L2 ISTN 2024-25
UE WEB

TP-Projet
Séquence serveur web dynamique
Mise en œuvre avec le système Flask

Objectif : *comprendre l'opposition statique-dynamique, comprendre la dualité métier-web, comprendre les routes et les templates.*

Le système Flask est un système de génération de serveur web dynamique. On dit en général *framework*, et dans le cas de Flask on parle même de *micro-framework* tellement il est réduit à l'essentiel. C'est cependant une erreur de perception car même si le noyau de Flask est très simple, il est accompagné de nombreux modules qui en font au contraire un système très riche. On s'intéressera principalement à la gestion des *routes* et à celles des *templates* via le module *Jinja*.

L'idée général est qu'un cycle de vie métier produit des documents qui ne sont pas des documents web mais qui doivent être présentés via des technologies web. Il faut donc les traduire. Le principe de Flask est de le faire au moment de répondre à des requêtes à l'aide des *templates* Jinja. Puisque les ressources web ne sont produites qu'au dernier moment, les routes des URL ne peuvent pas être des chemins d'accès vers ces ressources dans le système de fichier. Elles sont plutôt un langage d'expression de requêtes.

Le système Jinja s'appuie sur des opérateurs qui permettent de combiner des fragments de documents HTML afin de produire des pages entières. Il permet au moins trois usages :

- Personnaliser des pages HTML en y insérant des données simples ;
- Mettre en facteur des éléments répétitifs des pages HTML ;
- HTMLiser des documents structurés qui ne sont pas composés en HTML, ex. traduire une liste Python en une liste HTML.

Prise en main de Flask

Lire le début de la documentation de Flask (https://flask.palletsprojects.com/en/3.0.x/) : les parties qui s'intitulent « Installation », « Quickstart » et « Tutorial ».

Flask est une application Python, et on utilise donc les outils Python, et le savoir faire Python. En particulier, il est préférable d'utiliser un environnement virtuel qui permet d'encapsuler un projet et les packages Python dont il dépend.

Au total, on utilisera la séquence d'installation et lancement suivante. Dans ce qui suit, tout compte, l'orthographe, les espaces, les ponctutations, etc. Tout doit être respecté scrupuleusement.

- Se connecter et ouvrir une fenêtre Terminal. Vous devez vous trouver dans votre répertoire d'accueil. Exécuter la commande `pwd` (*print working directory*) pour le vérifier.
- Créer le répertoire projet

  ```
  % mkdir -p L2ISTN/WEB/webdynamique
  % cd L2ISTN/WEB/webdynamique
  ```

  Ce répertoire est le répertoire de travail de ce TP. Les commandes qui suivent sont exécutées dans ce répertoire.

  L'option `-p` (*path*) de la commande `mkdir` permet de créer tout un chemin d'un coup, tout en réutilisant le début du chemin si il existe déjà. Comme en principe vous avez déjà créé le chemin `L2ISTN/WEB/webstatique` il ne restera plus à `mkdir` qu'à créer `webdynaque` sous `WEB`. Et si vous ne l'avez pas déjà créé, ce qui est un tort, cela le créera.

  Éviter d'utiliser les espaces, caractères accentués et caractères spéciaux dans les noms des répertoires et fichiers. En effet, les interfaces graphiques s'en accommodent très bien car on n'y écrit jamais en toute lettre ces noms, on clique à la place. Mais ça conduit à des ennuis sans fin avec les interfaces en lignes de commande, comme les shells des fenêtres de type terminal, ainsi qu'avec les codages internet.

  Le `%` est ce qu'on appelle une invite (un prompt), c-à-d. l'indication qu'il faut saisir quelque chose. `%` et `$` sont des invites fréquentes. Des variantes ajoutent au *prompt* le rappel du chemin d'accès au répertoire courant. C'est une information très utile car ne pas faire l'opération désirée dans le bon répertoire est une source inépuisable de gros ennuis.
- **Créer l'environnement virtuel**

  ```
  % python3 -m venv .venv
  % source .venv/bin/activate.csh
  ```

  Cela permet de rassembler dans le répertoire courant un maximum de dépendances. Il est ainsi moins dépendant des autres projets. Le `.csh` indique d'utiliser la procédure d'activation de l'environnement virtuel qui est propre au shell `csh`, celui qui est installé dans les salles de TP. D'autres variantes existent pour d'autres environnements techniques ; lire la documentation.

  Vérifier dans le *prompt* que l'environnement est bien activé.

  Passer à la suite même si cette étape a échoué.
- Installer Flask

```
% python3 -m pip install "pelican[markdown]"
```

L'installation d'une application peut se faire essentiellement en trois endroits : (a) dans l'espace système, mais pour cela des droits d'administrateur sont nécessaires, (b) près de votre répertoire d'accueil, (c) près du répertoire de travail, par la méthode de l'environnement virtuel.

