CS2102 Database System: PCS Application

Chua Cheng Ling	Colin Ng Chenyu	Lee Yong Jie Richard	Nikole Pang	Tristan Chua Zhihao
A0187493M	A0189871J	A0170235N	A0187645N	A0180348A

1 Project Responsibilities

- ER Diagram: Everyone
- Teach how to use GitHub: Richard
- Schema: Nikole
- Triggers, Functions and Procedures: Tristan
- Drafting/Drawing of Design of Webpage: Everyone
- Creating of Design of Webpage:
  - FlaskApp: Cheng Ling
  - HTML/CSS: Colin and Nikole
  - Chat Function: Richard
- Creation of Fake Data for Testing: Richard
- Testing/Debugging of Functions: Everyone
- Report: Everyone
- Video: Everyone

2 Software Tools/Frameworks

- Heroku
- Python & Flask
- PostgreSQL & PL/pgSQL
- HTML/CSS

3 Application Data Requirements and Functionalities

- Admin
  - Only the Admin can add new Full-Time Caretakers.
- Pet Owner
  - Pet Owners are able to view their pending and accepted bookings for the next 2 weeks on their homepage.
  - Pet Owners are able to book Caretakers' service at most 2 weeks prior to the start date.
  - Pet Owners are only able to book up to 2 weeks worth of care-taking services at a time.
  - Pet Owners are able to give review of past completed transactions on their homepage.
  - Pet Owners are able to view all of their transactions regardless of its status (Completed/Pending/Accepted/Rejected).
  - Pet Owners are able to communicate with the Caretakers via the chat box upon confirmation of the booking.
- Caretaker
  - Caretakers are able to view their jobs for the next 2 weeks on their homepage.
  - Caretakers are able to view all of their transactions regardless of its status (Completed/Pending/Accepted/Rejected).
  - Caretakers are able to communicate with the Pet Owners via the chat box upon confirmation of booking.
  - Full-Time Caretakers can only apply for leave minimally one month in advance of the date of leave.
  - Full-Time Caretakers can only apply for leave of that specific year.
  - Part-Time Caretakers' desired price for each pet type has to be higher or equal to the price set by Admin for Full-Time Caretakers.
  - Caretakers are able to see the full breakdown of their salary for a particular month.
- Interesting and Non-trivial Aspects
  - Our group prompts users on the upcoming events for the next 2 weeks to keep them updated/reminded.
  - Our group has a chat function to allow communication between both parties to liaise on drop-off/pick up of the pet.

