CS2102 Project Report

1.0 Members & Responsibilities

The table below lists out our group members and respective responsibilities.

Member Name	Member Student No.	Responsibilities
Zhao Huan	A0211628L	Table creation, frontend implementation
Sun Yucheng	A0211299A	Heroku setup, backend adjustment
Huang Gexiang	A0183911E	SQL code writing, admin page
Xu ZhiZhi	A0205882B	SQL code writing, report writing
Zhang Xinyi	A0204934H	SQL code writing, report writing

2.0 Requirements and Functionalities

Many of our data requirements and functionalities follow what is given in the project specification document from CS2102 AY20/21 Sem1. In the following sections, you can find our requirements and functionalities, as well as data constraints.

2.1 Data requirements and constraints

Below is the data requirement of our project. Constraints are basically restatement of the requirements, and all the constraints captured are discussed in Section 3.

PCS application	Users ● There should be 3 kinds of users: Pet Owner, Care Taker and PCS Administrator. A user can be a pet owner and/or care taker, OR a pcs Administrator. A user must be at least 1 of these 3 kinds.
Pet Owner	Can own multiple pets. Bids ● Pet owner can only bid for Care Takers who are available and can care for their pet's type. ● Has to pay through a specified method (ie. cash/card). ● Has to agree on method of transportation of pet with target care taker. ● Can only leave rating and review for care takers who took care of their pets if bids are successful. Reviews can be viewed by all users.
Care Taker	Bids ● Daily price for each pet type depends on the base price set by the pcs Administrator and rating of one-self. Calculate using the formula (base price * (1 + rating/100)). ● Care takers can only care for pet types they are capable of caring for. ● Care takers cannot care for more than the maximum number of pets at any given time.

	Each care taker must be either a full time or part time employee. ● Full time care takers ○ Must work for 2*150 consecutive days a year ○ Treated as available till apply for leave ○ Can take care of maximum 5 pets ○ Always accepts bids immediately if possible ○ Receives $3000 if taken care of 60 pets or less in the past month. For excess pets (ie > 60), receives [(x - 60) * average daily price of given care taker over capable categories * 80%] as bonus. ● Part time care takers ○ Can specify availability from current date to 2 years later ○ When rating is below 4, can take care of maximum 2 pets. Otherwise when rating below 4.3, can take care of maximum 3 pets. Otherwise when rating below 4.6, can take care of maximum 4 pets. Otherwise can take care of 5 pets. ○ Receives 75% of stated price. Cannot apply for leave on days they have to take care of pets.
PCS Admin	Salary ● Specifies base daily price for each category of pets.
Pet	Profile ● A pet must belong to exactly one pet owner and can not be identified without knowing who the owner is. ● A pet must belong to a category. ● A pet might have some special requirements.

2.2 Application functionalities

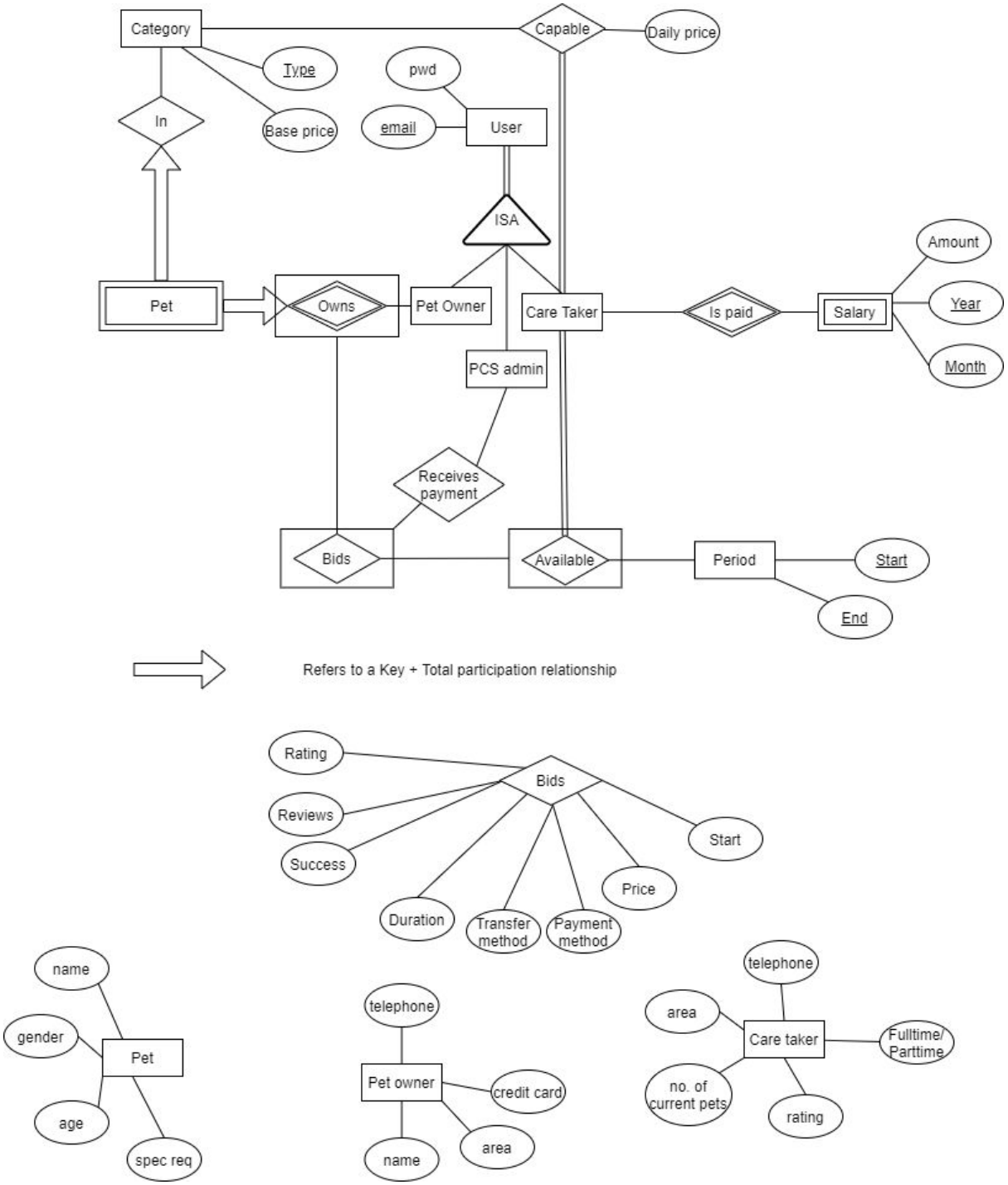
To fulfil the specifications stated in Section 2.1 and to ensure a complete and smooth user experience, our application supports the following functionalities:

