Jour-26 Python pour le Web

Python pour le Web

Python est un langage de programmation à usage général et il peut être utilisé pour de nombr eux endroits. Dans cette section, nous verrons comment nous utilisons Python pour le Web. I l existe de nombreux travaux de trame Web Python. Django et Flask sont les plus populaires. Aujourd'hui, nous verrons comment utiliser Flask pour le développement Web.

Ballon

Flask est un cadre de développement Web écrit dans Python. Flask utilise le moteur de modèle Jinja2. Le flacon peut également être utilisé avec d'autres bibliothèques avant modernes telles que React.

Si vous n'avez pas installé le package VirtualEnv, installez-le en premier. L'environnement vi rtuel permettra d'isoler les dépendances du projet à partir des dépendances locales de la mach ine.

Structure de dossier

Après avoir terminé toute l'étape, votre structure de fichier de projet devrait ressembler à ceci:



Configuration de votre répertoire de projet

Suivez les étapes suivantes pour commencer avec Flask.

StEP 1: Installez VirtualEnv en utilisant la communication suivante

PIP installe VirtualEnv

Étape 2:

asabeneh @ asabeneh: ~ / dektop \$ mkdir python_for_web asabeneh @ asabeneh: ~ / de bure au asabeneh @ asabeneh: ~ / dektop / python_for_web \$ source Venv / bin / active (Env) asabeneh @ asabène: ~ / bintop / python_for_web \$ piphe Asabeneh @ Asabeneh: ~ / Desktop / Python_For_Web \$ Pip Freeze Click == 7.0 Flask == 1.1.1 Its Dangereous == 1.1.0 Jinja2 == 2.10.3 MarkupSafe == 1.1.1 Werkze == 0,16.0 (Env) Asabeneh @ Asabeneh: ~ / Desktop / Python_For_Web \$

Nous avons créé un réalisateur de projet nommé Python_For_web. À l'intérieur du projet, nou s avons créé un environnement virtuel *venv* qui pourrait être n'importe quel nom mais je préfè re l'appeler *venv*. Nous avons ensuite activé l'environnement virtuel. Nous avons utilisé PIP F reeze pour vérifier les packages installés dans le répertoire du projet. Le résultat de PIP Freeze était vide car un package n'a pas encore été installé.

Maintenant, créons un fichier app.py dans le répertoire du projet et écrivons le code suivant. Le fichier app.py sera le fichier principal du projet. Le code suivant a un module FLASK, le modu le OS.

Création de routes

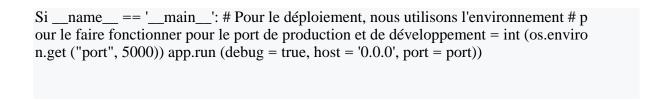
L'itinéraire domestique.

```
# Importons le ballon à partir du flacon Importation Importation OS # Importation du module du système d'exploitation

App = Flask (__ Name__)

@ app.Route ('/') # Ce décorateur crée la route de la maison def home (): return '< h1 > bienvenue < / h1 >'

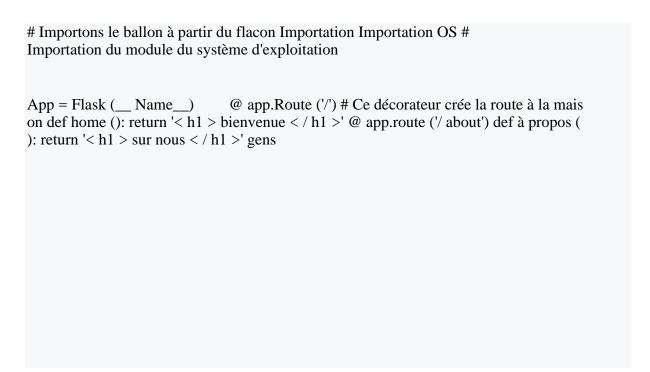
@ app.Route ('/ About') def about (): return '< h1 > à propos de nous < / h1 >'
```



Pour exécuter l'application FLASK, écrivez Python App.py dans le répertoire d'application Flask principal.

Après avoir exécuté python app.py, consultez l'hôte local 5000.

LET US Ajouter un itinéraire supplémentaire. Création de Rou



Maintenant, nous avons ajouté l'itinéraire à propos du code ci-dessus. Que diriez-vous si no us voulons rendre un fichier html au lieu de la chaîne? Il est possible de rendre le fichier HT ML à l'aide de la fonction *render_templae*. Créons un dossier appelé Modèles et créez Ho me.html et environ.html dans le répertoire du projet. Importons également la fonction *render_template* à partir de la balle.

