Solution: Retrieving a User at Login

In this lesson, we will be going over the solution of how we can modify the login method so that it uses the database.

WE'LL COVER THE FOLLOWING ^

- Solution
- Explanation

Solution

```
"""Flask Application for Paws Rescue Center."""
from flask import Flask, render_template, abort
from forms import SignUpForm, LoginForm
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)
```

```
# Commit changes in the session
try:
    db.session.commit()
except Exception as e:
    db.session.rollback()
finally:
    db.session.close()
"""Information regarding the Pets in the System."""
pets = [
            {"id": 1, "name": "Nelly", "age": "5 weeks", "bio": "I am a tiny kitten rescued by
            {"id": 2, "name": "Yuki", "age": "8 months", "bio": "I am a handsome gentle-cat.
            {"id": 3, "name": "Basker", "age": "1 year", "bio": "I love barking. But, I love
            {"id": 4, "name": "Mr. Furrkins", "age": "5 years", "bio": "Probably napping."},
@app.route("/")
def homepage():
    """View function for Home Page."""
    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>")
def pet_details(pet_id):
    """View function for Showing Details of Each Pet."""
    pet = next((pet for pet in pets if pet["id"] == pet_id), None)
    if pet is None:
        abort(404, description="No Pet was Found with the given ID")
    return render_template("details.html", pet = pet)
@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 = next((user for user in users if user["email"] == form.email.data and user["g
```

Explanation

Let's break down the steps to solve this challenge.

- 1. Previously, in the login view at line 96, we were searching the list for the user with the provided credentials. This was the part that we had to replace first.
- 2. Therefore, in line 97, we used User.query.filter_by() to query for an object with email = form.email.data and password = form.password.data. Then we chain the query with the first() method to retrieve the first result from the query.

```
user = User.query.filter_by(email = form.email.data, password = form.passw
ord.data).first()
```

- 3. If such a user exists, we will not get a None value in the user variable. As we had already placed this check, we did not have to change that logic.
- 4. For the next task, we had to replace **line 101**. Instead of directly putting the whole object in the session, we instead used user.id as a value for session['user']

```
You might be thinking: why did we not directly store the user object in the session variable?

That is a perfectly valid question. The reason is that this object is not JSON serializable and we can not use such objects in the session
```

dictionary.

We have finally gotten rid of the users list that we were using prior. Hurrah!

In the next few challenges, we will be working on the Pet model and modifying our application.