CS2102 Project Report

Group 51

E0412957: IVAN LIM DING WANG

E0199469: WONG CHIN KIAT, JOHN

E0323678: NG YEOW HWEE ANTONIO

E0273935: LIONG HONG XIANG

E0273888: ANG CHUN YANG

1. Project responsibilities of each member

Member	Responsibilities
Ang Chun Yang	ER Diagram, writing of http routes, writing triggers, finding out and guiding others on how to use Flask, project report.
Liong Hong Xiang	ER Diagram, writing of http routes, writing triggers, creation of mock data, project report.
Ivan Lim Ding Wang	ER Diagram, writing of http routes, handling of UI, project report.
Wong Chin Kiat, John	ER Diagram, writing of http routes, project report.
Ng Yeow Hwee Antonio	ER Diagram, project report

2. Description of application's data requirements and functionalities

General

- Allow for creation of an account for Care Taker (Full-time or part-time) or pet owner or both.
- Allow for deletion of account.
- Allow for update of password.

Full-time caretaker

- Allow to set their preferred mode of transport and payment during account registration.
- Allow to specify the category of pets he/she wishes to take care. However, the list of pet types that we allow for our app are Dog, Cat, Rabbit, Hamster, Fish, Mice, Terrapin and Bird.
- Allow to view completed/incomplete bids
- Allow to state their availability dates until the end of next year
- Allow to search for the salary received for a particular year and month

Part-time caretaker

- Allow part-time to specify the category of pets he/she wishes to take care and set the price for it. However, the list of categories is fixed.
- Allow to view completed/incomplete bids
- Allow to state their availability dates until the end of next year
- Allow to search for the salary received for a particular year and month

Pet Owner

- Allow to register their pet (pet name, age, category and special consideration). The list of categories to be registered for pet owner to register for their pet is limited. For example, if a pet owner wants to register a pet of a particular category which is not registered in the system, he/she cannot do it.
- Allow to search and bid for Care Taker based on the following preferences:

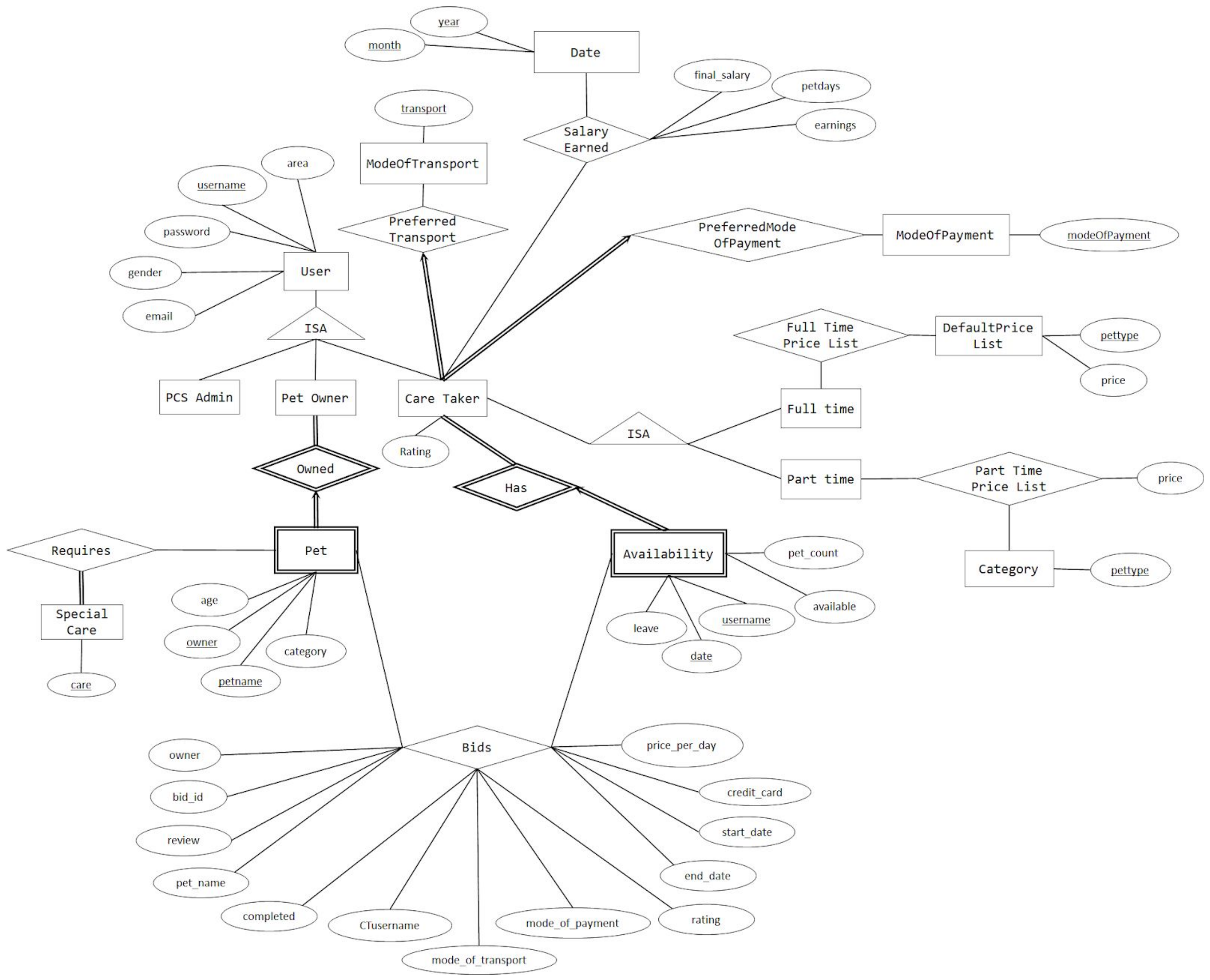
Full-time/part-time, pet type, rating, preferred mode of transport, preferred mode of payment, start date and end date

- Allow to give rating and review for Care Taker after a bid is completed.
- Allow to look for Care Taker living in the same area (Nearby Care Taker)
- Allow to look at completed bid

Application's constraint

- We restricted both the part time and full time Care Takers to only allow them to state their availability dates until the end of the next year, even though in the project requirements, this is only necessary to do so for Part Time Care Takers.
- If a pet owner bids for a caretaker for a particular period, he/she cannot make any subsequent bid for the same pet which has an overlapping time period as the bid made previously.
- We only allow the Part Time Care Takers to set prices for different pet types they want to take care of
- Our bid system runs on a first-come-first-serve basis, meaning a bid is automatically successful if a Care Taker is available and the pet owner has bid for his service. This also means the Care Taker has no choice but to accept the bid when it is made by a Pet Owner.
- Prices in the Price List tables are only the base prices per day. If the Care Taker has a rating of more than or equal to 4.5, the price per day will be the base price multiplied by 1.5. If he has a rating of more than or equal to 4, the price per day will be the base price multiplied by 1.25.