1) Support the manipulation of data for the different users.
    a) All users can create accounts using emails and sign in, protected by password authorization. (PCS Administrator account cannot be created through sign up)
    b) All users have profiles. Pets also have profiles.
2) Support data access for the different users.
3) Support for PCS Administrator:
    a) Browse the total number of Pet taken care of in the month.
    b) Browse the total salary to be paid to all Care Taker for the given month.
    c) Browse the month with the highest number of jobs.
    d) Browse the underperforming full-time Care Taker.
4) Support for Care Taker:
    a) Browse history jobs, reviews and salary.
    b) Browse and manage the user profile.
    c) View a summary of this month's statistics, including the number of cares, the number of pet days and accumulated salary so far etc.
5) Support for Pet Owner:
    a) Browse and manage the user profile.
    b) Search for all Care Takers and Pet Owners nearby. (to be done)
    c) Browse and manage their Pet information.
    d) Search for a list available caretakers for a given type of pet at given duration in descendent order of rating.

3.0 Application Design

The following sections describe in detail our design and implementation of the application, covering the entity-relation diagram and relational schema. There are also several non-trivial triggers and queries listed for inspection.

3.1 ER Diagram



3.2 Relational Schema

3.2.1 DDL statements showing relational database schema

No.	Relational Schema	Data Constraints Enforced	Normal Forms

1	`CREATE TABLE users(`
	`  email VARCHAR PRIMARY KEY,`
	`  password VARCHAR NOT NULL`
	`);`
		Primary Key Constraints:
		Each user has an `email` which uniquely identifies the user, acting as a primary key.
		
		Foreign Key Constraints: None
		
		Check Constraints:
		`password` must not be null.
			BCNF

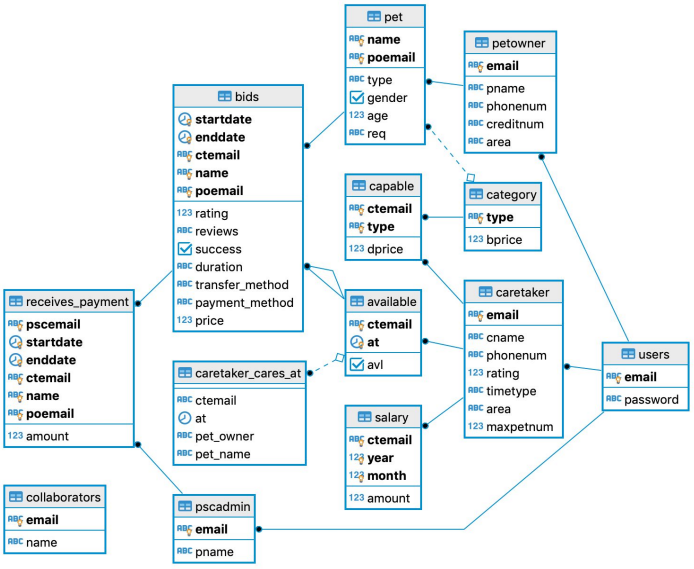
2	`CREATE TABLE petowner(`
	`  email VARCHAR PRIMARY KEY REFERENCES users(email),`
	`  pname VARCHAR NOT NULL,`
	`  phonenum VARCHAR(8) NOT NULL,`
	`  creditnum VARCHAR,`
	`  area VARCHAR NOT NULL`
	`);`
		Primary Key Constraints:
		A pet owner is a user, so each pet owner is uniquely identified by the `email`, acting as a primary key.
		
