CS2102 Project Report
Team 11

Zeon Chua Feiyi A0199845E
Wang Zesong A0211781L
Zhang Shukai A0180048J
Ni Yilun A0205147M
Feng Yuan A0205059J

1. Technology Used

1.1. Platform

- Flask micro web framework for python,
- Flask-Login, Flask-admin, Flask-User for validation
- Flask-SQLAlchemy
- SQL capability
- Flask-WTF and WTForms for creating forms to use in HTML
- Flask-Bcrypt for encrypting password dat
- Flask-Table to print out tables
- Jinja2
- MarkupSafe
- psycopg2
- itsdangerous
- SQLAlchemy for the actual implementation of SQL,
- Werkzeug
- gunicorn to deploy the app to Heroku

1.2. Database

- PostgreSQL v13.0

1.3. Languages used

- Python 3.9.0 for Application stack
- SQL for database Stack

1.4. Data generator

- generatedata.com

2. Members and Responsibilities

Zesong - back end, triggers, SQL queries

Shukai - back end, triggers, SQL queries

Feng Yuan - front end, SQL queries

Yilun - back end, triggers, SQL queries

Zeon - front end, SQL queries

3. Application's Data Requirements and Functionalities

3.1. Functionalities

All users can

- Create accounts and login
- View and update their profiles

Pet Owners can

- View and search for caretakers and view their profiles
- Add / Edit / Delete pets
- Make / Edit / Delete bids for caretakers for their pet
- Submit ratings and reviews for finished transactions.

Caretakers can

- View their working days and total salary for the month
- View and select bids for them
- Add availability slots
- Apply for leave days
- Add / remove pet categories to take care of
- Finish bids after bids end

Admins can

- View the summary data of caretakers
- Adjust the daily price under each pet category and caretaker rating
- Delete any users

3.2. Data Requirements

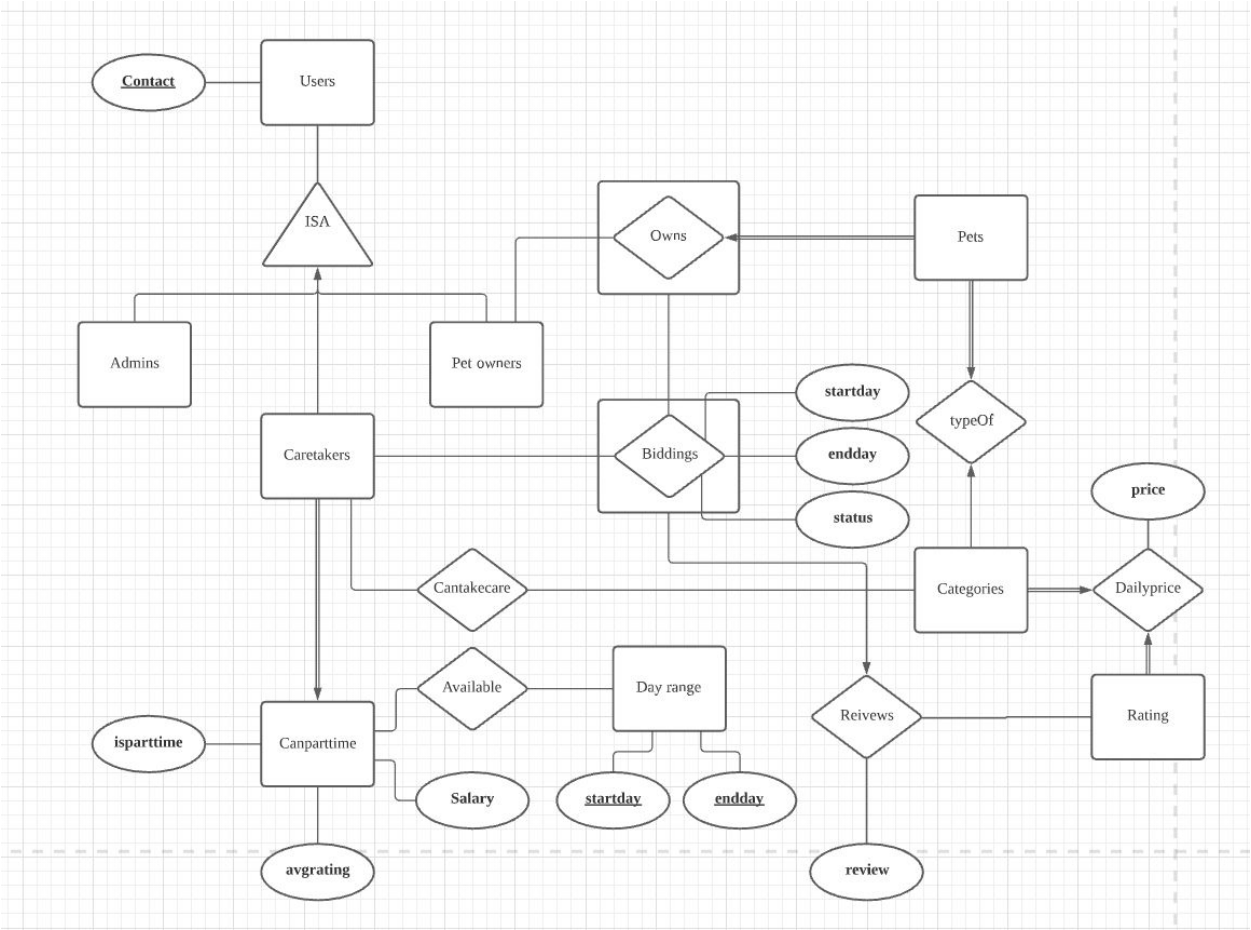
- User role can only be either of the following values: 'petowner', 'admin', 'caretaker'
- Pet category can only be the either of following values: 'bird', 'cat', 'dog','hamster', 'shark', 'fish', 'rabbit', 'spider', 'turtle', 'insect', 'horse', 'duck', hedgehog', 'snake', and 'guinea pig'
- Contacts that are registered needs to be distinctive for distinctive users
- End day should be after start day
- Postal code must be an integer number in the range from 100000 to 999999
- Rating of each review can only be an integer from 1 to 5
- Payment mode can only be 'creditcard' or 'cash'
- Delivery mode can only be 'delivery', 'pickup' or 'pcstransfer'
- Age of the pet can only be an integer between 0 and 99
- Daily price under each pet category and rating should be a positive integer

