CS 353 - Database Systems Term Project

Project Name: PetLink

Group No: 3

Design Report

15.11.2023

Instructor: Prof. Özgür Ulusoy **TA:** Mousa Farshkar Azari

Group Members:

Ahmet Alperen Yılmazyıldız - 22002712

Zeynep Doğa Dellal - 22002572

Borga Haktan Bilen - 22002733

Kardelen Ceren - 22003017

Yusuf Şenyüz - 21903105

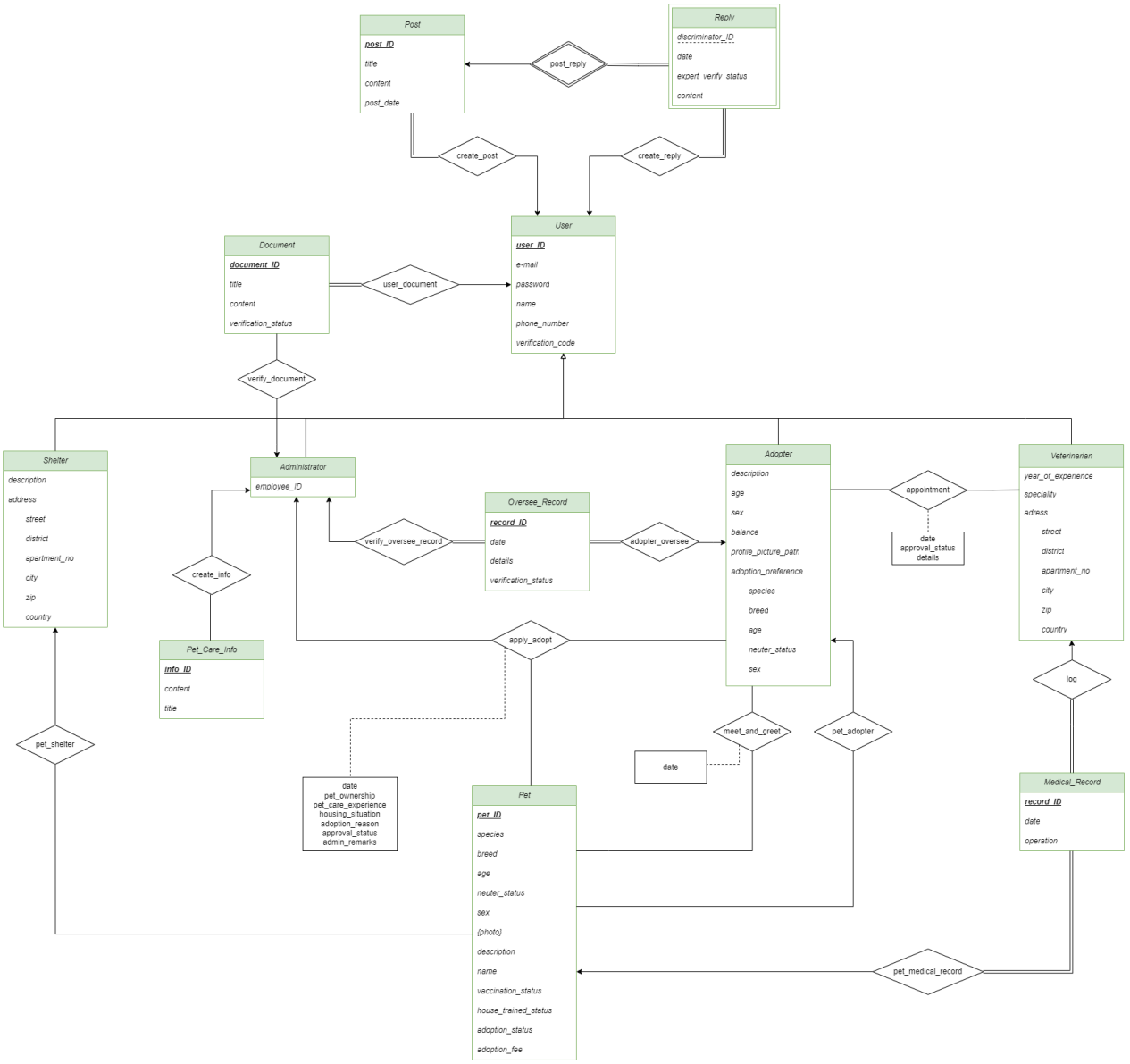
Table of Contents

1. Design of the Database

1.1 Revised E/R Diagram

After our feedback session with our TA, we agreed that no changes were needed to our E/R diagram. However, given the additional features requested in the design report, we made these modifications:

- Added a balance attribute to Adopter and an adoption fee to Pet.
- Changed the apply_adopt relation to include the Administrator instead of Shelter, as specified in the instructions.
- Added a meet_and_greet relation between Adopter and Pet.
- Added more attributes to the apply_adopt relation representing adoption applications, such as the adoption_reason and housing_situation, to encourage responsible pet adoption.



Link:
https://viewer.diagrams.net/?highlight=0000ff&nav=1&title=DB-Project-ER-Diagram.drawio#Uhttps%3A%2F%2Fdrive.google.com%2Fuc%3Fid%3D1LFV8WULKTMciQ2_akvlhcV9X9ZRII274%26export%3Ddownload

1.2 Table Schemas

Post

Relational Model: Post(post_ID, poster_ID, title, content, post_date)

Functional Dependencies:

post_ID → poster_ID, title, content, post_date

Candidate Keys: {(post_ID)}

Primary Key: {(post_ID)}

Foreign Keys: poster_ID references User(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Post (
        post_ID                 INT NOT NULL AUTO_INCREMENT,
        title                   VARCHAR(255) NOT NULL,
        content                 TEXT NOT NULL,
        post_date               DATE NOT NULL,
        FOREIGN KEY(poster_ID) REFERENCES User(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY(post_id)
);
```

Reply

Relational Model: Reply(post_ID, discriminator_ID, replier_ID, date, expert_verify_status, content)

Functional Dependencies:

post_ID, discriminator_ID → replier_ID, date, expert_verify_status, content

Candidate Keys: {(post_ID, discriminator_ID)}

Primary Key: {(post_ID, discriminator_ID)}

Foreign Keys: post_ID references Post(post_ID), replier_ID references User(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Reply(
        post_ID                 INT NOT NULL,
        discriminator_ID        INT NOT NULL AUTO_INCREMENT,
        date                    DATE NOT NULL,
        expert_verify_status    BOOLEAN,
        content                 TEXT NOT NULL,
        FOREIGN KEY(post_ID) REFERENCES Post(post_ID)
                ON DELETE CASCADE,
        FOREIGN KEY(replier_ID) REFERENCES User(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (post_ID, discriminator_ID),
);
```

User

Relational Model: User(user_ID, e-mail, password, name, phone_number, verification_code)

Functional Dependencies:

user_ID → e-mail, password, name, phone_number, verification_code

e-mail → user_ID, password, name, phone_number, verification_code

phone_number → user_ID, e-mail, password, name, verification_code

Candidate Keys: {(user_ID), (e-mail), (phone_number)}

Primary Key: {(user_ID)}

Foreign Keys: None

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS User(
        user_ID                 INT NOT NULL AUTO_INCREMENT,
        password                VARCHAR(255) NOT NULL,
        name                    VARCHAR(255) NOT NULL,
        verification_code       VARCHAR(4),
        phone_number            VARCHAR(20) NOT NULL,
        e-mail                  VARCHAR(255) NOT NULL,
        UNIQUE(e-mail),
        UNIQUE(phone_number),
        PRIMARY KEY (user_ID)
);
```

Document

Relational Model: Document(document_ID, user_ID, title, content, verification_status)

Functional Dependencies:

document_ID → user_ID, title, content, verification_status

Candidate Keys: {(document_ID)}

Primary Key: {(document_ID)}

Foreign Keys: user_ID references User(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Document(
        document_ID             INT NOT NULL AUTO_INCREMENT,
        title                   VARCHAR(255) NOT NULL,
        content                 TEXT NOT NULL,
        verification_status     VARCHAR(50) NOT NULL,
        FOREIGN KEY(user_ID) REFERENCES User(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (document_ID)
);
```

