Redirecting

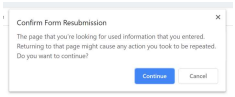
Objectives:

- Understand when to use a redirect
- Understand how to implement a redirect

As developers we should never render a template on a POST request. If we do, then it could result in some disastrous consequences.

- Duplication of Data in your application.
- Over-Charging Credit Cards (if processing payments).

So what can we do instead? When we have finished processing the POST data, we can perform a GET request on behalf of the client, which will now be the request that is completed should the client refresh the page. This is called ***redirecting***. **Always redirect after handling POST data to avoid data being handled more than once!**



Let's see how it works by going back to our form_test project. Our create_user method is the one processing the POST request. Run the project and submit the form. Hit the refresh button in your browser--you should see a pop up like the one on the right. The warning explains that we are sending the same form data to be processed again. While it's not a big deal yet, imagine our method was inserting a user into the database. Clicking *continue* would add that user to the database again. That would be no good!

Let's add a separate method that will be solely responsible for rendering the show page, and then change the last line of our method handling the POST data from `render_template` to `redirect` to the route that will render the page:

```python
from flask import Flask, render_template, request, redirect # don't forget to import redirect!

@app.route('/users', methods=['POST'])
def create_user():
    print("Got Post Info")
    print(request.form)
    name = request.form['name']
    email = request.form['email']
    return redirect("/show")

# adding this method
@app.route("/show")
def show_user():
    print("Showing the User Info From the Form")
    print(request.form)
    return render_template("show.html")
```

Now when we submit a form and hit refresh, we no longer receive the warning. Now, however, we've run into a new problem. Check the terminal to see what prints from our show_user method--the form data is empty! Why do you think this is?

Read on to learn one way we are able to hold on to, or *persist* data, across multiple requests!