NUS
National University
of Singapore

CS2102 Database Systems
Academic Year 2020/21 Semester 1
Project Report - Pet Caring Service

Team 24
Team Members:

Name	Matriculation No.
Goh Chang Wei, Levron	A0201530A
Mok Xiao Fan	A0202399B
Kenny Hermawan	A0200758H
Justin Tzuriel Krisnahadi	A0200794H
Theodoric Keith Lim	A0202057U

Submitted on: 7 November 2020

1. Project Responsibilities

Name	Responsibility

Goh Chang Wei, Levron	• CareTaker Pages:
	  ○ View Profile
	  ○ View Pet type and pricing
	  ○ View all reviews of caretaker
	  ○ View current jobs of caretakers
	  ○ View Full time leaves
	  ○ View Part time availability
	• Queries
	• Report
	• ERD

Mok Xiao Fan	• Admin pages:
	  ○ Creation, update and delete of admin profiles
	  ○ View user profiles (caretaker/pet owner)
	  ○ View leave of full timers
	  ○ View salary details of caretakers
	  ○ View job details
	  ○ Admin page dashboard (summary information)
	• Triggers
	• Queries
	• Report
	• ERD

Kenny Hermawan	• Sessions and Authentications
	  ○ Authentications
	  ○ Authentication-related middlewares
	• Pet Owner
	  ○ Sign in and sign out
	• Caretaker
	  ○ Caretaker activation
	  ○ Caretaker home page
	• Administrator
	  ○ Sign in and sign out
	  ○ Create new administrators
	• Triggers
	• Queries
	• Report
	• ERD

Justin Tzuriel Krisnahadi	• Sign up page
	  ○ Create new app users
	• CareTaker search
	  ○ Filter by dates and categories
	  ○ View caretaker details

	○ Confirm bidding details • Queries • Report • ERD
Theodoric Keith Lim	• PetOwner pages ○ View profile ○ View pets under PetOwner ○ View reservations under PetOwner • SQL Base • Queries • Report • ERD

2. Application Functionality

The Pet Caring Service application enables pet owners in finding a caretaker for their pets. Pet owners may perform searching by category of pets and by the availability of an individual caretaker. A pet owner has to be signed in to be able to bid for the chosen caretaker's service. A guest may still search for caretaker services but is required to register if he/she wishes to bid for services. All registered users are pet owners and can also choose to become a caretaker. A pet owner can register their pets and view their details afterwards. Special requirements may also be included to aid the caretaker in attending to the pet's needs.

A caretaker holds a set of ratings that is calculated from the jobs completed. Pet owners may provide ratings and reviews based on the caretaker's services. These ratings will influence the caretaker's average rating that in turn determines the price of the caretaker services. Additionally, caretakers are also categorised into either full-time or part-time. The full-time caretakers may apply for leave provided they have not applied for a leave for a minimum of 2x150 consecutive days in that work year.

A caretaker may take on any number of jobs across a period of time. A caretaker may also perform repeated jobs on the same client. The pet owner may view the job's status within his/her profile. The job status will be reflected as either CONFIRMED, COMPLETED, or REVIEWED. The pet owner may provide a review for the caretaker once the job's status has been set to DONE by the petowner.

Administrators can sign in through a separate page. Administrators are able to view summarised information of the current month. The dashboard provides information on the total number of pets cared for, along with the top 10 and bottom 10 caretakers for the month. To ease management of finances, the total salary payable for the month is also displayed. The company's performance for the year may also be monitored through a graph that is plotted at the side. Administrators can also view the list of users, jobs and profiles of other administrators. Additionally, the administrator is able to register other new administrators.

3. Data Constraints

PetOwner

- PetOwners are uniquely identified by a username.
- A PetOwner's name, email, password, address, gender, dateofbirth, joindate and isactive must be recorded.
- A PetOwner's email must be unique.
- A PetOwner can own any number of Pets.
- A PetOwner can register any number of CreditCards.

CareTaker

- A CareTaker is a PetOwner (IS-A relationship, not covering).
- A CareTaker can be either FullTime or PartTime but not both (covering, no overlap).
- A CareTaker's avgrating must be recorded (Initial value of a CareTaker's avgrating is always 0).
- A CareTaker takes care of a PetOwner's Pet by confirming a Job.
- A CareTaker can take care of the same Pet multiple times through different Jobs.
- A CareTaker can have any number of Jobs.
- A CareTaker can cater to any number of PetCategory.
- A CareTaker will have the price of each PetCategory they cater to recorded.

