TP Flask

Hello World avec Flask

Avant de se lancer dans le gestionnaire de contact (partie suivante), il peut être intéressant de se familiariser avec Flask dans son ensemble. Pour cela, reprendre les slides sur le "Hello world" **donnés dans le cours** :

0 : Créer un dossier de travail `hello/` . Dedans, initialiser un virtualenv `venv` dans lequel on installera `Flask` .

1 : Créer un fichier `hello.py` qui contient l'exemple du cours:

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

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

2 : Dans un terminal, lancer le serveur de test avec

```
$ export FLASK_APP=hello.py
$ flask run
```

Le serveur devrait se lancer et écouter sur le port 5000

3 : Valider que l'application fonctionne en visitant `http://127.0.0.1:5000/` depuis un navigateur web (dans la machine virtuelle).

Construction d'un gestionnaire de contacts avec Flask

On propose de réaliser une petite application web avec Flask, qui permet de gérer des contacts stockés dans une base SQL. Pour pouvoir se concentrer sur les aspects les plus importants, l'architecture du projet est déjà prémachée (récupérer l'archive auprès du formateur) et la partie vue (template) est déjà écrite. Nous allons surtout nous concentrer sur l'écriture du modèle et des controlleurs.

0 : Récupérer l'archive contenant l'architecture du projet auprès du formateur. La dézipper dans un dossier que l'on apellera par exemple `contactManager/` . Dedans, initialiser un virtualenv `venv` et installer à l'aide de `pip` les modules `Flask` et `Flask-SQLAlchemy` .

Edit par Aleks en 2023 : les versions de Flask ayant évolué depuis l'écriture de ce TP, il faut forcer les versions installées pour être compatitble avec le code:

```
pip3 install Flask==1.1.4 Flask-SQLAlchemy==2.5.1 MarkupSafe==2.0.1 SQLAlchemy==1.4.
```

1 : Dans un premier temps, concentrons nous sur définir le modèle que nous allons utiliser pour décrire et stocker nos contacts. Inspecter le fichier `models.py` et implémenter les choses demandées dans les commentaires.

2 : Pour tester que notre modèle fonctionne, nous souhaiterions créer les tables SQL correspondantes. Le fichier `common.py` défini déjà que celles-ci seront stockées dans une base de type SQLite, et en particulier le fichier `db.sqlite` (qui sera automatiquement créé par Flask le moment venu). Le fichier `initialize.py` est un petit script destiné à (ré)initialiser les tables SQL et à remplir ces tables avec des données de test. Inspecter le fichier `initialize.py` et implémenter les instructions demandées dans les commentaires, et tester d'executer ce script. (Eventuellement, faire un `print(Contact.query.all())` pour valider que la base contient bien quelque chose...)

3 : Inspecter le premier controlleur défini dans `app.py` , qui gère l'URL `/` . Comment faire pour récupérer tous les objets `Contact` existants et ainsi remplir la variable `contacts` qui est ensuite utilisée par la vue ? Une fois cette question résolue, vous pouvez tester l'application en lançant dans un terminal :

```
export FLASK_APP=app.py
export FLASK_ENV=development
flask run
```

puis en essayant d'accéder à `http://127.0.0.1:5000/` .

4 : Gérons maintenant le cas où l'utilisateur ajoute un nouveau contact depuis le formulaire de l'interface. Le clic sur le bouton 'Ajouter' déclenche une requête vers `/add` géré par le deuxième controlleur, dont certaines parties restent à implémenter. Les données du formulaire sont disponibles dans le dictionnaire `request.form` . À partir de celle-ci, il faut créer un nouveau `Contact` et l'ajouter à la BDD. Validez que votre programme fonctionne en tentant de remplir le formulaire. (N.B. : la question du parsing de la date d'anniversaire n'étant pas triviale, dans un premier temps on peut l'ignorer et forcer sa valeur à `None`)