HAI726I / Projets autour d'un serveur Flask

-

Pierre Pompidor

Le but du projet consiste à réaliser une application utilisant un **serveur HTTP écrit en Python** qui affiche suivant la recherche de l'utilisateur, des informations sur une thématique que vous aurez choisie :

- la géolocalisation et la recommandation de restaurants / bars ;
- la recherche d'informations sur une thématique particulière (par exemple, des pages extraites de wikipedia puis personnalisées) ;
- toute autre application mettant en œvre les mêmes prérequis.

L'affichage de ces informations pourra se faire dans un navigateur ou seulement dans un terminal.

Cette application se décompose en deux programmes informatiques :

- du côté serveur : un serveur Python utilisant le module Flask qui extrait et renvoie des données sauvées localement (a priori dans des fichiers, mais la possibilité d'utiliser une base de données n'est pas exclue) ;
- du côté client : un client qui recueille les critères de recherche de l'internaute, sollicite le serveur, et affiche les données renvoyées par le serveur (ce client peut être un navigateur ou un script Python).

L'intérêt de ce projet serait de pouvoir gérer sa propre base de connaissances, les données extraites des fichiers sauvés localement pouvant être à terme stockées dans une base de données...

1 Introduction à la mise en œuvre d'un serveur Flask :

Pour faire fonctionner ce qui suit (par exemple sur votre ordinateur personnel), vous devez sous Linux installer les modules **flask** et **requests** pour Python (ce dernier module sera utile si vous utilisez un client Python) :

```
sudo pip3 install flask requests
```

ou mieux encore sous Linux (car *apt* gère automatiquement les mises à jour) :

```
sudo apt install python-flask python-requests
```

1.1 Un premier exemple de serveur Flask

Voici un premier exemple de serveur Flask qui illustre la notion de **routes**, de **paramètres** et de **services web**. La route est la chaîne de caractères qui suit l'adresse IP du serveur et indique à celui-ci quelle action exécuter. Cette route est composée de un ou plusieurs fragments séparés par des /. Ces fragments peuvent être constants ou variables : dans ce dernier cas, ils sont appelés des paramètres et entourés par < et >..
Un service web est la fonction "annotée" par une route qui va être exécutée quand le serveur Flask reçoit une requête HTTP exprimant cette route.

```
from flask import Flask
app = Flask(__name__)

# Premier service web
@app.route('/')  # ici la route est vide
def index():
   return "Bonjour"

@app.route('/hello/<segment>')  # partie fixe (hello) et partie variable
def hello(phrase):
   return "Bonjour "+phrase

@app.route('/hello/<segment1>/<segment2>')
def hello2(segment1, segment2):
   return "Bonjour "+segment1+" "+segment2

app.run()
```

Par défaut, un serveur Flask "tourne" sur le port 5000.

Pour l'exécuter (dans votre terminal) : `python3 serveurFlask.py`

Pour le tester (via la barre d'URL de votre navigateur) :
`localhost:5000` ou `localhost:5000/` → Bonjour.
`localhost:5000/hello/Pierre` → Bonjour Pierre
`localhost:5000/hello/Pierre/Paul` → Bonjour Pierre Paul

2 Recherche et affichage de données géolocalisées :

2.1 D'un fichier de format *geojson* à une arborescence de fichiers :

Le fichier *geoson OSM_Metropole_restauration_bar.json* de l'Opendata de Montpellier contient les informations (types (que j'appellerai ensuite thèmes), noms, géolocalisations) des établissements (restaurants, bars...) de Montpellier.

```
"type": "FeatureCollection",
"crs": { "type": "name", "properties": { "name": "urn:ogc:def:crs:OGC:1.3:CRS84" } },
"features": [
    { "type": "Feature", "properties":
      { "osm_id": 702385889, "ad_street": null, "addr_numbr": null, "amenity": "bar",
        "name": "Fabulous", "brand": null, "operator": null, "ref": null, "wheelchair": null,
        "internet_access": null, "opening_hours": null, "drive_through": null, "building": null,
        "cuisine": null, "capacity": null,
        "tags": "", "metro_id": 1 },
        "geometry": { "type": "Point", "coordinates": [ 3.708865025634871, 43.550375084904218 ]
      }
    },
...
```

Ecrire un script Python *parseur_geoJSON.py* qui crée dans un dossier nommé *PI* auant de sous-dossiers qu'il y a de types d'établissements différents (*amenity*) et dans ceux-ci autant de fichiers texte que d'établissements :

```
              PI
      /        |         \
restaurant  fast-food ... bar
   /    \    /     \     /   \
resto1 ...  fastf1 ...  bar1 ...
```

