PawsPlan Web Application

Software Requirements Specification

Final Report

Alex Feldman, Andreas Georgoulakis

Table of Contents

1 Overview

1.1 Team Members

The team working on this project consists of the following people:

- Alex Feldman
- Andreas Georgoulakis

1.2 Current Application

It is easy for pet owners to become disorganized and lose track of everything that goes on in the Veterinarian process. PawsPlan's goal is to fix this and give pet owners a tool to help them keep track of their Veterinarian activities and pets needs.

1.2.1 Issues

Through interviews with pet owners, common problems regarding current veterinarian processes have been identified and are described below.

- Health records, vaccination details, and medication schedules are often stored in different places, making it hard to keep track
- Reliance on paper documents which can be lost, damaged, or hard to access when needed
- Pet owners often have to manually track vaccination dates and medication schedules, leading to missed doses or appointments
- Lack of automated alerts for upcoming vaccinations, check-ups, or medication refills
- Difficulty in keeping a consistent log of symptoms or behavioral changes that can help in diagnosing health issues
- Owners might forget to mention certain symptoms or changes to the vet, leading to incomplete health assessments
- Vital information needed during emergencies can be hard to gather quickly

- Without detailed and accessible health histories, vets may not be able to provide highly personalized care plans
- Physical records are susceptible to loss or damage, resulting in incomplete medical histories

Note: There are many other issues we have identified, but we feel these best represent the problems people are having

1.3 Project Name

The name of this application is PawsPlan. For this document, PawsPlan will commonly be referred to as “this application”.

2 Background

PawsPlan is a web-based application that is designed to help pet owners stay organized in managing their pet care. The application will do this by creating a user friendly environment in which anyone can easily evaluate their pets health and needs. PawsPlan will allow users to keep track of scheduled appointments, vaccinations, feeding schedule, and expense management. PawsPlan’s top priority is to keep pet owners informed, ensuring the safety of their pets and allowing them to focus on building strong connections with their furry friends.

2.1 Intended Use

This application is designed for use by pet owners. All features and functionalities in this application will be tailored to their needs.

2.2 User Roles

This application will consist of one user role.

2.2.1 Pet Owner

This will be the only role within this application. This user will have complete access to all of the app’s features and functionalities.

Users with the Pet Owner role will be able to do all of the following:

- Create a profile
- Login
- Track scheduled veterinarian appointments
- Track vaccinations
- Manage feeding schedules
- Manage expenses
- Change their profile information
- Add a new pet to the system

2.3 Project Functions

This application will be considered minimally viable if and only if the application includes all of the following features below.

2.3.1 Profile Creation

After selecting “Get Started”, the user will be prompted to input their username, email, and password. The user will also enter their pet name, breed, age, and medical history. After selecting “Create Profile”, the system checks that the username and email aren’t already taken. If they are both unique, then the user will have successfully created a profile. If they aren’t both unique, then the user will be prompted to enter different information.

2.3.2 Login

Next to “Get Started”, there will be an option to login. After selecting “Login”, the user will be prompted to enter their username and password. The user then selects “Sign In”. If the login information provided by the user is incorrect, then the system will prompt the user to enter the correct information. If the username or email field is left blank, then the system will prompt the user to enter their information.

2.3.3 User Settings Management

Users have the ability to manage their profile and pet details, as well as their notification preferences. This includes updating profile information such as

name, email, and password. Users can also add, edit, and delete pet profiles, which include the pet's name, breed, age, and medical history. Additionally, users can configure their notification preferences to receive timely reminders through email or app notifications.

2.3.4 Veterinary Appointment Scheduling

The user can schedule veterinary appointments for their pets. The scheduling feature will allow the user to select a pet, choose a date and time, pick a veterinarian, and add any relevant notes. The user can view, edit, and delete upcoming appointments.

2.3.5 Vaccination Tracking

The application allows users to track their pets' vaccinations. Users can add details for each vaccination, including the vaccine name, date administered, and the next due date. The application provides a history of all vaccinations with status indicators to show if they are up-to-date or overdue.

2.3.6 Feeding Schedule Management

Users can set up feeding schedules for their pets. This feature enables the user to set specific feeding times and meal descriptions for each pet. The user can view, edit, and delete scheduled feeding times.

2.3.7 Expense Tracking

Users can record and manage pet-related expenses. They can categorize expenses, enter amounts, select dates, and add descriptions. The application displays a table of all expenses with options to filter and sort the data. Users can view their expense summary, showing the total amount they've spent on their pets. They can also see their monthly spending details.

2.3.8 Dashboard Overview

The application features a dashboard that provides an overview of all pet care activities. The dashboard displays summaries of upcoming appointments,

recent expenses, and alerts for feeding times and vaccinations, giving users a comprehensive view of their pet care tasks.

3 User Stories

As a pet owner, I want to create an account so that I can manage all my pet care tasks in one place.

As a pet owner, I want to add my pets' profiles so that I can keep track of their specific needs and information.

As a pet owner, I want to schedule veterinary appointments so that I can keep my pets' health checkups organized.

As a pet owner, I want to track my pets' vaccinations so that I never miss an important vaccine.

As a pet owner, I want to set up feeding schedules for my pets so that I can ensure they are fed consistently and on time.

As a pet owner, I want to record my pet-related expenses so that I can manage my budget effectively.

As a pet owner, I want to receive reminders for upcoming appointments, vaccinations, and feeding times so that I don't forget important tasks.

As a pet owner, I want to see an overview of all my pet care activities on the dashboard so that I can easily stay on top of everything.

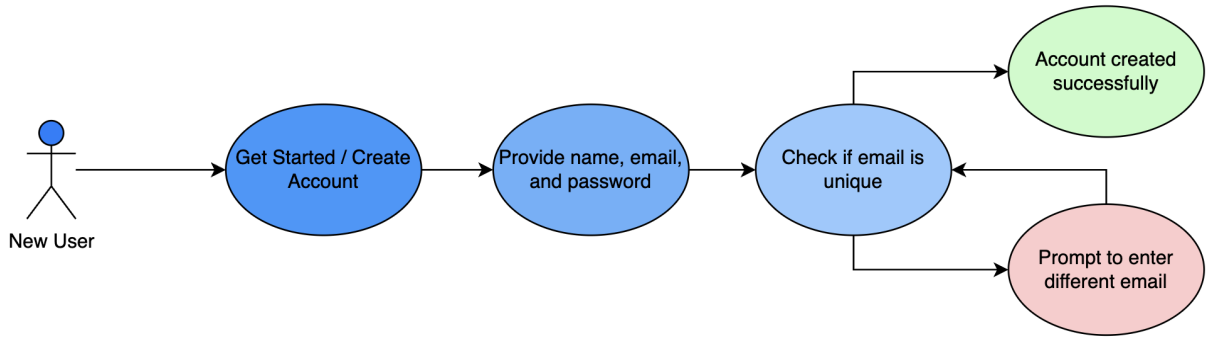
As a pet owner, I want to have a clear view of all scheduled veterinary appointments and feeding times so that I can plan ahead easily.

As a pet owner, I want to manage my user settings and pet information so that I can keep my account and pet details up-to-date.

4 Use Cases

4.1 Profile Creation

4.1.1 Diagram



4.1.2 Use Case

Main Success Scenario:

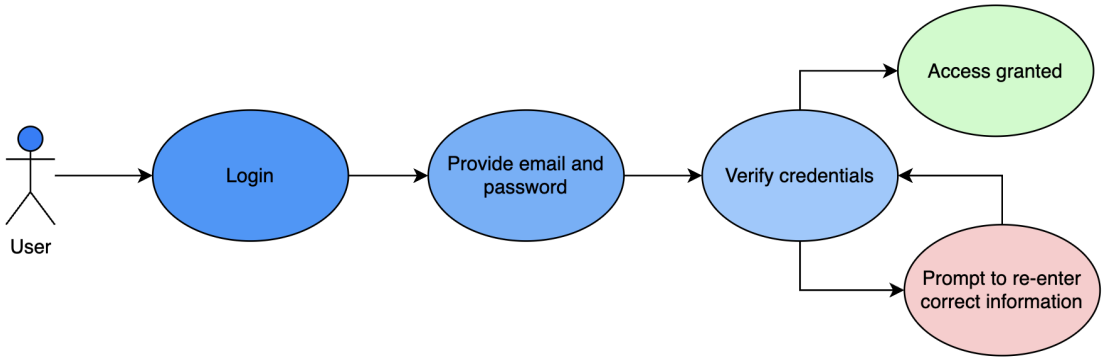
- User selects “Get Started” on the home page to begin customizing their profile
- User provides their name, email, and password
- User provides their pet’s name, breed, age, and medical history
- User selects “Create Profile”
- System checks if the email is unique
- Upon successful validation, the system creates the user profile

Error Scenario:

- If the email already exists, the system prompts the user to provide alternative information.

4.2 Login

4.2.1 Diagram



4.2.2 Use Case

Main Success Scenario:

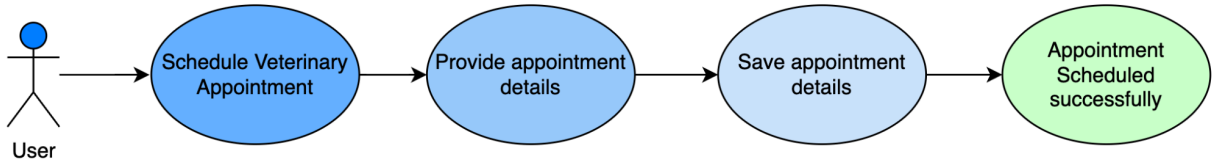
- User selects “Login” on the home page.
- User inputs their email and password.
- User selects “Sign In.”
- System verifies the login information.
- System grants access to the user.

Error Scenarios:

- If the username or password is incorrect, the system prompts the user to re-enter the correct information.
- If the fields are left blank, the system prompts the user to fill in the required information.

4.3 Schedule A Veterinary Appointment

4.3.1 Diagram



4.3.2 Use Case

Main Success Scenario:

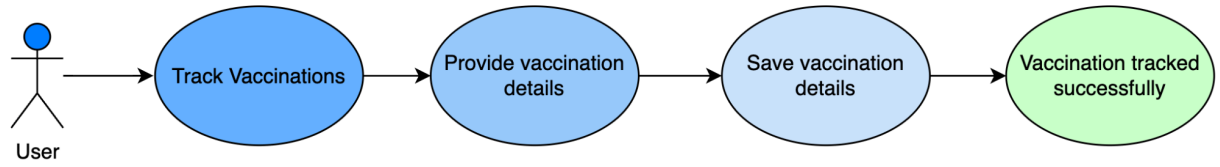
- User selects a pet, date/time, veterinarian, and adds a note
- User selects “Submit”
- System saves the appointment and displays it under “Upcoming Appointments”
- User selects “Edit” within an appointment and changes a detail
- User selects “Delete” within an appointment to get rid of it

Error Scenario:

- If a specification about an appointment is left blank, the system prompts the user to provide the missing information

4.4 Track Vaccinations

4.4.1 Diagram



4.4.2 Use Case

Main Success Scenario:

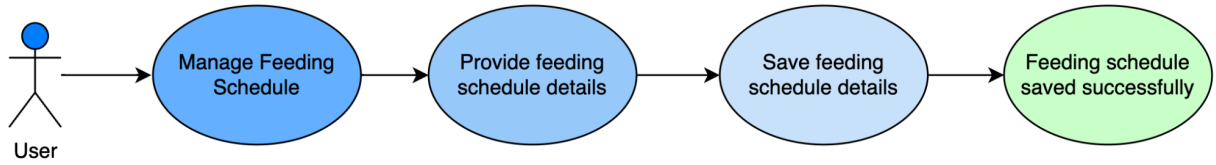
- User selects a pet, vaccine, date administered, and next due date
- User selects “Submit”
- System saves the information and displays it under “Vaccination History”

Error Scenario:

- If a specification about the vaccination schedule is left blank, the system prompts the user to provide the missing information

4.5 Manage Feeding Schedule

4.5.1 Diagram



4.5.2 Use Case

Main Success Scenario:

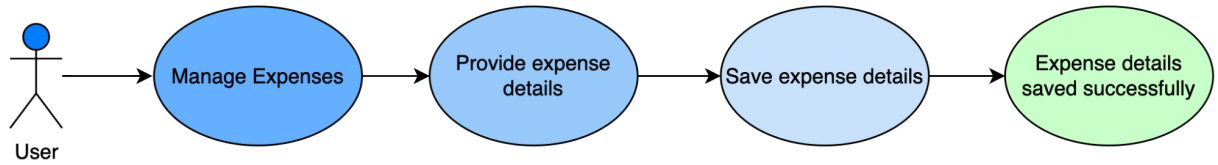
- User selects a pet, feeding time, and adds a meal description
- User selects “Submit”
- System saves the information and displays it under “Feeding Schedule”
- User selects “Edit” within a feeding schedule and changes a detail
- User selects “Delete” within a feeding schedule to get rid of it

Error Scenario:

- If a specification about the feeding schedule is left blank, the system prompts the user to provide the missing information

4.6 Manage Expenses

4.6.1 Diagram



4.6.2 Use Case

Main Success Scenario:

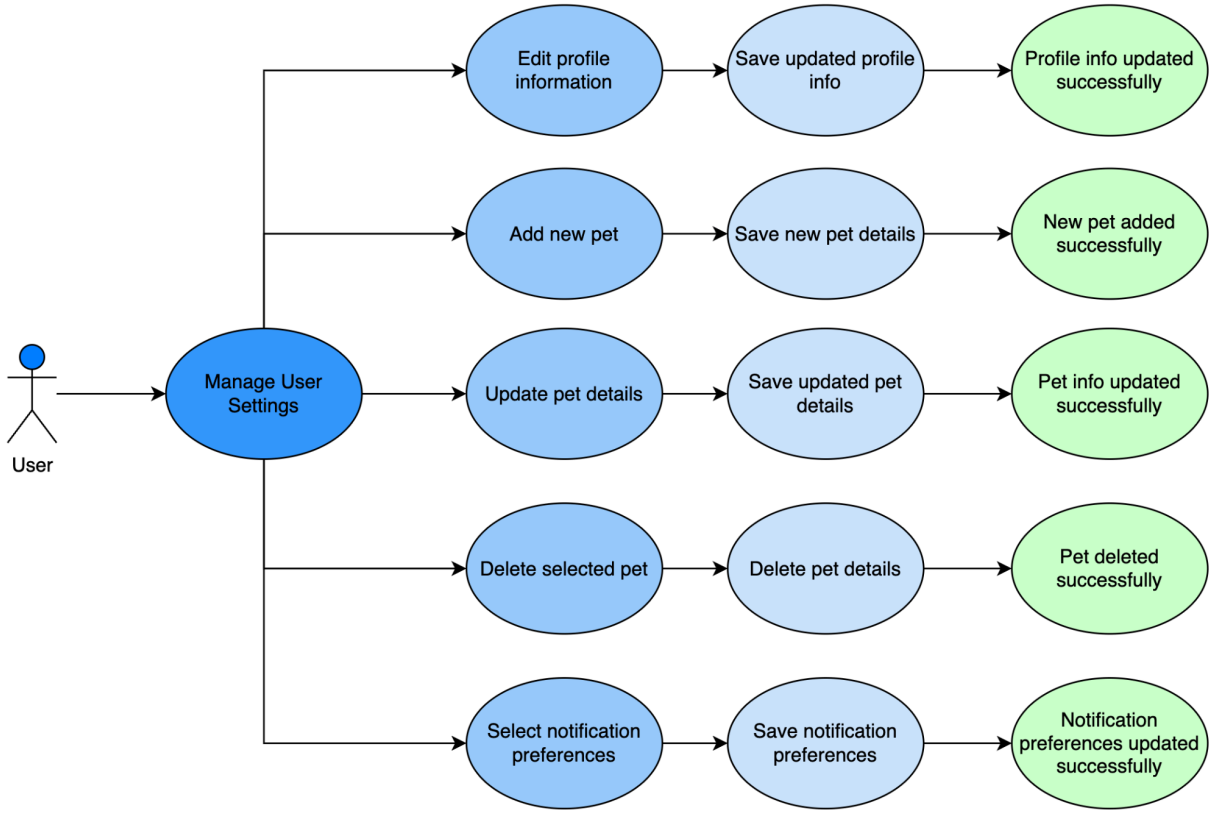
- User selects an expense category, dollar amount, date, and adds a description
- User selects “Submit”
- System saves the information and displays it under “Recorded Expenses”

