CS2102 PROJECT REPORT
Pet Caring Services Application

GROUP #61

NISHANTH ELANGO A0184373Y
DIVAKARAN HARITHA A0187915N
PHILIP ALEXANDER BOEDIMAN A0192235H
KELVIN HARRIS A0187349M
JOANNA SARA A0184557R

1. Introduction

Pet Caring Services(PCS) application is a website platform where pet owners can search for caretakers to take care of their pets for a certain period of time. A user can sign up as a full time caretaker, part time caretaker or a pet owner. This website allows for easy browsing of caretaker profiles for pet owners to choose from and guarantees a hassle-free method of bidding for suitables caretakers catered for specific animal types. It also allows caretakers to advertise their available slots easily, and ensure fast and secure transactions with pet owners.

1.1. Roles and Responsibilities

Nishanth:

- Set up user authentication with passport.js
- Worked on backend and SQL queries for retrieval of user information
- Worked on summary statistics backend for PCS administrators

Haritha:

- In charge of UI
- Designed Index, Register, Login and Browse Pages
- Worked on SQL schema and implemented triggers

Kelvin:

- Worked on backend and SQL queries for retrieval of user information
- Worked on rendering information on Browse and Profile pages

Joanna:

- Designed Profile Page for Caretakers, Petowners, and Pets
- Worked on backend and SQL queries

Philip:

- Worked on the ‘bid’ functionalities of the application and designed the UI for bid.
- Worked on backend and SQL queries.

1.2. Specification of Software and Tools

The architecture of our application follows the three-tier (clients, application server, database server) architecture of a Web-based database application with Web browsers as the clients and a Web server as the application server.

We used NodeJS to set up our application server, and used a combination of HTML, CSS (With Bootstrap) and JavaScript for the frontend. A PostgreSQL database management system is used to manage our data.

