CS 2102 Project Report
Pet Caring

Group 49

Jonathan Chan (A0189909E)
Mo Zongran (A0183818U)
Victor Varian (A0184586M)
Walter Kong (A0150039N)
William Ryan Kusnadi (A0187572R)

1. Team Dynamics

Name	Responsibilities
Chan Wai Hon Jonathan	Database (Tables, Constraints, Triggers)
Mo Zongran	Project Mgmt, DevOps, Backend, Database (Query)
Victor Varian	Database (Query), Backend
Walter Kong	
William Ryan Kusnadi	Frontend, FE+BE integration

2. Project Requirements & Functionalities

2.1 Project Description

Our pet caring service (PCS) system, called *CareTaker*, is an application that allows pet owner to search for care takers for their pets for certain periods of time. *CareTaker* allows

Pet Owners to
- Sign up for an account
- Add pets and view their information
- Search and bid for caretakers to look after their pets
- Leave reviews for caretakers
- View caretakers reviews
- View past selected bids

Care Takers to
- Sign up for an account
- Register themselves to take care of different kinds of pets
- Select bids to do
- View their salary
- View past jobs
- View statistics on this month

Admin to
- Create and set minimum price for categories
- Create other admins
- View aggregate statistics of caretakers and the system in general

2.2 Explanation of Non-trivial Functionalities

Functionality	Explanation
Search for PCS caretaker by a	A pet owner can just simply have a list of PCS

number of criteria and place bid	caretakers that fit the owner’s demand (pet category, min rating, max daily price, the period available), read the rates and review of the caretakers listed, and place bids on them.
PCS Admin is able to see the best performing and worst performing caretakers	PCS Admin could view the best and worst performing caretakers for admin purposes, such as issuing warning email or to add bonus if necessary.
PCS runs a query at end of every month which calculates the pay of the caretakers (part-time/full-time) and inserts them into the salary table	Generating past salary dynamically is a bad idea due to the possible data loss in the bids table. It would be better to calculate it once and store it, as we do not expect it to change after the month ends
PCS runs a query at midnight everyday to automatically select bids for free part time caretakers that were not selected by them	This is to give time for the part time caretaker to select their bid. Otherwise, the system chooses for them.
The website is designed to be a single page application in order to improve the user experience and limit the number of page refresh	In order to achieve this, we need to consider returning the necessary updated data to the front end so that we can achieve dynamic data change without the need of the user to refresh the page. For example, for a query that involves updating user information such as credit cards and saved pets, the api needs to return the updated User object with its updated attributes instead of just performing the update action. This needs to be carefully done because we need to sync the data in the front end and back end without the user refreshing the page.

2.3 Explanation of Constraints (By Entity)

Users

1. Users are identified by an email.
2. A User must have a password and name. (Non-null)
3. A User can be either a Pet owner and/or Care taker (is-a, can overlap), or a PCS Administrator.
4. A User can have a phone number and picture.

PetOwner

1. Pet Owners can have many pets.
2. Pet Owners can have many credit cards.

Pets

1. Pets are identified by a name and their owner.
2. Every Pet can have a description, special requirements (daily walk, types of food, etc), gender, date of birth
3. Every Pet has a Category. (cats, dogs, big dogs, lizards, etc) (Non-null)

(Dependent on PetOwner)

Categories

1. Categories are identified by name.
2. A category that has subcategories is called a parent category.
3. A category can have multiple subcategories (e.g. big dogs within dogs)
4. A subcategory only has 1 parent category.
5. A subcategory can also be a parent category for other subcategories

(Subcategories dependent on categories)

CareTaker

1. Every CareTaker has a daily price for every category of pets they can take care of.
2. The daily price indicated by the CareTaker must be positive and greater than the min daily price for the category set by the admin.
3. Every CareTaker can take care of more than one pet at any time.
4. Every CareTaker can have a salary.
5. Every CareTaker is either a Full-time (2x150 consecutive days/year) or part-time employee.
6. A Full-time CareTaker can take leave. This marks them as unavailable, otherwise, they would be available.
7. A Full-time CareTaker can only care for a pet if they are not on leave.
8. A Full-time CareTaker can take care of up to 5 pets once.
9. A full-time CareTaker will always accept any job if they have not applied for leave, they are not taking care of 5 pets at the same time yet, and they can take care of the pet in the job.
10. Every Part-Time CareTaker can have multiple availabilities.
11. A Part-Time CareTaker can only take care of a pet if they are available on all days.
12. A Part-time CareTaker cannot take care of more than 2 Pets unless they have a good rating (e.g., 4 out of 5) and cannot take care of more than 5 Pet regardless of rating.

PartTimeAvailabilities

1. Availabilities are identified by the CareTaker, start date, and end date.
2. Availabilities for the same CareTaker cannot overlap (start date between another availabilities' start and end date.
3. Start date should be before the end date.
4. Availabilities cannot be deleted for a CareTaker on a date where they are caring for a pet.

(Dependent on CareTaker)

FullTimeLeaves

1. Leaves are identified by the CareTaker, start date, and end date.
2. Leaves for the same CareTaker cannot overlap (start date between another leaves' start and end date)
3. Leaves cannot be created for a CareTaker on a date where they are caring for a pet.
4. Start date should be before the end date.

(Dependent on CareTaker)

Salary

1. Salary is identified by caretaker and month year, and must have an amount.
2. For a full-timer’s latest month, salary is calculated based on rating and the number of pets taken care of in a given month for how many days. This is called pet-day. The CareTaker will receive $3000 per month up to 60 pet-days, following which they receive 80% of the transaction price as a bonus.
3. For a part-timer’s latest month, the amount is 75% of their total transaction prices for the month.

(Dependent on CareTaker)

Credit Card

1. Credit card is identified by PetOwner and credit card number. It must have a holder name, expiry date.
2. Expiry date must be in the future.