FullTime

- A FullTime can apply for multiple Leaves.
- A FullTime can only apply for Leaves on days that are later than today.
- A FullTime can only apply for Leaves only if they have not applied for a leave for a minimum of 2x150 consecutive days in the current year.

PartTime

- A PartTime has multiple Availabilities.

Pet

- A Pet is uniquely identified among the other Pets of the same PetOwner by its name.
- A Pet's age, gender, description, and personality must be recorded.
- A Pet's specialreq may be null.
- A Pet is owned by exactly one PetOwner.
- A Pet cannot exist without a PetOwner.
- A Pet falls under exactly one PetCategory.

Job

- A Job is uniquely identified among other Jobs with the same CareTaker, PetOwner, and Pet by its startdate.
- A Job's deliverytype, paymenttype, enddate, requestdate, amountpaid, status must be recorded.
- A Job's rating and review may be null.
- A Job must belong to one CareTaker, one PetOwner, and one Pet.
- A Job cannot exist without the corresponding CareTaker, PetOwner, and Pet.

PetCategory

- A PetCategory is uniquely identified by its category.
- A PetCategory's baseprice must be recorded.
- A PetCategory can belong to any number of Pets.
- A PetCategory can be catered by any number of CareTakers.

CreditCard

- A CreditCard is uniquely identified by its cardnumber.
- A CreditCard's fullname, cvv, and expirydate must be recorded.
- A CreditCard must belong to exactly one PetOwner.
- A CreditCard cannot exist without a PetOwner.

Salary

- A Salary is uniquely identified among the other Salaries of the same CareTaker from its salarydate.
- A Salary's totalamount must be recorded.
- A Salary belong to exactly one CareTaker.
- A Salary cannot exist without its corresponding CareTaker.

Availability

- An Availability is uniquely identified among the other Availabilities of the same PartTime from its startdate.
- An Availability needs to record the enddate as well.
- An Availability can only belong to exactly one PartTime.
- An Availability cannot exist without its corresponding PartTime.

Leave

- A Leave is uniquely identified among the other Leaves of the same FullTime from its leavedate.
- A Leave belong to exactly one FullTime.
- A Leave cannot exist without its corresponding FullTime.

4. Other Constraints (Business Logic)

Job

- There are 3 deliverytype values for a job: POD (Pet Owner Delivery), CTP (Caretaker Pickup), or PTB (Physical Transfer in Building).
- There are 3 status values for a job: CONFIRMED, COMPLETED, or REVIEWED.
- Only the PetOwner can change the status of the job.
- A PetOwner must pay the CareTaker upfront with either credit card or cash.
- PetOwners can post only one review and one rating for one specific Job.
- A PetOwner may submit multiple reviews and ratings for a CareTaker if the CareTaker has taken care of the PetOwner's Pet multiple times.
- Reviews are available to all PetOwners.