4 Entity-Relationship Model

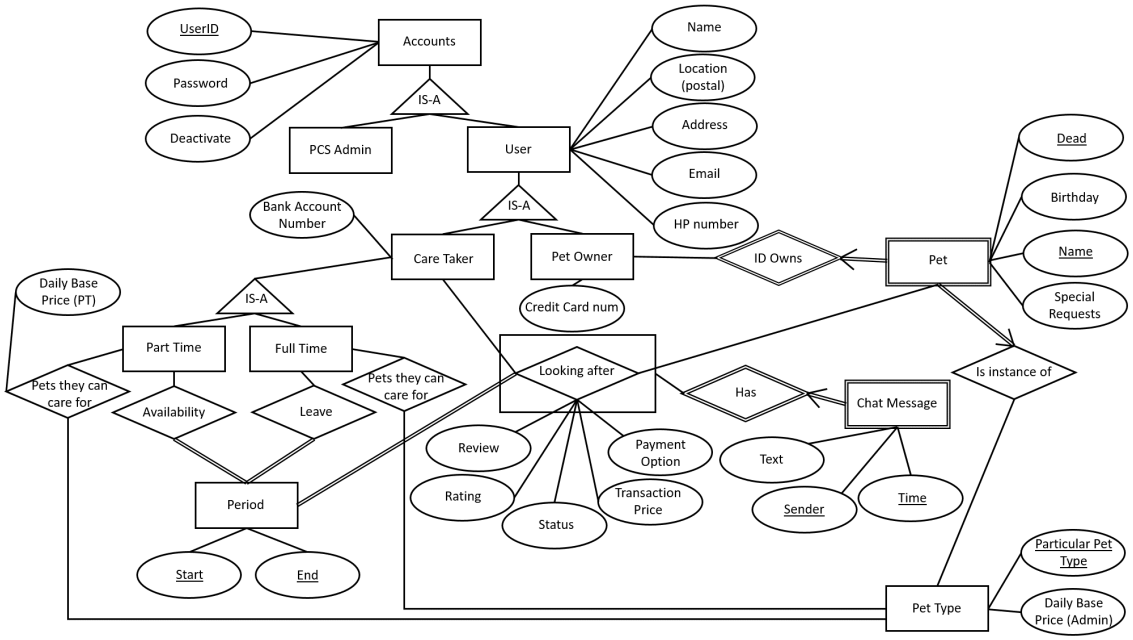

Figure 1: ER Diagram

- Entity and relationship requirements
  - The application can have many accounts, each storing their own user ID, password and deactivation status.
    - An account can be uniquely identified by the user ID.
    - The deactivation status is 'False' is the account is active, 'True' if the account is deactivated.
  - An account can either be an Admin account or a normal user.
    - This hierarchy satisfies covering constraints.
    - This hierarchy does not satisfy overlapping constraints.
  - A normal user has a name, location (postal), address, H/P number and email.
  - A normal user can either be a Caretaker or a Pet Owner.
    - This hierarchy satisfies covering constraints.
    - This hierarchy satisfies overlapping constraints.
  - A Pet Owner may have a credit card number saved for future payments.
  - A Pet Owner owns pets.
  - A pet has a birthday, name, special requests and a dead flag.
    - A pet can be uniquely identified by its pet name and dead flag knowing the Pet Owner. (Weak entity - identity dependency) This means an owner cannot have two (live) pets with the same name.
    - The dead flag is 0 if the pet is still in the care of the Pet Owner. If the Pet Owner 'deletes' the pet, the dead flag will be set to 1, or incrementally higher integers if the Pet Owner had previously-deleted pets with the same name.
  - A pet is an instance of a pet type.
  - A pet type contains details about the particular pet type and also its daily base price (set by the Administrator).
  - A Caretaker may choose what kind of pets they can care for.
    - A Part-Timer may have their own price set for the particular pet type.

  - A Caretaker can either be a Part-Timer or a Full-Timer.
    * This hierarchy satisfies covering constraints.
    * This hierarchy does not satisfy overlapping constraints.
  - A Part-Timer can indicate his/her availability for a period, comprising of a start and end date (inclusive).
  - A Full-Timer can indicate his/her leave periods, comprising of a start and end date (inclusive).
  - A Caretaker may look after a pet for a particular period, this will be termed as a transaction.
  - A transaction has a status, transaction price.
    * The status can either be 'Pending', 'Rejected', 'Accepted' or 'Completed' depending on the state of the transaction.
  - Each transaction can be accompanied by a series of chat messages which contains the time sent and the text message itself.
    * A chat message can be uniquely identified with the sender (an integer indicating if it's the Pet Owner, Caretaker or system), time and the transaction's key (Caretaker ID, Pet Owner ID, Pet Name, dead flag and time period) (Weak entity - identity dependency).

- Numerical Diagram-Enforceable Constraints
  - Pet Owner - Pets
    * A Pet Owner may own multiple pets.
    * A pet must be owned by one owner.
  - Pets - Pet type
    * A pet must be classified under one pet type.
    * A pet type classification may encompass many different pets.
  - Caretaker - Pet type
    * A Caretaker may care for various kinds of pet types.
    * Many Caretakers may care for the same pet type.
  - Part-Timer - Availabilities
    * A Part-Timer may indicate availabilities for multiple periods.
    * Many Part-Timers may indicate availabilities for the same period.
  - Full-Timer - Leave
    * A Full-Timer may apply for leave for multiple periods.
    * Many Full-Timers may apply for leave for the same period.
  - Looking after
    * A Caretaker may look after many pets.
    * Many Caretakers may look after the same pet for a different time period.

- Numerical diagram-nonenforceable constraints
  - A Part-Timer's availability periods cannot overlap.
  - A Full-Timer's leave periods cannot overlap.
  - A Full-Timer cannot apply for leave if the (150 consecutive working days * 2) requirement cannot be fulfilled after the application of leave.
  - A Caretaker cannot look after his/her own pet.
  - A review/rating for a transaction can only be filled after the transaction is marked as 'Completed'.
  - A pet that is being looked after by the Caretaker must be classified as one of the pet types that the Caretaker can care for.
  - Any two transactions that involve the same pet and do not have the status 'Rejected' cannot clash in terms of dates.

  - The number of combined 'Pending' and 'Accepted' transactions for any Caretaker for any particular day cannot exceed 5 for a Full-Timer or highly rated Part-Timer, and cannot exceed 2 for a non-highly rated Part-Timer.
  - An 'Accepted' or 'Completed' transaction for a Caretaker must fall within his/her availabilities (if he/she is a Part-Timer) and must not fall within his/her leave periods (if he/she is a Full-Timer).

- Other assumptions
  - The Admin will create a new account if he/she wishes to use the PCS.
  - Part-time Caretakers cannot become Full-time Caretakers and vice versa, without creating a new account.