3. ER Model



Non-trivial design decisions

- User is a PCS Admin, pet owner, caretaker, or both a pet owner and caretaker. This is because a user can be both a Care Taker and Pet Owner, which both requires same information.
- A Care Taker can be a Full Time or Part Time Care Taker, which is why we use a ISA relationship.
- Owned Pets is a Weak Entity set as every pet is owned by a pet owner and will be deleted if a pet owner is deleted. Also, a similar pet name can be used for different pet owners. Hence, identity dependency is used.
- Has Availability is a Weak Entity set as the availability information of the Care Taker will be removed if a Care Taker is deleted. Also, multiple Care Taker can be available for the same day.
- Key and total participation between Preferred Transport and Care Taker is because we only allow Care Taker to specify one mode of transport
- Key and total participation between PreferredModeOfPayment is and Care Taker is because we only allow Care Taker to specify one type of payment mode
- Total participation between Special Care

Constraints not captured by our ER Model:

- For the relationship where a User is a PCS Admin or Pet Owner or Care Taker, our ER Diagram does not show that covering constraint is satisfied. Furthermore, it does not show that overlapping constraint is satisfied for Pet Owner and Care Taker where a user can be a Pet Owner or Care Taker
- For the relationship where a Care Taker is a Full Time or Part Time Care Taker, our ER Diagram does not show that covering constraint is satisfied and overlapping constraint is not satisfied.

4. Relational Schema from ER Model

Tables

CREATE TABLE users(
username VARCHAR PRIMARY KEY,
email VARCHAR NOT NULL,
area VARCHAR NOT NULL,
gender VARCHAR NOT NULL,
password VARCHAR NOT NULL
);
	CREATE TABLE PCSAdmin (
	username VARCHAR PRIMARY KEY REFERENCES users(username) ON DELETE CASCADE
	);
		CREATE TABLE PetOwners (
		username VARCHAR PRIMARY KEY REFERENCES users(username) ON DELETE CASCADE
		);

CREATE TABLE CareTakers (
username VARCHAR PRIMARY KEY REFERENCES users(username) ON DELETE CASCADE,
rating NUMERIC DEFAULT 0
);
	CREATE TABLE ModeOfTransport (
	transport VARCHAR PRIMARY KEY
	);
		CREATE TABLE PreferredTransport (
		username VARCHAR PRIMARY KEY REFERENCES CareTakers(username) ON DELETE CASCADE,
		transport VARCHAR REFERENCES ModeOfTransport(transport)
		);

CREATE TABLE ModeOfPayment (
modeOfPayment VARCHAR PRIMARY KEY
);
	CREATE TABLE FullTime (
	username VARCHAR PRIMARY KEY REFERENCES CareTakers(username) ON DELETE CASCADE
	);
		CREATE TABLE PartTime (
		username VARCHAR PRIMARY KEY REFERENCES CareTakers(username) ON DELETE CASCADE
		);

CREATE TABLE PreferredModeOfPayment (
username VARCHAR PRIMARY KEY REFERENCES CareTakers(username) ON DELETE CASCADE,
modeOfPayment VARCHAR REFERENCES ModeOfPayment(modeOfPayment)
);
	CREATE TABLE OwnedPets (
	owner VARCHAR references PetOwners(username) ON DELETE CASCADE,
	pet_name VARCHAR NOT NULL UNIQUE,
	category VARCHAR NOT NULL,
	age INTEGER NOT NULL,
	Primary Key(owner, pet_name)
	);
		CREATE TABLE Category (
		pettype VARCHAR PRIMARY KEY
		);

CREATE TABLE SpecialCare(care VARCHAR PRIMARY KEY	CREATE TABLE RequireSpecialCare(	CREATE TABLE CaretakerAvailability(

```
);
```

```
   owner VARCHAR,
   pet_name VARCHAR,
   care VARCHAR
REFERENCES
SpecialCare(care),
   FOREIGN KEY(owner,
pet_name) REFERENCES
OwnedPets(owner, pet_name) ON
DELETE CASCADE,
   PRIMARY KEY(owner,
pet_name, care)
);
```

```
   date DATE,
   pet_count INTEGER DEFAULT
0,
   leave BOOLEAN DEFAULT
False,
   username VARCHAR
REFERENCES
CareTakers(username) ON
DELETE CASCADE,
   available BOOLEAN NOT
NULL DEFAULT True,
   PRIMARY KEY(username, date)
);
```

```
CREATE TABLE PartTimePriceList
(
  pettype VARCHAR REFERENCES
Category(pettype),
  username VARCHAR
REFERENCES
CareTakers(username) ON DELETE
CASCADE,
  price NUMERIC,
  PRIMARY KEY (pettype,
username)
);
```

```
CREATE TABLE
DefaultPriceList (
  pettype VARCHAR
REFERENCES
Category(pettype),
  price NUMERIC,
  PRIMARY KEY (pettype)
);
```

```
CREATE TABLE
FullTimePriceList(
  username VARCHAR
REFERENCES
CareTakers(username) ON
DELETE CASCADE,
  price NUMERIC,
  pettype VARCHAR,
  FOREIGN KEY (pettype)
REFERENCES
DefaultPriceList(pettype),
  PRIMARY KEY (pettype,
username)
);
```

```
CREATE TABLE
TotalJobPerMonthSummary (
  year INTEGER,
  month INTEGER,
  job_count INTEGER NOT NULL
DEFAULT 0,
  PRIMARY KEY(year, month)
);
```

```
CREATE TABLE CareTakerSalary (
  year INTEGER,
  month INTEGER,
  username VARCHAR REFERENCES CareTakers(username),
  petdays INTEGER NOT NULL DEFAULT 0,
  earnings NUMERIC NOT NULL DEFAULT 0,
  final_salary NUMERIC NOT NULL DEFAULT 0,
  PRIMARY KEY (year, month, username)
);
```

```
CREATE TABLE Bids (
   bid_id INTEGER,
   CTusername VARCHAR,
   owner VARCHAR,
   pet_name VARCHAR,
   FOREIGN KEY(owner, pet_name) REFERENCES OwnedPets(owner, pet_name) ON DELETE CASCADE,
   review VARCHAR DEFAULT NULL,
   rating INTEGER DEFAULT NULL, --to be updated after the bid
   mode_of_transport VARCHAR NOT NULL,
   mode_of_payment VARCHAR NOT NULL,
   credit_card VARCHAR,
   completed BOOLEAN DEFAULT FALSE,
   start_date DATE NOT NULL,
   end_date DATE NOT NULL,
   price_per_day NUMERIC NOT NULL,
   FOREIGN KEY (CTusername, start_date) REFERENCES CareTakerAvailability(username, date) ON
DELETE CASCADE ,
   FOREIGN KEY (CTusername, end_date) REFERENCES CaretakerAvailability(username, date) ON DELETE
CASCADE
);
```

Constraints not enforced by relational schema

- The total participation constraint from **Caretakers** to the **CareTakerAvailability** table is enforced using a trigger called 'insert_into_CareTakerAvailability_after_caretaker_insertion_trigger', which automates the insertion of availability dates into the **CareTakerAvailability** table after the registration of a CareTaker (meaning, upon the successful registration of a caretaker).
- Similarly, the total participation constraint from **Caretakers** to the **CareTakerSalary** table is enforced using a trigger called 'insert_into_salary_after_caretaker_insertion_trigger'.