2. Database Design

2.1. Entity-Relationship Diagram

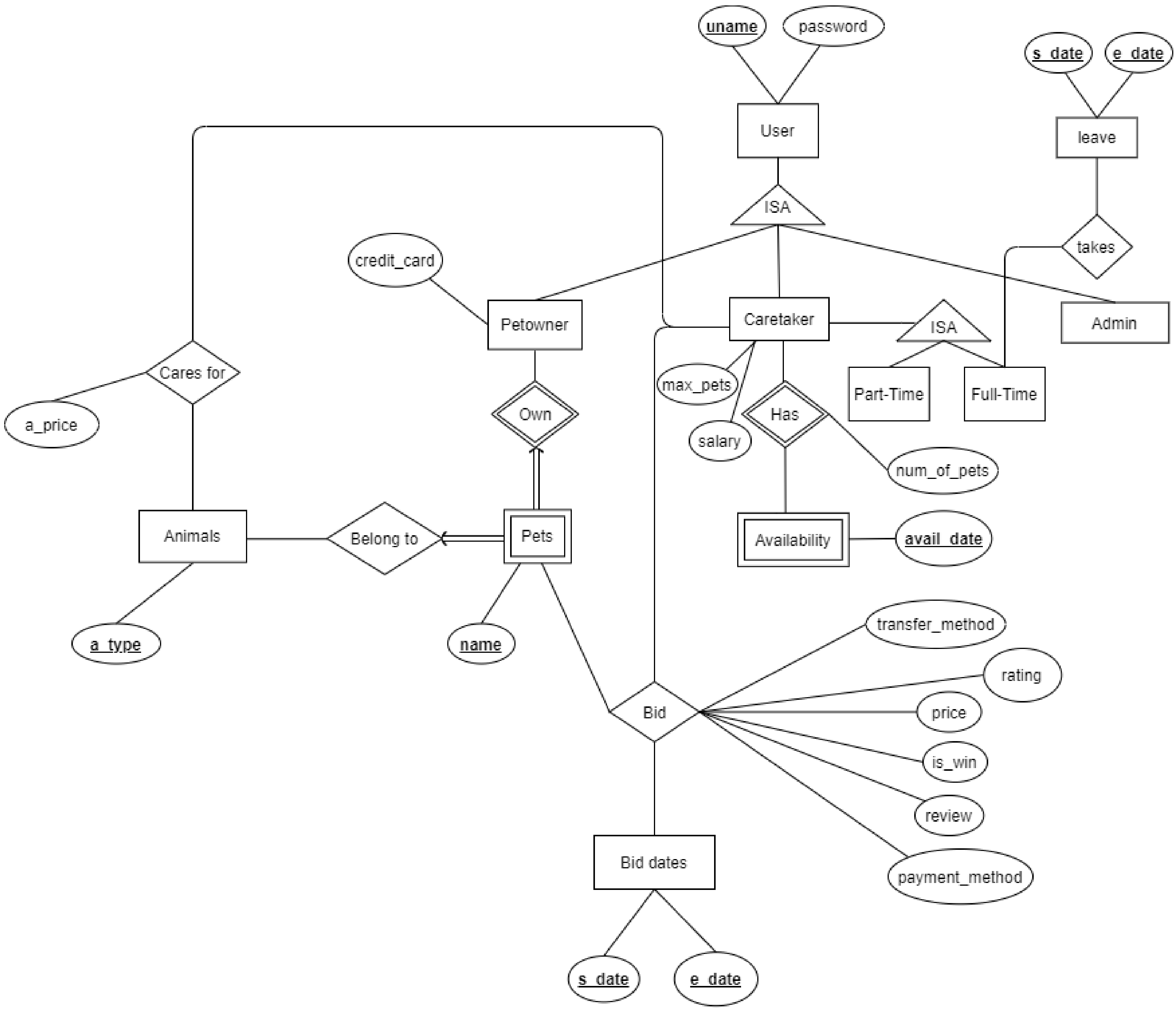


Figure 1: ER Diagram

2.1.1 Explanation for non-trivial design decisions

2.1.1.1 Bid Dates

The Bid dates entity was introduced to make s_date and e_date part of the Primary key for bid, to enforce the constraint that a pet can bid for a caretaker any number of times for different combinations of start and end dates.

2.1.1.2 Caretaker

Caretaker has an attribute max_pets to store the maximum number of pets that they are allowed to care for at any single point in time. This is to account for the differences between part time and full time caretaker. While full time caretakers have a set limit of 5 pets, part time caretakers have a limit of 2 to 5 pets depending on their rating.

2.1.1.3 Availability

Availability stores available dates indicated by the caretaker as separate entries in avail_dates. For example, if a part time caretaker indicates availability from 2020-05-04 to 2020-05-06, dates stored in avail dates will be 2020-05-04, 2020-05-05 and 2020-05-06 as separate entries. This will ensure that a pet can bid for any range of avail dates that the caretaker indicates as long as num_of_pets(pets that the caretaker is taking care of for that particular date) does not exceed max_pets. For full time caretaker, all the dates in a year will be stored in avail dates unless leave dates are specified.

2.2. Constraints

2.2.1. ER Constraints

1. There are 3 kinds of Users : Petowner, Caretaker and PCS admin.
2. Every Caretaker is either a Part-time or Full-time Caretaker.
3. Every Petowner is identified by their username.
4. Every Caretaker is identified by their username.
5. Every Pet must belong to a type of animal.
6. Every Pet is identified by their name and Petowner’s username.
7. Every Pet is owned by 1 Petowner.
8. Every Petowner can own any number of pets.
9. Every Caretaker can accept bids from a Pet any number of times for different combinations of start date and end date.
10. Pet can bid for Caretaker by indicating available bid dates as start date and end date.
11. Caretaker has/indicates availability.
12. Every Petowner whose Pet bid for Caretaker, can post a review and/or rating for each successful bid.
13. Every Caretaker can care for pets of any number of animal types. The Caretaker has a daily price for each animal type.
14. Full time Caretakers can take leave.

2.2.2. Non-ER Constraints

1. A Caretaker should not take care of pets that they cannot take care of.
2. The successful bidder is chosen by the Caretaker.
3. The total cost of caring for a Pet is the number of days times the daily price stated by the Caretaker.
4. At the end of the care period, the Petowner can post a review and/or rating for the Caretaker for that specific transaction.
5. Each full-time caretaker must work for a minimum of 2 x 150 consecutive days a year.
6. A Full-time Caretaker is treated as available until they apply for leave.
7. Full-time Caretaker cannot apply for leave if there is at least one Pet under their care.

