Templates ¶

You've written the authentication views for your application, but if you're running the server and try to go to any of the URLs, you'll see a `TemplateNotFound` error. That's because the views are calling **`render_template()`**, but you haven't written the templates yet. The template files will be stored in the `templates` directory inside the `flaskr` package.

Templates are files that contain static data as well as placeholders for dynamic data. A template is rendered with specific data to produce a final document. Flask uses the Jinja template library to render templates.

In your application, you will use templates to render HTML which will display in the user's browser. In Flask, Jinja is configured to *autoescape* any data that is rendered in HTML templates. This means that it's safe to render user input; any characters they've entered that could mess with the HTML, such as < and > will be *escaped* with *safe* values that look the same in the browser but don't cause unwanted effects.

Jinja looks and behaves mostly like Python. Special delimiters are used to distinguish Jinja syntax from the static data in the template. Anything between `{{` and `}}` is an expression that will be output to the final document. `{%` and `%}` denotes a control flow statement like `if` and `for`. Unlike Python, blocks are denoted by start and end tags rather than indentation since static text within a block could change indentation.

The Base Layout

Each page in the application will have the same basic layout around a different body. Instead of writing the entire HTML structure in each template, each template will *extend* a base template and override specific sections.

`flaskr/templates/base.html`

```
<!doctype html>
<title>{% block title %}{% endblock %} - Flaskr</title>
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}
<nav>
  <h1>Flaskr</h1>
  <ul>
    {% if g.user %}
      <li><span>{{ g.user['username'] }}</span>
      <li><a href="{{ url_for('auth.logout') }}">Log Out</a>
    {% else %}
      <li><a href="{{ url_for('auth.register') }}">Register</a>
      <li><a href="{{ url_for('auth.login') }}">Log In</a>
    {% endif %}
```

```
    </ul>
  </nav>
  <section class="content">
    <header>
      {% block header %}{% endblock %}
    </header>
    {% for message in get_flashed_messages() %}
      <div class="flash">{{ message }}</div>
    {% endfor %}
    {% block content %}{% endblock %}
  </section>
```

`g` is automatically available in templates. Based on if `g.user` is set (from `load_logged_in_user`), either the username and a log out link are displayed, otherwise links to register and log in are displayed. `url_for()` is also automatically available, and is used to generate URLs to views instead of writing them out manually.

After the page title, and before the content, the template loops over each message returned by `get_flashed_messages()`. You used `flash()` in the views to show error messages, and this is the code that will display them.

There are three blocks defined here that will be overridden in the other templates:

1. `{% block title %}` will change the title displayed in the browser's tab and window title.
2. `{% block header %}` is similar to `title` but will change the title displayed on the page.
3. `{% block content %}` is where the content of each page goes, such as the login form or a blog post.

The base template is directly in the `templates` directory. To keep the others organized, the templates for a blueprint will be placed in a directory with the same name as the blueprint.

Register

`flaskr/templates/auth/register.html`

```
{% extends 'base.html' %}

{% block header %}
  <h1>{% block title %}Register{% endblock %}</h1>
{% endblock %}

{% block content %}
  <form method="post">
    <label for="username">Username</label>
```

```
    <input name="username" id="username" required>
    <label for="password">Password</label>
    <input type="password" name="password" id="password" required>
    <input type="submit" value="Register">
  </form>
{% endblock %}
```

`{% extends 'base.html' %}` tells Jinja that this template should replace the blocks from the base template. All the rendered content must appear inside `{% block %}` tags that override blocks from the base template.

A useful pattern used here is to place `{% block title %}` inside `{% block header %}`. This will set the title block and then output the value of it into the header block, so that both the window and page share the same title without writing it twice.

The `input` tags are using the `required` attribute here. This tells the browser not to submit the form until those fields are filled in. If the user is using an older browser that doesn't support that attribute, or if they are using something besides a browser to make requests, you still want to validate the data in the Flask view. It's important to always fully validate the data on the server, even if the client does some validation as well.

Log In

This is identical to the register template except for the title and submit button.

`flaskr/templates/auth/login.html`

```
{% extends 'base.html' %}

{% block header %}
  <h1>{% block title %}Log In{% endblock %}</h1>
{% endblock %}

{% block content %}
  <form method="post">
    <label for="username">Username</label>
    <input name="username" id="username" required>
    <label for="password">Password</label>
    <input type="password" name="password" id="password" required>
    <input type="submit" value="Log In">
  </form>
{% endblock %}
```

Register A User

Now that the authentication templates are written, you can register a user. Make sure the server is still running (`flask run` if it's not), then go to http://127.0.0.1:5000/auth/register.

Try clicking the "Register" button without filling out the form and see that the browser shows an error message. Try removing the `required` attributes from the `register.html` template and click "Register" again. Instead of the browser showing an error, the page will reload and the error from **`flash()`** in the view will be shown.

Fill out a username and password and you'll be redirected to the login page. Try entering an incorrect username, or the correct username and incorrect password. If you log in you'll get an error because there's no `index` view to redirect to yet.

Continue to Static Files.