Dans mon implémentation les trois premières lignes des fichiers vont être le nom, la latitude et la longitude de l'établissement :

```
Fabulous
43.550375084904218
3.708865025634871
```

La finalité du projet est que ces fichiers texte puissent être complétés par des recomandations (une ligne par avis laissé par un client : "note de 1 à 5 : texte libre"

2.2 Mise en œuvre d'un serveur Flask qui renvoie les données géolocalisées :

Mettre en œuvre un serveur Flask qui interrogé sur une route `<theme>` renvoie la liste des établissements de ce thème (restaurant ; fastfood, bar...) :

```
import os, json
from flask import Flask
app = Flask(__name__)
```

```
@app.route('/<theme>')
def theme(theme):
   listePI = []
   if os.path.isdir("PI/"+theme) :
      liste = os.listdir("PI/"+theme)
      for fichier in liste :
         with open("PI/"+theme+"/"+fichier) as fd :
            [nom, latitude, longitude] = fd.readlines()
            listePI.append([nom.strip(), latitude.strip(), longitude.strip()])
   return json.dumps(listePI)

app.run()
```

Rajoutez un service web pour que sur la route / le serveur renvoie la liste des thèmes.

Rajoutez un service web pour que le serveur renvoie la liste des établissements d'un thème donné classé par leurs notes.

Rajoutez un service web pour que le serveur renvoie la liste des commentaires donnés à un établissement.

2.3 Du client Python vers un affichage graphique au sein d'un navigateur :

La dernière étape consisterait à utiliser un navigateur pour rechercher les établissements d'un thème donné et faire afficher sur une carte les marqueurs les concernant (ainsi qie via une popup les commentaires associés).

Voici le code HTML complété par du code JavaScript/Jquery qui permettrait d'interroger le serveur Flask pour créer une liste de cases à cocher exhbant les différents thèmes :
Attention, ce code ne fonctionnera pas en l'état pour des raisons de sécurité (CORS) : il faudrait que la page HTML soit aussi renvoyée par le serveur.

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"
            integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
            crossorigin="anonymous"></script>
    <script>
      $().ready(function(){
        $.getJSON("http://localhost:5000/", function(data) {
          for (let theme of data) {
            let html = "<input type='checkbox' name='"+theme+"'>"+theme+"</input>";
            $("#listeThemes").append(html);
          }
        });
      });
    </script>
  </head>
  <body>
    <div id="listeThemes" />
  </body>
</html>
```

Pour ceux qui seraient intéressés, des informations supplémentaires seront données en TP pour finaliser le client HTML/JavaScript.

3 Projet base de connaissances personnelle :

3.1 Un script bash qui sauve localement des pages issues de Wikipedia :

Par exemple, en utilisant la commande **curl** nous récupérons les trois pages "brutes" (c'est à dire sans code HTML) de Wikipedia concernant les bolets en général, le bolet de Bordeaux et le bolet satan.

Voici l'utilisation de cette commande pour récupérer la page générale sur les bolets :

```
curl "https://fr.wikipedia.org/w/index.php?title=bolet&action=raw" > bolet
```

Pour rendre ces importations plus aisées, nous écrivons le script bash suivant qui sauve les pages dans un dossier *PAGES* (vous remarquerez qu'au lieu de les lister dans une chaîne de caractères, les thèmes des pages pourraient être passés en paramètres) :

```
champs="bolet Boletus_edulis Bolet_Satan"
for champ in $champs
do
    echo importation de $champ
    curl "https://fr.wikipedia.org/w/index.php?title=${champ}&action=raw" > PAGES/$champ
done
```

3.2 Un serveur Flask qui revoie des données extraites de fichiers :

Soient les trois pages précédentes entreprosées dans un dossier nommé *PAGES*, voici un exemple de script Python qui va extraire de celles-ci toutes les **références** apparaissant dans les pages. Les références désignent les entités qui font l'objet d'autres articles Wikipedia (dans mon cas, principalement les autres espèces de champignons citées). Les références sont encadrées par des doubles crochets comme le montre l'exemple ci-dessous du début de la page consacrée aux bolets :

```
{{Infobox Biohomonymie
 | nom     = Bolet
 | autre   =
 | image   = Körnchenröhrling-1.jpg
 | légende = [[Bolet granulé]] (''Suillus granulatus'')
 | alt     =
 | upright =
 | taxons  = dans les genres suivants (famille des [[Boletaceae]]):
* ''[[Aureoboletus]]''
* ''[[Boletellus]]''
...
}}