5 Relational Schema

Below is the Relational Schema that our group is using for our project.

```
-- GENERAL ACCOUNTS, USER AND ADMIN --

CREATE TABLE Accounts (
        userid          VARCHAR PRIMARY KEY,
        password        VARCHAR NOT NULL,
        deactivate      BOOLEAN DEFAULT FALSE
);

CREATE TABLE Users (
        userid          VARCHAR PRIMARY KEY REFERENCES Accounts (userid),
        name            VARCHAR NOT NULL,
        postal          INTEGER NOT NULL,
        address         VARCHAR NOT NULL,
        hp              INTEGER NOT NULL,
        email           VARCHAR NOT NULL UNIQUE
);

CREATE TABLE Admin (
        userid          VARCHAR PRIMARY KEY REFERENCES Accounts (userid)
);


-- PET OWNER, CARETAKER, PET TYPE --

CREATE TABLE Pet_Owner (
        po_userid       VARCHAR PRIMARY KEY REFERENCES Users (userid) ON UPDATE CASCADE,
        credit          CHAR(16) DEFAULT NULL
);

CREATE TABLE Caretaker (
        ct_userid       VARCHAR PRIMARY KEY REFERENCES Users (userid) ON UPDATE CASCADE,
        bank_acc        CHAR(10) NOT NULL,
        full_time       BOOLEAN DEFAULT FALSE
);

CREATE TABLE Pet_Type (
        pet_type        VARCHAR PRIMARY KEY NOT NULL,
        price           FLOAT4 NOT NULL
);


-- VALID PET TYPES FOR CARETAKER, PET --

CREATE TABLE PT_validpet (
        ct_userid       VARCHAR REFERENCES Caretaker (ct_userid) ON UPDATE CASCADE,
        pet_type        VARCHAR NOT NULL REFERENCES Pet_type (pet_type),
        price           FLOAT4 NOT NULL,
        PRIMARY KEY (ct_userid, pet_type)
);

CREATE TABLE FT_validpet (
        ct_userid       VARCHAR REFERENCES Caretaker (ct_userid) ON UPDATE CASCADE,
        pet_type        VARCHAR NOT NULL REFERENCES Pet_type (pet_type)
);

CREATE TABLE Pet (
        po_userid       VARCHAR NOT NULL REFERENCES Pet_Owner (po_userid),
        pet_name        VARCHAR NOT NULL,
        dead            INTEGER NOT NULL DEFAULT 0,
        birthday        DATE DEFAULT NULL,
        spec_req        VARCHAR DEFAULT NULL,
        pet_type        VARCHAR NOT NULL REFERENCES Pet_Type (pet_type),
        PRIMARY KEY (po_userid, pet_name, dead)
);
```

```
-- PART-TIME, AVAILABILITY, FULL-TIME, LEAVE --

CREATE TABLE PT_Availability (
        ct_userid       VARCHAR NOT NULL REFERENCES Caretaker (ct_userid),
        avail_sd        DATE NOT NULL,
        avail_ed        DATE NOT NULL,
        CHECK (date(avail_sd) <= date(avail_ed)),
        PRIMARY KEY (ct_userid, avail_sd, avail_ed)
);

CREATE TABLE FT_Leave (
        ct_userid       VARCHAR NOT NULL REFERENCES Caretaker (ct_userid),
        leave_sd        DATE NOT NULL,
        leave_ed        DATE NOT NULL,
        CHECK (date(leave_sd) <= date(leave_ed)),
        PRIMARY KEY (ct_userid, leave_sd, leave_ed)
);


-- LOOKING AFTER --

CREATE TABLE Looking_After (
        po_userid       VARCHAR NOT NULL,
        ct_userid       VARCHAR NOT NULL REFERENCES Caretaker (ct_userid) ON UPDATE CASCADE,
        pet_name        VARCHAR NOT NULL,
        dead            INTEGER DEFAULT 0,
        start_date      DATE NOT NULL,
        end_date        DATE NOT NULL,
        status          VARCHAR NOT NULL DEFAULT 'Pending' CHECK(status = 'Completed' OR status = 'Accepted' OR status
        ↪  = 'Pending' OR status = 'Rejected'),
        trans_pr        FLOAT4 NOT NULL CHECK(trans_pr > 0),
        payment_op      VARCHAR NOT NULL CHECK(payment_op = 'Credit Card' OR payment_op = 'Cash'),
        rating          FLOAT8 DEFAULT NULL,
        review          VARCHAR DEFAULT NULL,
        CHECK (date(start_date) <= date(end_date)),
        PRIMARY KEY (po_userid, ct_userid, pet_name, dead, start_date, end_date),
        FOREIGN KEY (po_userid, pet_name, dead) REFERENCES Pet (po_userid, pet_name, dead) ON UPDATE CASCADE
);


-- CHAT --

CREATE TABLE Chat (
        po_userid       VARCHAR,
        ct_userid       VARCHAR,
        pet_name        VARCHAR,
        dead            INTEGER NOT NULL,
        start_date      DATE NOT NULL,
        end_date        DATE NOT NULL,
        time            TIMESTAMPTZ,
        sender          INTEGER CHECK (sender = 1 OR sender = 2 OR sender = 3),
        text            VARCHAR,
        FOREIGN KEY (po_userid, ct_userid, pet_name, dead, start_date, end_date) REFERENCES Looking_After (po_userid,
        ↪  ct_userid, pet_name, dead, start_date, end_date) ON UPDATE CASCADE,
        PRIMARY KEY (po_userid, ct_userid, pet_name, dead, start_date, end_date, time, sender)
);
```

