# CS2102 20/21 S1 Team 35

## Members

**Phyo Han** E0389166
**Derrick Teo** E0310242
**Eugene Tan Jianpeng** E0407410
**Ming Cheng** E0418001
**Chua Wee Kian, Glendon** E0415569

## Project Responsibilities

**Phyo Han** - ER model, Check relational schema/queries/triggers, Set up website/UI
**Derrick Teo** - ER model, Check relational schema/queries/triggers, Set up website/UI
**Eugene Tan** - ER model, Relational Schema, Queries/Triggers, Assist with website/UI
**Ming Cheng** - ER model, Relational Schema, Queries/Triggers, Assist with website/UI
**Chua Wee Kian, Glendon** - ER model, Relational Schema, Queries/Triggers, Assist with website/UI, Project Report

## Data Requirements & Functionalities

1. **The application allows new users to register as either a PCS admin, Pet Owner, Caretaker or a Pet Owner & Caretaker. Users are required to enter their email, name and password to register.**

   **Data Requirements:**
   - Users can register as either a “PCSadmin”, “Pet Owner” or “Caretaker” or “Pet Owner & Caretaker” if they intend to be both a Pet Owner and Caretaker. They can’t change their role afterwards.
   - User email and username must be alphanumeric, and not blank.
   - User password must be alphanumeric, at least 8 characters, and not blank.
   - Pet Owners and Caretakers have to provide their postal code(length >= 10) and address which cannot be blank. Additionally, Pet Owners can also provide a valid credit card number(length >= 10) .

   **Data Constraints:**
   - Every User has a name, password and email, and can be identified by their email.
   - Every User must be one or more of the following roles: PCSadmin, Pet Owner or Caretaker (Covering constraint).
   - A User can be both a Pet Owner and Caretaker(Overlapping constraint).
   - Every Caretaker has an address, postal code, and current rating, and can be identified by their email.
   - Every Pet Owner has an address, postal code, and credit card number, and can be identified by their email.

2. **The application allows Caretakers to indicate their available dates, the category of services and base price they want.**

   **Data Requirements:**
   - Caretakers must indicate the range of dates they are available to work, and cannot be left blank.
   - Caretakers must provide at least 1 service, and they can indicate the category of pets they are willing to take care of, as well as the base price they intend to charge for the service, both of which cannot be blank.

   **Data Constraints:**
   - Every Caretaker must either be a full timer or part timer(covering constraint).
   - Every Caretaker must provide at least 1 or more services.
   - Each Caretaker can specify the price for each of the services they provide. The stated price must be higher than the base price, which is set by the PCS.
   - Each Service has a category and base price, and can be identified by its category.
   - Each Availability has an available date range, and is identified by it.
   - Each Caretaker must declare their availability for the next year. If they are available for 2 * 150 consecutive days, they will be considered as a Full Timer for that year.
   - If the Caretaker is deleted, the Availability under them will also be deleted(identity dependency weak entity set).
   - The price of a full timer Caretaker depends on their rating:

     default price = base_price
     rating >= 3.5 ------> price = base_price * 1.05
     rating >= 4    ------> price = base_price * 1.1
     rating >= 4.5 ------> price = base_price * 1.15
   - The salary of a full timer Caretaker depends on the number of pet days they have. Their salary will be $3000 for up to 60 pet days, and for any excess pet days, they will receive 80% of their price as well.
   - The salary of a part time Caretaker will come from 75% of their stated price. The other 25% will go to the PCS as payment.
   - The pet limit of part time Caretakers depends on their current rating, as follows:

     rating >= 4.5 ------> 5 pets
     rating >= 4    ------> 4 pets
     rating >= 3.5 ------> 3 pets
     rating <   3.5 ------> 2 pets

3. **Pet Owners are able to register the pets that they own.**

   **Data Requirements:**
   - Pet Owners have to indicate their pet's name, and category of the pet, both of which cannot be blank.
   - Furthermore, Pet Owners can optionally indicate any special needs their pet requires.

   **Data Constraints:**
   - Every pet must have a Pet Owner.
   - Every pet has a name, category and special needs, and can be identified by its name and its owner.
   - Each Pet Owner must own at least 1 pet or more.
   - If the Pet Owner is deleted, the pets they own will also be deleted(identity dependency weak entity set).

4. **Pet Owners are able search for Caretakers and bid for the service they want. The system will then automatically match the Caretaker with the Pet Owner if the Caretaker is eligible for the job. Pet Owners can then choose to leave their ratings and review for the Caretaker once the care period is over.**

   **Data Requirements:**
   - To place a bid, Pet Owners have to indicate the Caretaker they want, the date range of the service, the name of their pet, the preferred payment mode, the preferred transfer mode, all of which cannot be left blank.
   - It is optional whether the PetOwner chooses to leave their ratings and review for the Caretaker for their service.