3.3. Justification for the use of ORM on registration and login query

Use of ORM is because the object has hidden attributes which are linked to other python libraries such as login_required or roles_required and current_user.

4. ER Diagram

4.1. Diagram



4.2. Uncaptured Constraints

These are our constraints that are uncaptured by ER model:

- Every full time caretaker has to work 2 x 150 consecutive days a year
- Every Full time caretaker have a limit of up to 5 pets at any one time
- Every part time caretaker also has a limit of up to 5 pets at any one time

- Every part time caretaker cannot take care of more than 2 pets at one time unless they have a good rating (defined as more than or equal to 3 out of 5)
- A full time caretaker is taken as available until they take leave
- The base daily price for a full time caretaker increases with the rating but will never be below the base price
- Full time caretakers cannot apply for leave unless they have no pets under their care
- A full-time Caretaker will receive a salary of $3000 per month for up to 60 days where they took care of any pets in a year. For any extra days, they will receive 80% of their price as bonus. For part-time caretakers, the PCS will take 25% of their price as payment (caretaker receives 75% of stated price).
- User ISA relationship is overlapping and covering
- Caretaker relationship is covering and not overlapping

4.3. Justifications for non-trivial design decisions in the ER model

- Separate Canparttime from Users table → 3NF
  A separate Canparttime Table is created in addition to Users Table to store the attribute “is part time”, “average rating” and “salary” so as to maintain 3NF rule for Users Table. If the above three attributes are stored in Users Table, null values will be assigned to these attributes for admins and pet owners, and this violates the 3NF rule.

- Remove finished bidding and insert to Reviews
  Trigger is used to delete finished biddings in the Biddings Table and insert these biddings into the Reviews table when the status is updated to “end”. This is to avoid the situation where deletion of pets might lead to deletion of finished biddings about these pets if using foreign keys to connect Reviews table with Biddings table and Pets table.

- Merge leave days for full time with available days for part-time
  Based on the criteria, full-time caretakers are considered as available unless they take leave, while part-time caretakers need to declare available dates for two years. Hence, date periods inserted by full-time caretakers indicate their unavailable days, while date periods inserted by part-time caretakers indicate their available days. These dates are stored as “startday” and “endday” in the Available Table, and the attribute “isparttime” determines whether these date periods refer to available days or unavailable days.

