Développer une application serveur avec Flask

https://palletsprojects.com/p/flask/

Flask est un micro-framework d'application web Python construit sur la bibliothèque WSGI (Web Server Gateway Interface) de Werkzeug . Flask peut être "micro", mais il est prêt pour une utilisation de production sur une variété de besoins.

Le «micro» dans le micro-cadre signifie que Flask vise à garder le noyau simple mais extensible.

Flask ne prendra pas beaucoup de décisions pour vous, comme la base de données à utiliser, et les décisions prises sont faciles à modifier. Tout est à vous, pour que Flask puisse être tout ce dont vous avez besoin et rien de ce que vous n'avez pas.

La communauté prend en charge un riche écosystème d'extensions pour rendre votre application plus puissante et plus facile à développer. À mesure que votre projet se développe, vous êtes libre de prendre les décisions de conception appropriées à vos besoins. Voici quelques exemples permettant de comprendre son fonctionnement

Nous allons définir les routes et y associer une fonction qui sera appelée à chaque fois qu'une requête sera envoyée :

Utilisez pip pour installer Flask :

```
pip install flask
```

I. Basics

Premier exemple pour la route principale :

```
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return "Hello, World !"


if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

Une route /coucou :

```
@app.route('/coucou')
def coucou():
    return "coucou salut  !"
```

Une route qui renvoie des infos :

```
from flask import jsonify
@app.route('/json')
def json():
    return jsonify({'key' : 'value', 'listkey' : [1,2,3]})
```

Une route avec un paramètre :

```
@app.route('/valeur/<valeur1>')
def affichervaleur(valeur1):
    return str(valeur1)
```

Une route avec deux paramètres :

```
@app.route('/somme/<valeur1>/<valeur2>')
def somme(valeur1,valeur2):
    resultat = valeur1 + valeur2
    return str(resultat)
```

Une route avec des paramètres par défaut :

```
@app.route('/home', methods=['POST', 'GET'], defaults={'name' : 'Default'})
@app.route('/home/<string:name>', methods=['POST', 'GET'])
def home(name):
    return '<h1>Hello {}, you are on the home page!</h1>'.format(name)
```

Une route avec des paramètres dans la query :

```
from flask import request
@app.route('/query')
def query():
    name = request.args.get('name')
    location = request.args.get('location')
    return '<h1>Hi {}. You are from {}.You are on the query page!</h1>'.format(name,
location)
```

On saisira dans l'explorateur http://localhost:5000/query?name=Pierre&location=Toulouse

Pour créer un formulaire :

```
@app.route('/theform')
def theform():
    return '''<form method="POST" action="/process">
                <input type="text" name="name">
                <input type="text" name="location">
                <input type="submit" value="Submit">
            </form>'''

@app.route('/process', methods=['POST'])
def process():
    name = request.form['name']
    location = request.form['location']

    return '<h1>Hello {}. You are from {}. You have submitted the form
successfully!<h1>'.format(name, location)
```

Pour créer un formulaire en une route :

```
@app.route('/theform2', methods=['GET', 'POST'])
def theform2():

    if request.method == 'GET':
        return '''<form method="POST" action="/theform2">
                    <input type="text" name="name">
                    <input type="text" name="location">
                    <input type="submit" value="Submit">
                </form>'''
    else:
        name = request.form['name']
        location = request.form['location']

        return '<h1>Hello {}. You are from {}. You have submitted the form
successfully!<h1>'.format(name, location)
```

Si on veut rediriger vers une autre page on peut écrire :

```
from flask import redirect
return redirect(url_for('home', name=name, location=location))
```

Pour créer une session :

Il faut une clé de chiffrement

```
from flask import session

app = Flask(__name__)
app.config['SECRET_KEY'] = 'laclesecrete' #pour chiffrer les cookies


@app.route('/')
def index():
    session.pop('name', None)
    return '<h1>Hello, World!</h1>'

@app.route('/home', methods=['POST', 'GET'], defaults={'name' : 'Default'})
@app.route('/home/<string:name>', methods=['POST', 'GET'])
def home(name):
    session['name'] = name
    return '<h1>Hello {}, you are on the home page!</h1>'.format(name)

@app.route('/json')
def json():
    if 'name' in session:
        name = session['name']
    else:
        name = 'NotinSession!'
    return jsonify({'key' : 'value', 'listkey' : [1,2,3], 'name' : name})
```

II. Templates

Les pages seront obligatoirement dans un dossier templates

```
from flask import render_template

@app.route('/')
def index():
    return render_template('index.html')
```

Avec la page web index.html suivante :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Accueil</title>
  </head>
  <body>
    <h1>Ceci est la page d'accueil.</h1>
  </body>
</html>
```

Une route qui intègre des résultats dans une page web :

```
@app.route('/affiche_somme/<valeur1>/<valeur2>')
def affiche_somme (valeur1,valeur2):
    d = date.today().isoformat()
    resultat = int(valeur1)+int(valeur2)
    return render_template("somme.html",la_date=d,v1=valeur1, v2=valeur2, res=resultat)
```