Our group also has some constraints that are not enforced by the relational schema which are as follows:

- We used a trigger to check that the price set by Part-Time Caretakers are at least as high as the price set by Admin for Full-Time Caretakers. This is to ensure that the Pet Owners would not solely only engage Part-Timers because they can set lower prices.
- We also used triggers to ensure that the (150 days * 2) requirement needed for Full-Timers will be able to be fulfilled or has already been fulfilled before the leave is approved.
- In the `PT_Availability` table, we need to ensure that the the Caretaker is a Part-Timer and in `FT_Leave` table, we need to ensure that the Caretaker is a Full-Timer.
- In the `Looking_After` table, we use triggers to ensure that once a booking has been confirmed, all other pending bookings under the same Caretaker which overlap with the accepted booking will be rejected.

6 Database Normal Forms

The schema R is known to have a Boyce-Codd Normal Form (BCNF) with respect to its functional dependency F if $\forall a \rightarrow A \in F^+$, one of the following properties is satisfied:

- $A \in a$ (where $a \rightarrow A$ is trivial)
- $a \in \mathbb{S}_R(F)$ (where a is a superkey)

In the process of designing our schema, all our tables have primary key(s) that uniquely identifies the rest of the rows in the schema. Since the primary keys are chosen minimal superkeys, all functional dependencies within our schema will satisfy the second condition where a is a superkey. Using the method of decomposition into normal form that produces BCNF fragments, we can conclude our database is in BCNF.

7 Interesting Triggers

7.1 Trigger 1: `trigger_price_check`

When the PCS Admin updates a new fixed price for Full-Timers for the various types of pets, the minimum price that a Part-Timer can charge will increase if the previous price would fall below this base price.

```
CREATE OR REPLACE FUNCTION trigger_price_check()
RETURNS TRIGGER AS
$$
DECLARE
        baseprice FLOAT4;
BEGIN
        -- Stores the new prices for each pet type that was updated
        baseprice = (SELECT price FROM Pet_Type WHERE pet_type = NEW.pet_type);

        -- Increase prices set by Part-time Caretakers if they would fall below this new price
        UPDATE PT_validpet
        SET price = baseprice
        WHERE (PT_validpet.ct_userid, PT_validpet.pet_type) IN (
                SELECT pt.ct_userid,pt.pet_type
                FROM PT_validpet pt
                INNER JOIN Pet_Type base
                ON pt.pet_type = base.pet_type
                WHERE pt.price < base.price
        );

        RETURN NULL;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS admin_changed_price ON Pet_Type;

CREATE TRIGGER admin_changed_price AFTER UPDATE ON Pet_Type
FOR EACH ROW EXECUTE PROCEDURE trigger_price_check();
```

7.2 Trigger 2: `trigger_ft_leave_check`

When the Full-Timers apply for leave, the requirement of (150 days * 2) must still be fulfilled, otherwise the leave application will be rejected. We also reject the leave application if it overlaps with an existing leave application for the same Full-Timer in the `FT_Leave` table.

```
CREATE OR REPLACE FUNCTION trigger_ft_leave_check()
RETURNS TRIGGER AS
$$
BEGIN
        DROP TABLE IF EXISTS leave_records;
        DROP TABLE IF EXISTS days_avail;

        -- Has all leave records for the user being inserted
        CREATE TEMPORARY TABLE leave_records AS (
                SELECT ftl.leave_sd, ftl.leave_ed
                FROM FT_Leave ftl
                WHERE NEW.ct_userid = ftl.ct_userid
        );

        -- Disallow leave application if caretaker would already be on leave
        IF (SELECT EXISTS(SELECT 1 FROM leave_records lr2
                WHERE NEW.leave_sd BETWEEN SYMMETRIC lr2.leave_ed AND lr2.leave_sd))
        OR (SELECT EXISTS(SELECT 1 FROM leave_records lr2
                WHERE NEW.leave_ed BETWEEN SYMMETRIC lr2.leave_ed AND lr2.leave_sd)) THEN
                RAISE EXCEPTION 'You are already on leave';
        END IF;

        -- Adding the newly-applied-for leave into leave_records
        INSERT INTO leave_records VALUES
        (NEW.leave_sd, NEW.leave_ed),
        (CAST(CONCAT(CAST(EXTRACT(YEAR FROM CURRENT_DATE) AS VARCHAR),'-01-01') AS DATE),
                CAST(CONCAT(CAST(EXTRACT(YEAR FROM CURRENT_DATE) AS VARCHAR),'-01-01') AS DATE)),
        (CAST(CONCAT(CAST(EXTRACT(YEAR FROM CURRENT_DATE) AS VARCHAR),'-12-31') AS DATE),
                CAST(CONCAT(CAST(EXTRACT(YEAR FROM CURRENT_DATE) AS VARCHAR),'-12-31') AS DATE));

        -- Calculates days between consecutive leaves, including number of days since start of year/to end of year,
        -- and the leave about to be inserted
        CREATE TEMPORARY TABLE days_avail AS
        SELECT LEAD(lr.leave_sd,1) OVER (ORDER BY leave_sd ASC) - lr.leave_ed AS diff
        FROM leave_records lr
        WHERE EXTRACT(YEAR FROM CURRENT_DATE) = EXTRACT(YEAR FROM lr.leave_sd)
        ORDER BY lr.leave_sd ASC;

        -- If the inserted leave results in the constraint of 2x150 days working being unfulfilled, raise exception,
        -- which interrupts the insert
        IF NOT (
                ((SELECT COUNT(*) FROM days_avail WHERE diff >= 150) = 2) OR
                ((SELECT COUNT(*) FROM days_avail WHERE diff >= 300) = 1)
        ) THEN
                RAISE EXCEPTION 'You must work 2x150 days a year';
        END IF;

        RETURN NEW;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS enforce_ft_avail ON FT_Leave;

CREATE TRIGGER enforce_ft_avail BEFORE INSERT ON FT_Leave
FOR EACH ROW EXECUTE PROCEDURE trigger_ft_leave_check();
```

