CS2102 Semester 1 AY20/21 Team 13

Select * from Names;

Tan Zheng Wen	A0183615A
Zhuang Yuan	A0166849J
Tay Kang Ming	A0164811J
Shannon Lee	A0184311N
Rusdi Haizim B Rahim	A0183139B

Contents

1. Project / Teammates Responsibilities **3**

2. Data requirements, functionalities and constraints covered **4**
   - 2.1 ER model 4
   - 2.2 Justifications for any non-trivial design decisions in your ER model 4
   - 2.3 Application's constraints captured 5

3. Relational schema derived from ER data model **9**

4. Discussion / Justification of database in 3NF or BCNF **12**

5. Details of three non-trivial/interesting triggers used **13**

8. SQL code of three of the most complex queries implemented in your application. Provide an English description of each of these queries. **13**

9. Specification of the software tools/frameworks used in your project. **16**

10. Two or three representative screenshots of your application in action. **17**

11. A summary of any difficulties encountered and lessons learned from the project. **18**

* represents additional constraints/features we added to the application

1. Project / Teammates Responsibilities

Zheng Wen - API used by caretakers excluding calls made to view their bids, past transactions and reviews. Frontend Calendar console for caretakers.

Rusdi - API used by pet owners to search for caretakers based on startDate, endDate, petCategory, and preferred Caretaker. Also handled API for selecting which pets are able to be inserted into the bids and whether base_payment amount for each caretaker is met.

Kang Ming - API to support the creation/deletion/update of data for the different users (Pet Owner, CareTaker, and PCS Administrator), triggers to enforce ISA constraints, set up frontend, backend and deployment

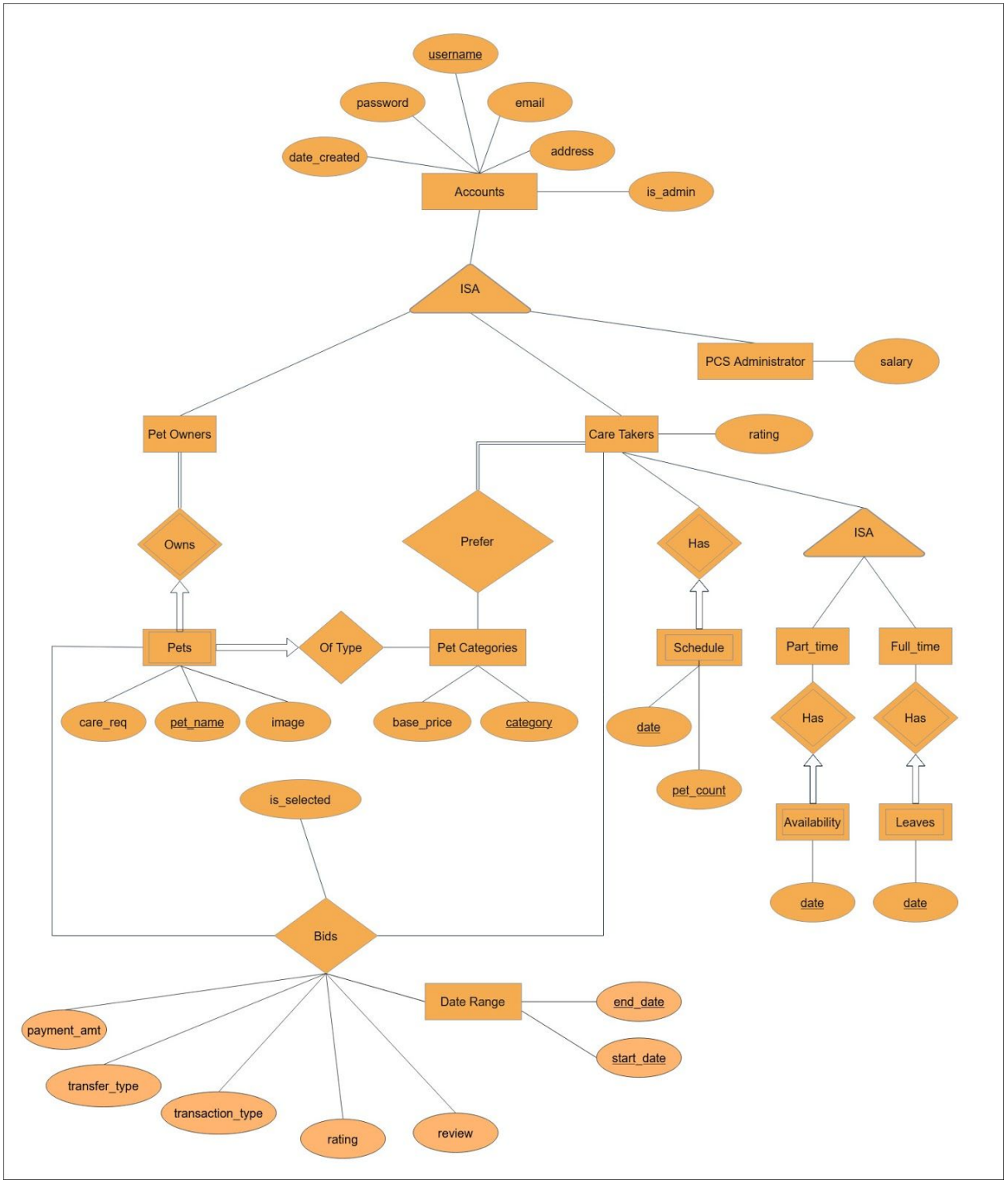
Zhuang Yuan - API used by caretakers to call to view their bids, past jobs and reviews and for pet owners to view their bid status

