Designing a Database for PAWsitive Dog Care

San Jose State University
Project Partners: Tien Nguyen and Shayna Gaulden

Report prepared for CS157A
Instructor Dr. Jahan Ghofraniha

TABLE OF CONTENTS

I. EXECUTIVE SUMMARY

A new company, PAWsitive Dog Daycare is seeking a database for its dog daycare center that offers boarding, grooming and training. The company wants a database that keeps track of their employees' schedules and customers' bookings information. The database must go through a formal database design lifecycle to be in at least third normal form. This report describes the process of going through the database design lifecycle to create the database, the detailed design of the database, and the final product/release version of the database and UI.

Based on the company's intended use for the database, the end result to access the database will be a web application that anyone can access through the URL. We designed a database that met the needs of the company and provided them with a usable web application. The initial implementation of the database can be done one time by the database designers but once the website is launched all future data can be input through the website.

II. INTRODUCTION/BACKGROUND

This report details the design process and steps taken to create a database for a fake business called PAWsitive Dog Daycare. The business is not real, but is treated as if it is real throughout this report to achieve the best database design. The design process required us to consider the potential needs of a real dog daycare and make decisions as if our company was real. In our scenario, PAWsitive Dog Daycare is a new business with some existing customers. The business is in need of a database that can keep track of their services and operations. The company is in need of software that can manage the day to day operations.

This database will need customer, daycare attendant and manager functionality. PAWsitive Dog Care is a doggy daycare that offers boarding, grooming, and training sessions for dogs. Customers will be checking in/out and paying at the daycare center. Customers will be able to see available time slots for grooming, training and boarding services and book services. Employees will need to check daily boarding reservations, groomers and trainers will need to see their booking schedules. Managers will need access to bookings/reservations for all departments and will need to keep track of employees and their different departments. The user interface for all employees and customers is a web application.

III. PROBLEM STATEMENT

This project is aimed at designing and creating a database for a new company PAWsitive Dog Daycare. The company currently has all their data stored in flat files and needs a better system that can be used by the employees and customers to allow online booking of services. They will need a way to transfer their current data onto the new database once it is created. The database will also be used to keep track of employees on staff, their information, the department they belong to and their schedules. It will store information on customers and their dogs and any bookings past and future made by customers. There will also be information on the services offered for each department and the price of each service.

IV. PURPOSE/MOTIVATION

The purpose is to provide PAWsitive Dog Care with a database to meet their current business needs. PAWsitive Dog Daycare is relatively new with only a handful of customers. This business hopes to open in a permanent location, but will need a database before their official opening. The database will be designed through the formal database design life cycle. It is important that the database is designed properly so that the business has the opportunity to expand and grow. A well thought out database design should be able to accommodate new changes as needed, and be designed to hold large amounts of data with minimal changes to its structure. This report will provide the details and steps taken throughout the process.

V. DESIGN

A. Conceptual

Before starting the design of the database, we needed to first do some requirement gathering to understand the business rules. Our requirement gathering consisted of interviewing the business owners for their database needs, the business owners of course being Tien Nguyen and Shayna Gaulden. What follows are the discovered business rules.

Business rules:

- A manager manages other employees and their schedules. She can change any employee's schedule and modify the price of services.
- Groomer/Trainer/Boarding attendant: These employees can only view their schedules but cannot make changes.
- There are 4 types of employees which are a manager, groomer, trainer, and boarding attendant, so the type of employee must be kept track of somehow.
- Customers can see all available time slots, and can't see any time slots that are full.
- Customers can reserve a booking time slot for one dog at a time.
- A customer account can have many different dogs and each reservation is made for only one dog at a time.
- The business must have a way to store up to date vaccine information for dogs.
- Both employees and customers can view the pricing and services, but only managers can edit or add new services.
- Customers can create an account before logging in. A customer must have an account before a booking is made and at least one dog must be added to the customer's account at the time of account creation.
- Because customers can have multiple dogs, each booking must have one and only one dog specified.
- Customers can add a dog after logging in.
- An employee can't have bookings at the same time, so their schedule hours cannot overlap.
- Customers pay at the time of check in at the register, if booked for multiple services at one time (common for grooming) then all bookings for that day will be summed together and paid for at once.
- An employee can only work in one department.
- Boarding employee maximum number of dogs per bookings is 8
- For now, the company only has private training sessions which means each training employee only trains one dog at a time.

From the business rules, we created our initial conceptual design which had 6 entities, Customer, Dog, Employee, Bookings, Schedule/Calendar, and Services. *Figure 1* shows the six entities with a description and their relationship. At this point in the design process, we had to know some of the business rules to understand how these entities would interact. A customer can have many dogs but they need to have at least one dog in the system. So we knew at the time of account creation a customer would also need to add a dog. A dog relates to one and only one customer, so for families, or couples with co-ownership, they would need to put one person's details in our system and share the account. A dog could be referenced by zero or many bookings but each booking had to reference one and only one dog. Schedule entity relates to zero or one bookings and bookings relate to one and only one schedule. Employees are referenced one or many times in the schedule and the schedule can also reference one or many employees. Finally, we have that a service can be referenced many or no times in a booking but a booking will have one and only one service per booking.

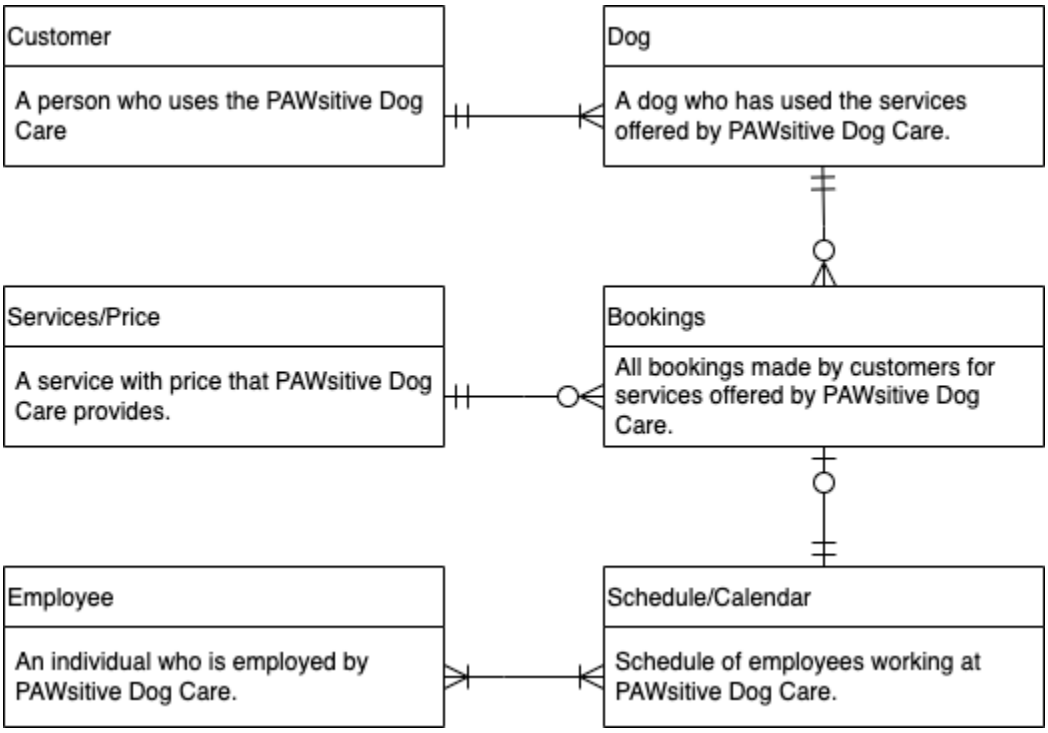


Fig 1: Conceptual design PAWsitive Dog Care

At this point in the design stage, we needed to consider how the database would be viewed by different people that would be accessing the database. We considered three main categories of people that would have different views of the database. The categories are employees, managers, and customers. Employees and managers should have similar views but managers will be able to see more sensitive information and can do things that employees can't do. Managers will be able to add a new employee, see a list of employees and their addresses, create employee's schedules, and update or add new services. *Figure 2* shows how we thought user flow would look like on our web application and what views and forms we would need. The following conceptual views were decided on, that would be implemented later on.

First, a view of all the employees with their social security number hidden. Second, a schedule view for employees and managers that would show all the employee's schedules and whether or not they had an upcoming booking along with the information of the customer name and dog's name. Originally, for this view, we were going to have each employee seeing only their own schedule, but we decided it would be useful for employees to see the schedule of other employees so they could know who was working and be able to promote to customers any openings in the schedule. For example, a dog trainer could recommend a puppy in a training class sign up for grooming after class if there are openings. We needed a separate view for customers to see available bookings in what we call the calendar view. There would need to be a view that showed all services their details and prices. Finally, we would need a view for customers to see their upcoming reservations.

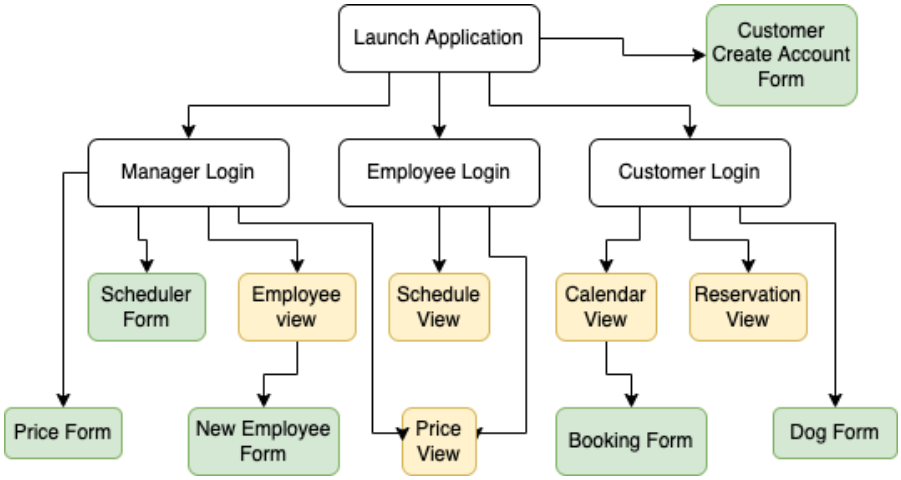


Fig 2: Storyboard of client flow.