7.3 Trigger 3: `trigger_pending_check`

When the status of a booking has been updated or inserted as 'Accepted', all other pending bookings for the pet in this booking will be cancelled if they overlap dates with the 'Accepted' booking.

```
CREATE OR REPLACE FUNCTION trigger_pending_check()
RETURNS TRIGGER AS
$$ BEGIN
-- Reject all other pending bids in an overlapping period, if an 'Accepted' status is updated/inserted for a specific
↪  pet
  IF NEW.status = 'Accepted' THEN
    UPDATE Looking_After la
    SET status = 'Rejected'
    WHERE la.po_userid = NEW.po_userid AND la.pet_name = NEW.pet_name AND la.status = 'Pending'
    AND NOT (la.start_date < NEW.start_date AND la.end_date < NEW.start_date)
    AND NOT (la.start_date > NEW.end_date AND la.end_date > NEW.end_date);
  END IF;
  RETURN NULL;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS cancel_pending_bids ON Looking_After;

CREATE TRIGGER cancel_pending_bids AFTER UPDATE OR INSERT ON Looking_After
FOR EACH ROW EXECUTE PROCEDURE trigger_pending_check();
```

8 Complex Queries

8.1 Query 1: bid_search

```
CREATE OR REPLACE FUNCTION bid_search (petname VARCHAR, sd DATE, ed DATE)
RETURNS TABLE (userid VARCHAR) AS
$func$
BEGIN
        RETURN QUERY(
                (
                        (
                                -- PTCT who can care for this pettype
                                SELECT pt.ct_userid FROM PT_validpet pt WHERE pt.pet_type IN
                                        (SELECT p.pet_type FROM Pet p WHERE p.pet_name = bid_search.petname AND
                                        ↪  p.dead = 0)
                        ) INTERSECT (
                                -- Available PTCT
                                SELECT PT_Availability.ct_userid FROM PT_Availability
                                WHERE bid_search.sd >= PT_Availability.avail_sd AND bid_search.ed <=
                                ↪  PT_Availability.avail_ed
                        ) EXCEPT (
                                -- REMOVE from available PTCT those who are fully booked
                                SELECT exp.ctuser FROM explode_date(sd, ed) exp
                                GROUP BY exp.ctuser, exp.day
                                HAVING COUNT(*) >=
                                CASE WHEN (
                                        SELECT avg(la.rating)
                                        FROM Looking_After la
                                        WHERE la.ct_userid = exp.ctuser AND (rating = 'Accepted' OR rating =
                                        ↪  'Pending')
                                ) > 4 THEN 5
                                ELSE 2
                                END
                        )
                )
                UNION
                (
                        -- FTCT who can care for this pettype
                        SELECT ft.ct_userid FROM FT_validpet ft WHERE ft.pet_type IN
                                (SELECT p.pet_type FROM Pet p WHERE p.pet_name = bid_search.petname AND p.dead = 0)
                        EXCEPT (
                                -- Remove FT who are unavailable
                                SELECT ftl.ct_userid FROM FT_Leave ftl
                WHERE NOT (
                        (bid_search.sd < ftl.leave_sd AND bid_search.ed < ftl.leave_sd) OR
                        (bid_search.sd > ftl.leave_ed AND bid_search.sd > ftl.leave_ed)
                )
                        ) EXCEPT (
                                --Remove FT caretakers who have 5 pets at any day in this date range
                                SELECT exp2.ctuser FROM explode_date(sd, ed) exp2
                                GROUP BY exp2.ctuser, exp2.day
                                HAVING COUNT(*) >= 5
                        )
                )
        );
END;
$func$
LANGUAGE plpgsql;
```

The `bid_search` function searches for the Caretaker IDs that fits the criterion set by the Pet Owner based on start date and end date for their particular pet.