(Dependent on PetOwner)

Bid

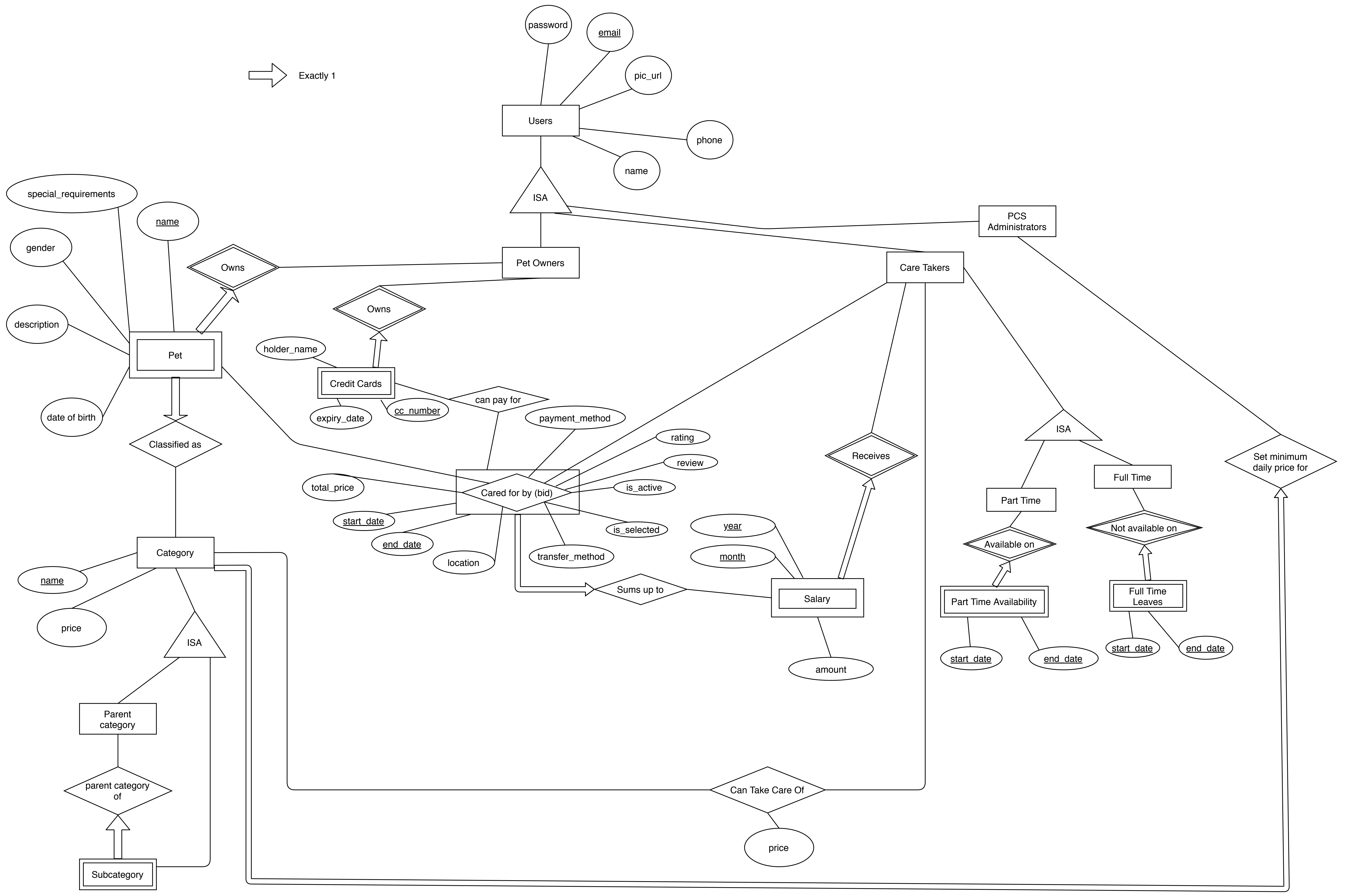
1. Bids are identified by pet, caretaker, start date and end date, and must have a transfer method (delivery, pickup, physical), location, total price, isActive, isSelected, payment method
2. Bids are made by PetOwners.
3. Bids cannot be created if the CareTaker is already taking care of the maximum number of pets they can care for.
4. Bids can be selected by the CareTakers.
5. If the bids were not selected two days before the start date, the bid with the highest daily rate is automatically selected by the system two days before the start date of the transaction.
6. New bids can only be for start dates more than two days in advance.
7. When a bid is accepted, all overlapping bids for the same dates for the same pet will be marked inactive, and if this bid causes the CareTaker to take care of the maximum number of pets they can take care of for some dates, all bids for those dates for this CareTaker will be invalidated.
8. Only selected bids can have a rating and a review (nullable).
9. If the payment method is by card, the bid should have a credit card number.
10. Start date should be before the end date.
11. Total price for the bid should be greater than both the admin set minimum daily price after accounting for the caretaker’s rating, and the caretaker’s own daily price for the category.

(Dependent on pet and caretaker)

PCS Admin

1. PCS Admin can specify the minimum positive daily price for a CareTaker for a particular pet type and the price increases with the rating of the CareTaker but will never be below the base price

3. Entity Relation Diagram


Exactly 1
password
email
pic_url
Users
phone
name
ISA
PCS Administrators
special_requirements
name
gender
Owns
Pet Owners
Care Takers
description
Owns
Pet
holder_name
Credit Cards
can pay for
expiry_date
cc_number
payment_method
date of birth
Classified as
rating
review
total_price
Cared for by (bid)
is_active
start_date
is_selected
end_date
location
transfer_method
Receives
ISA
Set minimum daily price for
Full Time
Part Time
Not available on
Available on
year
month
Sums up to
Salary
Part Time Availability
Full Time Leaves
Category
name
price
ISA
amount
start_date
end_date
start_date
end_date
Parent category
parent category of
Subcategory
Can Take Care Of
price

Non-trivial decisions

1. As per requirements, Pet Owners, Caretakers and PCS Administrators Is-A User.
2. As per requirements, a Caretaker can be either a Part-Time or Full-Time caretaker. This is established through a Is-A relationship as well.
3. Subcategories and Parent categories Is-A category. This is because they are both categories but with different roles depending on the relationship between two categories. E.g. Dog is a parent category of Big dogs, which is a parent category of Great Dane.
4. A bid (Cared for by) is an aggregation, as the relationship is used to determine the salary of caretakers.
5. Pets have a weak identity dependency on pet owners, as pets from different owners can have the same name.
6. Subcategories have a weak existential dependency on parent categories, as sub categories are uniquely identified by their own category name, and are simply put under parent categories so that users can easily gather some general information about the subcategory.
7. Salary has a weak identity dependency on caretakers as different caretakers can receive a salary in the same month.
8. Part Time Availabilities have a weak identity dependency on part time caretakers as different caretakers can declare their availability on the same start and end date.
9. Full Time Leaves have a weak identity dependency on full time caretakers as different caretakers can take their leaves on the same start and end date.