Error Scenario:

- If a specification about the expense is left blank, the system prompts the user to provide the missing information

4.7 Manage User Settings

4.7.1 Diagram



4.7.2 Use Case

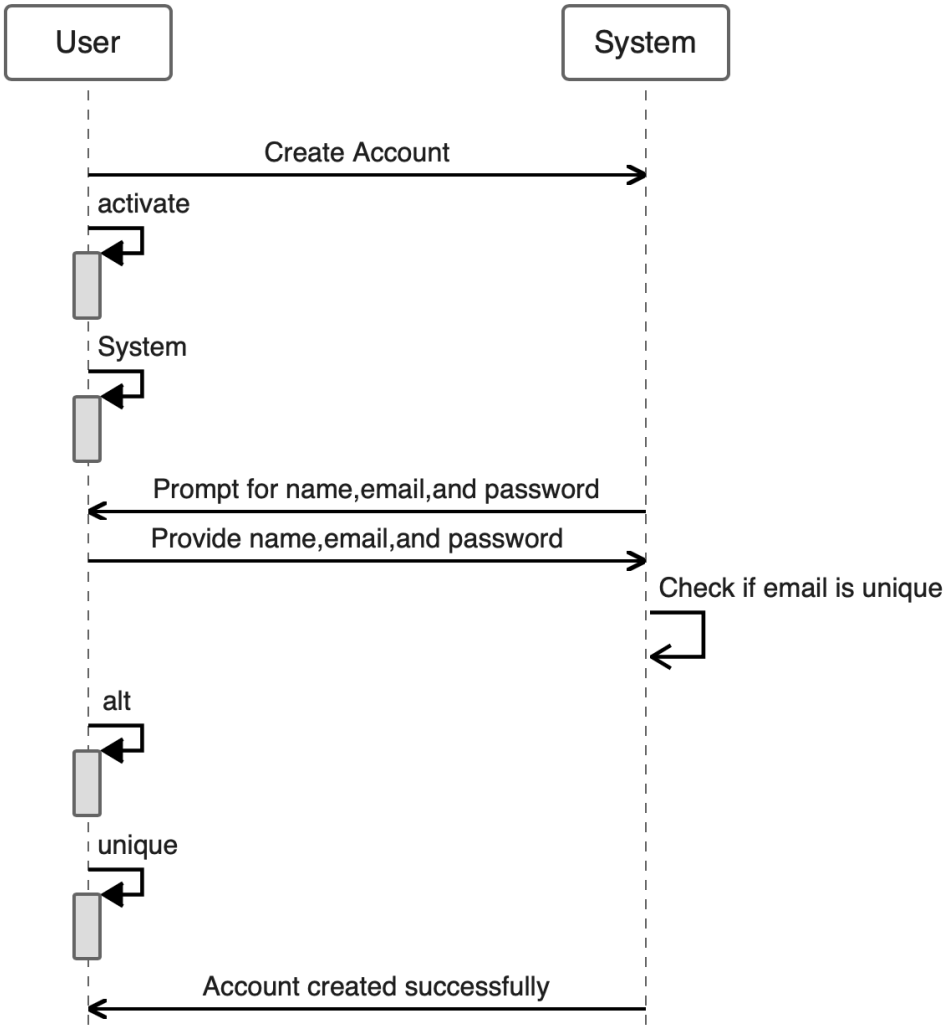
Main Success Scenario:

1. User changes their name, email, and password
2. User selects “Save”
3. System saves the updated information
4. User adds a pet by providing their pet’s name, breed, age, and medical history
5. User selects “Add Pet”
6. System saves the new pet information
7. User turns on notifications by selecting the checkboxes for “Appointment Reminders”, “Vaccination Alerts”, and “Feeding Reminders”
8. User selects “Save”
9. System saves the notification preferences

5 Sequence Diagrams

5.1 Account Creation

5.1.1 Diagram



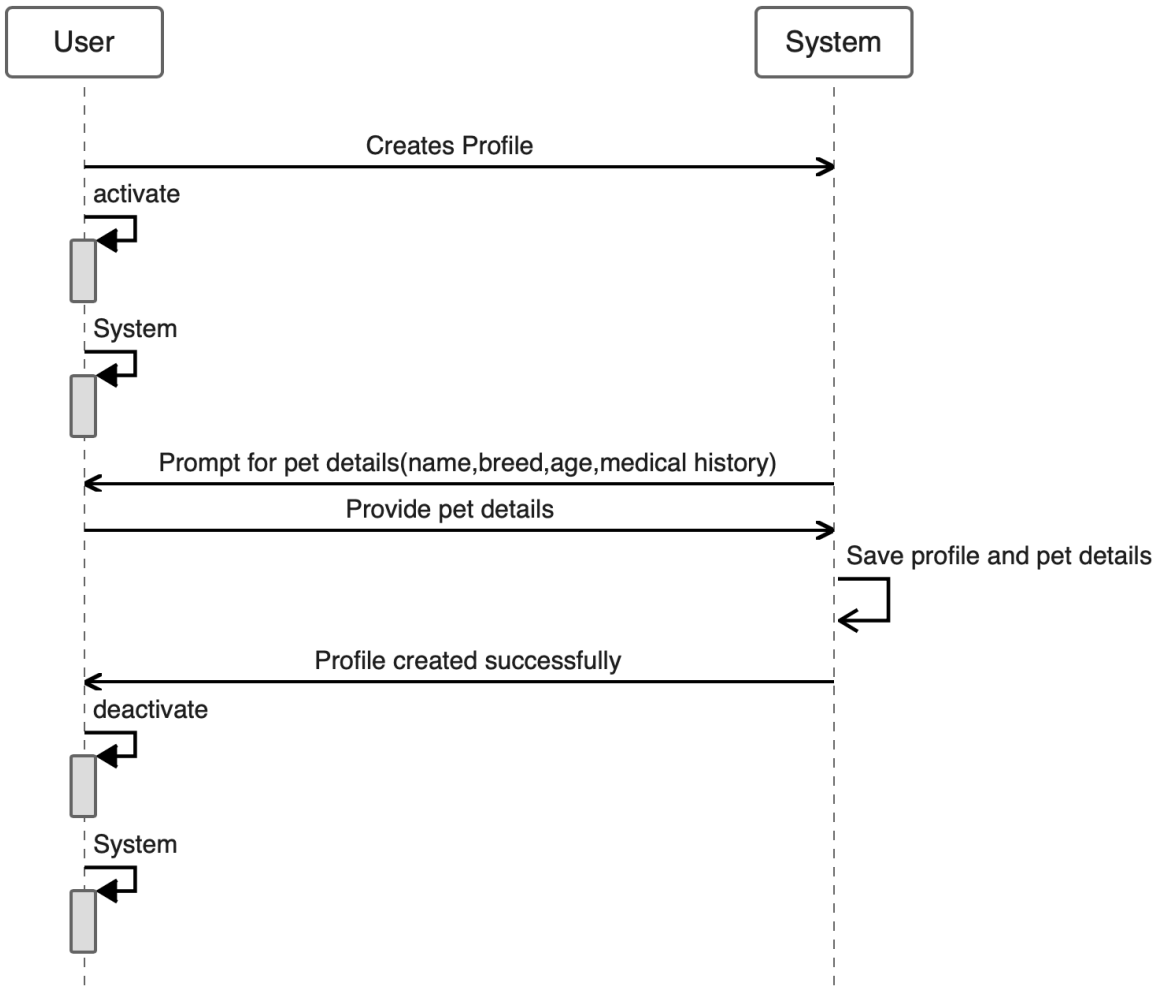
5.1.2 Description

1. The user starts the process to create a new account.
2. The user provides their name, email, and password.
3. The system checks if the provided email is unique.
4. If the email is unique, the system creates the account and confirms to the user that the account has been created successfully.

5. If the email is not unique, the system prompts the user to enter a different email.

5.2 Profile Creation

5.2.1 Diagram

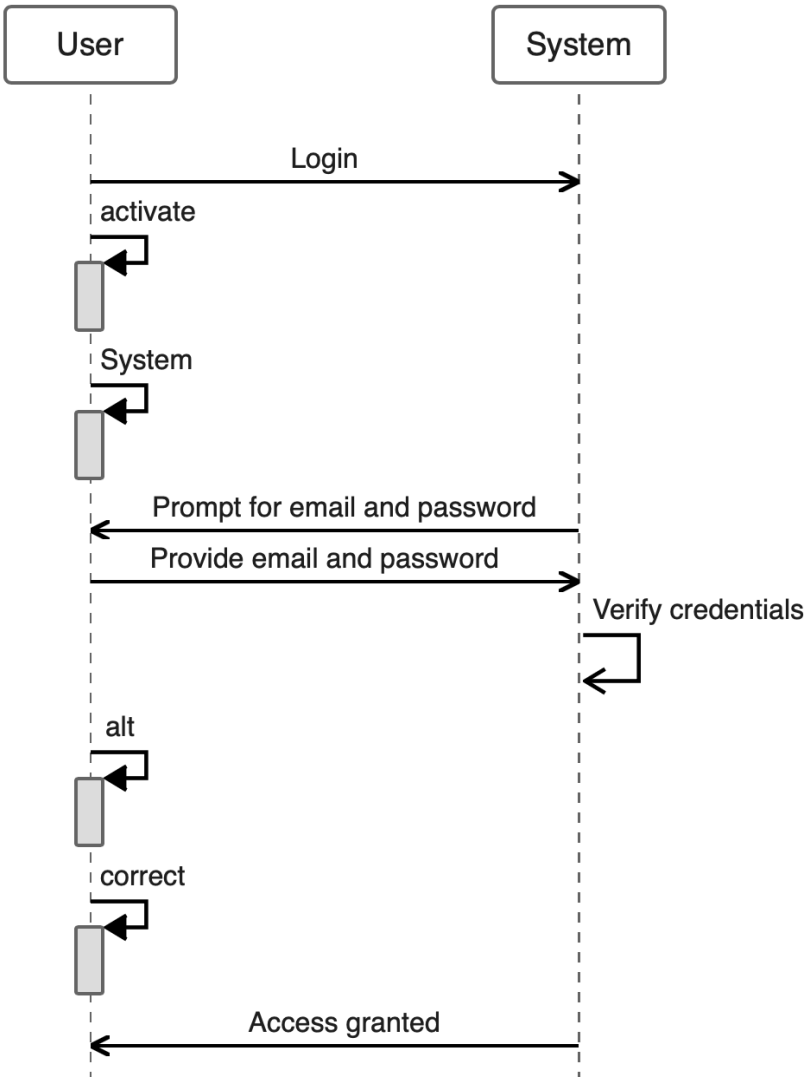


5.2.2 Description

1. The system prompts the user to enter pet details such as name, breed, age, and medical history.
2. The user provides the pet details.
3. The system saves the provided profile and pet details.
4. The system confirms to the user that the profile has been created successfully.

5.3 Login

5.3.1 Diagram

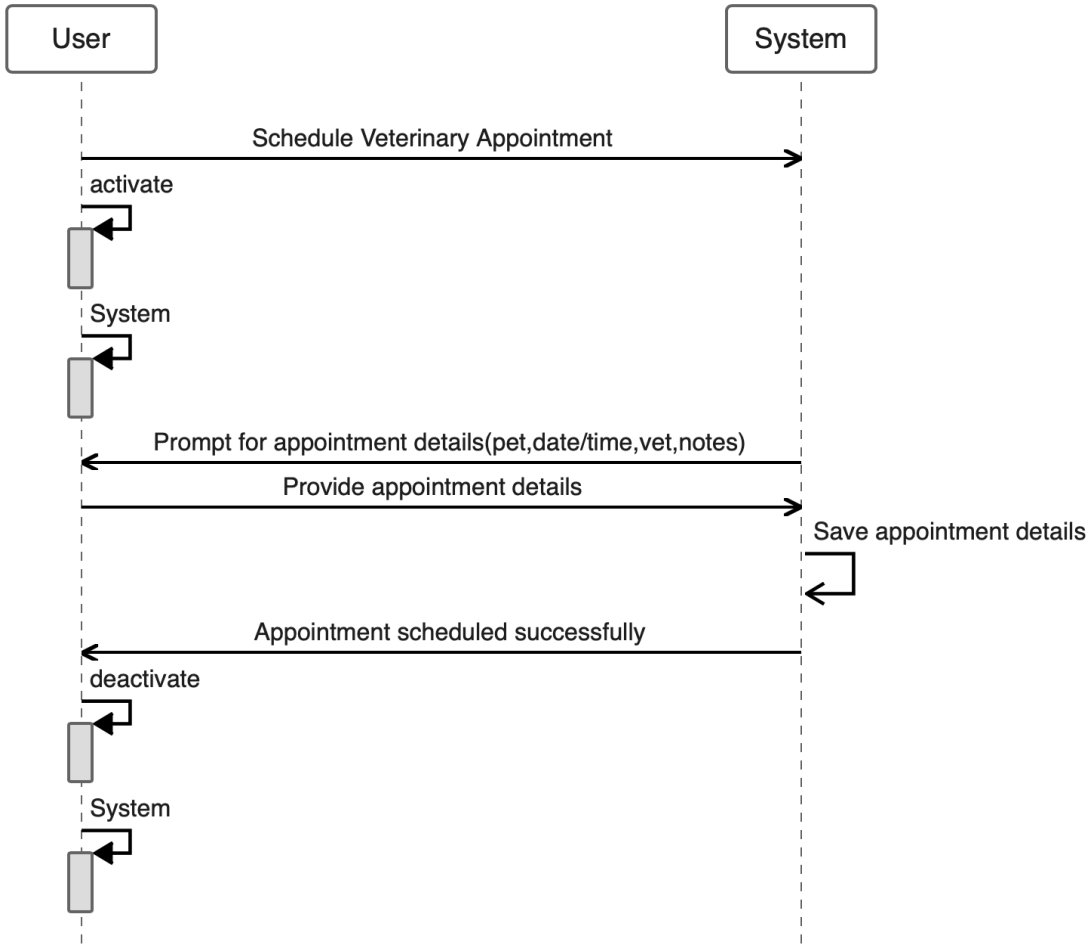


5.3.2 Description

1. The user starts the process to log into their account.
2. The system prompts the user to enter their email and password.
3. The user provides their email and password.
4. The system verifies the provided email and password.
5. If the credentials are correct, the system grants access to the user.
6. If the credentials are incorrect, the system prompts the user to re-enter the correct information.