8. A Full-time Caretaker can only take care of 5 pets at any given time.
9. For Part-time Caretakers, they should be able to indicate availability for the current year and the next year.
10. A Part-time Caretaker can only take care of 2 pets at any given time, unless they have a good rating of 4 out of 5. They cannot have more than 5 pets regardless of the rating.
11. A Full-time Caretaker will receive a salary of $3000 per month for up to 60 pet-days(Sum of all pets taken care of in a day). For any excess pet day, they will receive 80% of their price as bonus.
12. For Part-time Caretakers, PCS admin will take 25% of their price as commission.
13. Only 1 Caretaker can take care of a particular pet at any point in time.

2.3. Relational Schema

```
CREATE TABLE users(
    username varchar(64) PRIMARY KEY,
    password varchar(64) NOT NULL,
    first_name varchar(64),
    last_name varchar(64)
);

CREATE TABLE petowners(
    username varchar(64) PRIMARY KEY REFERENCES users(username),
    credit_card varchar(64)
);

CREATE TABLE caretakers(
    username varchar(64) PRIMARY KEY REFERENCES users(username),
    salary numeric DEFAULT 0,
    max_pets numeric
);


CREATE TABLE parttime(
    username varchar(64) PRIMARY KEY REFERENCES caretakers(username)
);

CREATE TABLE fulltime(
    username varchar(64) PRIMARY KEY REFERENCES caretakers(username)
);

CREATE TABLE animals(
    a_type varchar(64) PRIMARY KEY
);

CREATE TABLE pets(
    username varchar(64) REFERENCES petowners(username)
                         ON DELETE cascade,
    name varchar(64),
    a_type varchar(64) REFERENCES animals(a_type),
    PRIMARY KEY(username, name)
);

CREATE TABLE cares_for(
    ctuname varchar(64) REFERENCES caretakers(username),
    a_type varchar(64) REFERENCES animals(a_type),
    a_price numeric DEFAULT 0,
    PRIMARY KEY (ctuname, a_type)
);


CREATE TABLE admin(
    username varchar(64) PRIMARY KEY REFERENCES users(username)
);

CREATE TABLE availability(
     username varchar(64) REFERENCES users(username)
                          ON DELETE cascade,
     avail_date date,
     num_of_pets numeric DEFAULT 0 CHECK (num_of_pets <= 5 AND num_of_pets >=
0),
     PRIMARY KEY (username, avail_date)
);

CREATE TABLE bid_dates(
    s_date date,
    e_date date,
    PRIMARY KEY (s_date, e_date)
);

CREATE TABLE bids(
    pouname varchar(64),
    name varchar(64),
    ctuname varchar(64),
    price numeric NOT NULL,
    transfer_method varchar(64),
    is_win boolean DEFAULT FALSE,
    s_date date,
    e_date date,
    payment_method varchar(64),
    review varchar(256),
    rating integer CHECK ((rating IS NULL) OR (rating >= 0 AND rating <=5)),
    FOREIGN KEY(pouname, name) REFERENCES pets(username, name),
    FOREIGN KEY(ctuname) REFERENCES caretakers(username),
    FOREIGN KEY(s_date, e_date) REFERENCES bid_dates(s_date, e_date),
    PRIMARY KEY(pouname, name, ctuname, s_date, e_date)
```

```
);

CREATE TABLE takes_leave(
    ctuname varchar(64) REFERENCES fulltime(username),
    s_date date,
    e_date date,
    PRIMARY KEY(ctuname, s_date, e_date)
);
```

The ER constraints mentioned earlier have been enforced by this relational schema. The Non-ER constraints are not enforced by the relation schema, but are enforced using triggers instead.

2.4. Normal Forms

The database is in BCNF. For each relation Ri in the database, for all functional dependencies a -> A in F[Ri], either a -> A is trivial, or a is a superkey.

