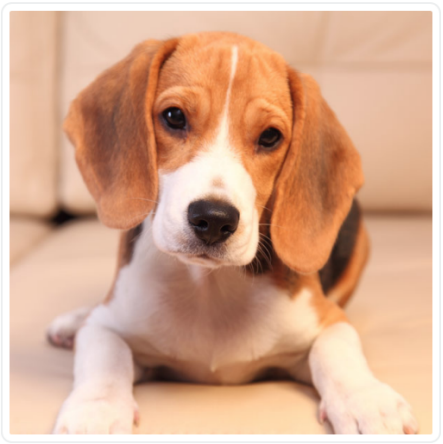
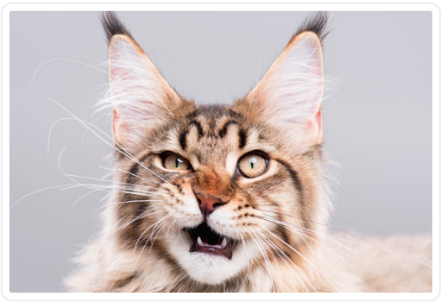
Adoption Agency

Pets



Woofly **is available!**

Porchetta **is available!**



Snargle **is available!**

Add a Pet

<_images/screen.png>

In this exercise, you'll build a web application for a Pet Adoption Agency.

Step 1: Create Database & Model

Create a Flask and Flask-SQLAlchemy project, "adopt".

Create a single model, ***Pet***. This models a pet potentially available for adoption:

- ***id***: auto-incrementing integer
- ***name***: text, required
- ***species***: text, required
- ***photo_url***: text, optional
- ***age***: integer, optional
- ***notes***: text, optional
- ***available***: true/false, required, should default to available

While setting up the project, add the Debug Toolbar.

Step 2: Make Homepage Listing Pets

The homepage (at route `/`) should list the pets:

- name

- show photo, if present
- display “Available” in bold if the pet is available for adoption

Step 3: Create Add Pet Form

Create a form for adding pets. This should use Flask-WTF, and should have the following fields:

- Pet name
- Species
- Photo URL
- Age
- Notes

This should be at the URL path `/add`. Add a link to this from the homepage.

Step 4: Create Handler for Add Pet Form

This should validate the form:

- if it doesn’t validate, it should re-render the form
- if it does validate, it should create the new pet, and redirect to the homepage

This should be a POST request to the URL path `/add`.

Step 5: Add Validation

WTForms gives us lots of useful validators; we want to use these for validating our fields more carefully:

- the species should be either “cat”, “dog”, or “porcupine”
- the photo URL must be a URL *(but it should still be able to be optional!)*
- the age should be between 0 and 30, if provided

Step 6: Add Display/Edit Form

Make a page that shows some information about the pet:

- Name
- Species
- Photo, if present
- Age, if present

It should also show a form that allows us to edit this pet:

- Photo URL
- Notes
- Available

This should be at the URL `/[pet-id-number]`. Make the homepage link to this.

Step 7: Handle Edit Form

This should validate the form:

- if it doesn't validate, it should re-render the form
- if it does validate, it should edit the pet

This should be a POST request to the URL path `/[pet-id-number]`.

Step 8: Clean Up Your Code!

A critical step for any project is to refactor and clean up code; it's good to do this iteratively, as you work.

Check for:

- function names: good functions names are "verby", like ***show_add_form***
- consistent use of good variable names
- every class or function should have a docstring describing its purpose
- add comments for any parts that would benefit from this
- add a file showing all of the Python requirements for this project

Further Study

There are some optional steps, if you'd like:

- Add "message flashing" to give feedback after a pet is added/edited
- Divide the homepage into two listings: available pets and no-longer-available pets.
- Add Bootstrap and a simple theme.
- Reduce duplication: you probably have lots of duplicate form code in both the add form and edit form; learn about Jinja2's "include" directive, and figure out how you can factor out that common code.
- **Harder**: in your add-pet route, you are probably extracting each field's data individually to instantiate the new pet. This is a little tedious and also would need to be updated if the add form changed. Given that there is already a dictionary of values from the form, can you instantiate a pet using this more directly?
- **Harder, Requires Research**: add a new field for a photo upload (in addition to the URL field, before). This will need to handle file uploads and then save the file into the ***/static*** directory so it can be served up. Make it so that only one of these two fields can be filled out (if you try to fill out both, you should get a validation error).

Solution

Solution <solution/index.html>