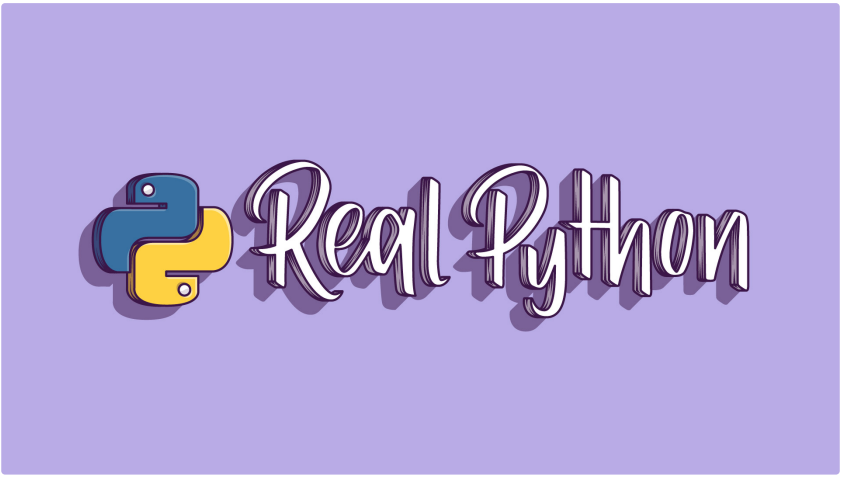

Discover Flask, Part 2 – Creating a Login Page

by Real Python 26 Comments basics flask web-dev

Tweet Share Email

Table of Contents

- Add a route to handle requests to the login URL
  - So, what's going on?
- Add a template for the login page
- Conclusion
- Video

Welcome to the Real Python *Discover Flask* series …

Series Overview

Visit discoverflask.com for the series summary—links to blog posts and videos.

Last time we went over how to set up a basic Flask structure and then developed a static site, styled with Bootstrap. In this second part of the series, we'll be adding a login page for end users to, well, login to.

Building on the code from the previous tutorial, we need to:

- Add a route to handle requests to the login URL; and
- Add a template for the login page

Add a route to handle requests to the login URL

Make sure your virtualenv is activated. Open *app.py* in your code editor and add the following route:

Python

```
# Route for handling the login page logic
@app.route('/login', methods=['GET', 'POST'])
def login():
    error = None
    if request.method == 'POST':
        if request.form['username'] != 'admin' or request.form['password'] != 'admin':
            error = 'Invalid Credentials. Please try again.'
        else:
            return redirect(url_for('home'))
    return render_template('login.html', error=error)
```

Make sure you also update the imports:

Python

```
from flask import Flask, render_template, redirect, url_for, request
```

So, what's going on?

1. First, notice that we specified the applicable HTTP methods for the route, GET and POST, as an argument in the route decorator.
2. GET is the default method. So, if no methods are explicitly defined, Flask assumes that the only available method is GET, as is the case for the previous two routes, `/` and `/welcome`.
3. For the new `/login` route we need to specifiy the POST method as well as GET so that end users can send a POST request with their login credentials to that `/login` endpoint.
4. The logic within the `login()` function tests to see if the credentials are correct. If they are correct, then the user is redirected to the main route, `/`, and if the credentials are incorrect, an error populates. Where do these credentials come from? The POST request, which you'll see in just a minute.
5. In the case of a GET request, the login page is simply rendered.

> **NOTE**: The `url_for()` function generates an endpoint for the provided method.

Add a template for the login page

Create a new file called *login.html*, adding it to the "templates" directory:

```
<html>
  <head>
```



```
      </form>
      {% if error %}
        <p class="error"><strong>Error:</strong> {{ error }}
      {% endif %}
    </div>
  </body>
</html>
```

Time for a quick test …

1. Fire up the server. Navigate to http://localhost:5000/login.
2. Enter the incorrect credentials, then press login. You should get this response: “Error: Invalid Credentials. Please try again.”
3. Now use “admin” for both the username and password and you should be redirected to the `/` URL.
4. Can you tell what’s happening here? When the form is submitted, a POST request is sent along with the form data, `value="{{request.form.username }}"` and `value="{{request.form.password }}"`, to the controller, `app.py` - which then handles the request and either responds with an error message or redirects the user to the `/` URL. **Be sure to check out the accompanying video to dig deeper into this with Chrome Developer Tools!**
5. Finally, we have some logic in our templates. Originally, we passed in None for the error. Well, if the error is not None, then we display the actual error message, which gets passed to the template from the views: `<p class="error"><strong>Error:</strong> {{ error }}</p>`. For info on how this works, check out this blog post to learn more about the Jinja2 templating engine.

Conclusion

What do you think? Simple, right? Don’t get too excited yet, as we still have much more to do with reguard to user management…

Now that users have the ability to login, we need to protect that URL `/` from unauthorized access. In other words, when an end user hits that endpoint, unless they are already logged in, then they should be immediately sent to the login page. Next time. Until then, go practice some jQuery.

Be sure to grab the code and watch the video.

Video

Discover Flask, Part 2 - Creating a login page