5.4 Create Veterinary Appointment

5.4.1 Diagram

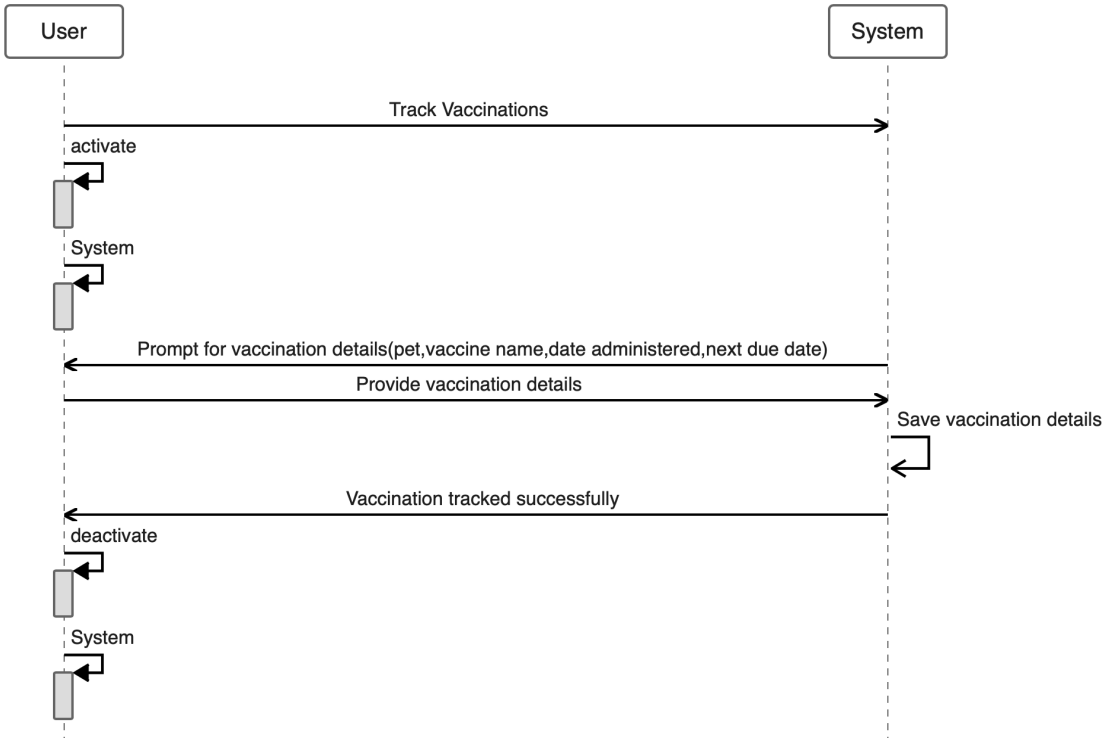


5.4.2 Description

1. The user starts the process to schedule a veterinary appointment.
2. The system prompts the user to enter appointment details such as pet, date/time, vet, and notes.
3. The user provides the appointment details.
4. The system saves the appointment details.
5. The system confirms to the user that the appointment has been scheduled successfully.

5.5 Track Vaccinations

5.5.1 Diagram

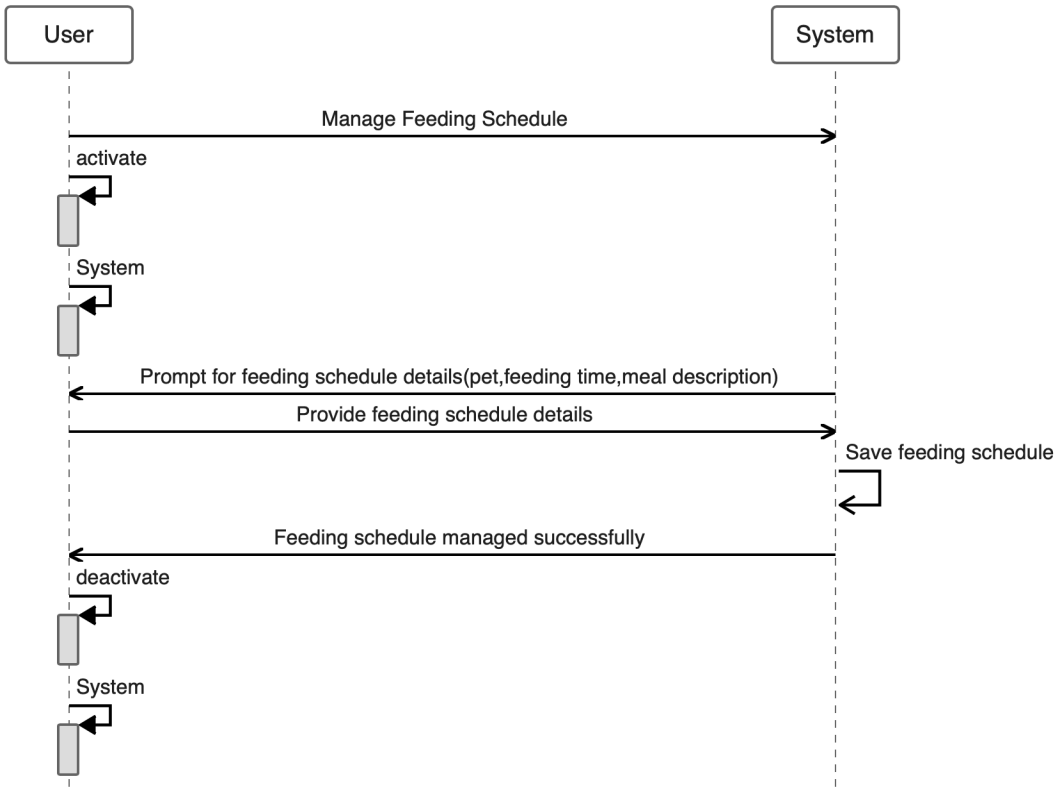


5.5.2 Description

1. The user starts the process to track their pet's vaccinations.
2. The system prompts the user to enter vaccination details such as pet, vaccine name, date administered, and next due date.
3. The user provides the vaccination details.
4. The system saves the vaccination details.
5. The system confirms to the user that the vaccination has been tracked successfully.

5.6 Manage Feeding Schedule

5.6.1 Diagram

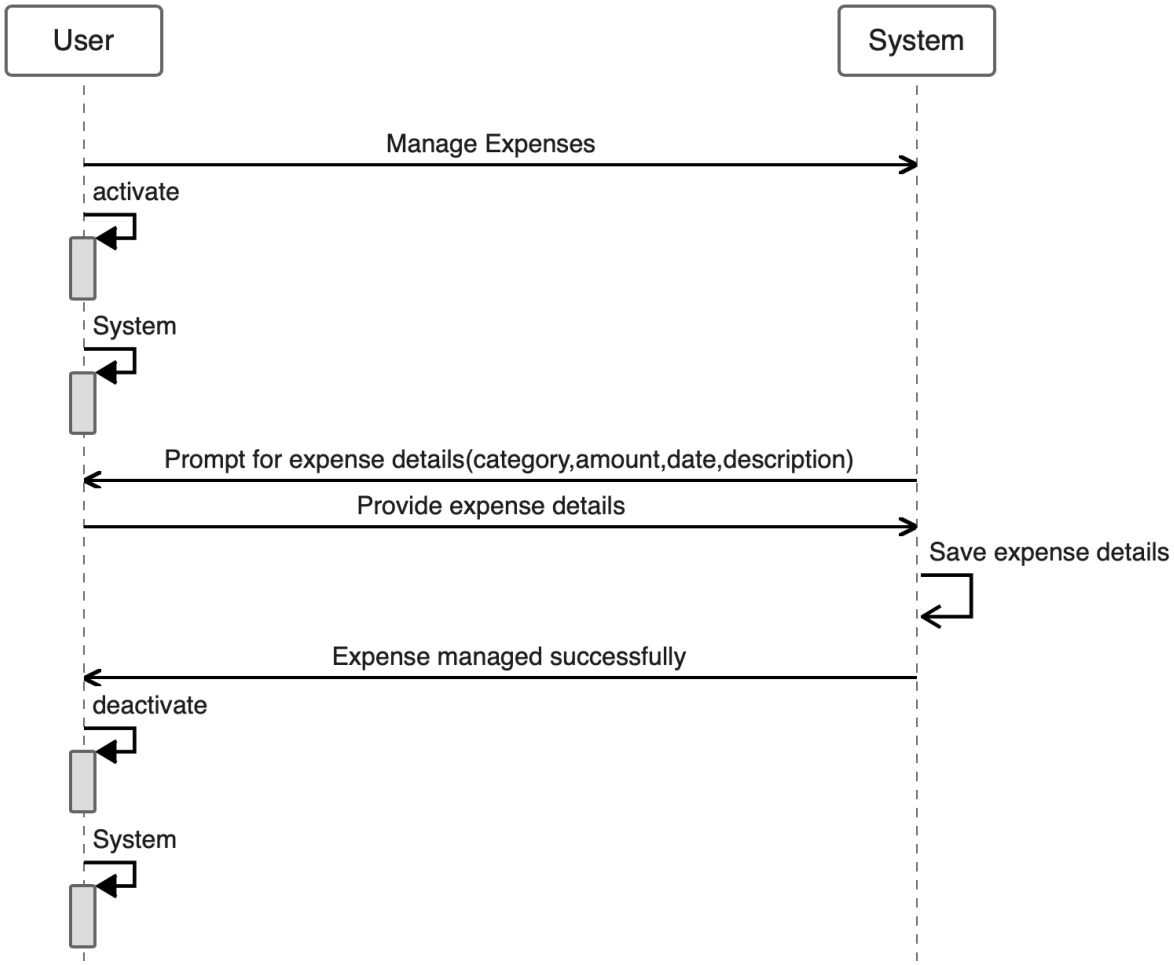


5.6.2 Description

1. The user starts the process to manage their pet's feeding schedule.
2. The system prompts the user to enter feeding schedule details such as pet, feeding time, and meal description.
3. The user provides the feeding schedule details.
4. The system saves the feeding schedule details.
5. The system confirms to the user that the feeding schedule has been managed successfully.

5.7 Manage Expenses

5.7.1 Diagram

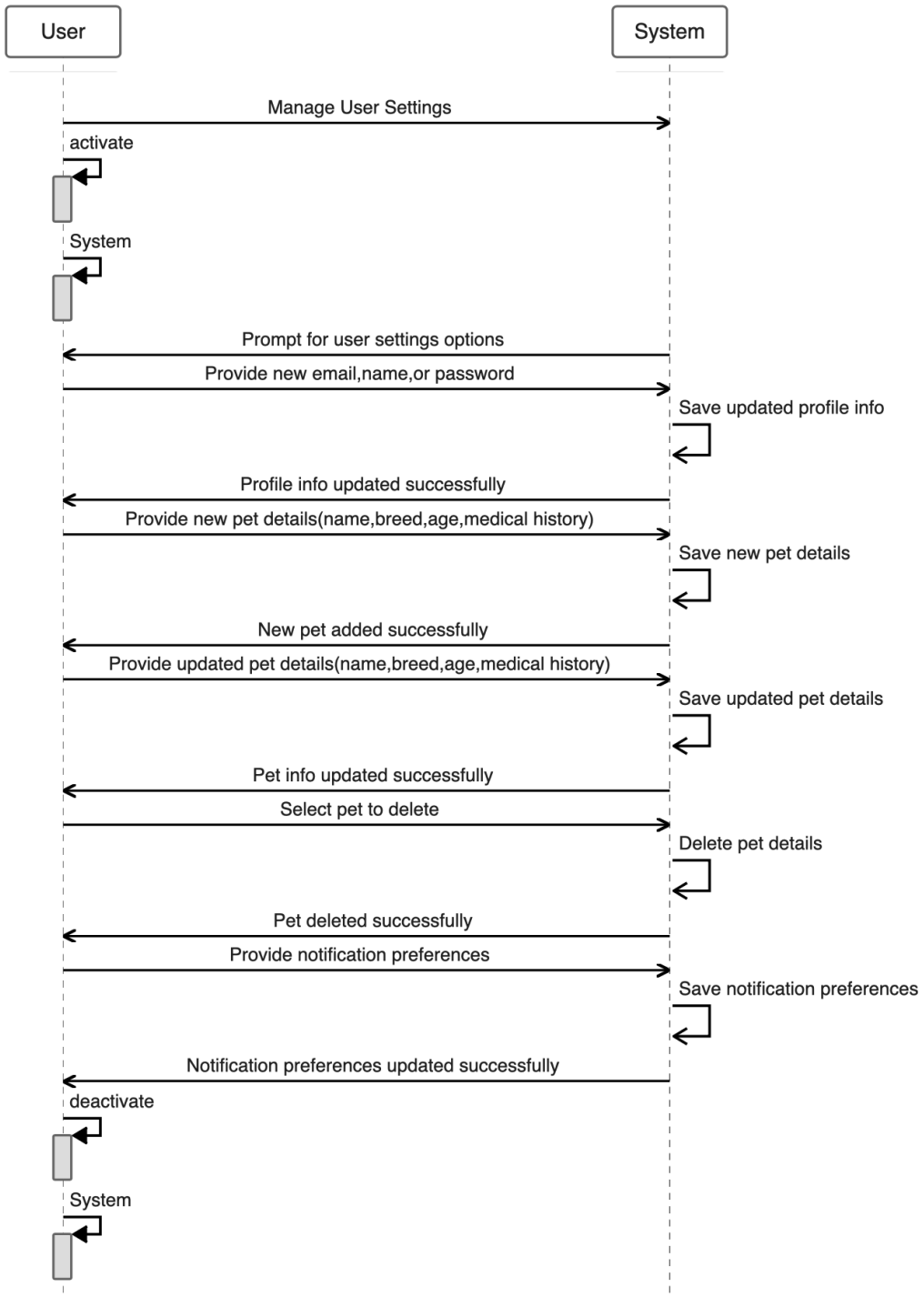


5.7.2 Description

1. The user starts the process to manage their pet-related expenses.
2. The system prompts the user to enter expense details such as category, amount, date, and description.
3. The user provides the expense details.
4. The system saves the expense details.
5. The system confirms to the user that the expense has been managed successfully.