Shannon - API used by PCS admin to view the total pet-days in a given month, the months with the most pet days, total revenue brought in and profit, underperforming caretakers, and the salary for each caretaker

* represents additional constraints/features we added to the application

2. Data requirements, functionalities and constraints covered

2.1 ER model



2.2 Justifications for any non-trivial design decisions in your ER model

1. The Schedule entity refers to the bids that the Caretaker has selected. We decided not to store all the working dates of a Caretaker. Instead, we made use of Availability to keep track of the dates that a Part-Timer is available and Leaves to keep track of the dates that a Full-Timer is not available.

* represents additional constraints/features we added to the application

2. Bids is a relationship set between Pets and Caretakers. Since Pets depend on Pet Owners, a pet is identified by the username and pet_name. The Bids relationship allows the following:

i) A Pet Owner can decide to make a bid for one of his / her pets to be taken care of by a Caretaker over a specific date range. The bid's is_selected attribute will be initially set to false.

ii) A Caretaker can select a bid, which means that he / she decided to take care of a Pet of a Pet Owner. The bid's is_selected attribute will be updated and set to true to signify that the Caretaker has accepted the bid made.

2.3 Application's constraints captured

Constraints covered by ER Diagram and Schema:

Constraint	**Description**
The application supports at least three kinds of users: Pet Owner, Care Taker and PCS Administrator	The ISA relationship between Accounts, Care_taker and Pet_owner does not indicate a covering or non-overlapping constraint. (covered by using triggers)
Each Pet are classified into categories to facilitate browsing	Yes, pets have an attribute called 'pet_category' that cannot be null
They (Pet) may also have special requirements related to how they need to be taken care of	Yes, pets have an attribute called 'care_req' that can be null
(Care Taker) may take care of more than one pets at any given time	In the Schedule entity, the pet_count attribute states the number of pets taken care of at any given date. There is no restriction on this
Both the Pet Owner and Care Taker should agree on how to transfer the Pet, which can be one of the following three (or more, if you have other possible ideas): 1. Pet Owner deliver 2. Care Taker pick up 3. Transfer through the physical building of PCS	Under the bids relation, the attribute transfer_type specifies how the Pet Owner and Care Taker agreed to transfer the pet.
The Pet Owner must pay for the amount upfront either by pre-registered credit card or paying cash	In the Bids relation, the transaction_type specifies how the payment is made
At the end of the care period, the Pet Owner can post review on and give rating to	Because the Pets entity is a weak entity of Pet_Owners, pet owners are in the Bids

* represents additional constraints/features we added to the application

the Care Taker.	relation and can write a review for a caretaker
The review and rating will only be for specific transaction. Pet Owner may submit multiple review/rating for a Care Taker if the Care Taker has taken care of the Pet Owner's Pet multiple times, including for the same pet.	Review and rating are attributes in the Bids relation, thus there can only be one review and rating per bid (transactions are bids where is_selected = true)
Full-time Care Taker are treated as available until they apply for leave	Full-time Care Taker has a relationship to the Leave entity, dates not in the Leave entity are considered available
Full-time Care Taker have a limit of up to 5 pet at any one time.	When a bid is selected by a care taker, there is a constraint check before the bid is inserted to ensure that the full time care taker does not already have to take care of 5 pets on that day.