- Roles table → allow roles_required feature

In our ER diagram we separated available days and leaves into separate days. This allows us to keep track of when employees take leaves, and also allows for part-time caretakers to choose their availability days for flexibility.

When a bidding is accepted and confirmed it will be moved into another table. This allows us to keep the biddings in less volatile records for accountability, in cases where a pet owner decides to delete a pet and the resulting cascade would not cause loss of transactional data.

5. Relational Schema

All the tables that we have created for this application are in BCNF or 3NF.

```
CREATE TABLE users (
    username VARCHAR NOT NULL,
    contact BIGINT PRIMARY KEY NOT NULL,
    card VARCHAR NOT NULL,
    password VARCHAR NOT NULL,
    usertype VARCHAR NOT NULL,
    postalcode BIGINT NOT NULL
);

CREATE TABLE categories (
    category VARCHAR PRIMARY KEY NOT NULL
);
```

```
CREATE TABLE role (
    name VARCHAR NOT NULL UNIQUE
);
```

```
CREATE TABLE user_roles (
    contact BIGINT PRIMARY KEY NOT NULL REFERENCES public.users(contact),
    usertype VARCHAR NOT NULL REFERENCES public.role(name)
);
```

```
CREATE TABLE canparttime (
    ccontact BIGINT PRIMARY KEY NOT NULL REFERENCES public.users(contact),
    isparttime BOOLEAN NOT NULL,
    avgrating FLOAT NOT NULL,
    petday INT NOT NULL,
    salary INTEGER NOT NULL
);
```

```
CREATE TABLE dailyprice (
    category VARCHAR NOT NULL REFERENCES public.categories(category),
    rating INTEGER NOT NULL,
    price INTEGER NOT NULL,
    PRIMARY KEY (category, rating)
);
```

```
CREATE TABLE pets(
    petname VARCHAR NOT NULL,
    pcontact BIGINT NOT NULL REFERENCES public.users(contact) ON DELETE CASCADE,
    category VARCHAR NOT NULL REFERENCES public.categories(category),
    age INTEGER NOT NULL,
    PRIMARY KEY (petName, pcontact)
);
```

```
CREATE TABLE available (
    startday DATE NOT NULL,
    endday DATE NOT NULL CHECK(endday - startday >= 0),
    ccontact BIGINT NOT NULL REFERENCES public.users(contact) ON DELETE CASCADE,
    PRIMARY KEY (ccontact, startday, endday)
);
```

```
CREATE TABLE cantakecare (
    ccontact BIGINT NOT NULL REFERENCES public.users(contact) ON DELETE CASCADE,
    category VARCHAR REFERENCES public.categories(category),
    PRIMARY KEY (ccontact, category)
);
```

```
CREATE TABLE biddings(
  pcontact BIGINT NOT NULL,
  ccontact BIGINT NOT NULL REFERENCES public.users(contact) ON DELETE CASCADE,
  petname VARCHAR NOT NULL,
  startday DATE NOT NULL,
  endday DATE NOT NULL CHECK(endday - startday >= 0),
  /* paymentmode is either 'creditcard' or 'cash' */
  paymentmode VARCHAR NOT NULL,
  /* delivermode is either 'pet owner deliver' or 'pick up' or 'transfer through
PCS' */
  deliverymode VARCHAR NOT NULL,
  /* status can be pending, success, fail or end */
  status VARCHAR NOT NULL,
  PRIMARY KEY (pcontact, ccontact, petname, startday, endday),
  FOREIGN KEY (pcontact, petname) REFERENCES
      public.pets(pcontact, petname) ON DELETE CASCADE

);
```

```
CREATE TABLE reviews(
    pcontact BIGINT NOT NULL,
    ccontact BIGINT NOT NULL,
    petname VARCHAR NOT NULL,
    startday DATE NOT NULL,
    endday DATE NOT NULL CHECK(endday - startday >= 0),
    rating INTEGER NOT NULL CHECK(rating <= 5 AND rating >= 0),
    review VARCHAR NOT NULL,
    PRIMARY KEY (pcontact, ccontact, petname, startday, endday)
);
```

