CS2102 Project Team 69

Lee Ze Xin A0203252X

Li Beining

Wong Zheng Zhi A0206020E

Guo Yichao A0204731R

Ni Jiaying A0202459H

List of the project responsibilities:(Hi, can all you of just fill up relevant part you are involved in)

Wong Zheng Zhi: Front end, Back end, DB, Register Login DB, bids related

Li Beining: Front end, Back end, DB, Authentication, owner carer related

Guo Yichao: Front end, Back end, DB, PSC administrator related

Lee Ze Xin: Front end, Back end, DB, bids carer related

Ni Jiaying: Front end, Back end, DB, owner related

Section One:

A description of your application's data requirements and functionalities. Highlight any interesting/non-trivial aspects of your application's functionalities/implementation. List down all the application's data constraints.

Data requirements and functionalities

User Features	Operational needs	Performance Parameters
User Registration and Login	- User should be able to create an account as a pet owner, a caretaker or an administrator - User should be able to login into his/her account - Administrator should be able to manage user information - All users should be able to view and update their account information	Table: Accounts - username (PK) - password_hash - name - phone - area - address Table: Admins, Owners, Carers which have the IS-A relationship with Accounts
Data Access for User	- Pet owner can view reviews of caretaker - Pet owner can view their own past orders - Caretaker can see their review, their past jobs and their salary	Table: Categories - category_name (PK) - base_price Table: Carers - carer_name (PK) - rating - is_fulltime Table: Bids - start_date (PK) - end_date (PK) - carer_name (PK) - owner_name (PK) - pname (PK) - Bid_date - daily_price - is_successful - credit_card_num - Payment_date - payment_mode - delivery_method - review_rating - review_content - review_date
Browsing and Searching of	- Pet owner should be able to view all	Table: Pets - pname (PK)

caretaker by pet owner	caretakers available and capable for their pets - Pet owner can sort caretakers based on their rating and price	- owner_name (PK) - belongs - requirements Table: Carers - carer_name (PK) - rating - is_fulltime
Browsing of summary information for PCS administrator	- Administrators can view the total number of pets taken care of in the month - Administrators can view the total salary to be paid to all caretakers for the given month - Administrators can view the month with the highest number of jobs - Administrators can view the underperforming full-time caretakers	Table: Carers - carer_name (PK) - rating - Is_fulltime Table: Bids - start_date (PK) - end_date (PK) - carer_name (PK) - owner_name (PK) - pname (PK) - Bid_date - daily_price - is_successful - credit_card_num - Payment_date - payment_mode - delivery_method - review_rating - review_content - review_date
Browsing of summary information for caretaker	- Caretakers can view the total number of pet-days this month - Caretakers can view their expected salary for this month	Table: Bids - start_date (PK) - end_date (PK) - carer_name (PK) - owner_name (PK) - pname (PK) - Bid_date - daily_price - is_successful - credit_card_num - Payment_date - payment_mode - delivery_method - review_rating - review_content - review_date Table: Working_days - working_date (PK) - carer_name (PK) - number_of_pets

Browsing of summary information for pet owner	- Pet owners can view all caretakers in their area - Pet owner can view their pets' information	Table: Pets - pname (PK) - owner_name (PK) - belongs - requirements

Interesting/non-trivial aspects

- Login authentication is implemented.
    - Three types of users (pet owner, caretaker, admin) can create accounts accordingly. To enhance data security, password policy is enforced: each password must be of a minimum length of 3 characters. Moreover, only hashed password will be stored
- Rating and Reviewing system is implemented.
    - Owners can rate and review caretakers after every service. Review date, content and rating will be captured in the database.
    - Caretakers with higher ratings will be rewarded extra commissions.