   **Data Constraints:**
   - Every bid has a Pet Owner email, Caretaker email, pet name, service date range, transfer mode, payment mode, total cost, rating, review, and a boolean attribute Successful. It can be identified by its Pet Owner email, Caretaker email, pet name, and service date range.
   - Each bid is placed by a Pet Owner/Pet pair where the Pet Owner owns the pet.
   - Each bid is chosen from a Caretaker/Service pair where the Caretaker provides the service.
   - A bid is immediately accepted to be successful if the Caretaker has not reached their pet limit. For full timers, the pet limit is 5 pets. For part timers, the pet limit is dependent on their rating as follows:

```
rating >= 4.5 ------> 5 pets
rating >= 4    ------> 4 pets
rating >= 3.5 ------> 3 pets
rating <   3.5 ------> 2 pets
```

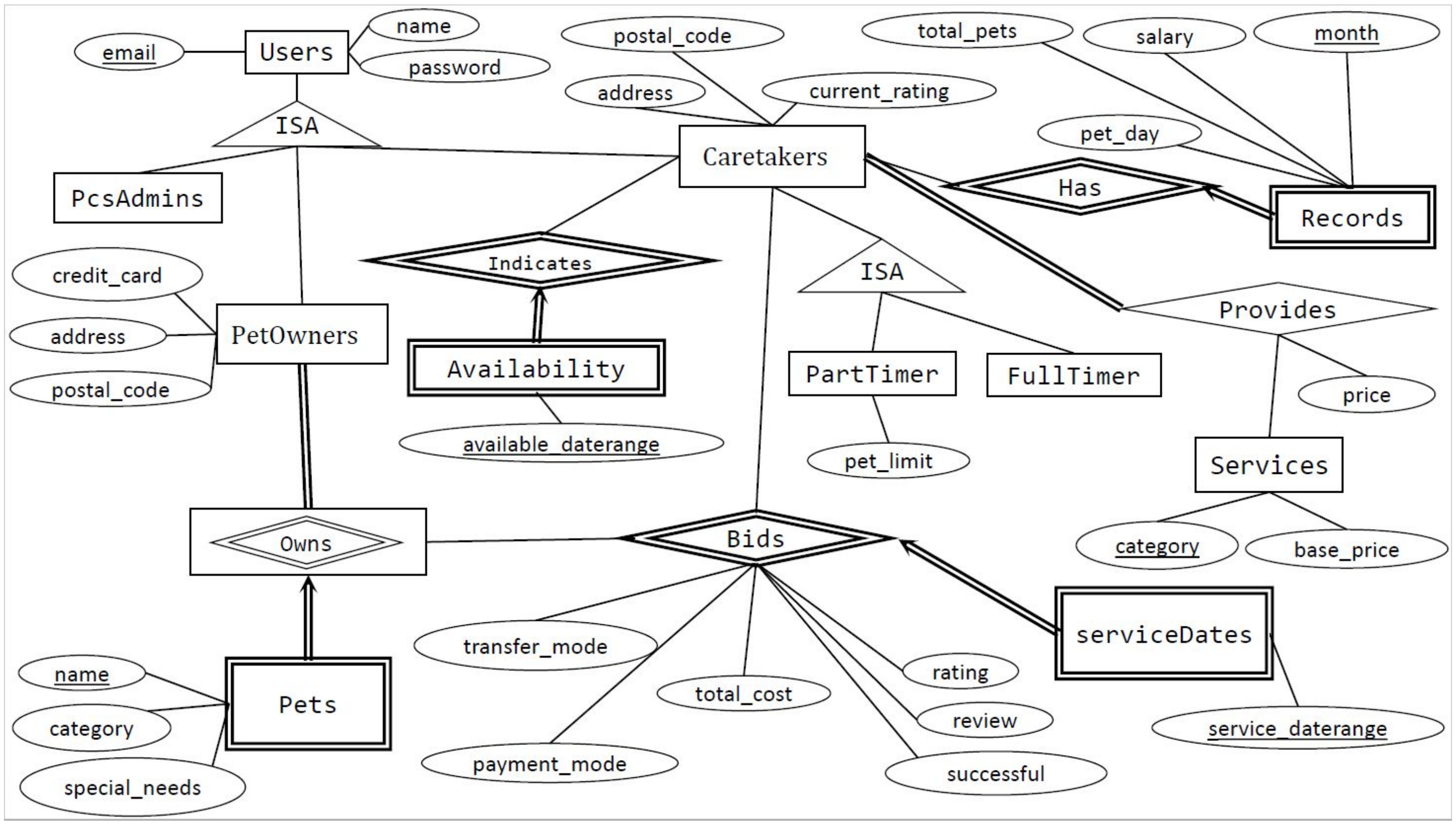
- In each Bid, the total cost is calculated by the number of days of the service period * daily price of the caretaker.
- For each bid, the Pet Owner can only bid for services for one of their pet (i.e. If the Pet Owner wants to have 2 of their pets taken care of, they have to place 2 separate bids.)
- After the care period, the Pet Owner can choose whether or not to leave a review and rating for the Caretaker.
- In each Bid, reviews and ratings can only be given once the care period is still over.
- After a bid is successful and the Pet Owner leaves a rating, the Caretaker's current rating will be updated.
- Every serviceDate has a service date range, and can be identified by it.
- The transfer mode of a Bid should be one of the following:
    - The Pet Owner delivers the pet('deliver')
    - The Caretaker picks up the pet('pickup')
    - The pet is transferred through the physical building of the PCS('thruPCS')

5. **PCS admins and Caretakers can check their own records to see their salary and pet-day for each month.**

**Data Requirements:**
- Employees have a monthly record of their salary, total number of pets taken care of for that month, and total pet days for that month.

