Solution: Rendering a Sign-Up Form and Navbar

In this lesson, we will take a look at the solution for rendering a sign-up form using Flask-WTF and also adding a navigation bar to all the templates.

WE'LL COVER THE FOLLOWING

- Solution
- Explanation
  - Modifications in `forms.py`
  - Modifications in `app.py`
  - Modifications in `signup.html`
  - Modifications in `base.html`

Solution

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

app = Flask(__name__)
app.config['SECRET_KEY'] = 'dfewfew123213rwdsgert34tgfd1234trgf'

"""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)


@app.route("/signup", methods=["POST", "GET"])
def signup():
    """View function for Showing Details of Each Pet."""
    form = SignUpForm()
    return render_template("signup.html", form = form)

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 `forms.py`

In `forms.py`, we created a `SignUpForm` class that inherits from `FlaskForm`. The creation of this form was very similar to the `LoginForm` example we covered in the lesson, "*Creating Forms using Flask-WTF and WTForms*". The only differences are that we have two extra fields: `full_name` and `confirm_password`.

- `full_name`: a simple `StringField` with the label *"Full Name"*. It only contains the `InputRequired` validator.
- `confirm_password`: a `PasswordField` with the label *"Confirm Password"*. Moreover, it contains two validators: `InputRequired` and `EqualTo`. We already know the purpose of `InputRequired`; however, `EqualTo` is a new one. This validator makes sure that the field it is applied to has the same value as the field being passed to the validator. Therefore, for the input to be **valid**, the `confirm_password` field should have the same value as the `password` field.

Modifications in `app.py`

At **line 38** of `app.py`, we added a view function called `signup`. It is associated with the route `/signup` and accepts both `GET` and `POST` methods. We created an instance of the `SignUpForm` in this function and returned it to the

`signup.html` template to render. Furthermore, at **line 6**, we also added a

`SECRET_KEY` in the application because we know that we will need it for the `csrf_token` later.

Modifications in `signup.html`

We have inherited the `signup.html` template form the `base.html` template so that it remains consistent with all other templates in the system. Since we have filled all the placeholders with the appropriate values, we can now render the `form` that we just passed from `app.py`. We used the same method as shown in the lesson "*Rendering Flask-WTF Forms in Templates*". The only difference is that we added extra fields here as well.

Modifications in `base.html`

To add a navigation bar in all the templates, we used the `base.html` template because all the **commonalities** should be placed in the ***parent template***. In **lines 9 to 15**, we added a simple horizontal list of links using the `<a>` tags within `<li>` tags. The links were generated using the `url_for()` function against the endpoints for the specified view functions.

In the next lesson, we will build upon this problem and **incorporate data handling** in the sign-up route.