5. Discussion on whether database is in 3NF or BCNF. Provide justifications for tables that are not in 3NF or BCNF

Tables with non-trivial FDs	FUNCTIONAL DEPENDENCIES	BCNF/ 3NF
users	{username} -> {area, gender, password, email}	BCNF
CareTakers	{username} -> {rating}	BCNF
PreferredTransport	{username} -> {transport}	BCNF
PreferredModeOfPayment	{username} -> {modeOfPayment}	BCNF
OwnedPets	{owner, pet_name} -> {age, category}	BCNF
RequireSpecialCare	{owner, pet_name} -> {care}	BCNF
CareTakerAvailability	{username, date} -> {pet_count, leave, available}	BCNF
Bids	{username, pet_name, owner, start_date} -> {bid_id, review, rating, mode_of_transport, mode_of_payment, credit_card, completed, end_date, price_per_day} {username, pet_name, owner, end_date} -> {bid_id, review, rating, mode_of_transport, mode_of_payment, credit_card, completed, start_date, price_per_day} {bid_id} -> {username, pet_name, owner, end_date, review, rating, mode_of_transport, mode_of_payment, credit_card, completed, start_date, price_per_day}	3NF
PartTimePriceList	{username, pettype} -> {price} Note that the pettype and price attribute here are not the same as the pettype and price attribute for DefaultPriceList and FullTimePriceList.	BCNF

DefaultPriceList	{pettype} -> {price}	BCNF
FullTimePriceList	{username, pettype} -> {price}	BCNF
CareTakerSalary	{username, month, year} -> {final_salary, petdays, earnings}	BCNF
TotalJobPerMonthSummary	{year, month} -> {job_count}	BCNF

Only tables with non-trivial functional dependencies are presented above. Tables not shown above are all in BCNF.
All tables are in BCNF except for bids.
All tables are in 3NF.

6. 3 most interesting triggers

First Trigger: To determine if the Care Taker can take a leave.

Code:

```
CREATE OR REPLACE FUNCTION check_if_fulltime_can_take_leave_function() RETURNS trigger AS $$
DECLARE
  counter INTEGER := 0;
  full_time BOOLEAN;
  consecutive BOOLEAN;
  leave_date_in_non_consecutive BOOLEAN := False;
  date1 DATE;
  date2 DATE;
  year_of_new_date INTEGER := EXTRACT(YEAR FROM NEW.date);
BEGIN
  IF EXTRACT(YEAR FROM NEW.date) > (EXTRACT(YEAR FROM current_date) + 1) THEN
    RAISE EXCEPTION 'You cannot take leave on this date as it is too far away from now';
  END IF;
  SELECT (SELECT MIN(date) FROM CareTakerAvailability CA
        WHERE CA.username = NEW.username
        AND EXTRACT(YEAR from NEW.date) = EXTRACT(year FROM CA.date)) INTO date1;
  date2 := date1 + 149;
  SELECT (NEW.username IN (SELECT username FROM FullTime)) INTO full_time;
  IF (EXTRACT(YEAR FROM NEW.date) = 2020) AND (full_time = TRUE) THEN
    RAISE EXCEPTION 'You cannot take leave on this date';
  END IF;
  IF NEW.date < current_date THEN
    RAISE EXCEPTION 'You cannot take leave on a date that has passed.';
  END IF;
  IF full_time = True THEN
    WHILE EXTRACT(YEAR from date2) < (year_of_new_date + 1) LOOP
      SELECT (150 = (SELECT COUNT(*)
              FROM CareTakerAvailability
              WHERE date >= date1
                AND date <= date2
```

```
            AND NEW.username = CareTakerAvailability.username
            AND leave is False)) INTO consecutive;
      IF consecutive = True THEN
        counter := counter + 1;
        date1 := date1 + 150;
        date2 := date2 + 150;
      ELSE
        date1 := date1 + 1;
        date2 := date2 + 1;
      END IF;
    END LOOP;
    IF counter < 2 THEN
      RAISE EXCEPTION 'You cannot take leave on this date as you don't meet the 2 x 150 consecutive days
requirement';
    END IF;
  END IF;
  RETURN NEW;
END
$$ LANGUAGE 'plpgsql';

DROP TRIGGER IF EXISTS check_if_fulltime_can_take_leave_trigger ON CareTakerAvailability;
CREATE TRIGGER check_if_fulltime_can_take_leave_trigger
  AFTER UPDATE OF leave
  ON CareTakerAvailability
  FOR EACH ROW
  EXECUTE PROCEDURE check_if_fulltime_can_take_leave_function();
```

Explanation:
The role of the trigger is to check if a **Full Time Caretaker** can take a leave or not. If the Care Taker is a Part Time, then the trigger will not affect them and he is definitely allowed to take it as long as the selected leave date is not before the current date. The constraint as to whether he can take a leave is if the date he wants to take a leave on will not violate the work for a minimum 2 x 150 consecutive days in a year as stated in the project requirements. We check if the selected leave date will violate this by finding out if there are no longer 2 x 150 consecutive working days if taken, meaning no 2 x 150 blocks where all the entries leave column is false. We use a while loop to find all possible 150 days blocks, starting from the date1(first date of the year), to date2(the date 149 days later), so there are 150 days between date1 and date2, inclusive of them. Date1 will be the first date of the year, where the year is the year of the date which the caretaker wants to take a leave on. So for example if the caretaker wants to take a leave on '2021-03-21', then date1 is '2021-01-01', and subsequently date2 will be '2021-05-30'. Using this example, if this is not a valid block where the user is not working 150 days consecutively, we find the next block from 2021-01-02 to 2021-05-31, until the end date of the block goes beyond 2021, which ends the loop. If we find a valid block of 150 consecutive working days where he is not taking any leave within this period, we increment the counter by 1. Furthermore, to ensure that the 2 x 150 days blocks do not intersect with each other, we add 150 days to both the first and last date of the current block before continuing on with the loop. For example, if 2021-01-01 to 2021-05-30 is a valid block, we will add 150 days to both 2021-01-01 and 2021-05-30, so the next iteration to find the next valid block will not intersect with the previous one. Finally, if counter < 2, means there does not exist 2 x 150 consecutive days where he is working, we raise an exception and disallow the leave to be taken. If counter is not less than 2, it means even if the Care Taker takes a leave on the selected leave date, he still meets the 2 x 150 consecutive working days requirement, and is allowed to take a leave on the selected date.