**Data Constraints:**
- Every record has a month, salary, total pet count, total pet days count, and can be identified by the month and the caretakers that it belongs to.
- If a Caretaker is deleted, the records under them will also be deleted(identity dependency weak entity set).

## ER model



## Non-trivial Design Decisions:

- **ISA hierarchy where PSCAdmins, Pet Owners, and Caretaker are subclasses of User** so that the 3 types of users will inherit the attributes(name, email and password) of a User while having different roles/functionality.
- Similarly, **ISA hierarchy is also used for part timers and full timers** which are subclasses of Caretakers, allowing them to inherit Caretaker's attributes while differentiating between the 2 types of Caretakers.
- **Pet owners, Owns, and Pets form a weak entity set with identity dependency.** Each pet can only be owned by exactly 1 pet owner, but the name of the pet cannot uniquely identify the Pet Owner since different Pet Owners can have Pets with the same name.
- **Caretakers, Has, and Records form another weak entity set with identity dependency.** Each record belongs to only one Caretaker, but its attributes(month is only a partial key) cannot uniquely identify the Caretaker it belongs to, since the data in records are not unique(possible for 2 different Caretakers to have identical records).

- **Caretakers, Indicates, and Availability form another weak entity set with identity dependency.** Each availability entity can only be indicated by one Caretaker, but its attributes(available date range) cannot uniquely identify the Caretaker it belongs to, since 2 different Caretakers can indicate the exact same dates they are available to work on.
- **Caretakers, Owns(aggregate), Bids and serviceDates form a ternary relationship set.** A ternary relationship allows Pet Owners to bid for a Caretaker multiple times(i.e. Pet owner has more than 1 pet and wants a Caretaker to take care of all of them) within the same date range.
    - The **Owns relation is aggregated** so that it can be treated as an entity set instead of a relationship set.
- **Caretakers, Owns(aggregate), Bids and serviceDates also form another weak entity set with identity dependency.** Each serviceDate can only belong to one PetOwner/Caretaker pair for each Bid, but its attributes(service_daterange) cannot uniquely identify the PetOwner/Caretaker pair it belongs to, since a Pet Owner can bid for a Caretaker multiple times within the same date range.
- **Pet Owners have a total participation constraint to the Owns relation** as they are required to have 1 pet at minimum, but they are not limited to how many pets they may own.
- **Caretakers have a total participation constraint to the Provides relation** as Caretakers have to provide 1 service at minimum, but also they are not limited to the number of services they wish to provide.

## Uncaptured constraints:

1. **Covering constraints cannot be captured by ER diagram**
    - Every User must either be a PCSadmin, Pet Owner, or a Caretaker.
    - Every Caretaker must either be a full timer or part timer.
2. **Overlapping constraints also cannot be captured by ER diagram**
    - A User can be both a Pet Owner and Caretaker.

# Relational Schema

```sql
CREATE TABLE users (
	email VARCHAR PRIMARY KEY,
	name VARCHAR NOT NULL,
	role ENUM('pcsadmin', 'petowner', 'caretaker') NOT NULL,
	password VARCHAR NOT NULL CHECK (LENGTH(password) >= 8)
);

CREATE TABLE pcsadmins (
	email VARCHAR PRIMARY KEY,
	FOREIGN KEY (email) REFERENCES users(email) ON DELETE CASCADE
);

CREATE TABLE petowners (
	email VARCHAR PRIMARY KEY,
	credit_card VARCHAR(19) CHECK (LENGTH(credit_card) >= 10),
	address VARCHAR NOT NULL,
	postal_code VARCHAR(6) NOT NULL CHECK (LENGTH(postal_code) = 6),
	FOREIGN KEY (email) REFERENCES users(email) ON DELETE CASCADE
);

CREATE TABLE caretakers (
	email VARCHAR PRIMARY KEY,
	address VARCHAR NOT NULL,
	postal_code VARCHAR(6) NOT NULL CHECK (LENGTH(postal_code) = 6),
	current_rating NUMERIC(2, 1) CHECK (current_rating >= 1 AND current_rating <= 5),
	FOREIGN KEY (email) REFERENCES users(email) ON DELETE CASCADE
);

CREATE TABLE fulltimers (
	email VARCHAR PRIMARY KEY REFERENCES caretakers(email) ON DELETE CASCADE
);

CREATE TABLE parttimers (
	email VARCHAR PRIMARY KEY REFERENCES caretakers(email) ON DELETE CASCADE,
	pet_limit INTEGER NOT NULL CHECK (pet_limit >= 2 AND pet_limit <= 5)
);

CREATE TABLE ownedpets (
	name VARCHAR NOT NULL,
	email VARCHAR NOT NULL REFERENCES petowners(email) ON DELETE CASCADE,
	category VARCHAR NOT NULL REFERENCES services(category),
	special_needs VARCHAR,
	PRIMARY KEY (name, email)
);

CREATE table availability (
	email VARCHAR NOT NULL REFERENCES caretakers(email) ON DELETE CASCADE,
	available_daterange DATERANGE NOT NULL,
	PRIMARY KEY (email, available_daterange)
);

CREATE table services (
	category VARCHAR PRIMARY KEY,
	base_price NUMERIC(4, 2) NOT NULL
);

CREATE table provides (
	email VARCHAR NOT NULL REFERENCES caretakers(email),
	category VARCHAR NOT NULL REFERENCES services(category),
	price NUMERIC(4, 2) NOT NULL,
	PRIMARY KEY (email, category)
);

CREATE TYPE transfermode AS ENUM ('deliver', 'pickup', 'thruPCS');

-CREATE table bids (
	oemail VARCHAR NOT NULL,
	pname VARCHAR NOT NULL,
	cemail VARCHAR NOT NULL REFERENCES caretakers(email),
	service_daterange DATERANGE NOT NULL,
	total_cost NUMERIC(7, 2) NOT NULL,
	transfer_mode transfermode NOT NULL,
	payment_mode VARCHAR NOT NULL,
	rating NUMERIC(2, 1) CHECK (rating >= 1 AND rating <= 5),
	review VARCHAR,
	successful BOOLEAN,
	FOREIGN KEY (oemail, pname) REFERENCES ownedpets(email, name),
	PRIMARY KEY (oemail, pname, cemail, service_daterange)
);

CREATE table records (
	email VARCHAR NOT NULL REFERENCES caretakers(email),
	month DATERANGE NOT NULL,
	salary NUMERIC(8, 2) NOT NULL,
	pet_day INTEGER NOT NULL DEFAULT 0,
	PRIMARY KEY (email, month)
);
```

