Sorbonne Université

Faculté des Sciences et Ingénierie

Master 1 Réseaux

Module PROGRES

Sébastien Tixeuil

Mini Projet 2 API WEB

Auteurs :

HALET Mohamed Larbi

MATARI Mohamed

I. Introduction

Une API (Application Programing Interface) est un ensemble de programmes, de protocoles et d'outils permettant de créer des logiciels et des applications.
L'API est une sorte d'interface qui dispose d'un ensemble de fonctions permettant aux programmeurs d'accéder à des fonctionnalités ou des données spécifiques d'une application d'un système d'exploitation ou d'autres services.
L'API Web, comme son nom l'indique, est une API sur le Web et donc accessible via le protocole HTTP.

II. Exercice 1 : l'API

1. Importation des bibliothèques :

Afin de réaliser l'API en python, nous avons utilisé principalement la bibliothèque ***bottle*** :

```
from bottle import*
from html.parser import HTMLParser
import requests
from lxml import etree as ET
from json import *
from re import *
```

2. Récupération du fichier :

```
#============================== Récupération du fichier ======================================

#Pour donner la possibilité de choisir le fichier à traiter en fonction des caractéristiques du PC
choix=0
while choix ==0:
    print('choix de fichiers à traiter :')
    print('1- Parsing dblp.xml')
    print('2- Parsing dblp_2020_2021.xml')
    print('3- Parsing dblp_2017_2021.xml')
    choix= int(input('choisir un chiffre : '))
    if choix ==1 :
        filename= 'dblp.xml'
    elif choix == 2:
        filename='dblp_2020_2021.xml'
    elif choix == 3:
        filename='dblp_2017_2021.xml'
    else : choix =0

#Dans notre cas nous avons utilisé 'dblp_2020_2021.xml' tronqué pour faire les tests (RAM insuffisante)
try:
    Parser = ET.XMLParser(recover=True)
    Tree = ET.parse(filename,parser=Parser)
    root = Tree.getroot()
except:
    print('must download ' +filename+ ' on your localhost')
```

Pour des soucis de capacités des ordinateurs, nous offrons à l'utilisateur le choix du fichier à traiter. Dans nos tests, nous avons utilisé le fichier '**dblp_2020_2021.xml**'.

3. Définition de la fonction check_name :

```
#Fonction pour vérifier la conformité des noms introduits par l'utilisateur
def check_name(name):

    regular = r'(([A-Z]{1}[a-z]+) ([A-Z]{1}([.][ ][A-Z])?[a-z]+)[ ]*[0-9]*){1}'

    if(fullmatch(regular, name)):

        result = True

    else:
        result = False
    return result
```

Cette fonction nous permet, grâce à une expression régulière, de savoir si un nom en entrée est valide ou non. Nous l'utiliserons par la suite dans toutes nos routes afin de gérer les erreurs dans les noms des auteurs.

4. **Définition de la fonction limit :**

```
def limit(List):
      st= False
      co= False
      result = []
      try:
          if  request.params.get('start') is not  None :
                start = int(request.params.get('start') )
                st= True

          if request.params.get('count') is not None :
                count = int (request.params.get('count'))
                co= True

          if st and co :
                result = List[start+1:start+count+1]

                if start > len(List):
                    result ='Error, there is no'+ start+'th element in List'
                elif count > 100:
                    result= 'Error, you can only display 100 elements'
                return result

          elif st:
               result = List[start+1:start+102]
          else:
               if len(List)>=100:

                    result = List[:101]
               else :
                    result = List
      except TypeError or ValueError:

          result=Error_Start

      return result
```

Cette fonction nous permet de satisfaire la condition de l'API à savoir :
« Chaque route qui retourne une liste, doit retourner au maximum 100 éléments et accepter des paramètre d'URL `start` et `count` qui permettent d'afficher `count` éléments, à partir du `start`-ième élément. ».
Ayant une liste en entrée, si les paramètres **start** et **count** ne sont pas précisés : les 100 premiers éléments sont affichés au maximum. Si ces paramètres sont mal saisis (doivent être des entiers), une erreur est générée.