		Foreign Key Constraints:
		Pet owner's `email` must reference the `email` in the user table.
		
		Check Constraints:
		`phonenum`, `pname` must not be null for information management;
		`area` must not be null for nearby caretaker searching purposes.
			BCNF

3	`CREATE TABLE caretaker(`
	`  email VARCHAR PRIMARY KEY REFERENCES users(email),`
	`  cname VARCHAR NOT NULL,`
	`  phonenum VARCHAR(8) NOT NULL,`
	`  rating NUMERIC CHECK(rating IS NULL OR (0 <= rating AND rating <= 5)),`
	`  maxpetnum INTEGER NOT NULL CHECK(0 <= maxpetnum AND maxpetnum <= 5),`
	`  timetype VARCHAR(9) NOT NULL CHECK(`
	`  timetype = 'full time'`
	`  OR timetype = 'part time'`
	`),`
	`  area VARCHAR NOT NULL`
	`);`
		Primary Key Constraints:
		A caretaker is a user, so each caretaker is uniquely identified by the `email`, acting as a primary key.
		
		Foreign Key Constraints:
		Caretaker's `email` must reference the `email` in the user table.
		
		Check Constraints:
		`phonenum`, `cname` must not be null for information management;
		`area` must not be null for nearby caretaker searching purposes;
		timetype must be specified as either `'full time'` or `'part time'`;
		`rating` can be NULL, or it must fall between 0 and 5, inclusive;
		the `maxpetnum` a caretaker can take
			BCNF

		care of must be specified, and it must fall between 0 and 5	
4	CREATE TABLE pcsadmin(email VARCHAR PRIMARY KEY REFERENCES users(email), pname VARCHAR NOT NULL);	Primary Key Constraints: A pcsadmin is a user, so each pcsadmin is uniquely identified by the `email`, acting as a primary key. Foreign Key Constraints: pcsadmin's `email` must reference the `email` in the user table. Check Constraints: `pname` must not be null for information management purposes.	BCNF
5	CREATE TABLE category(type VARCHAR PRIMARY KEY, bprice NUMERIC(6,2) NOT NULL CHECK(bprice >= 0));	Primary Key Constraints: Each category is uniquely identified by a `type`. Check Constraints: basic price for the pet type must be larger or equals to 0.	BCNF
7	CREATE TABLE pet(name VARCHAR NOT NULL, poemail VARCHAR REFERENCES petowner(email) ON DELETE CASCADE, type VARCHAR NOT NULL REFERENCES category, gender BOOLEAN, age INTEGER CHECK (age IS NULL OR age >= 0), req TEXT, PRIMARY KEY (name, poemail));	Primary Key Constraints: Each pet is uniquely identified by its name and email of the pet owner. Foreign Key Constraints: Owner's email must reference the email in the Pet Owner table. Each pet must have a type that references type in table category. Check Constraints: age of pet can be empty or positive.	BCNF
8	CREATE TABLE available(ctemail VARCHAR REFERENCES caretaker(email), at DATE NOT NULL, avl BOOLEAN, PRIMARY KEY (ctemail, at));	Primary Key Constraints: Each available date is identified by caretaker email and a date. Foreign Key Constraints: Each email must reference email in table caretaker.	BCNF
9	CREATE TABLE caretaker_cares_at(ctemail VARCHAR NOT NULL, at DATE NOT NULL, pet_owner VARCHAR NOT NULL, pet_name VARCHAR NOT NULL, PRIMARY KEY(ctemail, at, pet_owner, pet_name), FOREIGN KEY (pet_owner, pet_name) REFERENCES pet(poemail,	Primary Key Constraints: Caretaker_cares_at is identified by caretaker email, date (at), pet owner email and pet name. Foreign Key Constraints: Pet owner email and pet name reference pet owner email and pet name in the owns table.	BCNF

