1 Framework Flask

1.1 Présentation du TP

Dans ce TP nous allons faire fonctionner un serveur Web que l'on interrogera à partir d'un navigateur Web sur une machine cliente. Dans la pratique, nous utiliserons le même ordinateur comme serveur et comme client (le serveur est alors accessible avec le nom de domaine `localhost`).

Le serveur sera développé grâce au micro-framework Python **Flask** qui permet très facilement d'activer un serveur Web et de générer des pages Web dynamiques en utilisant Python.
Le framework Flask nous oriente vers une structure spécifique pour développer un site Web. Le but de ce TP n'est pas d'acquérir une maîtrise du framework et de sa structure, mais simplement de pouvoir générer des pages et d'observer les requêtes entre le serveur et le client, notamment lors de l'utilisation de formulaire.

1.2 Mise en place de la structure

- Créer un dossier `site` qui contiendra l'ensemble de l'application Web.
- Créer un fichier `views.py` dans ce dossier : c'est le fichier Python qui démarre le serveur Web et appelle les fonctions, appelées **vues**, pour générer les pages Web à servir.
- Créer un dossier `static` et un dossier `templates` dans le dossier `site`.
- Le dossier `static` contiendra tous les fichiers « statiques » (càd les images, les feuilles de styles ou les scripts JavaScript).
- Le dossier `templates` contiendra tous les patrons de fichiers HTML à générer.
  Ces fichiers ressemblent beaucoup à des simples fichiers HTML mais en fait ce sont des fichiers pris en charge **Jinja**, moteur de template, qui les transforme en vrais fichiers HTML avant que le serveur Flask ne les envoie au client. On les nommera avec une extension `.html` bien que n'importe quelle extension soit valable.
- On pourra créer dans le dossier `static` des sous-dossiers `img`, `css` et `js` pour contenir les fichiers images, feuilles de style ou JavaScript (facultatif, mais ça permet de bien ranger ses affaires !)

Exemple d'arborescence de site Web :

```
site
|-- static
|   |-- css
|   |   |-- style.css
|   |-- img
|   |   |-- photo.png
|   |-- js
|       |-- script.js
|-- templates
|   |-- formulaire.html
|   |-- index.html
|   |-- traitement.html
|-- views.py
```

2 Premiers pas avec Flask

Les exemples suivants ont pour but de faire comprendre la structure générale des sites écrits avec Flask.

2.1 Exemple 1

Voici le fichier `views.py` minimal :

```
# import des modules nécessaires :
from flask import Flask, render_template, request

app = Flask(__name__) # crée une application Web

@app.route('/accueil/') # URL http://localhost:5000/accueil
def index(): # le nom de cette fonction (c'est une vue) est libre
    return render_template("index.html")
    # lien vers le fichier du dossier templates pour générer la page avec Jinja

app.run(debug=True) # démarre le serveur
# accessible dans un navigateur http://localhost:5000
```

La ligne `@app.route('/accueil/')` est un *décorateur* (*concept hors programme*) : cette ligne est indispensable et doit être collée à la ligne suivante. Elle indique la « route » (URL) donnant accès à la vue.

Et le fichier `index.html` (dans le dossier `templates`) pourrait être :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8"/>
    <title>Intro</title>
</head>
<body>
    <h1>Beau site</h1>
    <p>Tout est OK</p>
</body>
</html>
```

1. Recopier et enregistrer ces fichiers.
2. Dans une console Python, exécuter le fichier `views.py`. Cela démarre le serveur Web.
3. Dans un navigateur, entrer l'URL : `http://localhost:5000/accueil`
   Le nom de domaine du serveur est `localhost` car il est hébergé sur la même machine que le client ; et le serveur Flask écoute par défaut sur le port 5000.

2.2 Exemple 2 : passer des paramètres au template

On peut ajouter des paramètres (sous la forme `nom=valeur`) dans la fonction `render_template`.
On aura ainsi dans le fichier `views.py` :

```
@app.route('/accueil/')
def index():
    t = "Accueil" # code Python standard
    n = "Henri"
    return render_template("index.html", titre=t, nom=n)
```

Et le fichier `index.html` (dans le dossier `templates`) pourrait contenir :

```
[...]
<head>
    <title>{{titre}}</title>
</head>
<body>
    <p>Je me présente, je m'appelle {{nom}}.</p>
[...]
```

Jinja récupère les paramètres passés par la fonction `render_template` et peut les utiliser avec la notation entre doubles accolades.

2.3 Exemple 3 : lier un fichier dans un template

Pour créer des liens vers des fichiers du dossier `static`, il faut utiliser la fonction :

`url_for('static', filename='chemin/vers/fichier')`

Par exemple, pour lier une feuille de style présente dans le dossier `static/css`, la ligne de lien classique se transforme ainsi :

