Project Challenge: Create a Login and Logout Mechanism

In this lesson, we will be authenticating users using login and maintaining a session until they logout.

WE'LL COVER THE FOLLOWING

Problem statement

In this challenge, you are required to add a **login and log out** mechanism in the project application.

In this challenge, we have already provided you with the `LoginForm`, `login.html` template and a simple `login` view function that returns the form to the template.

What are sessions in `Flask`?

To differentiate between one request and another, we use **sessions** in Flask. The session stores the information regarding each transaction in the form of **cookies**. **For example,** if we login to a website, and then click on another page, we do not get logged out. The reason is that the session maintains our user information.

> ### 💡 How do we use sessions in `Flask`?
>
> In `Flask`, we use the global `session` object to access the **current session**. This object is a simple dictionary. We can add or remove keys from it. **For example,** when a user logs in, we can insert a `'user'` key in the `session` with the value of the current user's object. Similarly, when the user logs out, we can remove the `'user'` key from the session.

Now that we know all about sessions, let's take a look at all the tasks that you are required to perform in this challenge.

1. **Authentication:** the user should be authenticated using the data received from the `form` in the `login` view. You will have to match the information from the `users` list to authenticate.

2. **Invalid user data:** in the case of wrong credentials, the `login` view should send a `message` to the template saying, *"Wrong credentials. Please try again."*

3. **Valid user data:** in case of valid user data, you have to return the message: *"Successfully logged in!"*.

4. **Initialize user in the session:** also, in the case of successful authentication, you have to add a `'user'` key in the session object before you return the template.

5. **Logout view and user removed from session:** now that we have logged in the user, we will have to give them a mechanism to logout. For this purpose, you will have to create a `logout` view function and route. It should log the user out by removing the `'user'` key from the `session` dictionary. Moreover, the logout view should redirect to the `homepage` view.

6. **Logout button in vavbar:** if a user is logged in, we do not want them to be able to see the "sign up" and "login" buttons in the navigation bar. Instead, we want them to see a "logout" button that triggers the `logout` view function.

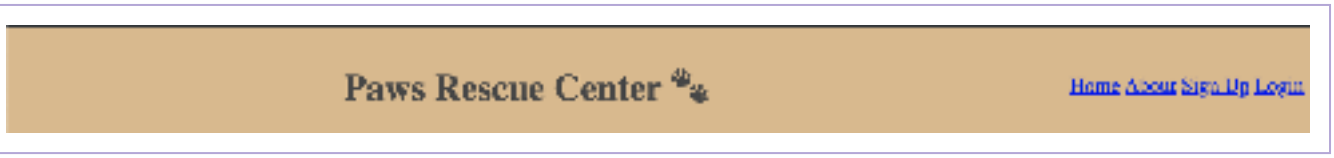
> 📌 **Note:**

1. We can access the `session` variable inside the templates. You will need it to solve the **6th** task.
2. `Flask` provides us with a `redirect()` function, which we can use to return from a view instead of a template. This function takes the `URL` for the view that we want to redirect to. You will need to use `url_for()` to create this `URL`. However, be sure to add the following arguments for the sake of security.

```
return redirect(url_for('view_name', _scheme='https', _external=True))
```

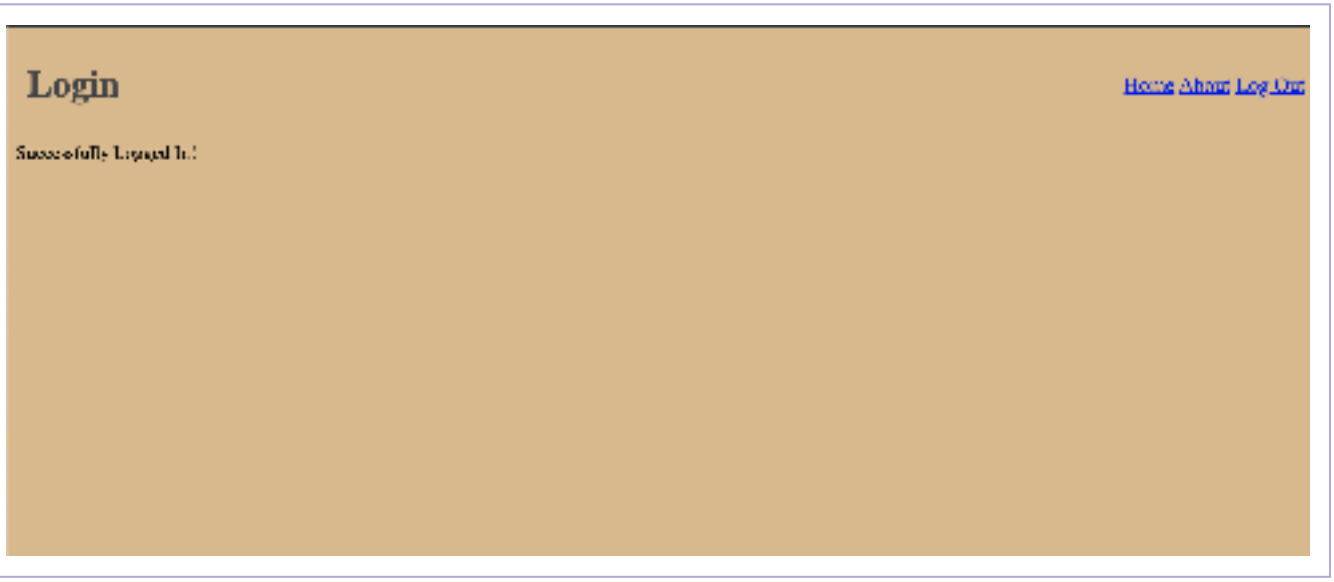
📌 **Note:** You might be able to use this function without the `_schema` and `_external` flags locally. However, the environment on the **Educative** platform is configured very securely and does not allow redirecting without them.