B. Logical

Below we walk through the steps we took to normalize our database design.

Step 1: Initial Design

CUSTOMER: *CUST_ID* (pk), Customer Name, Phone number, Customer Address, City, State, Zip Code, Country

DOG: *DOG_ID* (pk), CUST_ID, Dog Name, Size, Breed, Vaccine Expiration, Spayed/Neutered, Age, Gender

EMPLOYEE: *EMP_ID* (pk), Department, Name, Address, City, State, Zip Code, County, Social Security Number

SCHEDULE: (*EM P_ID, TIME-DATE* (pk))

BOOKINGS: (*DOG_ID, EMP_ID, TIME-DATE* (pk)), Service

SERVICES: *SERVICE* (pk), Department, Details, Price

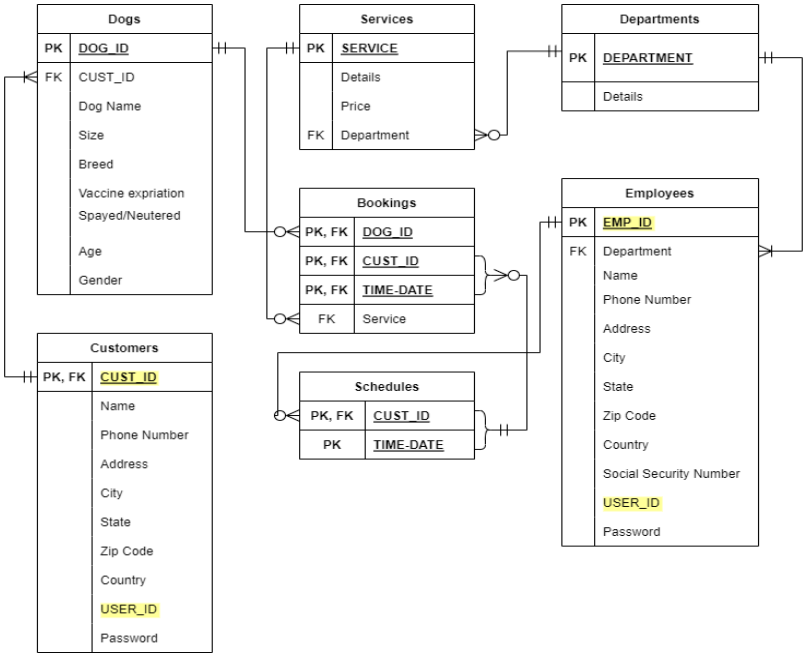


Fig 3: Initial ERD for PAWsitive Dog Care

We noticed that we needed a department entity which will be added in the next step. In order for customers or employees to log in to our system, we realized that we would need to store a USER_ID and password attribute that we add in the next step. We also realized that since boarding employees could handle multiple dogs for each schedule time slot, there would be repeated values in the BOOKINGS table. This means that our initial design is not in first normal form. To fix this we added a SUB_ID attribute in SCHEDULE and BOOKINGS that is an integer increment by 1 for each 'slot' that is available.

Step 2: Adding new attributes

CUSTOMER: *CUST_ID* (pk), Customer Name, Phone number, Customer Address, City, State, Zip Code, Country, USER_ID, Password
DOG: *DOG_ID* (pk), CUST_ID, Dog Name, Size, Breed, Vaccine Expiration, Spayed/Neutered, Age, Gender
EMPLOYEE: *EMP_ID* (pk), Department, Name, Address, City, State, Zip Code, County, Social Security Number, USER_ID, Password
SCHEDULE: (*EMP_ID, TIME-DATE, SUB_ID* (pk)),
BOOKINGS: (*DOG_ID, EMP_ID, TIME-DATE* (pk)), Service, SUB_ID
SERVICES: *SERVICE* (pk), Department, Details, Price
DEPARTMENT: *DEPARTMENT* (pk), Details

Each table is now in first and second normal form (1NF and 2NF). However, we thought adding USER_ID and password attribute violated the third normal form (3NF) because the password could depend on USER_ID causing a transitive dependency. One more problem is that USER_ID can be duplicated because it is unique for customers and employees but not unique between the two. For example, a customer and an employee could have the same USER_ID. Therefore, we created a User table in step 3.

Step 3: Creating new table to fix the 3NF violation

CUSTOMER: *USER_ID* (pk), Customer Name, Phone number, Customer Address, City, State, Zip Code, Country
DOG: *DOG_ID* (pk), CUST_ID, Dog Name, Size, Breed, Vaccine Expiration, Spayed/Neutered, Age, Gender
EMPLOYEE: *USER_ID* (pk), Department, Name, Address, City, State, Zip Code, County, Social Security Number
SCHEDULE: (*EMP_ID, TIME-DATE, SUB_ID* (pk)),
BOOKINGS: (*DOG_ID, EMP_ID, TIME-DATE* (pk)), Service, SUB_ID
SERVICES: *SERVICE* (pk), Department, Details, Price
DEPARTMENT: *DEPARTMENT* (pk), Details
USERS: *USER_ID* (pk), Password, User Type

User table has been added and now has USER_ID, password, and user type which indicates type of user such as employee or customer. USER_ID is unique for each customer or employee so we replace CUST_ID or EMP_ID by USER_ID to make USER_ID a primary key for the Customer and the Employee

table. Now every table follows up to the 3NF. See *Figure 3* for the entity relationship diagram of the final database design.

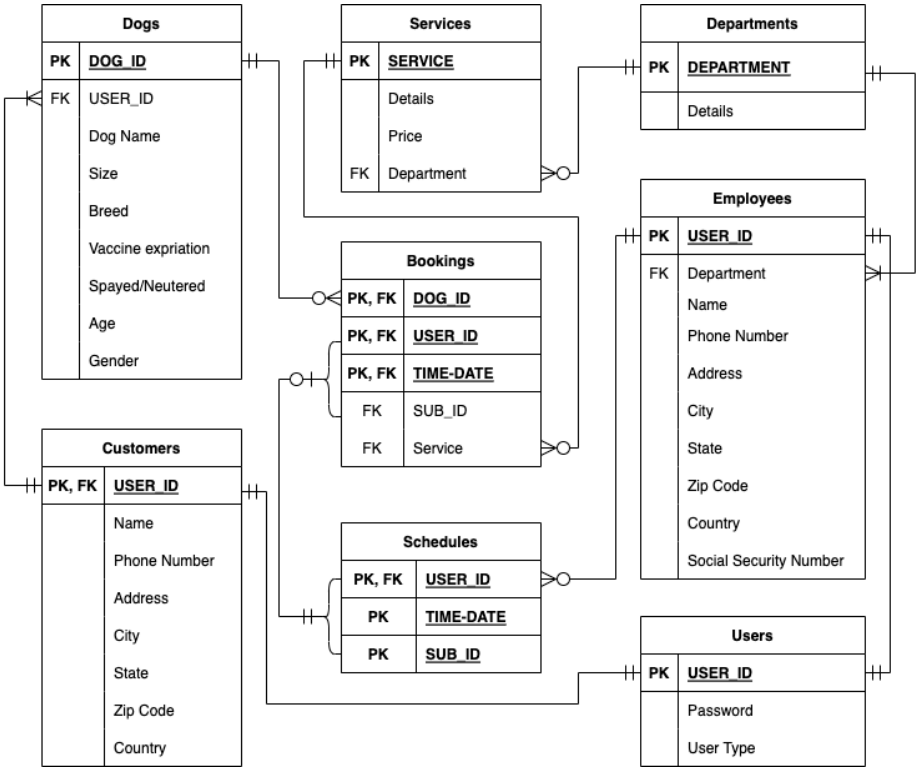


Fig 4: Final ERD for PAWsitive Dog Care

C. Physical

We initially implemented the database in mySQL which we used as a test environment to create the tables, add constraints, and create the views. Writing the SQL code to create the tables followed easily from the ERD diagram. We decided to create the tables and add the primary keys in the initial table creation, but add foreign keys afterwards. This choice was made simply as a stylistic choice to improve code readability. To view the SQL code to create the tables view appendix B figures B.2 - B.9

The four views are created in SQL, the code for the four views can be seen in appendix B figures B.11 - B.14. Below are details of the views.

- EMPLOYEES_VW is a view created for the managers, so they can insert/update/delete an employee and see the current employee information while hiding the employees social security number.
- CALENDAR_VW is for the customers to see only available schedules and therefore available slots to book. To create this view we used a left join to the bookings table so the schedule would not show up if it could be matched to a bookings entry.
- SCHEDUALS_ALL_VW is for employees and managers so they can see when employees are scheduled and see the dog/customer information if someone has made a booking. We made sure that the schedule will still show up even without a booking.

- TOTAL_PRICE_VW is the calculated total price after services are booked. When a customer has selected multiple bookings all of the service prices are summed up to create the total price. This will be shown under their upcoming reservations.

Originally, we did not include USER_ID in the views because we wanted it to be hidden from the customer/employee. However, we had to add them to the views so that it was easier for us to insert data in the webapp since in the webapp we can hide the USER_ID field but then also use it to insert data. Without this field being present in the view, we would need to make some assumptions such as employee and customer names must be unique which we wanted to avoid.

VI. IMPLEMENTATION & TEST REPORT

A. Implementation

Once the database was created in MySQL, we used Google Cloud Platform and launched an SQL instance. They offer a free 90 day trial, past the trial the business would need to take over this account to make payments for hosting the database in the cloud. We deployed one instance that is configured to allow SSL connections. We then need to use the IP address of our instance to connect to it in Python. In Python, we created a script that connected to the cloud instance and ran our SQL code to create the database, tables, constraints, and views. Since our company already had some data that was kept in excel files, we converted these files to CSV and used Python to insert the data into our SQL database.