'''Bolet''' est un [[nom vernaculaire]] ambigu désignant en [[français]] certains [[champignon]]s classés ...
```

Dans le script ci-dessous, l'expression régulière est masquée (????).

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-
import os, json, re
os.system("clear")

from flask import Flask
app = Flask(__name__)
```

```
@app.route('/references')
def references():
   print("/references")
   references = []
   liste = os.listdir("PAGES")
   for fichier in liste :
      print(fichier)
      fd = open("PAGES/"+fichier)
      for ligne in fd.readlines() :
         resultat = re.search( ???? , ligne)
         if resultat :
            references.append(resultat.group(1))
   return json.dumps(references)

app.run()
```

Si nous exécutons la commande :

```
curl localhost:5000/references
```

voici un extrait de ce qui est renvoyé par le serveur (formaté dans une liste transformée en chaîne de caractères aves *json.dumps()*) :

```
["Bolet granulé", "Boletaceae", "Aureoboletus", "Boletellus", "Boletinellus", "Boletus",
  "Buchwaldoboletus", "Chalciporus", "Gyroporus", "Leccinum", "Pulveroboletus", "Strobilomyces",
  "Suillus", "Tylopilus", "Xerocomus", "nom vernaculaire", "grec ancien",
  "Nom normalisé|noms normalisés", ... ]
```

3.3 Un serveur Flask qui revoie des données suivant un critère de recherche :

Nous voulons maintenant solliciter le serveur en lui transmettant un paramètre qui est un critère de recherche (et que nous voulons rechercher "à proximité" des références qui seront ainsi filtrées sur ce critère).
Bien entendu, pour la suite du projet, vous devrez permettre l'envoi d'un nombre arbitraire de critères de recherche et trvailler sur cette notion de proximité.

Voici le service web qui gère la route `/recherche/<critere>` :

```
@app.route('/recherche/<critere>')
def recherche_critere(critere):
   print("/recherche/"+critere)
   lignes = []
   liste = os.listdir("PAGES")
   for fichier in liste :
      fd = open("PAGES/"+fichier)
      for ligne in fd.readlines() :
         res1 = re.search( ???? , ligne)
         res2 = re.search(""+critere, ligne)
         if res1 and res2 :
            lignes.append(ligne)
   return json.dumps(lignes)
```

Si le serveur est invoqué via votre navigateur comme suit :

```
curl localhost:5000/recherche/jaune
```

voici le premier résultat qui est renvoyé par le serveur :

```
["L'[[hyménium]] se présente, comme chez tous les bolets, sous forme de tubes, appelés parfois ''foin'' en fran
```

3.4 Côté client avec un script Python :

Voici maintenant deux versions de scripts python qui sollicitent le serveur en l'appelant sur la route `/recherche/<critere>` et affiche les données renvoyées.

3.4.1 Première version en utilisant le module *system* :

Soit le script (maladroit) *recherche.py* :

```
# -*- coding:utf-8 -*-
import sys, os, json
os.system("curl localhost:5000/recherche/"+sys.argv[1]+" > resultat")
fd = open("resultat")
lignes = json.loads(fd.readline())
for ligne in lignes :
    print(ligne)
    print("--------------------")
```

Ce script est par exemple exécuté ainsi : `./recherche.py jaune`

3.4.2 Seconde version plus élégante en utilisant le module *requests* :

```
# -*- coding:utf-8 -*-
import sys, json, requests
reponse = requests.get("http://localhost:5000/recherche/"+sys.argv[1])
lignes = reponse.json()
for ligne in lignes :
    print(ligne)
    print("--------------------")
```

3.5 Côté client avec une page HTML/JavaScript :

Voici un exemple de page HTML / JavaScript qui ferait office de client :

```
<html lang="fr">
  <head> <meta charset="UTF-8" /> </head>
  <body>
    <h2> Recherche </h2>
    critere : <input type="input" id="critere" />
    <button onclick="recherche()"> Va chercher </button> <hr/>
    <h2> Résultats </h2>
    <ul id="resultats" />
  </body>

  <script src="http://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js" type="text/javascript"></script
  <script>
     function recherche() {
         var critere = $("#critere").val();
         console.log("Dans recherche() avec "+critere);
         $.getJSON("http://localhost:5000/recherche/"+critere, function(reponses) {
             for (let reponse of reponses) {
                 let url = "https://fr.wikipedia.org/w/index.php?title="+reponse[0];
                 let html = "<li>";
                 let phrase = reponse[1].replace(critere, "<font color='red'><b>"+critere+"</b></font>");
                 html += "<a href='"+url+"'>"+reponse[0]+"</a> : "+phrase;
                 html += "</li>";
                 $("#resultats").append(html);
             }
         });
     }
  </script>
</html>
```