Création de modèles

Créez le dossier HTML Fichiers à l'intérieur des modèles.

home.html

```
<! Doctype html > < html lang = "en" > < head > < meta charset = "utf-8" / > < M eta name = "Viewport" Content = "Width = Device-Width, Initial-Scale = 1.0" / > < title > home < / title < H1 > Bienvenue à la maison < / h1 > < / body > < / html > < / body > < / htmleful html > < / body > < / html > < / body > < / html > < / body
```

À propos.html

```
<! Doctype html > < html lang = "en" > < head > < meta charset} Content = "Wid th = Device-Width, Initial-Scale = 1.0" / > < Title > About < / title < H1 > À prop os de nous < / H1 > < / Body > < / Html >
```

Script python

```
app.py
```

```
# Importons le ballon
à partir du flacon de flacon, rendu_template import # module du systè
me d'exploitation d'importation

App = Flask (__ Name__)

@ app.Route ('/') # Ce décorateur Créer la route de la maison def home (): return re
nder_template ('home.html')

@ app.Route ('/ About') def about (): return render_template ('
about.html') si __name__ == '__main__':
```

```
# Pour le déploiement, nous utilisons l'environnement

# pour le faire fonctionner pour la production et le développement

port = int (os.environ.get ("port", 5000))

app.run (debug = true, host = '0.0.0.0', port = port)
```

Comme vous pouvez le voir pour aller à différentes pages ou pour naviguer, nous avons be soin d'une navigation. Ajoutons un lien à chaque page ou créons une mise en page que nous utilisons à chaque page.

Navigation

```
<ul > < li > < a href = "/" > home < / a > < / li > < li > < a href = "/ À propos" > À propos de < / a > < / li > < / ul >
```

Maintenant, nous pouvons naviguer entre les pages en utilisant le lien ci-dessus. Créons une page supplémentaire qui gère les données de formulaire. Vous pouvez l'appeler n'im porte quel nom, j'aime l'appeler post.html.

Nous pouvons injecter des données aux fichiers HTML à l'aide du moteur de modèle Jinja2.

Importons le ballon à partir du flacon d'importation FLASK, RENDER_TEMPLAT, DEM AND, REDIRECT, URL_FOR IMPORT OS # Importation du module du système d'exploit ation

App = Flask (___Name__) @ app.Route ('/') # Ce décorateur crée la route à la maison def home (): Techs = ['Intml', 'css', 'flask', 'python'] name = '30 jours de Python Programming ' = 'home') @ app.Route ('/ About') def about (): name = '30 Days of Python Programming 'return render_template ('about.html', name = name, title =' à propos de nous ') @ app.Route ('/ pos t ') def post ():) render_template ('post.html', nom = nom, titre = name)

Si __name__ == '__main__': # pour le déploiement # Pour le faire fonctionner pour la p roduction et le développement du port = int (os.environ.get ("port", 5000)) app.run (deb ug = true, host = '0.0.0.0', port = port)

Voyons aussi les modèles:

home.html

<! Doctype html > < html lang = "en" > < head > < meta charse Content = "Width = Appareil-Width, Initial-Scale = 1.0" / > < title > home < / title > < / head > < bo dy > < a href = "/" > home < / a > < a a href = "/ À propos de " > À propos de < / a > < h1 > Bienvenue à {{nom}} < / H1 > {% pour la technologie dans Techs%} {{tech}} {% endfor%} < / body > < / html >

À propos.html

<! Doctype html > < html lang = "en" > < head > < meta charset $\}$ "utf-8" / > < m éta-nom de méta = "Contenu = "Width =-Width, Initial-Scale = 1.0" / > < Title > About Us < / title > < / head > < body > > > < a href = "/" > home < / a >> > < a a href = "/ À propos" > À propos de < / a >> > < h1 > À propos de nous < / h1 >

```
< H2 > {{nom}}} < / h2 > < / bo dy > < / html >
```