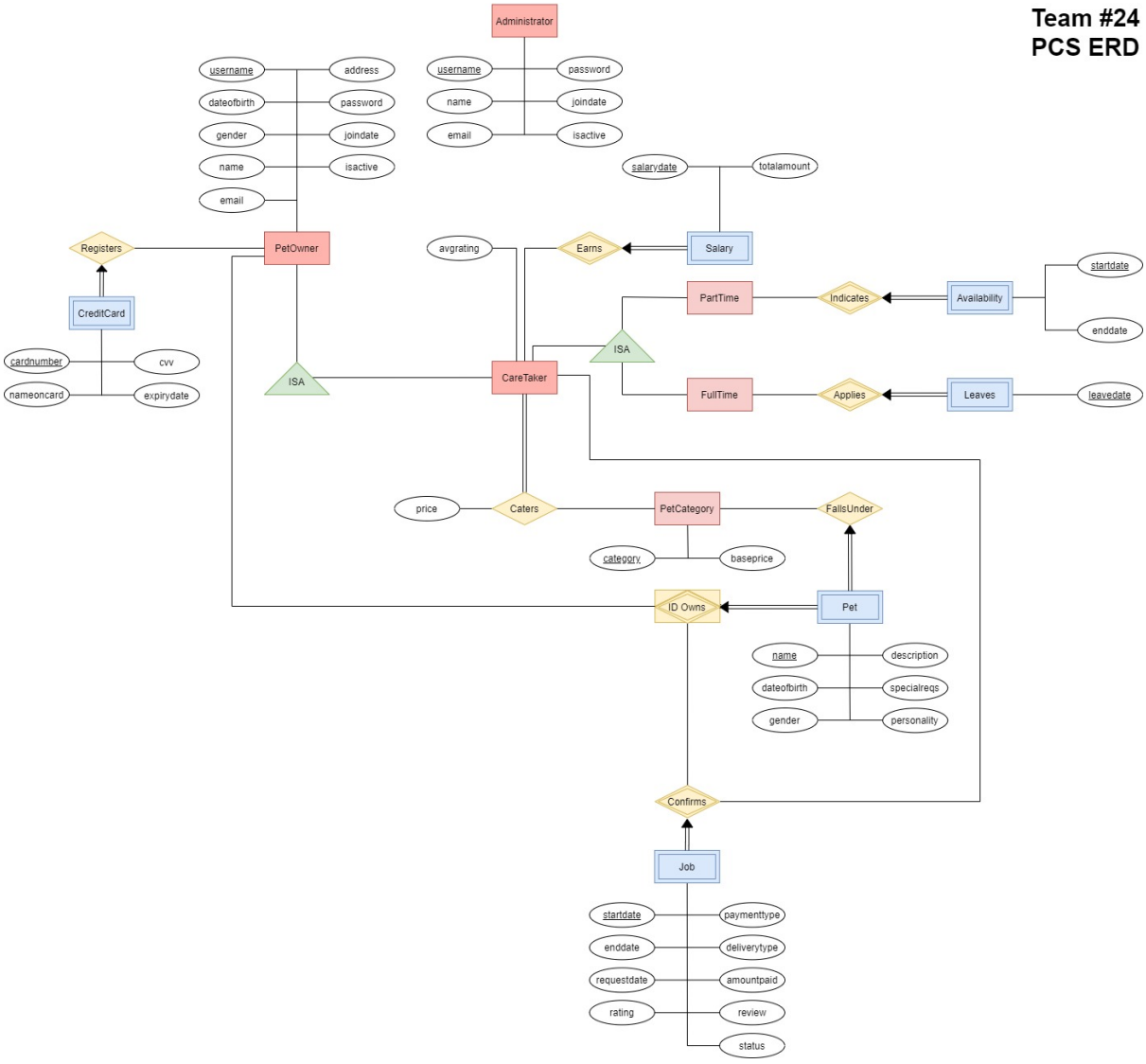
CareTaker

- A CareTaker only caters for Pets in their PetCategories.
- A CareTaker's price for each PetCategory they cater to is calculated based on their avgrating.

FullTime

- A FullTime is deemed available unless they apply for a Leave.

5. Entity Relationship Diagram (ERD)



ERD Non-Trivial Design Decisions

- Any user of the app is by default a PetOwner.
- A CareTaker is a PetOwner.
- ID Owns is an aggregate of PetOwner and Pet, in order to enforce that jobs are for Pets that the PetOwner owns.
- The Confirms relationship set is owned by the ID Owns and CareTaker entity sets.
- Job, Pet, Leaves, Availability, Salary are weak entities with identity dependencies.
- CreditCard is a weak entity with an existential dependency.

6. Schema

Entity-set/Relationship-set	Schema

Administrator	`CREATE TABLE Administrator (`
	`  username VARCHAR(20),`
	`  name VARCHAR(100) NOT NULL,`
	`  email VARCHAR(100) NOT NULL UNIQUE,`
	`  password VARCHAR(20) NOT NULL,`
	`  joindate DATE NOT NULL,`
	`  isactive BOOLEAN NOT NULL,`
	`  PRIMARY KEY(username)`
	`);`

Pet Owner	`CREATE TABLE PetOwner (`
	`  username VARCHAR(20),`
	`  name VARCHAR(100) NOT NULL,`
	`  email VARCHAR(100) NOT NULL UNIQUE,`
	`  password VARCHAR(20) NOT NULL,`
	`  joindate DATE NOT NULL,`
	`  isactive BOOLEAN NOT NULL,`
	`  gender VARCHAR(1) NOT NULL,`
	`  address VARCHAR(100) NOT NULL,`
	`  dateofbirth DATE NOT NULL,`
	`  PRIMARY KEY(username)`
	`);`

