Objectifs de la feuille

- Introduction Flask et objectifs pédagogiques
- Installation environnement virtuel & Flask
- Première vue
- Organisation d'un projet
- Fichier de configuration

Introduction

Vous avez une idée d'application web mais vous ne savez pas comment la réaliser ? Vous cherchez une alternative simple à Django, Symfony ou Laravel ? Je vous propose de gagner du temps en découvrant et en utilisant le micro Framework Flask écrit en Python. Pourquoi concevoir un site avec Flask ? Car il vous permet de créer une application web en 7 lignes de code seulement !

Dans ce cours vous apprendrez à :

- créer une application avec Flask en développant vos premières fonctionnalités ;
- organiser un projet MVT (Modèle / Vue / Template) ;
- utiliser un ORM pour interagir avec une base de données ;
- tester votre application avec des tests fonctionnels ;
- à mettre en ligne votre application Flask.

Mise en place d'un environnement virtuel

Tout d'abord vous devez créer un répertoire pour ce TP :

```
$ mkdir -p  TutoFlask
$ cd TutoFlask
```

Une bonne pratique, pour le développement d'applications web en python, est d'isoler chacune dans son propre environnement virtuel. Dans celui-ci, on pourra installer les packages que l'on veut, sans avoir besoin d'être administrateur. Aussi les installations et mises à jour dans l'environnement virtuel d'une application n'affectera pas celui d'une autre. La création d'un environnement virtuel n'est pas obligatoire mais fortement recommandée si vous êtes amenés à travailler sur de nombreux projets :

```
$ virtualenv -p python3 venv
```

Ceci a créé un répertoire venv que vous pouvez observer :

```
$ ls -l venv
```

Il faut maintenant activer le venv dans notre shell :

```
$ source venv/bin/activate
```

Notez que l'invite de commande a été modifiée pour vous indiquer que votre environnement virtuel est activé (venv). Cette activation modifie certaines variables d'environnement pour faire en sorte que Python utilise notre environnement virtuel au lieu de l'installation globale.

Attention : *quand on ouvre un nouveau terminal, il faut bien entendu refaire cette activation car elle est locale au shell qu'on utilise.*

Installation de Flask

Maintenant que nous avons activé l'environnement virtuel, nous allons installer le micro-framework Flask avec la commande suivante :

```
(venv) $ pip install flask
```

Première vue qui vous salue

Ensuite, créez un fichier views.py et collez le code suivant :

```
from flask import Flask
app = Flask(__name__)

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

if __name__== "__main__" :
        app.run( )
```

Afin de vérifier que Flask fonctionne, exécutez le script suivant : $ python views.py
Python vous informe que le serveur est lancé ! Il attend les requêtes. Ouvrez un navigateur en cliquant sur le lien http://127.0.0.1:5000. Vous verrez ″Hello world !″ en haut à gauche de votre page. Bravo, tout fonctionne ! Vous pouvez aussi lancer votre application à l'aide de la commande flask :

```
$ FLASK_APP=views.py flask run
```

La commande flask offre par défaut 2 sous-commandes : $ flask

```
Usage: flask [OPTIONS] COMMAND [ARGS]...

  A general utility script for Flask applications.
  An application to load must be given with the '--app' option, 'FLASK_APP'   environment variable, or with a
'wsgi.py' or 'app.py' file in the current  directory.

Options:
 -e, --env-file FILE      Load environment variables from this file, taking precedence over those set by '.env'
                          and '.flaskenv'. Variables set directly in the environment take highest precedence.
                          python-dotenv must be installed.
 -A, --app IMPORT         The Flask application or factory function to load, in the form 'module:name'. Module
                          can be a dotted import or file path. Name is not required if it is 'app', 'application',
                          'create_app', or 'make_app', and can be 'name(args)' to pass arguments.
 --debug / --no-debug     Set debug mode.
 --version                Show the Flask version.
 --help                   Show this message and exit.

Commands:
        routes            Show the routes for the app.
        run               Run a development server.
        shell             Run a shell in the app context.
```

Celle que nous avons utilisée est la sous-commande run dont vous pouvez consulter la documentation :

```
flask run --help
```

Installation du package python-dotenv

Il est malcommode d'avoir à préciser des variables d'environnement sur la ligne de commande. Le package python-dotenv va nous permettre de positionner ces variables dans un fichier .flaskenv à la racine de notre projet :

```
$ pip install python-dotenv
```

Puis nous créons un fichier .flaskenv à la racine du projet TutoFlask:

```
FLASK_APP= views.py
FLASK_ENV= development
```

Nous pouvons à présent tester que ça marche comme avant : $ flask run
Flask a automatiquement chargé les variables d'environnement du fichier .flaskenv.
Entre autre, FLASK_ENV= development active le mode debug.

Organisation et restructuration de votre projet

Et maintenant quoi ?
C'est bien beau mais ce ne serait pas plus utile d'afficher le contenu d'une page index.html ? Nous y reviendrons. Avant de faire cela, nous devons organiser notre projet un peu différemment. Flask, contrairement à d'autres frameworks, n'impose pas de structure. Vous pouvez tout à fait développer une application entière sur un seul et même fichier. La liberté, la vraie ! Mais encore une fois, ce n'est pas parce que vous pouvez le faire que vous devez le faire. Suivre certaines conventions vous sera très utile par la suite. Les voici :

- les feuilles de style, scripts, images et autres éléments qui ne seront jamais générés dynamiquement doivent être dans le dossier static,
- les fichiers HTML doivent être dans le dossier templates,
- les tests doivent être dans le dossier tests,
- le fichier views.py contient les différentes routes de l'application (nous y reviendrons aussi).