Shelter

Relational Model: Shelter(user_ID, description, street, district, apartment_no, city, zip, country)

Functional Dependencies:

user_ID → description, street, district, apartment_no, city, zip, country

Candidate Keys: {(user_ID)}

Primary Key: {(user_ID)}

Foreign Keys: user_ID references User(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Shelter(
        user_ID                 INT NOT NULL,
        description             TEXT,
        street                  VARCHAR(255) NOT NULL,
        district                VARCHAR(255) NOT NULL,
        apartment_no            VARCHAR(20) NOT NULL,
        city                    VARCHAR(255) NOT NULL,
        zip                     VARCHAR(20) NOT NULL,
        country                 VARCHAR(255) NOT NULL,
        FOREIGN KEY(user_ID) REFERENCES User(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (user_ID)
);
```

Administrator

Relational Model: Administrator(user_ID, employee_ID)

Functional Dependencies:

user_ID → employee_ID

Candidate Keys: {(user_ID)}

Primary Key: {(user_ID)}

Foreign Keys: user_ID references User(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Administrator(
        user_ID                 INT NOT NULL,
        employee_ID             INT NOT NULL,
        FOREIGN KEY(user_ID) REFERENCES User(user_ID)
            ON DELETE CASCADE,
        PRIMARY KEY (user_ID)
);
```

Adopter

Relational Model: Adopter(user_ID, description, age, sex, balance, profile_picture_path, species, breed, adoption_age, neuter_status, adoption_sex)

Functional Dependencies:

user_ID → description, age, sex, balance, profile_picture_path, species, breed, adoption_age, neuter_status, adoption_sex

Candidate Keys: {(user_ID)}

Primary Key: {(user_ID)}

Foreign Keys: user_ID references User(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Adopter(
        user_ID                 INT NOT NULL,
        description             TEXT,
        age                     INT,
        sex                     VARCHAR(10),
        balance                 DECIMAL(10, 2),
        profile_picture_path    VARCHAR(300),
        species                 VARCHAR(50),
        breed                   VARCHAR(50),
        adoption_age            INT NOT NULL,
        neuter_status           VARCHAR(20),
        adoption_sex            VARCHAR(10),
        FOREIGN KEY(user_ID) REFERENCES User(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (user_ID)
);
```

Veterinarian

Relational Model: Veterinarian(user_ID, year_of_experience, speciality, street, district, apartment_no, city, zip, country)

Functional Dependencies:

user_ID → year_of_experience, speciality, street, district, apartment_no, city, zip, country

Candidate Keys: {(user_ID)}

Primary Key: {(user_ID)}

Foreign Keys: user_ID references User(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Veterinarian(
        user_ID                 INT NOT NULL,
        year_of_experience      INT NOT NULL,
        speciality              VARCHAR(100) NOT NULL,
        street                  VARCHAR(100) NOT NULL,
        district                VARCHAR(100) NOT NULL,
        apartment_no            VARCHAR(50) NOT NULL,
        city                    VARCHAR(50) NOT NULL,
        zip                     VARCHAR(50) NOT NULL,
        country                 VARCHAR(100) NOT NULL,
        FOREIGN KEY(user_ID) REFERENCES User(user_ID)
        ON DELETE CASCADE,
        PRIMARY KEY (user_ID)
);
```

Pet_Care_Info

Relational Model: PetCareInfo(info_ID, administrator_ID, content, title)

Functional Dependencies:

info_ID → administrator_ID, content, title

Candidate Keys: {(info_ID)}

Primary Keys: {(info_ID)}

Foreign Keys: administrator_ID references Administrator(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS PetCareInfo(
        info_ID                 INT NOT NULL AUTO_INCREMENT,
        content                 TEXT NOT NULL,
        title                   VARCHAR(300) NOT NULL
        FOREIGN KEY(administrator_ID) REFERENCES Administrator(user_ID)
            ON DELETE CASCADE,
        PRIMARY KEY (info_ID)
);
```

Pet

Relational Model: Pet(pet_ID, shelter_ID, adopter_ID, species, breed, age, neutered_status, sex, description, name, vaccination_status, house_trained_status, adoption_status, adoption_fee)
Functional Dependencies:
pet_ID → shelter_ID, adopter_ID, species, breed, age, neutered_status, sex, description, name, vaccination_status, house_trained_status, adoption_status, adoption_fee
Candidate Keys: {(pet_ID)}
Primary Keys: {(pet_ID)}
Foreign Keys: shelter_ID references Shelter(user_ID), adopter_ID references Adopter(user_ID)
Normal Form: BCNF
SQL Definition:

```
CREATE TABLE IF NOT EXISTS Pet(
        pet_ID                  INT NOT NULL AUTO_INCREMENT,
        shelter_ID              INT,
        adopter_ID              INT,
        species                 VARCHAR(150) NOT NULL,
        breed                   VARCHAR(150) NOT NULL,
        age                     INT NOT NULL,
        neutered_status         BOOLEAN,
        sex                     VARCHAR(150) NOT NULL,
        description             TEXT NOT NULL,
        name                    VARCHAR(150) NOT NULL,
        vaccination_status      BOOLEAN,
        house_trained_status    BOOLEAN,
        adoption_status         BOOLEAN,
        adoption_fee            DECIMAL(10, 2) NOT NULL,
        FOREIGN KEY(shelter_ID) REFERENCES Shelter(user_ID)
                ON DELETE SET NULL,
        FOREIGN KEY(adopter_ID) REFERENCES Adopter(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (pet_ID)
);
```

Photo

Relational Model: Photo(pet_ID, photo_path)

Functional Dependencies:

none

Candidate Keys: {(pet_ID, photo_path)}

Primary Keys: {(pet_ID, photo_path)}

Foreign Keys: pet_ID references Pet(pet_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Photo(
        pet_ID                  INT NOT NULL,
        photo_path              TEXT NOT NULL,
        FOREIGN KEY(pet_ID) REFERENCES Pet(pet_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (pet_ID, photo_path)
);
```

Medical_Record

Relational Model: MedicalRecord(record_ID, pet_ID, veterinarian_ID, date, operation)

Functional Dependencies:

record_ID → pet_ID, veterinarian_ID, date, operation

Candidate Keys: {(record_ID)}

Primary Keys: {(record_ID)}

Foreign Keys: pet_ID references Pet(pet_ID), veterinarian_ID references Veterinarian(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS MedicalRecord(
        record_ID               INT NOT NULL AUTO_INCREMENT,
        pet_ID                  INT NOT NULL,
        veterinarian_ID         INT NOT NULL,
        date                    DATE NOT NULL,
        operation               TEXT,
        FOREIGN KEY(pet_ID) REFERENCES Pet(pet_ID)
                ON DELETE CASCADE,
        FOREIGN KEY(veterinarian_ID) REFERENCES Veterinarian(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (record_ID)
);
```

Oversee_Record

Relational Model: OverseeRecord(record_ID, administrator_ID, adopter_ID, date, details, verification_status)

Functional Dependencies:

record_ID → administrator_ID, adopter_ID, date, details, verification_status

Candidate Keys: {(record_ID)}

Primary Keys: {(record_ID)}

Foreign Keys: administrator_ID references Administrator(user_ID), adopter_ID references Adopter(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS OverseeRecord(
        record_ID               INT NOT NULL AUTO_INCREMENT,
        administrator_ID        INT NOT NULL,
        adopter_ID              INT NOT NULL,
        date                    DATE NOT NULL,
        details                 TEXT,
        verification_status     BOOLEAN,
        FOREIGN KEY(administrator_ID) REFERENCES  Administrator(user_ID)
                ON DELETE CASCADE,
        FOREIGN KEY(adopter_ID) REFERENCES Adopter(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (record_ID)
);
```