Constraints covered by Triggers:

Constraint	**Description**
Pet Owner and Care taker may use the same account and when they are mentioned as one, we will use the term User. * An account can either be a non PCS admin or a PCS admin. Where a non PCS admin can either be a pet owner, care taker or both. Covering and non-overlapping.	When an account is created, if the is_admin attribute is false, the account will be automatically set to be a Pet Owner account. Else, the account is a PCS Administrator account. We do not allow a user to only be a care taker but not a pet owner. The user can decide to be a Caretaker based on options specified on the application's frontend.
* A care taker can either be a full timer or part timer. Covering and overlapping.	If the user decides to be a caretaker, he / she will have the option to decide whether to be a full-timer or part-timer. A trigger will insert into the Caretaker table before a full-timer / part-timer is inserted into the Full-timer / Part-timer table.
* The information about a care taker's rating should always be up to date with any new reviews and rating he/she gets.	A trigger will update a care taker's rating everytime that care taker has a corresponding rating in the Bids table being updated by a pet owner. (given that the bid is_selected and has passed)
* For every bid that is selected, there should	A trigger will update the Schedules table

* represents additional constraints/features we added to the application

be a corresponding entry in schedules for the care taker being engaged. This entry in schedule captures each date for each care taker and the number of pets in their care on that day if there is at least 1 pet under their care	when the attribute is_selected in Bids is updated to true.

Constraints covered by Application Code:

Constraint	**Description**
For each part-time Care Taker, they should be able to specify their availability for the current year and the next year. For instance on 1 January 2020, a part-time Care Taker can already specify their availability until 31 December 2021.	The frontend uses an API endpoint to insert entries in the availability table is executed by a function which takes in the Care Taker's username and a list of dates of which the care taker specified his/her availability on.
Pet Owner can browse or search for Care Taker and bid for their services	The frontend uses an API endpoint which queries a suitable list of care takers and their daily price for pet owners based on a specified start date, end date and the pet in the bid.
The total cost of caring for a Pet is the number of days multiplied by the daily price stated by the Care Taker.	There is a check within the 'Insert' query that ensures the payment amount is at least the minimum total cost for caring for the pet whenever a bid is about to be inserted into the bids table.
The reviews will be available to all User	The rating of each Care Taker will be computed based on the average rating of all the reviews for that care taker using a trigger which triggers upon update to the attribute rating in the bids table. When finding a suitable care taker to bid for, a user can access his/her reviews.
Each full-time Care Taker must work for a minimum of 2x150 consecutive days a year. Care Taker cannot apply for leave if there is at least one Pet under their care	Implemented using a **SQL procedure** when full timer specifies their leave days for the year. The function will check for these 2 conditions before inserting leaves into the leaves table. For part timers, there will be a check when they delete their availability to ensure it is not on a pet day. For the case where care takers exceed 60 pet days, the extra salary they receive is taken as a ratio of the total payment they receive for the month * (total pet days - 60)

* represents additional constraints/features we added to the application

	/ (total pet days). This is because we feel that using the first 60 pet days may be unfair if the payment of this first 60 pet days differ significantly for the other pet days in the month.
At any single point in time, a part-time Care Taker cannot take care for more than 2 Pet unless they have a good rating (e.g., 4 out of 5, or any such threshold you define) and they cannot have more than 5 Pet regardless of rating.	This check constraint is enforced in an **SQL procedure** which updates is_selected attribute in the bids table to true.
This price increases with the rating of the Care Taker but will never be below the base price.	Implemented by a function where if a bid price is below the corresponding price of a care taker (based on rating and base price), an exception will be raised. If not, the bid will be inserted into the bids relation
Full time Care Takers will receive a base salary of $3000 If a full time Care Taker has more than 60 pet-days, he receives a bonus of 80% of their price Part time Care Takers will receive 75% of their price as their salary	Implemented by the **expected_salary** function where the base salary for full-timers is guaranteed, and the bonus is added on top of the care_taker's base salary. **See query 1 in section 8.**
Pet Owners can browse summary information on all Care Takers and other Pet Owners in their area	Implemented in SQL query to display a list of caretakers using sql string matching 'LIKE' for addresses and care taker names.
Pet Owners can browse information of their own pets	Implemented in SQL to query all relevant information about their own pets. They can also update and change the pictures of their pets
A Care Taker should not take care of pets they cannot care for	Care takers can specify which pets they can care for whenever they like. The new view of eligible care takers for pet owners as well as the new bids a care taker can accept will change based on that. Bids that have already been accepted for a pet that the care taker could previously care for will not change.
PCS Admin can view the number of pets taken care of in the month	Implemented with a query to find distinct <pet_name, pet_owner> in the Bids table
PCS Admin can view the total salary to be paid to all Care Takers for the month	Implemented with a query similar to expected_salary that gets the salary for all Care Takers **See query 1 in section 8.**

* represents additional constraints/features we added to the application

PCS Admin can view the month with the most jobs	Implemented with a query from the Schedule table to sum the pet_count for each month
PCS Admin can view summary of Care Takers	Implemented with a query from Care_Takers and Accounts table

Not enforced:

Constraint	**Description**
When bid by any Pet Owner, a full-time Care Taker will always accept the job immediately if possible	No constrained (Interpretation) If a bid by pet owner goes through, there is no need for bids as their request will always be fulfilled by the first selection
The base daily price for a full-time Care Taker is already specified by PCS Administrator for each pet type.	No relation between PCS admin and Pet_category entity (which has a base_price attribute)
Successful bidder could either be chosen by the Care Taker or automatically selected by the system based on some criteria.	The functionality for a Care Taker to select a bid is implemented. However, we did not implement an automatic system to make care takers accept bids. Care Takers can filter bids according to their priorities such as payment amount, area of care taker among other things and select the bid of their choice.
* Since we did not implement an automatic system, it is possible that a full time care taker can choose to not accept any bids and effectively have unlimited leaves if planned early.	