	name), FOREIGN KEY (ctemail, at) REFERENCES available(ctemail, at));	Caretaker email and date reference caretaker email and date (at) in the available table. Check Constraints: Caretaker email, date, pet owner email and pet name are not null.	
10	CREATE TABLE capable(ctemail VARCHAR REFERENCES caretaker(email), type VARCHAR REFERENCES category(type), dprice NUMERIC(6, 2) NOT NULL CHECK(dprice >= 0), PRIMARY KEY(ctemail, type));	Primary Key Constraints: Capable is identified by caretaker email and type of pet. Foreign Key Constraints: Caretaker email must reference to email in table caretaker Check Constraints: Price cannot be null or negative.	BCNF
11	CREATE TABLE salary(ctemail VARCHAR REFERENCES caretaker(email) ON DELETE CASCADE, amount NUMERIC(6, 2) NOT NULL CHECK(amount >= 0), year INTEGER NOT NULL CHECK(year > 1900 and year < 2100), month INTEGER NOT NULL CHECK(month <= 12 and month >= 1), PRIMARY KEY(ctemail, year, month));	Primary Key Constraints: Salary is identified by caretaker email, year and month. Foreign Key Constraints: Caretaker email reference email in table caretaker. Check Constraints: Amount to be paid must be greater or equal to 0. Month must be between 1 and 12 inclusive. Year must be between 1900 and 2100.	BCNF
12	CREATE TABLE bids(startDate DATE, endDate DATE, ctemail VARCHAR, name VARCHAR, poemail VARCHAR, rating NUMERIC(6, 2) CHECK(rating IS NULL OR (rating >= 0 AND rating <= 5)), reviews TEXT, success BOOLEAN, duration VARCHAR NOT NULL, transfer_method VARCHAR NOT NULL, payment_method VARCHAR NOT NULL, price NUMERIC(6, 2) NOT NULL, PRIMARY KEY (startDate, endDate, ctemail, name, poemail),	Primary Key Constraints: Bids are identified by start date, end date, caretaker email, pet name and pet owner email. Foreign Key Constraints: Caretaker email and start date together with caretaker email and end date reference caretaker email and date (at) in the available table. Pet name and pet owner email references pet name and pet owner email in the owns table. Check Constraints: Rating is either null or a numeric with two decimal precision between 0 and 5 (inclusive). Price should not be null and is a numeric	BCNF

	FOREIGN KEY (ctemail, startDate) REFERENCES available(ctemail, at), FOREIGN KEY (ctemail, endDate) REFERENCES available(ctemail, at), FOREIGN KEY (name, poemail) REFERENCES pet(name, poemail));	with two decimal place precision. Duration, transfer method, payment method should not be null.	
13	CREATE TABLE receives_payment(pcsEmail VARCHAR NOT NULL REFERENCES pcsadmin(email), amount NUMERIC(6, 2) NOT NULL CHECK(amount >= 0), startDate DATE, endDate DATE, ctemail VARCHAR, name VARCHAR, poemail VARCHAR, PRIMARY KEY(pcsEmail, startDate, endDate, ctemail, name, poemail), FOREIGN KEY(startDate, endDate, ctemail, name, poemail) REFERENCES bids(startDate, endDate, ctemail, name, poemail));	Primary Key Constraints: Receives_payment is identified by pcs email, start date of the bid, end date of the bid, caretaker email, pet name and pet owner email. Foreign Key Constraints: Start date and end date reference start date and end date in the bids table. Pet owner email and pet name reference pet owner email and pet name in the owns table. Caretaker email references email in the caretaker table. Check Constraints: Payment amount is more than 0, has a two decimal place precision and less than 6 digits (including the 2 decimal digits).	BCNF



Visualization of Relational Schema

3.2.2 Constraints enforced by triggers

Below contains application constraints enforced by triggers.

S/N	Constraints enforced	Trigger Name
1	Number of pets part time care-takers can take care of depends on their ratings.	`update_rating`
2	Whenever a bid is set to successful, a new entry for each day in the bid should be added in table `caretaker_cares_at`.	`update_pets`
3	Whenever a caretaker reaches his/her maximum number of pets per day, a caretaker shouldn't be available on that day anymore.	`update_pets`
4	Admin and pet owner/caretaker are non-overlapping; the three kinds of users are covering.	`adminOverlap, coverAdmin, coverPO, coverCT`
5	The price for each caretaker's type of capability cannot be lower than the base price.	`check_price`
6	Before a new bid is inserted, the specified caretaker must be available for the specified duration.	`check_available_before_insert`
7	For any bid, if the specified caretaker is full time and available, then the bid is auto-accepted.	`check_available_before_insert`
8	For any bid, if the specified caretaker is part time and available, then the bid is set to pending, and the caretaker can accept the duty if and only if he is available for the duration at the time he accepts.	`check_available_before_update`
9	For a pet owner to initiate a bid using card payment, he/she must have a credit card number available.	`check_creditnum`

3.2.3 Constraints enforced in the frontend

Below contains application constraints enforced by server/client side of our app

S/N	Constraints enforced	Application handling
1	Each caretaker is capable of taking care of at least one category of pets. Caretakers can only take care of the types he/she is capable of caring for.	When a new caretaker registers, he/she is required to specify at least one capable type. New capable type adding is allowed afterwards. When searching for caretakers to bid, checks for capability matching are also conducted.
2	The transfer method must be one of delivery, pick up, or physical building transfer. The payment method must be either card or cash. The category of a pet can only be chosen from a list.	Not enforced in relational schema; instead, the frontend limits the choices for these scenarios using radio buttons and drop down lists..