Appointment

Relational Model: Appointment(adopter_ID, veterinarian_ID, date, approval_status, details)

Functional Dependencies:

adopter_ID, veterinarian_ID → date, approval_status, details

Candidate Keys: {(adopter_ID, veterinarian_ID)}

Primary Keys: {(adopter_ID, veterinarian_ID)}

Foreign Keys: adopter_ID references Adopter(user_ID), veterinarian_ID references Veterinarian(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Appointment(
        adopter_ID              INT NOT NULL,
        veterinarian_ID         INT NOT NULL,
        date                    DATE NOT NULL,
        approval_status         BOOLEAN,
        details                 TEXT,
        FOREIGN KEY(adopter_ID) REFERENCES Adopter(user_ID)
                ON DELETE CASCADE,
        FOREIGN KEY(veterinarian_ID) REFERENCES Veterinarian(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (adopter_ID, veterinarian_ID)
);
```

Apply_Adopt

Relational Model: Apply_Adopt(adopter_ID, pet_ID, administrator_ID, date, pet_ownership, pet_care_experience, housing_situation, adoption_reason, approval_status, admin_remarks)

Functional Dependencies:

adopter_ID, pet_ID → date, pet_ownership, pet_care_experience, housing_situation, adoption_reason, approval_status, admin_remarks

Candidate Keys: {(adopter_ID, pet_ID)}

Primary Keys: {(adopter_ID, pet_ID)}

Foreign Keys: adopter_ID references Adopter(user_ID), pet_ID references Pet(pet_ID), administrator_ID references Administrator(user_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Apply_Adopt(
        adopter_ID              INT NOT NULL,
        pet_ID                  INT NOT NULL,
        administrator_ID        INT,
        date                    DATE NOT NULL,
        pet_ownership           BOOLEAN,
        pet_care_experience     INT,
        housing_situation       TEXT NOT NULL,
        adoption_reason         TEXT NOT NULL,
        approval_status         BOOLEAN,
        admin_remarks           TEXT,
        FOREIGN KEY(adopter_ID) REFERENCES Adopter(user_ID)
                ON DELETE CASCADE,
        FOREIGN KEY(pet_ID) REFERENCES Pet(pet_ID)
                ON DELETE CASCADE,
        FOREIGN KEY(administrator_ID) REFERENCES Administrator(user_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (adopter_ID, pet_ID)
);
```

Meet_Greet

Relational Model: Meet_Greet(adopter_ID, pet_ID, date)

Functional Dependencies:

adopter_ID, pet_ID → date

Candidate Keys: {(adopter_ID, pet_ID)}

Primary Keys: {(adopter_ID, pet_ID)}

Foreign Keys: adopter_ID references Adopter(user_ID), pet_ID references Pet(pet_ID)

Normal Form: BCNF

SQL Definition:

```
CREATE TABLE IF NOT EXISTS Meet_Greet(
        adopter_ID              INT NOT NULL,
        pet_ID                  INT NOT NULL,
        date                    DATE NOT NULL,
        FOREIGN KEY(adopter_ID) REFERENCES Adopter(user_ID)
                ON DELETE CASCADE,
        FOREIGN KEY(pet_ID) REFERENCES Pet(pet_ID)
                ON DELETE CASCADE,
        PRIMARY KEY (adopter_ID, pet_ID)
);
```

1.3 Views

User Login View

Purpose of this view is to store an instance of the User Login without displaying the sensitive data such as password and verification code

```
CREATE VIEW User_Login_View AS
SELECT user_ID, e-mail, name, phone_number
FROM User
```

Administrator with Least Number of Applications View

The view is used to select the administrator with the least number of applications. It is used when an administrator is assigned to a new application.

```
CREATE VIEW Administrator_with_Least_Applications AS
SELECT administrator_ID, COUNT(*) AS num_rows
FROM Apply_Adopt
GROUP BY administrator_ID
ORDER BY num_rows ASC
LIMIT 1;
```

Document without Document ID View

This view is used for concealing the private data of the user. For instance, we can get whether a user is verified or not without directly seeing the private document that he/she uploaded.

```
CREATE VIEW Document_without_Document_ID AS
SELECT user_ID, verification_status
FROM Document
```

1.4 Triggers

Delete Unadopted Pets When Shelter Is Deleted

```
CREATE TRIGGER delete_unadopted_pet
AFTER DELETE OF Shelter
REFERENCING OLD ROW AS orow
BEGIN ATOMIC
        DELETE FROM Pet
        WHERE Pet.adoption_status = false;
```

```
END;
```

Adoption Status Trigger:

```
CREATE TRIGGER change_adoption_status
AFTER UPDATE ON Apply_Adopt on (approval_status)
REFERENCING NEW ROW AS new_row
REFERENCING OLD ROW AS old_row
FOR EACH ROW
BEGIN
UPDATE Pet AS P
SET P.adoption_status = @approval_status
WHERE old_row.pet_ID = P.pet_ID
END;
```

Adoption Fee Payment Trigger

```
CREATE TRIGGER make_fee_payment
AFTER UPDATE ON Apply_Adopt on (approval_status)
REFERENCING NEW ROW AS new_row
REFERENCING OLD ROW AS old_row
FOR EACH ROW
BEGIN
UPDATE Adopter AS AD
SET AD.balance = AD.balance - (SELECT P.adoption_fee FROM Pet as P WHERE
P.pet_ID = new_row.pet_ID)
WHERE AD.user_ID = new_row.adopter_ID
END;
```

Admin Assignment To Adoption Requests Trigger

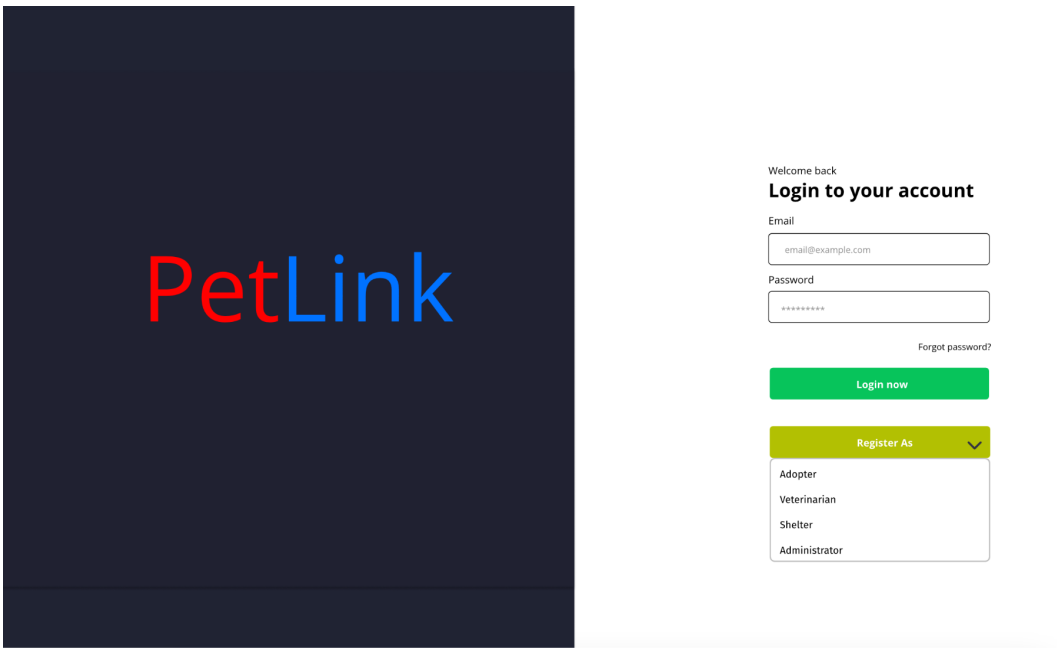
```
CREATE TRIGGER assign_admin
AFTER INSERT ON Apply_Adopt
REFERENCING NEW ROW AS nrow
REFERENCING OLD ROW AS orow
BEGIN ATOMIC
        UPDATE apply_adopt
        SET administrator_id = (SELECT administrator_ID FROM
        Administrator_with_Least_Applications)
        WHERE administrator_id = NULL;
END;
```

2. User Interface Design and Related SQL Statements

Throughout the UI sketches, we included profile pictures in all the sketches. However, it should be considered that only “Adopter” types of users have real profile pictures and remaining roles (Veterinarian, Administrator and Shelter) only have placeholders which aren't stored in the database. This design choice is made with the intention of making every scene more well-kept.