Unenforced Constraints

For this section and the next, the constraints that were half resolved will be split up and rephrased accordingly. For example, A full-time employee must have 2 x 150 consecutive days/year, split from CareTaker constraint 5, that a caretaker can be either full-time (2 x 150 consecutive days/year) or part-time.

1. A User must have a password and name. (Non-null)
2. A User can be either a Pet owner and/or Care taker (is-a, can overlap), or a PCS Administrator.
3. The daily price indicated by the CareTaker must be positive and greater than the min daily price for the category set by the admin.
4. A Full-time employee must have 2x150 consecutive days/year.
5. A Full-time CareTaker can only care for a pet if they are not on leave.
6. A Full-time CareTaker can take care of up to 5 pets once.
7. A full-time CareTaker will always accept any job if they have not applied for leave, they are not taking care of 5 pets at the same time yet, and they can take care of the pet in the job.
8. A Part-Time CareTaker can only take care of a pet if they are available on all days.
9. A Part-time CareTaker cannot take care of more than 2 Pets unless they have a good rating (e.g., 4 out of 5) and they cannot take care of more than 5 Pets regardless of rating.

10. Availabilities for the same CareTaker cannot overlap (start date between another availabilities' start and end date.
11. Availability start date should be before the end date.
12. Availabilities cannot be deleted for a CareTaker on a date where they are caring for a pet.
13. Leaves for the same CareTaker cannot overlap (start date between another leaves' start and end date)
14. Leaves cannot be created for a CareTaker on a date where they are caring for a pet.
15. Leave start date should be before the end date.
16. Salary must have an amount. (Non-null)
17. For a full-timer's latest month, salary is calculated based on rating and the number of pets taken care of in a given month for how many days. This is called pet-day. The CareTaker will receive $3000 per month up to 60 pet-day, following which they receive 80% of the transaction price as a bonus.
18. For a part-timer's latest month, the amount is 75% of their total transaction prices for the month.
19. A Credit Card must have a holder name, expiry date.
20. Credit card expiry date must be in the future.
21. Bids must have a transfer method (delivery, pickup, physical), location, total price, is active, is selected, payment method
22. Bids are made by PetOwners.
23. Bids cannot be created if the CareTaker is already taking care of the maximum number of pets they can care for.
24. Bids can be selected by the CareTakers.
25. If the bids were not selected two days before the start date, the bid with the highest daily rate is automatically selected by the system two days before the start date of the transaction.
26. New bids can only be for start dates more than two days in advance.
27. When a bid is accepted, all overlapping bids for the same dates for the same pet will be marked inactive, and if this bid causes the CareTaker to take care of the maximum number of pets they can take care of for some dates, all bids for those dates for this CareTaker will be invalidated.
28. Only selected bids can have a rating and a review (nullable).
29. If the payment method is by card, the bid should have a credit card number.
30. Bid start date should be before the end date.
31. Total price for the bid should be greater than both the admin set minimum daily price after accounting for the caretaker's rating, and the caretaker's own daily price for the category.
32. The price increases with the rating of the CareTaker but will never be below the base price set by the admin.

4. Relational Schemas

Schema	BCNF/3NF

```
CREATE TABLE users (
  email VARCHAR(256) PRIMARY KEY,
  password CHAR(64) NOT NULL,
  name VARCHAR(256) NOT NULL,
  phone VARCHAR(20),
  pic_url VARCHAR(256),
  is_admin BOOLEAN NOT NULL
);
```
	Let R1 = {email, password, name, phone, pic_url, is_admin}
	
	F_R1 = {email → password, name, phone, pic_url, is_admin}
	
	R1 is in BCNF as email is the superkey to R1

`CREATE TABLE caretakers (`
`  pcs_user VARCHAR(256) REFERENCES users(email) ON DELETE CASCADE ON UPDATE CASCADE PRIMARY KEY,`
`  is_part_time BOOLEAN NOT NULL`
`);`
	Let R2 = {pcs_user, is_part_time}
	
	F_R2 = {pcs_user → is_part_time}
	
	R2 is in BCNF as pcs_user is the superkey to R2

`CREATE TABLE part_time_availabilities (`
`  caretaker VARCHAR(256) REFERENCES caretakers(pcs_user) ON DELETE CASCADE ON UPDATE CASCADE,`
`  start_date DATE,`
`  end_date DATE CHECK (date(end_date) >= date(start_date)),`
`  PRIMARY KEY (caretaker, start_date, end_date)`
`);`
	Let R3 = {caretaker, start_date, end_date}
	
	F_R3 = {}
	
	R3 is in BCNF as all functional dependencies in F_R3 are trivial.

`CREATE TABLE full_time_leaves (`
`  caretaker VARCHAR(256) REFERENCES caretakers(pcs_user) ON DELETE CASCADE ON UPDATE CASCADE,`
`  start_date DATE,`
`  end_date DATE CHECK (date(end_date) >= date(start_date)),`
`  PRIMARY KEY (caretaker, start_date, end_date)`
`);`
	Let R4 = {caretaker, start_date, end_date}
	
	F_R4 = {}
	
	R4 is in BCNF as all functional dependencies in F_R4 are trivial.

`CREATE TABLE credit_cards (`
`  cc_number VARCHAR(50),`
`  holder_name VARCHAR(256) NOT NULL,`
`  expiry_date DATE NOT NULL CHECK (date(expiry_date) > CURRENT_DATE),`
`  owner VARCHAR(256) REFERENCES users(email) ON DELETE CASCADE ON UPDATE CASCADE,`
`  PRIMARY KEY (owner, cc_number)`
`);`
	Let R5 = {owner, cc_number, holder_name, expiry_date}
	
	F_R5 = {owner, cc_number → holder_name, expiry_date}
	
	R5 is in BCNF as owner and cc_number forms a superkey to R5

`CREATE TABLE categories (`
`  name VARCHAR(256) PRIMARY KEY,`
`  parent VARCHAR(256) REFERENCES categories(name) ON DELETE RESTRICT ON UPDATE CASCADE`
`);`
	Let R6 = {name, parent}
	
	F_R6 = {name → parent}
	
	R6 is in BCNF as name is the superkey to R6

`CREATE TABLE daily_prices (`
`  caretaker VARCHAR(256) REFERENCES caretakers(pcs_user) ON DELETE CASCADE ON UPDATE CASCADE,`
`  category VARCHAR(256) REFERENCES categories(name) ON DELETE CASCADE ON UPDATE CASCADE,`
`  price INT NOT NULL CHECK (price > 0),`
	Let R7 = {caretaker, category, price}
	