Explanation: We first select Part-Time Caretakers who can care for that pet type and are available from start date to end date. We also ensure they currently either have less than 2 pending/accepted transactions, or less than 5 pending/accepted transactions but with an average rating of more than 4. We do the same for Full-Time caretakers, less the rating caveat, then combine them to display all available Caretakers for the conditions given.

8.2 Query 2: `total_pet_day_mnth`

```
CREATE OR REPLACE FUNCTION total_pet_day_mnth(userid VARCHAR, year INT, month INT)
RETURNS INT AS
$func$
DECLARE
        firstday DATE := CAST(CONCAT(CAST(total_pet_day_mnth.year AS VARCHAR),
                '-', CAST(total_pet_day_mnth.month   AS VARCHAR),'-01') AS DATE);
        lastday DATE  := CAST(CONCAT(CAST(total_pet_day_mnth.year AS VARCHAR),
                '-', CAST(total_pet_day_mnth.month+1 AS VARCHAR),'-01') AS DATE);
BEGIN
        RETURN (
                SELECT GREATEST(
                        (
                                -- Transaction occurs completely in this month
                                SELECT SUM(CAST(EXTRACT(DAY FROM la.end_date) AS INT)
                                        - CAST(EXTRACT(DAY FROM la.start_date) AS INT) + 1)
                                FROM Looking_After la
                                WHERE total_pet_day_mnth.userid = la.ct_userid
                                AND la.start_date >= firstday AND la.end_date < lastday
                                AND la.status = 'Completed'
                                GROUP BY la.ct_userid
                        ), 0)
                + GREATEST(
                        (
                                -- Transaction starts before this month, but ends during
                                SELECT SUM(CAST(EXTRACT(DAY FROM lab.end_date) AS INT)
                                        - CAST(EXTRACT(DAY FROM firstday) AS INT) + 1)
                                FROM Looking_After lab
                                WHERE total_pet_day_mnth.userid = lab.ct_userid
                                AND lab.start_date < firstday AND lab.end_date < lastday AND lab.end_date >= firstday
                                AND lab.status = 'Completed'
                                GROUP BY lab.ct_userid
                        ), 0)
                - GREATEST(
                        (
                                -- Transaction starts during this month, but ends after
                                SELECT SUM(CAST(EXTRACT(DAY FROM lastday) AS INT)
                                        - CAST(EXTRACT(DAY FROM lac.start_date) AS INT) - 1)
                                FROM Looking_After lac
                                WHERE total_pet_day_mnth.userid = lac.ct_userid
                                AND lac.start_date < lastday AND lac.start_date >= firstday AND lac.end_date >
                                ↪  lastday
                                AND lac.status = 'Completed'
                                GROUP BY lac.ct_userid
                        ), -99999)
        );
END;
$func$
LANGUAGE plpgsql;
```

The `total_pet_day_mnth` function finds the number of pet-days for the particular Caretaker during a specified month and year. This is necessary in order to calculate salary for full-time caretakers, who receive $3000/month for up to 60 pet-days, and 80% of their price as bonus for excess pet-days.

Explanation: We split our problem into 3 parts that we sum to obtain the return value. For each part, we only look at Completed transactions in `Looking_After`. First, we consider Completed transactions which occur completely within the month being queried for. Second, we consider Completed transactions which start before, but end during, the month being queried for. Last, we consider Completed transactions which start during, but end after, the month being queried for. Summing the pet-days calculated in each of these three situations will return the total number of pet days in a given month, regardless of differing date periods of each Completed entry.