Then we made a web application in Javascript using React JS as a framework with components from Material UI. We implemented an API with Node JS and Express and we used Postman to test our api calls. We staged a development environment that was hosted on our local machines for easier testing so we could both work on our project. After testing, we were ready to fully deploy our first release version. First, we used heroku to deploy the backend which is a free option and it was deployed to this URL: https://pawsitivedogdaycare.herokuapp.com/. Once we had the URL for the backend, we changed the variables in our code for the frontend to point to the new URL instead of pointing to the local host. Finally, we used Vercel to deploy the front end of our web application. Once deployed, we tried a few test cases just to make sure everything was still working. We chose Heroku and Vercel for the deployment because it was the simplest choice and it is free. Our web app can be visited with the URL in appendix A. It will be live until the free trial expires in Google Cloud Platform.

B. Testing

Case 1: SQL Implementation for physical design

Case 1a: Run the SQL script to create table/views and add all foreign key constraints and ensure there are no errors and schema is properly created.

Case 1b: Insert all data into the database and query to select all from tables to ensure data is being added successfully.

Case 2: Insert/Delete/Update in SQL these test mostly ensure the foreign keys are properly created and the relationship between the tables make sense.

Case 2a: Delete an entry from each table to ensure when there is a foreign key the delete occurs as expected.

Case 2b: Insert an entry into each table looking out for foreign key errors.

Case 3c: Update one field in an entry in each table.

Case 3: Web application user input handling.

Case 3a: Try an incorrect combination for the username/password

Case 3b: Click on manager and log in with valid credentials for a customer or employee (user: ross.geller, password: 12345six) this should fail since customer/employees can't log in to the manager login page. Try the same with manager login credentials to the user login, this should also fail.

Case 3c: When logged in as a manager try to edit the price of a service by inputting something other than numbers, example: "hello" and an error should appear.

Case 3d: Make a booking for the same dog to go to boarding at the same time.

Case 4: Testing the manager portal: Click on manager and log in as a manager (user: fish.sauce, password: manager).

Case 4a: Navigate to the services tab and edit a service

Case 4b: Navigate to the services tab and add a new service

Case 4c: Navigate to the employees tab and add a new employee

Case 4d: Navigate to the schedules tab and create a new schedule.

Case 4e: Click 'sign out' to sign out of the manager account.

Case 5: Testing the user portal logged in as a customer: Click on user and log with a valid employee account (user: s.gaulden, password: gaulden).

Case 5a: Navigate to the services tab to view all services.

Case 5b: Navigate to the schedule tab to view all schedules

Case 5c: Navigate to the schedule tab and select an employee from the drop down menu to see only that employee's schedule.

Case 5d: Click 'sign out' to sign out of the employee account.

Case 6: Testing the user portal logged in as an employee: Click on user and log in with a valid customer account (user: ross.geller, password: 12345six).

Case 6a: Navigate to the dogs tab and add a new dog.

Case 6b: Navigate to the calendar tab and click 'book' under an available time slot to create a booking, fill out required information and complete the booking. Confirm this new booking shows up in the bookings tab.

Case 6c: Navigate to bookings tab and cancel a booking. Confirm it is gone from the bookings tab and that slot is now available in the calendar.

Case 6d: Navigate to the dogs tab and delete a dog the customer has a booking reserved for. Ensure the booking is removed from the bookings tab and that it returns to the calendar tab.

Case 7: Deployment

Case 7a: Test the web application can be opened on a different computer.

Case 7b: Test the web application can be accessed through Chrome browser.

VII. CONCLUSION

We have successfully designed a database and provided a system for customers and employees at PAWsitive Dog Care. We have achieved what we would call our first release version of this database. There are still some critical features that need to be included in our web application that might need to be added as patches. For example, a manager cannot currently delete an employee or update their information through the web application; they can only add new employees. We would also like to separate the past and upcoming reservations, perhaps a new tab in the web application that shows past bookings while the reservation tab shows only upcoming bookings. The same thing would need to be added for employee schedules.

This process has shown how much work goes into the database design life cycle. We would expect PAWsitive Dog Care company to have more changes in the future that could require a more robust database system. These changes would require different release versions of our database. For example, the next release version could allow the company to open group training sessions. This might not seem like something difficult to implement since our database can already handle multiple bookings, but because the service is selected at the time of booking as it is currently set up it would allow for multiple customers to book different training classes at the same time. We would also like to add a cancellation charge or a way to apply a discount to specific customers. We also think a company like this would like to be able to store more information on the financial side of their business and allow customers to pay online. We focussed more on the database being functional for bookings which was already complex in itself.

We've tested all of the cases, but if customers or employees have trouble using our system due to a new error, the existing system must be fixed immediately with a patch or update. When PAWsitive Dog Care grows and has more and more customers, storage must be added and space must be monitored. The business can decide when they might need more space and can choose to upgrade their Google Cloud Platform account. In conclusion, PAWsitive Dog Daycare has been set up with a working database and web application that will allow their business to keep track of important records and function day to day. Their minimum requirements for a database have been met.

APPENDIX A

Important Links

Link 1: https://doggy-frontend-shayna-gaulden.vercel.app/login/user

A link to the web application. Below is login information for several different types of test accounts.

User type	Username	Password	URL to login
Manager	fish.sauce	manager	https://doggy-frontend-shayna-gaulden.vercel.app/login/manager
Employee	t.nguyen	nguyen	https://doggy-frontend-shayna-gaulden.vercel.app/login/user
Customer	ross.geller	12345six	https://doggy-frontend-shayna-gaulden.vercel.app/login/user

Link 2: https://github.com/shayna-gaulden/pawsitive

Link to the github repository.

APPENDIX B

MySQL code

```
DROP TABLE IF EXISTS BOOKINGS;
DROP TABLE IF EXISTS DOGS;
DROP TABLE IF EXISTS CUSTOMERS;
DROP TABLE IF EXISTS SCHEDULES;
DROP TABLE IF EXISTS SERVICES;
DROP TABLE IF EXISTS EMPLOYEES;
DROP TABLE IF EXISTS DEPARTMENTS;
DROP TABLE IF EXISTS USERS;
DROP VIEW IF EXISTS EMPLOYEES_VW ;
DROP VIEW IF EXISTS CALENDAR_VW;
DROP VIEW IF EXISTS SCHEDULES_ALL_VW;
DROP VIEW IF EXISTS TOTAL_PRICE_VW;
```

Fig B.1. The order in which the tables need to be dropped due to their foreign key constraints

```
CREATE TABLE CUSTOMERS (
    USER_ID VARCHAR(50),
    CUST_NAME VARCHAR(40) NOT NULL,
    PHONE_NUMBER VARCHAR(20) NOT NULL,
    ADDRESS TEXT NOT NULL,
    CITY VARCHAR(20) NOT NULL,
    STATE VARCHAR(20) NOT NULL,
    ZIPCODE VARCHAR(10) NOT NULL,
    COUNTRY VARCHAR(20) NOT NULL,
    CONSTRAINT CUSTOMER_PK_USER_ID PRIMARY KEY (USER_ID)
);
```

Fig B.2. Create table SQL code for the CUSTOMERS table.

```
CREATE TABLE DOGS (
    DOG_ID INT NOT NULL AUTO_INCREMENT,
    USER_ID VARCHAR(50) NOT NULL,
    DOG_NAME VARCHAR(20) NOT NULL,
    SIZE VARCHAR(20) NOT NULL,
    BREED VARCHAR(20) NOT NULL,
    SPAYED_NEUTERED INT NOT NULL,
    VACCINATED INT NOT NULL DEFAULT 0,
    AGE INT NOT NULL,
    GENDER VARCHAR(1) NOT NULL,
    CONSTRAINT DOG_PK_DOG_ID PRIMARY KEY (DOG_ID)
);
```

Fig B.3. Create table SQL code for the DOGS table.

```
CREATE TABLE EMPLOYEES (
    USER_ID VARCHAR(50),
    DEPARTMENT VARCHAR(20) NOT NULL,
    EMP_NAME VARCHAR(40) NOT NULL,
    ADDRESS TEXT NOT NULL,
    CITY VARCHAR(20) NOT NULL,
    STATE VARCHAR(20) NOT NULL,
    ZIPCODE VARCHAR(10) NOT NULL,
    COUNTRY VARCHAR(20) NOT NULL,
    SOCIAL_SECURITY_NUMBER INT(9) UNIQUE,
    CONSTRAINT EMPLOYEE_PK_USER_ID PRIMARY KEY (USER_ID)
);
```

Fig B.4. Create table SQL code for the EMPLOYEES table.

```
CREATE TABLE SCHEDULES (
    USER_ID VARCHAR(50),
    TIME_DATE DATETIME NOT NULL,
    SUB_ID INT NOT NULL,
    CONSTRAINT SCHEDULES_PK_EMP_ID_TIME_DATE PRIMARY KEY (USER_ID, TIME_DATE, SUB_ID)
);
```

Fig B.5. Create table SQL code for the SCHEDULES table.

```sql
CREATE TABLE SERVICES (
    SERVICE VARCHAR(100),
    DETAILS TEXT,
    PRICE DECIMAL(10,2) NOT NULL,
    DEPARTMENT VARCHAR(20) NOT NULL,
    CONSTRAINT SERVICES_PK_SERVICE PRIMARY KEY (SERVICE)
);
```

Fig B.6. Create table SQL code for the SERVICES table.

```sql
CREATE TABLE BOOKINGS (
    DOG_ID INT,
    USER_ID VARCHAR(50),
    TIME_DATE DATETIME,
    SERVICE VARCHAR(40) NOT NULL,
    SUB_ID INT NOT NULL,
    CONSTRAINT BOOKINGS_PK_DOG_ID_EMP_ID_TIME_DATE PRIMARY KEY (DOG_ID, USER_ID, TIME_DATE)
);
```

Fig B.7. Create table SQL code for the BOOKINGS table.

```sql
CREATE TABLE DEPARTMENTS (
    DEPARTMENT VARCHAR(20) NOT NULL,
    DETAILS TEXT,
    CONSTRAINT DEPARTMENT_PK_DEPATMENT PRIMARY KEY (DEPARTMENT)
);
```