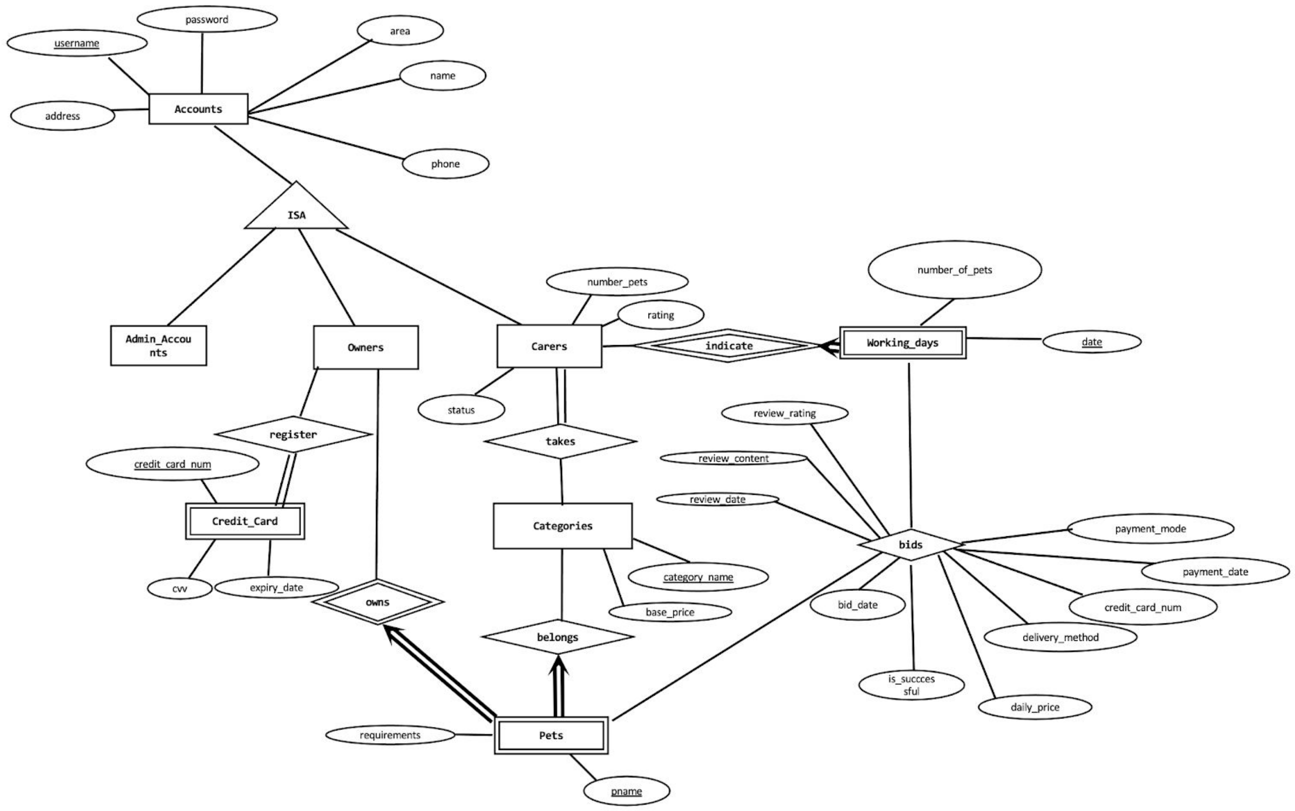
Data constraints

- Accounts and Users:
    - There are three types of Accounts, admin account owner account and carer account. A user can be both an owner and a carer(Covering and Overlapping).
- Owners:
    - Each Owner owns any number of pets, each pet must be owned by exactly one owner. (Weak Entity Set)
    - A pet cannot exist without an owner, each pet cannot be identified by itself, must be identified by the owner's user_id and its pet_id. (Identity Dependency)
    - Each owner can have any number of credit cards registered. Each registered credit card must belong at at least one owner.
- Carers:
    - A carer's identity of part-time or full-time depends on attribute status
    - A carer can indicate any number of availability. An availability must belong to exactly one carer.
    - A carer must take care of at least one category of pet. A category can be taken care of by any number of carers.
- Pet:
    - Each pet must belong to exactly one category, and each category can have any number of pets.
    - Each Owner owns any number of pets, each pet must be owned by exactly one owner
    - Each pet can bid for any number of availability. Each availability can be bided by any number of pets

Section Two:

The ER model of your application. If your ER model is too large (*i.e.*, spanning more than a page), you may want to include a single-page simplified ER model (*with non-key attributes omitted*) before presenting the detailed ER model. Provide justifications for any non-trivial design decisions in your ER model. List down all the application's constraints that are not captured by your ER model.