3.3 Non-trivial & interesting triggers

No.	Trigger code	Explanation

1.	CREATE OR REPLACE FUNCTION update_rating()
	RETURNS TRIGGER AS $$
	DECLARE rates NUMERIC;
	DECLARE ptype VARCHAR;
	BEGIN IF NEW.rating IS NULL THEN RETURN NEW;
	ELSE
	SELECT AVG(rating) INTO rates
	FROM bids B
	WHERE NEW.ctemail = B.ctemail;
	UPDATE caretaker
	SET rating = rates
	WHERE NEW.ctemail = email;
	SELECT P.type INTO ptype
	FROM pet P
	WHERE P.name = NEW.name
	AND P.poemail = NEW.poemail;
	UPDATE capable
	SET dprice = (
	SELECT C.bprice
	FROM category C
	WHERE type = ptype
	) * (1 + rates / 100)
	WHERE NEW.ctemail = ctemail
	AND type = ptype;
	IF rates < 4 THEN
	UPDATE caretaker
	SET maxpetnum = 2
	WHERE NEW.ctemail = email
	AND timetype = 'part time';
	ELSE
	UPDATE caretaker
	SET maxpetnum = (
	CASE
	WHEN rates < 4.3 THEN 3
	WHEN rates < 4.6 THEN 4
	ELSE 5
	END
	)
		The trigger is run to update the rating of the care taker after the pet owner adds a rating, as well as the daily price for the specific type of pet and max number of pets can take care of (if is part time care taker.)
		If a rating is not updated, do nothing.
		Else calculate average of rating given to the care taker across all bids and update in the care taker table.
		
		Find the type of pet for this bid.
		
		For that type of pet, retrieve base price from category table, calculate new daily price given new rating and update the daily price the care taker charge for that type of pet.
		
		If given care taker is part time employee, update the maximum number of pets care taker can take care of at given time accordingly.

	WHERE NEW.ctemail = email
	AND timetype = 'part time';
	END IF;
	RETURN NEW;
	END IF;
	END;
	$$ LANGUAGE plpgsql;
	CREATE TRIGGER update_rate
	AFTER
	UPDATE ON bids FOR EACH ROW EXECUTE PROCEDURE
	update_rating();	

2.	CREATE OR REPLACE FUNCTION check_available_before_insert() RETURNS TRIGGER AS $$ DECLARE available BOOLEAN; DECLARE available_2 BOOLEAN; DECLARE mail VARCHAR; DECLARE timetype VARCHAR; DECLARE max INT; DECLARE count INT; DECLARE capable BOOLEAN; BEGIN SELECT s.email, s.timetype, s.maxpetnum into mail, timetype, max FROM caretaker as s WHERE email = NEW.ctemail; SELECT s.at IS NULL available FROM caretaker_cares_at as s WHERE s.ctemail = mail AND s.at BETWEEN NEW.startDate AND NEW.endDate GROUP BY s.at HAVING count(*) >= max; SELECT COUNT(*) into count FROM available as s WHERE s.ctemail = mail AND s.avl = true AND s.at BETWEEN NEW.startDate AND NEW.endDate; SELECT count = NEW.endDate - NEW.startDate	The trigger is run when the pet owner makes a bid and before the bid is inserted into the table. It checks if the caretaker is available for the period in bid. If the caretaker is available and full-time, the bid will be set to successful and caretaker_cares_at table will be updated accordingly. If the caretaker is available and part-time, the bid will be pending. If the caretaker is not available, the bid will be set to unsuccessful. Finds caretaker attributes: email, time type, maximum pet number for use later. Finds if there is any day (in the caretaker_cares_at table) between start and end date of the bid where the caretaker is taking care of more than or equal to the max number of pets. If so, *available* will be false. If there is no such a day, *available* will be true. Count the number of available days between the start and end date of the bid indicated in the available table.

	+ 1 into available_2;
	
	SELECT COUNT(*) IS NOT NULL into capable
	FROM capable c
	WHERE c.ctemail = mail AND c.type = (SELECT type FROM pet WHERE pet.name = NEW.name);
	
	IF NOT available OR NOT available_2 OR NOT capable THEN
	NEW.success = false;
	RETURN NEW;
	ELSIF available AND available_2 AND capable AND timetype = 'part time' THEN
	NEW.success = null;
	RETURN NEW;
	ELSIF available AND available_2 AND capable AND timetype = 'full time' THEN
	NEW.success = true;
	INSERT INTO caretaker_cares_at
	SELECT NEW.ctemail, NEW.startDate + i, NEW.poemail, NEW.name
	FROM generate_series(0, (select NEW.endDate - NEW.startDate)) i;
	RETURN NEW;
	ELSE
	RAISE exception 'unexpected behaviour % % % %', available, available_2, count, timetype
	USING hint = 'please check if timetype is "part time" or "full time"';
	END IF;
	END
	$$ LANGUAGE plpgsql;
	
	CREATE TRIGGER check_available_before_insert
	BEFORE
	INSERT ON bids FOR EACH ROW EXECUTE PROCEDURE check_available_before_insert();
		If the number of available days is exactly the number of days from start to end date (inclusive), *available_2* is true.
		
		Checks if the caretaker is capable of taking care of this pet. If so, *capable* is true.
		