Fig B.8. Create table SQL code for the DEPARTMENTS table.

```sql
CREATE TABLE USERS (
    USER_ID VARCHAR(50),
    PW VARCHAR(30) NOT NULL,
    USER_TYPE VARCHAR(30) NOT NULL,
    CONSTRAINT USERS_PK_USER_ID PRIMARY KEY (USER_ID)
);
```

Fig B.9. Create table SQL code for the USERS table.

```
-- Add foriegn keys
ALTER TABLE CUSTOMERS ADD CONSTRAINT CUSTOMER_FK_USER_ID FOREIGN KEY (USER_ID)
    REFERENCES USERS (USER_ID) ON DELETE CASCADE;
ALTER TABLE DOGS ADD CONSTRAINT DOGS_FK_USER_ID FOREIGN KEY (USER_ID)
    REFERENCES CUSTOMERS (USER_ID) ON DELETE CASCADE;
ALTER TABLE SERVICES ADD CONSTRAINT SERVICES_FK_DEPARTMENT FOREIGN KEY (DEPARTMENT)
    REFERENCES DEPARTMENTS (DEPARTMENT) ON DELETE CASCADE;
ALTER TABLE BOOKINGS ADD CONSTRAINT BOOKINGS_FK_DOG_ID FOREIGN KEY (DOG_ID)
    REFERENCES DOGS (DOG_ID) ON DELETE CASCADE;
ALTER TABLE BOOKINGS ADD CONSTRAINT BOOKINGS_FK_SERVICE FOREIGN KEY (SERVICE)
    REFERENCES SERVICES (SERVICE) ON DELETE CASCADE;
ALTER TABLE BOOKINGS ADD CONSTRAINT BOOKINGS_FK_USER_ID_TIME_DATE_SUB_ID FOREIGN KEY (USER_ID, TIME_DATE,SUB_ID)
    REFERENCES SCHEDULES (USER_ID, TIME_DATE, SUB_ID) ON DELETE CASCADE;
ALTER TABLE SCHEDULES ADD CONSTRAINT SCHEDULES_FK_USER_ID FOREIGN KEY (USER_ID)
    REFERENCES EMPLOYEES (USER_ID) ON DELETE CASCADE;
ALTER TABLE EMPLOYEES ADD CONSTRAINT EMPLOYEES_FK_DEPARTMENT FOREIGN KEY (DEPARTMENT)
    REFERENCES DEPARTMENTS (DEPARTMENT) ON DELETE CASCADE;
ALTER TABLE EMPLOYEES ADD CONSTRAINT EMPLOYEES_FK_USER_ID FOREIGN KEY (USER_ID)
    REFERENCES USERS (USER_ID) ON DELETE CASCADE;
```

Fig B.10. Add the foreign constraints.

```
CREATE VIEW EMPLOYEES_VW AS (
    SELECT DEPARTMENT, EMP_NAME, ADDRESS, CITY, STATE, ZIPCODE, COUNTRY
    FROM EMPLOYEES
);
```

Fig B.11. Create a view of employees.

```
CREATE VIEW CALENDAR_VW AS(
    SELECT E.USER_ID, E.DEPARTMENT, S1.TIME_DATE , E.EMP_NAME, S1.SUB_ID
    FROM SCHEDULES AS S1 JOIN EMPLOYEES AS E ON S1.USER_ID = E.USER_ID
                         JOIN DEPARTMENTS AS D ON D.DEPARTMENT = E.DEPARTMENT
                         LEFT JOIN BOOKINGS AS B
                            ON S1.USER_ID = B.USER_ID AND S1.TIME_DATE = B.TIME_DATE AND B.SUB_ID = S1.SUB_ID
    WHERE B.USER_ID IS NULL AND (E.DEPARTMENT='Grooming' OR E.DEPARTMENT='Training' OR E.DEPARTMENT='Boarding')
);
```

Fig B.12. Create a view of available time slots for customers.

```
CREATE VIEW SCHEDULES_ALL_VW AS (
    SELECT E.DEPARTMENT, E.EMP_NAME, B.SERVICE, S.TIME_DATE, C.CUST_NAME, D.DOG_NAME, C.USER_ID, E.USER_ID AS EMP_ID
    FROM SCHEDULES AS S
    LEFT JOIN BOOKINGS AS B ON (B.USER_ID = S.USER_ID AND B.TIME_DATE = S.TIME_DATE AND B.SUB_ID = S.SUB_ID)
    JOIN EMPLOYEES AS E ON E.USER_ID = S.USER_ID
    LEFT JOIN DOGS AS D ON D.DOG_ID = B.DOG_ID
    LEFT JOIN CUSTOMERS AS C ON C.USER_ID = D.USER_ID
);
```

Fig B.13. Create a view of schedules with booking information for employees.

```
CREATE VIEW TOTAL_PRICE_VW AS(
    SELECT Y.USER_ID, Y.DOG_NAME, S.SERVICE, B.TIME_DATE,S.PRICE, Y.TOTAL
    FROM BOOKINGS AS B
        JOIN
            (SELECT USER_ID,DOG_ID,DOG_NAME, SUM(PRICE) AS TOTAL
            FROM (SELECT  D.USER_ID,D.DOG_ID, D.DOG_NAME, B.TIME_DATE, S.SERVICE, S.PRICE
            FROM BOOKINGS AS B JOIN SERVICES AS S ON B.SERVICE = S.SERVICE
                            JOIN DOGS     AS D ON D.DOG_ID  = B.DOG_ID) AS X
            GROUP BY 1,2) AS Y
        ON B.DOG_ID = Y.DOG_ID
        JOIN
            SERVICES AS S ON B.SERVICE = S.SERVICE
    ORDER BY Y.DOG_NAME
);
```

Fig B.14. Create a view that calculates the total price for customers upcoming bookings.

APPENDIX C

Test Cases

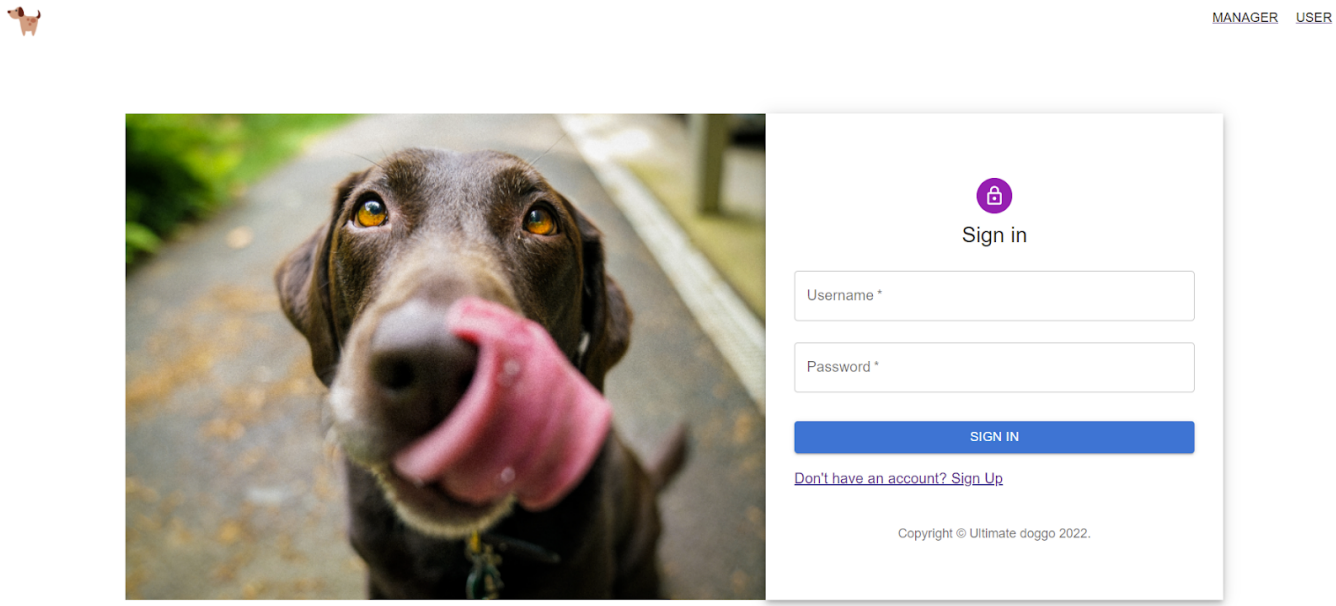
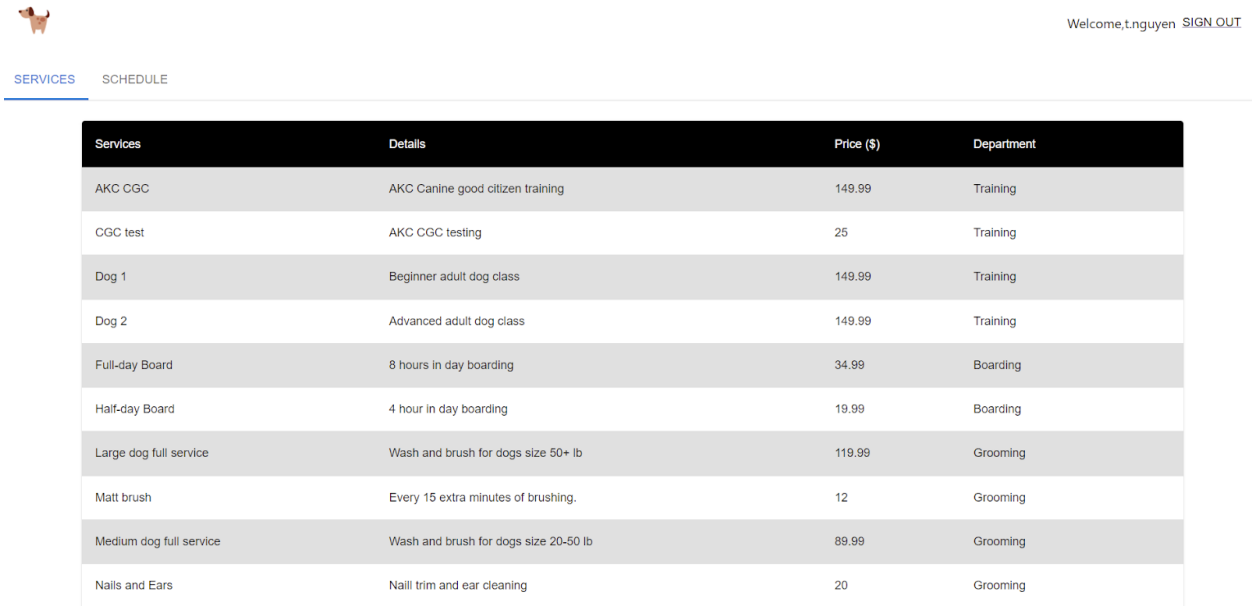


Fig C.1. User login page.



