TP n°1 en Systèmes distribues

Création d'API Web RESTful à l'aide de Flask et Python

– Flask est un framework micro web largement utilisé pour créer des API en Python. Il s'agit d'un framework Web simple mais puissant, conçu pour démarrer rapidement et facilement, avec la possibilité d'évoluer vers des applications complexes.

– API REST et services Web : Les API REST écoutent les méthodes HTTP telles que GET, POST et DELETE pour savoir quelles opérations effectuer sur les ressources du service Web.

Une ressource est toute donnée disponible dans le service Web qui peut être consultée et manipulée avec des requêtes HTTP à l'API REST. La méthode HTTP indique à l'API quelle action effectuer sur la ressource. Bien qu'il existe de nombreuses méthodes HTTP, les quatre méthodes répertoriées ci-dessous sont les plus couramment utilisées avec les API REST :

Méthode HTTP	Description
GET	Récupère une ressource existante
POST	Créer une nouvelle ressource.
PUT	Mettre à jour une ressource existante
DELETE	Supprimer une ressource.

1. Lancer un outil de Python (Spyder ou notebook…),

installer Flask using pip

```
pip install Flask
```

2. Créer le fichier tp11.py et saisir le code suivant :

```
import flask
app = flask.Flask(__name__)
app.config["DEBUG"] = True

@app.route('/', methods=['GET'])        # le type de requêtes http autorisées.
def home():
        return "<h1> API RestFul</h1>
        <p> Création d'API Web RESTful à l'aide de Flask et Python.</p>"

if __name__ == '__main__':
            app.run(port=5005)
```

– import Flask : Cette instruction permet d'importer la bibliothèque Flask, qui est disponible par défaut sous Anaconda.
– app = flask.Flask(__name__) : Crée l'objet application Flask, qui contient les données de l'application et les méthodes correspondant aux actions susceptibles d'être effectuées sur l'objet.
– app.config[« DEBUG » = True] : lance le débogueur, ce qui permet d'afficher un message autre que « Bad Gateway » s'il y a une erreur dans l'application.
– L'instruction : @app.route('/', methods=['GET']) apparaissant dans le programme indique à Flask que la fonction home correspond au chemin /.
– if __name__ == '__main__' → __name__ est une variable spéciale en Python qui prend la valeur du nom du script. Cette ligne garantit que notre application Flask ne s'exécute que lorsqu'elle est exécutée dans le fichier principal et non lorsqu'elle est importée dans un autre fichier.
– app.run( ) : permet d'exécuter l'application.

3. Avec un éditeur de texte, créer le fichier data.json

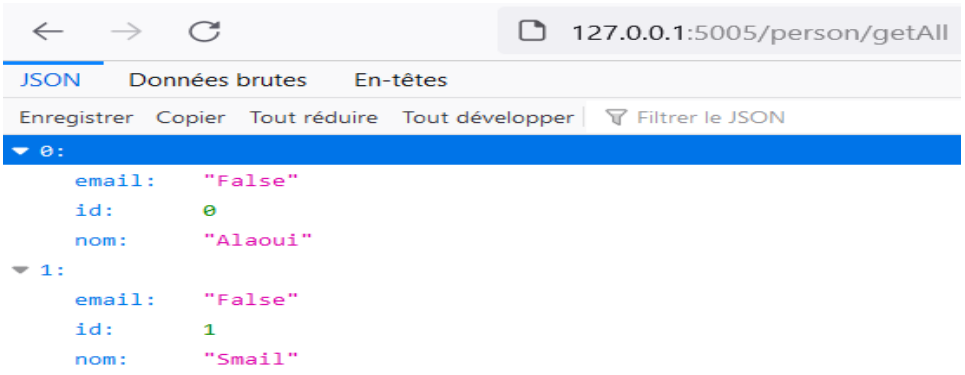
```
[
{"id": 0, "nom": "Alaoui", "email": "False"},
{"id": 1, "nom": "Smail", "email": "False"}
]
```