Assuming that the PCS app is released only this month (november) and there are no longer any 2 x 150 valid consecutive days remaining in 2020, we enforced that a full time Care Taker cannot take any leave in 2020, which explains the first IF condition. We also disallow users to take leave on any dates before the current_date today, which explains the second IF condition.

Second Trigger: To automate the update of pet_count for each day in the availability after a bid is made.
Code:

```
CREATE OR REPLACE FUNCTION update_caretaker_pet_count_function() RETURNS trigger AS $$
BEGIN
  UPDATE CareTakerAvailability C set pet_count = pet_count + 1
    WHERE C.username = NEW.CTUsername
      AND C.date >= NEW.start_date
      AND C.date <= NEW.end_date;
  UPDATE CareTakerAvailability C set available = FALSE
    WHERE C.username = NEW.CTUsername
    AND (NEW.CTUsername in (SELECT * FROM FullTime))
    AND C.date >= NEW.start_date
    AND C.date <= NEW.end_date
    AND C.pet_count = 5;
  UPDATE CareTakerAvailability C set available = FALSE
    WHERE C.username = NEW.CTUsername
    AND (NEW.CTUsername in (SELECT * FROM PartTime))
    AND ((SELECT rating FROM Caretakers WHERE username = NEW.CTUsername)>=4)
    AND C.date >= NEW.start_date
    AND C.date <= NEW.end_date
    AND C.pet_count = 5;
  UPDATE CareTakerAvailability C set available = FALSE
    WHERE C.username = NEW.CTUsername
    AND (NEW.CTUsername in (SELECT * FROM PartTime))
    AND ((SELECT rating FROM Caretakers WHERE username = NEW.CTUsername)<4)
    AND C.date >= NEW.start_date
    AND C.date <= NEW.end_date
    AND C.pet_count = 2;
  RETURN NEW;
END
$$ LANGUAGE 'plpgsql';

DROP TRIGGER IF EXISTS update_caretaker_petcount_after_bid_trigger ON Bids;
CREATE TRIGGER update_caretaker_petcount_after_bid_trigger
  AFTER INSERT
  ON Bids
  FOR EACH ROW
  EXECUTE PROCEDURE update_caretaker_pet_count_function();
```

Explanation:
This trigger is used to automate the increase in pet_count for each day in the CareTakerAvailability table after an insertion of an entry into the Bids table (meaning a successful bid has been made by a pet owner for a care taker).

We also automate the availability of the Care Taker. If he is a Full Time caretaker and has a pet_count of 5 after the increase in pet_count, we will set available = false, meaning he will not be able to take care of any more pets on that day.

For part time care takers, we further check if he has a rating of more or equal to 4 from the Caretakers table. If yes, his limit on the number of pets he can take care of each day is 5, and the updated pet_count on the date is 5, we set available to false. If his rating is less than 4, his limit of the number of pets to take care of in each day is 2, so if his updated pet_count on the date is 2, we set available to false.

Third Trigger: To automate the update of earnings and salary in the CareTakerSalary table

```
CREATE OR REPLACE FUNCTION update_salary() RETURNS trigger AS $$
DECLARE end_of_month date := (SELECT(date_trunc('month', NEW.start_date::date) + interval '1 month' -
interval '1 day')::date);
DECLARE start_of_end_date_month date;
DECLARE full_time_count integer := (SELECT COUNT(*) FROM fulltime WHERE username =
NEW.ctusername);
DECLARE curr_pet_day integer;
DECLARE num_days_exceed_60 integer;
BEGIN
IF full_time_count > 0 THEN
  IF NEW.end_date > end_of_month THEN
    start_of_end_date_month := (SELECT date_trunc('MONTH', NEW.end_date)::DATE);
    curr_pet_day := (SELECT petdays FROM CareTakerSalary WHERE
      username = NEW.ctusername AND
      year = (SELECT date_part('year', start_of_end_date_month)) AND
      month = (SELECT date_part('month', start_of_end_date_month)));
    IF curr_pet_day >= 60 THEN
      UPDATE CareTakerSalary
      SET earnings = earnings + ((NEW.end_date - start_of_end_date_month + 1) * NEW.price_per_day),
petdays = petdays + (NEW.end_date - start_of_end_date_month + 1),
      final_salary = final_salary + ((NEW.end_date - start_of_end_date_month + 1) * (NEW.price_per_day *
0.8))
      WHERE
      username = NEW.ctusername AND
      year = (SELECT date_part('year', start_of_end_date_month)) AND
      month = (SELECT date_part('month', start_of_end_date_month));
      NEW.end_date := end_of_month;
    ELSE
      num_days_exceed_60 := curr_pet_day + (NEW.end_date - start_of_end_date_month + 1) - 60;
      IF num_days_exceed_60 > 0 THEN
        UPDATE CareTakerSalary
        SET earnings = earnings + ((NEW.end_date - start_of_end_date_month + 1) * NEW.price_per_day),
petdays = petdays + (NEW.end_date - start_of_end_date_month + 1),
        final_salary = final_salary + ((60 - curr_pet_day) * 50) + (num_days_exceed_60 *
(NEW.price_per_day * 0.8))
        WHERE
```

```
            username = NEW.ctusername AND
            year = (SELECT date_part('year', start_of_end_date_month)) AND
            month = (SELECT date_part('month', start_of_end_date_month));
            NEW.end_date := end_of_month;
          ELSE
            UPDATE CareTakerSalary
            SET earnings = earnings + ((NEW.end_date - start_of_end_date_month + 1) * NEW.price_per_day),
petdays = petdays + (NEW.end_date - start_of_end_date_month + 1),
            final_salary = final_salary + ((NEW.end_date - start_of_end_date_month + 1) * 50)
            WHERE
            username = NEW.ctusername AND
            year = (SELECT date_part('year', start_of_end_date_month)) AND
            month = (SELECT date_part('month', start_of_end_date_month));
            NEW.end_date := end_of_month;
          END IF;
        END IF;
      END IF;
      curr_pet_day := (SELECT petdays FROM CareTakerSalary WHERE
          username = NEW.ctusername AND
          year = (SELECT date_part('year', NEW.start_date)) AND
          month = (SELECT date_part('month', NEW.start_date)));
      IF curr_pet_day >= 60 THEN
        UPDATE CareTakerSalary
        SET earnings = earnings + ((NEW.end_date - NEW.start_date + 1) * NEW.price_per_day), petdays =
petdays + (NEW.end_date - NEW.start_date + 1),
        final_salary = final_salary + ((NEW.end_date - NEW.start_date + 1) * (NEW.price_per_day * 0.8))
        WHERE
        username = NEW.ctusername AND
        year = (SELECT date_part('year', NEW.start_date)) AND
        month = (SELECT date_part('month', NEW.start_date));
      ELSE
        num_days_exceed_60 := curr_pet_day + (NEW.end_date - NEW.start_date + 1) - 60;
        IF num_days_exceed_60 > 0 THEN
          UPDATE CareTakerSalary
          SET earnings = earnings + ((NEW.end_date - NEW.start_date + 1) * NEW.price_per_day), petdays =
petdays + (NEW.end_date - NEW.start_date + 1),
          final_salary = final_salary + ((60 - curr_pet_day) * 50) + (num_days_exceed_60 * (NEW.price_per_day *
0.8))
          WHERE
          username = NEW.ctusername AND
          year = (SELECT date_part('year', NEW.start_date)) AND
          month = (SELECT date_part('month', NEW.start_date));
        ELSE
          UPDATE CareTakerSalary
          SET earnings = earnings + ((NEW.end_date - NEW.start_date + 1) * NEW.price_per_day), petdays =
petdays + (NEW.end_date - NEW.start_date + 1),
          final_salary = final_salary + ((NEW.end_date - NEW.start_date + 1) * 50)
          WHERE
          username = NEW.ctusername AND
          year = (SELECT date_part('year', NEW.start_date)) AND
          month = (SELECT date_part('month', NEW.start_date));
        END IF;
    END IF;
    RETURN NEW;
ELSE
    IF NEW.end_date > end_of_month THEN
    start_of_end_date_month := (SELECT date_trunc('MONTH', NEW.end_date)::DATE);
    UPDATE CareTakerSalary
    SET earnings = earnings + ((NEW.end_date - start_of_end_date_month + 1) * NEW.price_per_day), petdays =
petdays + (NEW.end_date - start_of_end_date_month + 1),
    final_salary = (earnings + ((NEW.end_date - start_of_end_date_month + 1) * NEW.price_per_day)) * 0.75
    WHERE
    username = NEW.ctusername AND
    year = (SELECT date_part('year', start_of_end_date_month)) AND
    month = (SELECT date_part('month', start_of_end_date_month));
    NEW.end_date := end_of_month;
    END IF;
    UPDATE CareTakerSalary
    SET earnings = earnings + ((NEW.end_date - NEW.start_date + 1) * NEW.price_per_day), petdays = petdays
+ (NEW.end_date - NEW.start_date + 1),
    final_salary = (earnings + ((NEW.end_date - NEW.start_date + 1) * NEW.price_per_day)) * 0.75
    WHERE
    username = NEW.ctusername AND
    year = (SELECT date_part('year', NEW.start_date)) AND
    month = (SELECT date_part('month', NEW.start_date));
    RETURN NEW;
  END IF;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS update_ct_salary ON bids;
CREATE TRIGGER update_ct_salary
AFTER INSERT
ON "bids"
FOR EACH ROW
EXECUTE PROCEDURE update_salary();
```