3. Relational schema derived from ER data model

```
CREATE TABLE IF NOT EXISTS Accounts (
  username VARCHAR(256),
  password VARCHAR(256) NOT NULL,
  email VARCHAR(256) NOT NULL ,
  address VARCHAR(256) NOT NULL,
  date_created DATE NOT NULL,
  is_admin BOOLEAN DEFAULT false,
```

* represents additional constraints/features we added to the application

```
    PRIMARY KEY (username),
    CONSTRAINT proper_email CHECK (email ~* '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$')
);

-- Implement covering & overlapping constraint

CREATE TABLE IF NOT EXISTS Pet_Owners (
    username VARCHAR(256),
    FOREIGN KEY (username) REFERENCES Accounts(username) ON DELETE CASCADE,
    PRIMARY KEY (username)
);

CREATE TABLE IF NOT EXISTS Care_Takers (
    cname VARCHAR(256),
    rating NUMERIC CHECK (rating <= 5 AND rating > 0),
    FOREIGN KEY (cname) REFERENCES Accounts(username) ON DELETE CASCADE,
    PRIMARY KEY (cname)
);

CREATE TABLE IF NOT EXISTS Part_Timer (
    cname VARCHAR(256) REFERENCES Care_Takers(cname) ON DELETE CASCADE,
    PRIMARY KEY (cname)
);

CREATE TABLE IF NOT EXISTS Availability (
    cname VARCHAR(256) REFERENCES Part_Timer(cname) ON DELETE CASCADE,
    date DATE,
    PRIMARY KEY(cname, date)
);

CREATE TABLE IF NOT EXISTS Full_Timer (
    cname VARCHAR(256) REFERENCES Care_Takers(cname) ON DELETE CASCADE,
    PRIMARY KEY (cname)
);


CREATE TABLE IF NOT EXISTS Leaves (
    cname VARCHAR(256) REFERENCES Full_Timer(cname) ON DELETE CASCADE,
    date DATE,
    PRIMARY KEY(cname, date)
);

CREATE TABLE IF NOT EXISTS Pet_Categories (
    category VARCHAR(256),
    base_price NUMERIC,
    PRIMARY KEY (category)
);

CREATE TABLE IF NOT EXISTS Prefers (
    cname VARCHAR(256) REFERENCES Care_Takers(cname) ON DELETE CASCADE,
    category VARCHAR(256) REFERENCES Pet_Categories(category) ON DELETE CASCADE,
```

* represents additional constraints/features we added to the application

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

CREATE TABLE IF NOT EXISTS Schedule (
    cname VARCHAR(256) REFERENCES Care_Takers (cname) ON DELETE CASCADE,
    date DATE,
    pet_count int,
    PRIMARY KEY (cname, date)
);

CREATE TABLE IF NOT EXISTS PCS_Administrator (
    username VARCHAR(256),
    salary numeric,
    FOREIGN KEY (username) REFERENCES Accounts(username) ON DELETE CASCADE,
    PRIMARY KEY (username)
);

CREATE TABLE IF NOT EXISTS Pets (
    pet_name VARCHAR(256),
    category VARCHAR(256) NOT NULL REFERENCES Pet_Categories(category) ON DELETE CASCADE,
    pname VARCHAR(256) NOT NULL REFERENCES Pet_Owners(username) ON DELETE CASCADE,
    care_req TEXT,
    PRIMARY KEY (pname, pet_name)
);

