Solution: Create a Dynamic Route for Pet Details

In this lesson, we will take a look at the solution for the "Create a Dynamic Route for Pet Details" challenge.

WE'LL COVER THE FOLLOWING

The complete implementation of the problem is provided below. Let's take a look at it!

Solution

```
"""Flask Application for Paws Rescue Center."""
from flask import Flask, render_template, abort
app = Flask(__name__)

"""Information regarding the Pets in the System."""
pets = [
            {"id": 1, "name": "Nelly", "age": "5 weeks", "bio": "I am a tiny kitten rescued b
            {"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)

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

Explanation

Let's take a look at how we solved this problem.

Modifications in `app.py`

1. We created a new *view* function called `pet_details` in **line 25**.
2. In **line 24**, we can see that the route associated with this view is `"/details/<int:pet_id>"`, where `pet_id` is a **variable rule** and `int` is a **converter**.
3. Due to the converter, we get an `integer` value in the `pet_id` variable passed to the function.
4. At **line 27**, we search the list of dictionaries for the dictionary containing `id == pet_id`.
5. If such a `pet` does not exist, then we call the `abort()` function with a `404` response code, and the `description` equal to *"No Pet was Found with the given ID"*, in **line 29**.
6. Otherwise, if the `pet` was found, we return the dictionary with the `details.html` template.

Modifications in `details.html`

The skeleton of this template is similar to the other templates in the system. Only the following things are different:

1. We added the `pet`'s name in the title of the page in **line 5**.
2. We used two `<div>` tags to create two columns.
3. The left column contains the `<img>` tag with the image of the respective `pet` in **line 10**.
4. The right column contains the `name`, `age` and `bio` as shown in **lines 13 to 15**.

Modifications in `home.html`

Modifications in `home.html`

1. We added an `<a>` tag around the `<img>` tag in each row of pets.

2. For the `href` attribute, we generated the link of the pet using the `url_for()` method. We gave it the endpoint of `pet_details` view function, and we set the value of the `pet_id` variable equal to `pet['id']`.

```
url_for('pet_details', pet_id=pet['id'])
```

In the next challenge, we will apply template inheritance in the project.