		If the bid does not satisfy either of the 3 conditions: caretaker not taking care of the max number of pets on all days in bid (*available*), caretaker indicated availability for all days in bid (*available_2*), caretaker capable of taking care of this type of pet (*capable*), bid will be automatically rejected and set to unsuccessful.
		
		If the bid satisfies all 3 conditions and the caretaker is part-time, bid will be pending and its success status is null.
		
		If the bid satisfies all 3 conditions and caretaker is full-time, bid will be accepted and set to successful. A row will be created for each day between the start and end date of bid (inclusive) under the caretaker and the pet in bid and inserted to caretaker_cares_at table.
		
		If the bid satisfies all 3 conditions but does not satisfy either time type, an exception will be raised.
		
		Trigger will run before inserting this bid to bids table.

3.	CREATE OR REPLACE FUNCTION updatePets()
	RETURNS TRIGGER AS $$
	DECLARE
	  d date;
	  c NUMERIC;
	  e NUMERIC;
	BEGIN
	  IF NEW.success = TRUE AND OLD.success = FALSE THEN
	  d := NEW.startDate;
	  WHILE d <= NEW.endDate
	  LOOP
	  SELECT COUNT(*) , MAX(caretaker.maxpetnum) INTO c, e
	  FROM caretaker_cares_at INNER JOIN caretaker ON caretaker_cares_at.ctemail = caretaker.email
	  WHERE caretaker_cares_at.ctemail = NEW.ctemail AND caretaker_cares_at.at = d;
	
	  IF (c >= e) THEN
	  UPDATE available
	  SET avl = FALSE
	  WHERE available.at = d AND available.ctemail = NEW.ctemail;
	
	  UPDATE bids
	  SET success = FALSE
	  WHERE bids.ctemail = NEW.ctemail AND d BETWEEN bids.startDate AND bids.endDate;
	  END IF;
	
	  d := d + interval '1 day';
	  END LOOP;
	
	  END IF;
	  RETURN NEW;
	END;
	$$ LANGUAGE plpgsql;
	
	CREATE TRIGGER updatePets
	  AFTER UPDATE ON bids
	  FOR EACH ROW
	  EXECUTE FUNCTION updatePets();
		The trigger checks for the number of pets a caretaker has to take care of after a successful bid for each day
		
		If the total number exceeds the current limit for a particular day, the trigger:
		
		1) Sets the availability of the caretaker on that day to false
		
		2) Automatically rejects any bids with periods that overlap with this day for the caretaker..

3.4 Complex Queries

No.	Query Code	Explanation
1.	CREATE OR REPLACE FUNCTION check_avl(s DATE, e DATE, mail VARCHAR) RETURNS BOOLEAN AS $$ DECLARE free BOOLEAN; BEGIN free := TRUE;	This query aims to find all care takers who can take care of a type of pet and is free from a start date to end date, ranked by rating and then daily price.

```
IF date(s) > date(e) THEN RETURN FALSE;
ELSE WHILE(
   date(s) <= date(e)
   AND free
) LOOP IF EXISTS (
   SELECT *
   FROM available A
   WHERE A.ctemail = mail
       AND A.at = date(s)
       AND A.avl
) THEN s := date(s) + 1;
ELSE free := FALSE;
END IF;
END LOOP;
RETURN free;
END IF;
END;
$$ LANGUAGE plpgsql;
CREATE OR REPLACE FUNCTION
get_caretaker_ranked(ptype VARCHAR, start_d
DATE, end_d DATE) RETURNS TABLE(
       ctname VARCHAR,
       rating NUMERIC(5, 2),
       price NUMERIC(5, 2)
   ) AS $$ BEGIN RETURN QUERY
SELECT CT.cname,
   CASE
       WHEN CT.rating IS NULL THEN -1
       ELSE CT.rating
   END,
   CA.dprice
FROM caretaker CT
   LEFT JOIN capable CA ON CT.email =
CA.ctemail
WHERE CA.type = ptype
   AND (
       SELECT check_avl(start_d, end_d,
CT.email)
   )
ORDER BY CT.rating DESC, CA.dprice ASC;
```

Function check_avl takes in a start date, end date, and email of care taker. If start date is after end date, return false.

If there is an entry with given care taker email on specific date with avl set to true in the available table, the care taker is available on that day.

If there is no such entry, the loop is stopped and function returns false.

If such an entry exists from start date to end date, both inclusive, function returns true.

Function get_caretaker_ranked takes in pet type, start date and end date. Returns name of all care takers that can take care of that pet type, rating of care taker and daily price they charge.

If the care taker does not have a rating yet, return -1 for rating.

Carry out caretaker CT LEFT JOIN capable CA ON CT.email = CA.ctemail. Return the row only when that care taker can take care of the pet type and is available from start to finish.

Rank the results first by rating from highest to lowest. For care takers with same ratings, rank them by price from lowest to highest.