Caretaker	`CREATE TABLE CareTaker (`
	`  username VARCHAR(20),`
	`  avgrating NUMERIC(2,1) NOT NULL,`
	`  PRIMARY KEY(username),`
	`  FOREIGN KEY(username) REFERENCES PetOwner(username)`
	`);`

Salary (Caretaker)	`CREATE TABLE CareTakerEarnsSalary (`
	`  username VARCHAR(20),`
	`  salarydate DATE NOT NULL,`
	`  totalamount NUMERIC(31, 2) NOT NULL,`
	`  PRIMARY KEY(username, salarydate),`
	`  FOREIGN KEY(username) REFERENCES CareTaker(username) ON DELETE CAS`
	`CADE`
	`);`

Full Time	`CREATE TABLE FullTime (`
	`  username VARCHAR(20),`
	`  PRIMARY KEY(username),`
	`  FOREIGN KEY(username) REFERENCES CareTaker(username)`
	`);`

Leaves (Full Time)	`CREATE TABLE FullTimeAppliesLeaves (`
	`  username VARCHAR(20),`
	`  leavedate DATE,`
	`  PRIMARY KEY(username, leavedate),`
	`  FOREIGN KEY(username) REFERENCES FullTime(username) ON DELETE CASC`
	`ADE`
	`);`

Part Time	`CREATE TABLE PartTime (`
	`  username VARCHAR(20),`
	`  PRIMARY KEY(username),`
	`  FOREIGN KEY(username) REFERENCES CareTaker(username)`

	);

Availability (Part Time)	CREATE TABLE PartTimeIndicatesAvailability (
	  username VARCHAR(20),
	  startdate DATE NOT NULL,
	  enddate DATE NOT NULL,
	  PRIMARY KEY(username, startDate, endDate),
	  FOREIGN KEY(username) REFERENCES PartTime(username) ON DELETE CASC
	ADE
	);

Credit Card	CREATE TABLE PetOwnerRegistersCreditCard (
	  username VARCHAR(20),
	  cardnumber VARCHAR(20) UNIQUE,
	  nameoncard VARCHAR(100) NOT NULL,
	  cvv VARCHAR(20) NOT NULL,
	  expirydate DATE NOT NULL,
	  PRIMARY KEY(username, cardnumber),
	  FOREIGN KEY(username) REFERENCES PetOwner(username) ON DELETE CASC
	ADE
	);

Pet Category	CREATE TABLE PetCategory (
	  category VARCHAR(20),
	  baseprice NUMERIC(31,2) NOT NULL,
	  PRIMARY KEY(category)
	);

Caters	CREATE TABLE CareTakerCatersPetCategory (
	  username VARCHAR(20) NOT NULL,
	  category VARCHAR(20) NOT NULL,
	  price NUMERIC(31,2) NOT NULL,
	  PRIMARY KEY(username, category),
	  FOREIGN KEY(username) REFERENCES CareTaker(username),
	  FOREIGN KEY(category) REFERENCES PetCategory(category)
	);

Pet	CREATE TABLE Pet (
	  username VARCHAR(20),
	  name VARCHAR(50),
	  dateofbirth DATE NOT NULL,
	  gender VARCHAR(1) NOT NULL,
	  description VARCHAR(100) NOT NULL,
	  specialreqs VARCHAR(100),
	  personality VARCHAR(100) NOT NULL,
	  category VARCHAR(20) NOT NULL,
	  PRIMARY KEY(username, name),
	  FOREIGN KEY(username) REFERENCES PetOwner(username),
	  FOREIGN KEY(category) REFERENCES PetCategory(category)
	);