5. **Définition de la fonction getInfo_by_Author :**

```
#Fonction qui permet de récupérer l'auteur, ses publications ou ses co-auteurs
def getInfo_by_Author(name,choix):

        Nombres_pub_coth = []
        Nombres_pub_coth.append(name)
        Publications=[]
        Coauthors = []
        n_pub=0
        n_cauth=0

        for types in publications:

                  for pubs in root.findall(types):
                       authors=pubs.findall(target_Elements[0]) #author

                       for author in authors:
                            if(author.text == name):
                                 Publications.append(pubs)

                                 n_pub+=1
                                 for coauthor in authors:
                                      if not( coauthor.text == name):
                                           n_cauth+=1

                                           Coauthors.append(coauthor.text)

        if choix == 'author':
             Nombres_pub_coth.append(n_pub)
             Nombres_pub_coth.append(n_cauth)
             return Nombres_pub_coth
        elif choix == 'publications':
             return Publications
        elif choix =='coauthor':
             return Coauthors
```

Cette fonction nous permet de récupérer un auteur, ses publications ou ses co-auteurs en fonction de la valeur de ***choix***. Le résultat est obtenu en analyse les auteurs de chaque publication (qui fait partie du groupe : articles, incollections ...etc), et en comparant le nom des auteurs avec celui saisi dans la route.

6. Définition de la fonction distance :

```
#Fonction qui calcule la distance entre 2 auteurs
def distance(name_origin,name_destination):
    d=[]
    pubs= getInfo_by_Author(name_origin, 'publications')

    for pub in pubs:

            authors=pub.findall(target_Elements[0])#author
            found = False
            for author in authors:
                if(author.text == name_origin):
                    org= authors.index(author)


                elif (author.text== name_destination):

                        dest=authors.index(author)
                        found = True
#ajouter cette distance à une liste des distances possibles entre 2 auteurs
            if found:
                d.append( abs(dest-org))

    if(len(d)!=0) :
        result = min(d)
    else:
        result = Error_distance
    return result
```

Cette fonction nous permet de rechercher pour chaque publication, le nom de deux auteurs et de calculer la distance les séparant et de l'ajouter (si elle existe) à une liste. Nous retournons la distance la plus petite de cette liste car deux auteurs peuvent être dans deux publications différentes et donc avoir deux distances différentes.
Le calcul de la distance se fait en vérifiant la présence des deux auteurs recherchés dans chaque publication, d'extraire leurs indices et ensuite les soustraire pour obtenir la distance.

7. Définition de la fonction getAuthors_by_string :

```
#Fonction qui permet de retrouver un auteur en introduisant une sous séquence de son nom
def getAuthors_by_string(searchString):

    result=[]
    for t in publications:
        for pubs in root.findall(t):

            authors= pubs.findall(target_Elements[0])#author

            for author in authors:


                if( searchString != '*'):

                    if  str(author.text).find(searchString.lower() )!=-1 or str(author.text).find(searchString.upper())!=-1 or str(author

                        result.append((author.text))


                else:
                    result.append(author.text)



    if len(result)!=0:
        result =list(set(result))
    else:
        result ='No result'
    print(len(result))

    return result
```

Cette fonction nous permet de récupérer tous les noms auteurs contenant la sous-séquence (ou caractère) d'entrée. En vérifiant si la chaîne de caractères appartient au groupe de caractères constituant le nom de chaque auteur.