Avec la page web somme.html suivante :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Somme</title>
  </head>
  <body>
    <h1>La date d'aujourd'hui est : {{ la_date }}</h1>
    <h1>Le résultat de {{ v1 }} + {{ v2 }} est {{ res }}.</h1>
  </body>
</html>
```

Pour remplacer le formulaire vu précédemment :

```
@app.route('/theform', methods=['GET', 'POST'])
def theform():

    if request.method == 'GET':
        return render_template('form.html')
    else:
        name = request.form['name']
        location = request.form['location']

        return redirect(url_for('home', name=name, location=location))
```

Avec la page form.html suivante :

```
<html>
<head>
  <title>The Form</title>
</head>
<body>
  <h1>Please fill out the form</h1>
  <form method="POST" action="/theform">
            <input type="text" name="name">
            <input type="text" name="location">
            <input type="submit" value="Submit">
  </form>
</body>
</html>
```

On peut utiliser des conditions dans les templates :

```
@app.route('/home', methods=['POST', 'GET'], defaults={'name': 'Default'})
@app.route('/home/<string:name>', methods=['POST', 'GET'])
def home(name):
    session['name'] = name
    return render_template('home.html', name=name, display=False)
```

Avec la page home.html suivante :

```
<h1>Hello {{ name }}, you are on the home page!</h1>
{% if display %}
<h2>This is being displayed!</h2>
{% else %}
<h2>This is not being displayed!</h2>
{% endif %}
```

On peut utiliser des boucles dans les templates :

```
@app.route('/home', methods=['POST', 'GET'], defaults={'name': 'Default'})
@app.route('/home/<string:name>', methods=['POST', 'GET'])
def home(name):
    session['name'] = name
    return render_template('home.html', name=name, display=False, mylist=['one', 'two',
'three', 'four'], listofdictionaries=[{'name' : 'Zach'}, {'name' : 'Zoe'}])
```

En rajoutant à la page home.html :

```
...

{% for x in mylist %}
<h3>{{ x }}</h3>
{% endfor %}

{% for x in listofdictionaries %}
<h4>{{ x.name }}</h4>
{% endfor %}
```

III. BDD

Bases de données :

On crée une base de données sqlite avec adminSQL qui s'appelle data.db

Créer une table users avec id, name, location.

Connection à la base de données

```
from flask import g
import sqlite3
def connect_db():
    sql = sqlite3.connect('data.db')
    sql.row_factory = sqlite3.Row #avoir un dict plutôt qu'un tuple
    return sql


def get_db():
    if not hasattr(g, 'sqlite_db'):
        g.sqlite_db = connect_db()
    return g.sqlite_db


@app.teardown_appcontext
def close_db(error):
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()
```

Pour voir un résultat :

```
@app.route('/viewresults')
def viewresults():
    db = get_db()
    cur = db.execute('select id, name, location from users')
    results = cur.fetchall()
    if len(results) == 0 :
        return '<h1>No database values.</h1>'
    else :
        return '<h1>The ID is {}. The name is {}. The location is
{}.</h1>'.format(results[0]['id'], results[0]['name'], results[0]['location'])
```

Pour insérer un résultat :

```
@app.route('/theform', methods=['GET', 'POST'])
def theform():
    if request.method == 'GET':
        return render_template('form.html')
    else:
        name = request.form['name']
        location = request.form['location']

        db = get_db()
        db.execute('insert into users (name, location) values (?, ?)', [name, location])
        db.commit()

        return redirect(url_for('home', name=name, location=location))
```

Pour voir tous les résultats :

```
@app.route('/home', methods=['POST', 'GET'], defaults={'name' : 'Default'})
@app.route('/home/<string:name>', methods=['POST', 'GET'])
def home(name):
    session['name'] = name
    db = get_db()
    cur = db.execute('select id, name, location from users')
    results = cur.fetchall()

    return render_template('home.html', name=name, display=False, \
        mylist=['one', 'two', 'three', 'four'], listofdictionaries=[{'name' : 'Zach'},
{'name' : 'Zoe'}], results=results)
```

Avec la page home.html qui contiendra :

```
<h5>Database Results!</h5>
{% for item in results %}
<h6>ID: {{ item.id }} Name: {{ item.name }} Location: {{ item.location }}</h6>
{% endfor %}
```

IV. Extras

Pour uploader un fichier :

```
@app.route('/upload')
def upload():
    return render_template('upload.html')


@app.route('/uploader', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        f = request.files['file']
        content = f.read()
        #f.save(f.filename) #si on veut le sauvegarder
        maClasse = json.loads(content)
        classe.exportExcel(maClasse, "classe.xlsx")
        return send_file("classe.xlsx", attachment_filename="classe.xlsx")
```

Avec le fichier upload.html suivant :