Trigger explanation

This trigger allows the update of pet-day and salary for care taker on the CareTakerSalary table for every bid inserted into the Bids table. The reason for the update of salary to occur for every bid entry is because we do not allow the cancel of bid to happen. The update for salary is as follows:

For full-time Care Taker, if the pet day is lesser than or equal to 60 for that month, for every pet day increase for that month, his/her salary will increase by $50. For pet-day that is greater than 60, he/she will receive a salary increase of 80% of the price paid by the pet owner per pet-day. For example, if the current date is 25 May and the current pet day of the caretaker for the month is 58 and he received a request to take care of a pet with price per day $80 (price per day taken into account the user's rating) from 28 - 30 May. The care taker's pet-day will increase by 3 to 61 and his salary will increase by $164 ($50 + $50 + $64). Whereas for part-time care taker, the salary will increase by 75% of the price charged on the pet owner.

This trigger also takes into consideration that the start date and the end date of bid may span over a month, hence, the following is done: If the start date is 28 May and the end date is 5 June, the pet-day for May will increase by 3 and the pet-day for June will increase by 5 days. The increase in salary for May will be for the 3 pet-day and June for the 5 pet-day.

This trigger also updates the 'earnings' attribute which is the revenue earned by the company based on the caretaker. For example, if a pet owner bids for a caretaker for 3 pet-days such that each pet-day is $80. The 'earnings' attribute for the caretaker will increase by $240.

7. 3 most interesting queries

First Query: To find out top 10 underperformers per month.
Code:

```
"SELECT username, rating_in_month FROM (SELECT DISTINCT username, ROUND(AVG(B.rating), 2) AS
rating_in_month
  FROM CareTakerSalary S
  INNER JOIN Bids B ON B.CTusername = S.username
  WHERE S.year = '{}' AND S.month = '{}' AND petdays < 20 AND completed = True
  AND (EXTRACT(YEAR FROM B.start_date) = '{}' OR EXTRACT(YEAR FROM B.end_date) = '{}')
  AND (EXTRACT(MONTH FROM B.start_date) = '{}' OR EXTRACT(MONTH FROM B.end_date) = '{}')
  GROUP BY username) AS DUMMY
  WHERE rating_in_month < 3.5
  ORDER BY rating_in_month
  LIMIT 10".format(year, month, year, year, month, month)
```

Explanation:
This query is used for the admin to view the top 10 underperforming caretakers any month or year that he selected. We define a caretaker as underperforming if he has an average rating of less than 3.5 and has less that 20 petdays in a month. The query works by joining the CareTakerSalary and Bids Table, so that we can obtain information of the number of petdays for each caretaker from the CareTakerSalary table, and information on the average rating of the Care Taker in a month by taking the average of the rating in the Bids table. We selected rows from this table will be the ones where the start_date or end_date in the Bids table fall in the month and year selected by the Admin. The admin will then see an ordered list of the top 10 underperformer in ascending order of average rating in the month and year selected by him, where the worst performer will be shown in the top. This query is interesting because the Admin would be able to be notified who are the underperformers and take on further actions on these care takers if necessary, such as giving them warnings. In terms of the code, it is interesting as we made use of a subquery to help with the logic.