Services	Details	Price ($)	Department
AKC CGC	AKC Canine good citizen training	149.99	Training
CGC test	AKC CGC testing	25	Training
Dog 1	Beginner adult dog class	149.99	Training
Dog 2	Advanced adult dog class	149.99	Training
Full-day Board	8 hours in day boarding	34.99	Boarding
Half-day Board	4 hour in day boarding	19.99	Boarding
Large dog full service	Wash and brush for dogs size 50+ lb	119.99	Grooming
Matt brush	Every 15 extra minutes of brushing.	12	Grooming
Medium dog full service	Wash and brush for dogs size 20-50 lb	89.99	Grooming
Nails and Ears	Naill trim and ear cleaning	20	Grooming

Fig C.2. Employee landing page after login; the services tab.

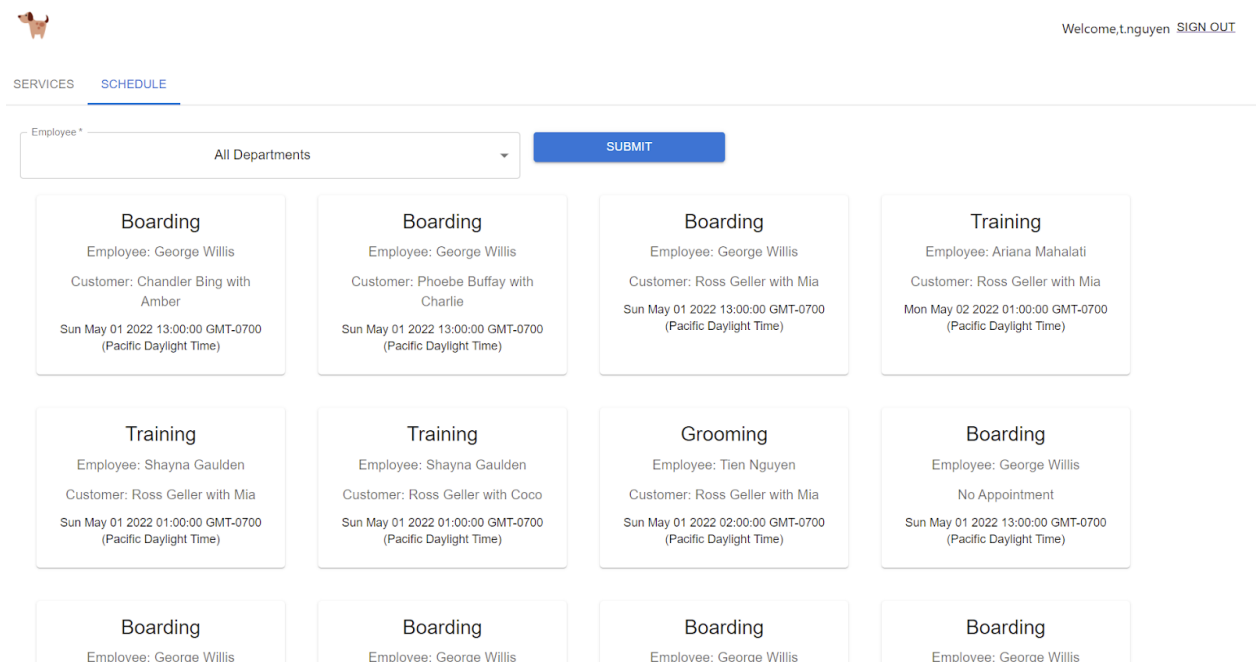


Fig C.3. Schedule tab while logged in as an employee.

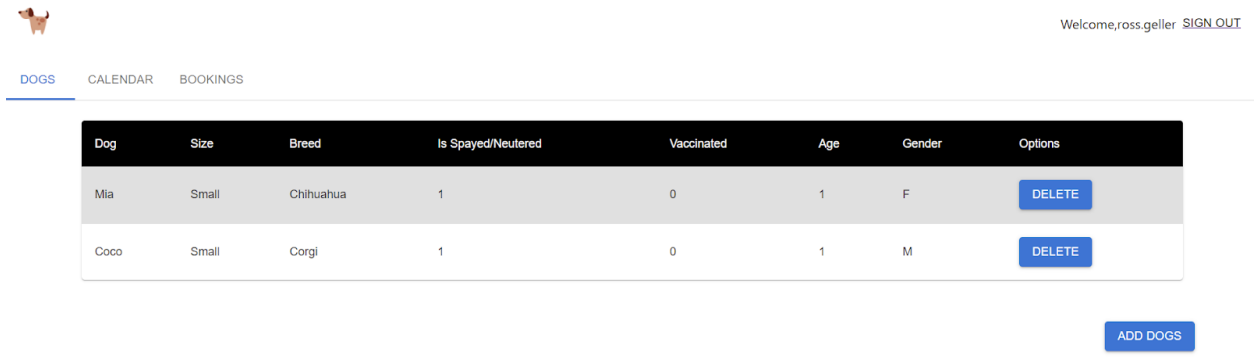


Fig C.4. Customer landing page after login; the dogs tab.

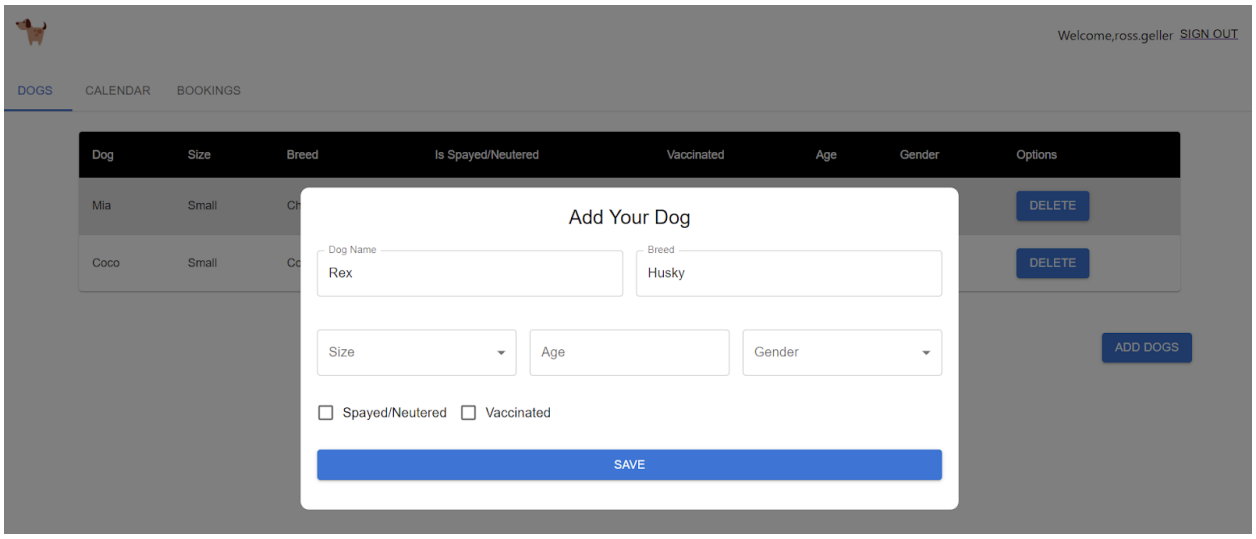


Fig C.5. Add a dog form while logged in as a customer

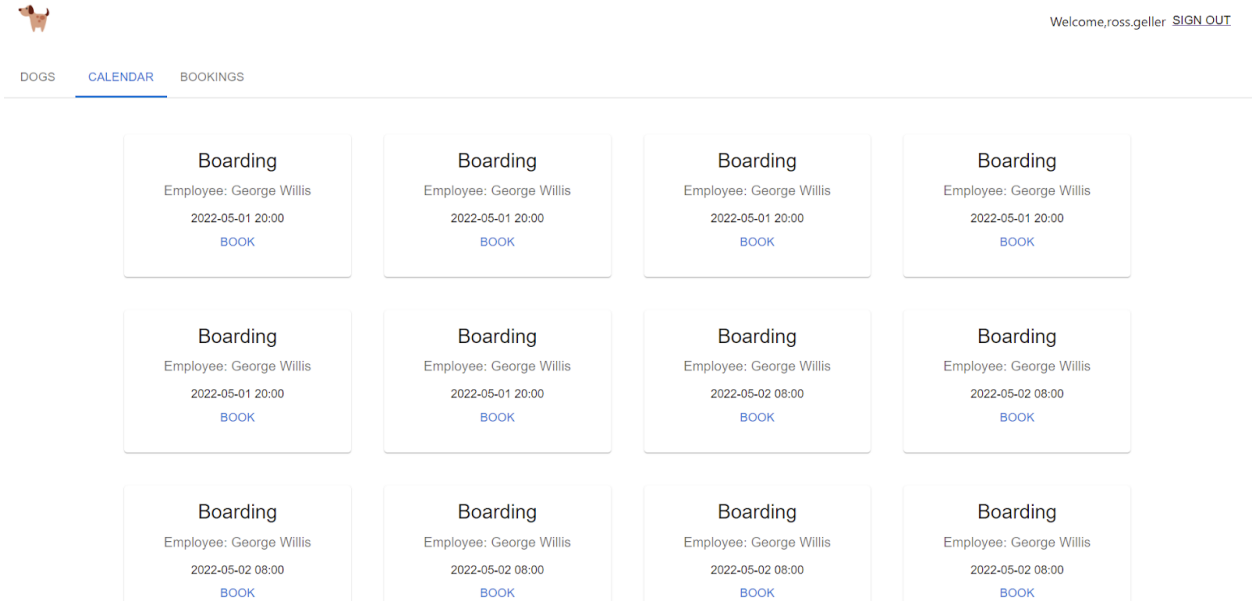


Fig C.6. Calendar tab while logged in as a customer.