CREATE TABLE IF NOT EXISTS Bids (
    pname VARCHAR(256),
    pet_name VARCHAR(256),
    cname VARCHAR(256),
    start_date DATE,
    end_date DATE,
    rating NUMERIC CHECK (rating <= 5 AND rating > 0),
    is_selected BOOLEAN,
    payment_amt NUMERIC,
    transaction_type VARCHAR(30),
    review VARCHAR(256),
    PRIMARY KEY(pname, pet_name, start_date, end_date),
    FOREIGN KEY (pname, pet_name) REFERENCES Pets(pname, pet_name) ON DELETE CASCADE,
    FOREIGN KEY (cname) REFERENCES Care_Takers(cname) ON DELETE CASCADE
);
```

The act of inserting into bids possesses a constraint that is enforced through

- Limiting the date range of the desired caretaker based on past search on the catalogue.
- Limiting the pets that can be inserted into the bid; only allowing pet owners to select from a list of valid pets for that date period (that doesn't conflict with those pets that already exist in the bids table within that date period).
- Only bids that meet a certain payment amount (based off base_price of pet category and multiplier due to caretaker rating) can be inserted.

* represents additional constraints/features we added to the application

We did not enforce cascade delete in the bids relation as it serves as a record of all tasks. So there may be dangling foreign keys should any entity related to it if they are deleted. This would lead to an edge case where we are unable to delete existing valid bids in the table i.e. a pet owner who just bidded but his/her pet got ill

We assume that when a caretaker selects a pet category, the caretaker is able to take care of that pet category

4. Discussion / Justification of database in 3NF or BCNF

Our database is in BCNF as all our tables created are in BCNF.
Lemma 2: Any schema with exactly 2 attributes is in BCNF
Conditions for BCNF:

1. a → A is trivial
2. a is a superkey

Table	**3NF / BCNF**	**Explanation**
Accounts	BCNF	Username attribute is the only superkey, and it is on the LHS of each FD
Pet Owners	BCNF	Only 1 attribute, Username, which is trivial
Care Takers	BCNF	Only 2 attributes (Lemma 2)
Full Timer	BCNF	Only 2 attributes (Lemma 2)
Part Timer	BCNF	Only 2 attributes (Lemma 2)
Availability	BCNF	Only 2 attributes (Lemma 2) + Trivial
Leaves	BCNF	Only 2 attributes (Lemma 2) + Trivial
Pet Categories	BCNF	Only 2 attributes (Lemma 2)
Prefers	BCNF	Only 2 attributes (Lemma 2) + Trivial
Schedule	BCNF	(cname, pet_count) is a superkey and the only non-trivial FD
PCS_Administrator	BCNF	Only 2 attributes (Lemma 2)
Pets	BCNF	(pname, pet_name) is a superkey and it is on the left side of each FD
Bids	BCNF	(pname, pet_name, start_date, end_date) is a superkey and the only non-trivial FD

* represents additional constraints/features we added to the application

5. Details of three non-trivial/interesting triggers used

1. update_care_taker_rating: This trigger updates the average rating of a caretaker based on the individual rating the caretaker received for each pet he/she took care of. The trigger is invoked when there is an update to the attribute rating in the Bids Table.

2. add_caretaker : This trigger enforces the covering but not overlapping constraint in the ISA between the Caretakers and Full-timers and Part-timers. When a user decides to sign up to be a Caretaker, he / she needs to choose whether to become a Full-timer or Part-timer. Based on the selection, the trigger will create a row in the Caretakers table before a new row is created in the full-timer table or part-timer table.

3. create_account_trigger: This trigger enforces the covering but not overlapping constraint in the ISA between the Accounts and Pet_Owners and PCS Administrator. When a user signs up for an account, the user must either be a Pet_Owner or a PCS Administrator but cannot be both. Based on the selection, the trigger will create a row in the Pet_Owners table or PCS_Administrator table after a new row is created in the Accounts table.

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

1. Query to get the salary as well as the revenue generated for a caretaker for a given month.

```
SELECT
    CASE
        WHEN '${username}' IN (SELECT cname FROM part_timer) THEN SUM(payment_amt / (end_date -
start_date + 1)) * 0.75
        WHEN '${username}' IN (SELECT cname FROM full_timer) AND COUNT(*) <= 60 THEN 3000
        WHEN '${username}' IN (SELECT cname FROM full_timer) THEN 3000.0 + 1.0 * (COUNT(*) - 60)
/ COUNT(*) * SUM(payment_amt / (end_date - start_date + 1)) * 0.8
    END salary,
    SUM(payment_amt / (end_date - start_date + 1)) revenue
FROM Schedule NATURAL JOIN Bids
WHERE cname = '${username}' AND date <= end_date AND date >= start_date AND is_selected
GROUP BY to_char(date, 'MM-YYYY')
HAVING to_char(date, 'MM-YYYY') = '${month}';
```

2. Query for caretakers to select bids.

```
CREATE OR REPLACE FUNCTION specify_leaves(username VARCHAR(256), leaves_ date[]) RETURNS void AS
$$
```

* represents additional constraints/features we added to the application

```
    DECLARE
        contiguous_150_blocks INTEGER:= 0;
        previous_date DATE:= date_trunc('year', leaves_[1]);
        previous_leaves DATE[] := ARRAY(SELECT date FROM leaves WHERE username=cname AND
to_char(date, 'YYYY') = to_char(previous_date, 'YYYY'));
        x DATE;
    BEGIN
        --Check if any leaves applied are before the current_date
        IF (SELECT COUNT(*) FROM unnest(leaves_) dates WHERE dates <= current_date) > 0 THEN
            RAISE EXCEPTION'Date has already passed';
        END IF;