ER model



Section Three:

The relational schema derived from your ER data model (*i.e.*, show the SQL CREATE TABLE statements and possibly ALTER TABLE if any for all your database tables). List down all the application's constraints that are not enforced by your relational schema (*e.g.*, the constraints that are enforced using triggers, or cannot be enforced at all).

- Credit card must have at least one user using it is enforced by credit-card's primary key is (owner_name, credit_card_num)
- Total participations are enforced by using not null foreign key as weak entity's attribute.
- Working_days' number_of_pets must be equal to the number of successful bids that includes this working day in its duration. This is enforced by adding a trigger
- Each carer must have at least one category to take care of, during account registration

```
CREATE DATABASE petcaringdb;
```

```sql
CREATE TABLE accounts(

 username VARCHAR(20) NOT NULL,

 password_hash VARCHAR(255) NOT NULL,

 name VARCHAR(30) NOT NULL,

 phone VARCHAR(20) NOT NULL,

 area VARCHAR(5),

 address VARCHAR(80),

 PRIMARY KEY(username)

);



CREATE TABLE admins(

 admin_name VARCHAR(20) NOT NULL REFERENCES accounts(username),

 PRIMARY KEY(admin_name)

);



CREATE TABLE owners(

  owner_name VARCHAR(20) NOT NULL REFERENCES accounts(username) ON DELETE
Cascade,

 PRIMARY KEY(owner_name)

);



CREATE TABLE categories (

 category_name VARCHAR(20) PRIMARY KEY,
 base_price NUMERIC(5,2) NOT NULL

);


CREATE TABLE pets(

 pname VARCHAR(20) NOT NULL,

 owner_name VARCHAR(20) NOT NULL REFERENCES owners(owner_name) ON DELETE CASCADE,

 requirements TEXT,

 belongs VARCHAR(20) NOT NULL REFERENCES categories(category_name),

 PRIMARY KEY(pname, owner_name)

);


CREATE TABLE credit_cards(

    owner_name VARCHAR(20) REFERENCES owners(owner_name) ON DELETE Cascade,

    credit_card_num VARCHAR(16) NOT NULL,

    cvv INTEGER NOT NULL,

    expiry_date DATE NOT NULL,

    PRIMARY KEY(owner_name, credit_card_num)

);


CREATE TABLE carers(

 carer_name VARCHAR(20) NOT NULL REFERENCES accounts(username),

 rating NUMERIC(3, 2),

 is_fulltime BOOL NOT NULL,
 PRIMARY KEY(carer_name)

);


CREATE TABLE takes_care (

  carer_name VARCHAR(20) NOT NULL REFERENCES carers(carer_name),

  category_name VARCHAR(20) NOT NULL REFERENCES categories(category_name),

  carer_price NUMERIC(5,2) NOT NULL ,

  PRIMARY KEY(carer_name, category_name)

);


CREATE TABLE working_days (

 working_date DATE NOT NULL,

 carer_name VARCHAR(20) NOT NULL REFERENCES carers(carer_name) ON DELETE Cascade,

 number_of_pets INT,

 PRIMARY KEY(working_date, carer_name)

);


CREATE TABLE bids(

  start_date DATE NOT NULL,

  end_date DATE NOT NULL,

  carer_name VARCHAR(20) NOT NULL,

  owner_name VARCHAR(20) NOT NULL,

  pname VARCHAR(20) NOT NULL,
  bid_date DATE NOT NULL,

  daily_price NUMERIC NOT NULL,

  is_successful BOOL NULL,

  credit_card_num VARCHAR(16),

  payment_date DATE NOT NULL,

  payment_mode VARCHAR(50) NOT NULL,

  delivery_method VARCHAR(50) NOT NULL,

  review_rating INTEGER NULL,

  review_content VARCHAR(500) NULL,

  review_date DATE NULL,

  FOREIGN KEY (start_date, carer_name) REFERENCES working_days(working_date,
carer_name ) ON DELETE Cascade,

  FOREIGN KEY (end_date, carer_name) REFERENCES working_days(working_date,
carer_name ) ON DELETE Cascade,

  FOREIGN KEY (pname, owner_name) REFERENCES pets(pname, owner_name),

  PRIMARY KEY(start_date, end_date, carer_name, owner_name, pname)

);
```