6. Interesting Triggers

6.1. Trigger to move ended bidding to Reviews table

```
---------------------------------- Bid end trigger
CREATE OR REPLACE FUNCTION moveToReview()
RETURNS TRIGGER AS $$
BEGIN
 INSERT INTO Reviews VALUES (
       NEW.pcontact, NEW.ccontact, NEW.petname,
       NEW.startday, NEW.endday, 5.0, '');
 DELETE FROM biddings WHERE pcontact = NEW.pcontact and ccontact = NEW.ccontact
and petname = NEW.petname
   and startday = NEW.startday and endday = NEW.endday;
 RETURN NULL;
END;
$$
LANGUAGE PLPGSQL;

DROP TRIGGER IF EXISTS
finishedBidding ON biddings;
CREATE TRIGGER finishedBidding
   AFTER UPDATE OF status ON biddings
   FOR EACH ROW
     WHEN (NEW.status = 'end')
       EXECUTE FUNCTION moveToReview();
---------------------------------- End of bid end trigger
```

When the status of a row in bidding is updated to ‘end’, we will move the data in that row to the Reviews table. After that the newly updated bidding will be removed from biddings table. The deletion and insertion are done by calling moveToReview() function. It is deleted to remove the foreign key constraint on that row, so that the user can delete the corresponding pet after the bidding is finished, without causing integrity error to the database.

6.2. Trigger to calculate the caretaker’s average ratings

```
------------------------------------ Calculate rating trigger
CREATE OR REPLACE FUNCTION log_new_successful_bidding()
 RETURNS TRIGGER
 LANGUAGE PLPGSQL
 AS
$$
BEGIN
 IF NEW.status <> OLD.status  AND NEW.status = 'success' THEN
    INSERT INTO employee_audits(employee_id,last_name,changed_on)
    VALUES(OLD.id,OLD.last_name,now());
 END IF;

 RETURN NEW;
```

```
END;
$$

CREATE OR REPLACE FUNCTION calculate_avg_rating()
RETURNS TRIGGER AS $$
DECLARE
 ratingsum FLOAT;
 countsum FLOAT;
 avgnum FLOAT;
BEGIN
 SELECT SUM(R.rating), COUNT(*) INTO ratingsum, countsum
 FROM reviews R
 WHERE R.ccontact = NEW.ccontact;
 SELECT CAST((ratingsum / countsum) AS FLOAT) INTO avgnum;
 UPDATE canparttime  SET avgrating = avgnum
   WHERE ccontact = NEW.ccontact;
 RETURN NULL;
END;
$$ LANGUAGE PLPGSQL;

DROP TRIGGER IF EXISTS
calculate_avg_rating_trigger ON reviews;
CREATE TRIGGER calculate_avg_rating_trigger
 AFTER UPDATE OF rating OR INSERT
 ON reviews
 FOR EACH ROW
 EXECUTE FUNCTION calculate_avg_rating();
------------------------------------ End of calculate rating trigger
```

When the bidding is reviewed, the default rating stored previously in the reviews table will be updated. The insertion of reviews or update of attribute rating will call the 'calculate_avg_rating_trigger' trigger which calls the 'calculate_avg_rating()' function. In this function, we use a query to find the sum and counts of ratings of the selected caretaker and store them in local variables. Then, we divide the sum by the total counts to obtain an average rating value. After that, we update the 'avgrating' attribute in the 'canparttime' table with the computed value.

6.3. Trigger to calculate the caretaker's total salary

```
------------------------------------------- Salary trigger
CREATE OR REPLACE FUNCTION
calcMoney(start DATE, endy DATE, price INTEGER)
RETURNS INTEGER AS $$
   BEGIN RETURN price * (endy - start);
   END; $$
LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION addSalary()
```

```
RETURNS TRIGGER AS $$
BEGIN
 UPDATE canparttime cp SET petday = petday + NEW.endday - NEW.startday;

 UPDATE canparttime cp SET salary = salary + (
 (
  SELECT calcMoney(NEW.startday, NEW.endday,
    (SELECT price FROM dailyprice WHERE
     category = (SELECT category FROM pets P
     WHERE NEW.petname = P.petname AND NEW.pcontact = P.pcontact)
     AND
     rating = (SELECT CEIL(avgrating) FROM canparttime cpt
     WHERE NEW.ccontact = cpt.ccontact)
    )
   )
)) * (
  CASE
   WHEN isparttime THEN 0.75
   WHEN petday >= 60 THEN 0.80
   ELSE 0
  END
)

 WHERE NEW.ccontact = cp.ccontact;
 RETURN NULL;
END;
$$
LANGUAGE PLPGSQL;

DROP TRIGGER IF EXISTS
newSuccessBidding ON biddings;
CREATE TRIGGER newSuccessBidding
   AFTER UPDATE OF status ON biddings
   FOR EACH ROW
     WHEN (NEW.status = 'success')
       EXECUTE FUNCTION addSalary();
------------------------------------------- End of salary trigger
```