Remarques :

- Le nom d'un auteur peut être présent dans plusieurs publications. Pour éviter la duplication lors de la recherche de cet auteur, nous avons utilisé l'instruction suivante qui renvoie des noms d'auteurs distinct :

```
if len(result)!=0:
    result =list(set(result))
else:
    result ='No result'
print(len(result))

return result
```

- Quand deux auteurs ne sont jamais coauteurs dans la base de données, la fonction **distance()**, fait un calcul sur une liste vide qui peut engendrer une erreur que nous avons gérée avec l'instruction suivante :

```
    if(len(d)!=0) :
        result = min(d)
    else:
        result = Error_distance
```

8. Définition des routes :

Toutes les routes utilisent le contrôle de conformité des noms en faisant appel à la fonction **check_name()**.

```
#Question1 : prendre en entrée le nom de l'auteur et retourner ['nom_auteur', nbr_publication,
@route('/authors/<name>')
def authors(name):
    validation = check_name(name)
    #print(name)
    if(validation):

        result=getInfo_by_Author(name,'author')

    else:

       result = Error_name
    #print(result)
    return dumps(result)


#Question2 : prendre en entrée le nom de l'auteur et retourner la liste de ses publications
@route('/authors/<name>/publications',method = 'GET')
def list_pub(name):
    validation = check_name(name)
    if validation:
        result=[]
        result_s=[]
        s = getInfo_by_Author(name,'publications')
        if len(s)!=0:
            for pub in s :
                result_s.append(pub.find('title').text)
                result = limit(result_s)
        else:
            result =['No result']

    else :

        result = Error_name

    return dumps(result)
```

La route '/authors/<name>' utilise la fonction **getInfo_by_Author()** pour récupérer le nom de l'auteur, le nombre de publications et de coauteurs en précisant le nom de l'auteur dans la <name>, une liste ['nom_auteur', nombre_publications, nombre_coauteurs] est retournée.

La route '/authors/<name>/publications' utilise la fonction **getInfo_by_Author()** pour récupérer les publications de l'auteur concerné et la renvoie comme résultat.

```
#Question4 : prendre en entrée le nom d'un auteur et retourner la liste de ses co-auteurs
@route('/authors/<name>/coauthors') #Prend en entrrée le nom d'un auteur puis liste ces co-auteurs
def coauthors(name):
    validation = check_name(name)
    if validation:
        s=getInfo_by_Author(name,'coauthor')
        if len(s)!=0:
            result = limit(s)
        else:
            result =['No result']
    else:
        result = Error_name

    return dumps(result)


#Question5 : Prendre une sous séquence et retourner tous les auteurs dont le nom contient cette séquence
@route('/search/authors/<searchString>') # Prend comme paramètre un caractère puis retourne tous les nom des auteurs qui c
def search(searchString):
    result = getAuthors_by_string(searchString)
    return dumps(result)

#Question6 : Prendre en entrée le noms de 2 auteurs et retourner la distance les séparant
@route('/authors/<name_origin>/distance/<name_destination>')
def dist(name_origin,name_destination):
    validation = check_name(name_origin) and check_name(name_destination)
    if validation:
        print(name_origin,name_destination)
        result= distance(name_origin,name_destination)
        print(result)
    else:
        result= Error_name
    return dumps(result)
```

La route '/authors/<name>/coauthors' utilise la fonction **getInfo_by_Author()** pour récupérer la liste des coauteurs de l'auteur concerné et la renvoyer comme résultat.

La route '/search/authors/<searchString>' utilise la fonction **getAuthor_by_string()** pour récupérer la liste des noms d'auteurs contenant la sous séquence saisie dans <searchString> et la retourner comme résultat.

La route 'authors/<name_origin>/distance/<name_destination>' fait appel à la fonction **distance()** pour calculer la distance minimale entre les deux auteurs et la renvoyer comme résultat.