## Constraints not enforced by relational schema

1. The price of a full timer Caretaker depends on their rating:
   default price = base_price
   rating >= 3.5 ------> price = base_price * 1.05
   rating >= 4 ------> price = base_price * 1.1
   rating >= 4.5 ------> price = base_price * 1.15
   **A trigger is used to enforce and update the price according to the current rating of the Caretaker.**
2. Full time Caretakers must work for a minimum of 2 * 150 consecutive days. **A trigger is used to enforce it by making sure they fulfil the requirements before being considered a full timer.**
3. The salary of a full timer Caretaker depends on the number of pet days they have. Their salary will be $3000 for up to 60 pet days, and for any excess pet days, they will receive 80% of their price as well. **A trigger is used to calculate the number of pet days a Caretaker has, and another trigger is used to calculate their salary.**
4. The salary of a part time Caretaker will come from 75% of their stated price. The other 25% will go to the PCS as payment. **A trigger is used to calculate their salary.**
5. The pet limit of part time Caretakers depends on their current rating, as follows:
   rating >= 4.5 ------> 5 pets
   rating >= 4 ------> 4 pets
   rating >= 3.5 ------> 3 pets
   rating < 3.5 ------> 2 pets
   **A trigger is used to update the pet limit of part timer Caretakers based on their current rating after any rating given from a successful Bid.**
6. In each Bid, the total cost, calculated by the number of days of the service period * daily price of the caretaker, is not captured. **A trigger is used to calculate it.**
7. In each Bid, reviews and ratings can only be given once the care period is still over. **A trigger can be used to ensure that Pet Owners will only be able to leave a rating and review once the care period is over.**
8. A bid is immediately accepted to be successful if the Caretaker has not reached their pet limit. For full timers, the pet limit is 5 pets. For part timers, the pet limit is dependent on their rating. **A trigger is used to ensure that bids are only allowed to be set as successful if the Caretaker fulfils the criteria.**
9. After a bid is successful and the Pet Owner leaves a rating, the Caretaker's current rating will be updated. **A trigger is used to update it.**

## BCNF or 3NF Discussion

**These are the data constraints that we want to enforce in our table.**

Table | FD

- **Pcsadmins:** Admin’s email → Admin’s email
- **Fulltimers:** Full Timer’s email → Full Timer’s email
- **Parttimers:** Part Timer’s email → Pet Limit
- **Users:** User’s email → Username, password, and User role
- **Petowners:** Pet Owner’s email → Pet Owner’s credit card, address, and postal code
- **Caretakers:** Caretaker's email → Caretaker’s address, postal code and their current rating
- **Ownedpets:** Pet’s name, Pet Owner’s email → Pet category, Pet’s special needs(if any)
- **Availability:** Caretaker's email, available period → Caretaker's email, available period
- **Services:** Pet category → base price
- **Provides:** Care Taker's email, pet category → price
- **Bids:** Pet owner’s email, Care Taker's email, Pet’s name, service period → Mode of Transfer, payment method, Total Cost, rating, review, successful
- **Records:** Care Taker's email, month → month’s salary, number of pet days fulfilled in month

**Since all functional dependencies for all table fragments fulfil at least one of the two conditions for BCNF, the database is in BCNF.**

**Since all schemas are in BCNF, it follows that all schemas are also in 3NF.**

## 3 Non-trivial/Interesting Triggers

1. **The price of a full time Caretaker depends on their rating, as follows:**

```
default price = base_price
rating >= 3.5 ------> price = base_price * 1.05
rating >= 4   ------> price = base_price * 1.1
rating >= 4.5 ------> price = base_price * 1.15
```