When a bidding is accepted, the status of that row changes from ‘pending’ to ‘success’. This calls the ‘newSuccessBidding’ trigger which calls the addSalary function. We then used a subquery to obtain the daily price of this caretaker based on the DailyPrice table using his rating and the pet category in the bid. Then function calcMoney takes in the daily price and start and end day of the updated bidding and returns the salary for this bid. The corresponding caretaker’s total salary is then updated with total salary = previous salary + salary for this bid.

7. Complex Queries

7.1. Verify the caretaker does not take more than max number of bidding

```
flag = True
   for selected in daterange(datetime.strptime(startday, '%Y-%m-%d'),
datetime.strptime(endday, '%Y-%m-%d')):
       query = """SELECT COUNT (*) FROM biddings WHERE '{}' - startday >= 0 AND
endday - '{}' >= 0
           AND ccontact = '{}' AND status = 'success'""".format(selected,
selected, ct)
       count = db.session.execute(query).fetchone()
       if parttime.isparttime == True and parttime.avgrating < 3:
           if count[0] > 2:
               flag = False
               count[0] = 0
               break
       else:
           if count[0] > 5:
               flag = False
               count[0] = 0
               break
```

To verify the new bidding does not lead to one caretaker having more than 5 pets (or 2 pets for part time caretakers with rating less than 3.0), we calculate the number of successful biddings that covers the period of the new bidding. For each day in the period of the new bidding, we count the number of existing successful bids that also covers that day. If for any day it exceeds the max limit, the bidding cannot be accepted.

7.2. Search the caretakers based on contact or postal code

```
checkContinuous = """
           SELECT SUM(diff)
           FROM
           (
           SELECT FLOOR(COALESCE(EXTRACT(DAY FROM (f2.st - date_trunc('year',
f2.st))), 365) / 150) diff
           from (
           (SELECT startday AS st, endday AS en FROM available WHERE ccontact =
:cc)
           UNION (SELECT :startday, :endday)
           ORDER BY st LIMIT 1
           ) AS f2
           UNION
           SELECT FLOOR(COALESCE(f2.st - lag(f2.en) over (order by f2.st, f2.en),
0) / 150) diff
           FROM (
           SELECT min(st) as st, max(en) as en
           FROM (SELECT st, en,
```

```
                    max(new_start) OVER (ORDER BY st,en) AS left_edge
                    FROM (SELECT st, en,
                        CASE WHEN st < max(le) OVER (ORDER BY st,en) THEN null
ELSE st END AS new_start
                        FROM (
                            ((SELECT startday AS st, endday AS en, lag(endday)
OVER (ORDER BY startday,endday) AS le FROM available WHERE ccontact = :cc)
                            UNION (SELECT :startday, :endday, :endday))) s1) s2)
s3
                    GROUP BY left_edge
            LIMIT 1
            ) AS f2
            UNION
            SELECT FLOOR(COALESCE(EXTRACT( DAY FROM (date_trunc('year',f2.en) +
INTERVAL '1' YEAR) - f2.en), 365) / 150)
            FROM (
            (SELECT startday AS st, endday AS en
            FROM available WHERE ccontact = :cc) UNION
            (SELECT :startday, :endday)
            ORDER BY en DESC LIMIT 1
            ) AS f2) AS everything
            """
            parameters = dict(cc = ccontact, startday = startday, endday = endday)
            numberOfperiods = db.session.execute(checkContinuous,
parameters).fetchall()
            if numberOfperiods[0][0] < 2:
```

To check if there are 2 continuous 150 days, we divide each date range by 150 and sub up the integer section of the result. If sum is greater than 2 then it satisfies the condition.

