Student Details + Roles & Responsibilities

Team 48

Name	Student Number	Roles & Responsibilities
Alex Koh Nan Yu	E0309846	Backend/Cloud Deployment
Goh Rui Pink Samantha	E0319185	Frontend Development
Ning Sheng Ying	E0309568	Database Design & Administration
Tan Zheng Fu Justin	E0311076	Backend Development, Database Design & Administration
Yeo Dong Han	E0321479	Frontend development

Application Data Requirements & Functionalities

On creation of the user account, the user will be registered as a Pet Owner and a Part Time Caretaker. This follows from real life applications like Carousell, where upon registration, the user can be both a buyer and seller.

A pet owner is able to add his pet and any special requirements it has. He is also able to see and edit his pet's information except for the pet name. In addition, the pet owner can search for his desired dates and see a list of available dates and prices posted by the caretaker. He can then choose which caretaker he wants and submit a bid. Although the term 'bid' is used, the pet owner is unable to specify his price. In other words, what he sees is what he will pay.

The caretaker can be categorized into 2 types, Full timer and part timer. A part timer is able to convert and is able to convert to a full time caretaker by providing 2 x 150 consecutive available days.

For both types of caretakers, they are able to provide their available dates, as well as the pet type he/she can care for on those dates. The caretaker will not be able to provide overlapping dates for the

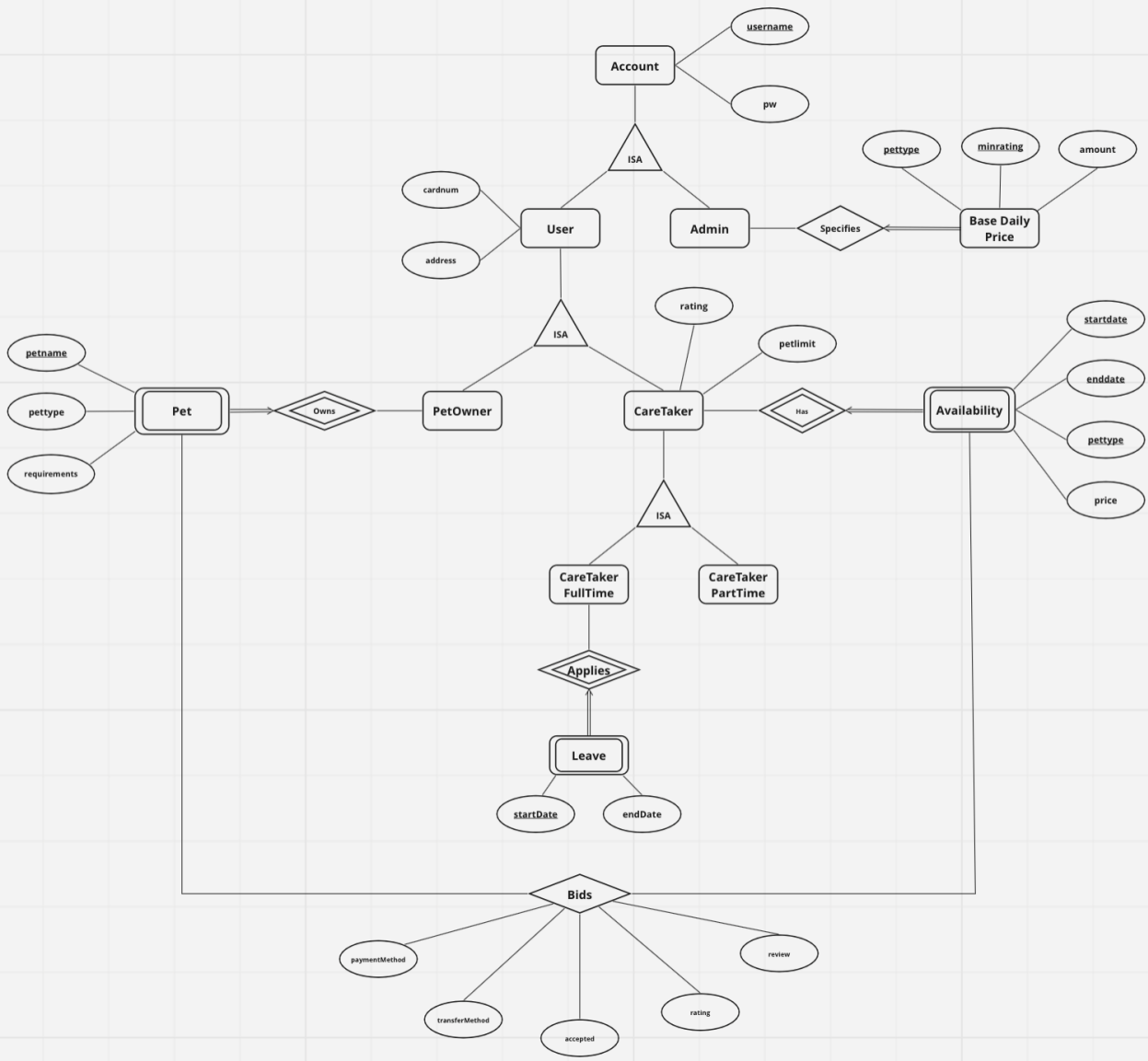
availability. For example, supposed the caretaker has submitted an availability from 2020/12/01 to 2020/12/10 for the pet type ‘Dog’. He will not be able to submit another availability which overlaps the above range, like 2020/11/19 to 2020/12/05. However, he is able to submit an availability has the same range but for a different pet type. Using the same example as above, he is able to submit another availability from 2020/12/01 to 2020/12/10 for the pet type ‘Cat’. This means that he can take care of 2 different pet types for that period.