9. Lancement du serveur :

```
#Lancement du serveur accessible : https//localhost:8085
run(host='localhost', port=8085)
```

III. Exercice 2 : le test unitaire

```
class TestAPIMethods(unittest.TestCase):
    server_ip= "127.0.0.1"
    server_port = 8085

#Vérifier que la route 1 renvoit bien Daniel Gruss avec 13 publications et 32 co-auteurs
    def test_authors(self):
        req = get(f"http://{self.server_ip}:{self.server_port}/authors/Daniel Gruss")
        result = loads(req.text)
        self.assertEqual(result[0],'Daniel Gruss')
        self.assertEqual(result[1],13)
        self.assertEqual(result[2],32)

#Vérifier que l'auteur Daniel Gruss possède 13 publications
    def test_publications(self):
        req = get(f"http://{self.server_ip}:{self.server_port}/authors/Daniel Gruss/publications")
        result= loads(req.text)
        self.assertEqual(len(result),13)

#Vérifier que l'auteur Daniel Gruss à bien 32 co-auteurs
    def test_coauthors_by_name(self):
        req = get(f"http://{self.server_ip}:{self.server_port}/authors/Daniel Gruss/coauthors")
        result = loads(req.text)
        self.assertEqual(len(result), 32)

#Vérifier que la sous séquence 'Daniel' est présente dans 2763 noms d'auteurs
    def test_search_authors(self):
        req = get(f"http://{self.server_ip}:{self.server_port}/search/authors/Daniel")
        result = loads(req.text)
        self.assertEqual(len(result),2763)

#Vérifier que 5 publications de l'auteur Wei Wei ont été affichées uniquement (à partir de la 50ème publication)
    def test_start_count(self):
        req = get(f"http://{self.server_ip}:{self.server_port}/authors/Wei Wei/coauthors?start=50&count=5")
        result= loads(req.text)
        self.assertEqual(len(result),5)

#Vérifier que la distance minimale entre Daniel Gruss et Daniel Genkin est égale à 1
    def test_distance(self):
        req = get(f"http://{self.server_ip}:{self.server_port}/authors/Daniel Gruss/distance/Daniel Genkin")
        result= loads(req.text)
        self.assertEqual(result,1)

if __name__ == '__main__':
    unittest.main()
```

Ce programme Unitest permet de tester nos routes et de comparer leurs résultat avec ceux obtenus manuellement.

Pour les routes nous avons pris comme exemple l'auteur 'Daniel Gruss' qui a 13 publications, 32 coauteurs, et une distance minimale avec 'Daniel Genkin' de 1.

Nous avons également utilisé l'auteur 'Wei Wei' qui a 677 coauteurs afin de tester les paramètres **start** et **count**, et l'affichage de 100 éléments maximum.

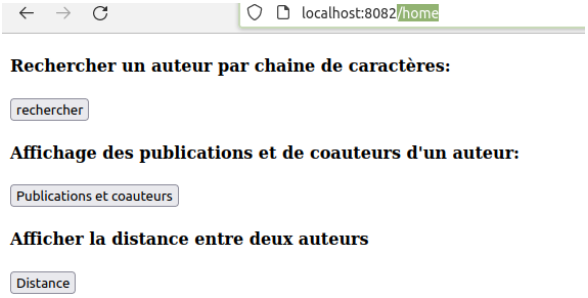
```
mohamed@Mohamed:~$ cd Bureau/Mini_projet/Final/A_rendre
mohamed@Mohamed:~/Bureau/Mini_projet/Final/A_rendre$ python3 Exercice2_fi.py
......
----------------------------------------------------------------------
Ran 6 tests in 7.833s

OK
mohamed@Mohamed:~/Bureau/Mini_projet/Final/A_rendre$
```

Les tests unitaires se sont déroulés parfaitement, et les résultats obtenus manuellement concordent avec les résultats de ces tests.