Navigation bar (not logged in) - expected output



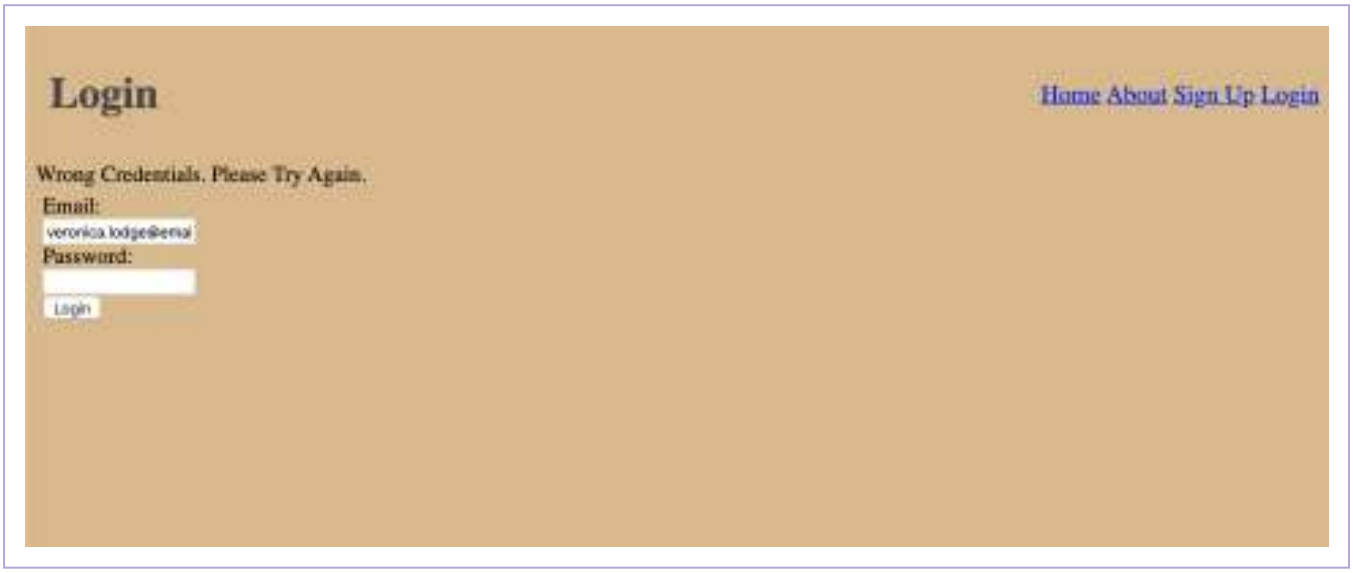
Navigation Bar (Not Logged In) - Expected Output

Login page and navigation bar (logged in) - expected output



Login Page and Navigation Bar (Logged In) - Expected Output

Login page (invalid user data) - expected output



Login Page (Invalid User Data) - Expected Output

Your implementation

Implement the features described above in the application provided below.

> ⚠ **Disclaimer:** if you edit the contents of `style.css`, you might notice that the application does not reflect the changes immediately. The reason for this is that the previous version is saved in your browser's cache. To solve this problem:
>
> 1. You can hard refresh your browser to clear the cache.
> 2. Alternatively, you can use inline `CSS` in the `HTML` templates using the `style` attribute. **(Recommended)**

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

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)


@app.route("/login", methods=["POST", "GET"])
def login():
    form = LoginForm()
    return render_template("login.html", form = form)

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

In the next lesson, let's look at the solution to this challenge. Stay tuned!