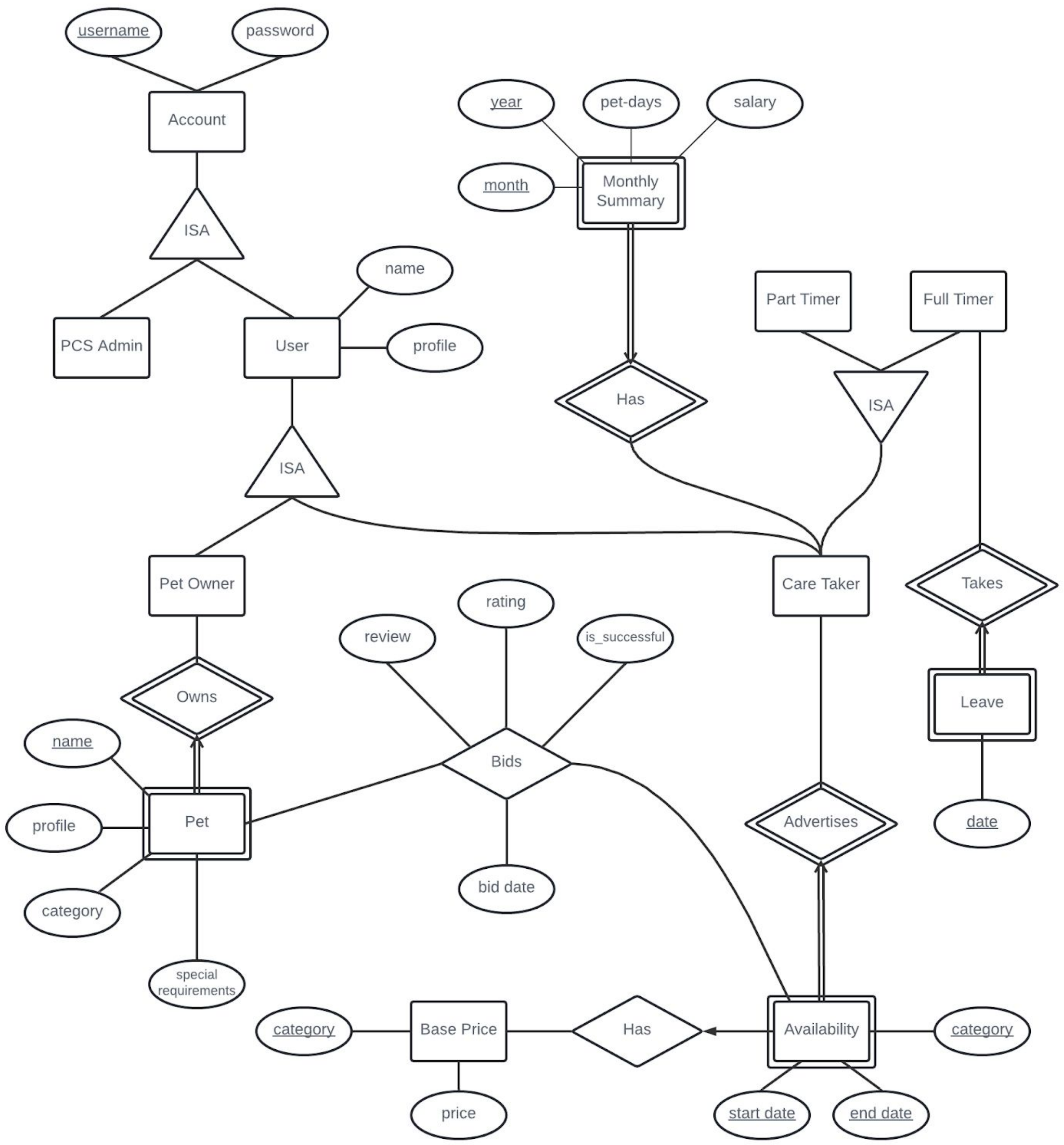
## Monthly Summary

1. A Monthly Summary consists of year, month, pet-days and salary
2. Every Care Taker has a Monthly Summary for each Month they have worked
    - Identity dependency
    - Year and Month as partial key
3. Each successful bid adds one pet day to the monthly summary on the month that the bid takes place in
4. Each successful bid adds the price of the category of the pet to the monthly summary on the month that the bid takes place in

## Base Price

1. The Base Price of different categories of Pets should be recorded in a list, and can be modified by PCS Admin at any time
    - No way to show only PCS Admin can modify the table

# ER Model



The following are the application's constraints that are unable to be captured by our ER model:

- Availabilities within the same pet category should not have overlapping start/end dates
- A Bid is only valid if the category of the Pet matches the category of the Availability
- A Full Timer cannot create an Availability on days covered by their Leave
- A Caretaker cannot take care of more than 5 Pets on any single day
- Calculation of salary and pet days for monthly summary
- Only PCS Admins are able to modify the Base Price table

## Relational Schema

The following code block shows the relational schema derived from our ER model:

```
--- Accounts, covering constraint on pcs_admins and users

CREATE TABLE IF NOT EXISTS pcs_admins (
  username VARCHAR(200) PRIMARY KEY,
  password VARCHAR(200) NOT NULL
);

CREATE TABLE IF NOT EXISTS users (
  username VARCHAR(200) PRIMARY KEY,
  password VARCHAR(200) NOT NULL,
  name VARCHAR(200),
  profile VARCHAR(200)
);

CREATE VIEW accounts AS
  SELECT username, password FROM pcs_admins
  UNION
  SELECT username, password FROM users;

--- Users, covering and overlapping constraints

CREATE TABLE IF NOT EXISTS pet_owners (
  username VARCHAR(200) PRIMARY KEY REFERENCES users(username) ON DELETE
CASCADE
);

CREATE TABLE IF NOT EXISTS care_takers (
  username VARCHAR(200) PRIMARY KEY REFERENCES users(username) ON DELETE
CASCADE
);

CREATE TABLE IF NOT EXISTS full_timers (
  username VARCHAR(200) PRIMARY KEY REFERENCES care_takers(username) ON DELETE
CASCADE
);

CREATE TABLE IF NOT EXISTS part_timers(
  username VARCHAR(200) PRIMARY KEY REFERENCES care_takers(username) ON DELETE
CASCADE
```

```
);

--- Base Prices

CREATE TABLE IF NOT EXISTS base_prices (
  category VARCHAR(200) PRIMARY KEY,
  price INT NOT NULL
);

--- Pets

CREATE TABLE IF NOT EXISTS pets (
  poname VARCHAR(200) REFERENCES pet_owners(username) ON DELETE CASCADE,
  pname VARCHAR(200),
  profile VARCHAR(200),
  category varchar(200) REFERENCES base_prices(category) NOT NULL,
  special_requirements VARCHAR(200),
  PRIMARY KEY (poname, pname)
);

--- Availabilities

CREATE TABLE IF NOT EXISTS availabilities (
  username VARCHAR(200) REFERENCES care_takers(username) ON DELETE CASCADE,
  start_date DATE,
  end_date DATE CHECK (start_date <= end_date),
  category VARCHAR(200) REFERENCES base_prices(category),
  PRIMARY KEY(username, start_date, end_date, category)
);

--- Bids

CREATE TABLE IF NOT EXISTS bids (
  start_date DATE,
  end_date DATE,
  category VARCHAR(200) REFERENCES base_prices(category),
  poname VARCHAR(200),
  pname VARCHAR(200),
  ctuname VARCHAR(200),
  bid_date DATE,
  review VARCHAR,
  rating INT CHECK( (rating IS NULL) OR (rating >= 0 AND rating <= 5) ),
  is_successful BOOLEAN DEFAULT FALSE,

  FOREIGN KEY (poname, pname) REFERENCES pets (poname, pname),
  FOREIGN KEY (ctuname, start_date, end_date, category) REFERENCES
availabilities(username, start_date, end_date, category),

  PRIMARY KEY (
    start_date,
    end_date,
    category,
    poname,
    pname,
    ctuname,
    bid_date
  ),

  CHECK (poname <> ctuname),
  CHECK ( (start_date <= bid_date) AND (bid_date <= end_date) )
);

--- Monthly Summary

CREATE TABLE IF NOT EXISTS monthly_summary (
  ctname varchar(200) REFERENCES care_takers(username) ON DELETE CASCADE,
  year INT CHECK(year >= 0),
  month INT CHECK( month >= 1 AND month <= 12 ),
  pet_days INT CHECK(pet_days >= 0),
  salary INT CHECK(salary >= 0),
  PRIMARY KEY(ctname, year, month)
);

--- Leaves

CREATE TABLE IF NOT EXISTS leaves (
  username varchar(200) REFERENCES full_timers(username) ON DELETE CASCADE,
  leave_date DATE,
  PRIMARY KEY (username, leave_date)
);
```

The following are constraints unable to be enforced using our relational schema and triggers are used instead:

- Enforcing the covering constraint on the caretakers table
- Limiting the number of pets that a caretaker can care for in a day