5.8 Manage User Settings

5.8.1 Diagram



5.8.2 Description

1. The user starts the process to manage their user settings.
2. The system prompts the user with available options for managing their settings (edit profile info, add/edit/delete pet info, update notification preferences).

Edit Profile Information:

3. The user provides new profile information such as email, name, or password.
4. The system saves the updated profile information.
5. The system confirms to the user that the profile information has been updated successfully.

Add New Pet:

6. The user provides details for a new pet, including name, breed, age, and medical history.
7. The system saves the new pet details.
8. The system confirms to the user that the new pet has been added successfully.

Edit Existing Pet Details:

9. The user provides updated details for an existing pet, including name, breed, age, and medical history.
10. The system saves the updated pet details.
11. The system confirms to the user that the pet information has been updated successfully.

Delete a Pet:

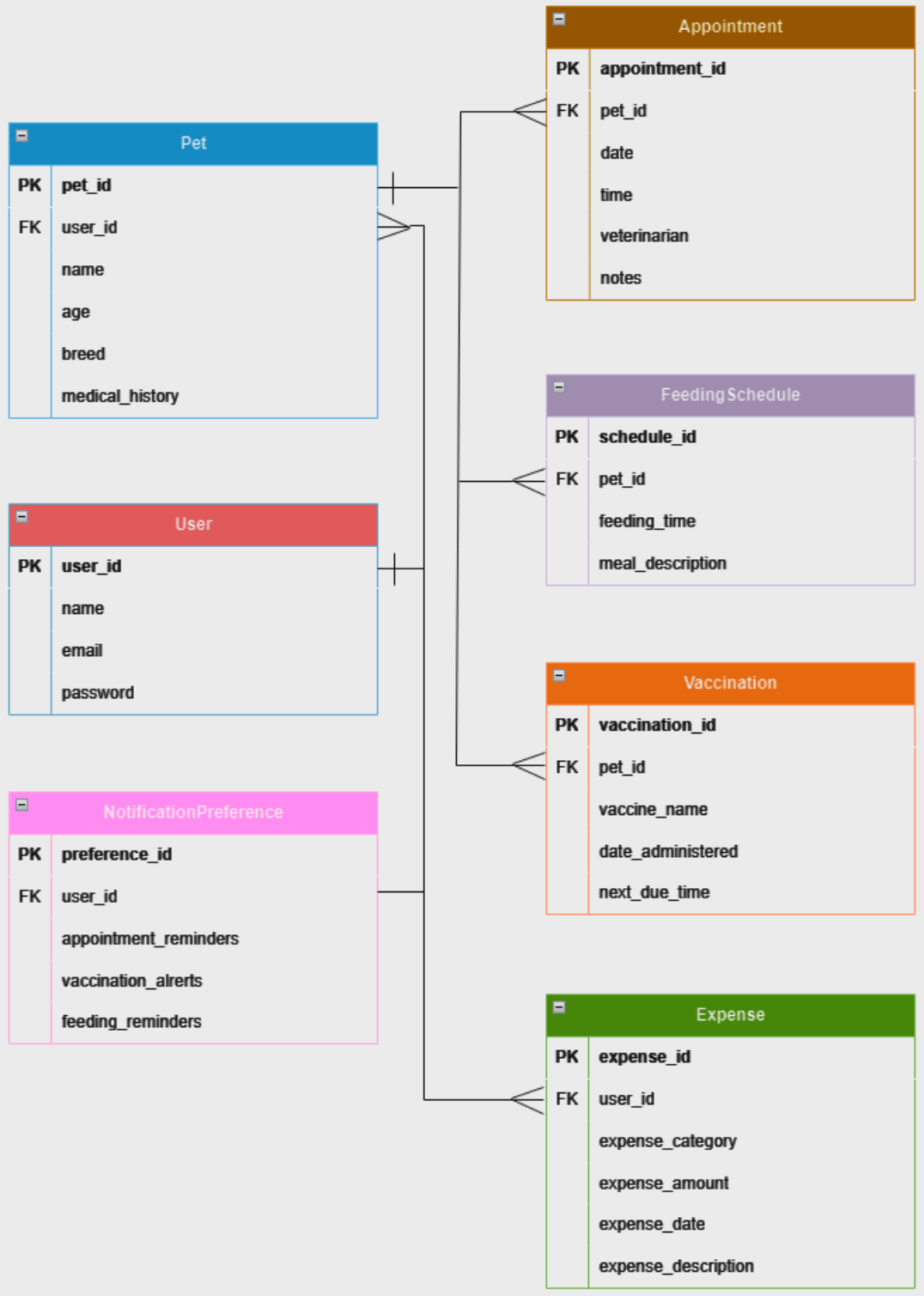
12. The user selects a pet to delete.
13. The system deletes the selected pet's details.
14. The system confirms to the user that the pet has been deleted successfully.

Update Notification Preferences:

15. The user provides their notification preferences, such as opting in or out of specific alerts.
16. The system saves the updated notification preferences.
17. The system confirms to the user that the notification preferences have been updated successfully.

6 Entity Relationship Diagram

6.1 Diagram



6.2 Legend

- Tables in Red: User
- Tables in Blue: Pet
- Tables in Brown: Appointment
- Tables in Orange: Vaccination Tracker
- Tables in Purple: Feeder Schedule
- Tables in Green: Track Expenses

6.3 Description

6.3.1 User

Entries in this table are the users of the application

6.3.1.1 Columns

- User Id: Primary Key
- Name: The name of the user
- Email: The email of the user
- Password: The password of the user

6.3.2 Pet

Entries in this table contain pet information for the owner

- Pet Id: Primary Key
- User Id: Foreign Key, the user that owns the pet
- Name: The name of the pet
- Age: The age of the pet
- Breed: The breed of the pet
- Medical History: The medical history of the pet

6.3.3 Appointment

Entries in this table contain vet appointment information

- Appointment Id: Primary Key
- User Id: Foreign Key, the user that owns the pet
- Pet Id: Foreign Key, the pet that the appointment is for
- Date: The day the appointment is for
- Time: The time of day for the appointment
- Veterinarian: The name of the Veterinarian caring for your pet
- Owner Note: Additional information for the appointment

6.3.4 Vaccination Tracker

Entries in this table contain vaccination tracking information

- Vaccination Id: Primary Key
- Pet Id: Foreign Key, the pet that's receiving the vaccine
- Vaccine Name: The specific vaccine that was administered to your pet
- Date Administered: The date the vaccine was given to your pet
- Next Due Date: The next due date for a vaccine

6.3.5 Feeding Schedule

Entries in this table contain feeding schedule information

- Schedule Id: Primary Key
- Pet Id: Foreign Key, the pet with the feeding schedule
- Feeding Time: How often the user feeds their pet
- Meal Description: The pet's meal plan

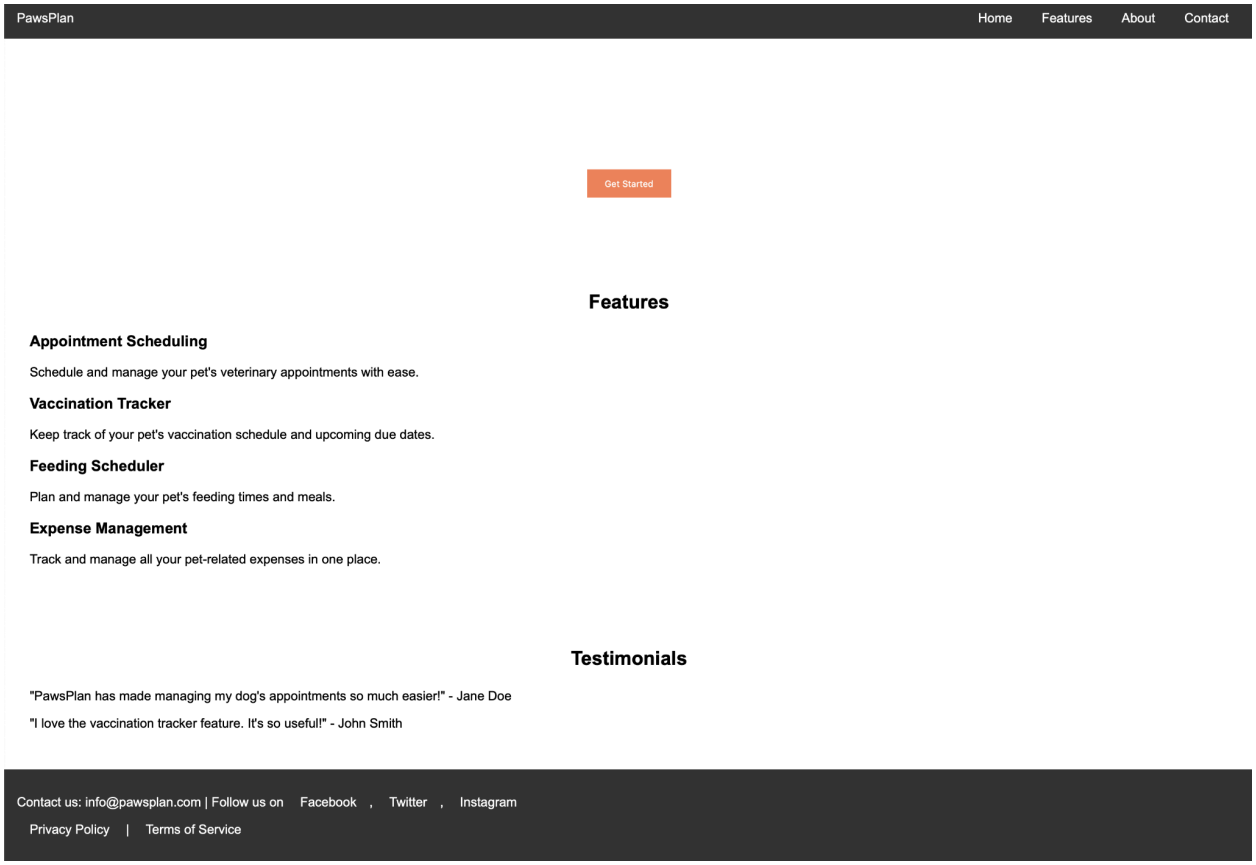
6.3.6 Expense

Entries in this table contain expenses information

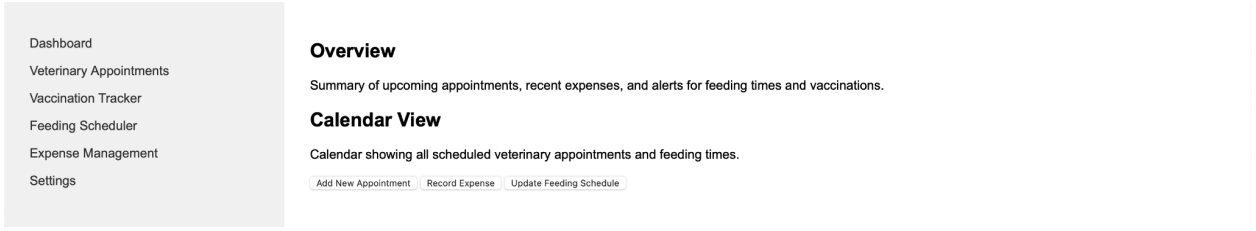
- Expense Id: Primary Key
- User Id: Foreign Key, the user that has expense
- Expense Category: What you're being billed for
- Expense Amount: How much the expense costs
- Expense Date: The date the expense took place
- Expense Description: Additional information for the expense