Job	CREATE TABLE Job (
	  ctusername VARCHAR(20),
	  pousername VARCHAR(20),
	  petname VARCHAR(20),
	  startdate DATE,
	  enddate DATE NOT NULL,
	  requestdate TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
	  status VARCHAR(10) DEFAULT 'CONFIRMED' NOT NULL,
	  rating NUMERIC(2,1),
	  paymenttype VARCHAR(20) NOT NULL,
	  deliverytype VARCHAR(20) NOT NULL,
	  amountpaid NUMERIC(31,2) NOT NULL,

```
  review VARCHAR(1000),
  PRIMARY KEY(pousername, ctusername, petname, startdate),
  FOREIGN KEY(pousername, petname) REFERENCES Pet(username, name),
  FOREIGN KEY(ctusername) REFERENCES CareTaker(username),
  CHECK(pousername != ctusername),
  CHECK(startdate < enddate),
  CHECK(requestdate < enddate)
);
```

7. Database Normal Form

Administrator	**BCNF**
PetOwner	-
CareTaker	BCNF
CareTakerEarnsSalary	BCNF
FullTime	BCNF
FullTimeAppliesLeaves	BCNF
PartTime	BCNF
PartTimeIndicatesAvailability	BCNF
PetOwnerRegistersCreditCard	1NF
PetCategory	BCNF
CareTakerCatersPetCategory	BCNF
Pet	3NF
Job	3NF

Most of the tables are already in normal form, because we modelled the schema from the ERD. Each table is based on a particular entity-set/relationship-set. PetOwner is not in normal form because the address attribute is not atomic. PetOwnerRegistersCreditCard is in 1NF because a credit card can already be identified by its card number.

8. Triggers

1

```
CREATE OR REPLACE FUNCTION update_baseprice()
  RETURNS TRIGGER AS
$$
DECLARE
    currbaseprice NUMERIC(31,2);
  newprice NUMERIC(31,2);
  cater CareTakerCatersPetCategory%rowtype;
BEGIN

  FOR cater IN SELECT * FROM CareTakerCatersPetCategory WHERE username = new.username LOOP
    currbaseprice = (SELECT baseprice FROM petcategory WHERE category = cater.category);

    IF new.avgrating = 5.0 THEN
      UPDATE CareTakerCatersPetCategory
      SET price = currbaseprice * 2
      WHERE username = new.username AND category = cater.category;
      RETURN NEW;
    ELSEIF new.avgrating < 5.0 AND new.avgrating >= 4.5 THEN
      UPDATE CareTakerCatersPetCategory
      SET price = currbaseprice * 1.75
      WHERE username = new.username AND category = cater.category;
      RETURN NEW;
    ELSEIF new.avgrating < 4.5 AND new.avgrating >= 4.0 THEN
      UPDATE CareTakerCatersPetCategory
      SET price = currbaseprice * 1.5
      WHERE username = new.username AND category = cater.category;
      RETURN NEW;
    ELSEIF new.avgrating < 4.0 AND new.avgrating >= 3.5 THEN
      UPDATE CareTakerCatersPetCategory
      SET price = currbaseprice * 1.25
      WHERE username = new.username AND category = cater.category;
      RETURN NEW;
    ELSE
      UPDATE CareTakerCatersPetCategory
      SET price = currbaseprice
      WHERE username = new.username AND category = cater.category;
      RETURN NEW;

    END IF;
  END LOOP;
END
$$
LANGUAGE plpgsql;

CREATE TRIGGER update_baseprice
AFTER UPDATE
ON caretaker
FOR EACH ROW
EXECUTE PROCEDURE update_baseprice();
```

When caretaker's average rating is updated, their daily price of each pet category that they can take care of will increase accordingly (increases as their rating is higher)

```
CREATE OR REPLACE FUNCTION calc_job_price()
  RETURNS TRIGGER AS
$$
BEGIN
  IF new.status = 'PENDING' THEN
  new.amountpaid := 0.0;
  ELSEIF new.status = 'CANCELLED' THEN
  new.amountpaid := 0.0;
  ELSE
  new.amountpaid := (date_part('day', new.enddate::timestamp - new.startdate::timestamp)
          * (SELECT price FROM caretakercaterspetcategory WHERE username = new.ctusername AND category
        = (SELECT category FROM pet WHERE username = new.pousername AND name = new.petname)));
  END IF;
  RETURN NEW;
END
$$
LANGUAGE plpgsql;

CREATE TRIGGER calc_job_price
BEFORE INSERT
ON job
FOR EACH ROW
EXECUTE PROCEDURE calc_job_price();
```

When a new job is created, automatically calculate the price of the job based on the number of days and the care taker's pet category's daily price of that pet.