        --Merge leaves applied now with previous leaves for a year and add first jan next year
to the mix
        leaves_ = leaves_::date[] || previous_leaves::date[];

        --append next year's first day to date[]
        leaves_ = leaves_::date[] || (previous_date + interval '1 year')::date;

        --sort leaves_ and filter duplicates
        leaves_ = ARRAY(SELECT DISTINCT date FROM unnest(leaves_) date ORDER BY date
ASC)::DATE[];

        FOREACH x in ARRAY leaves_ -- O(~64)
            LOOP
                --Ensure leaves are from the same year
                IF to_char(x - 1, 'YYYY') <> to_char(previous_date, 'YYYY') AND x <>
date_trunc('year', previous_date) THEN
                    RAISE EXCEPTION'Attempted to apply across years';
                END IF;

                -- Checks if a pet is under the caretaker's care on that day
                IF (SELECT AVG(pet_count) FROM Schedule WHERE date = x AND cname = username) > 0
THEN
                    RAISE EXCEPTION'Cannot apply for leave if there is at least one pet';
                END IF;

                -- Checks for the 2x150 blocks
                IF x - previous_date >= 150 THEN
                    contiguous_150_blocks := contiguous_150_blocks + (x - previous_date) / 150;
                END IF;
                RAISE NOTICE 'Segment: %',x - previous_date;

                IF x != (date_trunc('year', leaves_[1]) + interval '1 year')::date THEN
                -- It is now save to apply for leave x
                    INSERT INTO leaves VALUES(username, x) ON CONFLICT(cname, date) DO NOTHING;
                END IF;

                -- update to be the next working day
                previous_date := x + 1;
            END LOOP;
```

* represents additional constraints/features we added to the application

```
        IF contiguous_150_blocks < 2 THEN
            RAISE EXCEPTION'Failed to meet the requirement of 2x150 pet days, %',
contiguous_150_blocks;
        END IF;
    END;
$$ LANGUAGE plpgsql;

SELECT specify_leaves('${username}'::VARCHAR(256), '${dates}'::date[])
    FROM full_timer
    WHERE cname = '${username}';
```

3. Query for Petowner to find valid caretakers for their pets.

```
WITH cte_valid_caretakers AS (
  SELECT cname
  FROM (
    SELECT DISTINCT F.cname, L.date
    FROM full_timer F, prefers P, accounts A, pets PET,
    (SELECT generate_series(TO_DATE('${startDate}', 'DD-MM-YYYY'), TO_DATE('${endDate}',
'DD-MM-YYYY'),'1 day'::interval) AS date) AS L
    WHERE F.cname = P.cname AND P.cname = A.username
    AND PET.pet_name = '${petName}' AND PET.pname = '${pName}'
    AND PET.category = P.category
    AND P.cname LIKE '${cName}'
    AND A.address LIKE '${address}' AND F.cname != '${pName}'
    AND NOT EXISTS (
      SELECT DISTINCT L1.date
      FROM leaves L1
      WHERE L.date = L1.date AND F.cname = L1.cname
      AND L1.date >= TO_DATE('${startDate}', 'DD-MM-YYYY') AND L1.date <= TO_DATE('${endDate}',
'DD-MM-YYYY')
    )
    AND NOT EXISTS (
      SELECT S.date
      FROM schedule S
      WHERE L.date = S.date AND F.cname = S.cname
      AND S.pet_count = 5
    )
  ) AS FT
  GROUP BY FT.cname
  HAVING TO_DATE('${endDate}', 'DD-MM-YYYY') - TO_DATE('${startDate}', 'DD-MM-YYYY')+1 =
COUNT(*)
  UNION
  SELECT cname
  FROM (
    SELECT DISTINCT A.cname, A.date
    FROM availability A, prefers P, accounts AC, pets PET
    WHERE A.date >= TO_DATE('${startDate}', 'DD-MM-YYYY') AND A.date <= TO_DATE('${endDate}',
'DD-MM-YYYY')
```

* represents additional constraints/features we added to the application