7 UI Design Draft

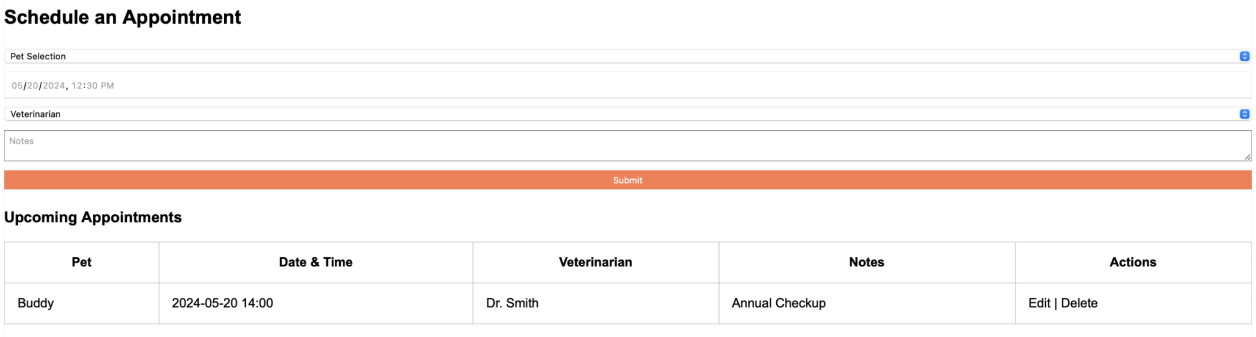
7.1 Landing Page



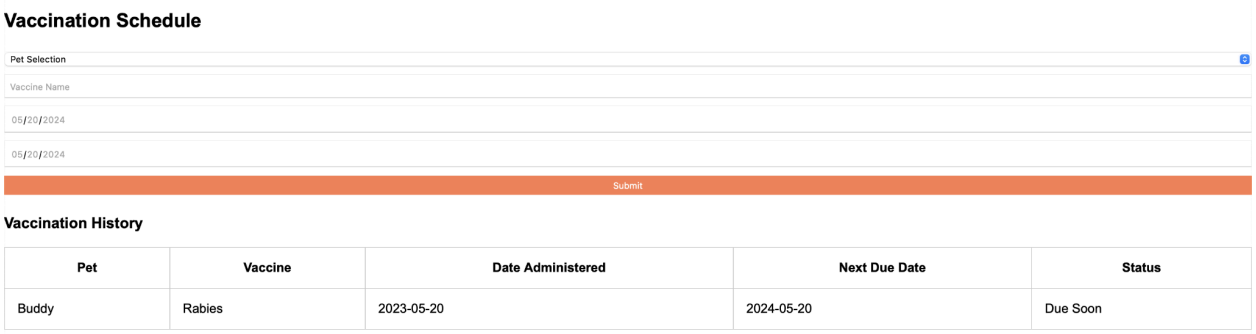
7.2 Dashboard



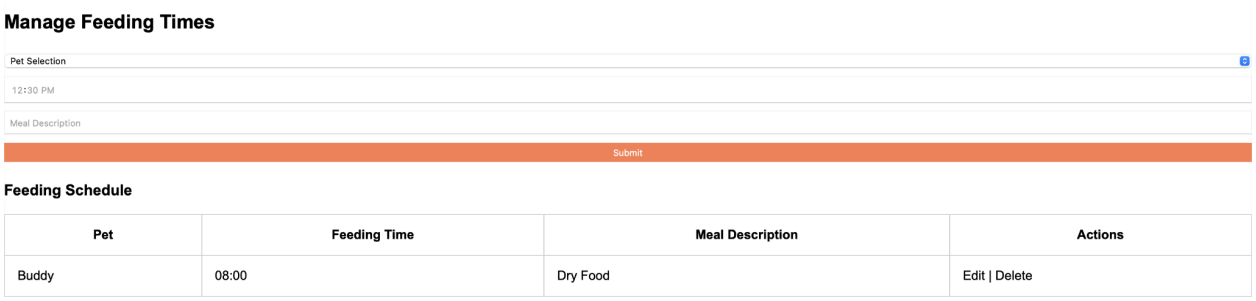
7.3 Scheduling an Appointment



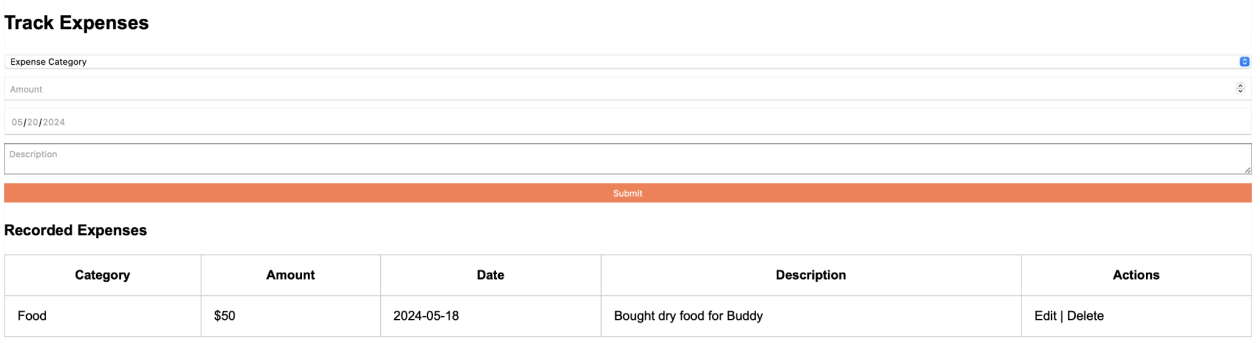
7.4 Scheduling a Vaccine



7.5 Manage Feeding Times



7.6 Tracking Expenses



7.7 User Settings

8 Specifications

The overall design involves the creation of a user-friendly web application for various pet care tasks, focusing on managing veterinary appointments, vaccination schedules, feeding routines, and expenses. The backend will be powered by Python Flask, and the frontend will utilize HTML, CSS, and JavaScript. SQLite will be used as the database to store and manage data efficiently.

8.1 Flask

Flask is a Python-based micro-framework for developing web applications. It provides a simple and flexible way to create APIs and manage database interactions.

Documentation: Link

8.1.1 Flask Features:

Blueprints: Flask allows for the creation of modular components known as blueprints. Each module can have its own models, routes, and templates.

ORM (SQLAlchemy): Flask integrates well with SQLAlchemy, an ORM for managing database tables and queries. Python classes represent database tables, and changes to these classes can be managed through migrations.

Event-Driven Architecture: Flask can be extended with various libraries to create an event-driven architecture, making it easier to manage asynchronous tasks and events.

8.2 Frontend

The frontend will be developed using HTML, CSS, and JavaScript to ensure a responsive and interactive user experience.

8.3 SQL Database

A SQL database will be used for storing and managing data. SQLite is chosen for its simplicity and ease of use during development.

- SQLite: Flask provides SQLite as a default database for development.

Documentation: Link

8.4 Non-Software Requirements

- GitHub: For code sharing and version control.
- Discord: For team communication.

- Google Drive: For sharing non-code documents and resources.

9 Implementations - Code Excerpts

9.1 Core Algorithms and Functionalities

9.1.1 User Authentication

 # models.py

```
class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(20), unique=True, nullable=False)
    email = db.Column(db.String(120), unique=True, nullable=False)
    image_file = db.Column(db.String(20), nullable=False, default='default.jpg')
    password = db.Column(db.String(60), nullable=False)
    is_verified = db.Column(db.Boolean, nullable=False, default=False)
```

 # routes.py

```
@bp.route("/register", methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('main.dashboard'))
    form = RegistrationForm()
    if form.validate_on_submit():
        hashed_password =
bcrypt.generate_password_hash(form.password.data).decode('utf-8')
        user = User(username=form.username.data, email=form.email.data,
password=hashed_password)
        db.session.add(user)
        db.session.commit()
        send_verification_email(user)
        flash('An email has been sent to verify your account.', 'info')
        return redirect(url_for('main.login'))
    return render_template('register.html', title='Register', form=form)

@bp.route("/login", methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('main.dashboard'))
    form = LoginForm()
    if form.validate_on_submit():
```

```
        user = User.query.filter_by(email=form.email.data).first()
        if user and bcrypt.check_password_hash(user.password, form.password.data):
            if user.is_verified:
                login_user(user, remember=form.remember.data)
                next_page = request.args.get('next')
                return redirect(next_page) if next_page else
redirect(url_for('main.dashboard'))
            else:
                flash('Your account is not verified. Please check your email.',
'warning')
                return redirect(url_for('main.login'))
        else:
            flash('Login Unsuccessful. Please check email and password', 'danger')
    return render_template('login.html', title='Login', form=form)
```

User Authentication Explanation

- **User Model**: This defines the structure of the User table, with fields for username, email, password, and verification status.
- **Registration**: The register route handles user registration by validating the form data, hashing the password, and saving the user to the database. It also sends a verification email.
- **Login**: The login route checks if the user is already authenticated, validates the login form, checks the user's credentials, and logs them in if they are verified.

9.1.2 Pet Management

 # models.py

```
class Pet(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    gender = db.Column(db.String(6), nullable=False)
    breed = db.Column(db.String(100), nullable=False)
    age = db.Column(db.Integer, nullable=False)
    picture = db.Column(db.String(100), nullable=False)  # storing filename
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
    histories = db.relationship('PetHistory', backref='pet', lazy=True)
    appointments = db.relationship('VetAppointment', backref='pet', lazy=True)
    vaccine_records = db.relationship('VaccineRecord', backref='pet', lazy=True)
```

```
    feeding_schedules = db.relationship('FeedingSchedule', backref='pet', lazy=True)
```

 # routes.py

```
@bp.route("/add_pet", methods=['GET', 'POST'])
@login_required
def add_pet():
    form = AddPetForm()
    if form.validate_on_submit():
        picture_file = secure_filename(form.picture.data.filename)
        picture_path = os.path.join(current_app.root_path, 'static/pet_pics',
picture_file)
        form.picture.data.save(picture_path)
        pet = Pet(
            name=form.name.data,
            age=form.age.data,
            breed=form.breed.data,
            gender=form.gender.data,
            picture=picture_file,
            user_id=current_user.id
        )
        db.session.add(pet)
        db.session.commit()
        flash('Your pet has been added!', 'success')
        return redirect(url_for('main.dashboard'))
    return render_template('add_pet.html', title='Add Pet', form=form)
```

Pet Management Explanation

- **Pet Model**: This model defines the structure of the 'Pet' table with fields for name, breed, age, gender, picture, and user relationship.
- **Add Pet**: The 'add_pet route' handles adding a new pet by validating the form data, saving the pet's picture, and storing the pet information in the database.

9.1.3 Vaccine Tracking

 # models.py

```
class VaccineRecord(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    vaccine_type = db.Column(db.String(100), nullable=False)
    date_administered = db.Column(db.Date, nullable=False)
```

```
    next_due_date = db.Column(db.Date, nullable=False)
    pet_id = db.Column(db.Integer, db.ForeignKey('pet.id'), nullable=False)
    notification_sent = db.Column(db.Boolean, nullable=False, default=False)
```

 # routes.py

```
@bp.route('/vaccine_tracking', methods=['GET', 'POST'])
@login_required
def vaccine_tracking():
    form = VaccineTrackingForm()
    pets = Pet.query.filter_by(user_id=current_user.id).all()
    pet_id = request.args.get('pet')
    records_query = VaccineRecord.query.join(Pet).filter(Pet.user_id ==
current_user.id)
    if pet_id:
        records_query = records_query.filter(VaccineRecord.pet_id == pet_id)
    records = records_query.all()
    if form.validate_on_submit():
        vaccine_record = VaccineRecord(
            pet_id=form.pet.data,
            vaccine_type=form.vaccine_type.data,
            date_administered=form.date_administered.data,
            next_due_date=form.next_due_date.data,
            notification_sent=False
        )
        db.session.add(vaccine_record)
        db.session.commit()
        flash('Vaccine record added successfully!', 'success')
        return redirect(url_for('main.vaccine_tracking'))
    return render_template('vaccine_tracking.html', form=form, records=records,
pets=pets)
```

Vaccine Tracking Explanation

- **Vaccine Record Model**: Defines the structure of the ‘VaccineRecord’ table with fields for vaccine type, date administered, next due date, and pet relationship.
- **Vaccine Tracking**: The ‘vaccine_tracking’ route handles adding and displaying vaccine records. It validates form data, creates a new record, and saves it to the database.

9.1.4 Expense Management

 # models.py

```
class Expense(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    pet_id = db.Column(db.Integer, db.ForeignKey('pet.id'), nullable=False)
    expense_type = db.Column(db.String(100), nullable=False)
    amount = db.Column(db.Numeric(10, 2), nullable=False)
    date = db.Column(db.Date, nullable=False)
```

 # routes.py

```
@bp.route('/expense_management', methods=['GET', 'POST'])
@login_required
def expense_management():
    form = ExpenseForm()
    pets = Pet.query.filter_by(user_id=current_user.id).all()
    expense_types = ['Appointments','Vaccination','Food', 'Vet', 'Toy and
Entertainment','Training','Insurance','Pet
Sitting','Travel','Registration','Accessories','Miscellaneous Supplies','Dietary
Supplements','Subscriptions','Events','Other']

    if form.validate_on_submit():
        expense = Expense(
            pet_id=form.pet.data,
            expense_type=form.expense_type.data,
            amount=form.amount.data,
            date=form.date.data
        )
        db.session.add(expense)
        db.session.commit()
        flash('Expense added successfully!', 'success')
        return redirect(url_for('main.expense_management'))
```

Expense Management Explanation

- Expense Model: Defines the structure of the ‘Expense table’ with fields for expense type, amount, date, and pet relationship.
- Expense Management: The ‘expense_management’ route handles adding and displaying expenses. It validates form data, creates a new expense record, and saves it to the database.

9.2 External APIs/Libraries Used

9.2.1 APScheduler

- Manages scheduled tasks like sending email reminders.
- Example:

```
from apscheduler.schedulers.background import
BackgroundScheduler scheduler = BackgroundScheduler()
scheduler.add_job(id='check_vaccine_due_dates',
func=check_vaccine_due_dates, trigger='interval', minutes=1)
```

9.2.2 WeasyPrint

- Converts HTML/CSS to PDF for generating reports.
- Example:

```
from weasyprint import HTML
html = render_template('report_template.html', data=data)
pdf = HTML(string=html).write_pdf()
```

10 User Installation and Deployment Guide

10.1 Prerequisites

- Python 3.11: Ensure Python 3.11 is installed on your machine.
- Git: Ensure Git is installed on your machine.
- Virtual Environment: Recommended to avoid dependency conflicts.

10.2 Steps to Install and Run PawsPlan

1. Clone the Repository
   - git clone https://github.com/ageorgoulakis/PawsPlan.git
   - cd PawsPlan

2. Set Up a Virtual Environment
   - python -m venv myenv
   - source myenv/bin/activate # On Windows, use `myenv\Scripts\activate`

3. Install Dependencies
   - pip install -r requirements.txt

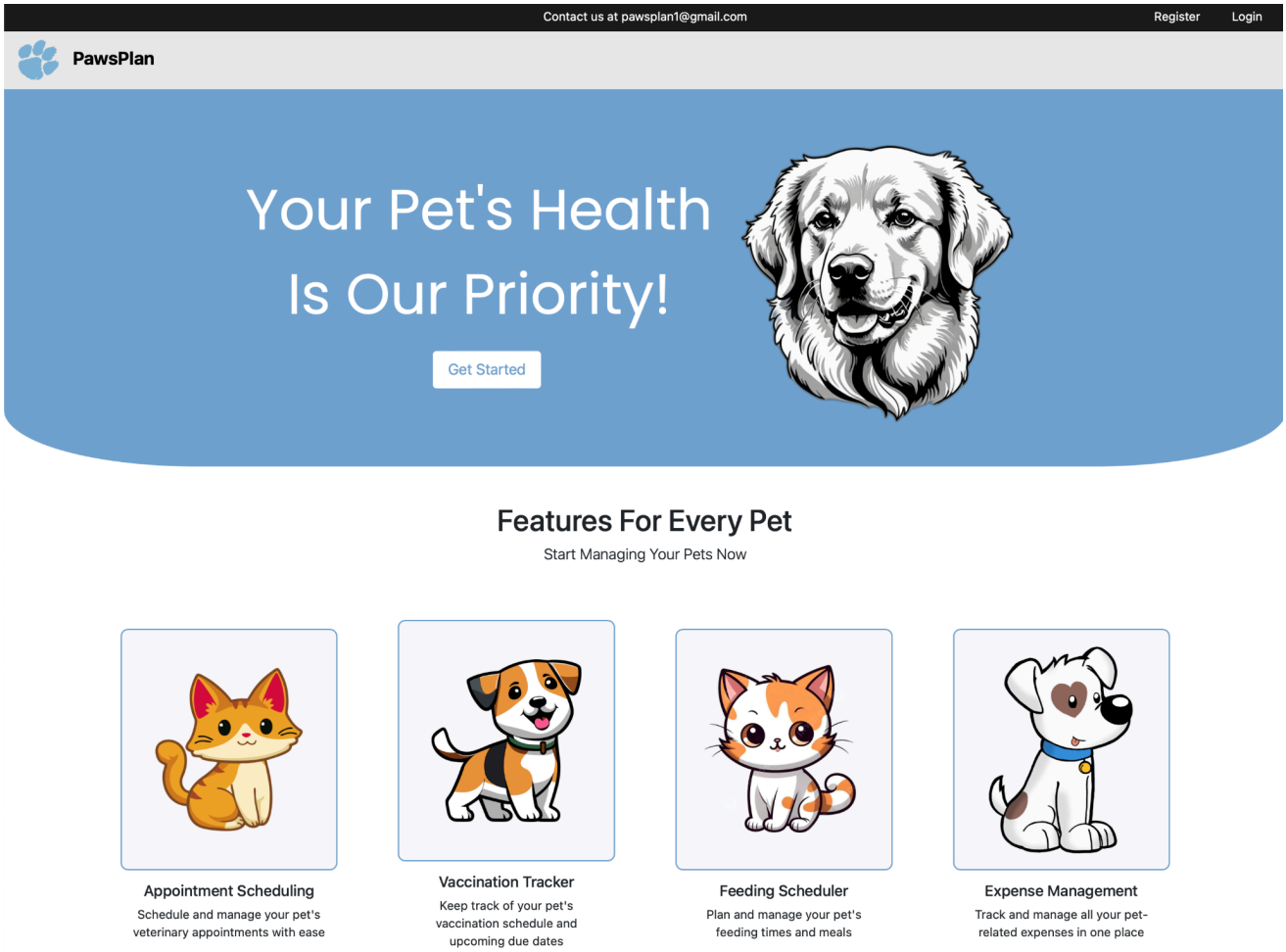
4. Set Up the Database
    - flask db init
    - flask db migrate -m "Initial migration."
    - flask db upgrade