For both types of caretaker, they can view upcoming bids that they have accepted, as well as for the current period, the pet they are taking care of. They can also view the amount that was owed to them.

For full time caretakers, their bids are automatically accepted until they have reached the pet limit for the period.

An admin can edit the base minimum price of a pet. Lastly, they can also view the total salary to be paid to all caretakers.

ER model



Constraints not enforced by our ER Diagram:

1. The cost of caring for a pet is the number of days times the daily price stated by the caretaker.
2. Once selected by the caretaker, the petowner must pay for the amount upfront either by pre-registered credit card or paying cash.
3. A full-time caretaker is treated as available until they apply for leave.
4. They cannot apply for leave if there is at least one pet under their care.

5. When bid by any petowner, a full-time caretaker will always accept the job immediately if possible.
6. For each part-time caretaker, they should be able to specify their availability for the current year and the next year.
7. At any single point in time, a part-time caretaker cannot take care of more than 2 pets unless they have a good rating.
8. The salary of a full-time caretaker depends on how many pets are taken care of in a given month for how many days.
9. A full-time caretaker will receive a salary of $3000/month for up to 60 pet-days. For any excess pet-day, they will receive 80% of their price as bonus. For part-time caretaker, the PCS will take 25% of their price as payment.
10. The successful bidder could either be chosen by the Care Taker or automatically selected by the system based on some criteria.
11. Each full-time caretaker must work for a minimum of 2 x 150 consecutive days a year.

