

Modèles

Vous avez écrit les vues d'authentification pour votre application, mais si vous exécutez le serveur et essayez d'aller sur l'une des URL, vous verrez une erreur `TemplateNotFound`. C'est parce que les vues appellent `render_template()`, mais vous n'avez pas encore écrit les modèles. Les fichiers de modèles seront stockés dans le répertoire `templates` du paquet `flaskr`.

Les modèles sont des fichiers qui contiennent des données statiques ainsi que des espaces réservés pour des données dynamiques. Un modèle est rendu avec des données spécifiques pour produire un document final. Flask utilise la bibliothèque de modèles [Jinja](#) pour rendre les modèles.

Dans votre application, vous utiliserez des modèles pour rendre le [HTML](#) qui s'affichera dans le navigateur de l'utilisateur. Dans Flask, Jinja est configuré pour *auto-échapper* toutes les données qui sont rendues dans les modèles HTML. Cela signifie que le rendu de la saisie de l'utilisateur est sûr ; tous les caractères qu'il a entrés et qui pourraient perturber le HTML, tels que `<` et `>`, seront *échappés* avec des valeurs *sûres* qui auront le même aspect dans le navigateur mais ne provoqueront pas d'effets indésirables.

Jinja se présente et se comporte essentiellement comme Python. Des délimiteurs spéciaux sont utilisés pour distinguer la syntaxe Jinja des données statiques du modèle. Tout ce qui se trouve entre `{{` et `}}` est une expression qui sera affichée dans le document final. `{%` et `%}` dénote une instruction de flux de contrôle comme `if` et `for`. Contrairement à Python, les blocs sont désignés par des balises de début et de fin plutôt que par une indentation, car le texte statique à l'intérieur d'un bloc peut modifier l'indentation.

La disposition de la base

Chaque page de l'application aura la même mise en page de base autour d'un corps différent. Au lieu d'écrire la structure HTML complète dans chaque modèle, chaque modèle *étendra* un modèle de base et remplacera des sections spécifiques.

`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</li>
      <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` est automatiquement disponible dans les modèles. Selon que `g.user` est défini (à partir de `load_logged_in_user`), soit le nom d'utilisateur et un lien de déconnexion sont affichés, soit des liens pour s'enregistrer et se connecter sont affichés. `url_for()` est aussi automatiquement disponible, et est utilisé pour générer des URLs pour les vues au lieu de les écrire manuellement.

Après le titre de la page, et avant le contenu, le modèle boucle sur chaque message renvoyé par `get_flashed_messages()`. Vous avez utilisé `flash()` dans les vues pour afficher les messages d'erreur, et voici le code qui va les afficher.

Il y a trois blocs définis ici qui seront remplacés dans les autres modèles :

1. `{% block title %}` modifiera le titre affiché dans l'onglet et le titre de la fenêtre du navigateur.
2. `{% block header %}` est similaire à `title` mais changera le titre affiché sur la page.
3. `{% block content %}` est l'endroit où va le contenu de chaque page, comme le formulaire de connexion ou un article de blog.

Le modèle de base se trouve directement dans le répertoire `templates`. Pour garder les autres organisés, les modèles pour un *blueprint* seront placés dans un répertoire avec le même nom que le *blueprint*.

S'inscrire

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>

```

 v: latest ▾

```

<input type="password" name="password" id="password" required>
<input type="submit" value="Register">
</form>
{% endblock %}

```

`{% extends 'base.html' %}` indique à Jinja que ce modèle doit remplacer les blocs du modèle de base. Tout le contenu rendu doit apparaître à l'intérieur des balises `{% block %}` qui remplacent les blocs du modèle de base.

Un modèle utile utilisé ici consiste à placer `{% block title %}` à l'intérieur de `{% block header %}`. Cela permettra de définir le bloc title puis d'afficher sa valeur dans le bloc header, de sorte que la fenêtre et la page partagent le même titre sans l'écrire deux fois.

Les balises `input` utilisent ici l'attribut `required`. Cela indique au navigateur de ne pas soumettre le formulaire tant que ces champs ne sont pas remplis. Si l'utilisateur utilise un ancien navigateur qui ne prend pas en charge cet attribut, ou s'il utilise autre chose qu'un navigateur pour faire des requêtes, vous devez quand même valider les données dans la vue Flask. Il est important de toujours valider complètement les données sur le serveur, même si le client effectue également une certaine validation.

Connexion

Ce modèle est identique au modèle pour s'inscrire, à l'exception du titre et du bouton d'envoi.

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 %}

```

Inscrire un utilisateur

Maintenant que les modèles d'authentification sont écrits, vous pouvez enregistrer  [v: latest](#) ▼. Assurez-vous que le serveur est toujours en cours d'exécution (`flask run` s'il ne l'est pas), puis allez à <http://127.0.0.1:5000/auth/register>.

Essayez de cliquer sur le bouton « Register » sans remplir le formulaire et voyez si le navigateur affiche un message d'erreur. Essayez de supprimer les attributs `required` du modèle `register.html` et cliquez à nouveau sur « Register ». Au lieu que le navigateur affiche une erreur, la page se recharge et l'erreur de `flash()` dans la vue s'affiche.

Remplissez un nom d'utilisateur et un mot de passe et vous serez redirigé vers la page de connexion. Essayez de saisir un nom d'utilisateur incorrect, ou un nom d'utilisateur correct et un mot de passe incorrect. Si vous vous connectez, vous obtiendrez une erreur car il n'y a pas encore de vue `index` vers laquelle rediriger.

Continuer vers [Fichiers statiques](#).