2.1 Common Functionalities

General Login Page

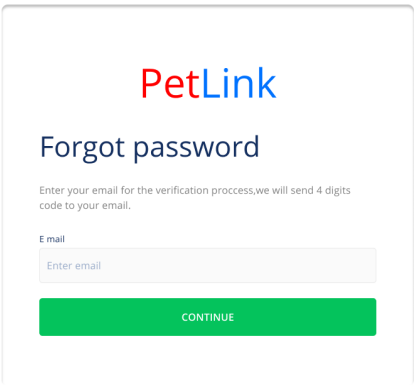


SQL Statements:

To login:

```
SELECT user_ID, name, e-mail, phone_number, verification_code
FROM User
WHERE e-mail = @email AND password = @password;
```

Forgot Password - E-Mail



SQL Statements:

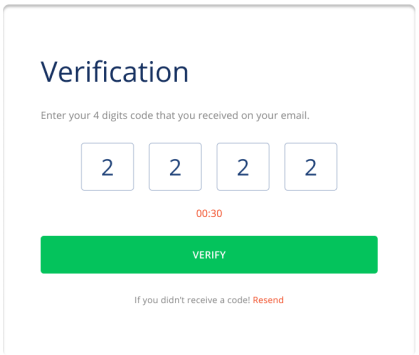
To check whether a user with the given email exists:

```
SELECT COUNT(user_ID)
FROM user
WHERE e-mail = @email;
```

To set the user's verification code to a code generated with internal logic:

```
UPDATE user
SET verification_code = @generatedcode
WHERE e-mail =@email;
```

Forgot Password - Verification Code

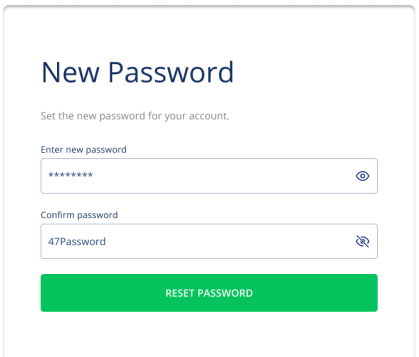


SQL Statements:

To verify whether the given code matches the code sent to user:

SELECT user_ID, e-mail
FROM user
WHERE e-mail = @email AND verification_code = @user_code;

Forgot Password - Reset Password

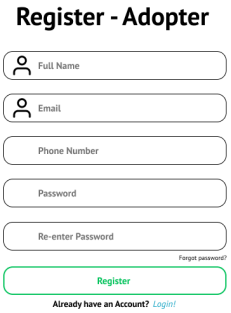


SQL Statements:

To update user's password:
UPDATE user
SET password = @new_password
WHERE e-mail = @email;

To clear the user's verification code so it cannot be reused:
UPDATE user
SET verification_code = NULL
WHERE e-mail = @email;

Register - Adopter



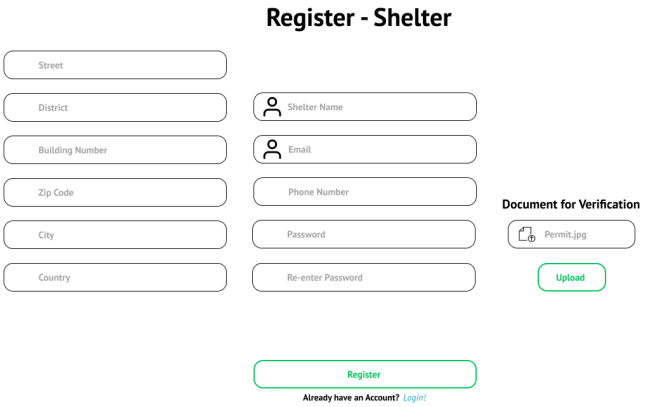
SQL Statements:

To add the User:

INSERT INTO User(user_ID, e-mail, password, name, phone_number, verification_code)
VALUES(@generatedID, @email, @password, @fullname, @phonenumber, NULL);

To add the Adopter:
INSERT INTO Adopter(user_ID, description, age, sex, balance, profile_picture_path, species, breed, adoption_age, neuter_status, adoption_sex)
VALUES(@generatedID, NULL, NULL, NULL, 0, NULL, NULL, NULL, NULL, NULL, NULL);

Register - Shelter



SQL Statements:
To add the User:

INSERT INTO User(user_ID, e-mail, password, name, phone_number, verification_code)
VALUES(@generatedID, @email, @password, @sheltername, @phonenumber, NULL);

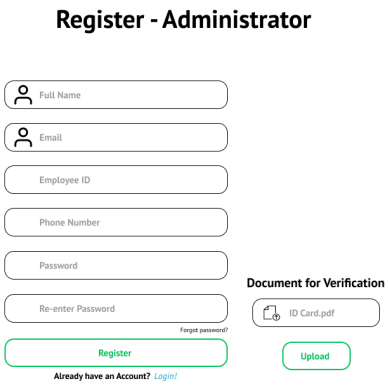
To add the Shelter:

INSERT INTO Shelter(user_ID, description, street, district, apartment_no, city, zip, country)
VALUES(@generatedID, NULL, @street, @district, @buildingnumber, @city, @zipcode, @country);

To update the Document:

INSERT INTO Document(document_ID, title, content, verification_status)
VALUES(@generatedID, @documenttitle, @documentcontent, NULL);

Register - Administrator



SQL Statements:
To add the User:

INSERT INTO User(user_ID, e-mail, password, name, phone_number, verification_code)
VALUES(@generatedID, @email, @password, @fullname, @phonenumber, NULL);

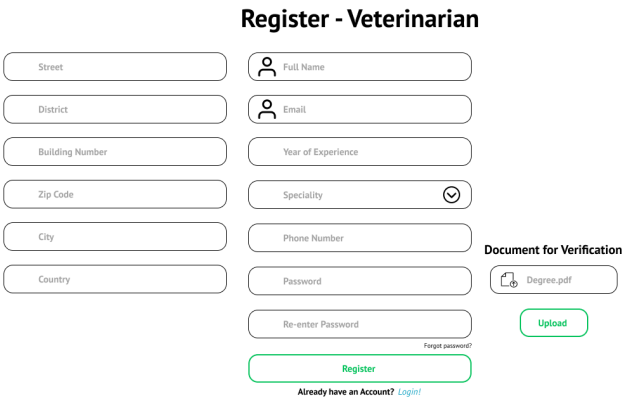
To add the Administrator:

INSERT INTO Administrator(user_ID, employee_ID)
VALUES(@generatedID, @employeeID);

To update the Document:

INSERT INTO Document(document_ID, title, content, verification_status)
VALUES(@generatedID, @documenttitle, @documentcontent, NULL);

Register - Veterinarian



SQL Statements:
To add the User:

INSERT INTO User(user_ID, e-mail, password, name, phone_number, verification_code)
VALUES(@generatedID, @email, @password, @fullname, @phonenumber, NULL);

To add the Veterinarian:

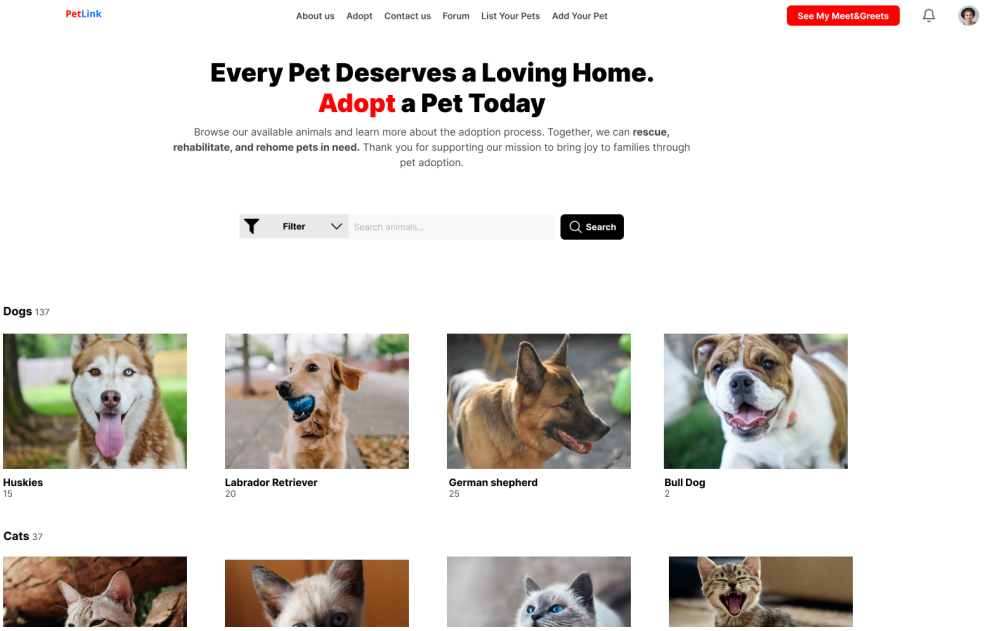
INSERT INTO Veterinarian(user_ID, year_of_experience, speciality, street, district, apartment_no, city, zip, country)
VALUES(@generatedID, @yearofexperience, @speciality, @street, @district, @buildingnumber, @city, @zipcode, @country);

To update the Document:

INSERT INTO Document(document_ID, title, content, verification_status)
VALUES(@generatedID, @documenttitle, @documentcontent, NULL);

General Home Page - User Specific

Adopter Home Page



SQL Statements:

To list dog breeds available to adopt:

```
SELECT Ph.photo_path, P.breed, COUNT(DISTINCT P.pet_ID) AS count_in_website
FROM Pet P NATURAL JOIN Photo Ph
WHERE species = 'Dog' AND adoption_status = false
GROUP BY P.breed
ORDER BY P.breed
LIMIT 4;
```

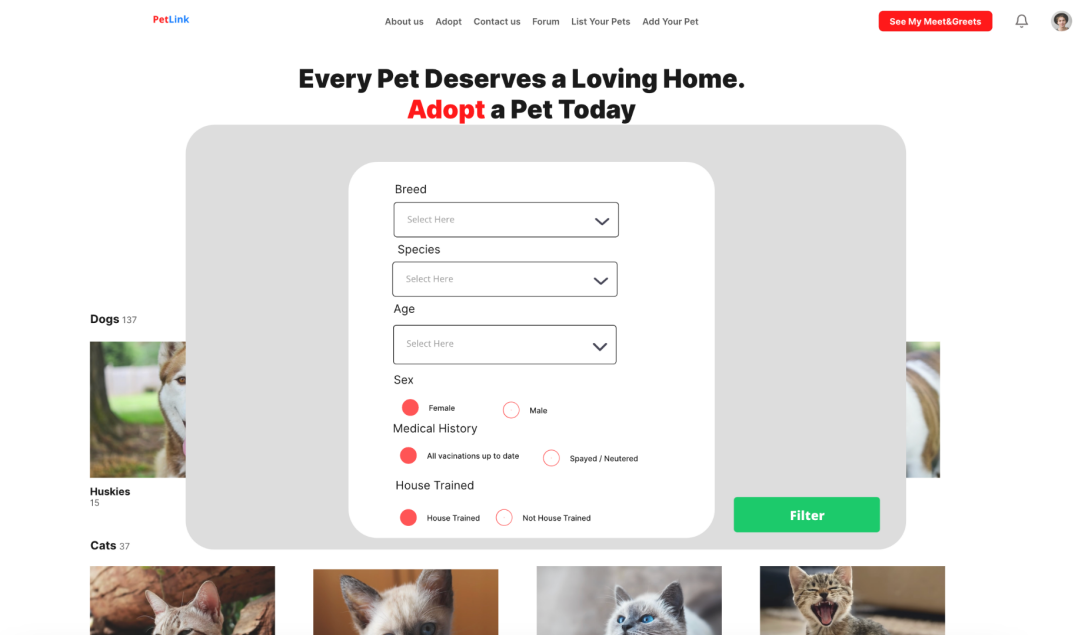
To list cat breeds available to adopt:

```
SELECT Ph.photo_path, P.species, COUNT(DISTINCT P.pet_ID) AS count_in_website
FROM Pet P NATURAL JOIN Photo Ph
WHERE species = 'Cat' AND adoption_status = false
GROUP BY P.breed
ORDER BY P.breed
LIMIT 4;
```

To show user's profile picture:

```
SELECT profile_picture_path
FROM adopter
WHERE user_ID =  @user_ID
```

Adopter Filtering Preferences



SQL Statements:

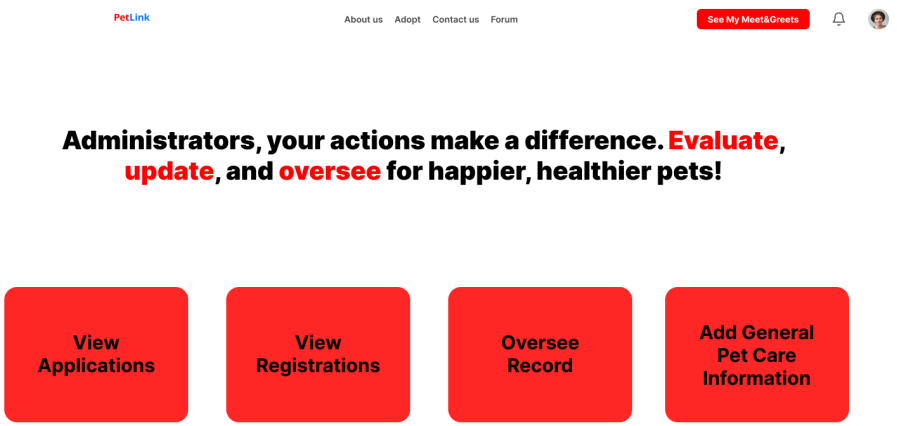
To list animal results based on search criteria:

```
SELECT pet_ID, species, breed, name, age
FROM Pet
WHERE (CASE WHEN @filter-gender = 1 THEN gender END = @chosen-gender
        AND CASE WHEN @filter-neuter-status = 1 THEN neuter_status END =
    @chosen-neuter
    AND CASE WHEN @filter-vaccination-status = 1 THEN vaccination_status END =
    @chosen-vaccination
    AND CASE WHEN @filter-house-trained-status = 1 THEN house_trained_status
    END = @chosen-house-trained
    AND CASE WHEN @filter-sex = 1 THEN sex END = @chosen-sex);
```

To show user's profile picture:

```
SELECT profile_picture_path
FROM adopter
WHERE user_ID =  @user_ID
```