	F_R7 = {caretaker, category → price}
	
	R7 is in BCNF as caretaker and category forms a superkey to R7

`  PRIMARY KEY (caretaker, category)`
`);`	

`CREATE TABLE min_daily_prices (`
`  category VARCHAR(256) REFERENCES categories(name) ON DELETE CASCADE ON UPDATE CASCADE PRIMARY KEY,`
`  price INT NOT NULL CHECK (price > 0)`
`);`
	Let R8 = {category, price}
	
	F_R8 = {category → price}
	
	R8 is in BCNF as category is the superkey to R8

`CREATE TABLE pets (`
`  name VARCHAR(256),`
`  owner VARCHAR(256) REFERENCES users(email) ON DELETE CASCADE ON UPDATE CASCADE,`
`  description TEXT,`
`  special_requirements TEXT,`
`  gender VARCHAR(20),`
`  date_of_birth DATE,`
`  category VARCHAR(256) REFERENCES categories(name) ON DELETE RESTRICT ON UPDATE CASCADE NOT NULL,`
`  PRIMARY KEY (name, owner)`
`);`
	Let R9 = {owner, name, description, special_requirements, gender, date_of_birth, category}
	
	F_R9 = {owner, name → description, special_requirements, gender, date_of_birth, category}
	
	R9 is in BCNF as owner and name form a superkey to R9

`CREATE TYPE transfer_method AS ENUM ('deliver', 'pickup', 'pcs');`
`CREATE TYPE payment_method AS ENUM ('cash', 'cc');`
 	
`CREATE TABLE bids (`
`  pet_owner VARCHAR(256),`
`  pet VARCHAR(256),`
`  caretaker VARCHAR(256) REFERENCES caretakers(pcs_user) ON DELETE RESTRICT ON UPDATE CASCADE NOT NULL,`
 	
`  start_date DATE NOT NULL,`
`  end_date DATE NOT NULL CHECK (date(start_date) <= date(end_date)),`
 	
`  transfer_method transfer_method NOT NULL,`
`  location VARCHAR(256) NOT NULL,`
`  total_price INT NOT NULL CHECK (total_price > 0),`
`  is_active BOOLEAN NOT NULL,`
`  is_selected BOOLEAN NOT NULL,`
 	
`  payment_method payment_method NOT NULL,`
`  cc_number VARCHAR(50)`
`  CHECK ((payment_method = 'cash' AND cc_number IS NULL)`
`      OR (payment_method = 'cc' AND cc_number IS NOT NULL)),`
 	
`  rating SMALLINT`
	Let R10 = {pet_owner, pet, caretaker, start_date, end_date, transfer_method, location, total_price, payment_method, cc_number, rating, review}
	
	F_R10 = {pet_owner, pet, caretaker, start_date, end_date → transfer_method, location, total_price, payment_method, cc_number, rating, review,
	cc_number → payment_method,
	rating → is_selected,
	review → is_selected }
	
	R10 is not in BCNF as cc_number → payment_method is non-trivial and cc_number is not a superkey to R10.
	
	This table is not in BCNF as the last 3 functional dependencies cannot be extracted out to its own table. They are all just attributes of bids that either should be modified together (cc_number and payment_method), or are only valid on certain types of bids (rating and review are only valid where is_selected = true). The alternative for the latter is to define a transaction table to make them distinct, but this would result in duplicated data for the selected bids.

```
  CHECK (rating IS NULL OR (rating
IS NOT NULL AND date(end_date) <=
CURRENT_DATE AND is_selected)),
  review TEXT CHECK ((rating IS NULL
AND review IS NULL) OR (rating IS
NOT NULL AND review IS NOT NULL)),

  FOREIGN KEY (pet_owner, pet)
REFERENCES pets(owner, name) ON
DELETE RESTRICT ON UPDATE CASCADE,
  FOREIGN KEY (pet_owner, cc_number)
REFERENCES credit_cards(owner,
cc_number),
  PRIMARY KEY (pet_owner, pet,
caretaker, start_date, end_date)
);
```

```
CREATE TYPE month ENUM
('1','2','3','4','5','6','7','8','9'
,'10','11','12')

CREATE TABLE salary (
  caretaker VARCHAR(256) REFERENCES
caretakers(pcs_user) ON UPDATE
CASCADE,
  month month NOT NULL,
  year CHAR(4) NOT NULL,
  amount INT NOT NULL CHECK (amount
>= 0),
  PRIMARY KEY (caretaker, month,
year)
);
```
	Let R10 = {caretaker, month, year, amount}
	
	F_R10 = {caretaker, month, year → amount}
	
	R10 is in BCNF as caretaker, month and year form a superkey to R10

Since the bids table is not in BCNF, the database is not in BCNF.

Unenforced Constraints

1. A User can be either a Pet owner and/or Care taker (is-a, can overlap), or a PCS Administrator. (A PCS Admin can be a pet owner and/or caretaker as well with our schema)
2. The daily price indicated by the CareTaker must be greater than the min daily price for the category set by the admin.
3. A Full-time employee must have 2x150 consecutive days/year.
4. A Full-time CareTaker can only care for a pet if they are not on leave.
5. A Full-time CareTaker can take care of up to 5 pets once.
6. A full-time CareTaker will always accept any job if they have not applied for leave, they are not taking care of 5 pets at the same time yet, and they can take care of the pet in the job.
7. A Part-Time CareTaker can only take care of a pet if they are available on all days.
8. A Part-time CareTaker cannot take care of more than 2 Pets unless they have a good rating (e.g., 4 out of 5) and they cannot take care of more than 5 Pet regardless of rating.
9. Availabilities for the same CareTaker cannot overlap (start date between another availabilities' start and end date.
10. Availabilities cannot be deleted for a CareTaker on a date where they are caring for a pet.
11. Leaves for the same CareTaker cannot overlap (start date between another leaves' start and end date)
12. Leaves cannot be created for a CareTaker on a date where they are caring for a pet.