4. Ajouter le code suivant au fichier tp11.py

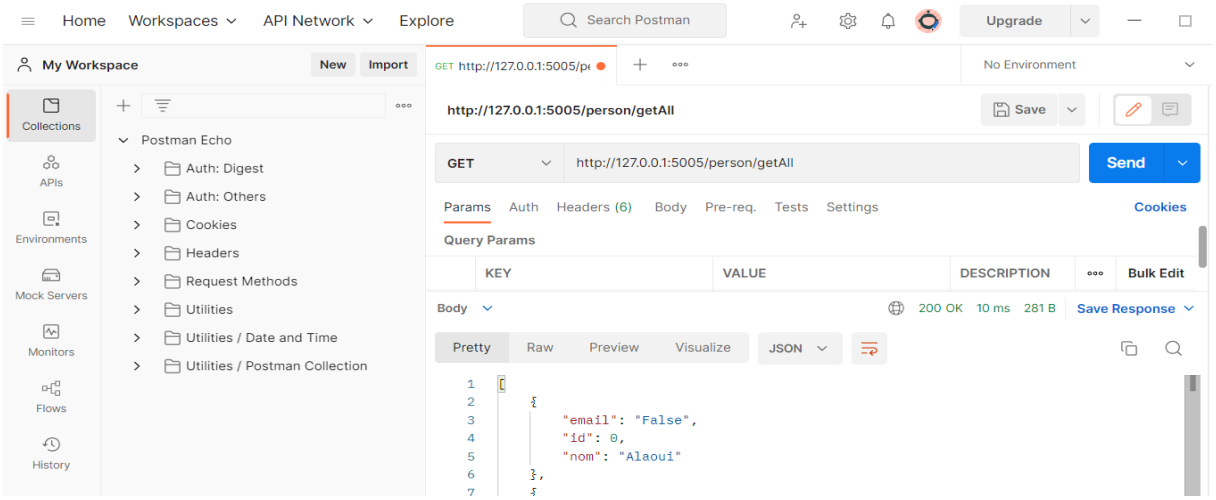
```
import json
from flask import Flask, request, jsonify

nom_file='D:/TPFlask/data.json'

@app.route('/person/getAll', methods=['GET'])
def get_all():
        with open(nom_file) as f:
                data = json.load(f)
        return jsonify(data)
```

La valeur de retour d'une fonction dans une application Flask doit être sérialisable JSON. La fonction jsonify encapsule json.dumps() pour transformer la sortie JSON en un objet Response avec le type mime application/json.

Exécuter le fichier, et sur navigateur web taper l'adresse l'url



Vous pouvez utiliser outil Postman pour tester les méthodes du Restful (télécharger et installer Postman)



5. Compléter les méthodes suivantes :

get_records_id () : retourne l'enregistrement d'id=id du paramètre de l'url
http://127.0.0.1:5005/person/getId?id=1
post_records () : ajoute un enregistrement à la fin du fichier json.data
put_records_id() : modifie l'enregistrement d'id=id du paramètre de l'url
del_records_id () : supprime l'enregistrement d'id=id du paramètre de l'url

Pour accéder aux données de la requête :

request.data	Accéder aux données de la requête entrante sous forme de chaîne
request.args	Accéder aux paramètres d'URL.
request.form	Accéder aux paramètres du formulaire
request.values	Renvoie CombinedMultiDict qui combine les arguments et la forme
request.json	Renvoie les données JSON analysées si le type mime est application/json
request.files	Renvoie l'objet MultiDict qui contient tous les fichiers téléchargés. Chaque clé est le nom du fichier et la valeur est l'objet FileStorage.
request.authorization	Renvoie un objet de la classe Authorization .Il représente un en-tête Authorization envoyé par le client

```
@app.route('/person/getId', methods=['GET'])
def get_records_id():
   if 'id' in request.args:
      id = int(request.args['id'])
   else:
      return " Erreur: id n'existe pas."
….
```

```
@app.route('/person/post', methods=['GET','POST'])
def post_record():
        with open(nom_file) as f:
                data = json.load(f)
        results=[]
        for item in data:
                results.append(item)
….
```

```
@app.route('/person/put', methods=['GET','PUT'])
def put_record_id():
   if 'id' in request.args:
      id = int(request.args['id'])
   else:
      return " Erreur: id n'existe pas."
……
```

```
@app.route('/person/del', methods=['GET','DELETE'])
def del_record_id():
   if 'id' in request.args:
      id = int(request.args['id'])
   else:
      return " Erreur: id n'existe pas."
……
```

6. Transférer les données format json vers une base données (persistance) : La base de données utilisée est SQLite, un moteur de base de données très léger et disponible sous Python par défaut.

Créer un autre fichier tp12.py, et redéfinir les méthodes définies dans tp11.py

```
# Uniquement si sqlite n'est pas installé
# pip installer db-sqlite3
```

```
import json
from flask import Flask, request, jsonify
import sqlite3
```

```
app = Flask(__name__)
nom_file='D:/TPFlask/persons'

def dict_factory(cursor, row):
    d = {}
    for idx, col in enumerate(cursor.description):
        d[col[0]] = row[idx]
    return d

def get_users():
    conn = sqlite3.connect('persons')
    conn.row_factory = dict_factory
    cur = conn.cursor()
    all_users = cur.execute("SELECT * FROM users").fetchall()
    return all_users

@app.route('/person/getAll', methods=['GET'])
def get_all():
    all_users=get_users()
    return jsonify(all_users)
……

@app.route('/person/getId', methods=['GET'])
def get_records_id():
….
```

```
@app.route('/person/post', methods=['GET','POST'])
def post_record():
….
```

```
@app.route('/person/put', methods=['GET','PUT'])
def put_record_id():
……
```

```
@app.route('/person/del', methods=['GET','DELETE'])
def del_record_id():
    ……
```