The trigger checks the current rating of the full time Caretaker and updates the price of the Caretaker accordingly.

```
CREATE FUNCTION update_price()
RETURNS TRIGGER AS
$$ DECLARE ctx NUMERIC;
   BEGIN
    SELECT current_rating INTO ctx FROM caretakers c WHERE NEW.cemail = c.email;
    IF NEW.cemail IN (SELECT email FROM fulltimers) THEN
        IF ctx >= 4.5 THEN
            UPDATE provides SET price = (SELECT base_price FROM services s WHERE
provides.category = s.category) * 1.15
            WHERE email = NEW.cemail;
            RETURN NEW;
        ELSIF ctx >= 4 THEN
            UPDATE provides SET price = (SELECT base_price FROM services s WHERE
provides.category = s.category) * 1.1
            WHERE email = NEW.cemail;
            RETURN NEW;
        ELSIF ctx >= 3.5 THEN
            UPDATE provides SET price = (SELECT base_price FROM services s WHERE
provides.category = s.category) * 1.05
            WHERE email = NEW.cemail;
            RETURN NEW;
        ELSE
            UPDATE provides SET price = (SELECT base_price FROM services s WHERE
provides.category = s.category)
            WHERE email = NEW.cemail;
            RETURN NEW;
        END IF;
    ELSE
        RETURN NULL;
    END IF;
   END;
```

```
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_price
AFTER UPDATE ON bids
FOR EACH ROW EXECUTE PROCEDURE update_price();
```

2. **Calculate the salary of each caretaker for that month.**

```
CREATE FUNCTION calculate_salary()
RETURNS TRIGGER AS
$$ DECLARE ctx NUMERIC;
   BEGIN
    SELECT SUM(total_cost) INTO ctx FROM bids b
        WHERE b.cemail = NEW.email AND LOWER(b.service_daterange) <@ NEW.month
AND b.successful = true;
    IF NEW.email IN (SELECT email FROM parttimers) THEN
        NEW.salary = ctx * 0.75;
        RETURN NEW;
    ELSE
        IF NEW.pet_day <= 60 THEN
            NEW.salary = 3000;
            RETURN NEW;
        ELSE
            NEW.salary = (3000 + (ctx/NEW.pet_day * (NEW.pet_day - 60) * 0.8));
            RETURN NEW;
        END IF;
    END IF;
   END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER calculate_salary
BEFORE INSERT ON records
FOR EACH ROW EXECUTE PROCEDURE calculate_salary();
```

3. **Full-time Caretakers must work for a minimum of 2 * 150 consecutive days. The trigger checks if the Caretaker fulfils the criteria(from the availability indicated by the Caretaker) before the Caretaker is added to the fulltimers table. If the condition is not fulfilled, the Caretaker will be considered as part timers.**

```
CREATE OR REPLACE FUNCTION check_shift()
RETURNS TRIGGER AS
$$ DECLARE ctx NUMERIC;
  BEGIN
    SELECT COUNT(*) INTO ctx FROM availability a
    WHERE NEW.email = a.email AND UPPER(NEW.daterange) - LOWER(NEW.daterange) >=
150
    IF ctx < 2 THEN
      RETURN NULL;
    ELSE
      RETURN NEW;
    END IF; END; $$
LANGUAGE plpgsql;

CREATE TRIGGER check_full_timer
BEFORE INSERT OR UPDATE ON fulltimers
FOR EACH ROW EXECUTE PROCEDURE check_shift();
```

4. **In a Bid, if the Caretaker has not reached their pet limit, the Bid is immediately accepted to be successful. For full timers, the pet limit is 5 pets. For part timers, their pet limit is dependent on their rating. The trigger checks if the criteria is fulfilled before updating the Bid to be successful.**

```
CREATE FUNCTION check_caretaker_limit() RETURNS TRIGGER AS $$
DECLARE pet_count INTEGER;
   BEGIN
    SELECT COUNT(*) INTO pet_count FROM bids b
        WHERE b.cemail = NEW.cemail AND LOWER(NEW.service_daterange) <@
b.service_daterange AND b.successful = true;
    IF NEW.cemail IN (SELECT email FROM parttimers) THEN
        IF pet_count >= (SELECT pet_limit FROM parttimers p WHERE p.email =
NEW.cemail) THEN
            NEW.successful = false;
            RETURN NEW;
        ELSE
            NEW.successful = true;
            RETURN NEW;
        END IF;
    ELSE
        IF pet_count >= 5 THEN
            NEW.successful = false;
            RETURN NEW;
        ELSE
            NEW.successful = true;
            RETURN NEW;
        END IF;
    END IF;
   END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_caretaker_limit
BEFORE INSERT ON bids
FOR EACH ROW EXECUTE PROCEDURE check_caretaker_limit();
```

### 5. Calculate the number of pet days for each caretaker for that month.

```
CREATE FUNCTION calculate_pet_day()
RETURNS TRIGGER AS
$$ DECLARE ctx INTEGER;
   BEGIN
    SELECT SUM(days) INTO ctx FROM (SELECT UPPER(service_daterange) -
LOWER(service_daterange) AS days FROM bids WHERE cemail = NEW.email AND
LOWER(service_daterange) <@ NEW.month AND successful = true) AS b;
    NEW.pet_day = COALESCE(ctx, 0);
    RETURN NEW;
   END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER calculate_pet_day
BEFORE INSERT ON records
FOR EACH ROW EXECUTE PROCEDURE calculate_pet_day();
```