8.3 Query 3: `total_trans_pr_mnth`

```
CREATE OR REPLACE FUNCTION total_trans_pr_mnth(userid VARCHAR, year INT, month INT)
RETURNS FLOAT4 AS
$func$
DECLARE
        firstday DATE;
        lastday DATE;
BEGIN
        firstday = cast(concat(cast(year AS VARCHAR), '-', cast(month AS VARCHAR),'-01') AS date);
        lastday = cast(concat(cast(year AS VARCHAR), '-', cast((month+1) AS VARCHAR), '-01') AS date);

        RETURN 0 +
                -- Transaction occurs completely in this month
                COALESCE(
                (
                        SELECT sum(la.trans_pr)
                        FROM Looking_After la
                        WHERE userid = la.ct_userid
                        AND la.start_date >= firstday AND la.end_date <= lastday
                        AND la.status = 'Completed'
                ), 0)
                -- Transaction starts before this month, but ends during
                + COALESCE(
                (
                        -- Multiplies trans_pr by no. of days that transaction was in this month
                        SELECT sum(lab.trans_pr * (lab.end_date - firstday)/(lab.end_date - lab.start_date))
                        FROM Looking_After lab
                        WHERE userid = lab.ct_userid
                        AND lab.start_date < firstday AND lab.end_date < lastday AND lab.end_date >= firstday
                        AND lab.status = 'Completed'
                ), 0)
                -- Transaction starts during this month, but ends after
                + COALESCE(
                (
                        -- Multiplies trans_pr by no. of days that transaction was in this month
                        SELECT sum(lac.trans_pr * (lastday - lac.start_date)/(lac.end_date - lac.start_date))
                        FROM Looking_After lac
                        WHERE userid = lac.ct_userid
                        AND lac.start_date <= lastday AND lac.start_date >= firstday AND lac.end_date > lastday
                        AND lac.status = 'Completed'
                ), 0);
END;
$func$
LANGUAGE plpgsql;
```

The `total_trans_pr_mnth` function takes in a user-ID of a Caretaker, the month and the year we are interested in and outputs the total price of the transactions handled by the Caretaker in that month.

Explanation: Similarly to the previous query, the calculation is also split into three parts for transactions that occur completely within the month, transactions that started before the month and transactions that end after the month. Some interpolation for obtaining the transaction price for the latter two cases were needed as the spillover of the transaction price into the following months had to be adjusted for.

9 Application Interface

Below consists of several screenshots of our application.