```
<html>
   <body>
      <form action = "http://localhost:5000/uploader" method = "POST"
         enctype = "multipart/form-data">
         <input type = "file" name = "file" />
         <input type = "submit"/>
      </form>
   </body>
</html>
```

Une route qui trace et affiche une courbe :

```
@app.route('/courbe')
def courbe():
    plt.plot([1, 2, 3, 4])
    plt.ylabel('some numbers')
    figfile = BytesIO()
    plt.savefig(figfile, format='png')
    figfile.seek(0)
    figdata_png = base64.b64encode(figfile.getvalue()).decode('ascii')

    return render_template("courbe.html",imgCourbe=figdata_png)
```

V. Créer une API

Nous allons créer une API pour gérer des membres dans une BDD

Dépendances :
Flask, Jinja2, MarkupSafe, Werkzeug, Click, itsdangerous

Outils :
SQliteAdmin : http://download.orbmu2k.de/download.php?id=19
Postman : https://dl.pstmn.io/download/latest/win64

Création du squelette de l'API

```
from flask import Flask

app = Flask(__name__)

@app.route('/member', methods=['GET'])
def get_members():
    return 'This returns all the members.'

@app.route('/member/<int:member_id>', methods=['GET'])
def get_member(member_id):
    return 'This returns one member by ID'

@app.route('/member', methods=['POST'])
def add_member():
    return 'This adds a new member.'

@app.route('/member/<int:member_id>', methods=['PUT', 'PATCH'])
def edit_member(member_id):
    return 'This updates a member by ID.'

@app.route('/member/<int:member_id>', methods=['DELETE'])
def delete_member(member_id):
    return 'This removes a member by ID.'

if __name__ == '__main__':
    app.run(debug=True)
```

Création de la BDD

Dans un autre fichier database.py

```
from flask import g
import sqlite3

def connect_db():
    sql = sqlite3.connect('')
    sql.row_factory = sqlite3.Row
    return sql

def get_db():
    if not hasattr(g, 'sqlite_db'):
        g.sqlite_db = connect_db()
    return g.sqlite_db
```

Et rajouter dans API.py

```
from .database import get_db
.....

@app.teardown_appcontext
def close_db(error):
    if hasattr(g, 'sqlite_db'):
        g.sqlite_db.close()
```

Pour ajouter un membre on va modifier add_member()

```
from flask import Flask, g, request

@app.route('/member', methods=['POST'])
def add_member():
    new_member_data = request.get_json()

    name = new_member_data['name']
    email = new_member_data['email']
    level = new_member_data['level']

    db = get_db()
    db.execute('insert into members (name, email, level) values (?, ?, ?)',
               [name, email, level])
    db.commit()

    return 'The name is {}, the email is {}, and the level is {}'.format(name, email, level)
```

Pour vérifier et renvoyer le member inséré :
modifier la fin de add_member()

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

Member_cur = db.execute('select id, name, email, level from members where name = ?', [name])
new_member = member_cur.fetchone()

return jsonify({'id' : new_member['id'], 'name' : new_member['name'], 'email' :
new_member['email'], 'level' : new_member['level']})
```

Pour récupérer tous les membres :

```
@app.route('/member', methods=['GET'])
def get_members():
    db = get_db()
    members_cur = db.execute('select id, name, email, level from members')
    members = members_cur.fetchall()

    return_values = []

    for member in members:
        member_dict = {}
        member_dict['id'] = member['id']
        member_dict['name'] = member['name']
        member_dict['email'] = member['email']
        member_dict['level'] = member['level']

        return_values.append(member_dict)

    return jsonify({'members' : return_values})
```

Pour récupérer un membre :

```
@app.route('/member/<int:member_id>', methods=['GET'])
def get_member(member_id):
    db = get_db()
    member_cur = db.execute('select id, name, email, level from members where id = ?',
[member_id])
    member = member_cur.fetchone()

    return jsonify({'member' : {'id' : member['id'], 'name' : member['name'], 'email' :
member['email'], 'level' : member['level']}})
```

Pour modifier un membre :

```
@app.route('/member/<int:member_id>', methods=['PUT', 'PATCH'])
def edit_member(member_id):
    new_member_data = request.get_json()

    name = new_member_data['name']
    email = new_member_data['email']
    level = new_member_data['level']

    db = get_db()
    db.execute('update members set name = ?, email = ?, level = ? where id = ?', [name,
email, level, member_id])
    db.commit()

    member_cur = db.execute('select id, name, email, level from members where id = ?',
[member_id])
    member = member_cur.fetchone()

    return jsonify({'member' : {'id' : member['id'], 'name' : member['name'], 'email' :
member['email'], 'level' : member['level']}})
```

Pour supprimer un membre :

```
@app.route('/member/<int:member_id>', methods=['DELETE'])
def delete_member(member_id):
    db = get_db()
    db.execute('delete from members where id = ?', [member_id])
    db.commit()

    return jsonify({'message' : 'The member has been deleted!'})
```