# 3 Complex Queries

1. **View number of jobs done and pet days by all caretakers for a particular month from lowest to highest and also their salaries from highest to lowest. Replace **** with the daterange of that month e.g. January = DATERANGE('[2020-01-01, 2020-01-31]')**

```
SELECT email, COALESCE(jobs, 0) AS jobs_done, pet_day, salary FROM records LEFT
JOIN
(SELECT cemail, COUNT(*) AS jobs FROM bids WHERE service_daterange <@
DATERANGE('[****-**-**, ****-**-**]') AND successful = true GROUP BY cemail) AS b
ON email = cemail AND month = DATERANGE('[****-**-**, ****-**-**]')
ORDER BY jobs_done ASC, pet_day ASC, salary DESC;
```

2. **Find all available Caretakers that take care of a particular type of pet within a certain period ordered by price from lowest to highest and rating from highest to lowest. Replace **** with the starting date, ending date, and the category of pets.**

```
SELECT DISTINCT c.email, c.name, p.price, c.current_rating
FROM (caretakers c INNER JOIN availability a ON c.email = a.email) INNER JOIN
provides p ON c.email = p.email
WHERE DATERANGE('[****-**-**, ****-**-**]') <@ a.available_daterange AND
p.category = '***'
ORDER BY p.price ASC, c.current_rating DESC;
```

3. **Find the service offered that has the worst value for money**

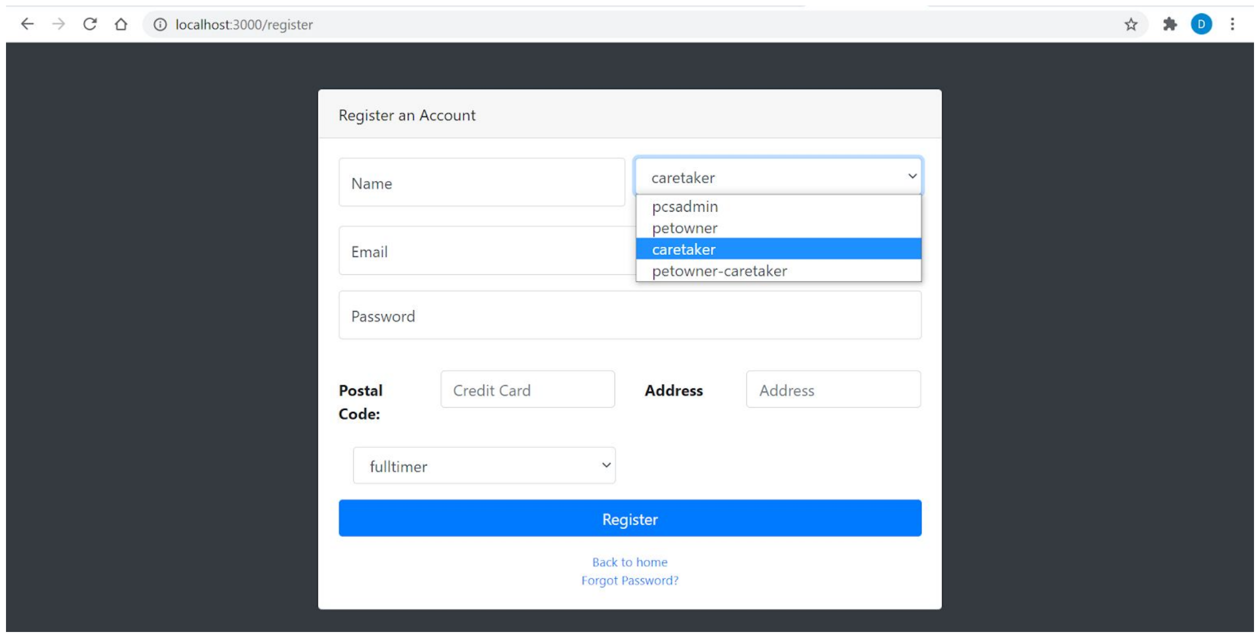
```
SELECT t3.name, t3.email, t3.current_rating, t3.category, t3.price
FROM ((SELECT c1.name, c1.email, c1.current_rating FROM caretakers c1 ORDER BY
c1.current_rating ASC LIMIT 10) AS t1 JOIN (SELECT p1.email AS cemail,
p1.category, p1.price FROM provides p1 ORDER BY email) AS t2 ON t2.cemail =
t1.email) AS t3
ORDER BY t3.price DESC, t3.current_rating ASC;
```

## Software tools/frameworks

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

- **Heroku -** To host the database and website online.
- **NodeJS -** To host logic
- **ExpressJs -** Backend web application framework
- **PostgreSQL** - Language used to manage the database in Heroku.
- **Git/Github -** Version control and source code management.