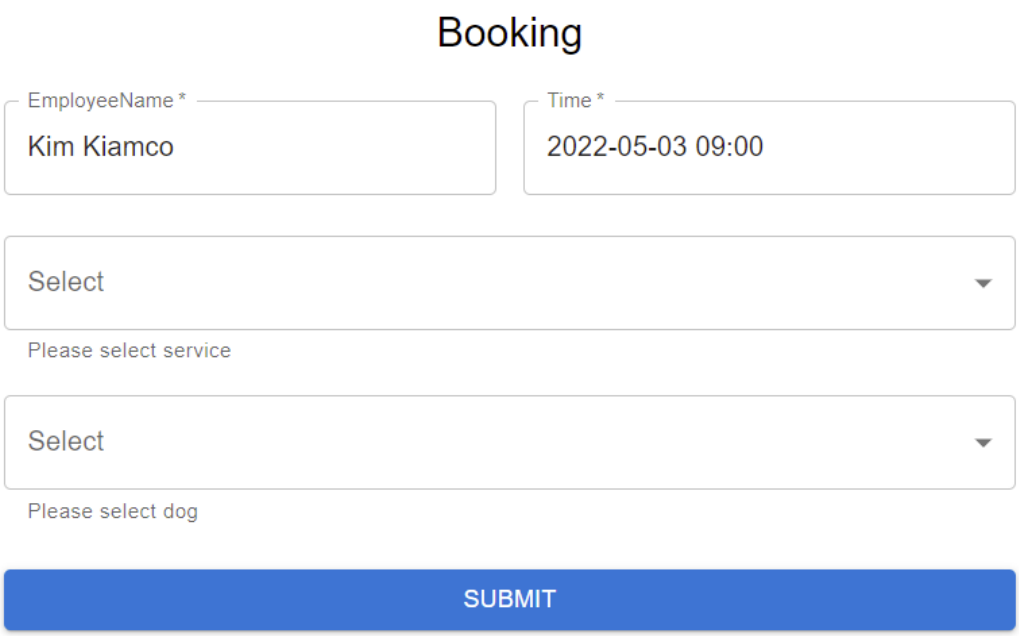


Fig C.7. Reserve a booking form while logged in as a customer.

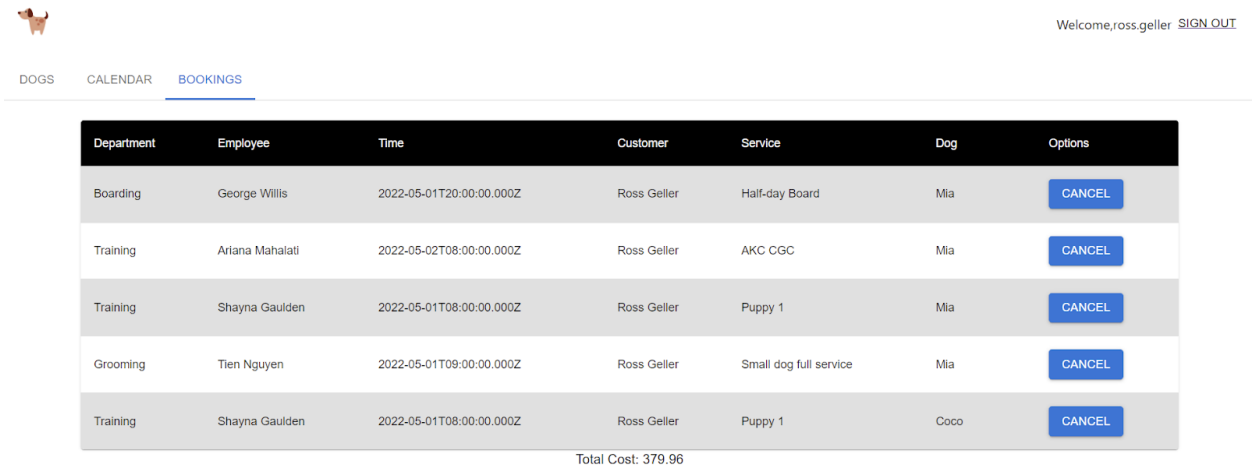


Fig C.8. Bookings tab while logged in as a customer.

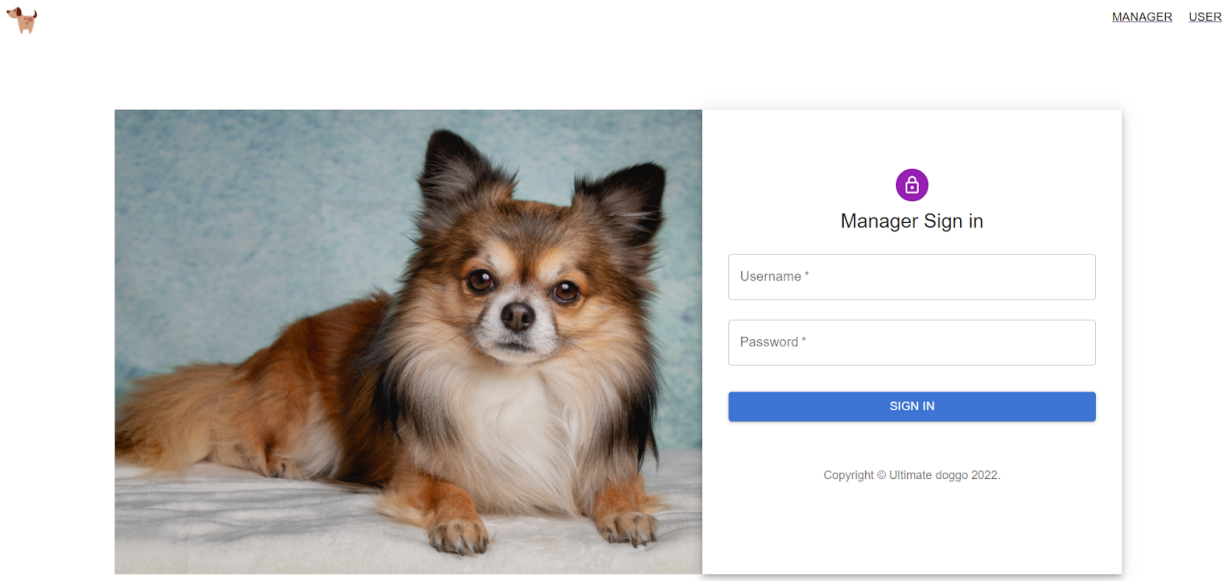


Fig C.9. Manager login page.

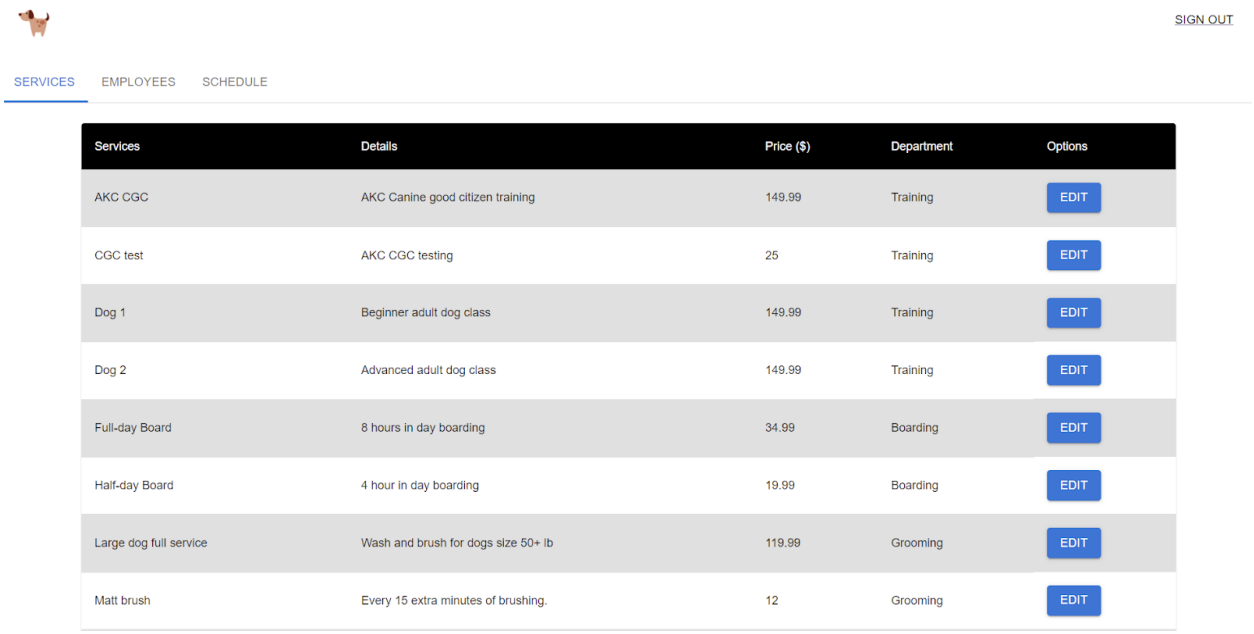


Fig C.10. Manager landing page after login; the services tab.