Second Query: To find out the filtered list of caretakers for the Pet Owners
Code:

```
"SELECT DISTINCT username, gender, CASE WHEN rating >= 4.5 THEN price * 1.5 WHEN rating >=4
THEN price * 1.25 ELSE price END price, rating
  FROM users U
  NATURAL JOIN FullTimePriceList NATURAL JOIN CareTakers NATURAL JOIN PreferredTransport
NATURAL JOIN PreferredModeOfPayment
  NATURAL JOIN CareTakerAvailability
  WHERE pettype = '{}' AND rating >= '{}' AND transport = '{}' AND modeofpayment = '{}'
  AND U.username <> '{}' AND True = ALL(SELECT available FROM CaretakerAvailability C
                              WHERE C.username = U.username
                              AND C.date >= '{}'
                              AND C.date <= '{}')"
    .format(category, rating, transport, payment, current_user.username, startDate, endDate)
```

Explanation

This query is used to help filter the Full Time Care Takers who meet the specifications stated by a Pet Owner when a Pet Owner is browsing for a Care Taker to take care of a pet. The most interesting part of this query is the use of the aggregate function 'ALL', which helps in finding out if the Care Taker is available in all the days from the start_date to the end_date. Furthermore, since the price in the FullTimePriceList table is the base price per day, we use a case analysis to multiply the base price by a multiplier, which is dependent on the rating of the caretaker selected, as this is the price that the Pet Owner needs to pay. If the caretaker has a rating >= 4.5, the multiplier will be 1.5, if the rating is 4 <= rating < 4.5, then the multiplier is 1.25, otherwise the multiplier will be 1 and the price the Pet Owner has to pay will be the base price.

Third Query: To find out the pet count of each type of pet that is taken care of in each month.

```
WITH cte(year, month, dog, cat, bird, terrapin, rabbit, hamster, fish, mice) AS (
  SELECT CTS.year, CTS.month, CASE WHEN dog IS NULL THEN 0 ELSE dog END dog,
                        CASE WHEN cat IS NULL THEN 0 ELSE cat END cat,
                        CASE WHEN bird IS NULL THEN 0 ELSE bird END bird,
                        CASE WHEN terrapin IS NULL THEN 0 ELSE terrapin END terrapin,
                        CASE WHEN rabbit IS NULL THEN 0 ELSE rabbit END rabbit,
                        CASE WHEN hamster IS NULL THEN 0 ELSE hamster END hamster,
                        CASE WHEN fish IS NULL THEN 0 ELSE fish END fish,
                        CASE WHEN mice IS NULL THEN 0 ELSE mice END mice
  FROM CareTakerSalary CTS
  LEFT JOIN (SELECT C.year, C.month, COUNT(*) AS dog
  FROM CareTakerSalary C RIGHT JOIN Bids B ON C.username = B.CTusername NATURAL JOIN
OwnedPets O
  WHERE (C.year = EXTRACT(YEAR FROM B.start_date) AND C.month = EXTRACT(MONTH FROM
B.start_date))
  AND O.category = 'Dog'
  GROUP BY C.year, C.month) AS dummy1 ON CTS.year = dummy1.year AND CTS.month = dummy1.month
  LEFT JOIN (SELECT C.year, C.month, COUNT(*) AS cat \
  FROM CareTakerSalary C RIGHT JOIN Bids B ON C.username = B.CTusername NATURAL JOIN
OwnedPets O
```

```
  WHERE (C.year = EXTRACT(YEAR FROM B.start_date) AND C.month = EXTRACT(MONTH FROM
B.start_date))
  AND O.category = 'Cat'
  GROUP BY C.year, C.month) AS dummy2 ON dummy2.year = CTS.year AND dummy2.month = CTS.month \
  LEFT JOIN (SELECT C.year, C.month, COUNT(*) AS bird \
  FROM CareTakerSalary C RIGHT JOIN Bids B ON C.username = B.CTusername NATURAL JOIN
OwnedPets O
  WHERE (C.year = EXTRACT(YEAR FROM B.start_date) AND C.month = EXTRACT(MONTH FROM
B.start_date))
  AND O.category = 'Bird'
  GROUP BY C.year, C.month) AS dummy3 ON dummy3.year = CTS.year AND dummy3.month = CTS.month
  LEFT JOIN (SELECT C.year, C.month, COUNT(*) AS terrapin
  FROM CareTakerSalary C RIGHT JOIN Bids B ON C.username = B.CTusername NATURAL JOIN
OwnedPets O
  WHERE (C.year = EXTRACT(YEAR FROM B.start_date) AND C.month = EXTRACT(MONTH FROM
B.start_date))
  AND O.category = 'Terrapin'
  GROUP BY C.year, C.month) AS dummy8 ON CTS.year = dummy8.year AND CTS.month = dummy8.month
  LEFT JOIN (SELECT C.year, C.month, COUNT(*) AS rabbit
  FROM CareTakerSalary C RIGHT JOIN Bids B ON C.username = B.CTusername NATURAL JOIN
OwnedPets O
  WHERE (C.year = EXTRACT(YEAR FROM B.start_date) AND C.month = EXTRACT(MONTH FROM
B.start_date))
  AND O.category = 'Rabbit'
  GROUP BY C.year, C.month) AS dummy4 ON CTS.year = dummy4.year AND CTS.month = dummy4.month
  LEFT JOIN (SELECT C.year, C.month, COUNT(*) AS hamster
  FROM CareTakerSalary C RIGHT JOIN Bids B ON C.username = B.CTusername NATURAL JOIN
OwnedPets O
  WHERE (C.year = EXTRACT(YEAR FROM B.start_date) AND C.month = EXTRACT(MONTH FROM
B.start_date))
  AND O.category = 'Hamster'
  GROUP BY C.year, C.month) AS dummy5 ON CTS.year = dummy5.year AND CTS.month = dummy5.month
  LEFT JOIN (SELECT C.year, C.month, COUNT(*) AS fish
  FROM CareTakerSalary C RIGHT JOIN Bids B ON C.username = B.CTusername NATURAL JOIN
OwnedPets O
  WHERE (C.year = EXTRACT(YEAR FROM B.start_date) AND C.month = EXTRACT(MONTH FROM
B.start_date))
  AND O.category = 'Fish'
  GROUP BY C.year, C.month) AS dummy6 ON CTS.year = dummy6.year AND CTS.month = dummy6.month
  LEFT JOIN (SELECT C.year, C.month, COUNT(*) AS mice
  FROM CareTakerSalary C RIGHT JOIN Bids B ON C.username = B.CTusername NATURAL JOIN
OwnedPets O
  WHERE (C.year = EXTRACT(YEAR FROM B.start_date) AND C.month = EXTRACT(MONTH FROM
B.start_date))
  AND O.category = 'Mice'
  GROUP BY C.year, C.month) AS dummy7 ON CTS.year = dummy7.year AND CTS.month = dummy7.month)
SELECT year, month, dog, cat, bird, terrapin, rabbit, hamster, fish, mice,
     (dog + cat + bird + terrapin + rabbit + hamster + fish + mice) AS total FROM cte
```