3. SQL Queries

3.1. Complex Queries

3.1.1. Complex query 1

This complex query is used to view the number of pet profiles, petowners and caretakers for each animal type on PCS admin page.

```
SELECT a_type, COALESCE(pets, 0) AS pets, COALESCE(petowners, 0) AS petowners,
COALESCE(caretakers,0) AS caretakers FROM
(SELECT a_type, COUNT(*) AS pets FROM pets
GROUP BY a_type) AS pets
NATURAL LEFT JOIN
(SELECT a_type, COUNT(DISTINCT(username)) AS petowners FROM pets
GROUP BY a_type) AS petowners
NATURAL LEFT JOIN
(SELECT a_type, COUNT(DISTINCT(ctuname)) AS caretakers FROM cares_for
GROUP BY a_type) AS caretakers;
```

3.1.2. Complex Query 2

The availability table stores the range of available dates as individual dates in each row. This complex query groups and combines consecutive available dates for each caretaker, to be displayed on the browse page.

```
SELECT username, first_name, last_name, s_date, e_date FROM (
  SELECT t.username, min(t.avail_date) AS s_date,
    max(t.avail_date) AS e_date
    FROM
      (SELECT b.avail_date, b.username,
        b.avail_date + (INTERVAL '1 day' * -ROW_NUMBER()
          OVER (PARTITION BY b.username ORDER BY b.avail_date)) as i
        FROM (SELECT * FROM availability NATURAL JOIN caretakers) b)
      t
    GROUP BY t.username, i ORDER BY t.username, s_date) av
  NATURAL JOIN users;
```

3.1.3. Complex query 3

This complex query is used on the caretaker to filter the list of bids to get the maximum number of bids based on the category specified for the same time period. The current categories that the caretaker can select from is pouname (pet owner's username), name (pet names) and a minimum price.

```
  CREATE OR REPLACE FUNCTION get_max_by(cat varchar(64))
  RETURNS TABLE (category VARCHAR(64), start_date DATE, end_date DATE,
max_amount NUMERIC) AS
  $$ BEGIN
   IF cat = 'pouname' THEN RETURN QUERY(SELECT pouname, s_date, e_date,
MAX(price) FROM bids GROUP BY pouname, s_date, e_date);
   ELSIF cat = 'animal type' THEN RETURN QUERY (SELECT a_type, s_date, e_date,
MAX(price) FROM bids NATURAL JOIN pets GROUP BY a_type, s_date, e_date);
   ELSE RETURN QUERY(SELECT name, s_date, e_date, MAX(price) FROM bids WHERE
bids.price >= CAST(cat AS NUMERIC) GROUP BY name, s_date, e_date);
  END IF; END; $$
LANGUAGE plpgsql;

SELECT * FROM get_max_by('21');
```

3.2. Triggers

3.2.1. Trigger 1

This trigger is used to ensure that the number of pets that are being taken care of by a Caretaker on a particular date is updated correctly. When a bid is accepted, the number of pets for that caretaker for the corresponding dates in availability are incremented. The constraint enforced by this trigger is that the number of pets that a Caretaker can take care of cannot exceed the maximum number of pets that they can take care of (i.e full time Caretakers can take care of maximum 5 pets whereas part-time Caretakers can take care of maximum 2 pets unless they have a good rating). This trigger ensures that the num_of_pets that the caretaker is taking care of is incremented whenever a bid is won, subsequently, trigger 3 will ensure that it will not exceed the maximum num_of_pets by deleting the availability of Caretaker. This trigger also enforces another constraint that only 1 Caretaker can take care of a particular pet at any point in time. When a bid is successful, the trigger deletes all other bids from that pet for that caretaker that falls within the specified date range.