Section Four:

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

```
CREATE TABLE accounts(

 username VARCHAR(20) NOT NULL,
 password_hash VARCHAR(255) NOT NULL,

 name VARCHAR(30) NOT NULL,

 phone VARCHAR(20) NOT NULL,

 area VARCHAR(5),

 address VARCHAR(80),

 PRIMARY KEY(username)

);
```

“accounts” table: not BCNF as address -> area and address is not super key. Not 3NF because address is a non-prime attribute and it implies area which is another non-prime attribute. Thus, this table is in 2NF.

```
CREATE TABLE admins(

 admin_name VARCHAR(20) NOT NULL REFERENCES accounts(username),

 PRIMARY KEY(admin_name)

);

CREATE TABLE owners(

  owner_name VARCHAR(20) NOT NULL REFERENCES accounts(username) ON DELETE
Cascade,

 PRIMARY KEY(owner_name)

);
```

```
CREATE TABLE categories (

 category_name VARCHAR(20) PRIMARY KEY,

 base_price NUMERIC(5,2) NOT NULL

);
```

BCNF due to binary relation.

```
CREATE TABLE pets(

 pname VARCHAR(20) NOT NULL,

 owner_name VARCHAR(20) NOT NULL REFERENCES owners(owner_name) ON DELETE
CASCADE,

 requirements TEXT,

 belongs VARCHAR(20) NOT NULL REFERENCES categories(category_name),

 PRIMARY KEY(pname, owner_name)

);
```

(pname, owner_name) -> (requirements, belongs)

BCNF. For each non-trivial FD X->Y, X is superkey.

```
CREATE TABLE credit_cards(

    owner_name VARCHAR(20) REFERENCES owners(owner_name) ON DELETE
Cascade,

    credit_card_num VARCHAR(16) NOT NULL,

    cvv INTEGER NOT NULL,

    expiry_date DATE NOT NULL,

    PRIMARY KEY(owner_name, credit_card_num)

);
```

(owner_name, credit_card_num) -> (cvv, expiry_date)

BCNF. For each non-trivial FD X->Y, X is superkey.

```
CREATE TABLE carers(

 carer_name VARCHAR(20) NOT NULL REFERENCES accounts(username),
 rating NUMERIC(3, 2),

 is_fulltime BOOL NOT NULL,

 PRIMARY KEY(carer_name)

);
```

Carer_name -> (rating, is_fulltime)

Carer_name here is the username of a carer so its unique. BCNF. For each non-trivial FD X->Y, X is superkey.

```
CREATE TABLE takes_care (

  carer_name VARCHAR(20) NOT NULL REFERENCES carers(carer_name),

  category_name VARCHAR(20) NOT NULL REFERENCES categories(category_name),

  carer_price NUMERIC(5,2) NOT NULL ,

  PRIMARY KEY(carer_name, category_name)

);
```