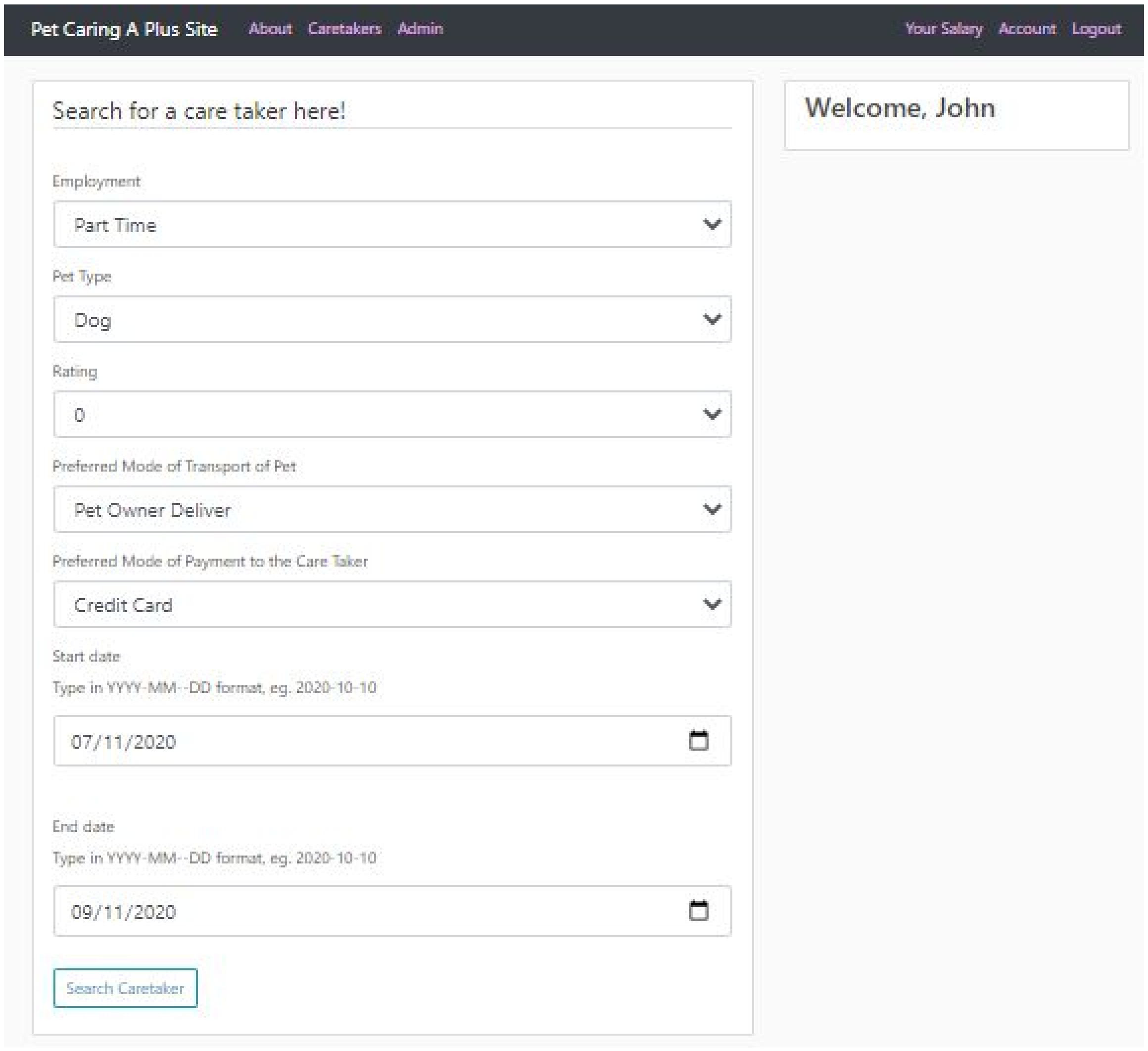
ORDER BY year, month;
Explanation: The use of the third query is for the Admin to get a summary of the number of jobs taken up by Care Takers for every month in every year, splitted up by the category of the pet that is taken care of. Business wise, this query is useful as the Admin would be able to know the trend of the more popular types of pets that has been taken care of and the less popular wise. We treat 1 job to be taken up in moving forward, this summary allows them to think of marketing strategies such as encouraging more people to sign up as care takers to take care of the more popular pet categories. For the less popular pet categories, it might be because the price per day set is too high, and the Admin might plan to lower the price if necessary. Code wise, this query is interesting because it uses a CTE to find out the number of jobs from each separate category. Furthermore, in order to get the summary, within the CTE, nested queries are used to create dummy tables and they are joined together using outer join methods

8. Specification of software tools/ frameworks used

	Framework/Tools
Frontend	CSS, Bootstrap
Backend	Flask, SQLAlchemy, psycopg2, WTForms
Database	PostgreSQL

9. Screenshots of applications in action

Here is the page where Pet Owners can choose Care Takers from a list of options in our selection page.



Search for a care taker here!

Employment: Part Time

Pet Type: Dog

Rating: 0

Preferred Mode of Transport of Pet: Pet Owner Deliver

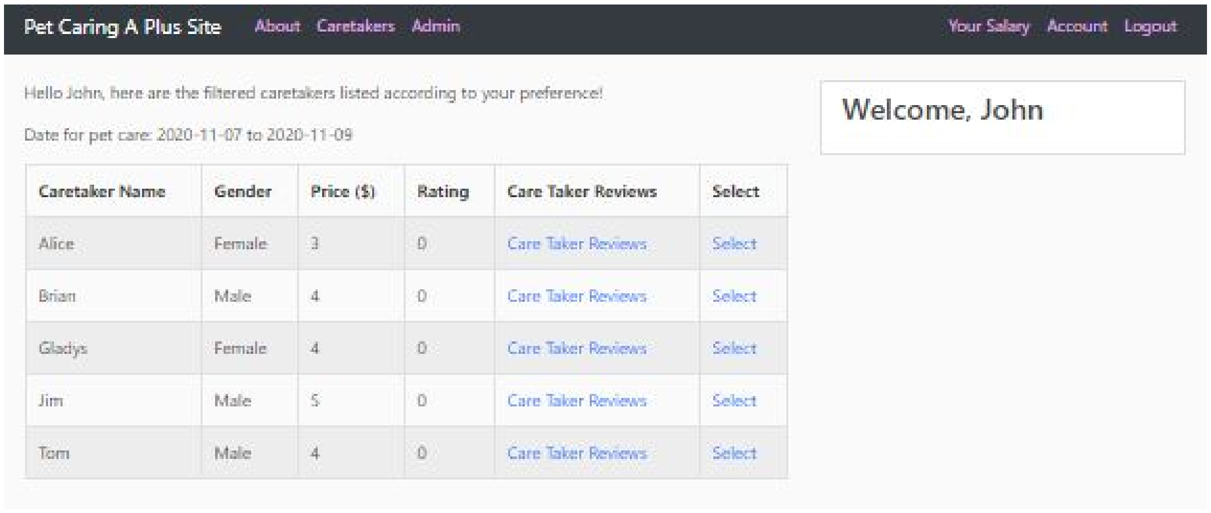
Preferred Mode of Payment to the Care Taker: Credit Card

Start date
Type in YYYY-MM--DD format, eg. 2020-10-10
07/11/2020

End date
Type in YYYY-MM--DD format, eg. 2020-10-10
09/11/2020

Welcome, John

This leads to a results page showing all Care Takers who fulfill the requirements selected by the Pet Owner.