- **La partie verte ne doit être exécutée que si la partie rouge a échoué. Le logiciel Flask est alors installé près de votre répertoire d'accueil, en un endroit un peu caché. Le rapport d'installation dit où, `~/.local/bin`, et il précise même que la variable PATH ne connaît pas ce chemin d'accès. La variable `PATH` indique au système d'exploitation (ici Linux) où chercher les fichiers exécutables (les binaires, d'où `bin`). Faire `echo ${PATH}` pour se rendre qu'elle ne connaît pas `~/.local/bin`. Pour le faire connaître il faut modifier le contenu de la variable `PATH`.**

  ```
  % setenv PATH ~/.local/bin:${PATH}
  ```

  Attention : cette commande est très dépendante du *shell* utilisé. Les variantes

  ```
  % PATH=~/.local/bin:${PATH}
  ```

  et

  ```
  % export PATH=~/.local/bin:${PATH}
  ```

  La seconde solution (`PATH=...`) semble marcher sur toutes les machines de TP.

  Refaire `echo ${PATH}` pour se rendre compte du résultat de la modification.

- Créer un répertoire pour l'exploration de Flask

  ```
  % mkdir exploration
  ```

  et un autre pour le développement du projet « magasin de vélo »

  ```
  % mkdir bikeshop
  ```

  Aller dans le répertoire d'exploration

  ```
  % cd exploration
  ```

  Tout ce qui suit se fera depuis ce répertoire de travail. Vérifier que le chemin d'exécution de l'application Flask est toujours bien connu

  ```
  % whereis flask
  ```

- Copier dans le répertoire courant l'application `hello.py` depuis la page Moodle du module WEB, séquence « serveur web dynamique ».

  Pour cela vous pouvez télécharger le fichier `hello.py` puis le déplacer dans le répertoire courant

  ```
  % mv ~/Downloads/hello.py .
  ```

  Les détails de l'emplacement réel du répertoire de téléchargement peuvent varier d'une installation à l'autre. Vérifier que tout s'est bien passé

  ```
  % ls -l
  ```

- Lancer le serveur Flask

  ```
  % python -m flask --app hello.py run
  ```

  Remarquer la réponse `Running on http://127.0.0.1:5000`. C'est l'analogue du `Serving site at: http://127.0.0.1:8000` du serveur Pelican. C'est l'adresse (l'URL) du site. Il suffit alors de la coller dans la fenêtre d'un navigateur pour accéder au site. Remarquer aussi l'alerte en rouge

  WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

Le serveur créé par Flask n'est pas du tout sécurisé afin de faciliter la mise au point de l'application, y compris sa modification depuis le navigateur du client. Pour un déploiement effectif, il faudra suivre les instructions qui sont données dans le manuel de référence de Flask.

Essayer la commande

```
% python -m flask --app hello.py routes
```

Elle liste toutes les routes de l'application. Tester les toutes. Le cas échéant tester les liens proposés par le navigateur. Essayer des routes non définies par l'application.

Suivre ce qui se passe depuis la fenêtre de lancement du serveur, et depuis la fenêtre du navigateur (onglet Outil > Outils du navigateur > Outils de développement web).

- L'application hello.py n'est pas très intéressante. Essayons de faire mieux. Le TP-projet « Serveur web statique » a permis de créer de nombreuses pages HTML avec une navigation riche. Nous allons essayer d'utiliser Flask pour servir ces pages.

Créer un répertoire pour recevoir les fichiers HTML d'une application Pelican

```
% mkdir -p templates/output
```

Le répertoire `templates` est le répertoire standard en Flask pour déposer les fichiers à faire servir par l'application. Le sous-répertoire `output` sert à confiner les fichiers qui viennent de Pelican. D'autres fichiers pourraient être présents dans `templates`.

Copier le contenu du répertoire `output` d'une application Pelican

```
% cp -r path_to_webstatique/output/* templates/output
```

L'option `-r` (*recursive*) de la commande `cp` (*copy*) demande de copier les répertoires avec leurs sous-répertoires, leurs sous-sous-répertoires, etc.

Ajouter la route suivante à l'application `hello.py`

```
@app.route("/output")
def output():
    print("output")
    return render_template("output/index.html")
```

Exécuter la commande

```
% python -m flask --app hello.py routes
```

et tester cette route. Que se passe-t-il ? Trouver une explication, et éventuellement une solution.

- La route `/output` conduit au fichier `index.html`, mais rien qu'à lui. Et aucune route ne conduit aux autres fichiers du répertoire `output`. Il faut donc prévoir une route capable de lire tous les fichiers du répertoire `output` et de ses sous-répertoires : ex. `pages` et `category`.

  Ajouter la route suivante à l'application `hello.py`

```
@app.route('/<path:page>')
def show_page(page):
    print("pelican page")
    fnrel = os.path.join("output", page)
    print(f"{fnrel = }")
    fnabs = os.path.join(TEMPLATES, "output", page)
    print(f"{fnabs = }")
# checks whether page exists
    if os.path.exists(fnabs):
        print("render_template")
        return render_template(fnrel)
    else:
        print("ko")
        abort(404)
```