5. Relational schema derived from ER model

```
-- accounts table
CREATE TABLE accounts (
	username VARCHAR(20) PRIMARY KEY,
	pw	VARCHAR(20) NOT NULL
);

-- users table
CREATE TABLE users (
	username VARCHAR(20) PRIMARY KEY REFERENCES accounts(username)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	cardnum  VARCHAR(20),
	address  VARCHAR(20)
```

```
);

-- admins table
CREATE TABLE admins (
	username VARCHAR(20) PRIMARY KEY REFERENCES accounts(username)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);

-- basedailyprices table
CREATE TABLE basedailyprices (
	pettype		VARCHAR(20),
	minrating	NUMERIC(3,2),
	username_admin VARCHAR(20) REFERENCES admins(username)
		ON DELETE SET NULL
		ON UPDATE CASCADE,
	amount		NUMERIC,

	PRIMARY KEY(pettype, minrating)
);

-- petowners table
CREATE TABLE petowners (
	username VARCHAR(20) PRIMARY KEY REFERENCES users(username)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);
```

```
-- pets table
CREATE TABLE pets (
	username_petowner VARCHAR(20) REFERENCES petowners(username)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	petname		VARCHAR(20),
	pettype		VARCHAR(20) NOT NULL,
	requirements	TEXT,

	PRIMARY KEY(username_petowner, petname)
);

-- caretakers table
CREATE TABLE caretakers (
	username VARCHAR(20) PRIMARY KEY REFERENCES users(username)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
	rating   NUMERIC(3,2),
	petlimit INTEGER NOT NULL,

	CHECK (rating >= 0 AND rating <= 5),
	CHECK (petlimit = 2 OR petlimit = 5)
);

-- availabilities table
CREATE TABLE availabilities takers (
	username_caretaker VARCHAR(20) REFERENCES caretakers(username)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
```

```
	startdate		DATE		NOT NULL,
	enddate			DATE		NOT NULL,
	pettype			VARCHAR(20) NOT NULL,
	price			NUMERIC		NOT NULL,

	PRIMARY KEY (username_caretaker, startdate, enddate, pettype),
	CHECK (enddate - startdate >= 150),
	CHECK (price >= 0)
);

-- full time caretakers table
CREATE TABLE caretakers_ft (
	username VARCHAR(20) PRIMARY KEY REFERENCES caretakers(username)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);

-- part time caretakers table
CREATE TABLE caretakers_pt (
	username VARCHAR(20) PRIMARY KEY REFERENCES caretakers(username)
		ON DELETE CASCADE
		ON UPDATE CASCADE
);

-- leaves table
CREATE TABLE leaveschedule (
	username  VARCHAR(20) REFERENCES caretakers_ft(username)
		ON DELETE CASCADE
		ON UPDATE CASCADE,
```

```
	startdate DATE,
	enddate   DATE NOT NULL,

	PRIMARY KEY (username, startdate),
	CHECK (enddate >= startdate)
);

-- bids table
CREATE TABLE bids (
	username_petowner  VARCHAR(20),
	petname            VARCHAR(20),
	pettype            VARCHAR(20),
	username_caretaker VARCHAR(20),
	startdate          DATE,
	enddate            DATE,
	price              NUMERIC NOT NULL,
	accepted           BOOLEAN,
	transfermethod     VARCHAR(20),
	paymentmethod      VARCHAR(20),
	rating             NUMERIC(3,2),
	review             TEXT,

	PRIMARY KEY (username_petowner, petname, pettype, username_caretaker,
startdate, enddate),
	FOREIGN KEY (username_petowner, petname) REFERENCES pets(username_petowner,
petname),
	FOREIGN KEY (username_caretaker, startdate, enddate, pettype) REFERENCES
availabilities(username_caretaker, startdate, enddate, pettype)
);
```

Constraints not enforced by our relational schema:

1. The cost of caring for a pet is the number of days times the daily price stated by the caretaker.
2. Once selected by the caretaker, the petowner must pay for the amount upfront either by pre-registered credit card or paying cash.
3. A full-time caretaker is treated as available until they apply for leave.
4. They cannot apply for leave if there is at least one pet under their care.
5. When bid by any petowner, a full-time caretaker will always accept the job immediately if possible.
6. For each part-time caretaker, they should be able to specify their availability for the current year and the next year.
7. At any single point in time, a part-time caretaker cannot take care of more than 2 pets unless they have a good rating.
8. The salary of a full-time caretaker depends on how many pets are taken care of in a given month for how many days.
9. A full-time caretaker will receive a salary of $3000/month for up to 60 pet-days. For any excess pet-day, they will receive 80% of their price as bonus. For part-time caretaker, the PCS will take 25% of their price as payment.
10. The successful bidder could either be chosen by the Care Taker or automatically selected by the system based on some criteria.

3NF vs BCNF

The ‘accounts’ table has the following attributes: { username, password }. The set of Functional Dependencies of the ‘accounts’ table = { username -> password }. Since ‘username’ is the primary key, the ‘accounts’ table is in BCNF.

The ‘admins’ table has the following attributes: { username }. The set of Functional Dependencies of the ‘admins’ table = { username -> username }. Since there is only one trivial FD, the ‘admins’ table is in BCNF.

The ‘availabilities’ table has the following attributes: { username_caretaker, pettype, startdate, enddate, price }. The set of Functional Dependencies of the ‘availabilities’ table = { (username_caretaker, pettype, startdate, enddate) -> price }. Since there is only one FD, and (‘username_caretaker’, ‘pettype’, ‘startdate’, ‘enddate’) is the superkey, the ‘availabilities’ table is in BCNF.