7.3. Submitting biddings

```
petNameQuery = """SELECT petname FROM pets
                    WHERE pcontact = '{}' AND
                    category in (SELECT category FROM cantakecare WHERE ccontact
= '{}')""".format(contact, cn)
   petNames = db.session.execute(petNameQuery).fetchall()
```

When submitting biddings, the user can only select the user's own pets, which is a category that the caretaker can take care of. For instance, if an owner has a cat and a dog, while the caretaker is only able to take care of cats, then only the owner's cat will be in the selection field.

```
intersection = """
            SELECT 1
            FROM   (SELECT min(st) as st, max(en) as en
                    FROM (SELECT st, en,
                        max(new_start) OVER (ORDER BY st,en) AS left_edge
                    FROM (SELECT st, en,
```

```
                        CASE WHEN st < max(le) OVER (ORDER BY st,en) THEN null
ELSE st END AS new_start
                  FROM (SELECT startday AS st, endday AS en, lag(endday) OVER
(ORDER BY startday, endday)
                      AS le FROM available WHERE ccontact = '{}') s1) s2) s3
                  GROUP BY left_edge) AS f2
          WHERE tsrange('{}', '{}', '[]') * tsrange(f2.st, f2.en, '[]') =
tsrange('{}', '{}', '[]');
          """.format(cn, startday, endday, startday, endday)
          hasFullOverage = db.session.execute(intersection).fetchone()
          print(hasFullOverage, flush=True)
          if(not hasFullOverage):
              isValidPeriod = False
```

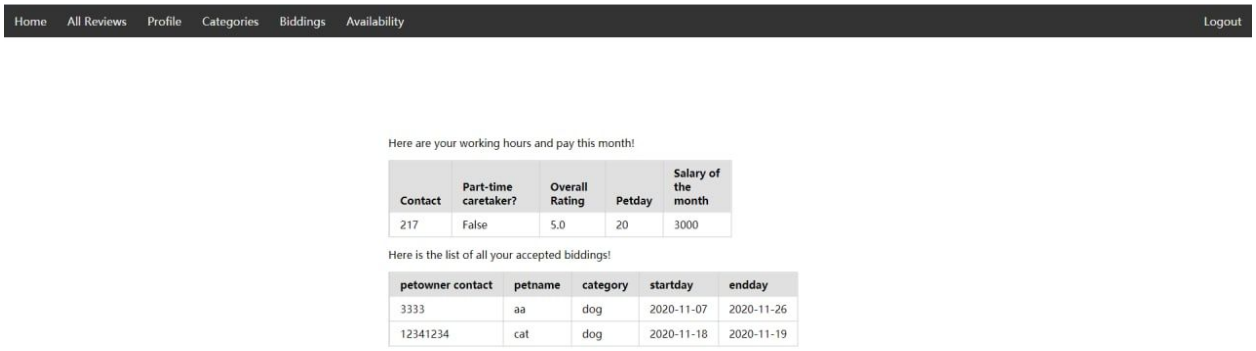
When a pet owner submits the bid, the database will return a list of date ranges. If the caretaker that he is bidding for is a part-time caretaker, the range will represent available days. By calculating the bid date range and available date range and checking if it is a subset of any given date range, we can verify whether the caretaker is completely available for that period.

```
overLapQuery = """
          SELECT 1
          FROM   (SELECT min(st) as st, max(en) as en
                  FROM (SELECT st, en,
                      max(new_start) OVER (ORDER BY st,en) AS left_edge
                  FROM (SELECT st, en,
                          CASE WHEN st < max(le) OVER (ORDER BY st,en) THEN null
ELSE st END AS new_start
                  FROM (SELECT startday AS st, endday AS en, lag(endday) OVER
(ORDER BY startday, endday)
                       AS le FROM available WHERE ccontact = '{}') s1) s2) s3
                  GROUP BY left_edge) AS f2
          WHERE tsrange('{}', '{}', '[]') && tsrange(f2.st, f2.en, '[]');
          """.format(cn, startday, endday)
          hasOverlap = db.session.execute(overLapQuery).fetchone()
          print(hasOverlap, flush=True)
          if(hasOverlap):
              isValidPeriod = False
```

If the caretaker is a full time caretaker instead, the range represents the leave periods of the caretaker. By calculating if the bid date range and available date range has any overlaps, we can verify whether the caretaker is completely available for the period.