13. For a full-timer’s latest month, salary is calculated based on rating and the number of pets taken care of in a given month for how many days. This is called pet-day. The CareTaker will receive $3000 per month up to 60 pet-day, following which they receive 80% of the transaction price as a bonus.
14. For a part-timer’s latest month, the amount is 75% of their total transaction prices for the month.
15. Bids are made by PetOwners.
16. Bids cannot be created if the CareTaker is already taking care of the maximum number of pets they can care for.
17. Bids can be selected by the CareTakers.
18. If the bids were not selected two days before the start date, the bid with the highest daily rate is automatically selected by the system two days before the start date of the transaction.
19. New bids can only be for start dates more than two days in advance.
20. When a bid is accepted, all overlapping bids for the same dates for the same pet will be marked inactive, and if this bid causes the CareTaker to take care of the maximum number of pets they can take care of for some dates, all bids for those dates for this CareTaker will be invalidated.
21. Total price for the bid should be greater than both the admin set minimum daily price after accounting for the caretaker’s rating, and the caretaker’s own daily price for the category.
22. The price increases with the rating of the CareTaker but will never be below the base price set by the admin

5. Non-Trivial Triggers

5.1 When a bid is accepted, all overlapping bids for the same dates for the same pet will be marked inactive, and if this bid causes the CareTaker to take care of the maximum number of pets they can take care of for some dates, all bids for those dates for this CareTaker will be invalidated.

```
CREATE OR REPLACE FUNCTION remove_other_bids_if_selected()
RETURNS TRIGGER AS
$$ DECLARE max_pet INTEGER := 5;
   DECLARE i DATE := NEW.start_date;
BEGIN
  IF (NOT OLD.is_selected) AND NEW.is_selected
  THEN
      IF (SELECT C.is_part_time
        FROM caretakers C
        WHERE C.pcs_user = NEW.caretaker)
        AND get_average_rating(NEW.caretaker) < 4
      THEN max_pet := 2;
      END IF;
      -- remove all bids for the pet where there is an overlap
      UPDATE bids B
      SET is_active = false
      WHERE B.pet_owner = NEW.pet_owner AND B.pet = NEW.pet
      AND is_active = true
      AND (B.start_date, B.end_date + 1) OVERLAPS (NEW.start_date,
NEW.end_date + 1);
      -- remove all bids involving days where caretaker is now full
      WHILE i <= NEW.end_date LOOP
```

```
        IF (
          SELECT COUNT(*)
          FROM bids B
          WHERE B.caretaker = NEW.caretaker
          AND B.is_selected
          AND i BETWEEN B.start_date AND B.end_date) >= max_pet
        THEN
          UPDATE bids B
          SET is_active = false
          WHERE B.caretaker = NEW.caretaker
          AND B.is_active
          AND i BETWEEN B.start_date AND B.end_date;
        END IF;
      i := i + 1;
    END LOOP;
  END IF;
  RETURN NEW;
END; $$
LANGUAGE plpgsql;
CREATE TRIGGER bid7_remove_others_if_selected
AFTER UPDATE ON bids
FOR EACH ROW EXECUTE PROCEDURE remove_other_bids_if_selected();
```

5.2 A Full-time employee must have 2x150 consecutive days/year.

Note that we defined a helper function here that is ONLY used for this trigger, otherwise it would lead to repetitive code which is undesirable. The trigger function by itself is relatively trivial as it just calls this function, but the function is non-trivial.

```
-- Other triggers would have guaranteed that there are no overlapping leaves
CREATE OR REPLACE FUNCTION ft_year_can_meet_req(year_to_check DATE,
caretaker_user VARCHAR(256))
RETURNS BOOLEAN
AS
$$ DECLARE num_match INTEGER := 0;
   DECLARE prev_end DATE; -- day before working day, so if working day was
1st jan, this would be 31st dec
   DECLARE first_iter BOOLEAN := true;
   DECLARE temprow full_time_leaves%ROWTYPE;
BEGIN
  FOR temprow IN (
      SELECT F.start_date, F.end_date
      FROM full_time_leaves F
      WHERE F.caretaker = caretaker_user
      AND (year_to_check = date_trunc('year', F.start_date)
      OR year_to_check = date_trunc('year', F.end_date))
      ORDER BY F.start_date
  ) LOOP
      IF first_iter AND date_trunc('year', temprow.start_date) !=
year_to_check
      THEN
      prev_end := temprow.end_date;
      first_iter := false;
      CONTINUE;
      ELSE
      prev_end := year_to_check - interval '1 day';
      first_iter := false;
      END IF;
      IF temprow.start_date - prev_end - 1 >= 150
      THEN num_match := num_match + 1;
      END IF;
      prev_end := temprow.end_date;
  END LOOP;
  -- to count remaining days in the year after the last leave
  IF date_trunc('year', prev_end) = year_to_check
  AND year_to_check + interval '1 year' - prev_end - interval '1 day' >=
interval '150 days'
  THEN num_match := num_match + 1;
  END IF;
  IF date_trunc('year', prev_end) = year_to_check
  AND year_to_check + interval '1 year' - prev_end - interval '1 day' >=
interval '300 days'
  THEN num_match := num_match + 1;
  END IF;
  IF num_match >= 2 OR first_iter
  THEN RETURN true;
  ELSE RETURN false;
  END IF;
END; $$
LANGUAGE plpgsql;

-- 1. check whether still possible to meet full-time requirement of
-- 2 x 150 consecutive days / yr. otherwise reject leave
CREATE OR REPLACE FUNCTION ft_meets_req()
RETURNS TRIGGER AS
$$ BEGIN
  -- reject if more than 365 days, as that would mean fail to
  -- meet requirements in one of the years.
  IF NEW.end_date - NEW.start_date + 1 >= 365
  THEN RAISE EXCEPTION 'full time requirements will fail if this leave is
accepted';
  END IF;
  -- same year
  IF date_trunc('year', NEW.start_date) = date_trunc('year', NEW.end_date)
    AND ft_year_can_meet_req(date(date_trunc('year', NEW.start_date)),
NEW.caretaker)
  THEN
    RETURN NEW;
  -- different years
  ELSIF date_trunc('year', NEW.start_date) != date_trunc('year',
NEW.end_date)
    AND ft_year_can_meet_req(date(date_trunc('year', NEW.start_date)),
NEW.caretaker)
    AND ft_year_can_meet_req(date(date_trunc('year', NEW.end_date)),
NEW.caretaker)
  THEN
```

```
    RETURN NEW;
  ELSE
    RAISE EXCEPTION 'full time requirements will fail if this leave is
accepted';
  END IF;
END; $$
LANGUAGE plpgsql;
CREATE TRIGGER ft3_validate_meet_reqs
BEFORE INSERT OR UPDATE ON full_time_leaves
FOR EACH ROW EXECUTE PROCEDURE ft_meets_req();
```