5. Run the Application
    - flask run

6. Access the Application
    - Open your web browser and go to ‘http://127.0.0.1:5000’.

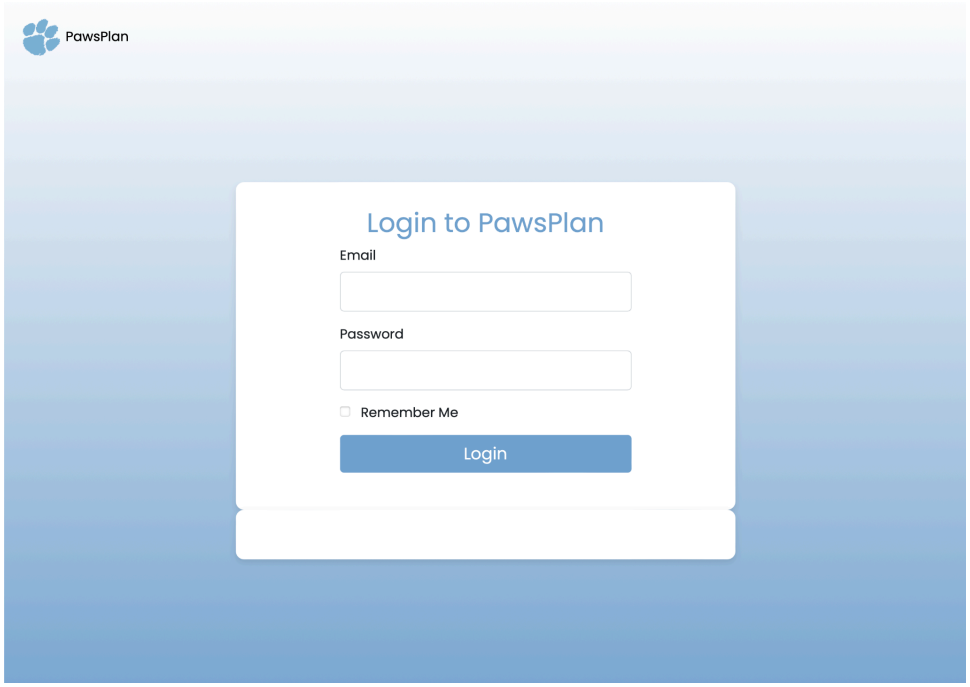
11 Screenshots

11.1 Landing Home Page

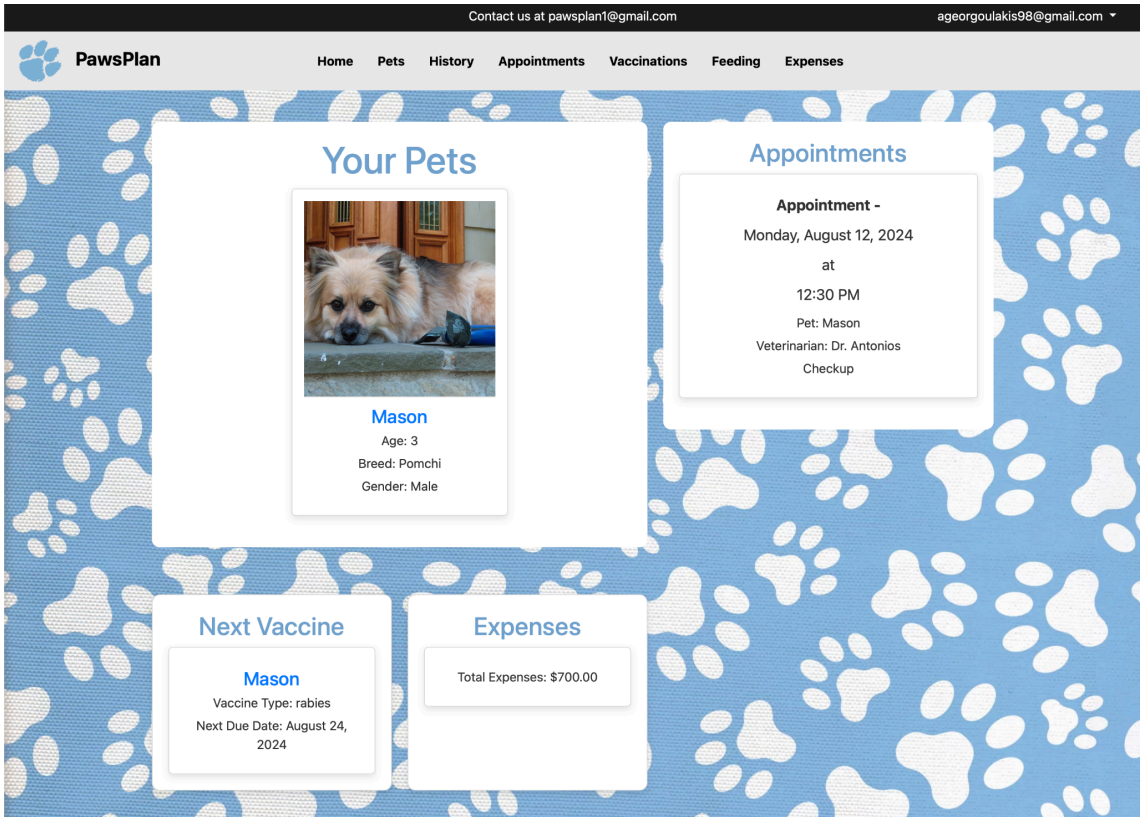


11.2 Register Page

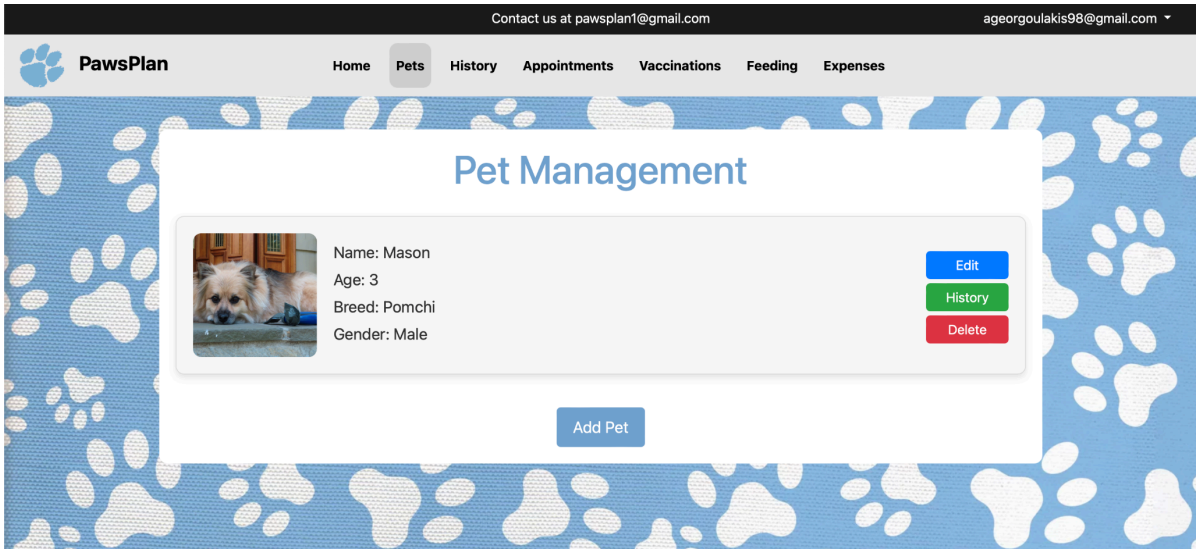
11.3 Login Page



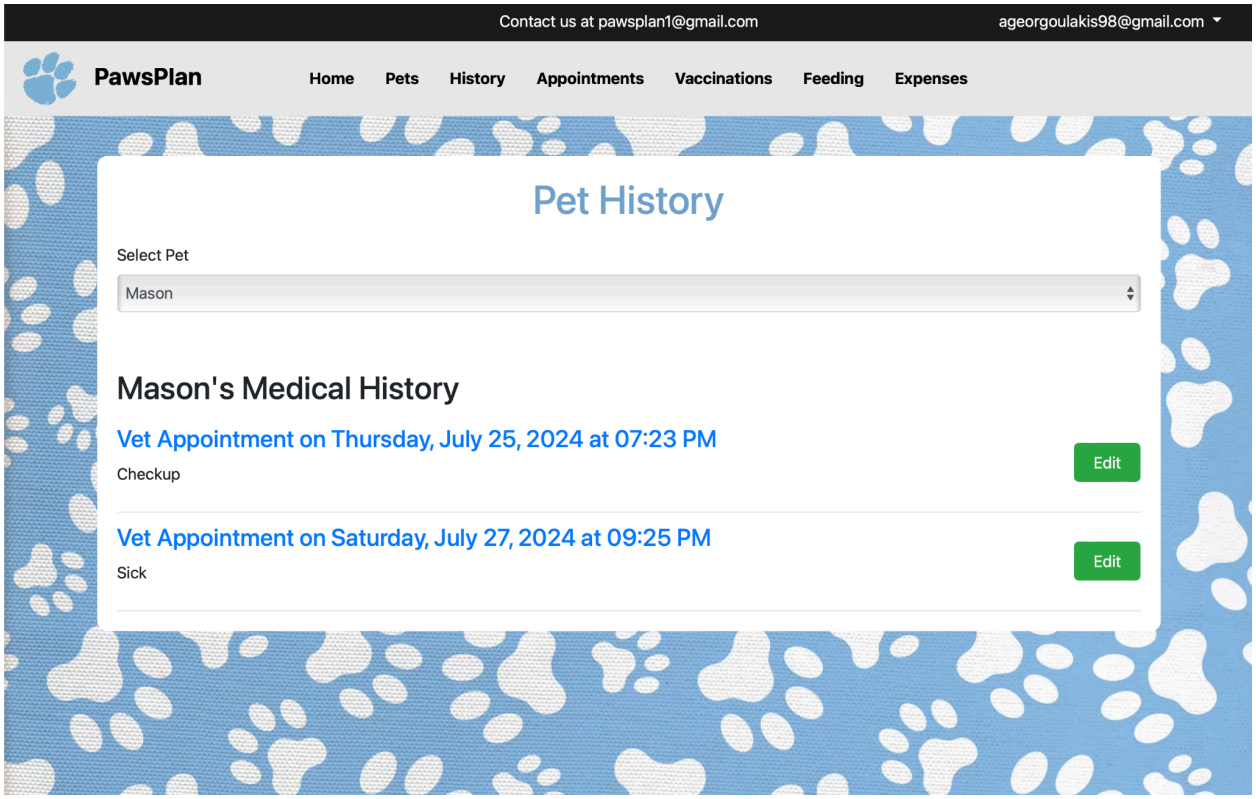
11.4 User Home Page



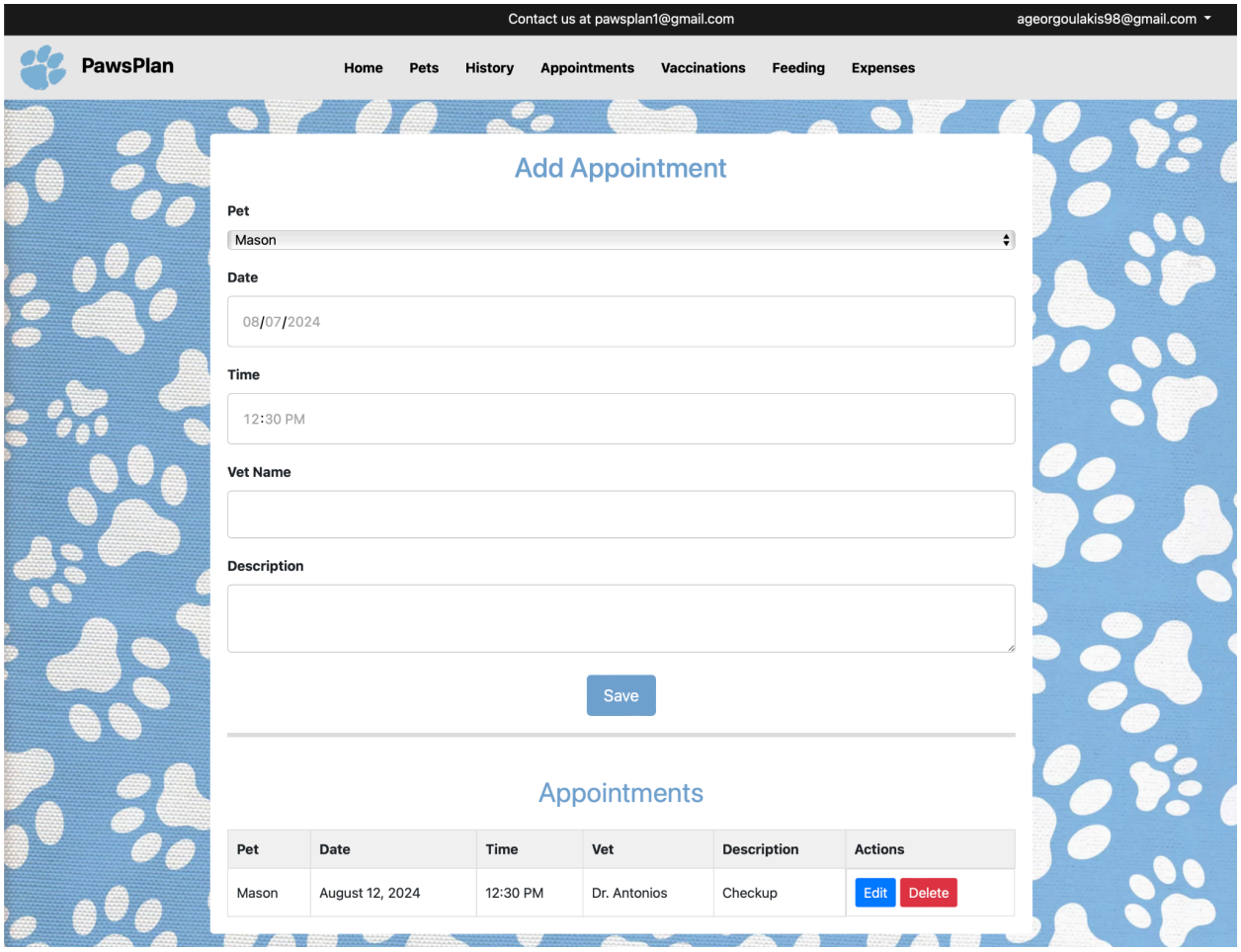
11.5 Manage Pets



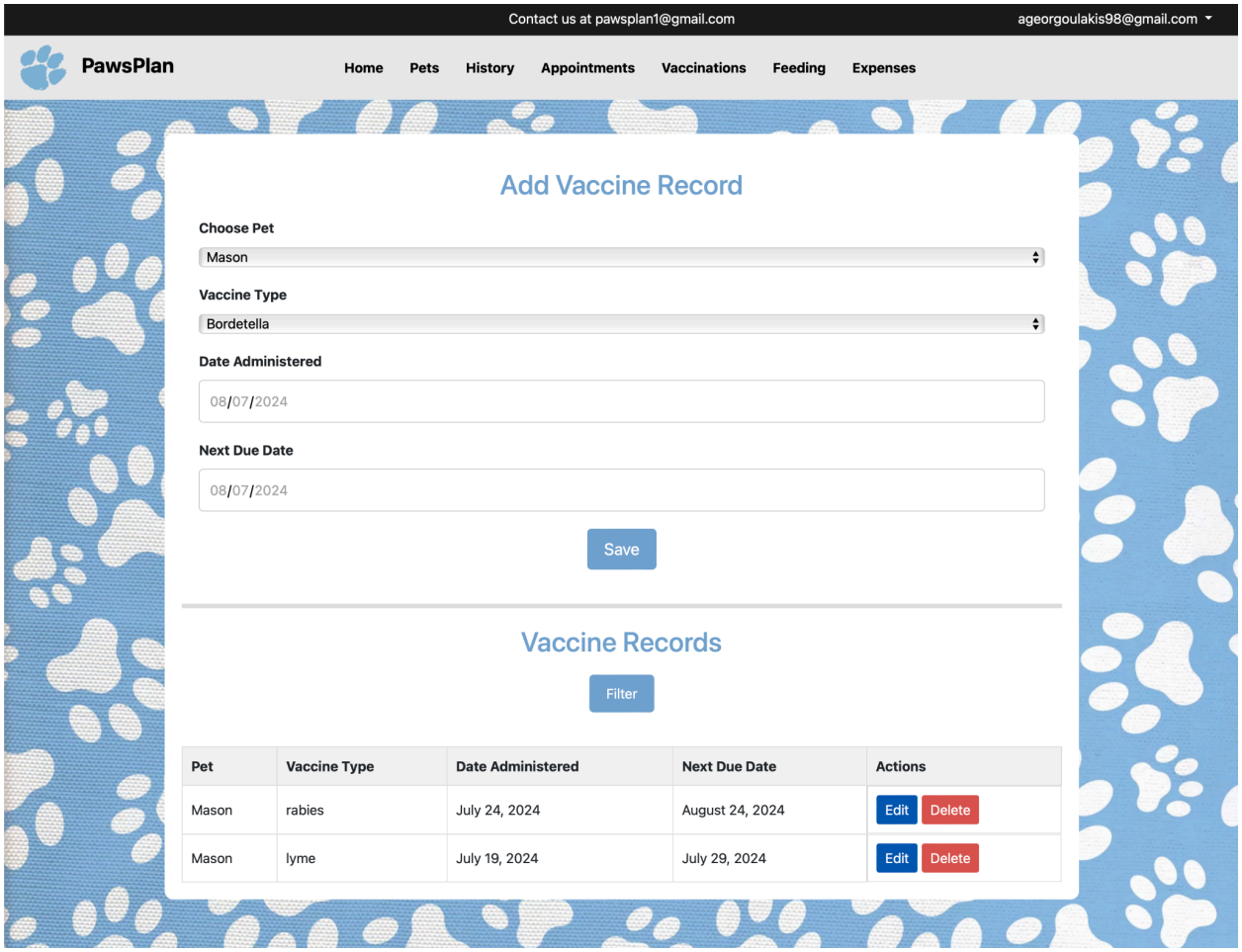
11.6 History


Contact us at pawsplan1@gmail.com
ageorgoulakis98@gmail.com
PawsPlan
Home
Pets
History
Appointments
Vaccinations
Feeding
Expenses
Pet History
Select Pet
Mason
Mason's Medical History
Vet Appointment on Thursday, July 25, 2024 at 07:23 PM
Checkup
Edit
Vet Appointment on Saturday, July 27, 2024 at 09:25 PM
Sick
Edit

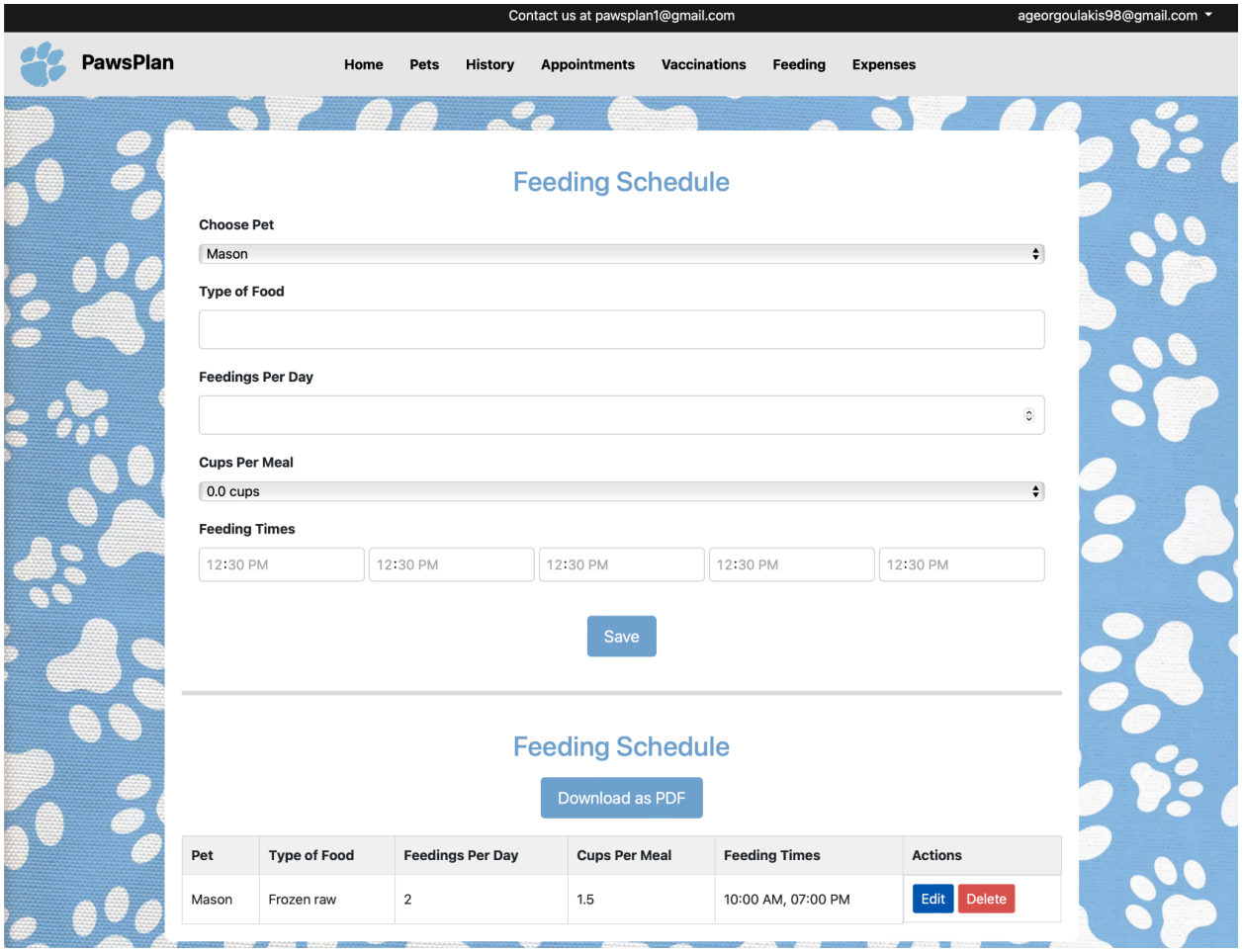
11.7 Appointments



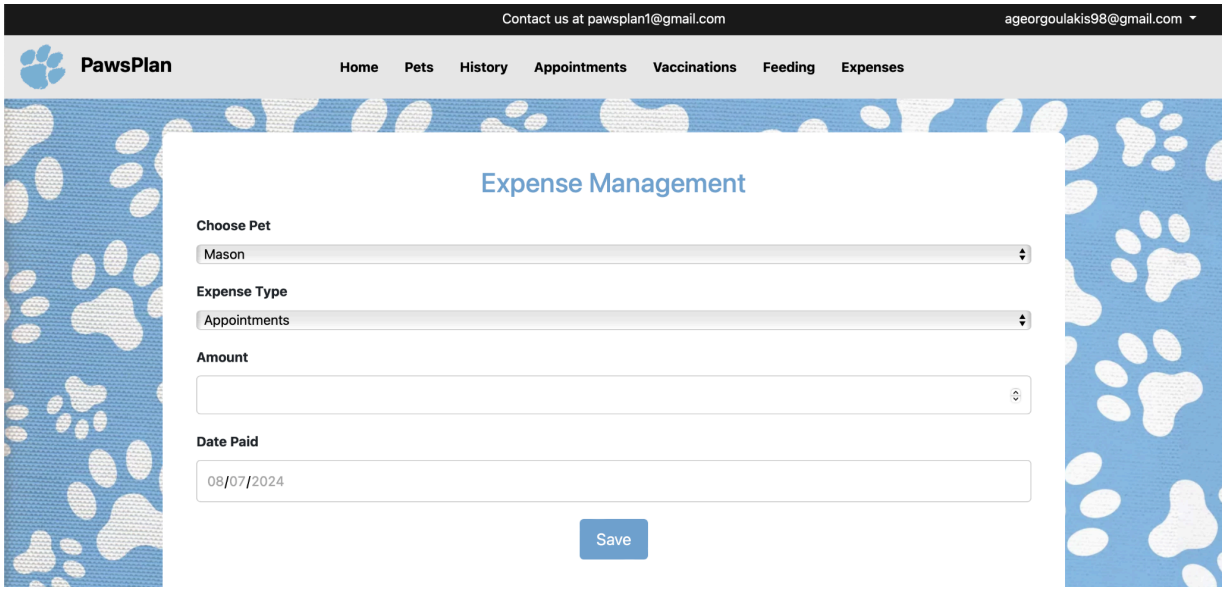
11.8 Vaccinations

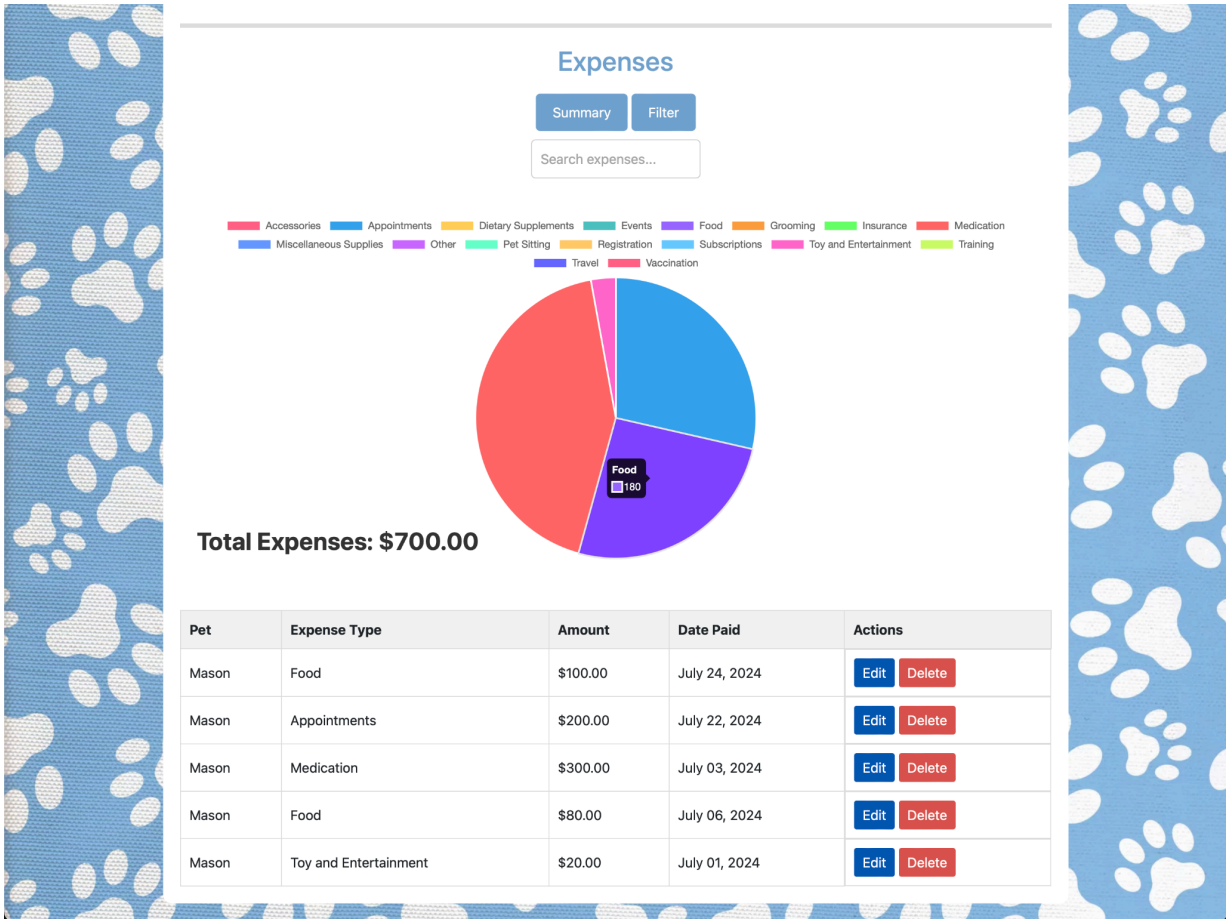


11. 9 Feeding


Contact us at pawsplan1@gmail.com
ageorgoulakis98@gmail.com
PawsPlan
Home
Pets
History
Appointments
Vaccinations
Feeding
Expenses
Feeding Schedule
Choose Pet
Mason
Type of Food
Feedings Per Day
Cups Per Meal
0.0 cups
Feeding Times
12:30 PM
12:30 PM
12:30 PM
12:30 PM
12:30 PM
Save
Feeding Schedule
Download as PDF
Pet
Type of Food
Feedings Per Day
Cups Per Meal
Feeding Times
Actions
Mason
Frozen raw
2
1.5
10:00 AM, 07:00 PM
Edit
Delete

11.10 Expenses





12 State of Implementation

12.1 User Authentication

Description: Allows users to register, log in, and log out securely.
Extent: Fully implemented and tested.
Details: Uses Flask-Login for session management and Flask-Bcrypt for password hashing.

12.2 Pet Management

Description: Users can add, edit, and delete their pets.
Extent: Fully implemented and tested.
Details: Supports storing pet details like name, breed, age, gender, and picture.

12.3 Pet History

Description: Users can maintain a history of events related to their pets (e.g., vet visits, treatments).
Extent: Fully implemented and tested.
Details: Allows adding, editing, and viewing historical events for each pet.