IV. Exercice 3: l'interface graphique

Dans cet exercice, nous avons crée une interface graphique qui permet à l'utilisateur d'accéder aux fonctionnalités de l'API de l'exercice 1.

```
@route("/home", method=['GET','POST'])
def home():
    return '''
    <form action="/input" method="post">
        <h4> Rechercher un auteur par chaine de caractères: </h4>
        <a href="/input_car"><input type="button" value="rechercher" /></a>
        <h4> Affichage des publications et de coauteurs d'un auteur: </h4>
        <a href="/input_auteur"><input type="button" value="Publications et coauteurs" /></a>
        <h4> Afficher la distance entre deux auteurs </h4>
        <a href="/input_auth_d"><input type="button" value="Distance" /></a>
        </form>
        '''
```



Nous avons crée une page d'accueil utilisant la route '/home' qui offre trois choix à l'utilisateur : la recherche d'un auteur, les publications et les coauteurs d'un auteur, ainsi que la distance entre deux auteurs.

```python
@route("/input_auteur")
def input_auteur():
    return '''
        <form action="/input_auteur" method="post">
        <h4> Donner le nom d'un auteur </h4>
        Auteur : <input name="name" type="text" />
        <input value="Search" type="submit" />
        </form>
        '''

@route("/input_auteur", method='POST')
def on_intput_auteur():

    name= request.forms.getunicode('name')
    req=get(f"http://{server_ip}:{server_port}/authors/{name}/publications")
    req1=get(f"http://{server_ip}:{server_port}/authors/{name}/coauthors")
    ce=check_error(req.text)
    ce1=check_error(req1.text)
    if len(ce)==2 and len(ce1)==2:
        return ce[0]+ce1[0]
    else:
        tri=disp_List(req.text)
        tri2=disp_List(req1.text)
        return f"<h2>Publications de {name} triés : {tri} </h2> <h2> les co-auteurs de {name} triés: {tri2} </h2>"
```

Quand l'utilisateur choisit d'afficher les publications et les coauteurs d'un auteur par exemple, une nouvelle route est enclenchée '/input_auteur' renvoyant une page où l'utilisateur doit saisir le nom d'un auteur, et d'appuyer sur le bouton **search**.
La route '/input_auteur' avec la méthode POST faisant appel à la fonction **on_input_auteur()** permet de récupérer le contenu à partir de l'API de l'exercice 1.

Publications de Daniel Gruss triés :

ConTExT: A Generic Approach for Mitigating Spectre.
Donky: Domain Keys - Efficient In-Process Isolation for RISC-V and x86.
Evolution of Defenses against Transient-Execution Attacks.
How Trusted Execution Environments Fuel Research on Microarchitectural Attacks.
LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection.
Malware Guard Extension: abusing Intel SGX to conceal cache attacks.
Meltdown: reading kernel memory from user space.
Plundervolt: How a Little Bit of Undervolting Can Create a Lot of Trouble.
Plundervolt: Software-based Fault Injection Attacks against Intel SGX.
RAMBleed: Reading Bits in Memory Without Accessing Them.
Spectre attacks: exploiting speculative execution.
The Evolution of Transient-Execution Attacks.
Speculative Dereferencing of Registers: Reviving Foreshadow.

les co-auteurs de Daniel Gruss triés:

Anders Fogh
Andrew Kwong
Berk Sunar
Cl
Claudio Canella
Daniel Genkin
Daniel Moghimi
David Oswald
David Schrammel
Flavio D. Garcia
Florian Kargl
Frank Piessens
Jann Horn

Après avoir appuyé sur le bouton **search**, la fonction **on_input_auteur()**, au biais de la méthode POST, récupère le nom de l'auteur qui va être utilisé pour générer deux requêtes GET vers l'API pour obtenir la liste des publications et la liste des coauteurs de cet auteur.
Pour l'affichage des données récupérées, deux fonctions ont été implémentées : **check_error ()** et **disp_List()**

```python
def disp_List(resultat):
    resultat = resultat.strip('[')
    resultat = resultat.strip(']')
    resultat= resultat.replace('"','')
    result = resultat.split(",")
    if result != ['']:
        result = sorted(result)
        Titre = ""
        Tabp='<table border="1" cellpadding="10" cellspacing="1" width="100%">'
        for j in result:
                Titre+='<tr><td>'
                Titre+=str(j)
                Titre+='</tr></td>'
        Tabp+=Titre+'</table>'

        return Tabp
    else:
        print(result)
        result ='No result'
        return result



def check_error(resultat):
    if(resultat.find('ERROR')!= -1):
        resultat = loads(resultat)

        Tab='<table border="1" cellpadding="10" cellspacing="1" width="100%"><tr><th>Solution</th>'
        Tab+=f'<tr><td>{resultat}</td></tr>'
        Tab+= '<html><body>'+'<BODY BGCOLOR="#FF0000">'+Tab+'</body></html>'
        return [Tab,'']
    else:
        return resultat
```