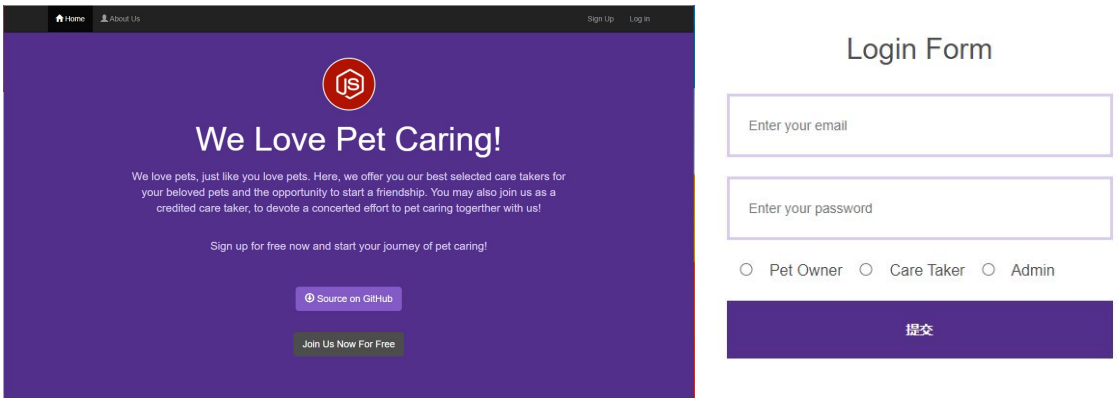
	END; $$ LANGUAGE plpgsql;	

2.	CREATE OR REPLACE FUNCTION calc_salary(mail VARCHAR, month INT, year INT) RETURNS NUMERIC AS $$
	DECLARE salary NUMERIC;
	DECLARE ttype VARCHAR;
	BEGIN
	SELECT timetype into ttype FROM caretaker WHERE email = mail;
	IF ttype = 'part time' THEN
	SELECT sum(b.price * (b.endDate - b.startDate + 1) * 0.75) into salary
	FROM bids as b
	WHERE b.ctemail = mail and b.success = true and EXTRACT (MONTH FROM b.endDate) = month and EXTRACT (YEAR FROM b.endDate) = year;
	ELSIF ttype = 'full time' THEN
	SELECT CASE WHEN SUM(bonus.price) IS NULL then 3000
	ELSE SUM(bonus.price * 0.8) + 3000
	END into salary
	FROM (SELECT DISTINCT c.ctemail, c.at, c.pet_owner, c.pet_name, b.price
	FROM caretaker_cares_at as c JOIN bids as b on b.ctemail = c.ctemail and b.success and b.poemail = c.pet_owner and b.name = c.pet_name and c.at BETWEEN b.startDate AND b.endDate
	WHERE c.ctemail = mail and EXTRACT (MONTH FROM c.at) = month and EXTRACT (YEAR FROM c.at) = year
	OFFSET 60) as bonus;
	ELSE
	return null;
	END IF;
		This query calculates salary of a caretaker in that given month.
		
		Getting the time type of the caretaker and storing into *ttype*.
		If caretaker is part-time, the salary will be bid price (daily) * number of days in bid * 0.75. Bids are taken as bids that ended in that onth and year.
		
		If caretaker is full-time, the salary is 3000 + bonus. Bonus is the total number of pet days excluding the first 60 pet days multiplied by the daily price in the bid. If the caretaker worked less than or equal to 60 pet days, bonus will be 0. Bids are bids that ended in that month and year too.

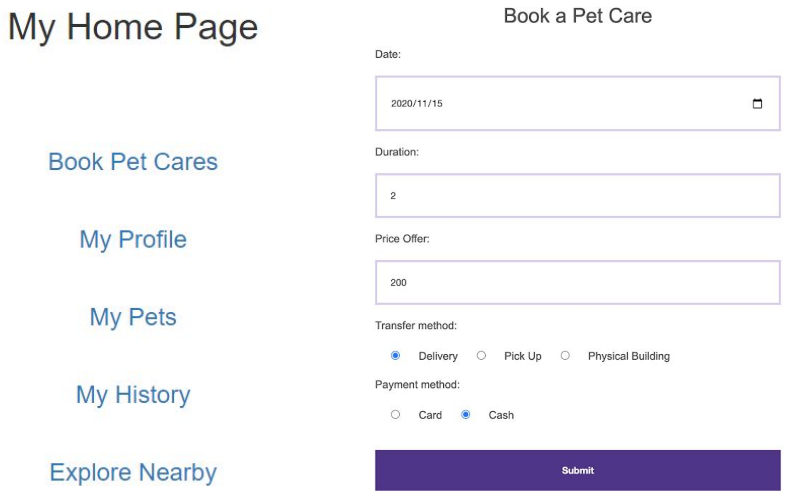
	RETURN salary; END; $$ LANGUAGE plpgsql;	
3.	CREATE OR REPLACE FUNCTION statistics(month INT, year INT) RETURNS table(ctname VARCHAR, ctemail VARCHAR, total BIGINT, petdays BIGINT, salary NUMERIC) LANGUAGE plpgsql AS $$ BEGIN RETURN QUERY SELECT caretaker.cname, aretaker.email, count(*), count(a.distinct_dates), calc_salary(caretaker.email, month, year) FROM (SELECT DISTINCT caretaker_cares_at.at AS distinct_dates, caretaker_cares_at.ctemail AS ct FROM caretaker_cares_at) AS a, caretaker INNER JOIN caretaker_cares_at ON caretaker.email = caretaker_cares_at.ctemail WHERE EXTRACT(MONTH FROM caretaker_cares_at.at) = month AND a.ct = caretaker.email GROUP BY caretaker.email ORDER BY total; END;$$;	This function returns the statistics for the month, it is used by pcsadmin. It returns a table containing information about the: 1) Total number of pets cared for 2) Total number of workdays this month 3) Salary of the caretaker this month The total number of pets is calculated by counting the number of entries the caretaker has in the caretaker_cares_at table The number of pet days is calculated by counting the number of entries with distinct dates, since a caretaker can take care of more than one pet per day The salary is calculated using the function above. This function helps sieve out underperforming caretakers.