12.4 Vet Appointments

Description: Users can schedule, view, and manage vet appointments.
Extent: Fully implemented and tested.
Details: Supports appointment details such as date, time, vet name, and description.

12.5 Vaccine Tracking

Description: Users can track pet vaccinations, including the date administered and next due date.
Extent: Fully implemented and tested.
Details: Includes email notifications for upcoming vaccinations.

12.6 Feeding Schedule

Description: Users can set and manage feeding schedules for their pets.
Extent: Fully implemented and tested.
Details: Supports multiple feeding times per day and different food types.

12.7 Expense Management

Description: Users can track expenses related to their pets.
Extent: Fully implemented and tested.
Details: Allows adding, editing, and viewing expenses, including type, amount, and date.

12.8 Email Notifications

Description: Sends email reminders for upcoming vaccinations and appointments.
Extent: Fully implemented and tested.
Details: Uses APScheduler to manage scheduled tasks for sending emails.

12.9 PDF Reports

Description: Generates PDF reports for feeding schedules and vaccination records.
Extent: Fully implemented and tested.
Details: Uses WeasyPrint to convert HTML/CSS to PDF.

13 Testing and Evaluation

13.1 Testing

We conducted thorough manual testing throughout the development process to ensure the proper functionality of each feature. This approach involved manually navigating through the application and verifying that every aspect of the system worked as intended. Here's a detailed breakdown of our manual testing process:

1. Feature-by-Feature Testing:

- Description: Each feature was individually tested by interacting with the application's interface.
- Objective: Ensure each feature operates correctly and provides the expected results.
- Outcome: Immediate detection and resolution of bugs, ensuring each feature's reliability before moving to the next.

2. User Flow Testing:
    - Description: Simulated typical user interactions by performing tasks such as registering, logging in, adding pets, scheduling appointments, and tracking expenses.
    - Objective: Validate that the user can perform end-to-end tasks without encountering issues.
    - Outcome: Confirmed that all user flows are smooth, logical, and free from errors.
3. Cross-Browser Testing:
    - Description: Tested the application on various web browsers, including Chrome, Firefox, and Safari.
    - Objective: Ensure compatibility and consistent performance across different browsers.
    - Outcome: Verified that the application maintains functionality and a consistent appearance on all tested browsers.
4. Responsive Design Testing:
    - Description: Tested the application on different devices, such as desktops, tablets, and smartphones.
    - Objective: Ensure the interface is responsive and user-friendly on various screen sizes and resolutions.
    - Outcome: Confirmed that the application provides a seamless user experience across all device types.

Based on the extensive manual testing conducted, all critical issues have been identified and resolved. At this stage, no further testing is required. However, as the application evolves and new features are added, additional testing will be performed to maintain the system's integrity and usability.

The project successfully meets the objectives we set out to achieve. Each feature has been implemented and tested to ensure it functions as intended. The application is comprehensive and user-friendly, providing a robust platform for pet care management.

13.2 Evaluation

The system is fully usable and designed to be intuitive and user-friendly. Key usability aspects include:

1. Ease of Use:
    - Description: The application features a straightforward interface, making it easy for users to navigate and use.
    - Outcome: Users can quickly learn how to use the system and efficiently manage their pet care tasks.
2. Responsiveness:
    - Description: The application is responsive and works well across various devices, including desktops, tablets, and smartphones.
    - Outcome: Users can access the system from any device, providing flexibility and convenience.
3. User Feedback:
    - Description: Initial user feedback has been positive, with users appreciating the comprehensive feature set and ease of use.
    - Outcome: The system meets the needs of pet owners, and continuous feedback will help us make further improvements.

14 Lessons Learned and Reflection

14.1 What Went Well

Effective Team Collaboration

- Description: We worked cohesively, leveraging each other's strengths.
- Outcome: This collaboration led to efficient problem-solving and timely completion of tasks.

Robust Planning and Design

- Description: We invested significant time in the initial planning and design phases.
- Outcome: This groundwork facilitated a smoother development process, reducing the need for major redesigns later.

Continuous Testing

- Description: We implemented continuous manual testing throughout the development lifecycle.
- Outcome: Early detection and resolution of issues ensured a stable and reliable application.

14.2 What We Would Do Differently

More Frequent User Feedback

- Reflection: Although we incorporated user feedback, doing so more frequently could have further refined the user experience.
- Future Action: Schedule regular user testing sessions throughout the development process to continuously align the product with user expectations.

Improved Time Management

- Reflection: Some phases took longer than anticipated, leading to a tight schedule towards the end.
- Future Action: Allocate more buffer time in the project timeline for unexpected delays and ensure a more balanced workload distribution.

14. 3 Instances of Changing Course and Reconsidering Design Decisions

Database Schema Redesign

- Initial Decision: Our original database schema was simpler but lacked flexibility for future feature additions.
- Change: Midway through development, we realized the need for a more scalable design and revised the schema to accommodate additional functionalities.
- Outcome: The new schema supports future expansions without requiring significant overhauls.

Switch to AWS for Deployment

- Initial Decision: We initially planned to use a simpler cloud service for deployment.

- Change: After evaluating the benefits of AWS, including scalability and reliability, we decided to switch to AWS.
- Outcome: AWS provided a more robust and scalable deployment environment, enhancing the application's performance and availability.

User Authentication Method Update

- Initial Decision: The original authentication method was basic and lacked advanced security features.
- Change: Based on further research, we upgraded to a more secure authentication method using Flask-Login and bcrypt for password hashing.
- Outcome: This change improved the security of user data, aligning with best practices and user expectations.

14.4 Reflection on the Development Process

Developing PawsPlan has been an enlightening experience, teaching us the importance of adaptability and user-centric design. Our team learned to balance robust planning with the flexibility to pivot when necessary. Continuous testing and incorporating user feedback were crucial in creating a product that not only meets but exceeds user expectations.

Future Considerations

- More Comprehensive Testing: Integrate automated testing to complement manual efforts.
- Regular User Feedback: Increase the frequency of user feedback sessions to continuously improve the user experience.
- Scalability: Focus on designing scalable features from the start to facilitate easier expansion.

15 Version 2.0

Given an additional month or two, we would focus on implementing the following features and improvements to enhance the functionality and user experience of PawsPlan:

15.1 Mobile Application

- Description: Develop a mobile app version of PawsPlan for iOS and Android.
- Benefit: Provides users with on-the-go access to manage their pet's care.
- Implementation: Utilize frameworks like React Native or Flutter to build a cross-platform mobile application.

15.2 Integration with Veterinary Systems

- Description: Integrate PawsPlan with veterinary systems to sync appointment schedules and medical records.
- Benefit: Streamlines the process of managing veterinary appointments and accessing pet medical records.
- Implementation: Use APIs provided by veterinary systems to fetch and sync data with PawsPlan's database.

15.3 Automated Feeding Schedule Reminders

- Description: Integrate automated notifications to remind users of their pets' feeding times.
- Benefit: Ensures pets are fed on time, enhancing user convenience and pet care.
- Implementation: Use APScheduler to trigger email or mobile notifications based on the feeding schedule set by the user.

15.4 Advanced Reporting and Analytics

- Description: Add detailed analytics and reporting features to provide insights into pet care trends.
- Benefit: Helps users make informed decisions about their pets' health and care routines.
- Implementation: Generate visual reports and charts using libraries like Plotly or Chart.js to display data such as expense trends, vaccination schedules, and feeding habits.

15.5 Enhanced User Interface and Experience (UI/UX)

- Description: Revamp the UI/UX to make the platform more intuitive and visually appealing.
- Benefit: Improves user satisfaction and engagement with the application.
- Implementation: Conduct user testing sessions to gather feedback and iterate on the design. Implement changes using modern design principles and tools.

These additional features would significantly enhance PawsPlan, making it more comprehensive and user-friendly. The mobile application would provide convenience, integration with veterinary systems would streamline pet health management, automated reminders would ensure timely pet care, advanced analytics would offer valuable insights, and an enhanced UI/UX would improve overall user satisfaction.

16 Conclusion

The journey of developing PawsPlan has been a significant learning experience. This project has provided us with the opportunity to acquire new skills, deepen our understanding of various technologies, and tackle real-world challenges. Here, we reflect on the process of developing these skills and the resources that facilitated our learning.

16.1 Tools, APIs, and Skills Acquired

1. Flask (Python)

- Skill Development: We gained a thorough understanding of Flask, a micro web framework in Python, which is essential for developing web applications. This included learning about routing, templating, and form handling.
- Challenges: The initial challenge was to understand the framework's structure and how to efficiently organize the project files. We also had to ensure secure user authentication and session management.

2. SQLite

- Skill Development: Working with SQLite allowed us to understand database management systems. We learned how to design schemas,

perform CRUD operations, and manage relationships between different entities using SQLAlchemy.
- Challenges: Designing a normalized database schema that efficiently supports all features without redundancy was challenging. Ensuring data integrity and handling complex queries were also part of the learning curve.

3. Bootstrap (HTML/CSS/JS)

- Skill Development: Bootstrap provided us with the tools to create a responsive and visually appealing user interface. We learned how to quickly design and customize web pages using Bootstrap's grid system and components.
- Challenges: The main challenge was to customize Bootstrap components to fit our specific design needs without breaking responsiveness. Ensuring cross-browser compatibility also required careful testing.

4. APScheduler

- Skill Development: Implementing APScheduler taught us how to manage scheduled tasks within our Flask application. This was crucial for features like automated email reminders for vaccine due dates.
- Challenges: Understanding how to set up and manage background tasks without affecting the main application's performance was challenging. We also had to ensure that scheduled tasks executed reliably under different scenarios.

5. WeasyPrint

- Skill Development: WeasyPrint enabled us to convert HTML/CSS content into PDF reports. We learned how to design printable documents and integrate this functionality into our application.
- Challenges: Designing complex PDF reports that looked professional and were correctly formatted was challenging. We also had to ensure that the conversion process was efficient and error-free.

6. AWS

- Skill Development: Deploying PawsPlan on AWS provided us with practical experience in cloud computing. We learned how to set up and manage instances, configure security groups, and deploy a web application on a scalable infrastructure.
- Challenges: The initial setup and configuration of AWS services were complex. We had to ensure that our deployment was secure, scalable, and cost-effective.

16.2 Resources Utilized

To achieve our project goals, we relied on various resources:

- Online Tutorials and Documentation: Websites like Flask's official documentation, SQLAlchemy documentation, and Bootstrap's official website were invaluable.
- Community Forums: Platforms like Stack Overflow provided quick solutions to specific problems and guidance from experienced developers.
- Educational Platforms: Courses on platforms like Udemy and Coursera helped us understand the fundamental concepts and advanced techniques required for our project.

16.3 Main Challenges and Solutions

1. Integration of Multiple Technologies

- Challenge: Integrating various technologies like Flask, SQLite, Bootstrap, and APScheduler required careful planning and coordination.
- Solution: We followed a modular development approach, breaking down the project into smaller, manageable components. This allowed us to focus on integrating each part seamlessly.

2. Ensuring Data Security and Integrity

- Challenge: Handling user data securely and ensuring the integrity of our database were critical challenges.
- Solution: We implemented robust authentication mechanisms, used prepared statements to prevent SQL injection, and conducted thorough testing to ensure data integrity.

3. Managing Project Scope and Time

- Challenge: Balancing project scope with available time was a constant challenge.
- Solution: We prioritized core features, set realistic milestones, and used agile methodologies to manage our development process efficiently.

16.4 Reflection and Future Directions

What Went Well

- The project's modular design allowed for smooth integration and testing of individual components.
- Regular code reviews and pair programming sessions helped us maintain code quality and resolve issues quickly.

Areas for Improvement

- Better initial planning and time management could have reduced last-minute rushes.
- More focus on automated testing could improve the robustness of the application.

Future Work

- Developing a mobile application to complement the web platform.
- Integrating with veterinary systems to provide seamless management of pet health records.
- Enhancing the user interface based on feedback to improve user experience.

In conclusion, developing PawsPlan has been a rewarding journey that significantly contributed to our professional growth. We gained hands-on experience with technologies like Flask, SQLite, Bootstrap, APScheduler, and AWS, enhancing our technical skills and understanding.

This project improved our project management and problem-solving abilities, teaching us the importance of setting realistic milestones, working collaboratively, and adapting to challenges. Overcoming these challenges made us more resilient and resourceful developers.

The knowledge and confidence gained from PawsPlan have prepared us for future projects, making this journey an invaluable part of our growth as developers.

17 Project Timeline

17.1 Strategy

Our strategy involves completing foundational features before implementing advanced functionalities. Initial sprints will focus on project scaffolding, setting up the environment, and implementing core features. Later sprints will focus on refining features and incorporating feedback.

17.1 Timeline Milestones

- Early June: Minimally viable functionality defined
- June 4: SRS draft/update completed
- Late June: Elements of system prototype implemented and demoed
- July: Full implementation mode
- Early August: Final presentation
- Weekly meetings: Feedback on progress

17.2 Sprint Work Plan

The project will be executed across a series of simplified sprints aligned with the milestones.

17.2.1 Sprint 1: Initial Setup and Environment Configuration (May 20 - May 26)

- Initialize Flask project on GitHub.
- Create Flask blueprints for modular development.
- Set up basic routes and models for core features.

17.2.2 Sprint 2: Basic Functionality Implementation (May 27 - Jun. 2)

- Implement basic backend and frontend components.
- Set up user authentication and profile management.

17.2.3 Sprint 3: Defining Minimally Viable Functionality and SRS Draft (Jun. 3 - Jun. 9)

- Define minimally viable functionality based on initial requirements.
- Outline key features and user stories.
- Draft and update the Software Requirements Specification (SRS).

17.2.4 Sprint 4: System Prototype Development (Jun. 10 - Jun. 16)

- Begin implementing elements of the system prototype.
- Develop core functionalities for veterinary appointment scheduling, vaccination tracking, and feeding schedule management.

17.2.5 Sprint 5: System Prototype Demo and Feedback (Jun. 17 - Jun. 23)

- Complete the system prototype.
- Demo the prototype and gather feedback.
- Refine features based on feedback.

17.2.6 Sprint 6: Full Implementation Mode (Jun. 24 - Jul. 21)

- Continue implementing advanced features.
- Develop an expense tracking module.
- Integrate notifications and reminders system.
- Improve UI/UX design based on feedback.
- Refine and optimize frontend components.

17.2.7 Sprint 7: Full Implementation and Testing (Jul. 22 - July 28)

- Complete all remaining features and ensure integration.
- Implement advanced security measures and route protection.
- Optimize database queries and backend performance.
- Conduct thorough testing and bug fixing.

17.2.8 Sprint 8: Wrap Up Implementation and Documentation (Jul. 29 - Aug. 4)

- Wrap up implementation and writing of required technical

documentation or report.
- Conduct final code review.
- Prepare for a mandatory practice presentation.

17.2.9 Final Sprint: Final Project Presentations (Aug. 5 - Aug. 6)

- Final testing, bug fixes, and documentation updates.
- Conduct mandatory practice presentation.
- August 6th, 1-3pm: Final project presentations.