Introduction à Flask

Université d'Angers - M2 Data Science
11 décembre 2023

Structure du fichier app.py

- **Dans un dossier concernant une API Flask, le fichier `app.py` définit les différentes routes de l'API.**
- **Principe général :**
  - **`app = Flask(_name_)` définit l'appli Web**
  - **pour définir une route de l'API :**
    - **on la déclare avec le décorateur `@app.route("/route1", methods = ["GET"])` `[ou "POST"]`**
    - **lors d'un appel à cette route par une requête, la fonction décorée est exécutée et l'API renvoie la sortie de cette fonction**
  - **app.run() permet de faire fonctionner l'appli**
  - **pour faire tourner le service Web (faire fonctionner l'API), se placer dans le dossier contenant `app.py` et lancer la commande `python app.py` dans le terminal.**

- **NB :** la fonction décorée peut aussi renvoyer un tuple (sortie voulue, code erreur HTTP) pour informer le client d'une erreur éventuelle. Par défaut, le code erreur est 200 ("pas d'erreur") comme dans l'ex. ci-contre.

```
from flask import Flask, request, jsonify

# déf de l'appli web

app = Flask(__name__)

# déf d'une route nommée "route1" avec la méthode
# GET sans paramètres

@app.route("/route1", methods = ["GET"])
def resultat_route1():
      return "hello world!"
      # équivaut à return "hello world!", 200


if __name__ == '__main__':
    # run app in debug mode on port 5000
    app.run(debug = True, port = 5000)
```

Test d'une route GET simple

- **Pour tester la route nommée "`route1`" qu'on vient de définir, il suffit de :**
  - **lancer le service avec** `python app.py`
  - **ce qui donne ceci (indiquant notamment l'URL sur laquelle tourne le service, ici** `http://127.0.0.1:5000/` **qui est simplement l'adresse de mon PC en local appelable uniquement depuis mon PC, avec le port 5000 qui est le port par défaut de Flask) :**

```
mohamed@mohamed-ThinkPad-T480s:~/Documents/flask_training/simple_app$ python app.py
 * Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

  - **envoyer la requête GET, i.e. aller dans un navigateur Internet à l'adresse** `http://127.0.0.1:5000/route1` **constater l'affichage de la chaîne de caractères "hello world!" comme défini dans** `app.py`
  - **constater que la sortie du terminal a ajouté cette nouvelle ligne avec le statut HTTP 200 indiquant que la requête a fonctionné :**

```
127.0.0.1 - - [02/Nov/2021 15:46:34] "GET /route1 HTTP/1.1" 200 -
```

Route GET avec paramètres

- On peut demander des paramètres au client dans une requête GET. Ci-contre, la route "`/print-param`" demande les paramètres `param1` et `param2` pour les afficher ensuite.
- Le client envoie la requête avec l'URL : `http://[url]:[port]/print-param?param1=[valeur1]&param2=[valeur2]`
- On récupère les valeurs des paramètres dans le code côté serveur grâce à :
  - `flask.request.args[nom_paramètre]` si on veut créer une erreur HTTP 400 si le paramètre n'est pas dans la requête du client
  - `flask.request.args.get(nom_paramètre, default = None)` si on veut éviter cette erreur 400 (auquel cas la valeur par défaut du paramètre est dans l'argument `default`).

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

# déf de l'appli web

app = Flask(__name__)

# déf d'une route nommée "print-params" avec la
méthode GET avec paramètres "param1" et "param2"

@app.route("/print-param", methods=['GET'])
def affiche_parametres():

    # if key doesn't exist, returns None
    val_param1 = request.args.get("param1")

    # if key doesn't exist, returns a 400, bad
    # request error
    val_param2 = request.args["param2"]

    return f"<p>{val_param1}: {val_param2}<p>"
```

Route POST avec JSON

- Ci-contre, la route “/predictions” demande le paramètre `param1` et récupère le contenu du fichier JSON envoyé par la requête POST grâce à `request.json`.

- On renvoie nous-mêmes un JSON avec `flask.jsonify` qui convertit un dict Python au format JSON.

- Charge au client de nous envoyer une requête POST avec le JSON voulu.

- Contrairement aux requêtes GET, les requêtes POST ne peuvent être traitées par un navigateur web classique (sauf avec Postman) => Comment envoyer une requête POST avec JSON en utilisant Python ? Comment tester nous-mêmes le code ?

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

app = Flask(__name__)

# déf d’une route nommée “predictions” avec la méthode
POST avec paramètre “user-name” et envoi de JSON par
le client

@app.route("/add-json-value", methods=['POST'])
def add_json_value():

    # récup. du JSON envoyé par la requête POST
    # sous forme de dictionnaire
    json_data = request.json

    if json_data is not None:
      json_data['test'] = 'ok'
    else:
      json_data['test'] = 'error'

    return jsonify(json_data)
```

Test d'une requête POST

- Pour tester la route POST qu'on vient de créer, il suffit d'exécuter le script Python ci-contre :

- On utilise la librairie `requests`. Elle propose une fonction `post()` qui permet d'envoyer une requête POST. Elle prend 4 arguments principaux :
  - `url` = l'URL du service, formée sur le même principe que pour une requête GET (dans notre cas IP locale, port 5000 et route 'predictions')
  - `json` = le JSON à envoyer si besoin
  - `data` = des données à envoyer si besoin (types Python possibles : dict, list of tuples, bytes, file object)
  - `params` = des paramètres à envoyer dans l'URL de la requête avec `?param=valeur&...` (types possibles : dict, list of tuples, bytes).

```
import requests
import pandas as pd

r = requests.post(
    url = 'http://127.0.0.1:5000/predictions',
    json = pd.read_csv('new_data.csv').to_dict('records'),
    params = {'user_name':' Sabrina'}
)

print(r.text) # contenu de la réponse
r.json
```

- Elle renvoie un objet de la classe `requests.Response` avec des attributs comme :
  - `r.text` : le contenu de la réponse sous forme de texte Unicode
  - `r.encoding` : l'encodage de r.text comme 'utf8'...
  - `r.status_code` : le code HTTP de la réponse
  - `r.json` : le JSON inclus dans la réponse, s'il existe.


FIN