- Ensuring that every successful bid is accounted for in the monthly summary

## Discussion of 3NF/BCNF

The following table lists all the relational tables and explains why each table is in 3NF or BCNF.

| Table | BCNF/3NF | FDs | Reason |
|---|---|---|---|
| pcs_admin | BCNF | {username -> password} | 2 attributes |
| users | BCNF | {username -> password, name, profile} | Username is superkey |
| pet_owners | BCNF | | Single attribute |
| full_timers | BCNF | | Single attribute |
| part_timers | BCNF | | Single attribute |
| base_prices | BCNF | {category -> price} | 2 attributes |
| pets | BCNF | {poname, pname -> profile, category, special_requirements} | (poname, pname) is superkey |
| availabilities | BCNF | {(username, start_date, end_date, category) -> daily_price} | (username, start_date, end_date, category) is superkey |
| bids | 3NF | {poname, pname -> categories;<br><br>Start_date,end_date,category,poname, pname,ctuname, bid_date -> start_date,end_date, review, rating, is_successful} | (poname, pname) is not superkey, but categories is a prime attribute of the table. |
| monthly_summary | BCNF | {ctname, year, month -> pet_days, salary} | (ctname, year, month) is superkey |
| leaves | BCNF | | 2 attributes |

Since all tables are in at least 3NF form, database is in 3NF.

# Most interesting triggers

## Enforcing the Covering Constraint on Caretakers Table

The trigger check_pcs_admin checks whether a row exists in the users table with the same username as the row being inserted. If so, it raises an exception.

```
CREATE FUNCTION check_not_user() RETURNS TRIGGER LANGUAGE PLPGSQL AS $$
DECLARE ctx NUMERIC;
BEGIN
SELECT COUNT(*) INTO ctx
FROM users U
WHERE NEW.username = U.username;
IF ctx > 0 THEN RAISE EXCEPTION 'pcs_admin is also a user';
ELSE RETURN NEW;
END IF;
END;
$$;
CREATE FUNCTION check_not_pcs_admin() RETURNS TRIGGER LANGUAGE PLPGSQL AS $$
DECLARE ctx NUMERIC;
BEGIN
SELECT COUNT(*) INTO ctx
FROM pcs_admins P
WHERE NEW.username = P.username;
IF ctx > 0 THEN RAISE EXCEPTION 'user is also a pcs_admin';
ELSE RETURN NEW;
END IF;
END;
$$;
CREATE TRIGGER check_pcs_admin BEFORE
INSERT
   OR
UPDATE ON pcs_admins FOR EACH ROW EXECUTE PROCEDURE check_not_user();
CREATE TRIGGER check_user BEFORE
INSERT
   OR
UPDATE ON users FOR EACH ROW EXECUTE PROCEDURE check_not_pcs_admin();
CREATE VIEW accounts AS
SELECT username,
   password
FROM pcs_admins
UNION
SELECT username,
   password
FROM users;
```

## Enforcing business logic - Limit on Number of Pets Care Taker can Take Care in a Day

The trigger check_bid checks whether there are already 5 rows in the bids table which are successful, bidded for that particular caretaker and is on the same day as the new bid. If so, it raises an exception.

```
CREATE FUNCTION check_caretaker_pet_limit() RETURNS TRIGGER LANGUAGE PLPGSQL
AS $$
DECLARE ctx NUMERIC;
BEGIN
SELECT COUNT(*) INTO ctx
FROM bids B
WHERE NEW.ctuname = B.ctuname
   AND NEW.bid_date = B.bid_date
   AND B.is_successful = TRUE;
IF ctx = 5 THEN RAISE EXCEPTION 'caretaker can only take care of up to 5 pets
in a day';
ELSE RETURN NEW;
END IF;
END;
$$;
CREATE TRIGGER check_bid BEFORE
INSERT
OR
UPDATE ON bids FOR EACH ROW EXECUTE PROCEDURE check_caretaker_pet_limit();
SELECT *
FROM bids B
WHERE NEW.ctuname = B.ctuname
   AND NEW.bid_date = B.bid_date
   AND B.is_successful = TRUE;
```