5.3 Ensure a bid’s total price is less than both the minimum price set by the administrator and the minimum price set by the caretaker.

```
-- 6. when a bid is created, check to ensure total_price > min_price
CREATE OR REPLACE FUNCTION validate_bid_price()
RETURNS TRIGGER AS
$$ DECLARE cat VARCHAR(256);
BEGIN
  IF NOT NEW.is_active THEN RETURN NEW; END IF;
  SELECT P.category INTO cat
  FROM pets P
  WHERE P.owner = NEW.pet_owner AND P.name = NEW.pet;
  IF NEW.total_price < (
      SELECT C.price * (date(NEW.end_date) - date(NEW.start_date) + 1)
      FROM categories C
      WHERE C.name = cat
      ) * (SELECT CASE
            WHEN get_average_rating(NEW.caretaker) > 4.5 THEN 1.15
            WHEN get_average_rating(NEW.caretaker) > 4 THEN 1.1
            WHEN get_average_rating(NEW.caretaker) > 3 THEN 1.05
            ELSE 1
            END)
      OR NEW.total_price < (
      SELECT D.price * (date(NEW.end_date) - date(NEW.start_date) + 1)
      FROM daily_prices D
      WHERE D.caretaker = NEW.caretaker AND D.category = cat
      )
  THEN
      RAISE EXCEPTION 'price % is less than minimum possible price',
NEW.price;
  ELSE
      RETURN NEW;
END IF; END; $$
LANGUAGE plpgsql;
CREATE TRIGGER bid6_validate_total_price
BEFORE INSERT OR UPDATE ON bids
FOR EACH ROW EXECUTE PROCEDURE validate_bid_price();
```

6. Complex Queries

6.1 Get high rating caretaker details within the last some months

$1 - some integer indicating the value of last $1 months
$2 - top $2 performing caretaker

```
SELECT DISTINCT U1.email, U1.name, C1.is_part_time, AVG(B1.rating) AS
  avg_rating, COUNT(*)
FROM (caretakers AS C1 INNER JOIN users AS U1 ON C1.pcs_user = U1.email)
  INNER JOIN bids B1 ON B1.caretaker = C1.pcs_user
WHERE B1.rating is NOT NULL AND B1.is_selected = true AND B1.end_date >
  date_trunc('month', current_date - interval '" + $1 + " month') AND
  B1.end_date <= date_trunc('month', current_date)
GROUP BY B1.caretaker, C1.pcs_user, U1.email
HAVING AVG(B1.rating) >= 4
ORDER BY avg_rating DESC
LIMIT $2;
```

6.2 Calculate salary for all caretakers in the system for some month

$1 - some date in the month that we wish to calculate the salary for

```
WITH pet_days(day, pet_owner, pet, caretaker, daily_price, transfer_method,
location, payment_method, rating) AS (
  SELECT D.day, B.pet_owner, B.pet, B.caretaker, B.total_price / (B.end_date
+ 1 - B.start_date) AS daily_price, B.transfer_method,
      B.location, B.payment_method, B.rating
  FROM (SELECT date(generate_series(first_day_of_month($1),
last_day_of_month($1), interval '1 day'))
      AS day) D INNER JOIN bids B ON D.day BETWEEN B.start_date AND
B.end_date
  GROUP BY B.caretaker, B.pet, B.pet_owner, B.start_date, B.end_date, D.day
  ORDER BY D.day
)
SELECT C.pcs_user AS caretaker, C.is_part_time AS is_part_time,
COUNT(pet_days.day) AS pet_days,
  COALESCE(SUM(pet_days.daily_price), 0) AS revenue,
  CASE
      WHEN C.is_part_time THEN COALESCE(SUM(pet_days.daily_price),0) * 0.75
      ELSE 3000 + (SELECT COALESCE(SUM(PD.daily_price),0)
                      FROM (SELECT * FROM pet_days
                      WHERE pet_days.caretaker = C.pcs_user
                      OFFSET 60) PD) * 0.8
  END salary
FROM caretakers C LEFT OUTER JOIN pet_days ON C.pcs_user = pet_days.caretaker
GROUP BY C.pcs_user, C.is_part_time
ORDER BY salary DESC;
```

6.3 Search for caretakers with a few criteria

$1 pet category
$2 minimum rating of caretaker
$3 maximum daily price
$4 start date
$5 end date

```
CREATE OR REPLACE FUNCTION get_average_rating(caretaker_user VARCHAR(256))
RETURNS NUMERIC AS
$$ BEGIN
RETURN (SELECT COALESCE(AVG(rating), 0)
 FROM bids B
 WHERE B.caretaker = caretaker_user AND B.is_selected AND B.end_date <=
CURRENT_DATE);
END; $$
LANGUAGE plpgsql;

SELECT * FROM users U
WHERE  get_average_rating(U.email) > $2
 AND EXISTS (
   SELECT * FROM daily_prices P
   WHERE U.email=P.caretaker AND P.category=$1 AND P.price<$3)
 AND
   CASE
     WHEN U.email IN (SELECT caretaker FROM full_time_leaves)
       THEN NOT EXISTS (
         SELECT * FROM full_time_leaves L
         WHERE L.caretaker=U.email
           AND (date(L.start_date), date(L.end_date) + 1) OVERLAPS
(date($4), date($5) + 1))
     WHEN U.email IN (SELECT caretaker FROM part_time_availabilities)
       THEN (
         SELECT COUNT(*)
         FROM (SELECT generate_series(date($4), date($5), '1 day') AS day) D
         WHERE EXISTS (
           SELECT * FROM part_time_availabilities A
           WHERE D.day BETWEEN A.start_date AND A.end_date)
       ) = (date($5) - date($4) + 1)
     ELSE FALSE
   END
 AND NOT EXISTS (
   SELECT * FROM (SELECT generate_series(date($4), date($5), '1 day') AS
day) D
   WHERE (SELECT COUNT(*)
     FROM bids
     WHERE is_selected=TRUE AND D.day BETWEEN start_date AND end_date
   ) >= (
     CASE
       WHEN EXISTS (
         SELECT pcs_user FROM caretakers
         WHERE pcs_user=U.email AND is_part_time=FALSE)
         THEN 5
```

```
        WHEN EXISTS (
          SELECT pcs_user FROM caretakers
          WHERE pcs_user=U.email
            AND is_part_time=TRUE)
          THEN CASE WHEN get_average_rating(U.email) < 4 THEN 2
                    ELSE 5 END
        ELSE -1
      END
    )
  );
```