The ‘basedailyprices’ table has the following attributes: { pettype, minrating , username_admin, amount }. The set of Functional Dependencies of the ‘basedailyprices’ table =
{ (pettype, minrating) ->(username_admin, amount) }. Since there is only one FD, and (‘pettype’, ‘minrating’) is the superkey, the ‘basedailyprices’ table is in BCNF.

The bids’ table has the following attributes: { username_petowner, petname, pettype, username_caretaker, startdate, enddate, price, accepted, transfermethod, paymentmethod, rating, review, price }. The primary key of this table contains the following attributes: { username_petowner, petname, pettype, username_caretaker, startdate, enddate }. This primary key uniquely identifies the other columns, i.e. { price, accepted, transfermethod, paymentmethod, rating, review, price }. The set of Functional Dependencies of the ‘bids’ table = { (username_petowner, petname, pettype, username_caretaker, startdate, enddate) -> (price, accepted, transfermethod, paymentmethod, rating, review, price) }. Since there is only one FD, this table is in BCNF.

The ‘caretakers’ table has the following attributes: { username, rating, petlimit }. The set of Functional Dependencies of the ‘caretakers’ table = { username -> (rating, petlimit) }. Since there is only one FD, and ‘username’ is the superkey, the ‘caretakers’ table is in BCNF.

The ‘caretakers_ft’ table has the following attributes: { username }. The set of Functional Dependencies of the ‘caretakers_ft’ table = { username -> username }. Since there is only the trivial FD, the ‘caretakers’ table is in BCNF.

The ‘caretakers_pt’ table has the following attributes: { username }. The set of Functional Dependencies of the ‘caretakers_pt’ table = { username -> username }. Since there is only the trivial FD, the ‘caretakers_pt’ table is in BCNF.

The ‘leaveschedule’ table has the following attributes: { username, startdate, enddate }. The set of Functional Dependencies of the ‘leaveschedule’ table = { (username, startdate) -> enddate }. Since (username, startdate) is the primary key and there is only one FD, the ‘leaveschedule’ table is in BCNF.

The ‘petowners’ table has the following attributes: { username }. The set of Functional Dependencies of the ‘petowners’ table = { username -> username }. Since there is only the trivial FD, the ‘petowners’ table is in BCNF.

The ‘pets’ table has the following attributes: { username_petowner, petname, pettype, requirements }. The set of Functional Dependencies of the ‘pets’ table = { (username_petowner, petname, pettype) -> requirements }. Since (username_petowner, petname, pettype) is the primary key, the ‘pets’ table is in BCNF.

The ‘pets’ table has the following attributes: { username_petowner, petname, pettype, requirements }. The set of Functional Dependencies of the ‘pets’ table = { (username_petowner, petname, pettype) -> requirements }. Since (username_petowner, petname, pettype) is the primary key, the ‘pets’ table is in BCNF.

The ‘pets’ table has the following attributes: { username_petowner, petname, pettype, requirements }. The set of Functional Dependencies of the ‘pets’ table = { (username_petowner, petname, pettype) -> requirements }. Since (username_petowner, petname, pettype) is the primary key, the ‘pets’ table is in BCNF.

The ‘users’ table has the following attributes: { username, cardnum, address }. The set of Functional Dependencies of the ‘users’ table = { username -> (cardnum, address) }. Since username is the primary key, the ‘users’ table is in BCNF.

Thus in summary, our database is in BCNF.

Three non-trivial/interesting triggers used

This trigger ensures that the price submitted by the caretaker for his availability is not lower than the base daily price as specified by PCS admin. It works by comparing the rating of the identified caretaker with the base price specified. If the base price is greater than the price by the caretaker, then an error will be raised.

checkBasePrice()

```
CREATE OR REPLACE FUNCTION checkBasePrice() RETURNS TRIGGER AS
‘DECLARE
    basePrice NUMERIC; rating NUMERIC;
  BEGIN
    SELECT c.rating INTO rating FROM caretakers c WHERE c.username =
NEW.username_caretaker;
    SELECT b.amount INTO basePrice FROM basedailyprices b WHERE b.minrating = rating
AND b.pettype = NEW.pettype;
    IF new.price < basePrice THEN RAISE EXCEPTION 'cannot be lower than base price';
    END IF;
  RETURN NEW;
END;’

CREATE TRIGGER checkBasePrice
BEFORE INSERT ON availabilities
FOR EACH ROW EXECUTE PROCEDURE checkBasePrice();
```

checkAvailabilityOverlap()