## Adding a successful bid to the monthly summary

In order to calculate the salary and pet days for the monthly summary of a care taker, we have to create the following trigger that will be executed when the care taker accepts a bid. The trigger simply creates or updates the corresponding monthly summary of the care taker for the month and year that the bid occurs on.

```
CREATE FUNCTION add_successful_bid_to_monthly_summary()
RETURNS TRIGGER LANGUAGE PLPGSQL AS $$
  DECLARE salaryToAdd INT;
  BEGIN
    SELECT price INTO salaryToAdd
    FROM base_prices WHERE category = NEW.category;

    IF EXISTS (SELECT * from monthly_summary
                 WHERE ctname = NEW.ctuname
                   AND year = EXTRACT(YEAR FROM NEW.bid_date)
                   AND month = EXTRACT(MONTH FROM NEW.bid_date) + 1
              )
    THEN UPDATE monthly_summary
      SET pet_days = pet_days + 1, salary = salary + salaryToAdd
      WHERE ctname = NEW.ctuname
        AND year = EXTRACT(YEAR FROM NEW.bid_date)
        AND month = EXTRACT(MONTH FROM NEW.bid_date) + 1;

    ELSE INSERT INTO monthly_summary
      VALUES (NEW.ctuname,
              EXTRACT(YEAR FROM NEW.bid_date),
              EXTRACT(MONTH FROM NEW.bid_date) + 1,
              1,
              salaryToAdd);
    END IF;
  END;
$$;

CREATE TRIGGER add_successful_bid_to_monthly_summary
  AFTER UPDATE ON bids
  FOR EACH ROW
  WHEN (NEW.is_successful = TRUE)
EXECUTE PROCEDURE add_successful_bid_to_monthly_summary();
```

# Most interesting queries

The following section describes three of the most complex queries implemented in our application.

## Getting the account type

The following query is used to determine the type of the account for conditional rendering of the express view. There are currently 2 types of account: admin, and user. The account can either be an admin or a user. The user can either be a pet owner or care taker or both.

```
pool.query(`
      SELECT COALESCE(
        (SELECT 'admin' FROM pcs_admins WHERE username = $1::text),
        (SELECT 'both'
           FROM (
             SELECT * FROM pet_owners INTERSECT SELECT * FROM care_takers
           ) result WHERE username = $1::text
        ),
        (SELECT 'pet_owner' FROM pet_owners WHERE username = $1::text),
        (SELECT 'care_taker' FROM care_takers WHERE username = $1::text)
      );
    `,
    [username]
  )
```

## Caretaker Bid Selection

Caretaker selects all bids that have a date greater than today and ordered in asc order.

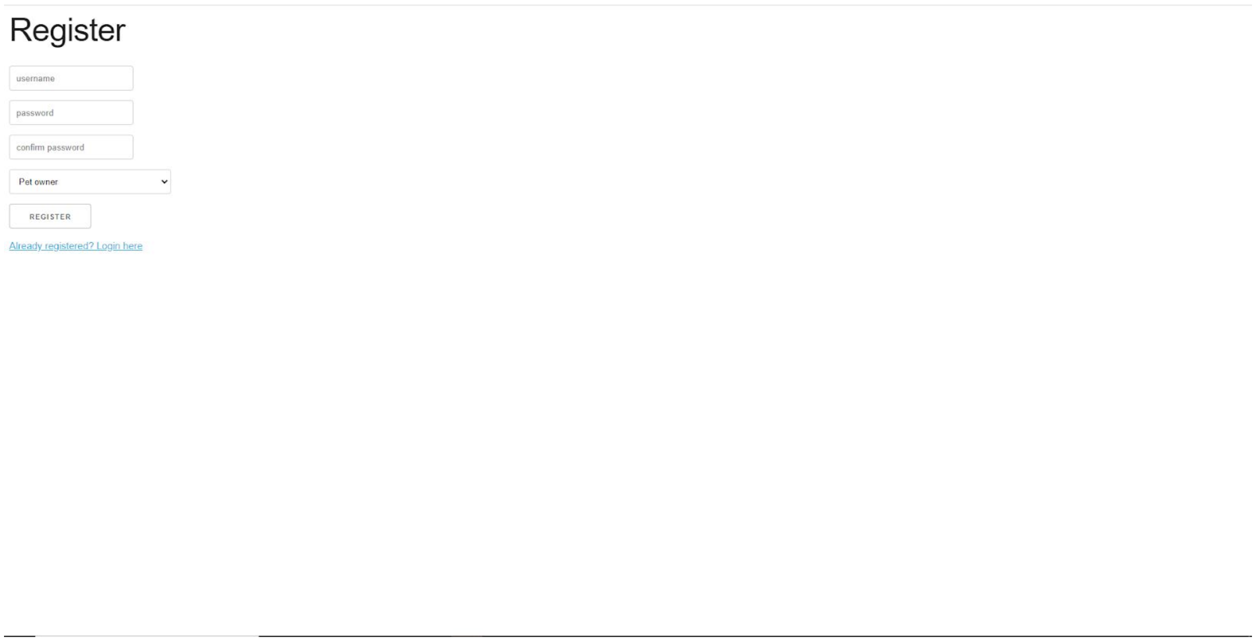
```
const query = `
 SELECT *
 FROM bids
 WHERE ctuname = $1
 AND start_date >= CURRENT_DATE
 ORDER BY
 start_date ASC
 `;
const values = [username];
```