3

```
CREATE OR REPLACE FUNCTION limit_leaves()
    RETURNS TRIGGER AS
$$
DECLARE
  prevdate fulltimeappliesleaves%rowtype;
    prevprevdate DATE;
    lastdate DATE;
    consecdays integer := 0;
BEGIN

    IF new.leavedate < CURRENT_DATE THEN
      RAISE EXCEPTION 'Please select a future date';
    END IF;

    FOR prevdate IN SELECT * FROM fulltimeappliesleaves
        WHERE username = new.username
          AND date_part('year', leavedate) = date_part('year', CURRENT_DATE)
          ORDER BY leavedate DESC LOOP

      prevprevdate = (SELECT leavedate FROM fulltimeappliesleaves
        WHERE username = new.username
          AND leavedate < prevdate.leavedate
          AND date_part('year', leavedate) = date_part('year', CURRENT_DATE)
          ORDER BY leavedate DESC
          LIMIT 1);

      IF prevprevdate != null THEN
        IF date_part('day', prevdate.leavedate - prevprevdate) >= 300 THEN
          consecdays := consecdays + 2;
        ELSEIF date_part('day', prevdate.leavedate - prevprevdate) >= 150 THEN
          consecdays := consecdays + 1;
        END IF;
      END IF;

    END LOOP;
  lastdate := (SELECT leavedate FROM fulltimeappliesleaves
      WHERE username = new.username
        AND leavedate < CURRENT_DATE
        AND date_part('year', leavedate) = date_part('year', CURRENT_DATE)
        ORDER BY leavedate DESC LIMIT 1);

  IF lastdate != null THEN
    IF date_part('day', CURRENT_DATE - lastdate) >= 300 THEN
      consecdays := consecdays + 2;
    ELSEIF date_part('day', CURRENT_DATE - lastdate) >= 150 THEN
      consecdays := consecdays + 1;
    END IF;
  ELSE
    IF date_part('day', new.leavedate - date_trunc('year', new.leavedate))  <= 65 THEN
      consecdays := consecdays + 2;
    ELSEIF date_part('day', date_trunc('year', new.leavedate) + interval '1 year - 1 day' - new.leavedate) <= 65 THEN
      consecdays := consecdays + 2;
    END IF;
  END IF;

  IF consecdays < 2 THEN
    RAISE EXCEPTION 'Invalid date, you need to work for at least 2x150 consecutive days a year.';
  END IF;

  RETURN NEW;
END
$$
LANGUAGE plpgsql;

CREATE TRIGGER limit_leaves
BEFORE INSERT
ON fulltimeappliesleaves
FOR EACH ROW
EXECUTE PROCEDURE limit_leaves();
```

Check whether caretakers are applicable to apply for the leave date they specified, they can only apply if they can work for a minimum of 2x150 consecutive days that year

9. SQL Queries

	SQL Query
1	`search_caretaker: `SELECT * FROM fulltime f JOIN petowner u ON f.username = u.username AND f.username <> $3 JOIN caretaker c ON f.username = c.username JOIN caretakercaterspetcategory cat ON f.username = cat.username AND cat.category = $4 WHERE NOT EXISTS (SELECT leavedate FROM fulltimeappliesleaves WHERE username = f.username AND leavedate >= $1::date AND leavedate <= $2::date) UNION SELECT * FROM parttime p JOIN petowner u ON p.username = u.username AND p.username <> $3 JOIN caretaker c ON p.username = c.username JOIN caretakercaterspetcategory cat ON p.username = cat.username AND cat.category = $4 WHERE EXISTS (SELECT * FROM parttimeindicatesavailability WHERE username = p.username AND startdate <= $1::date AND enddate >= $2::date)` Searches for caretakers for a specific pet category within a given time frame, excluding those who are unavailable.
2	`underperforming_ct: `SELECT job.ctusername AS username, SUM(date_part('day', job.enddate::timestamp - job.startdate::timestamp)) AS petdays, SUM(job.amountpaid) AS amountearned, c.avgrating AS rating FROM job INNER JOIN caretaker c ON job.ctusername = c.username WHERE date_part('month',job.startdate) = date_part('month', CURRENT_DATE) AND date_part('year',job.startdate) = date_part('year', CURRENT_DATE) GROUP BY job.ctusername, c.avgrating ORDER BY petdays ASC LIMIT 10;`, Queries for the 10 most underperforming caretakers, based on their pet-days.
3	`job_performance: `SELECT TO_CHAR(TO_DATE(m.month::text, 'MM'), 'Mon') AS month, COALESCE(SUM(job.amountpaid), 0) AS amountpaid FROM generate_series(1,12) AS m(month) LEFT OUTER JOIN job ON date_part('year', job.startdate) = date_part('year', CURRENT_DATE) AND date_part('month', job.startdate) = m.month GROUP BY m.month ORDER BY m.month`, Queries for each month’s performance for the current work year.