Créer une disposition

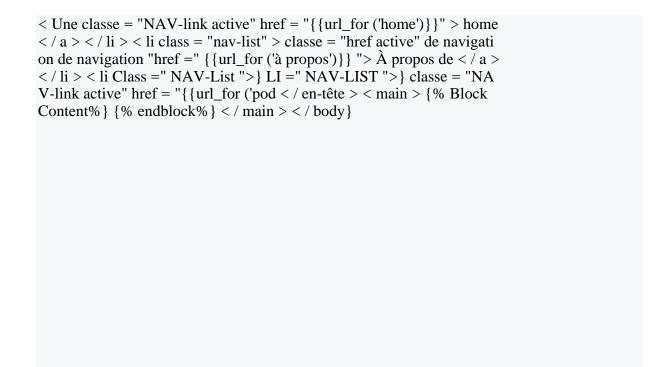
Dans les fichiers de modèle, il y a beaucoup de codes répétés, nous pouvons écrire une mise en page et nous pouvons supprimer la répétition. Créons Layout.html à l'intérieur du dossier Modèles. Après avoir créé la mise en page, nous importerons dans chaque fichier.

Servir le fichier statique

Créez un dossier statique dans votre répertoire de projet. À l'intérieur du statique F plus ancie n, créez un dossier CSS ou Styles et créez une feuille de style CSS. Nous utilisons le module *url for* pour servir le fichier statique.

disposition.html

<! Doctype html > < html lang = "en" > < head > < meta charset = "utf-8" / > < meta nom = "Content = "Width = Appareil-Width, Initial-Scale = 1.0" / > < lien href = "https://fonts.goo gleapis.com/css?family; ito: 300,400 | raleway: 300,400,500 & display = swap "rel =" Styles heet "/ > < lien rel =" Stylesheet "href =" {{url_por Nom de fichier = 'CSS / main.css')}} "/ > {% if title%} < title > 30 jours de python - {{title}} < / title > {% Else%} < Title >} {% de%} < Title >} {% de%} < Title >} {% de%} < Title > {% endif %} < / head > < body > < Header > < div class = "menu-Contenant" >} < < une classe = "Br and-Name Nav-link" href = "/" > 30daysofpython < / a > < / div} < li classe = "NAV-list" >



Maintenant, supprime tout le code répété dans les autres fichiers de modèle et importez la m ise en page.html. Le HREF utilise la fonction *url_for* avec le nom de la fonction d'itinéraire pour connecter chaque route de navigation.

home.html

 $\label{eq:content} $$\{\%$ étend 'Layout.html'\%$ } $$\{\%$ Block Content%$ } < div class = "conteneur" > < h1 > bienv enue sur $$\{name\}$ } < / h1 > le nombre de mots, de caractères et de mots les plus fré quents dans le texte. Vérifiez-le en cliquant sur Text Analyzer au menu. Vous avez besoin des technologies suivantes pour créer cette application Web: < UL Classe = "Tech-Lists" > $$ pour la technologie dans Techs%$ > Li classe = "Tech" > $$ {Tech}$ $$ endfor%$ < / div > $$ Endblock%$$}$

À propos.html

 $\begin{tabular}{ll} { \begin{tabular}{ll} $\{\%$ Block Content%} & < div class = "conteneur" > < h1 > à propos de $\{\{nom\}\} & < / h1 > Il s'agit d'un Python de 30 jours. Si vous codiez aussi loin, vous êt es génial. Félicitations pour le travail bien fait! & < / div > $\{\%$ endblock%} \end{tabular}$

poster.html

{% endblock%}

```
 \begin{tabular}{ll} {\it \% Etend 'Layout.html'\%} &{\it \% Block Content\%} &{\it class} = "conteneur" > < h1 > analyseur de texte < / h1 > < formulaire Action = "https: // ThirtyDaySofpython- v1.herokuapp.com/post "Method = "Post" > < div > < textarea rows = "25" name = "contenu" contenu "contenu" contenu "contenu" contenu "Autofocus > < / textarea > < / div > < Type d'entrée = "Soum is" Class = "btn" value = "Process text" / > < / div >
```

Méthodes de demande, il existe différentes méthodes de demande (obtenir, publier, mettre, s upprimer) sont les méthodes de demande courantes qui nous permettent de faire un opération CRUD (créer, lire, mettre à jour, supprimer).

