créer une application web en utilisant Flask en Python 3

NSI Tle - JB Duthoit

0.0.1 Intro

Introduction Flask est un petit framework web Python léger, qui fournit des outils et des fonctionnalités utiles qui facilitent la création d'applications web en Python. Il offre aux développeurs une certaine flexibilité et constitue un cadre plus accessible pour les nouveaux développeurs puisque vous pouvez construire rapidement une application web en utilisant un seul fichier Python.
Flask est également extensible et ne force pas une structure de répertoire particulière ou ne nécessite pas de code standard compliqué avant de commencer.

Flask utilise le moteur de modèle Jinja pour construire dynamiquement des pages HTML en utilisant des concepts Python familiers tels que les variables, les boucles, les listes, etc. Vous utiliserez ces modèles dans le cadre de ce projet.

0.0.2 Installation

```
pip install flask
```

0.0.3 Créer une application de base

Créer un fichier `hello.py` qui servira d'exemple minimal pour traiter les requêtes HTTP.
À l'intérieur de ce fichier, vous importerez l'objet Flask et créerez une fonction qui renvoie une réponse HTTP.
Écrivez le code suivant à l'intérieur `hello.py` :

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello():
    return 'Hello, World!'
if __name__ == '__main__':
    app.run(debug = True)
```

Dans le bloc de code précédent, vous importez d'abord l'objet Flask du paquet flask.
Vous l'utilisez ensuite pour créer votre instance d'application Flask avec le nom app.
Vous passez la variable spéciale ___name___ qui contient le nom du module Python actuel. Il est utilisé pour indiquer à l'instance où elle se trouve.

Une fois que vous avez créé l'instance de l'application, vous l'utilisez pour traiter les demandes web entrantes et envoyer des réponses à l'utilisateur.
@app.route est un décorateur qui transforme une fonction Python ordinaire en une fonction d'affichage Flask, qui convertit la valeur de retour de la fonction en une réponse HTTP à afficher par un client HTTP, tel qu'un navigateur web. Vous passez à la valeur '/' à @app.route() pour

indiquer que cette fonction répondra aux requêtes web pour l'URL /, qui est l'URL principale.

La fonction d'affichage `hello()` renvoie la chaîne `"Hello, World!"` comme réponse.
Exécutez ensuite le fichier python, vous obtiendrez quelque chose comme :

```
# Running the app with options chosen by Thonny. See Help for details.
* Serving Flask app 'app1' (lazy loading)
* Environment: development
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Ouvrez un navigateur et tapez l'URL `http://127.0.0.1:5000/`, vous recevrez la chaîne de caractères *Hello, World !* en réponse, cela confirme que votre application fonctionne correctement.

0.0.4 Utilisation des modèles html

Actuellement, votre application n'affiche qu'un simple message sans aucun HTML.
Les applications web utilisent principalement le HTML pour afficher des informations à l'intention du visiteur.
Vous allez donc maintenant travailler à l'incorporation de fichiers HTML à votre application, qui pourront être affichés sur le navigateur web.

Flask fournit une fonction d'aide render_template() qui permet l'utilisation du moteur de modèle Jinja.
Cela facilitera grandement la gestion du HTML en écrivant votre code HTML dans des fichiers .html et en utilisant la logique dans votre code HTML.
Vous allez utiliser ces fichiers HTML (modèles) pour construire toutes les pages de votre application, comme la page principale où vous afficherez les articles de blog en cours, la page de l'article de blog, la page où l'utilisateur peut ajouter un nouvel article, etc.

Dans ce nouveau fichier, vous allez importer l'objet Flask pour créer une instance d'application Flask comme vous l'avez déjà fait. Vous importerez également la fonction d'aide render_template() qui vous permet de rendre les fichiers de modèles HTML qui existent dans le dossier des modèles que vous êtes en train de créer. Le fichier aura une fonction d'affichage unique qui sera responsable de la gestion des demandes vers la principale route /.
Ajoutez le contenu suivant :

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')


if __name__ == '__main__':
    app.run(debug = True)
```

La fonction d'affichage `index()` renvoie le résultat de l'appel de render_template() avec index.html en argument, ce qui indique à render_template() qu'il doit rechercher un fichier

appelé index.html dans le dossier des modèles. Le dossier et le fichier n'existent pas encore, vous obtiendrez une erreur si vous deviez exécuter l'application à ce stade.

Créez le dossier `templates` et le dossier `static` (pour l'hébergement des fichiers statiques, tels que les fichiers CSS, les fichiers JavaScript et les images utilisées par l'application.)

A l'intérieur du dossier `templates`, créez le fichier `index.html` suivant :

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<title>Mon premier repertoire</title>
</head>
<body>
<h1>Bienvenue dans mon repertoire !!</h1>
</body>
</html>
```

Exécutez le fichier python et rendez vous sur `http://127.0.0.1:5000/` (ou actualisez la page web).

☛ Cette fois, le navigateur doit afficher le texte "Bienvenue dans mon répertoire!!" dans une balise <h1>.

0.0.5 Ajout de style

Ouvrez le fichier `index.html` et ajoutez :

```
<head>
<meta charset="UTF-8">
<link type="text/css" rel="stylesheet"
href="{{ url_for('static', filename='css/mon_style.css') }}" />
<title>Mon premier repertoire </title>
</head>
```

Vous pouvez maintenant créer un fichier `style.css` dans le dossier `static/css` :

```
h1 {
    border: 2px #eee solid;
    color: brown;
    text-align: center;
    padding: 10px;
}
```

0.0.6 Afficher des variables

```
from datetime import date
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
def index():
    d = date.today().isoformat()
    return render_template('index.html',la_date = d)
```

```
if __name__ == '__main__':
    app.run(debug = True)
```

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<link type="text/css" rel="stylesheet"
href="{{ url_for('static', filename='css/mon_style.css') }}" />
<title>Mon premier repertoire </title>
</head>
<body>
<h1>Bienvenue dans mon repertoire !!</h1>
<p> Voici la date : {{la_date}} </p>
</body>
</html>
```

0.0.7 Ajouter des hyperliens

```
from datetime import date
from flask import Flask, render_template
app = Flask(__name__)
@app.route('/')
def index():
    d = date.today().isoformat()
    return render_template('index.html',la_date = d)
@app.route("/new_page/")
def nlle_page():
    return render_template('new.html')
if __name__ == '__main__':
    app.run(debug = True)
```

```
<!DOCTYPE html>
<html lang="fr">
<head>
<meta charset="UTF-8">
<link type="text/css" rel="stylesheet"
href="{{ url_for('static', filename='css/mon_style.css') }}" />
<title>Mon premier repertoire </title>
</head>
<body>
<h1>Bienvenue dans mon repertoire !!</h1>
<p> Voici la date : {{la_date}} </p>
<p> <a href="{{url_for('nlle_page')}}" >
 Vers une nouvelle page ... </a> </p>
</body>
</html>
```

☛ N'oubliez pas de créer une page `new.html`