## Find Average Rating of Bids

Get average rating of bids grouped by months.

```
SELECT AVERAGE(rating),
       TO_CHAR(Timestamp bid_date, ‘Month’),
       EXTRACT(YEAR FROM bid_date)
FROM bids
WHERE ctuname = $1::text
GROUP BY TO_CHAR(Timestamp bid_date, ‘Month’), EXTRACT(YEAR FROM bid_date)
ORDER BY bid_date ASC;
```

## Screenshots



Register

username

password

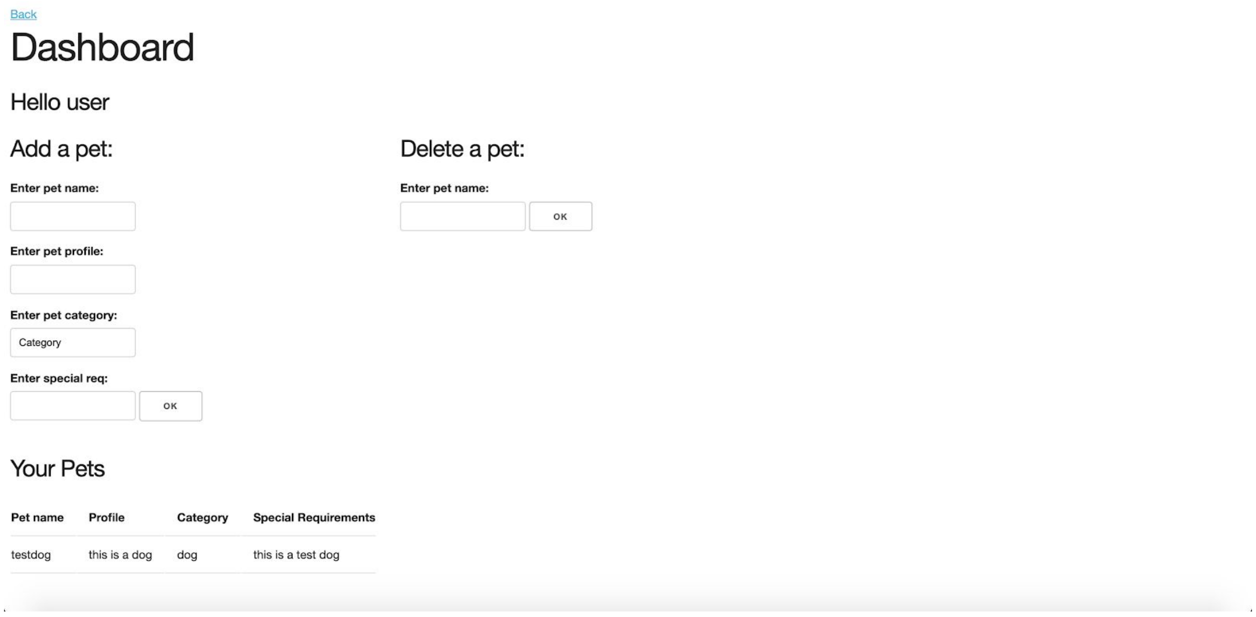
confirm password

Pet owner

REGISTER

Already registered? Login here

Fig 1: Screenshot of Register page



Back

Dashboard

Hello user

Add a pet:

Enter pet name:

Enter pet profile:

Enter pet category:

Category

Enter special req:

OK

Delete a pet:

Enter pet name:

OK

Your Pets

| Pet name | Profile | Category | Special Requirements |
|---|---|---|---|
| testdog | this is a dog | dog | this is a test dog |

Fig 2: Screenshot of user dashboard with functions

## Software tools/Frameworks used

1. Backend Runtime Environment: Node.js
2. Database: Postgres 13
3. Frontend: Ejs
4. Web Framework: Express

## Learning points

- It was very interesting to learn to enforce many constraints on the database side. Usually, most of the constraints are enforced on the backend side because it was simply easier and more efficient to not write any SQL code
- Making a web application with a frontend and backend with a database without prior knowledge of web development is pretty challenging
- Hard to manipulate dates using sql
- Use of triggers for enforcing non-overlap constraint
- Frontend is very troublesome and time-consuming to implement