BCNF. For each non-trivial FD X->Y, X is superkey.

```
CREATE TABLE working_days (

 working_date DATE NOT NULL,

 carer_name VARCHAR(20) NOT NULL REFERENCES carers(carer_name) ON DELETE
Cascade,

 number_of_pets INT,

 PRIMARY KEY(working_date, carer_name)

);
```

BCNF. For each non-trivial FD X->Y, X is superkey.

```
CREATE TABLE bids(

  start_date DATE NOT NULL,
  end_date DATE NOT NULL,

  carer_name VARCHAR(20) NOT NULL,

  owner_name VARCHAR(20) NOT NULL,

  pname VARCHAR(20) NOT NULL,

  bid_date DATE NOT NULL,

  daily_price NUMERIC NOT NULL,

  is_successful BOOL NULL,

  credit_card_num VARCHAR(16),

  payment_date DATE NOT NULL,

  payment_mode VARCHAR(50) NOT NULL,

  delivery_method VARCHAR(50) NOT NULL,

  review_rating INTEGER NULL,

  review_content VARCHAR(500) NULL,

  review_date DATE NULL,

  FOREIGN KEY (start_date, carer_name) REFERENCES
working_days(working_date, carer_name ) ON DELETE Cascade,

  FOREIGN KEY (end_date, carer_name) REFERENCES working_days(working_date,
carer_name ) ON DELETE Cascade,

  FOREIGN KEY (pname, owner_name) REFERENCES pets(pname, owner_name),

  PRIMARY KEY(start_date, end_date, carer_name, owner_name, pname)

);
```

(start_date, end_date, carer_name, owner_name, pname) -> (bid_date, daily_price, is_successful, credit_card_num, payment_date, payment_mode, delivery_method, review_rating, review_content, review_date)

PAs: start_date, end_date, carer_name, owner_name, pname

BCNF. For each non-trivial FD X->Y, X is superkey.

Section Five:

Present the details of three non-trivial/interesting triggers used in your application by providing an English description of the constraints enforced by each trigger and showing the code of the trigger implementation.

1. **Update daily price of a caretaker for a category is updated whenever they receive a new review rating. The new price is calculated based on the average rating of the specific category. Also updates the overall average rating of the caretaker.**

```
CREATE OR REPLACE FUNCTION update_carer_price()
RETURNS TRIGGER AS
$$
DECLARE bid_pet_category VARCHAR;
DECLARE category_base_price NUMERIC;
DECLARE category_avg_rating NUMERIC;

BEGIN

SELECT belongs INTO bid_pet_category
FROM pets p
WHERE
  p.pname = NEW.pname AND
  p.owner_name = NEW.owner_name;

SELECT base_price INTO category_base_price
FROM categories c
WHERE c.category_name = bid_pet_category;

SELECT (SUM(review_rating) + NEW.review_rating + 0.0) / (COUNT(*) + 1) INTO
category_avg_rating
FROM bids b
WHERE
  b.carer_name = NEW.carer_name AND
  b.review_rating IS NOT NULL AND
  (SELECT belongs
  FROM pets p
  WHERE
    p.pname = b.pname AND
    p.owner_name = b.owner_name
  ) = bid_pet_category;

UPDATE carers SET rating = (
  SELECT (SUM(review_rating) + NEW.review_rating + 0.0) / (COUNT(*) + 1)
  FROM bids b
  WHERE
```

```
    b.carer_name = NEW.carer_name AND
    b.review_rating IS NOT NULL
)
WHERE carers.carer_name = NEW.carer_name;

-- NULL check
IF
  category_avg_rating IS NULL
THEN
  SELECT 0.0 + NEW.review_rating INTO category_avg_rating;
END IF;


IF
  NEW.review_rating IS NOT NULL
THEN
  UPDATE takes_care t
  SET carer_price = category_base_price * (1.00 + category_avg_rating / 10.00)
  WHERE
    t.carer_name = NEW.carer_name AND
    t.category_name = bid_pet_category;
END IF;
RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER update_carer_price_trigger
BEFORE UPDATE OR INSERT ON bids
FOR ROW EXECUTE PROCEDURE update_carer_price();
```