Arrêtez le serveur et créez donc les dossiers suivants : static, templates et tests.
Normalement, votre dossier de travail devrait actuellement ressembler à ceci : $ ls

```
static  templates  tests  venv  views.py
```

C'est un peu le bazar, vous ne trouvez pas ? Je vous propose de regrouper les fichiers de notre application sous un seul et même dossier, monApp, et de le transformer en paquet. Nous pourrons ainsi, plus tard, créer d'autres applications qui cohabiteront ensemble. Déplacez tous les fichiers et dossiers (sauf, évidemment, celui de votre environnement virtuel et le fichier .flaskenv) dans monApp.

Puis créez un fichier app.py dans monApp. C'est ce dernier script que nous utiliserons pour lancer le projet. Ouvrez-le et ajoutez les lignes suivantes :

```
from flask import Flask
app=Flask(__name__)
```

Enfin créez un fichier __init__.py dans monApp. il permet d'indiquer que le dossier monApp n'est pas un simple dossier mais un paquet.

Un paquet est essentiellement un ensemble de modules python rassemblés dans un même dossier. Il est ensuite possible de rajouter facilement un sous-module d'un paquet dans l'espace de noms courant, afin de pouvoir utiliser les variables et les méthodes qu'il propose.

Par exemple, si le paquet P contient un module A proposant une fonction foo( ), alors on peut importer cette fonction avec la commande from P.A import foo. Cela permet aussi à un autre module B du même paquet d'avoir également une fonction foo()` sans que cela ne pose de problème (il suffit de préciser le module from P.B import foo). N'hésitez pas à jeter un coup d'œil à la documentation python pour plus de détails.

Puisque nous importons app, le fichier __init__.py doit contenir cet objet. Ouvrez-le et ajoutez les lignes suivantes :

```
from .app import app
import monApp.views
```

Aussi, le fichier views.py dans monApp devient :

```
from .app import app

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

if __name__ == "__main__":
    app.run()
```

Par ailleurs, n'oubliez pas de modifier le fichier .flaskenv à la racine du projet TutoFlask :

```
FLASK_APP= monApp
FLASK_ENV= development
```

Parfait ! Si vous lancez le script avec la commande flask run, vous constaterez que tout fonctionne toujours.

Création du fichier de configuration

Tout projet web qui grandit est amené à travailler avec des variables que nous utiliserons un peu partout dans le projet (l'emplacement de la base de données par exemple). Par convention, ces variables sont dans le fichier config.py à la racine de TutoFlask. Créons-le tout de suite ! Puis ouvrons-le pour ajouter les variables.

```
#>>>import random, string, os
#>>>"".join([random.choice(string.printable) for _ in os.urandom(24) ] )
SECRET_KEY = "2lzUl{$*D6#`8uXqlU."
```

Qu'est-ce que cette "SECRET_KEY" ? Est-elle obligatoire ?
La "Secret Key", clé secrète en français, permet de générer toutes les données chiffrées. Par exemple, elle permet de générer les cookies. Elle est donc obligatoire ! Il faut également garder cette clé de manière confidentielle. La clé sert à signer les cookies afin d'être sur qu'ils n'ont pas été altérés entre le client et le serveur. Vous verrez cela plus tard en crypto ! (https://stackoverflow.com/questions/22463939/demystify-flask-app-secret-key)

Pour les besoins du cours, vous y avez accès. Mais dans le cas d'une "vraie" application, la clé ne doit en aucun cas être accessible par d'autres personnes. Dans un environnement de production, il est déconseillé d'écrire la clé en dur (en particulier si le code est partagé via github par ex) mais plutôt de passer par une variable d'environnement. Concrètement ici, il s'agit simplement d'une chaîne de caractères aléatoire qui peut être générée avec le code indiqué en commentaires. Copiez le code et collez-le dans config.py

Flask vous permet d'importer toutes les variables en une fois dans votre projet en utilisant la méthode config.from_object(file_name).

Ouvrez app.py et modifiez-le de manière à ce qu'il ressemble à ceci :

```
from flask import Flask
app = Flask ( __name__ )

# Config options - Make sure you created a 'config.py' file.
app.config.from_object('config')
# To get one variable, tape app.config['MY_VARIABLE']
```

En résumé :

- Flask est disponible via le gestionnaire de paquets Python, PyPI
- Pour voir votre application, Flask vient avec un serveur de développement accessible à l'adresse locale http://127.0.0.1:5000/
- Il est recommandé de suivre une structure de projet en séparant les vues, les templates, les tests, les fichiers statiques et les fichiers de configuration.
- SECRET_KEY est une variable de configuration requise pour gérer les éléments nécessitant du chiffrement (session, cookies)

C'est à vous de jouer !

- créer une variable ABOUT = "Bienvenue sur la page à propos de Flask !" dans le fichier config.py
- afficher ce message dans une nouvelle vue about accessible via l'url "about/"
- faire de même avec une page de contact. Je vous laisse le choix du message et de l'url.