7. Software Tools & Frameworks

Category	Tools & Frameworks
Frontend	React (TypeScript)
Backend / Server	ExpressJS (TypeScript)
Database	Postgresql
Other Frameworks	Deployment: Heroku, Docker

8. Representative Screenshots

8.1 Login Page

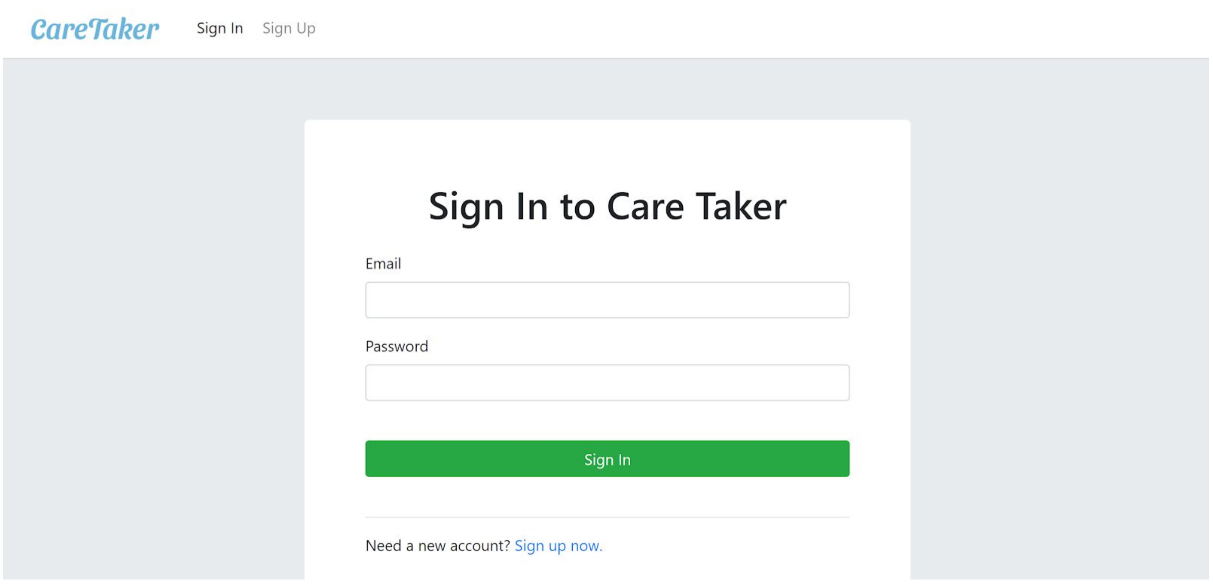
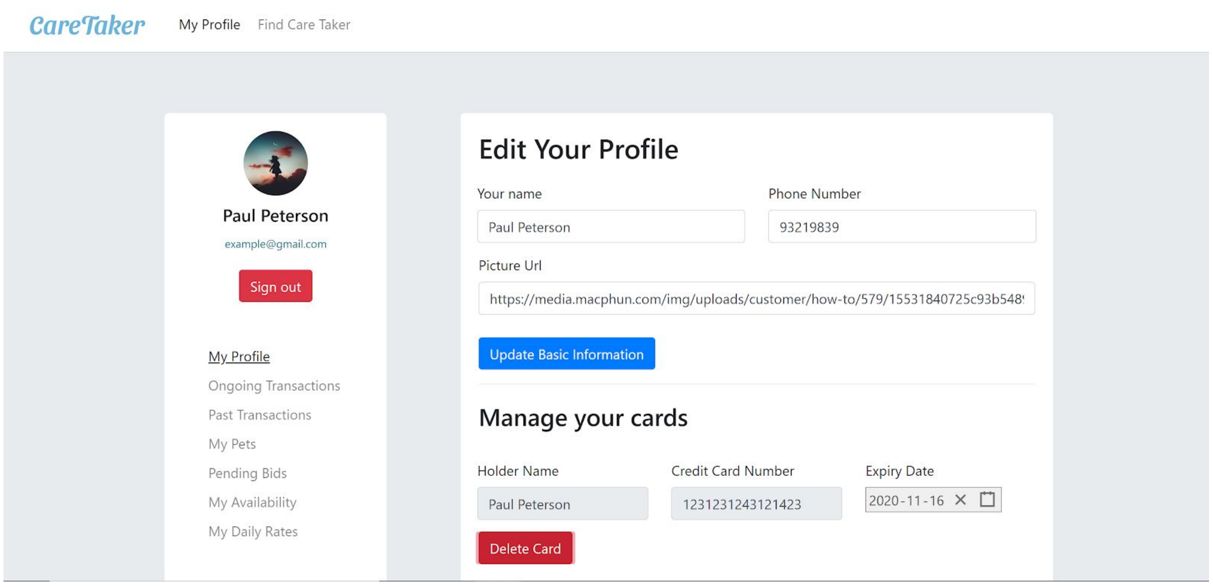


Figure 1. Brief overview of the signin page

8.2 Main Page



CareTaker My Profile Find Care Taker

Paul Peterson
example@gmail.com
Sign out

My Profile
Ongoing Transactions
Past Transactions
My Pets
Pending Bids
My Availability
My Daily Rates

Edit Your Profile

Your name: Paul Peterson
Phone Number: 93219839
Picture Url: https://media.macphun.com/img/uploads/customer/how-to/579/15531840725c93b548