2. **Recalculates the daily price of each caretaker for a particular category whenever the base price is updated.**

```
CREATE OR REPLACE FUNCTION recalculate_carer_prices()
RETURNS TRIGGER AS
$$
declare loopRow RECORD;
declare carer_rating NUMERIC;
BEGIN
FOR loopRow IN
  SELECT DISTINCT carer_name FROM takes_care tc WHERE tc.category_name =
OLD.category_name
LOOP
  SELECT AVG(review_rating) INTO carer_rating
  FROM bids b
  WHERE
    b.carer_name = loopRow.carer_name AND
    OLD.category_name = (
      SELECT belongs
      FROM pets p
      WHERE
```

```
        p.pname = b.pname AND
        p.owner_name = b.owner_name
  );

  UPDATE takes_care
  SET carer_price = CASE
    WHEN carer_rating IS NULL
    THEN NEW.base_price
    ELSE NEW.base_price * (1.00 + carer_rating / 10.00) END
  WHERE carer_name = loopRow.carer_name AND category_name = OLD.category_name;
END LOOP;
RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER recalculate_carer_price_trigger
BEFORE UPDATE ON categories
FOR ROW EXECUTE PROCEDURE recalculate_carer_prices();
```

3. Update number of pets taken care of by a carer for each day when a bid is successful. Also, part-time caretakers cannot take care of more than 2 pets per day if their rating is below 4.00 and for all other caretakers, the limit is 5 pets per day.

```
CREATE OR REPLACE FUNCTION increment_working_day_pet()
RETURNS TRIGGER AS
$$
DECLARE carer_limit INTEGER;

BEGIN

IF
  NOT NEW.is_successful AND NEW.review_rating IS NULL
THEN
  RAISE EXCEPTION 'Bid must be successful in order to leave a review rating!';
ELSIF
  NOT NEW.is_successful
THEN
  RETURN NEW;
END IF;

IF
  (SELECT (is_fulltime OR rating >= 4.00) FROM carers WHERE carers.carer_name =
NEW.carer_name)
THEN
  SELECT 5 INTO carer_limit;
ELSE
  SELECT 2 INTO carer_limit;
END IF;

IF
  (SELECT MAX(number_of_pets)
  FROM working_days
  WHERE carer_name = NEW.carer_name
  AND working_date >= NEW.start_date
  AND working_date <= NEW.end_date) >= carer_limit
THEN
  RAISE EXCEPTION 'Carer will exceed the pet days limit!';
END IF;

UPDATE working_days
SET number_of_pets = number_of_pets + 1
WHERE carer_name = NEW.carer_name
AND working_date >= NEW.start_date
AND working_date <= NEW.end_date;
RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER bid_turns_success
AFTER UPDATE OF is_successful ON bids
FOR ROW EXECUTE PROCEDURE increment_working_day_pet();

-- Function and trigger to ensure that carer_price in takes_care table will not
be lower than the base price --
CREATE OR REPLACE FUNCTION base_price_check()
RETURNS TRIGGER AS
$$
BEGIN
IF
  NEW.carer_price < (SELECT base_price FROM categories c WHERE c.category_name =
NEW.category_name)
THEN
  RAISE EXCEPTION 'Carer price cannot be lower than base price.';
END IF;
RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER base_price_check_trigger
BEFORE UPDATE OR INSERT ON takes_care
FOR ROW EXECUTE PROCEDURE base_price_check();
```

Section Six:

Show the SQL code of three of the most complex queries implemented in your application. Provide an English description of each of these queries.