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

CREATE OR REPLACE FUNCTION update_avail_pets()
RETURNS trigger AS
$$
BEGIN
UPDATE availability
SET num_of_pets = num_of_pets + 1
WHERE availability.avail_date >= NEW.s_date AND availability.avail_date <=
NEW.e_date AND availability.username = new.ctuname;
DELETE FROM bids
WHERE bids.name = new.name AND bids.pouname = new.pouname AND is_win = FALSE
AND ((bids.s_date >= new.s_date AND bids.s_date <= new.e_date)
OR (bids.e_date >= new.s_date AND bids.e_date <= new.e_date));
RETURN NEW;
END;
$$
 LANGUAGE plpgsql;
```

3.2.2. Trigger 2

This trigger is used to enforce the constraint that ratings can only be added by a petowner at the end of the care period. If a petowner tries to add a rating before the end date of the care period, it will not get updated.

```
CREATE TRIGGER rating_review_update
BEFORE UPDATE
ON bids
FOR EACH ROW
WHEN (NEW.rating IS NOT NULL)
EXECUTE PROCEDURE check_valid_rating();

CREATE OR REPLACE FUNCTION check_valid_rating()
RETURNS TRIGGER AS
$$
BEGIN
IF NOW() >= new.e_date AND new.is_win = TRUE THEN
    RETURN NEW;
ELSE
    RETURN NULL;
END IF;
END;
$$
LANGUAGE plpgsql;
```

3.2.3. Trigger 3

This trigger is used to remove availability when the number of pets is equal to the maximum number of pets that the caretaker is allowed to have at one time. This enforces the constraint of the caretaker not being able to take care of pets more than the maximum num_of_pets allowed.

```
CREATE OR REPLACE FUNCTION remove_availability()
RETURNS trigger AS
$t$ BEGIN IF new.num_of_pets IN (SELECT max_pets FROM caretaker c  WHERE
c.username = new.username) THEN  DELETE FROM availability WHERE
availability.avail_date = new.avail_date AND availability.username =
new.username; END IF; RETURN NULL; END; $t$
LANGUAGE plpgsql;