Administrator Home Page

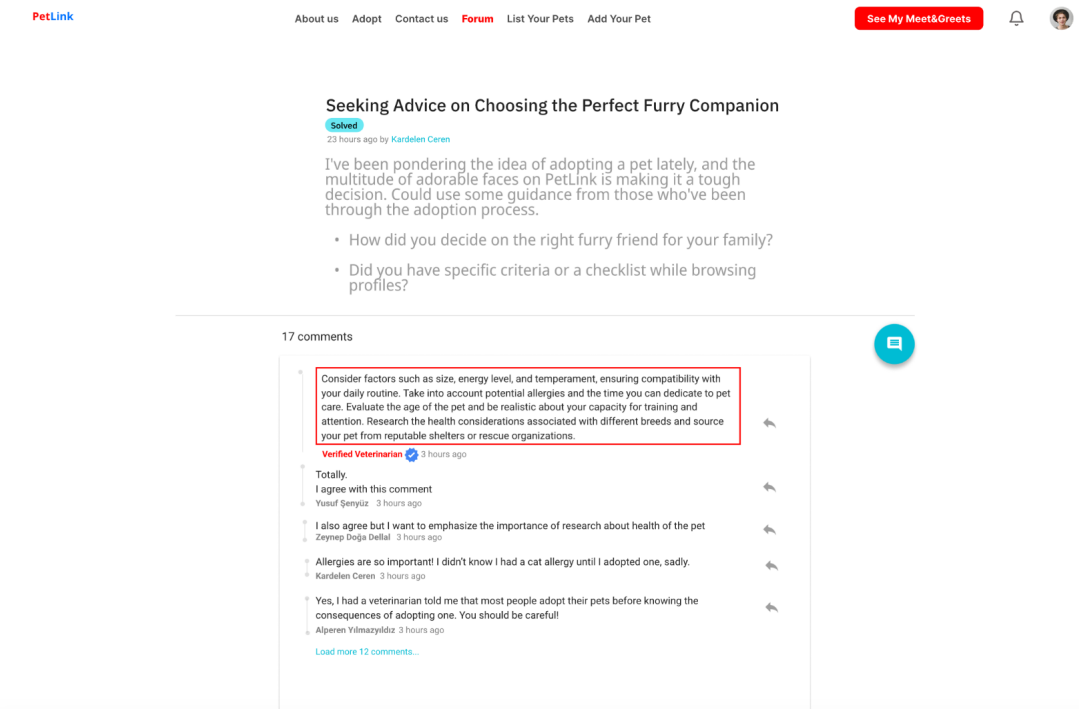


This page is designed for administrators to navigate to various sections. Therefore, no SQL statements are needed.

2.3 Topic Specific Functionality

2.3.1 Forum Pages

Post Page



SQL Statements:

To show the post's title, contents, replies, number of replies:

```
SELECT P.title, P.date, P.content, U.name, R.content, R.date, R.expert_verify_status
FROM (Post P
NATURAL JOIN Reply R)
JOIN User_Login_View U ON P.poster_ID = U.user_ID
WHERE P.post_ID = @post_ID
```

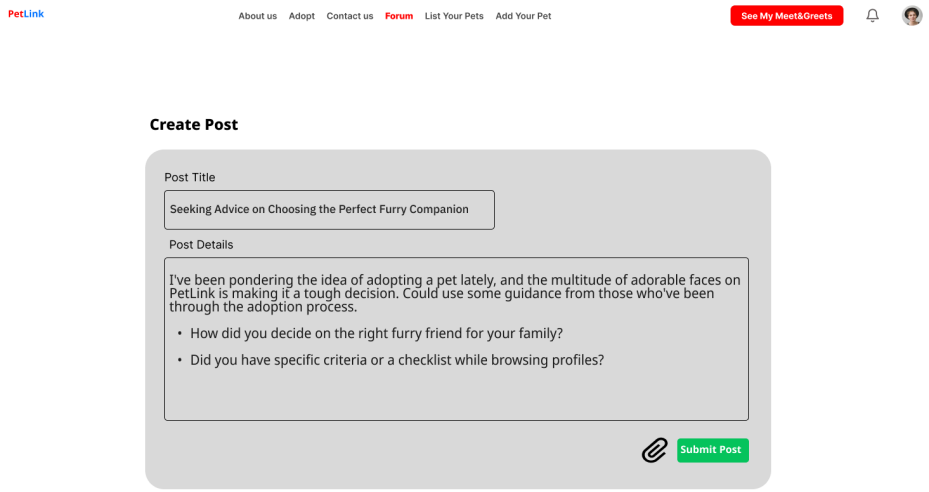
To show the number of replies:

```
SELECT COUNT(*) as reply_count
FROM Post P
NATURAL JOIN Reply R
WHERE P.post_ID = @post_ID
```

To reply to a post:

```
INSERT INTO Reply(date, expert_verify_status, content)
VALUES(GETDATE(), CASE WHEN EXISTS (SELECT 1 FROM Veterinarian WHERE
user_ID = @user_ID) THEN 1 ELSE 0 END, @content)
```

Create Post Page

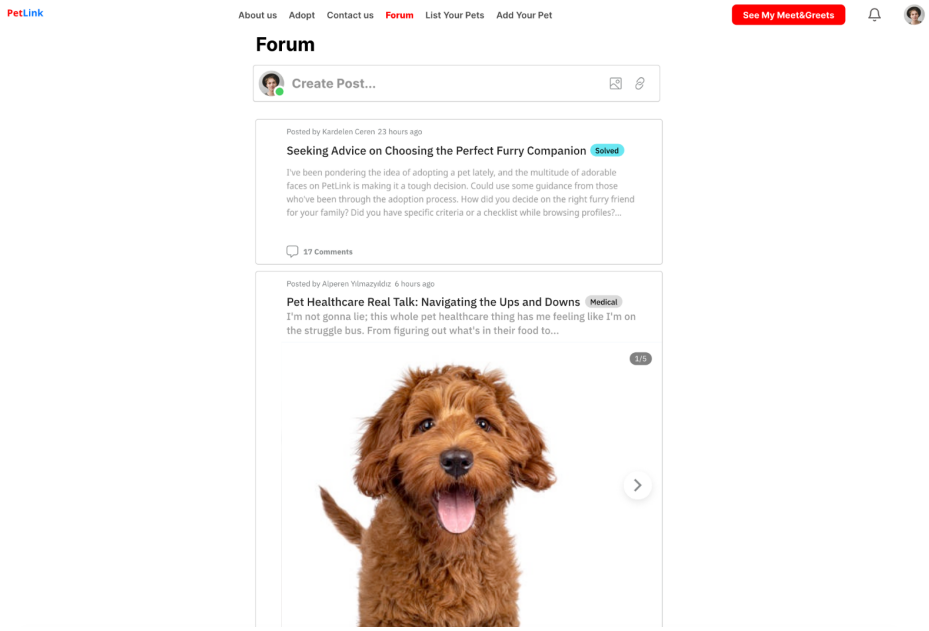


SQL Statements:

To add a new post:

INSERT INTO Post(post_ID, poster_ID, title, content, post_date)
VALUES(@post_ID, @user_ID, @title, @content, GETDATE())

Forum Main Page



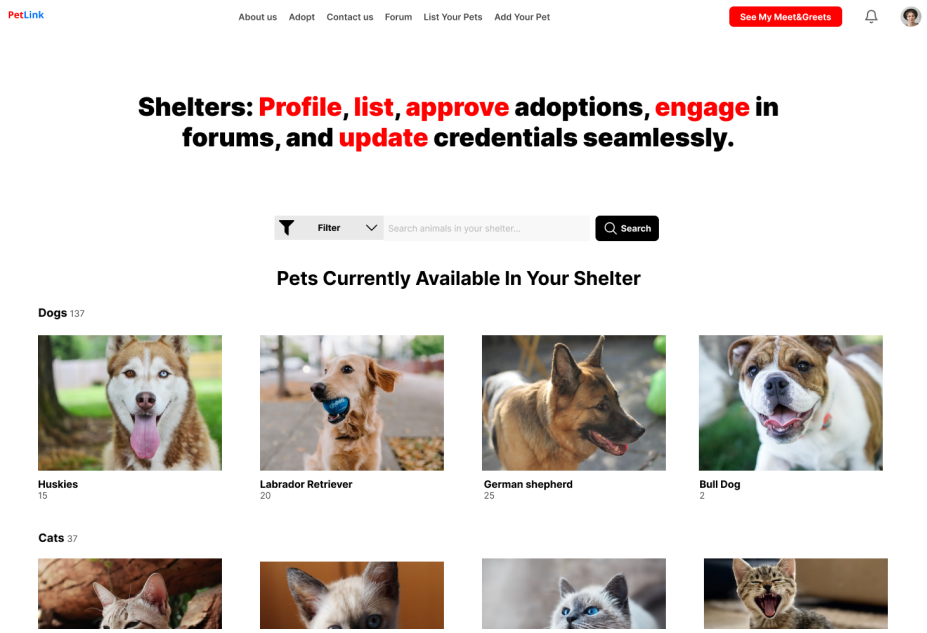
SQL Statements:

To list posts, poster's user name and their reply number:

```
SELECT P.title, P.date, P.content, U.name, COUNT(R.discriminator_ID) as reply_count
FROM (Post P
NATURAL JOIN Reply R)
JOIN User_Login_View U ON P.poster_ID = U.user_ID
GROUP BY P.post_ID, P.title, P.date, P.content, U.name
ORDER BY P.date DESC
```

2.3.2 Pet Adoption Pages

Shelter Home Page



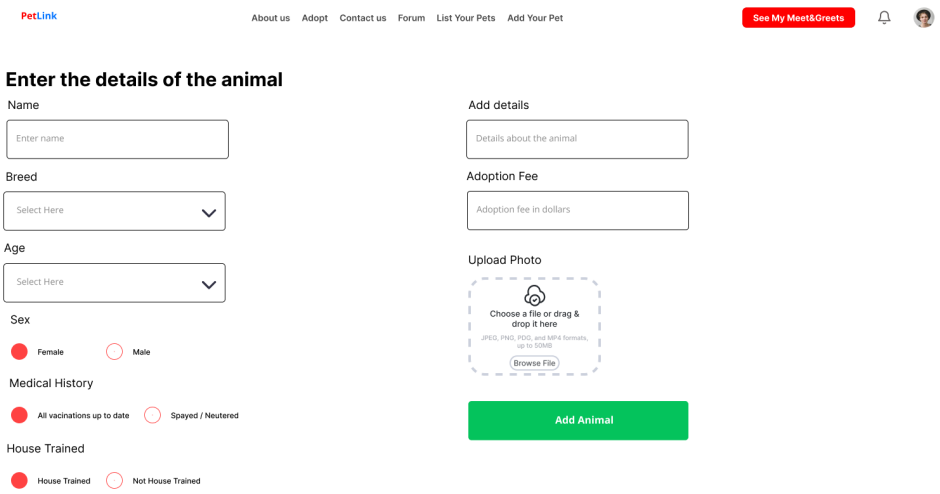
SQL Statements:

To search animals based on filtering for shelters:

```
SELECT *
FROM Pet
WHERE (CASE WHEN @filter-gender = 1 THEN gender END = @chosen-gender
        AND CASE WHEN @filter-neuter-status = 1 THEN neuter_status END =
@chosen-neuter
        AND CASE WHEN @filter-vaccination-status = 1 THEN vaccination_status END =
                @chosen-vaccination
AND CASE WHEN @filter-house-trained-status = 1 THEN house_trained_status END =
@chosen-house-trained
        AND CASE WHEN @filter-adoption-status = 1 THEN adoption_status END =
                @chosen-adoption
```

AND CASE WHEN @filter-sex = 1 THEN adoption_status END =
@chosen-sex
AND CASE WHEN @filter-adoption-status = 1 THEN adoption_status END =
@chosen-adoption);

Animal Adding Page For Shelters



SQL Statements:

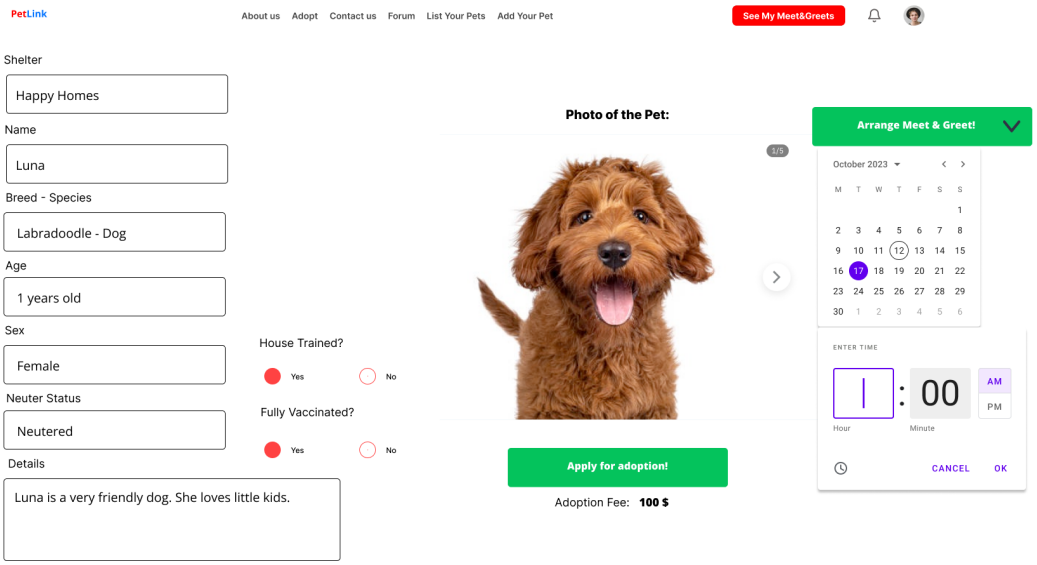
To add new pet:

INSERT INTO Pet(pet_ID, shelter_ID, adopter_ID, species, breed, age, neutered_status, sex, description, name, vaccination_status, house_trained_status, adoption_status, adoption_fee)
VALUES (@pet_ID, @user_ID, NULL, @species, @breed, @age, @neuter_status, @sex, @description, @name, @vaccination_status, @house_trained_status, 0, @adoption_fee);

To add pet's photos (one statement for each photo):

INSERT INTO Photo(pet_ID, photo_path)
VALUES(@pet_ID, @photo_path)

Animal Details Page



SQL Statements:

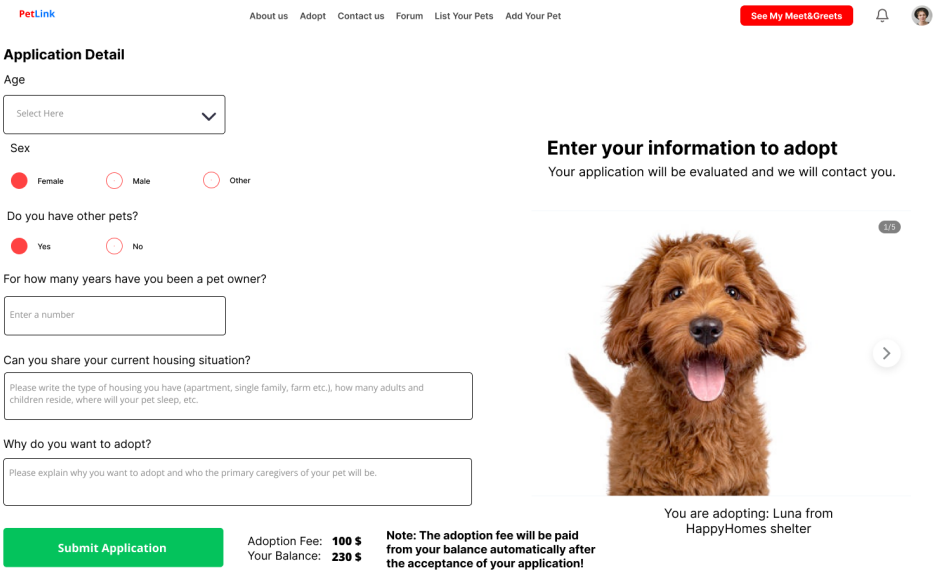
To arrange meet and greet:

```
INSERT INTO Meet_Greet(adopter_ID, pet_ID, date)
VALUES(@adopter_ID, @pet_ID, @meet_date)
```

To display pet information:

```
SELECT *
FROM Pet NATURAL JOIN Photo
WHERE pet_ID = @pet_ID
```