## Screenshot of Application


localhost:3000/register

Register an Account

Name

caretaker

pcsadmin
petowner
caretaker
petowner-caretaker

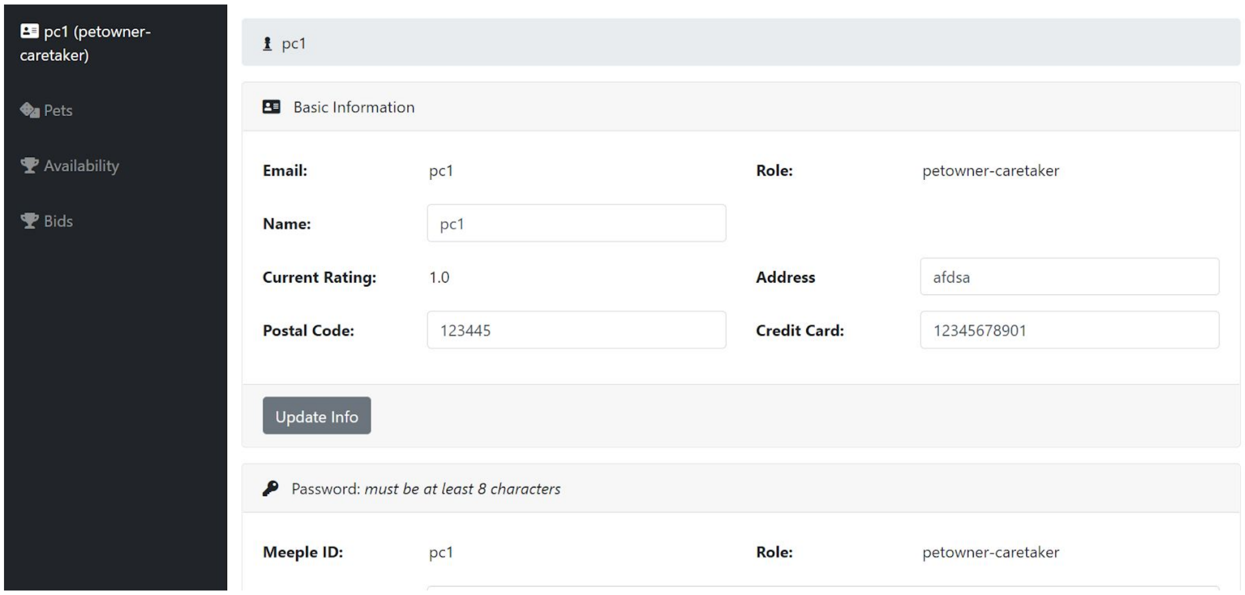
Email

Password

**Postal Code:** Credit Card **Address** Address

fulltimer

Figure 1. Registration Page

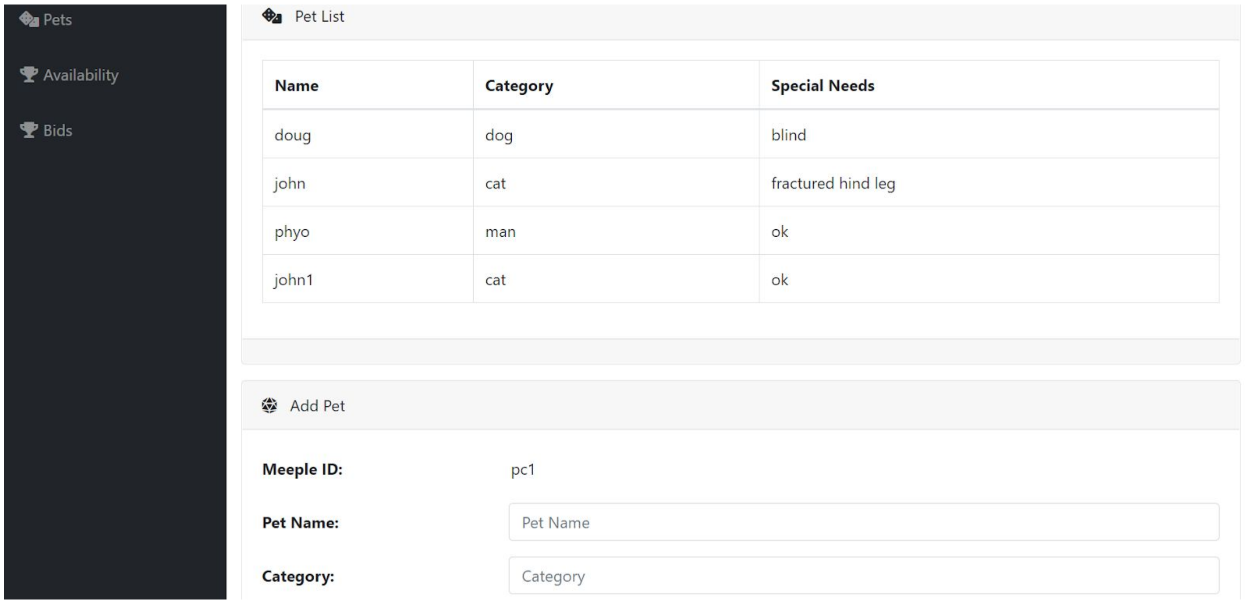


| | | | |
|---|---|---|---|
| **Email:** | pc1 | **Role:** | petowner-caretaker |
| **Name:** | pc1 | | |
| **Current Rating:** | 1.0 | **Address** | afdsa |
| **Postal Code:** | 123445 | **Credit Card:** | 12345678901 |

Password: *must be at least 8 characters*

| | | | |
|---|---|---|---|
| **Meeple ID:** | pc1 | **Role:** | petowner-caretaker |