```
<link rel="stylesheet" type="text/css"
 href="{{url_for('static', filename='css/style.css')}}">
```

Bien noter l'utilisation des doubles accolades et la succession des " et '.

Pour créer des liens, on utilise aussi la fonction `url_for('nom_de_la_vue')` :

```
<a href="{{url_for('page1')}}">lien</a>
```

Le nom de la vue est le nom de la fonction définie dans le fichier `views.py` permettant de générer la page désirée.

Compléments :
`https://openclassrooms.com/fr/courses/1654786-creez-vos-applications-web-avec-flask`

3 Création de formulaires

3.1 Du côté template HTML

Il faut un template décrivant un formulaire qui demande par exemple d'entrer un nom et un prénom (deux éléments *input* de type *text*).
La valeur de l'attribut *action* est la route vers le fichier qui va effectuer le traitement du formulaire sur le serveur.
La méthode POST ou GET définit la façon d'envoyer les données.

```
<form action="{{url_for('traitement')}}" method="POST">
  <label for="id_nom">Nom :</label>
      <input type="text" name="nom" id="id_nom"><br/>
  <label for="id_prenom">Prénom :</label>
      <input type="text" name="prenom"  id="id_prenom"><br/>
  <input type="submit" value="Envoyer" />
</form>
```

3.2 Du côté serveur flask

Les données transmises au serveur sont récupérées dans des dictionnaires à travers l'objet `request.form` dans le cas de la méthode **POST**, ou `request.args` dans le cas de la méthode **GET**.

Dans les 2 cas, les clefs sont les attributs *name* du formulaire et les valeurs sont les données correspondantes.

La méthode est précisée en paramètre dans le décorateur de la vue comme montré dans les 2 exemples ci dessous :

— Méthode POST :

```
@app.route('/traitement/', methods=['POST'])
def traitement():
    reponse = request.form # request.form pour la méthode POST
    n = reponse['nom']
    p = reponse['prenom']
    return render_template("traitement.html", nom=n, prenom=p)
```

— Méthode GET :

```
@app.route('/traitement/', methods=['GET'])
def traitement():
    reponse = request.args # request.args pour la méthode GET
    n = reponse['nom']
    p = reponse['prenom']
    return render_template("traitement.html", nom=n, prenom=p)
```

Remarque : si on ne précise le paramètre de méthode, par défaut, c'est la méthode GET qui est utilisée.

4 Exercices

1. En s'appuyant sur toute la documentation précédente, mettre en place un serveur Web Flask qui crée un site proposant un formulaire permettant de rentrer son identité : nom et prénom. On pourra télécharger sur Moodle l'archive qui contient toute la structure de base de ce projet. Dans cette archive, seul le formulaire avec la méthode GET est implémenté.
   Après l'envoi du formulaire, une page doit s'afficher en reprenant les informations saisies et en les présentant de façon « agréable ».
2. Tester le site ainsi mis en place et bien observer l'URL qui s'affiche dans la barre d'adresse du navigateur Firefox lorsque le formulaire est envoyé.
   Peut-on voir les données transmises dans l'URL ?
3. Modifier à la main les données directement dans l'URL et vérifier que ces données sont bien prises en compte.
4. Activer les outils de développement du navigateur et observer les requêtes dans l'onglet Réseau. On s'intéressera en particulier à la méthode de la requête, et on pourra visualiser en détail son en-tête, ses paramètres et la réponse.
5. Développer le projet pour gérer un autre fichier contenant le même formulaire, mais cette fois envoyé avec la méthode POST.
6. L'URL du fichier de traitement laisse-t-elle apparaître les données transmises ?
7. Peut-on voir ces données avec les outils de développement ? En déduire si la méthode POST est sécurisée ? (rappeler le protocole à utiliser pour avoir des transmissions réellement sécurisées).

5 Compléments

1. Ajouter des champs dans le formulaire. Voir captures d'écrans (fichier complément)
2. Lier une feuille de style.
3. Lier un fichier Javascript qui possède une fonction appelée par le click sur le bouton "Vérifier mot de passe" qui affiche un message d'alerte (utilisant la fonction `alert` prédéfinie en JS) pour prévenir l'utilisateur de bien remplir un mot de passe « bidon » car tout le monde va le voir !
4. Pour récupérer les données d'une série de caches à cocher qui portent la même valeur de l'attribut *name* (ex : `spe`), il faut utiliser :

```
reponse = request.args # ou request.form
spe = reponse.getlist('spe')
```

5. La page (`https://jinja.palletsprojects.com/en/3.1.x/templates/`) donne la syntaxe du langage Jinja pour effectuer des traitements un peu plus complexes. C'est très similaire au langage Python, sauf que chaque instruction est encadrée d'une accolade et un symbole pourcent. *Exemple* :

```
{% if age > 18 %}
<p>Tu es majeur</p>
{% else %}
<p>Tu es mineur</p>
{% endif %}
```