```
    AND P.cname = A.cname AND A.cname = AC.username
    AND PET.pet_name = '${petName}' AND PET.pname = '${pName}'
    AND PET.category = P.category
    AND P.cname LIKE '${cName}'
    AND AC.address LIKE '${address}' AND P.cname != '${pName}'
    AND NOT EXISTS (
      SELECT DISTINCT S.cname
      FROM schedule S, care_takers C
      WHERE A.cname = S.cname AND S.date = A.date
      AND S.cname = C.cname AND ((C.rating <= 2 AND S.pet_count = 2) OR (C.rating > 2 AND
S.pet_count = CEILING(C.rating)))
    )
  ) AS PT
  GROUP BY PT.cname
  HAVING TO_DATE('${endDate}', 'DD-MM-YYYY') - TO_DATE('${startDate}', 'DD-MM-YYYY')+1 =
COUNT(*)
)
SELECT CVC.cname,
  CASE
    WHEN CT.rating IS NULL
      THEN -1
    ELSE
      CT.rating
  END AS rating
, P.category,
  CASE
    WHEN CT.rating IS NOT NULL
      THEN ROUND(PC.base_price + (PC.base_price * (CEILING(CT.rating) - 1) / 4)::numeric, 2)
    ELSE
      ROUND(PC.base_price::numeric, 2)
  END AS minPrice,
  A.address
FROM cte_valid_caretakers CVC, care_takers CT, prefers P, pet_categories PC, accounts A
WHERE CVC.cname = CT.cname AND CT.cname = P.cname AND P.cname = A.username