CREATE TRIGGER is_full
AFTER UPDATE
ON availability
FOR EACH ROW
```

```
EXECUTE PROCEDURE remove_availability();
```

4. Web Application Design

4.1. Index Page

The homepage seen by the user when they first enter the website. From the navigation bar, they can register for a new account, log in to their existing account or access the about us page.

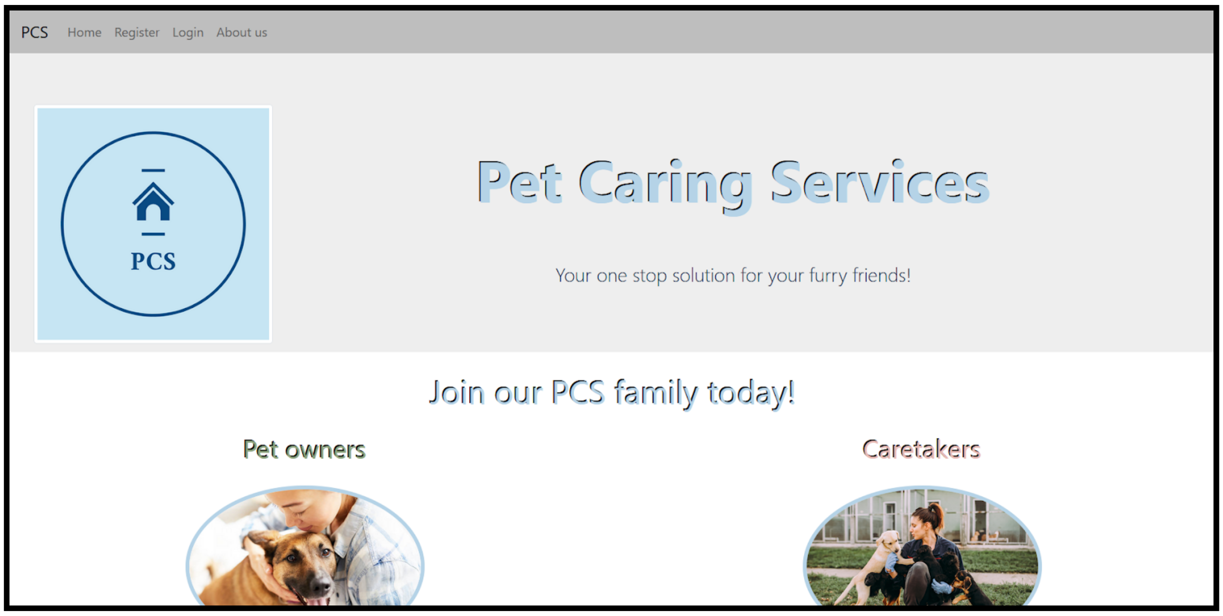
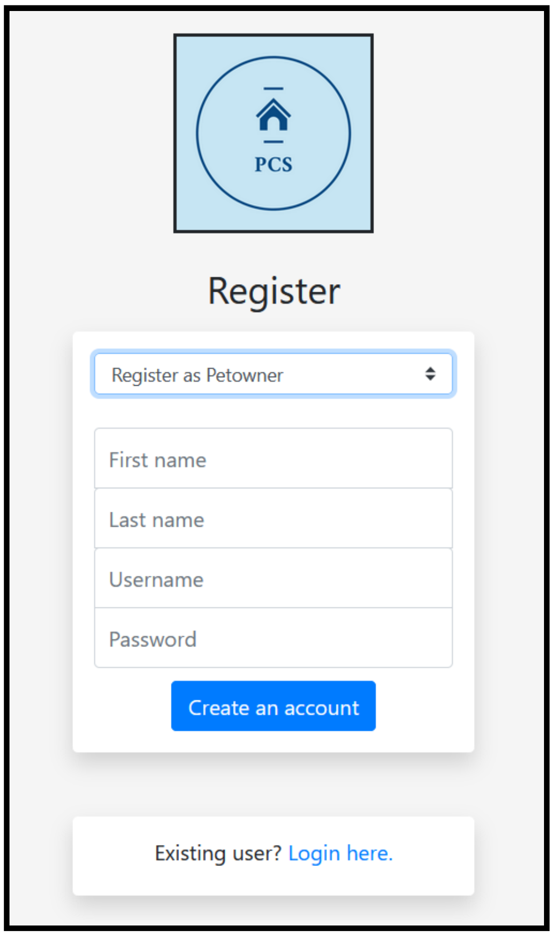

PCS
Home
Register
Login
About us
PCS
Pet Caring Services
Your one stop solution for your furry friends!
Join our PCS family today!
Pet owners
Caretakers

Figure 2: Homepage

4.2. Register and Login Pages

New users can register and existing users can log in through these pages. Users can register as a Petowner, Part-time Caretaker or Full-time Caretaker by selecting it in the drop down box.



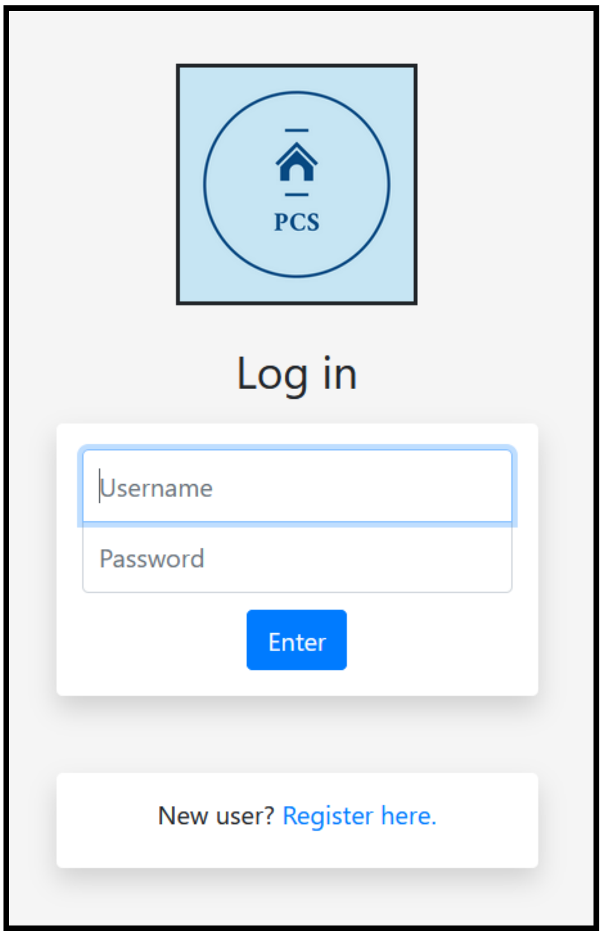
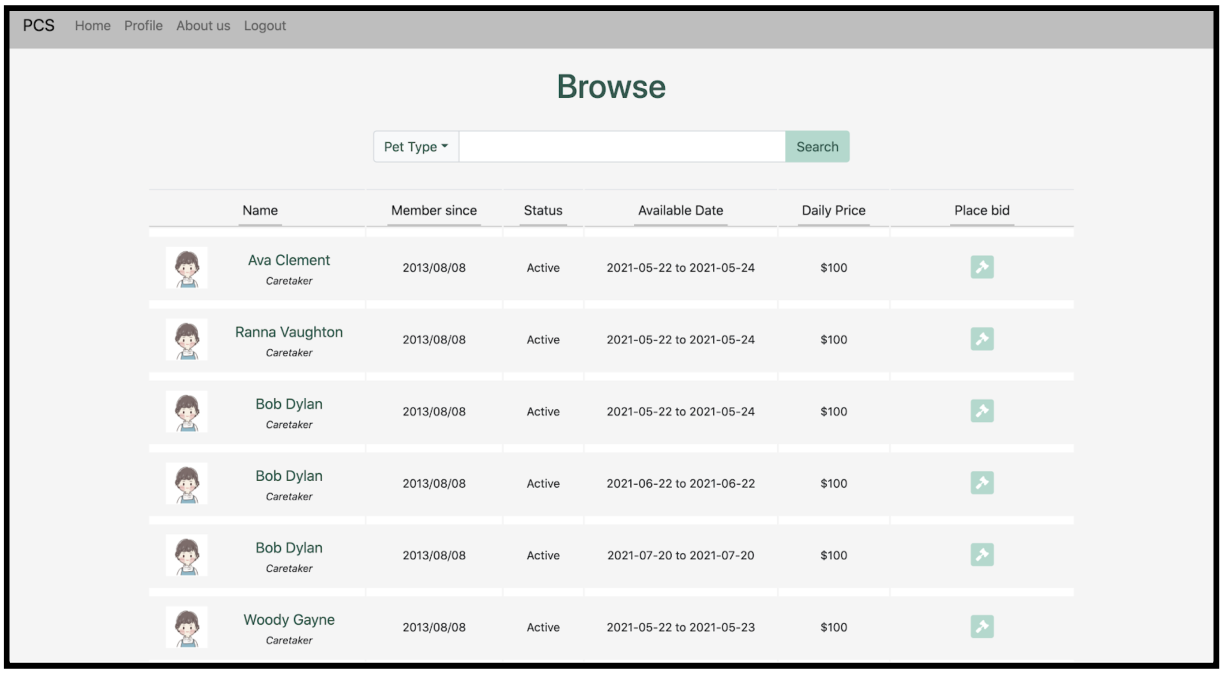


Figure 3: Register page

Figure 4: Log in page

4.3. Browse and Bid Pages

For both Caretakers and Petowners to browse and view caretaker profiles. Only Petowners can place a bid by clicking the bid button on the right side.

