Environement de travail

Nous allons installer un environement virtuel de travail dans lequel nous allons installer flask avec pip.

Attention dans un environement virtualisé (dans le dossier, voir plus bas) si vous utilisez Git vous devez créer un fichier .gitignore afin de ne pas versionner le dossier de virtualisation :

Fichier .gitignore

```
/env
```

L'utilisation d'un environement virtualisé est courant, il permet de ne pas installer des dépendances sur la machine et facilite le travail entre développpeur.

```
# vérifier que virtualenv est bien installé.
pip install virtualenv

# puis dans le dossier de travail

virtualenv -p python3 env

# si vous travaillez avec Windows exécuter cette commande
# dans le PowerShell
virtualenv -p $env:python3 env
```

Cette commande crée un dossier env dans lequel on utilise Python3. Pour activer l'environement il faudra taper la commande suivante :

```
source env/bin/activate

# dans le PowerShell sous Windows
./env/scripts/activate.ps1
```

En utilisant la commande which ou where pour Windows vous pouvez constater que le chemin de l'exécutable Python n'est plus le même, il correspond à l'environement que vous venez de créer.

```
which python
# ou sous Windows

where python
```

Attention, pour chaque session ou console séparé vous devez relancer l'environement pour l'activer.

Pour désactiver et supprimer l'environement tapez la commande suivante :

```
deactivate

rm -rf env

# pour désactiver l'environement il suffit simplement d'exécuter la commande seule :

deactivate
```

Création d'un fichier de dépendance

Afin de préciser les dépendances utilisées dans votre projet vous devez créer le fichier suivant, cela permettra de partager vos projets et de les migrer facilement sur un autre poste de travail.

```
# commande permettant d'écrire les dépendances dans ce fichier avec leurs versions.
pip freeze > requirements.txt

# Si vous êtes sous Windows dans le PowerShell

pip list > requirements.txt
```

Installation de Flask

Une fois l'environement démarré installer Flask dans ce dernier dans un dossier "flaskapp"

```
source env/bin/activate
# dans le PowerShell sous Windows
# ./env/scripts/activate.ps1

(env) project_flask pip install flask
```

Quelques conseils vous pouvez vous organiser comme vous le souhaiter, mais nous vous conseillons l'organisation suivante :

- Un dossier "static" pour les assets.
- Un dossier "templates" pour les vues.
- Un dossier "tests" pour les tests.

Créez également le fichier app.py il contiendra les routes de l'application à la racine du dossier. Lancez maintenant le serveur :

```
python run.py
```

Coupez le serveur, nous allons créer un dossier particulier pour packager l'application. Créez le dossier flaskapp, mettez tous les fichiers et dossier dans ce dernier dossier, sauf le dossier env et .gitignore.

Relancez le serveur pour voir si tout marche correctement.

SQLite

Nous allons connecter la base de données SQLite à notre projet, Python possède nativement un "connecteur" à SQLite, il suffit d'importer dans le projet l'ORM Sqlalchemy.

```
pip install flask_sqlalchemy
```

Ajoutez les lignes suivantes dans le fichier run.py :

```
from flask import Flask

app = Flask(__name__)

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

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

Création d'un modèle User

Un ORM (object relation model) permet de créer des modèles liés aux tables SQL de la base de données, c'est une couche d'abstraction utilisant des pattern de conception. Elle permet de s'abstraire du SQL pour les requêtes classiques et implémente la sécurité et les bonnes pratiques pour la connexion et récupération des données dans les scripts Python.

```
from flask_sqlalchemy import SQLAlchemy

import logging as lg

# Create database connection object
db = SQLAlchemy(app)

class User(db.Model):
    # définition de la structure de la table
    id = db.Column(db.Integer, primary_key=True)
```

```
    name = db.Column(db.String(100), nullable=False)
    grade = db.Column(db.Integer(), nullable=False)
    notes = db.Column(db.Text(), nullable=True)

    def __init__(self, name, grade, notes):
        self.name  = name
        self.grade = grade
        self.notes = notes

users = [
    {'name' : 'Hero', 'grade' : 1, 'notes' : [15,17,11,9,10] },
    {'name' : 'Dunn', 'grade' : 2, 'notes' : [10,11,11,19,7] },
    {'name' : 'Sue', 'grade' : 2, 'notes' : [13,17,9,9,7] },
    {'name' : 'Thor', 'grade' : 1, 'notes' : [15,17,20,19,16] },
    {'name' : 'Hicks', 'grade' : 1, 'notes' : [13,10,11,6,12] },
]

def init_db():
    db.drop_all()
    db.create_all()
    for user in users:
        notes = user['notes']
        db.session.add(User(user['name'], user['grade'], "{notes}"))
    db.session.commit()
    lg.warning('Users created')
```

Pour lancer ce code tapez maintenant en ligne de commande :

```
export FLASK_APP=run.py flask
# Pour connaitre l'ensemble des méthodes que vous pouvez utiliser
flask

# Puis pour initialiser les données par rapport au nom de votre décorateur
flask init
```

Vous pouvez interagir avec les objets du projet dans la console :

```
# Permet de lister les commandes disponibles
flask

# La console Flask
flask shell
>>> from run import db, User

# vérifier que les données sont bien créées :
>>> User.query.all()
```

Mise en place des vues

Dans le dossier templates créer les deux fichiers suivants :

index.html et base.html

Le fichier base.html permet de créer un template de “base” qu’hérieront les autres vues composites :

```
<!doctype html>
<title>{% block title %}{% endblock %} - Flaskr</title>
<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
<nav>
  <h1>Flaskr</h1>
  <ul>
  </ul>
</nav>
<section class="content">
  <header>
    {% block header %}{% endblock %}
  </header>

  {% block content %}{% endblock %}
</section>
```

La vue index.html héritera de la vue base.html

```
{% extends 'base.html' %}

{% block header %}
<h1>Hello Flask</h1>
{% endblock %}

{% block content %}
<ul class=users>

</ul>
{% endblock %}
```

Passer les données à la vue

Pour terminer nous allons passer les données à la vue :

```
from flask import Flask, render_template

@app.route("/")
```

```
def hello():
    users = User.query.all()
    return render_template('index.html', users=users)
```

Essayez d'afficher maintenant les donnes dans la vue, pensez à lancer le serveur à l'aide de la commande suivante :

```
flask run
```