This trigger ensures that the availability that the caretaker submitted does not overlap with any of his current availability. For example, a caretaker submits an availability indicating that he is free for caretaking from Monday to Friday. Then, he is not able to create another availability that overlaps with this time period, for example, Thursday to Saturday or Sunday to Saturday.

```
CREATE OR REPLACE FUNCTION checkAvailabilityOverlap() RETURNS TRIGGER AS
‘BEGIN
    IF (1 IN (SELECT 1 FROM availabilities av WHERE av.username_caretaker =
NEW.username_caretaker AND NEW.startdate = av.startdate AND NEW.enddate =
av.enddate))
THEN NEW.username_caretaker = NEW.username_caretaker;

    ELSIF (1 IN (SELECT 1 FROM availabilities av WHERE av.username_caretaker =
NEW.username_caretaker AND NEW.startdate < av.startdate AND NEW.enddate >
av.enddate))
    OR (1 IN (SELECT 1 FROM availabilities av WHERE av.username_caretaker =
NEW.username_caretaker
    AND NEW.startdate BETWEEN av.startdate AND av.enddate OR NEW.enddate BETWEEN
av.startdate AND av.enddate))
    THEN RAISE EXCEPTION 'Cannot have overlapping availabilities'; END IF; RETURN
NEW; END;’

CREATE TRIGGER checkAvailabiltiesOverlap
BEFORE INSERT ON availabilities
FOR EACH ROW EXECUTE PROCEDURE checkAvailabilityOverlap();
```

checkBidsInRangeOfAvailability()

This trigger allows the petowner to submit a bid that is within the date range of a caretaker’s availability. For example, supposed that a caretaker had submitted an availability indication that he is free for 10 days from 1st November to 10th November. Then, the petowner can submit a bid with the dates of the range 1st November to 10th November.

```
CREATE OR REPLACE FUNCTION checkBidsInRangeOfAvailability() RETURNS TRIGGER AS
‘BEGIN IF (1 NOT IN (SELECT 1 FROM availabilities av
```

```
WHERE av.username_caretaker = NEW.username_caretaker
AND av.pettype = NEW.pettype
AND (NEW.startdate BETWEEN av.startdate AND av.enddate)
AND (NEW.enddate BETWEEN av.startdate AND av.enddate)))
THEN RAISE EXCEPTION 'Bids date range not in Availabilities'; END IF; RETURN NEW;
END;’

CREATE TRIGGER checkBidsInRangeOfAvailability
BEFORE INSERT ON bids
FOR EACH ROW EXECUTE PROCEDURE checkBidsInRangeOfAvailability();
```

updateRating()

This trigger ensures that when a successful bid is updated with the rating, it will automatically update the caretaker’s overall rating.

```
CREATE OR REPLACE FUNCTION updateRating() RETURNS TRIGGER AS
‘ DECLARE r INTEGER;
BEGIN SELECT AVG(bids.rating) INTO r FROM bids WHERE bids.username_caretaker =
NEW.username_caretaker AND bids.rating IS NOT NULL;
IF r IS NULL THEN r = 3; END IF;
UPDATE caretakers SET rating = r WHERE username = NEW.username_caretaker;
RETURN NEW;
END;’

CREATE TRIGGER updateRating
AFTER UPDATE ON bids
FOR EACH ROW EXECUTE PROCEDURE updateRating();
```

Complex SQL Queries

Get the total salary to be paid to all caretakers for a specific period.

This query takes in 2 inputs, a start-date and end-date, and returns the salary to be paid to all caretakers who have worked for that period.