Query 1.

```
        SELECT owner_name, SUM((end_date - start_date)*daily_price) AS
money_spend
        FROM bids WHERE is_successful = True
        AND EXTRACT(MONTH FROM bid_date) = $1
        AND EXTRACT(YEAR FROM bid_date) = $2
        GROUP BY owner_name
        ORDER BY money_spend DESC`, [
          month, year
      ]);
```

- It ranks the owners by the amount they paid in a particular month.
- Returns a list of (owner_name, paid_amount);
- Group successful bids by owner_name then order by money_spend

Query 2:

```
`SELECT (SUM (b.daily_price) OVER ()) * $1 + $2 AS salary

FROM (

    SELECT generate_series(

      $3,

      (DATE($3) + INTERVAL '1 month' - INTERVAL '1 day')::DATE,

      '1 day'::interval

    )::date AS day

) days_in_month

CROSS JOIN bids b

WHERE

  b.start_date <= day AND b.end_date >= day AND

  b.carer_name = $4

ORDER BY

  day ASC,

  daily_price ASC
```

```
      OFFSET $5;`;

    const getSalary = await pool.query(query, [portion, base_pay,
start_of_month, carer_name, offset]);
```

- It calculates the salary of a particular caer in a particular month.
- Returns a single number
- It will be calculated by salary = ‘portion * petday + base_pay’
- Full time and part time carers have different portion and base_pay

Query 3:

```
       `SELECT COUNT(*)

       FROM bids

       WHERE carer_name = $1

       AND EXTRACT(MONTH FROM start_date) = $2

       AND EXTRACT(YEAR FROM start_date) = $3;`,

     [carer_name, month, year]
```

- It selects total number of pets taken care of by carer in a month;
- Returns a single number
- By filtering with carer_name and check if year and month are the required year and month.

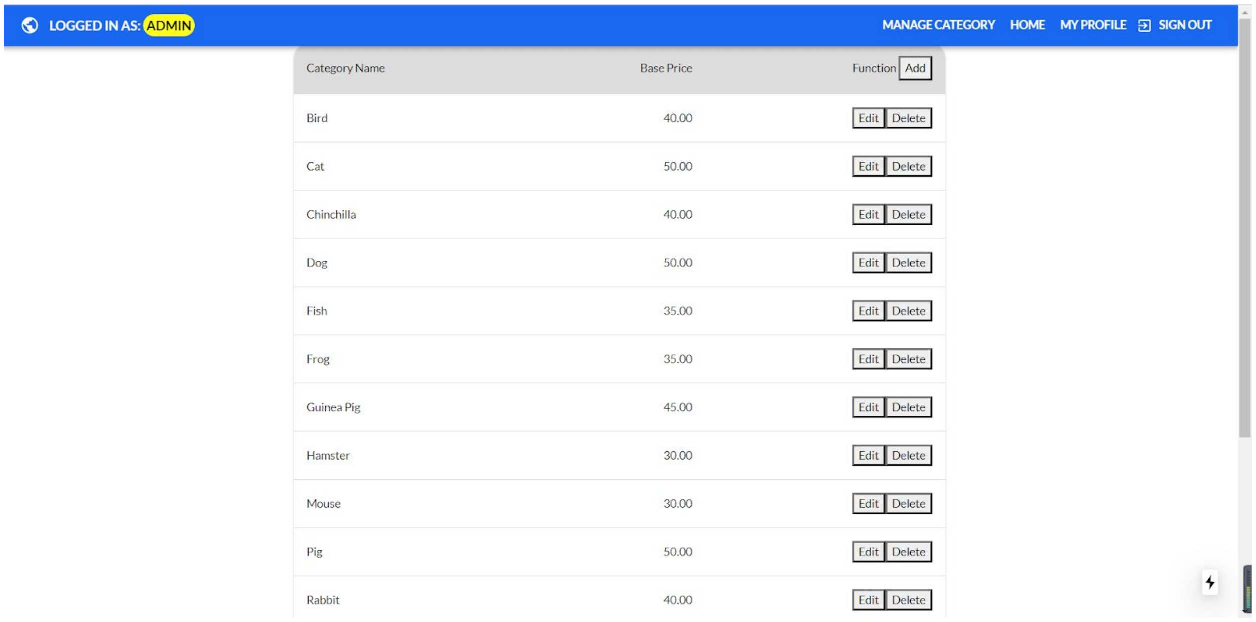
Section Seven:

Specification of the software tools/frameworks used in your project.