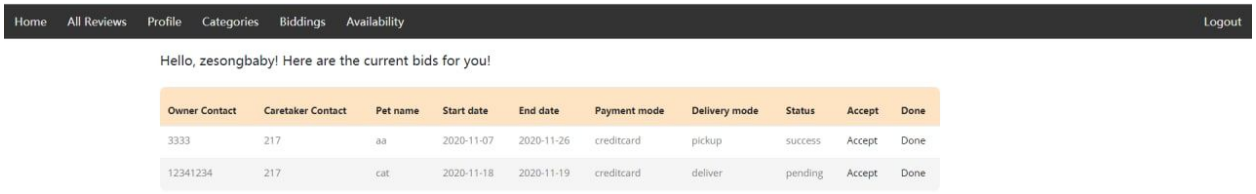
8. Application Screenshots and Description

Caretaker main page



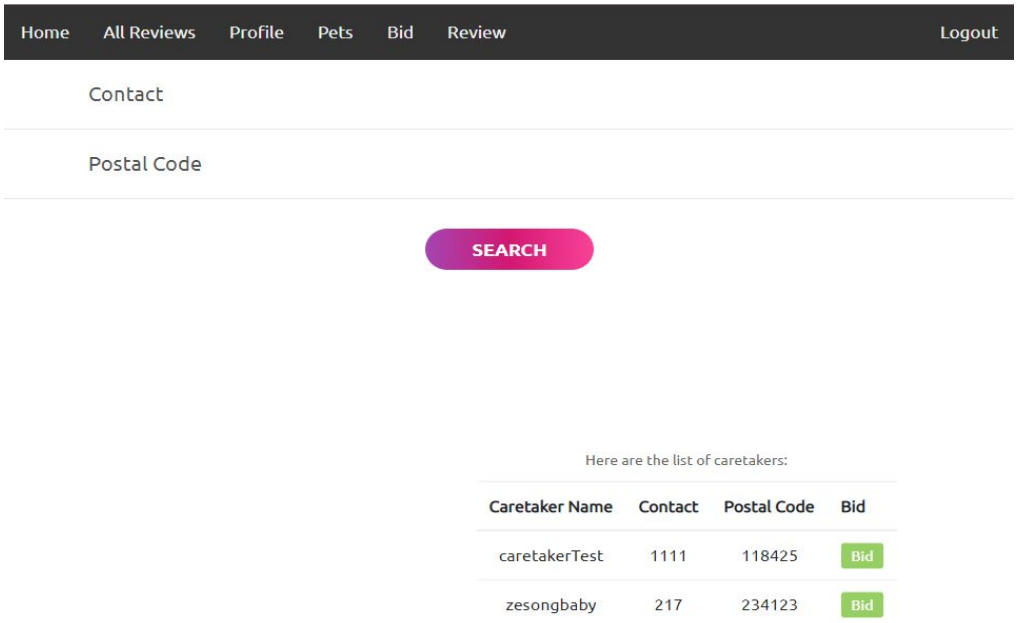
The caretaker can check the bids accepted here, together with how many pet days and the salary for the month.

Pet owner placing biddings page



Caretakers are able to accept bids, or mark bids as done here..

Pet owner main page:



Here, pet owners can check bids for caretakers or search for caretakers .

9. Summary and Lessons Learnt

This project was very challenging and difficult as we had very little front-end knowledge, such as making websites with HTML and CSS. A lot of time was spent trying to make the front-end functional and this manifested in our queries being comparatively simpler as we were not trying to work with our SQL skills but instead trying to fit our SQL skills into our limited knowledge of front-end coding.

Notable difficulties included learning how to write triggers. There was very little experience with triggers in the curriculum and it accounted for a good portion of the project marks so we had to organise a meeting to figure out how to utilise triggers and where to implement them in the project.

Debugging the front end was also difficult due to our inexperience. More often than not when an expected behaviour did not work it was due to a front end bug that was not diagnosed properly, such as the sending and editing of forms. After reading logs over a few weeks the debugging was easier as we were able to identify the sources of the problems better however.