Using CASE, we identified the type of caretaker and used the following logic to calculate their salary. If the caretaker is a full timer and has < 60 pet days, then $3000. Else if he has >= 60 pet days, $3000 + 0.8 * asking price of each successful bid. If the caretaker is a part timer, 0.25 * asking price of each successful bid.

```
CREATE OR REPLACE FUNCTION getTotalSalaryToBePaid(sd DATE, ed DATE)
RETURNS TABLE (
	username VARCHAR(20),
	caretakertype TEXT,
	dayswork BIGINT,
	salary NUMERIC
)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
    RETURN QUERY
	WITH caretaker_daysworked AS (
	SELECT ct.username, 'fulltime' AS caretakertype, ft.dayswork FROM (SELECT
S.username_caretaker, SUM(S.numdaysworked) AS dayswork
	FROM (SELECT username_caretaker, enddate-startdate+1 AS numdaysworked FROM
bids WHERE accepted = 'True' AND startdate >= sd AND enddate <= ed)
	AS S GROUP BY S.username_caretaker) AS ft INNER JOIN caretakers_ft ct ON
ct.username = ft.username_caretaker
	UNION
```

```
	SELECT ct.username, 'parttime' AS caretakertype, pt.dayswork FROM (SELECT
S.username_caretaker, SUM(S.numdaysworked) AS dayswork
	FROM (SELECT username_caretaker, enddate-startdate+1 AS numdaysworked FROM
bids WHERE accepted = 'True' AND startdate >= sd AND enddate <= ed)
	AS S GROUP BY S.username_caretaker) AS pt INNER JOIN caretakers_pt ct ON
ct.username = pt.username_caretaker
	)

SELECT *, CASE WHEN cw.caretakertype = 'fulltime' THEN
	CASE WHEN cw.dayswork < 60 THEN 3000 ELSE
	(SELECT sum((b.enddate - b.startdate + 1) * b.price) FROM bids b
	WHERE accepted = 'True' AND b.startdate >= sd AND b.enddate <= ed AND
b.username_caretaker = cw.username
	GROUP BY b.username_caretaker) * 0.8 + 3000
	END
	WHEN cw.caretakertype = 'parttime' THEN (SELECT sum((b.enddate - b.startdate +
1) * b.price) FROM bids b
	WHERE accepted = 'True' AND b.startdate >= sd AND b.enddate <= ed AND
b.username_caretaker = cw.username
	GROUP BY b.username_caretaker) * 0.25
	END AS salary FROM caretaker_daysworked cw;
END; $BODY$;
```

Get caretaker total days work

This query returns an ordered list of all the caretakers registered with the company and the number of days work, ordered in descending values.

```
`SELECT S.username_caretaker, SUM(S.numdaysworked) AS dayswork
```

```
        FROM (SELECT username_caretaker, enddate-startdate+1 AS numdaysworked FROM
bids WHERE accepted = 'True' AND startdate >= '${startdate}' AND enddate <=
'${enddate}')
        AS S GROUP BY S.username_caretaker
        UNION
        SELECT username, 0 AS dayswork
        FROM (SELECT username FROM caretakers EXCEPT SELECT S.username_caretaker
        FROM (SELECT username_caretaker, enddate-startdate+1 AS numdaysworked FROM
bids WHERE accepted = 'True' AND startdate >= '${startdate}' AND enddate <=
'${enddate}')
        AS S GROUP BY S.username_caretaker) AS B
        ORDER BY dayswork DESC`
```

Get eligibility to convert to full time

This query checks if a part time caretaker is able to convert to a full time caretaker by checking if there is 2 * 150 consecutive days block in his indicated availability.

```
SELECT
    CASE WHEN EXISTS ((SELECT 1 FROM (SELECT COUNT(*) cnt FROM (
            SELECT DISTINCT enddate-startdate+1 AS days
            FROM availabilities
            WHERE username_caretaker = '${usernamect}') blocklengths
            WHERE days >= 150) c WHERE c.cnt = 2) UNION
            (SELECT 1
            FROM (SELECT count(*)
            FROM (SELECT DISTINCT enddate-startdate+1 AS availdays
            FROM availabilities WHERE username_caretaker = '${usernamect}') as c
            WHERE availdays >= 300) AS n
            WHERE n.count = 1))
```

```
        THEN 'eligible'
        ELSE 'not eligible'
    END;
```

Software tools & Frameworks

Frontend

The web-based User Interface was built using **Svelte**, an open-source frontend JavaScript framework. Svelte was architected to be lighter and faster compared to other modern frontend JavaScript frameworks/libraries, such as Angular and React. This is mainly because Svelte has its own compiler to compile svelte files into client-side JavaScript code at build time. In addition, the generated code manipulates the DOM directly as well, instead of manipulating a Virtual DOM, which React infamously does.