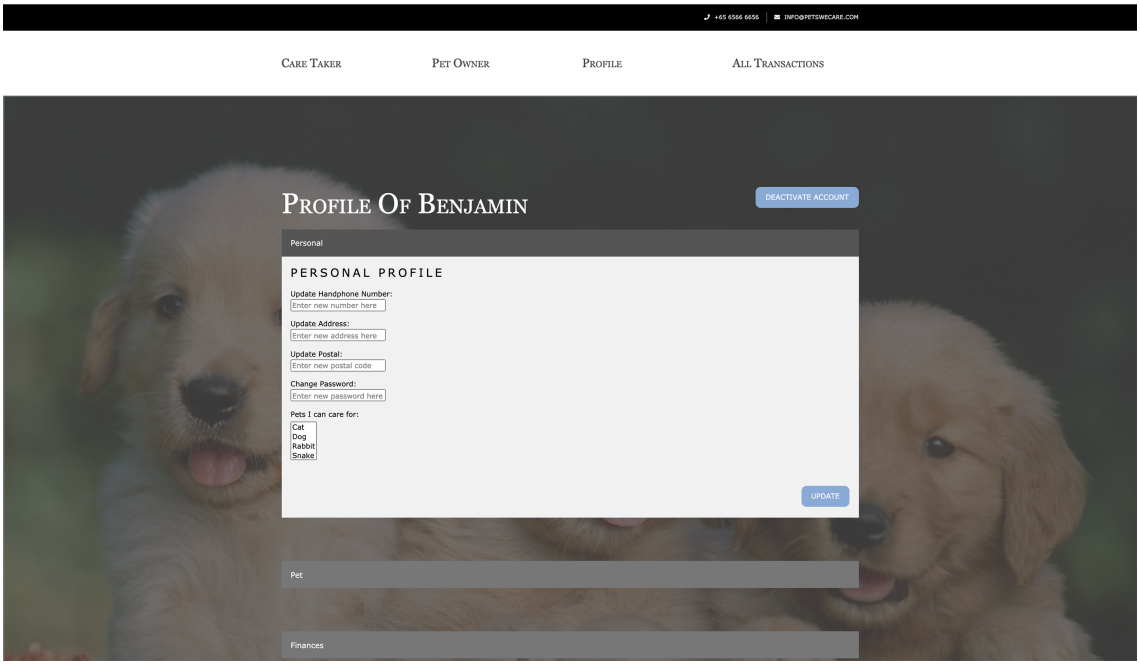

Figure 2: Screenshot of Application Profile Page

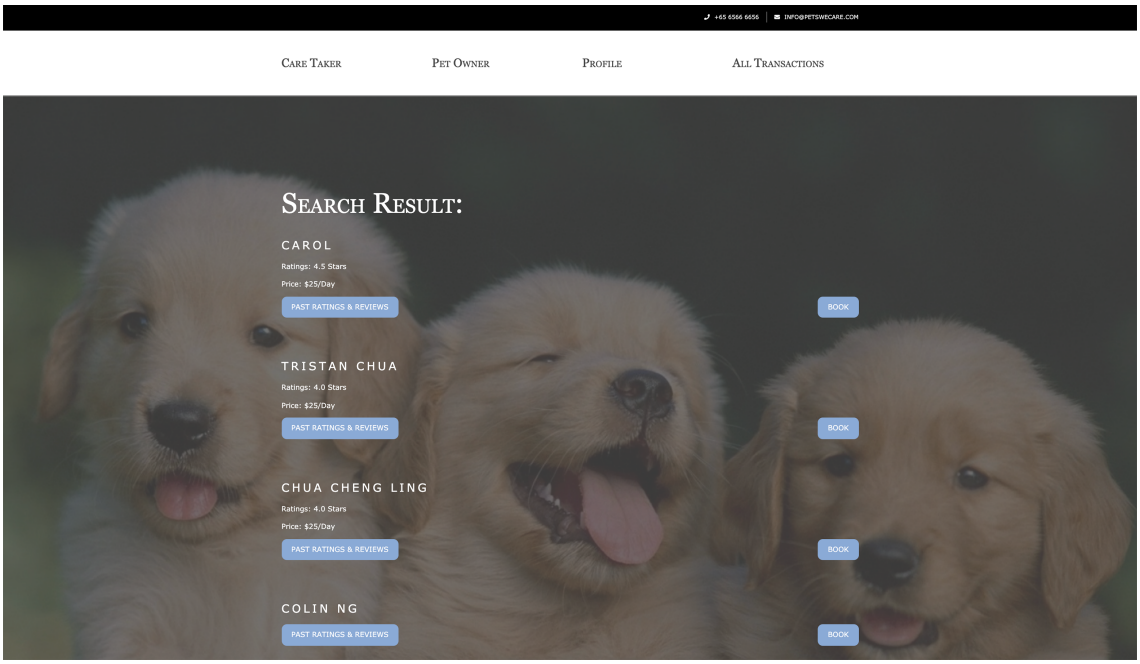

Figure 3: Screenshot of Application Search Result

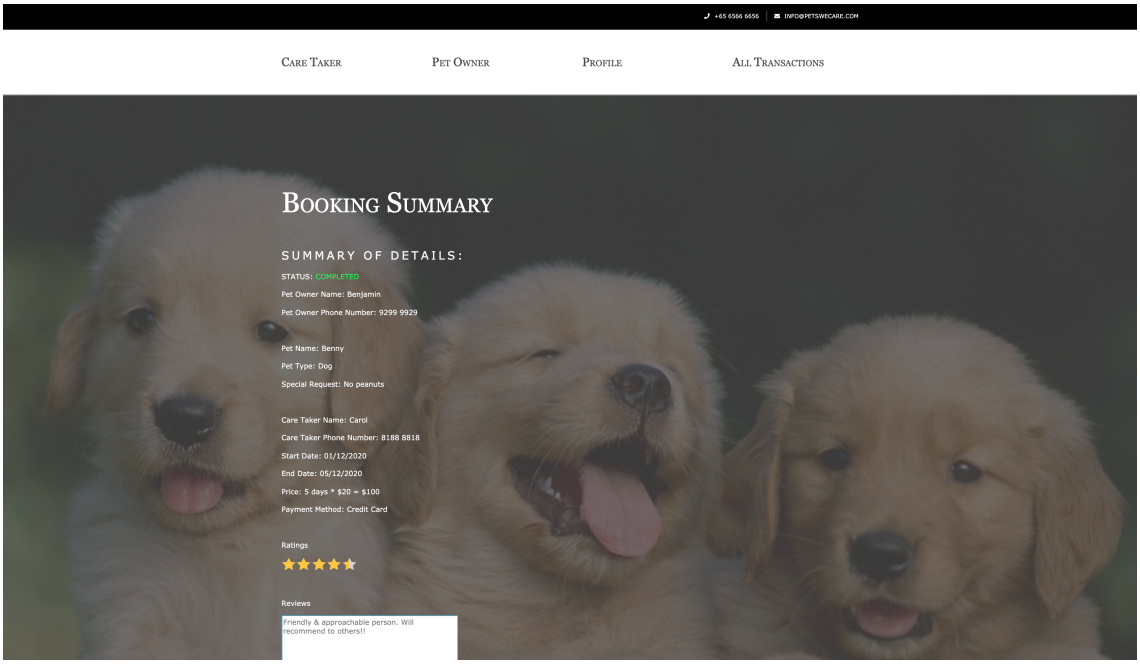

Figure 4: Screenshot of Application Confirmation Screen

10 Summary of Project

- Difficulties Faced
  - Had to experience a steep learning curve when creating the web application since no one had prior of web-design knowledge of FlaskApp, HTML and CSS.
  - Linking between front-end and back-end
  - Not being able to use serial types was inconvenient
- Lessons Learnt
  - We have learnt how databases can aid an application by serving data to its back-end
  - Importance of Database Normal Forms to ensure good structure and integrity of the way we store our data