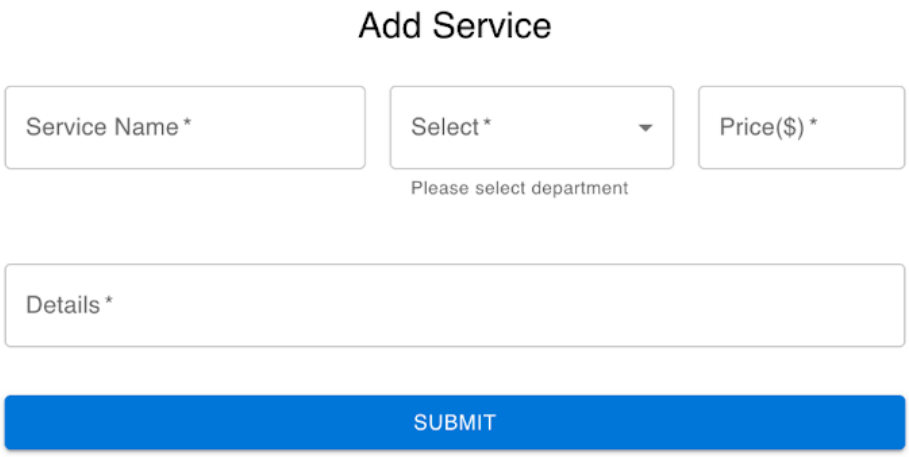


Fig C.11. Add a service form while logged in as a manager.

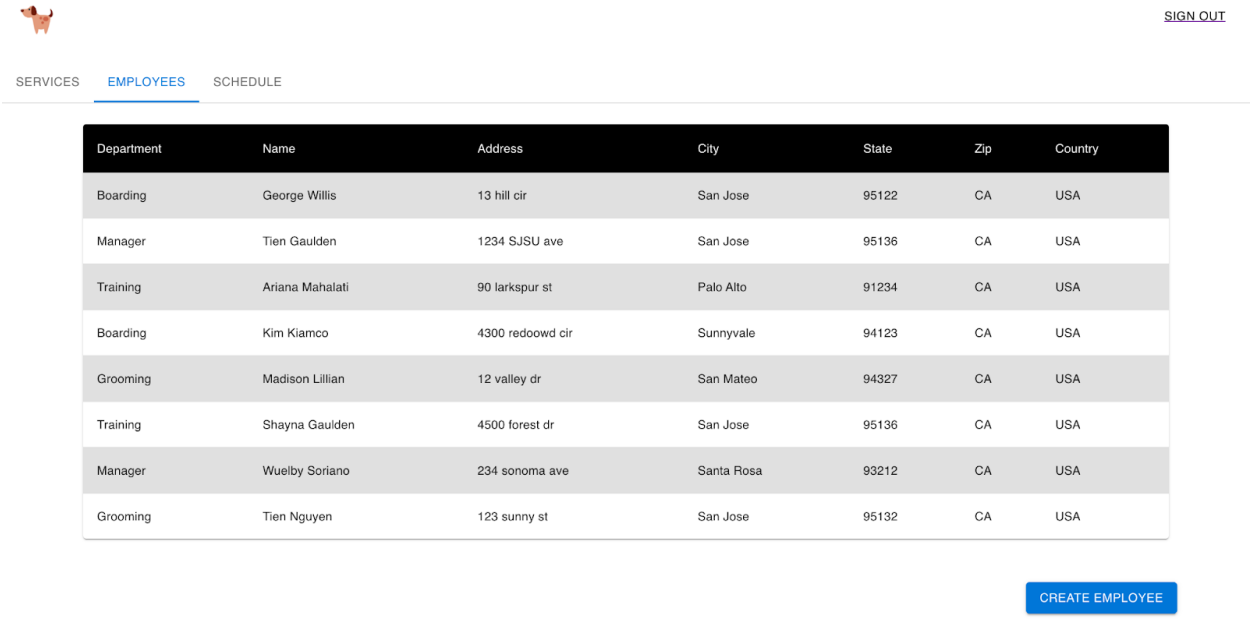


Fig C.12. Employees tab while logged in as a manager.

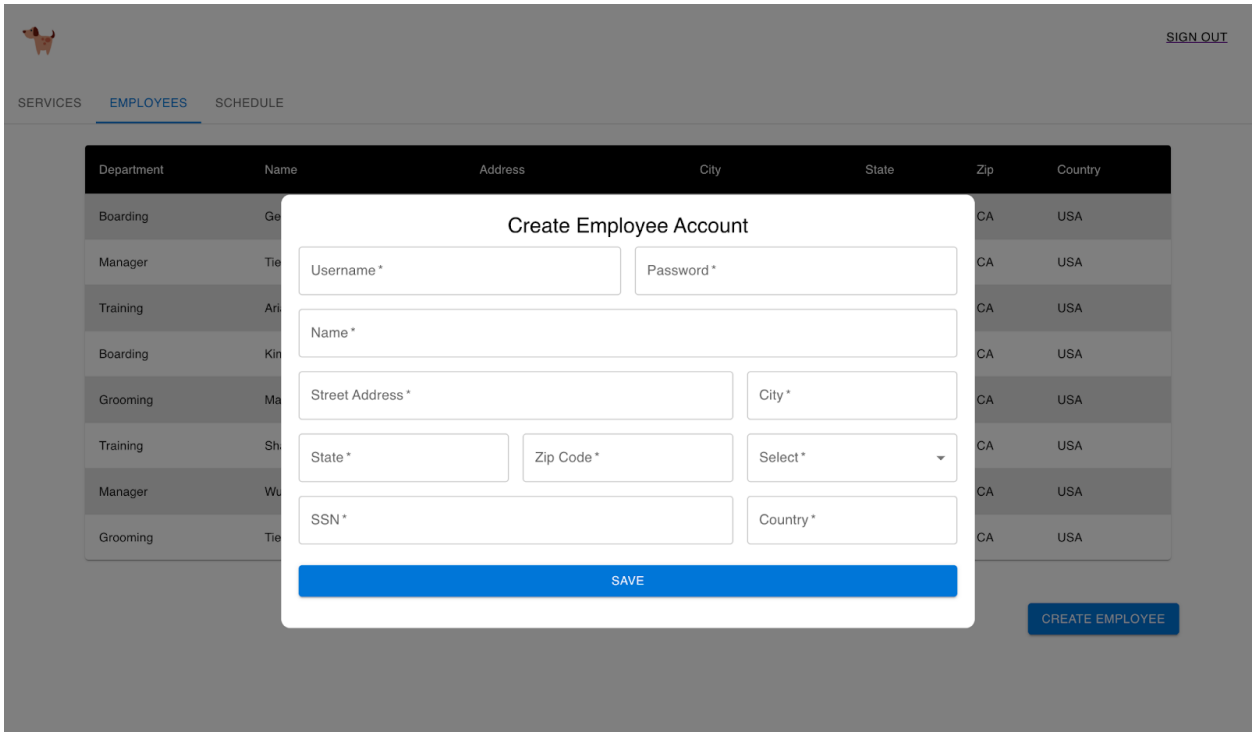


Fig C.13. Add an employee form while logged in as a manager.

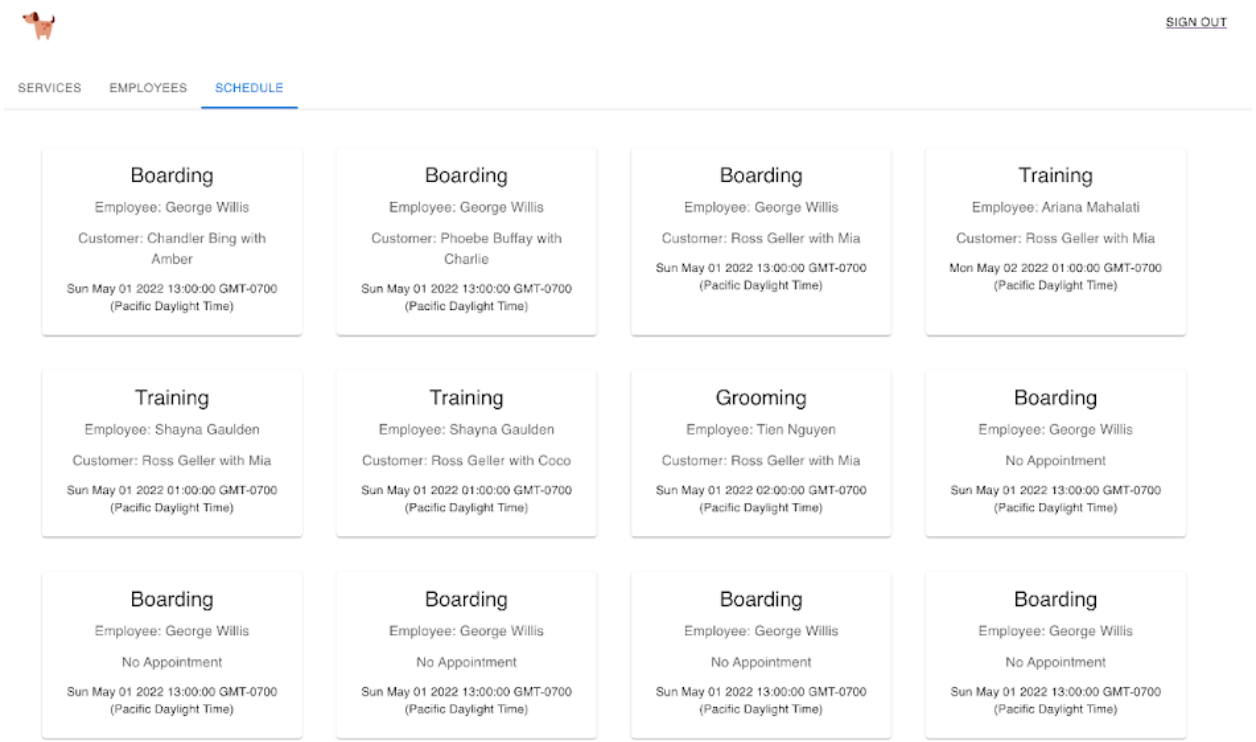


Fig C.14. Schedules tab while logged in as a manager.