La réponse obtenue par la requête GET constitue une chaîne de caractère, que nous avons transformé à l'aide de **disp_List()** afin d'avoir un format de liste et de l'afficher par la suite dans un tableau HTML.

La réponse peut être un format d'erreur ou un format de données. Afin de dissocier ces deux cas, nous avons utilisé la fonction **check_error()** qui renvoie deux éléments dont le premier contient l'erreur.
Si la taille de cette liste est de 2, alors la réponse obtenue est un format d'erreur. Sinon, la liste contient un format de données à afficher.

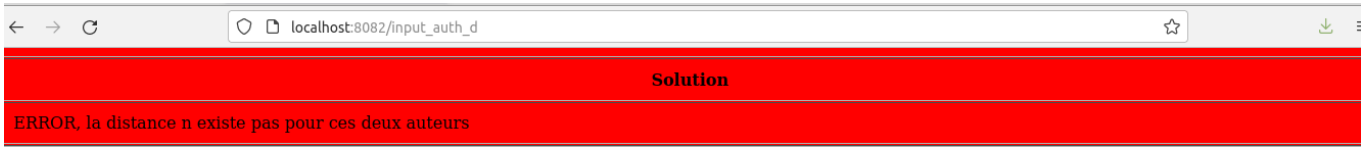
Affichages de format d'erreurs possibles :

- Erreur de saisie du nom d'auteur :

Quand le nom d'auteur ne respecte pas l'expression régulière utilisé dans l'API de l'exercice 1, un format d'erreur est généré par la fonction **check_error()**, puis affiché à l'écran en proposant une solution à l'utilisateur.

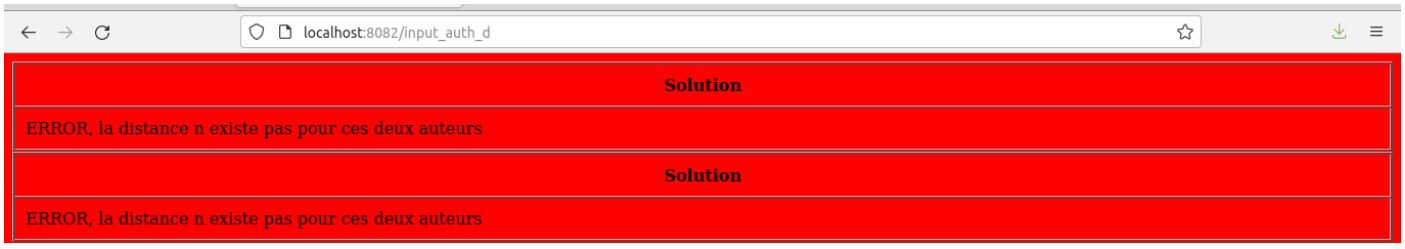
De la même manière les étapes précédentes sont utilisées pour chaque option de recherche qui se trouve dans la page d'accueil à savoir : la distance, et la recherche par caractère.

- Erreur lors du calcul de distance :



Si les deux auteurs ne sont jamais coauteurs cela veut dire que la distance n'existe pas entre eux et donc l'API renvoie ce format d'erreur.

Problèmes rencontrés :



Lors de l'affichage du format d'erreur par l'API web, une requête peut être générée plusieurs fois vers l'API, et donc l'affichage de l'erreur se multiplie pour chaque requête.