10. Specifications

a) Front-End
   - HTML/CSS/JS
   - EJS

b) Back-End
   - NodeJS
   - Express
   - Express-sessions
   - Passport JS

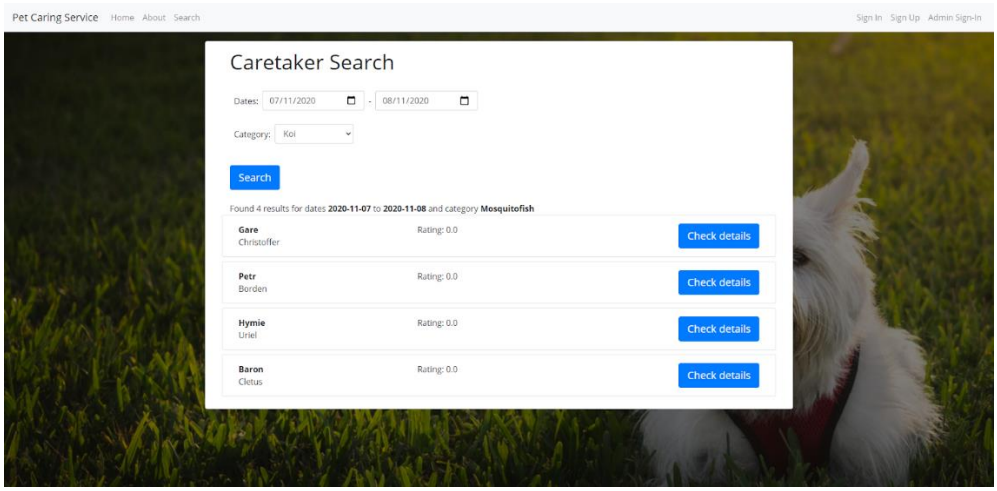
c) Database
   - PostgreSQL (node-pg)