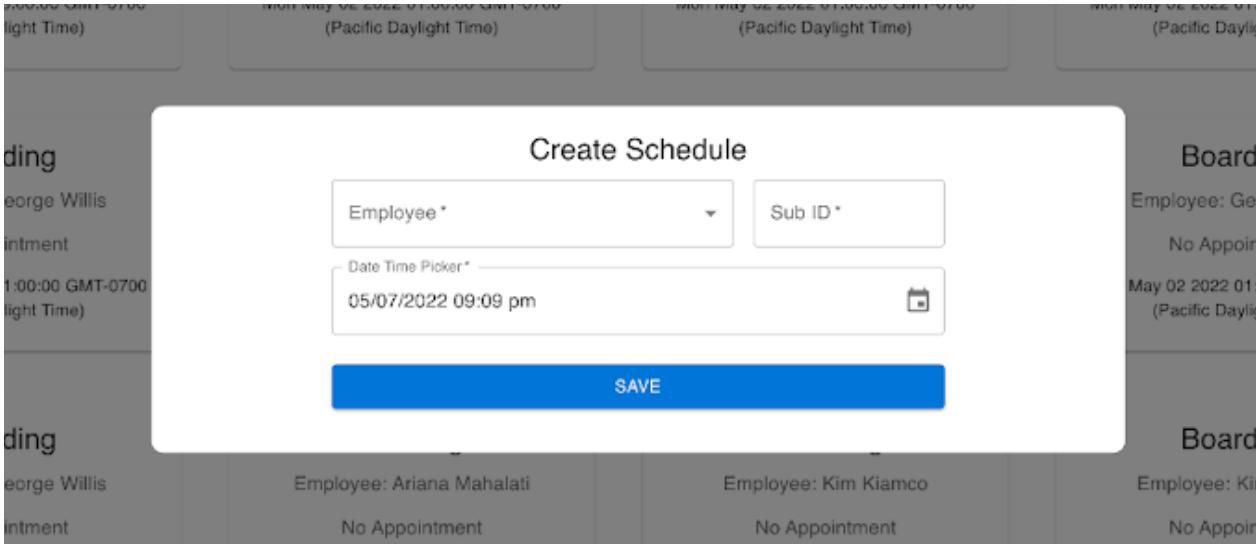


Fig C.15. Create a schedule form while logged in as a manager.

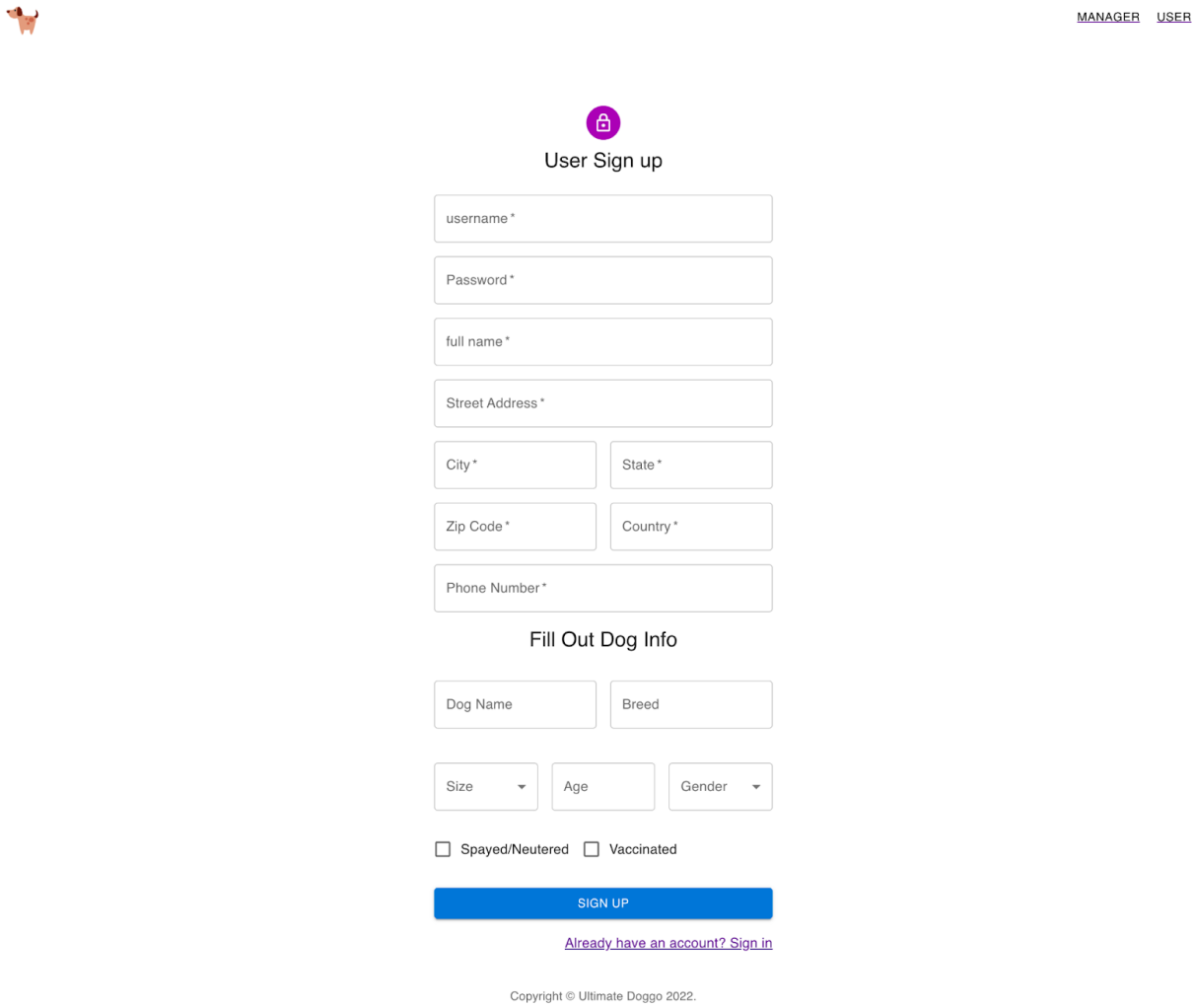


Fig C.16. Sign up as a customer form.

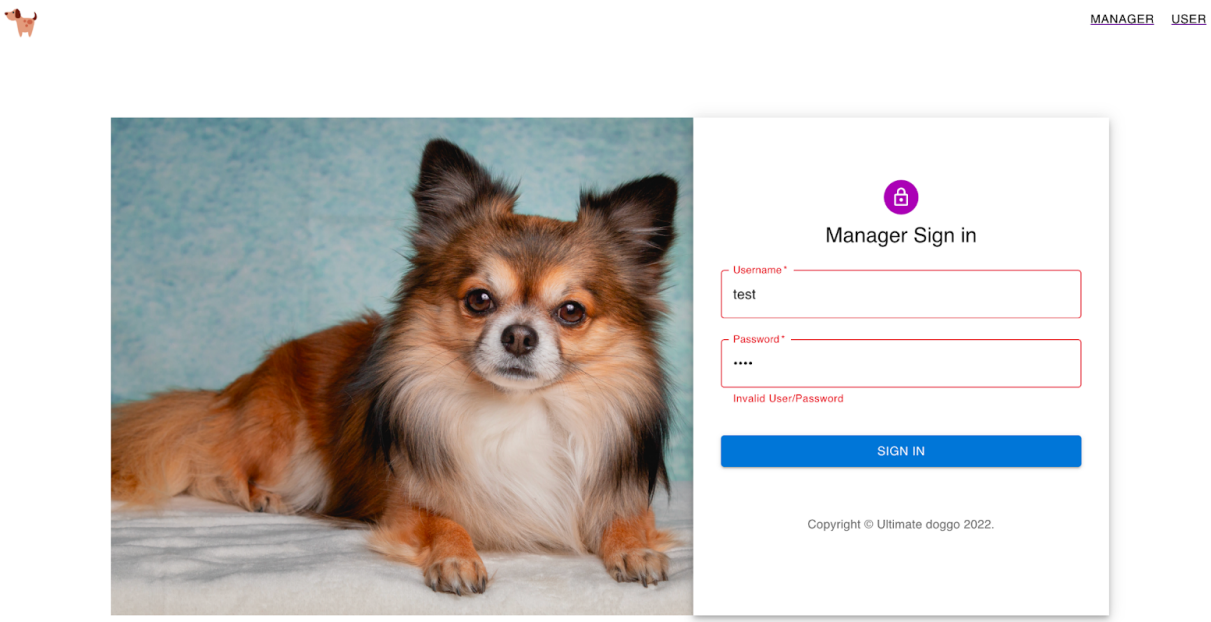


Fig C.17. User input wrong credentials bad login.

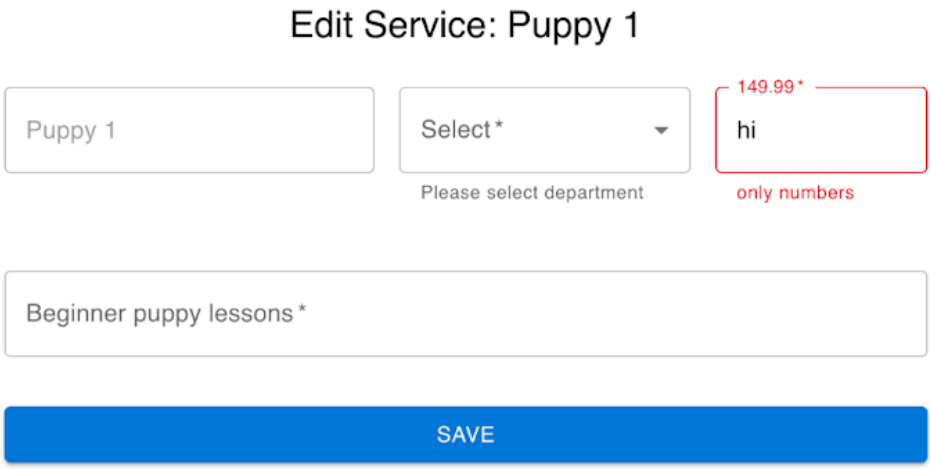


Fig C.16. User input invalid price while editing a service.