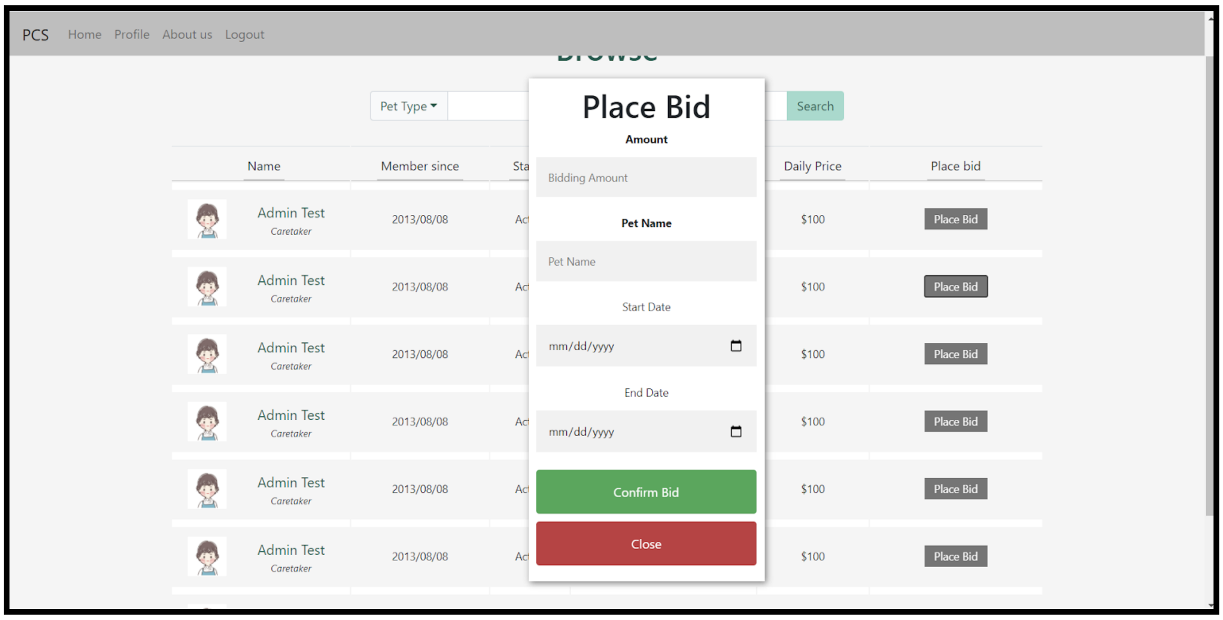

PCS Home Profile About us Logout

Browse

Pet Type Search

Name	Member since	Status	Available Date	Daily Price	Place bid
Ava Clement *Caretaker*	2013/08/08	Active	2021-05-22 to 2021-05-24	$100	
Ranna Vaughton *Caretaker*	2013/08/08	Active	2021-05-22 to 2021-05-24	$100	
Bob Dylan *Caretaker*	2013/08/08	Active	2021-05-22 to 2021-05-24	$100	
Bob Dylan *Caretaker*	2013/08/08	Active	2021-06-22 to 2021-06-22	$100	
Bob Dylan *Caretaker*	2013/08/08	Active	2021-07-20 to 2021-07-20	$100	
Woody Gayne *Caretaker*	2013/08/08	Active	2021-05-22 to 2021-05-23	$100	

Figure 5: Browse Page


PCS Home Profile About us Logout

Pet Type Search

Name	Member since	Sta	Daily Price	Place bid
Admin Test *Caretaker*	2013/08/08	Ac	$100	Place Bid
Admin Test *Caretaker*	2013/08/08	Ac	$100	Place Bid
Admin Test *Caretaker*	2013/08/08	Ac	$100	Place Bid
Admin Test *Caretaker*	2013/08/08	Ac	$100	Place Bid
Admin Test *Caretaker*	2013/08/08	Ac	$100	Place Bid
Admin Test *Caretaker*	2013/08/08	Ac	$100	Place Bid

Place Bid

Amount

Bidding Amount

Pet Name

Pet Name

Start Date

mm/dd/yyyy

End Date

mm/dd/yyyy

Confirm Bid

Close

Figure 6: Bid popup on Browse page

5. Conclusion

5.1. Challenges Faced

- We faced difficulties when trying to satisfy all the constraints of the application in the ER diagram and had to go through multiple iterations of amendments before finalizing the design.
- We had issues with authentication and took some time to understand the concept of cookies and sessions.
- We were not familiar with NodeJs, HTML and CSS and had to spend a large portion of our time trying to familiarize ourselves with the syntax and integration.

5.2. Lessons Learnt

- We learnt how to design an effective database system, from an ER model to the relational schema. We were able to apply the concept of Normal Forms learnt in the course to reduce redundancies in the relations.
- We also picked up skills in web development. We gained valuable experience in both front end and back end development.
- We learnt how to manage our time and prioritize tasks. With limited time, we had to prioritize the essential functionalities of the application.