Dans le poste, nous utiliserons l'alternative Get et Post Method en fonction du type de dema nde, vérifiez à quoi il ressemble dans le code ci-dessous. La méthode de demande est une fo nction pour gérer les méthodes de demande et également accéder aux données du formulaire . app.py

```
# let's import the flask
from flask import Flask, render_template, request, redirect,
url_for
import os # importing operating system module

app = Flask(__name__)
# to stop caching static file
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0
```

@ app.Route ('/') # Ce décorateur crée la route à la maison def home (): Techs = ['html', 'css', 'flask', 'python'] name = '30 days of python programming 'return rend er_template (' home.html ', techs = techs, name = name, title =' home ') @ app.Route ('/ About') Def About (): Nom = '30 Days of Python Programming 'return render_template (' about.html ', nom = name, title =' à propos de nous ') @ app.Route (' / resul t') def Result (): return rende_template («Result.html') @ app.route ('/ post', V9 ['Get', 'post']) def post (): name = 'analyseur de texte' if request.method == 'get': return rende_template ('pos t.html', name = name, title = name) if request.method == 'post': contenu = requête. redirectio n (url_for ('result')) si __name__ == '__main__': # pour le déploiement # pour le faire fonctio nner pour la production et le développement du port = int (os.environ.get ("port", 5000)) app. run (debug = true, hôte = '0.0.0.0'

Jusqu'à présent, nous avons vu comment utiliser le modèle et comment injecter des données sur le modèle, comment une mise en page commune. Maintenant, permet de gérer le fichier statiqu e. Créez un dossier appelé Static dans le directeur de projet et créez un dossier appelé CSS. À l'intérieur du dossier CSS Créez main.css. Votre principal. Le fichier CSS sera lié à la mise en p age.html.

Vous n'avez pas à rédiger le fichier CSS, à le copier et à l'utiliser. Passons au déploiement.

Déploiement

Créer un compte Heroku

Heroku fournit un service de déploiement gratuit pour les applications frontal et fullst ack. Créez un compte sur Heroku et installez la machine Heroku CLI pour votre machine. Après avoir installé Heroku, écrivez la commande suivante

Connectez-vous à Heroku

asabeneh @ asabeneh: ~ \$ Heroku Login Heroku: appuyez sur n'importe quelle touche pour o uvrir le navigateur pour se connecter ou Q pour quitter:

Voyons le résultat en cliquant sur n'importe quelle touche du clavier. Lorsque vous appuyez s ur n'importe quelle touche du clavier, il ouvrira la page de connexion Heroku et cliquez sur la page de connexion. Ensuite, vous serez connecté à la machine locale au serveur Heroku dista nt. Si vous êtes connecté à un serveur distant, vous le verrez.

Asabeneh @ Asabeneh: ~ \$ Heroku Login Heroku: Appuyez sur n'importe quelle touche pour ouvrir le navigateur pour se connecter ou Q pour quitter: ouvrir le navigateur à https://CLI-auth.heroku.com/auth/browser/BE12987C-583A-4458-A2C2-BA2CE7F41610 asabeneh@gmail.com asabeneh @ asabeneh: ~ \$

Créer des exigences et ProCFile

Avant que nous Hustotre code sur le serveur distant, nous avons besoéthades Besoinm

```
•exigence.txt • procfil e
```

(Env) asabeneh @ asabeneh: ~ / Desktop / Python_For_Web \$ PIP Freeze Click == 7.0 Flask == 1.1.1 ITANGEUER == 1.1.0 Jinja2 == 2.10.3 Markupsafe == 1.1.1 werkzeg == 0,16. (Env) asabeneh @ asabeneh: ~ / dektop / python_for_web \$ Touch exigences.txt (Env) asabeneh @ asabeneh: ~ / Desktop / python_for_web \$ pip gezé > exigences.txt (Env) asabeneh @ asabene h: ~ / dektop / python_for_web \$ Cat exigences.txt cliquez == 7.0 flask == 1.1.1 itsdangeous == 1.1.0 jinja2 == 2.10.3 bancupsafe == 1.1.1 Werkzug == 0.16.0

(Env) as abeneh @ asabeneh: \sim / dektop / python_for_web \$ touch profile (Env) as abeneh @ asabeneh: \sim / de bureau asabeneh @ asabeneh: \sim / bourse / python_for_w eb \$

Le ProCFile aura la commande qui exécutera l'application sur le serveur Web dans notre ca s sur Heroku.

Web: python app.py

Pousser le projet à Heroku

Maintenant, il est prêt à être déployé. Étapes pour déployer la demande sur Heroku

- 1. Git init 2. Git Add. 3. Git commit -m "Commit Messag e" 4. Heroku Créer le nom de l'application comme un mot
- '5. Git push Heroku Master 6. Heroku Open (pour lancer l' application déployée)

Après cette étape, vous obtiendrez une application comme celle-ci

Exercices: Jour 26

1. Vous construirez cette application. Seule la partie de l'analyseur de text e est laissée Félicitations!