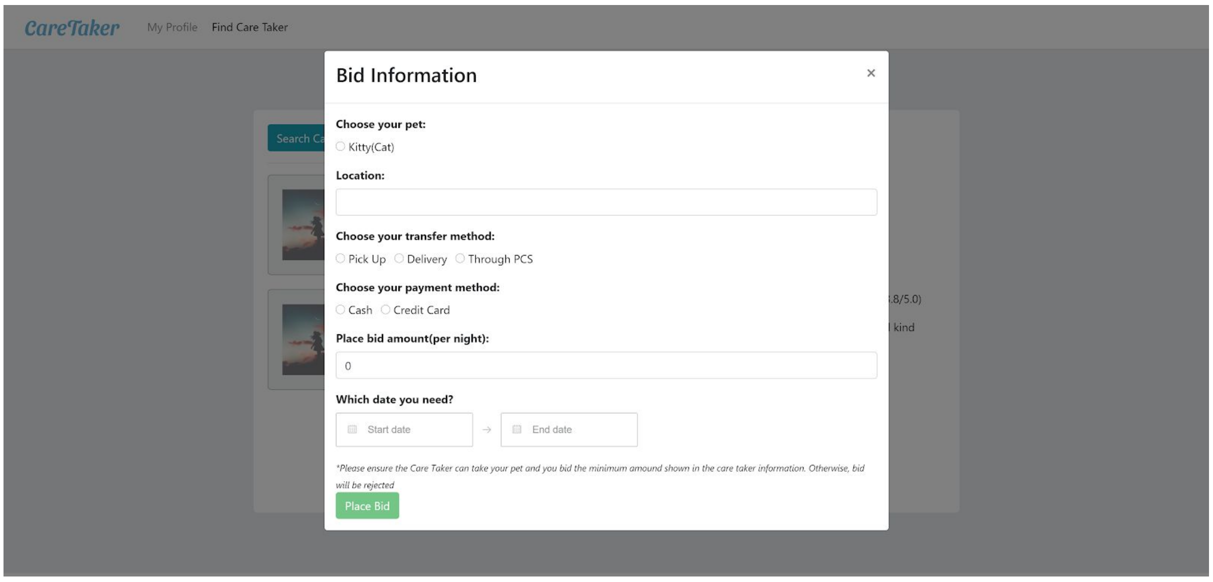
Update Basic Information

Manage your cards

Holder Name: Paul Peterson
Credit Card Number: 1231231243121423
Expiry Date: 2020-11-16

Delete Card

Figure 2. User main page with tabs that support multiple functionalities such as editing user profile and browsing through transactions/bids



CareTaker My Profile Find Care Taker

Bid Information

Choose your pet:
Kitty(Cat)

Location:

Choose your transfer method:
Pick Up Delivery Through PCS

Choose your payment method:
Cash Credit Card

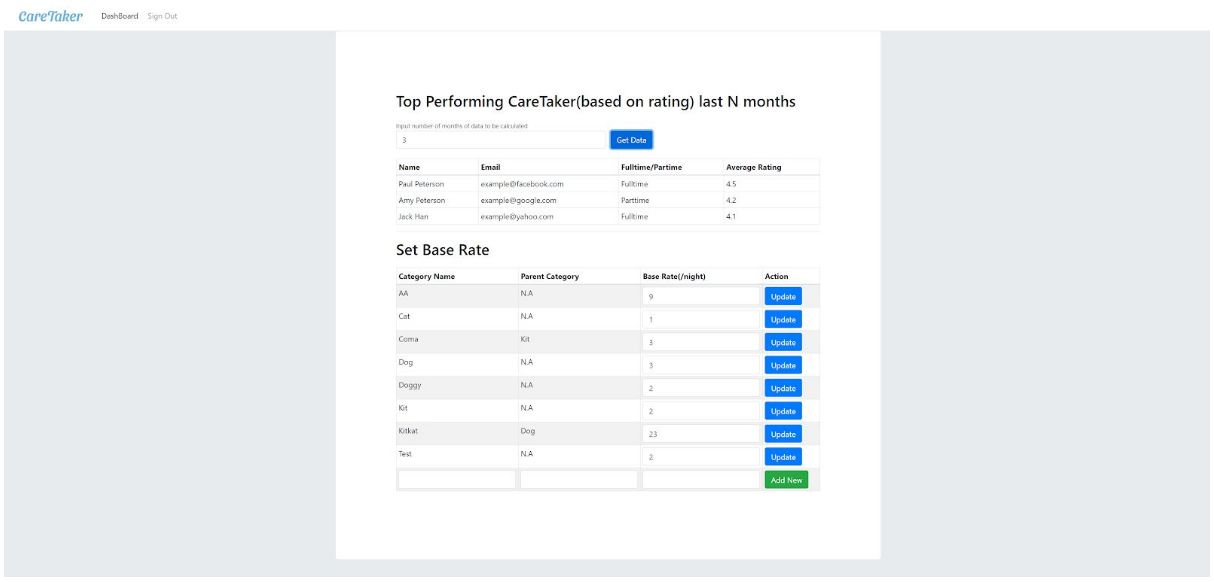
Place bid amount(per night):
0

Which date you need?
Start date → End date

*Please ensure the Care Taker can take your pet and you bid the minimum amound shown in the care taker information. Otherwise, bid will be rejected

Place Bid

Figure 3. Bidding page for pet owners to make a bid. Pet owners can search for caretaker and check their information such as rates/reviews in this page too


CareTaker DashBoard Sign Out

Top Performing CareTaker(based on rating) last N months

Input number of months of data to be calculated: 3 Get Data

Name	Email	Fulltime/Partime	Average Rating
Paul Peterson	example@facebook.com	Fulltime	4.5
Amy Peterson	example@google.com	Parttime	4.2
Jack Han	example@yahoo.com	Fulltime	4.1

Set Base Rate

Category Name	Parent Category	Base Rate(/night)	Action
AA	N.A	9	Update
Cat	N.A	1	Update
Coma	Kit	3	Update
Dog	N.A	3	Update
Doggy	N.A	2	Update
Kit	N.A	2	Update
Kitkat	Dog	23	Update
Test	N.A	2	Update
			Add New

Figure 4. Dashboard Page for PCS admin to set base rate for different pet categories and see a few application summary(top caretakers, etc.).

9. Summary of Difficulties and Lessons Learnt

1. Time management

Building a web app is not easy. We were ambitious at the start and wanted to try out using different unfamiliar technologies, such as gRPC, only to realise we spent quite a lot of time fiddling with things to make things work, which could have been spent actually implementing and improving the application instead. We also spent longer than we expected developing certain parts of the application, and should have planned more buffer time instead.

2. Plan more thoroughly about the database and the business use case

While we thought that we have thoroughly planned our database to ensure it was future proof, towards the end, we also realised that some decisions we made in our database might not have been ideal, and thus had to make last-minute changes to it. One example was the salary table. We were initially planning to get past month information by using queries and aggregating on the bids table, which would mean running the query everytime we wanted to retrieve, say, the salary of some caretaker 5 months ago. However, we realised that information like this should be constant once 5 months has passed, as there should not suddenly be more or less successful bids after the month is over. Hence, we shifted to storing past month's data inside a table instead, which also makes querying for this information much more efficient. In this case, the changes made do not significantly affect the backend or the frontend. However, if this had happened to the other tables, a lot of changes would have been needed to accommodate for this schema change.