Solution: Updating and Deleting Pets

In this lesson, we will learn how to perform the update and delete operations on the pets in our application.

WE'LL COVER THE FOLLOWING

Solution

```
"""Flask Application for Paws Rescue Center."""
from flask import Flask, render_template, abort
from forms import SignUpForm, LoginForm, EditPetForm
from flask import session, redirect, url_for
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SECRET_KEY'] = 'dfewfew123213rwdsgert34tgfd1234trgf'
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///paws.db'
db = SQLAlchemy(app)

"""Model for Pets."""
class Pet(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String, unique=True)
    age = db.Column(db.String)
    bio = db.Column(db.String)
    posted_by =  db.Column(db.String, db.ForeignKey('user.id'))


"""Model for Users."""
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    full_name = db.Column(db.String)
    email = db.Column(db.String, unique=True)

    password = db.Column(db.String)
    pets = db.relationship('Pet', backref = 'user')

db.create_all()

# Create "team" user and add it to session
team = User(full_name = "Pet Rescue Team", email = "team@petrescue.co", password = "adminpass
db.session.add(team)

# Create all pets
nelly = Pet(name = "Nelly", age = "5 weeks", bio = "I am a tiny kitten rescued by the good pe
yuki = Pet(name = "Yuki", age = "8 months", bio = "I am a handsome gentle-cat. I like to dres
basker = Pet(name = "Basker", age = "1 year", bio = "I love barking. But, I love my friends m
mrfurrkins = Pet(name = "Mr. Furrkins", age = "5 years", bio = "Probably napping.")

# Add all pets to the session
db.session.add(nelly)
db.session.add(yuki)
db.session.add(basker)
db.session.add(mrfurrkins)

# Commit changes in the session
try:
    db.session.commit()
except Exception as e:
    db.session.rollback()
finally:
    db.session.close()

@app.route("/")
def homepage():
    """View function for Home Page."""
    pets = Pet.query.all()
    return render_template("home.html", pets = pets)


@app.route("/about")
def about():
    """View function for About Page."""
    return render_template("about.html")


@app.route("/details/<int:pet_id>", methods=["POST", "GET"])
def pet_details(pet_id):
    """View function for Showing Details of Each Pet."""
    form = EditPetForm()
    pet = Pet.query.get(pet_id)
    if pet is None:
        abort(404, description="No Pet was Found with the given ID")
    if form.validate_on_submit():
        pet.name = form.name.data
        pet.age = form.age.data
        pet.bio = form.bio.data
        try:
            db.session.commit()
        except Exception as e:
            db.session.rollback()
            return render_template("details.html", pet = pet, form = form, message = "A Pet w
    return render_template("details.html", pet = pet, form = form)
```

```
@app.route("/delete/<int:pet_id>")
def delete_pet(pet_id):
    pet = Pet.query.get(pet_id)
    if pet is None:
        abort(404, description="No Pet was Found with the given ID")
    db.session.delete(pet)
    try:
        db.session.commit()
    except Exception as e:
        db.session.rollback()
    return redirect(url_for('homepage', _scheme='https', _external=True))

@app.route("/signup", methods=["POST", "GET"])
def signup():
    """View function for Showing Details of Each Pet."""
    form = SignUpForm()
    if form.validate_on_submit():
        new_user = User(full_name = form.full_name.data, email = form.email.data, password =
        db.session.add(new_user)
        try:
            db.session.commit()
        except Exception as e:
            print(e)
            db.session.rollback()
            return render_template("signup.html", form = form, message = "This Email already
        finally:
            db.session.close()
        return render_template("signup.html", message = "Successfully signed up")
    return render_template("signup.html", form = form)


@app.route("/login", methods=["POST", "GET"])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(email = form.email.data, password = form.password.data).
        if user is None:
            return render_template("login.html", form = form, message = "Wrong Credentials. 
        else:
            session['user'] = user.id
            return render_template("login.html", message = "Successfully Logged In!")
    return render_template("login.html", form = form)

@app.route("/logout")
def logout():
    if 'user' in session:
        session.pop('user')
    return redirect(url_for('homepage', _scheme='https', _external=True))

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0", port=3000)
```

Explanation

Let’s break down the solution of this challenge to figure out how we solved it.

Update

For the **update operation**, we made the following modifications in the application.

Modifications in `forms.py`

In `forms.py`, in **lines 20 to 24**, we created a new form called `EditPetForm`. The fields of this form correspond to the columns of the `Pet` model with the addition of only the `submit` field. We have also added the `InputRequired()` validator for the input fields so that empty values are not stored in the database.

Modifications in `app.py`

In `app.py`, we first imported the `EditPetForm` in **line 3**. Then, we created an object of this class in the `pet_details` view and passed it to the `render_template()` function so that we could render it in the template. Now, to handle the scenario of getting back data from the template, we used an `if` condition on `form.validate_on_submit()` in **line** 75. Then, if this condition returned `true`, we modified the `pet`. Afterward, we committed the modified `session` in **line 80**. However, we have a constraint on the `name` of the `pet` that only allows **unique** values. Therefore, the `commit()` function can raise an `Exception`. In the case of an `Exception`, we `rollback()` and return the `message` saying, *“A pet with this name already exists!”*

Modifications in `details.html`

In `details.html`, we used Jinja syntax to render the `form` in **lines 26 to 38**. Also, we added an `<h2>` heading for the form and an `if` condition to show the `message` if it is available in lines **16 to 23**.

Deletion

For the **delete operation**, we made the following modifications in the application.

Modifications in `details.html`

We added a **Delete Pet** button in **line 8** of `details.html`. This button redirects to the location: `url_for('delete_pet', pet_id = pet.id)`. We haven’t yet explained the creation of the entry point `delete_pet`, so let’s do that next.

Modifications in `app.py`

Finally, in `app.py`, we created a new view function in **line 86** called `deleet_pet`. This function has a dynamic rule `/delete/<int:pet_id>` where `pet_id` is a variable that determines which **pet** should be deleted. In **line 88**, we retrieve the `pet` with `id` equal to `pet_id`. If such a `pet` does not exist, `abort()` is called with a `404` error. Otherwise, this `pet` is deleted from the database using `db.session.delete()` in **line 91**. Then, the changes are committed in a `try-except` block to handle any exceptions. Afterward, we used `redirect()` to redirect to the `homepage` view in **line 96**.

Congratulations! You have solved all the challenges in this course.