d) Deployment
   - Heroku

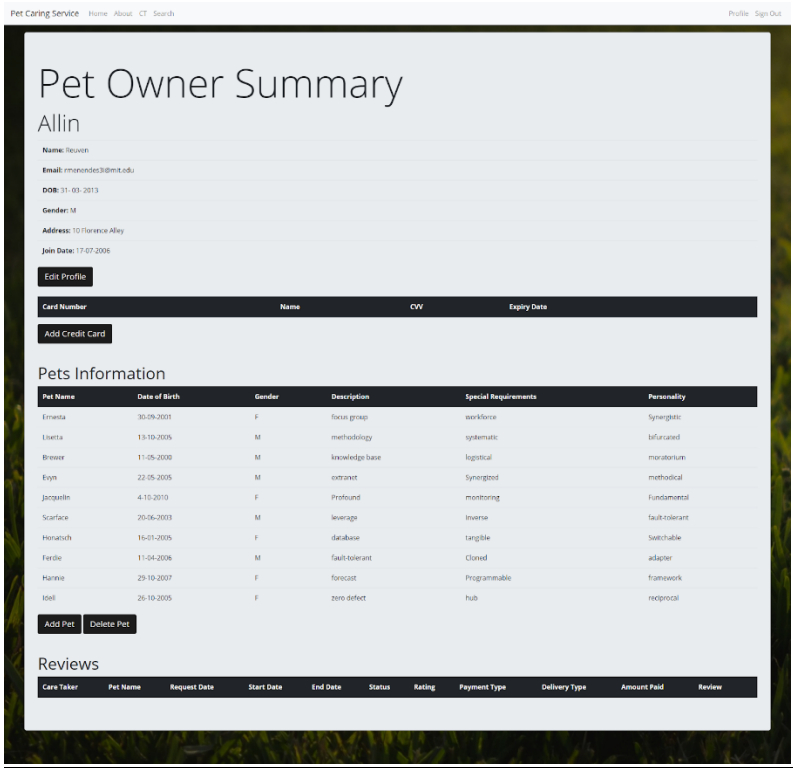
e) Source Control
   - Git/GitHub

11. Application Screenshots

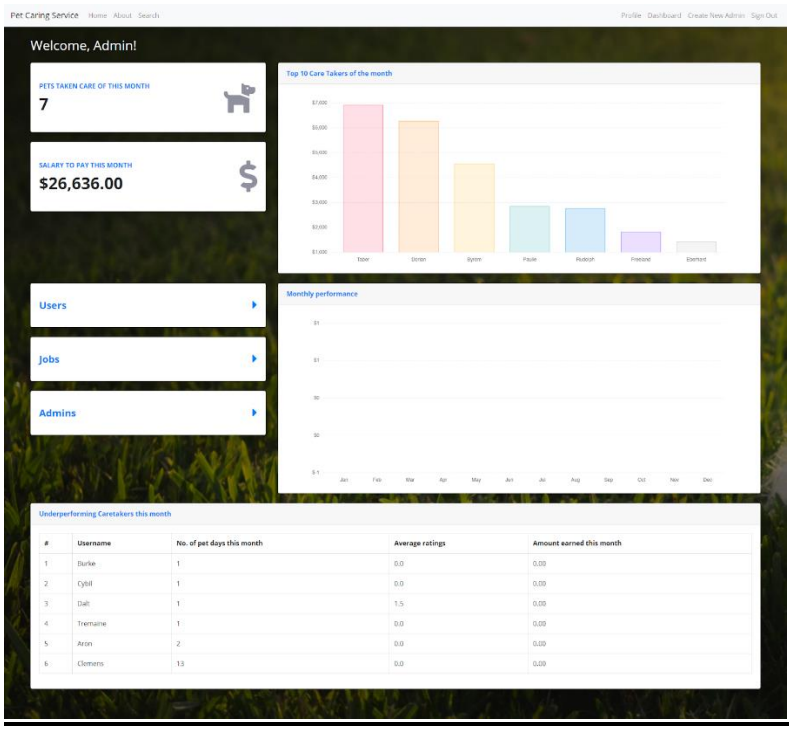
Search for caretakers



Pet owner profile page



Admin dashboard page



12. Summary

Difficulties/challenges we faced

Time management

The project had a great start as we managed to meet all of our personal deadlines. However, as the semester progressed, the project had been side-lined due to the increase in workload. A combination of poor time management and miscommunications had caused our project to come to a stop. A better sense of urgency would have allowed for more time to learn up the web development portion of the project. Thankfully, the project was completed on time with complete features.

Learning a new language

Web development was foreign to a few of us in the group. An example will be the difficulty reconciling the intricacies of how database information is displayed onto a webpage. With languages such as HTML and JS, there was much more to learn and this greatly hindered our progress. This led to some frustration as we were unable to present our results as we had intended due to the lack of familiarity on front-end development with the language at hand.

Miscommunication

This semester is our first full semester where due to COVID-19, we had to stay at home during the entire term. As it is difficult to meet up with each other face to face, there were certain miscommunications from time to time since all interactions were done online. Although we had frequent zoom meetings and were constantly in touch with one another, it is not surprising to have misunderstood one another. This led to some snowballing and double work as we tried to iron out our misunderstandings and work to resolve them. To clear our misunderstandings, we had constantly clarified with each other on our respective parts and constantly updated each other of any changes or clarifications.

Things learnt

Although web development was not the focus of the module, the assignment had exposed us to the Node.js framework, and also HTML and CSS. A high level of proficiency may not have been acquired, but a basic level of exposure has provided us with a glimpse of the challenges and processes of web development. This will definitely be a useful experience for the future.

The flexibility of SQL queries is noteworthy in the course of this project. Simple queries to retrieve all relevant data can be performed with simple conditional WHERE clauses. Whereas more complex functions can be used to allow the extraction of a more selective set of data that would otherwise have to be processed on the client-side.

The project was designed with the Model-View-Controller (MVC) architecture in mind. Having had the opportunity to create a full stack web application, both the front-end and back-end had to be planned out and designed simultaneously to avoid any conflicts. This was by no means an easy task but a great learning experience nonetheless.