AND P.category = PC.category
```

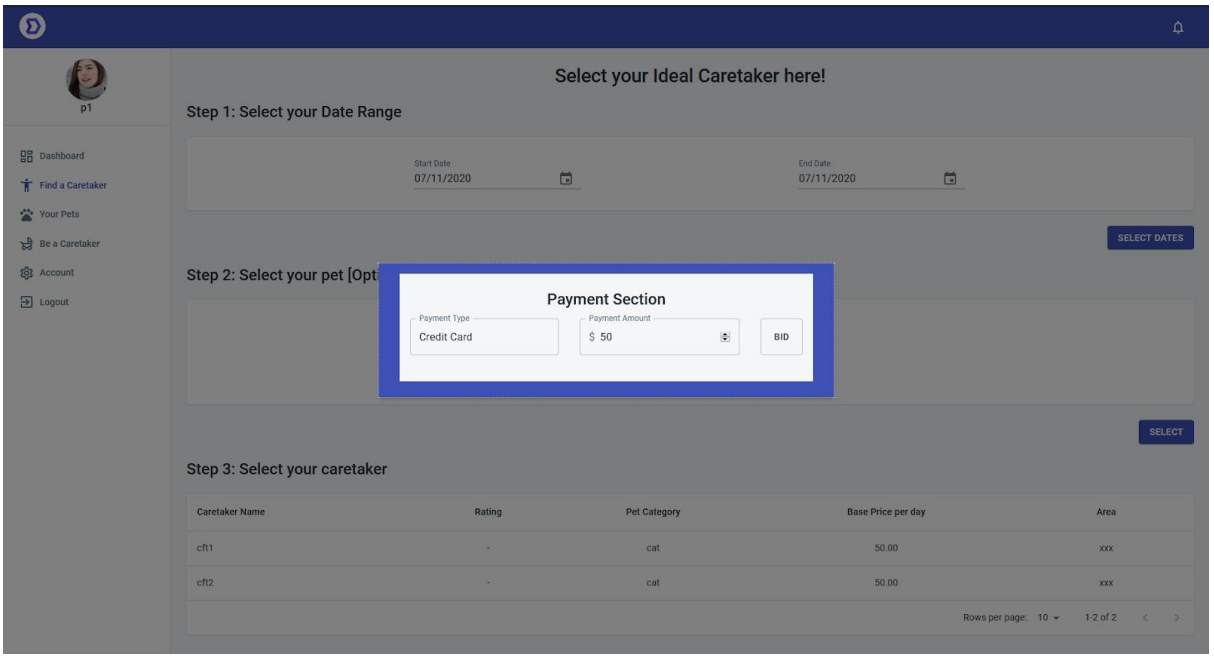
7. Specification of the software tools/frameworks used in your project.

Frontend: React
Backend: NodeJS, Express,
Debugging: Postman, Chrome devtools
Database: AWS RDS (PostgreSQL)
Infrastructure: Docker-compose, Github, Heroku

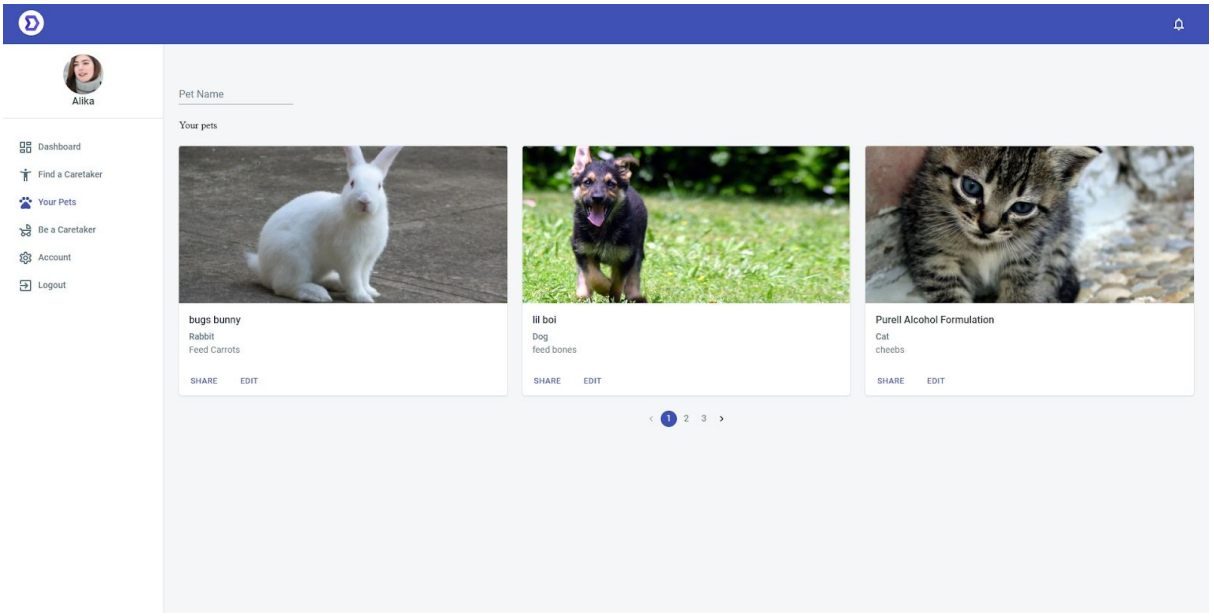
* represents additional constraints/features we added to the application

8. Two or three representative screenshots of your application in action.

1. **Catalogue - Applying bids (pet owner) [rusdi]**

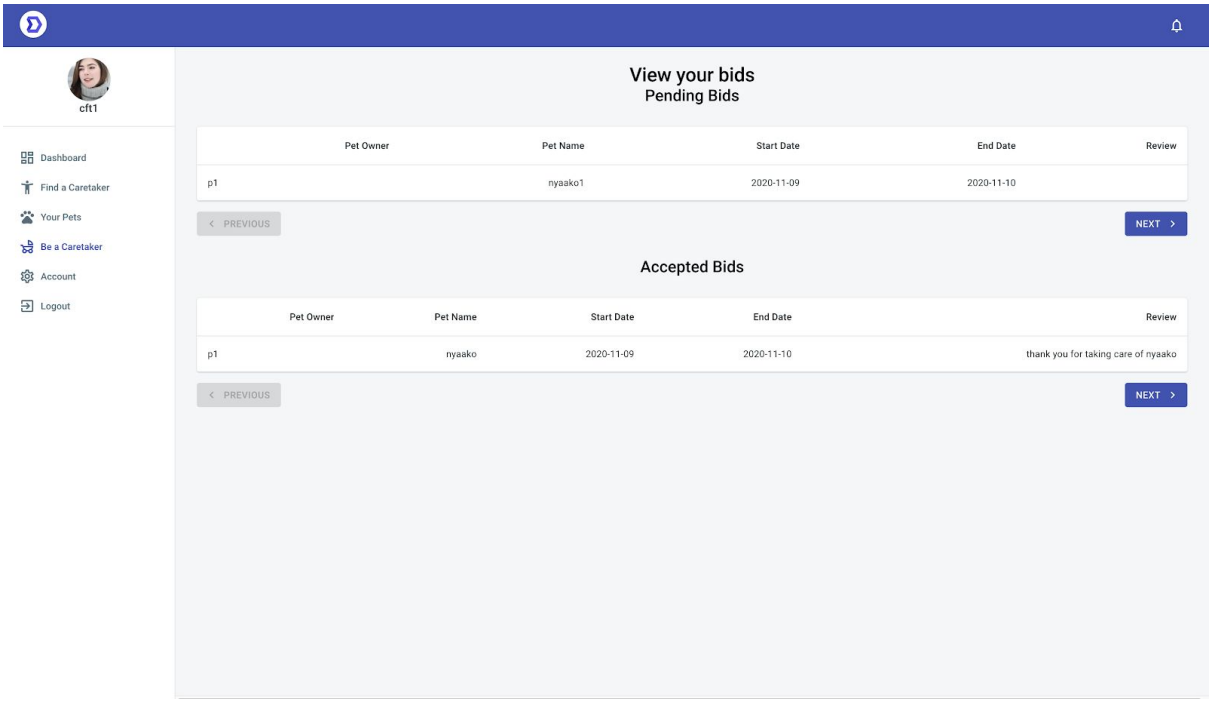


2. **Pets**



3. **Caretaker - View Bids**

* represents additional constraints/features we added to the application



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

1. Adding additional attributes to an entity
2. Thinking of the requirements for each SQL query for every API endpoint
3. Thinking of how each API endpoint will be useful/implemented on the frontend
4. Remote working and Covid-19

* represents additional constraints/features we added to the application