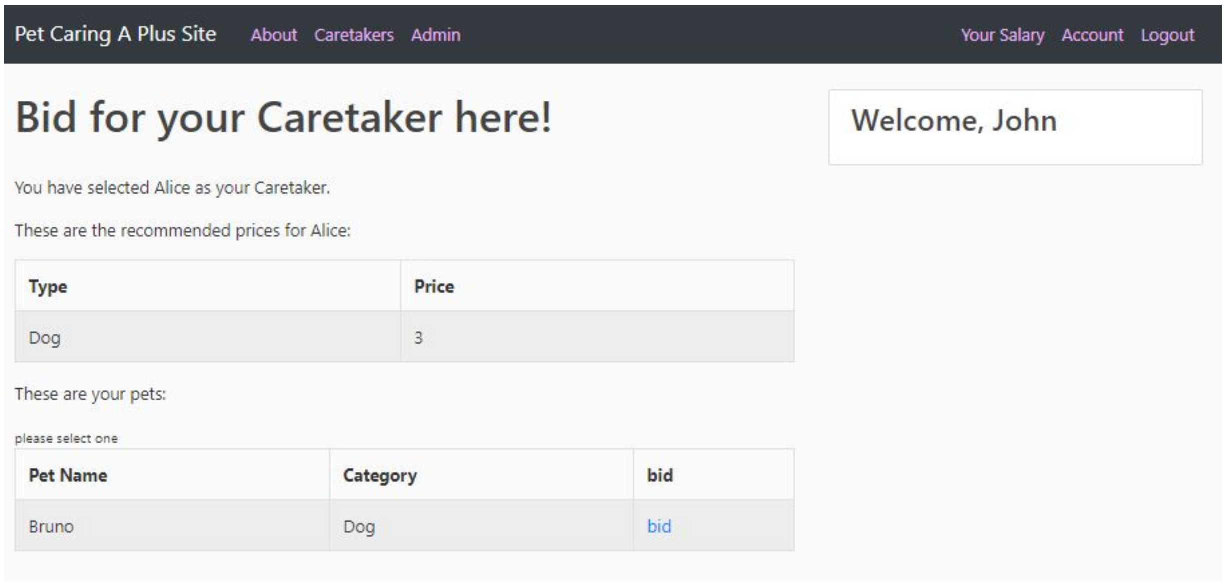


Hello John, here are the filtered caretakers listed according to your preference!

Date for pet care: 2020-11-07 to 2020-11-09

Caretaker Name	Gender	Price ($)	Rating	Care Taker Reviews	Select
Alice	Female	3	0	Care Taker Reviews	Select
Brian	Male	4	0	Care Taker Reviews	Select
Gladys	Female	4	0	Care Taker Reviews	Select
Jim	Male	5	0	Care Taker Reviews	Select
Tom	Male	4	0	Care Taker Reviews	Select

Here is the confirmation page for Pet Owner John to bid for Care Taker Alice for his dog Bruno's care in this case.



Bid for your Caretaker here!

You have selected Alice as your Caretaker.

These are the recommended prices for Alice:

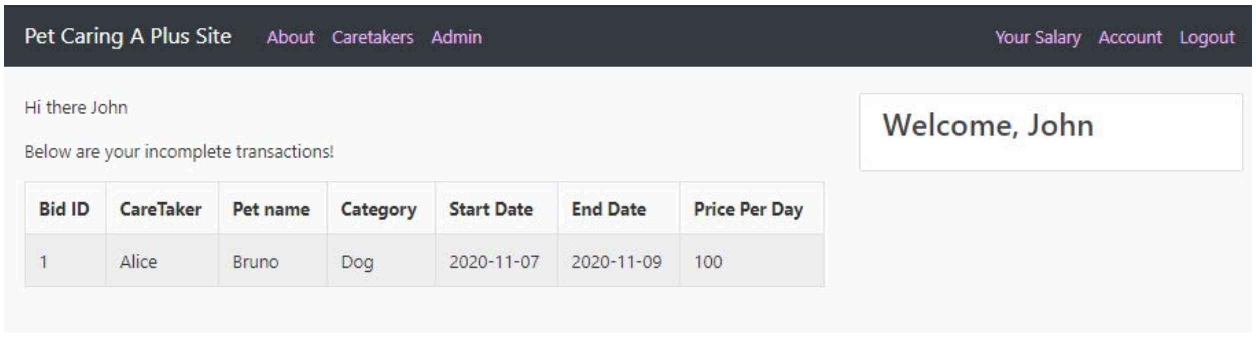
Type	Price
Dog	3

These are your pets:

please select one

Pet Name	Category	bid
Bruno	Dog	bid

Finally, the page here shows the unconfirmed bid by John for his pet Bruno to be cared for by Alice.



Hi there John

Below are your incomplete transactions!

Bid ID	CareTaker	Pet name	Category	Start Date	End Date	Price Per Day
1	Alice	Bruno	Dog	2020-11-07	2020-11-09	100

Welcome, John

10. Summary of any difficulties and lessons encountered.

Some difficulties that we encountered in this project is learning how to even use a backend development stack to do the web development component as most of our members have no prior experience with web development. In the end, we chose to use Flask as most of our members are more comfortable with Python rather than JavaScript, though quite some time was still needed to learn and get used to Flask.

Another difficulty for us is to make use of triggers as far as possible to handle the logic at the backend rather than using Python to do so. We only realised on Week 11 that we were not really following the project guidelines of the need to use Triggers to handle constraints such as Total Participation constraint, which caused some panic. However, once we got the hang of it, we realised how much easier and interesting it was to make use of triggers to handle the backend logic and constraints compared to writing the code in Python. In the end, we managed to handle most of the important backend using triggers, and the code written in Python was mainly for querying only.

Some lessons we learnt from this project is that it is important to have a good database design before even starting to code. After the first submission of the ER Diagram, we redesigned our ER Diagram so that relationships between entities are simpler and removed unnecessary entities, which definitely made the coding part a lot easier as everything was much clearer based on the ER Diagram.