Solution: Sign-Up Form Data Handling

In this lesson, we will take a look at the solution to the challenge presented in the previous lesson.

WE'LL COVER THE FOLLOWING

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."},
        ]

"""Information regarding the Users in the System."""
users = [
            {"id": 1, "full_name": "Pet Rescue Team", "email": "team@pawsrescue.co", "passwor
        ]


@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 = {"id": len(users)+1, "full_name": form.full_name.data, "email": form.emai
        users.append(new_user)
        return render_template("signup.html", message = "Successfully signed up")
    return render_template("signup.html", form = form)

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

Explanation

Let's breakdown the solution of this challenge to figure out how we solved it.

Modifications in `app.py`

We have used the `validate_on_submit()` function of the `FlaskForm` class at **line 47** in the `signup` view. This shortcut function only returns `true` when the data is both **submitted** and **valid**. Therefore, inside the `if` condition, we create a `new_user` dictionary with the data received from the `form`. Then, we `append()` this `new_user` to the `users` list. Afterward, we render the `signup.html` template again, this time without the `form`, but with a `message` variable to indicate success.

Finally, in the case where `validate_on_submit()` returns `false`, we have returned the `signup.html` template, again, with the `form` object.

Modifications in `signup.html`

In the `signup.html` template, we made two major changes:

1. **Display errors:** to display the error messages that indicate validation failure, we used the same method mentioned in the lesson "*Form Validation, Data, and Error Handling with Flask-WTF*". We used a `for` loop in Jinja to print all the errors associated with a field underneath it.

   To give it the feel of an alert, we used the color `tomato` for its text.

2. **Display success message:** as mentioned above, if the `new_user` is added to the `users` list, we send a variable called `message` to indicate success. In **line 13**, we check if this variable was received. Then, we print its value in **line 14** and close the `if` condition.

In the next challenge, we will be dealing with logging in and out.