4.0 Summary

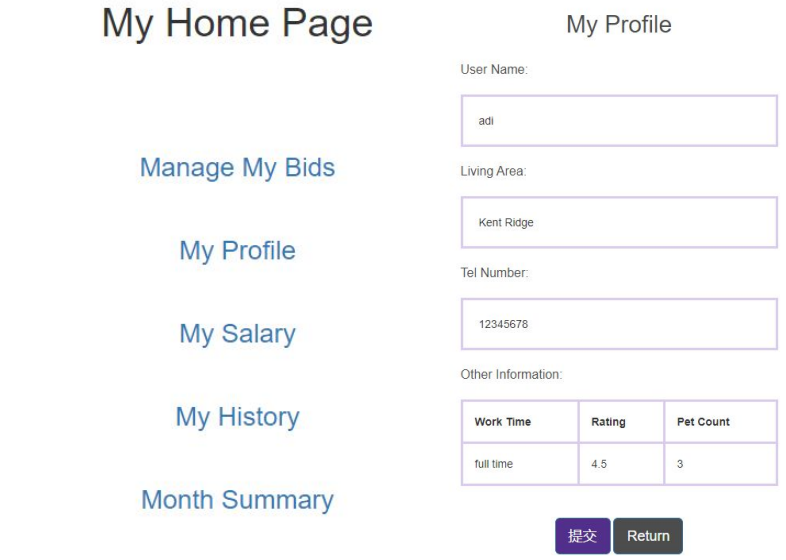
Screenshot Demo



Website index & login form



Pet Owner: homepage & book pet cares



Care Takers: homepage, profile and bid management

PCSadmin caretaker monthly salary and pet days lookup

Looking Back

When attempting this project, we came across many challenges but also picked up some skills in web development and writing SQL queries. One of the first challenges we faced was expressing the design of the system in ER diagrams. As much as we can grasp what is needed as constraints to make the application functional, it is much easier writing out than using the ER diagram which we were not very familiar with then. Our first iteration of the ER diagram did not manage to capture all the constraints accurately, not to mention some constraints were flawed or unclear too. Since then, we have reviewed our constraints to make sure it is coherent and adjusted our ER diagram accordingly.

Most of the group were inexperienced in web development too. It was challenging for us to pick up how to use frameworks such as node.js in a short period. Fortunately, the more experienced members in the group were able to spearhead the process of web development so we can start quite early.

There were some disheartening moments where our queries and triggers are not working as expected when we tested them out but in the end we were able to resolve the issues. We believe what we learnt from debugging would be useful in our future SQL writing.

Overall, working on this project is a valuable experience and we learnt a lot along the way. It has definitely made us more familiar with the usage of databases and web development.

Tools & Frameworks

1. Web Server: https://www.heroku.com/
    a. Database service: heroku-postgresql
    b. Plan hobby-dev
    c. Version: 12.4
2. Main framework: nodeJS
    a. NPM version:

```
{
        'cs2102-2021-s1-team56': '1.0.0',
        npm: '6.14.4',
        ares: '1.15.0',
        brotli: '1.0.7',
        cldr: '35.1',
        http_parser: '2.8.0',
        icu: '64.2',
        llhttp: '2.0.1',
        modules: '72',
        napi: '5',
        nghttp2: '1.40.0',
        node: '12.14.1',
        openssl: '1.1.1d',
        tz: '2019c',
        unicode: '12.1',
        uv: '1.33.1',
        v8: '7.7.299.13-node.16',
        zlib: '1.2.11'
}
```

    b. Dependencies:
        i. express@4.17.1
        ii. express-session@1.17.1
        iii. body-parser@1.19.0
        iv. ejs@2.7.4
3. Miscellaneous tools:
    a. ER drawer: https://app.diagrams.net/
    b. Scheme visualizer: DBeaver; Version: 7.0.2