Backend

The web server software was built using the **JavaScript** programming language. More specifically, it was built using **Node.js**, since Node.js is a JavaScript runtime environment that executes JavaScript code outside a web browser.

To accelerate the development of our REST APIs, we employed the **Koa** framework. Koa is a relatively new web backend framework, and it was designed by the team behind the famous Express framework. It leverages on async functions, allowing backend developers to avoid legacy JavaScript pitfalls such as “callback hell”.

For the relational database, we used **PostgreSQL 12**, in line with the usual practice in CS2102 this semester.

Cloud Deployment

The frontend Svelte application was deployed on **Netlify**, currently running on its generous free tier. The public domain is as follows:

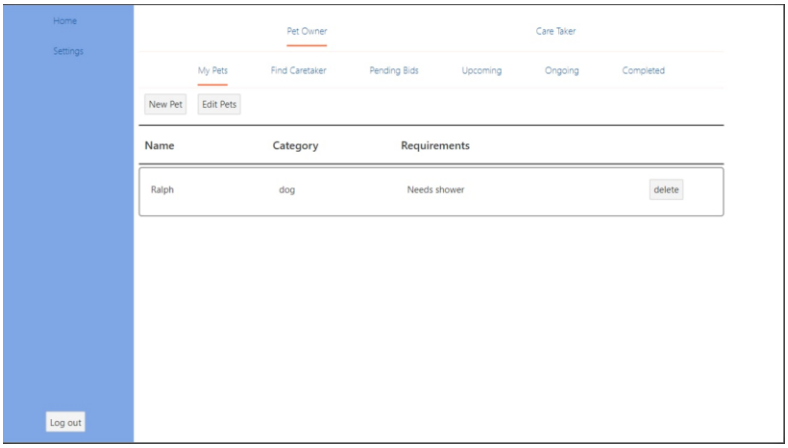
www.petpikker.netlifyapp.com

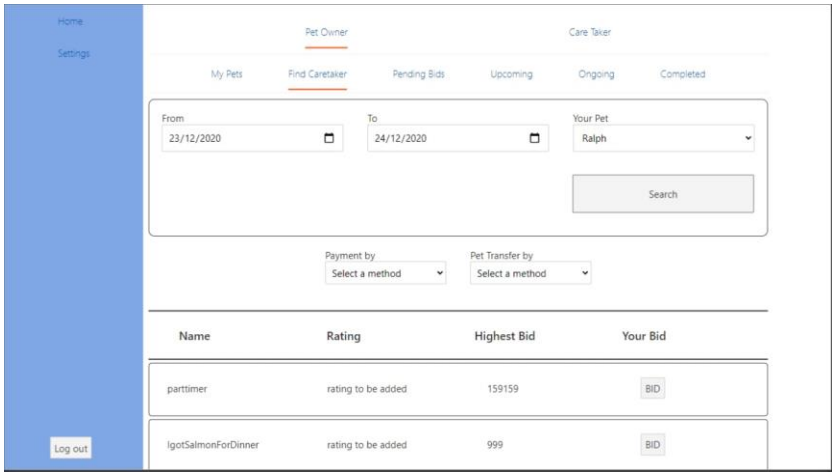
For the backend, both the web server and the database were deployed on **Amazon Web Services (AWS).**

The Node.js web server was deployed on **Amazon Elastic Compute Cloud (Amazon EC2)**, a virtual machine running on Ubuntu 18.04. The Node.js web server software is running in the background on this virtual machine, on the relatively generous free tier.

The PostgreSQL database was deployed on **Amazon Relational Database Service (Amazon RDS)**. The database is currently running on free tier, which gives 20 GB of disk storage.

Application in Action





Project Insights

- Learning new syntax like JavaScript
- Expanding knowledge of SQL to form complex queries.

- Coordination and integration of the frontend and backend.
- Communication and expectations
- Date-range check is a chore in SQL.
- Scale of this project and juggling different project from other modules
- Have to think way in advance because if there is anything wrong with the database then the logic of the application will change drastically.