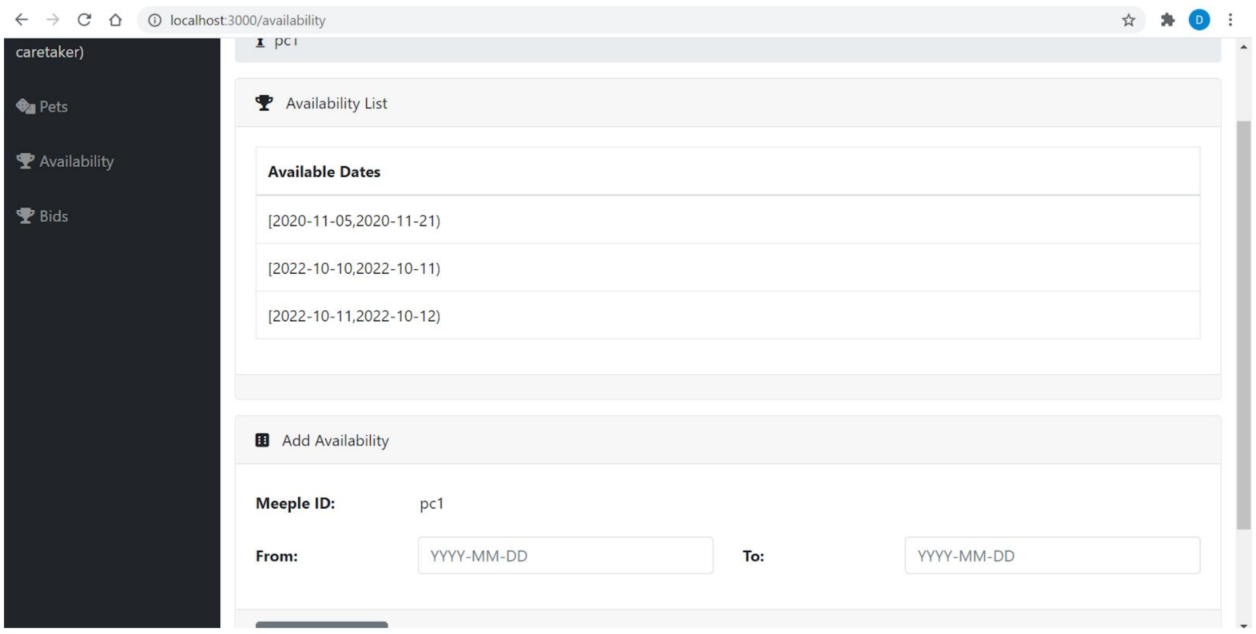
Figure 2. Dashboard



| **Name** | **Category** | **Special Needs** |
|---|---|---|
| doug | dog | blind |
| john | cat | fractured hind leg |
| phyo | man | ok |
| john1 | cat | ok |

| | |
|---|---|
| **Meeple ID:** | pc1 |
| **Pet Name:** | Pet Name |
| **Category:** | Category |

Figure 3. Pets Page



Availability List

| Available Dates |
|---|
| [2020-11-05,2020-11-21) |
| [2022-10-10,2022-10-11) |
| [2022-10-11,2022-10-12) |

Add Availability

Meeple ID: pc1

From: YYYY-MM-DD To: YYYY-MM-DD

Fig 4. Availability Page

## Difficulties Encountered/Lessons Learnt

### Difficulties encountered:

**ER Diagram and SQL tables:** Our ER diagram and sql tables have gone through multiple revisions throughout the semester as we are constantly finding new issues and/or better ways to implement certain things while working on the project (e.g. removing redundant attributes or adding a new entity table to enforce certain constraints. As such, we are still updating and refining the ER diagram and our tables even towards the end of the project.

**Setting up the Website:** All of us have very little experience with working with a website. As such, working on the website is a difficult task for us since we need to continuously refer guides and help online. The use of async functions in the context of Network communication is another challenge as it is often the culprit of the website not responding as intended.

**Use of Skeleton:** Though Prof Adi is kind enough to provide us with a skeleton which really saves our time to set up a website, all of us are inexperienced working on top of other codes. Combined with the fact that it is our first time using Heroku, npm and postgresql, we spent a lot of time just to get all these different components to work together.

**Proving if the database is in BCNF:** BCNF and 3NF are concepts that are not easy to grasp quickly, and we found ourselves struggling to check if our database is in BCNF. Moreover, we started the design of the database with the ER-diagram and based our Postgresql implementation based on the ER-diagram hence it is harder to check if the database is in BCNF. In addition, the database is so huge that the process of proving it becomes difficult.

## Lessons Learnt:

**Starting Earlier:** As we are working on something new, we require more time to self learn and familiarise ourselves with Heroku. NodeJS and even on the basics of Git/Github. It also took some time to set up all the required systems. Learning the SQL concepts in lectures and tutorials are also very different from actually implementing them in a project. The last few weeks of the semester are highly packed with assignments and reports from other mods as well, hence we should have started earlier and worked with a consistent timeline.

**Using SQL to model real life problems/databases:** We realised that it was not easy to accurately model a real life problem, as there were many things to consider. Coming up with a good ER diagram took a while, after much discussion and revisions, and even then the ER diagram probably is not able to capture every single constraints we wanted. Creating SQL tables and utilising triggers to enforce every single constraint we wanted was also tough. Most importantly, actually implementing all the SQL code with the application we set up on Heroku took a lot of time and effort.