- Frontend: framework: react.js next.js UI library: material-UI, antd
- Backend: framework: express.js node.js postgresql

Section Eight:

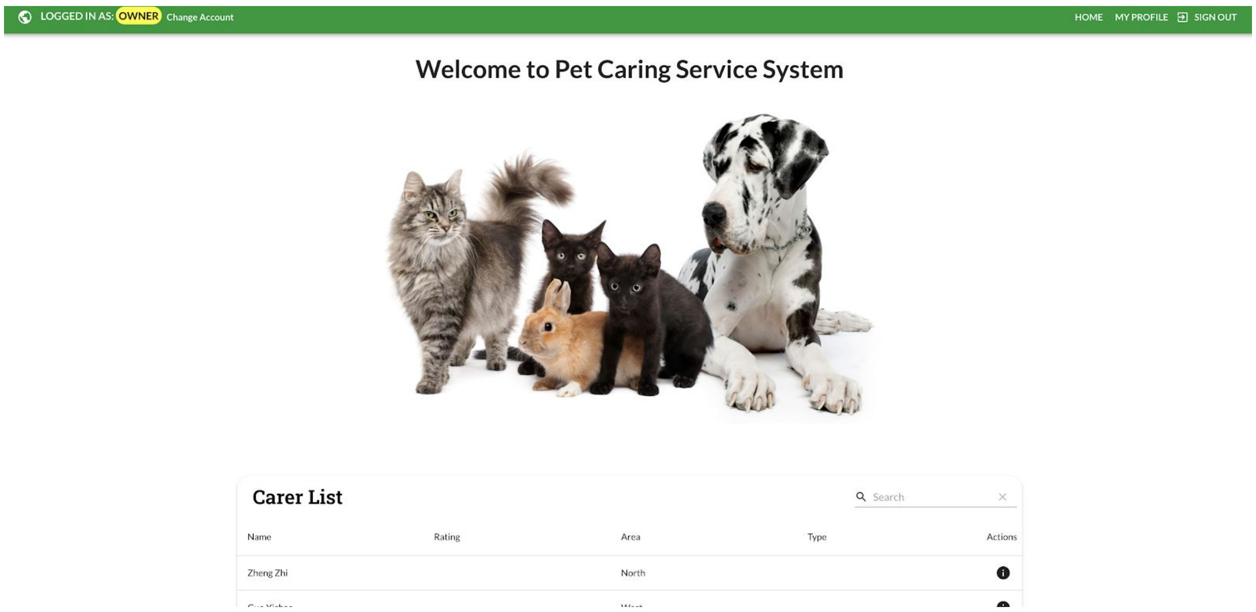
Two or three representative screenshots of your application in action.



LOGGED IN AS: ADMIN

MANAGE CATEGORY HOME MY PROFILE SIGN OUT

Category Name	Base Price	Function Add
Bird	40.00	Edit Delete
Cat	50.00	Edit Delete
Chinchilla	40.00	Edit Delete
Dog	50.00	Edit Delete
Fish	35.00	Edit Delete
Frog	35.00	Edit Delete
Guinea Pig	45.00	Edit Delete
Hamster	30.00	Edit Delete
Mouse	30.00	Edit Delete
Pig	50.00	Edit Delete
Rabbit	40.00	Edit Delete



LOGGED IN AS: OWNER Change Account

HOME MY PROFILE SIGN OUT

Welcome to Pet Caring Service System

Carer List

Search

Name	Rating	Area	Type	Actions
Zheng Zhi		North		●

Section Nine:

A summary of any difficulties encountered and lessons learned from the project.

- Start project early
- Update code change to other group member so everyone is on the same page
- Have a clear step by step process of implementation. E.g. when to set up what, until to what extend