Application Submit Page For Adopters



SQL Statements:

To show user's age, sex and balance:

This is done in order to be updated if any of these attributes of an Adopter changes in any time.

```
SELECT age, sex, balance
FROM adopter
WHERE user_ID = @user_ID;
```

To show pet's name, adoption fee, shelter name, and photos:

```
SELECT P.name, P.adoption_fee, U.name, Ph.photo_path
FROM pet P
NATURAL JOIN photo Ph
JOIN User_Login_View U ON P.shelter_ID = U.user_ID
WHERE P.pet_ID = @pet_ID ;
```

To update user's age and sex:

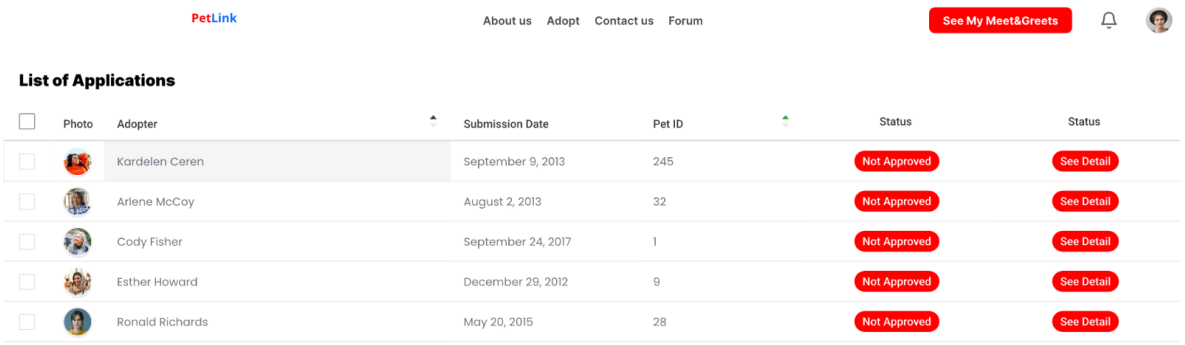
```
UPDATE user
SET age = @age AND sex = @sex
WHERE user_ID = @user_ID;
```

To submit an application:

INSERT INTO Apply_Adopt(adopter_ID, pet_ID, administrator_ID, date, pet_ownership, pet_care_experience, housing_situation, adoption_reason, approval_status, admin_remarks)

VALUES(@user_ID, @pet_ID, NULL, GETDATE(), @pet_ownership, @pet_care_experience, @housing_situation, @adoption_reason, 0, NULL)

Application List Page For Administrators

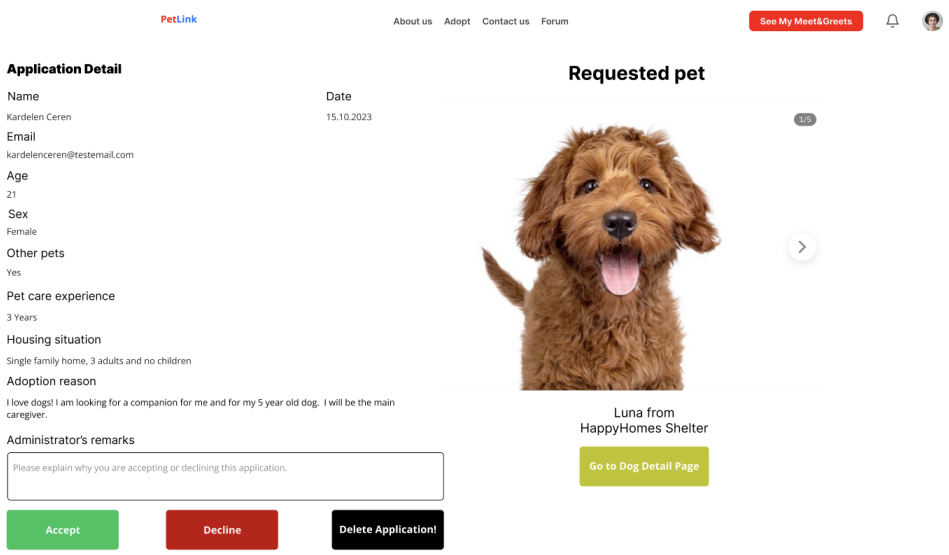


SQL Statements:

To list applications:

SELECT A.profile_picture_path, U.name, AA.date, AA.pet_ID, AA.approval_status
FROM Apply_Adopt AA
JOIN User_Login_View U ON U.user_ID = AA.adopter_ID
JOIN Adopter A ON U.user_ID = A.user_ID
WHERE AA.administrator_ID = @user_ID;

Application Details Page For Adopters



SQL Statements:

To show user's name, email, age, sex, application date, other pets, pet care experience, housing situation and adoption reason:

```
SELECT U.name, U.email, A.age, A.sex, App.date, App.pet_ownership,
App.pet_care_experience, App.housing_situation, App.adoption_reason
FROM adopter A
NATURAL JOIN User_Login_View U
JOIN apply_adopt App ON U.user_ID = App.adopter_ID
WHERE App.adopter_ID = @adopter_ID AND App.pet_ID = @pet_ID;
```

To show pet's name, shelter name and photos:

```
SELECT P.name, U.name, Ph.photo_path
FROM pet P
NATURAL JOIN photo Ph
JOIN User_Login_View U ON P.shelter_ID = U.user_ID
WHERE P.pet_ID = @pet_ID ;
```

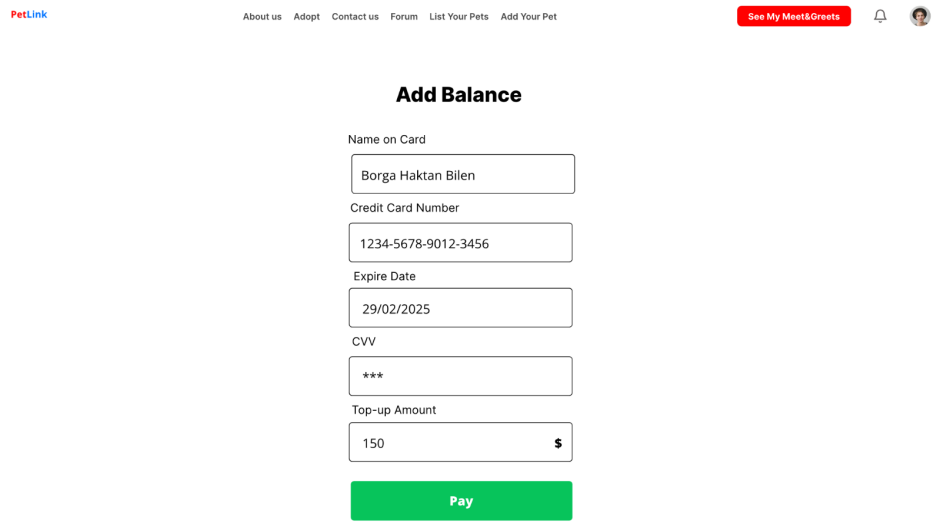
To update the application's status and remarks:

```
UPDATE apply_adopt
SET approval_status = @approval_status AND admin_remarks = @admin_remarks
WHERE  App.adopter_ID = @adopter_ID AND App.pet_ID = @pet_ID;
```

Delete application:

```
DELETE from apply_adopt
WHERE adopter_ID = @adopter_ID AND pet_ID = @pet_ID;
```

Add Balance Page



SQL Statements:

To add balance:

```
UPDATE adopter
SET balance = balance + @top_up_amount
WHERE user_ID = @user_ID;
```

3. Implementation Plan

We will use the MySQL database system as our database management system, taking advantage of its broad support for the latest features required for our project. The backend of the application will be built with the Python Flask framework, while the frontend will be built with the React.JS framework and TypeScript. We'll integrate the Tailwind CSS framework and employ Material UI components to improve the appearance and user experience. Notably, all database-application interactions will be limited to SQL queries, with no automation tools or libraries such as ORM used.