Projet Twitter

Technologie du Web

30/04/2013
Master 2 IDL parcours GLIA
Benjamin BARATE

Table des matières

Introduction : 2

Installation de Flask et de l'application Minitwit : 2

Présentation de l'application réalisée : 2

- Minitwit : 2
- Ajout de la fonctionnalité 'Classement des twittos' : 3
- Ajout de la gestion des hashtags : 7

Conclusion : 9

Introduction :

Dans le cadre de l'UE « Technologie du Web », il nous a été demandé de réaliser un « twitter-like » en mettant en place le design-pattern MVC vu en cours magistraux.

Afin de m'aider dans cette tâche j'ai décidé d'utiliser le micro-framework « Flask ». « Flask » est basé sur le langage « Python ». Avec « Flask », on peut :

- Avoir à disposition un serveur http
- Implémenter un contrôleur en quelques lignes de code
- Définir des templates html
- Insérer des données dans les templates html
- Retourner les pages générés dynamiquement

Enfin, l'intégralité des données sont stockées dans une base de données « Sqlite ».

Installation de Flask et de l'application Minitwit :

L'installation de « Flask » est plutôt simple lorsque l'on suit l'article présent à l'adresse suivante : http://flask.pocoo.org/docs/installation/#installation.

Personnellement, je l'ai installé sans accroc sur un système d'exploitation Windows 7. Attention, Flask n'est pas compatible avec les versions 3.X de Python. Je vous conseille donc d'installer la version 2.7 de l'interpréteur.

Une fois que vous avez mis en place « Flask » et votre environnement virtuel, vous pouvez décompressez le contenu de l'archive minitwit.zip dans votre dossier « venv ». Une fois cela fait, vous pouvez lancer l'application « Minitwit » avec les commandes suivantes :

```
C:\Users\Master\Documents\tmp>cd venv
C:\Users\Master\Documents\tmp\venv>Scripts\activate
(venv) C:\Users\Master\Documents\tmp\venv>cd minitwit
(venv) C:\Users\Master\Documents\tmp\venv\minitwit>python minitwit.py
 * Running on http://127.0.0.1:5000/
 * Restarting with reloader
```

A noter, que ce document est également joint avec une autre archive nommée venv.zip. Cette archive contient l'ensemble de mon environnement virtuel et l'application « minitwit » se trouve dans le répertoire « src ».

Présentation de l'application réalisée :

Minitwit :

Pour réaliser mon projet je suis parti d'un exemple de code fourni sur le site de « Flask ». Cette exemple de code s'appelle « Minitwit » et implémente une version rudimentaire de Twitter.

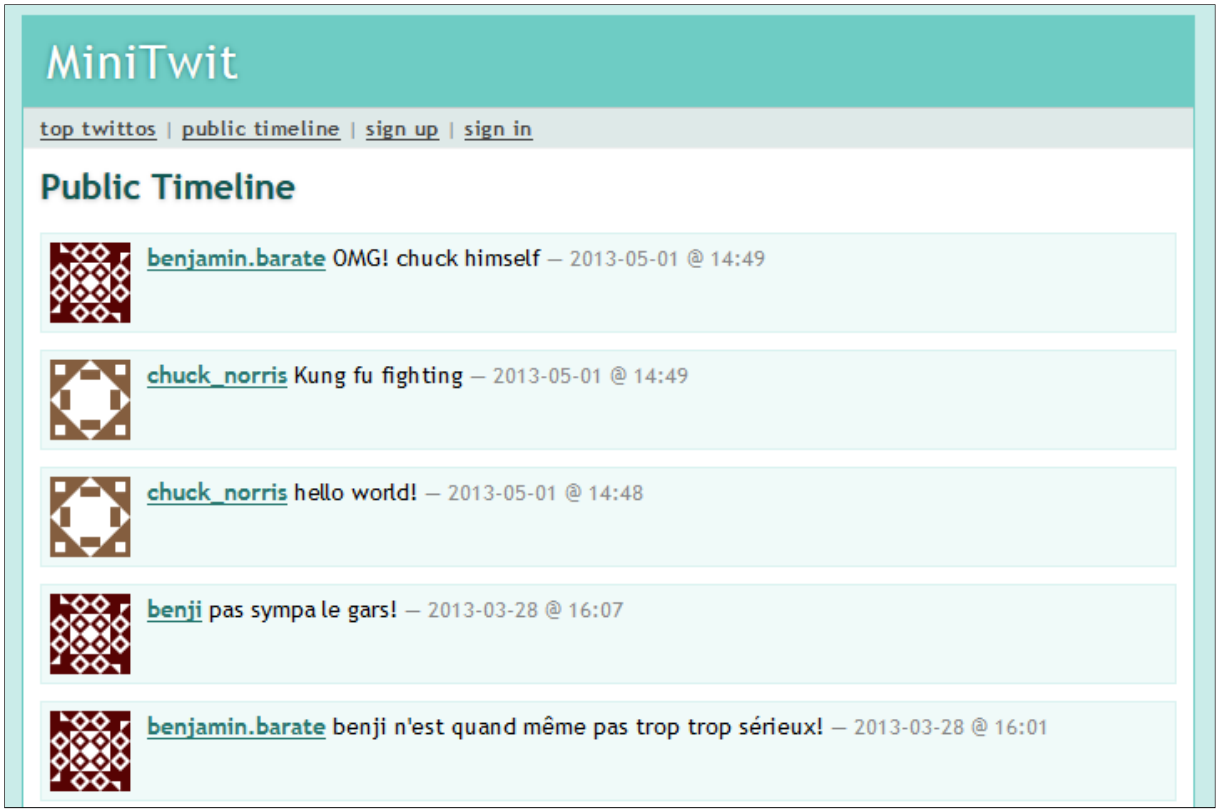

Figure 1: Timeline de minitwit

Cette application permet de :

- Créer de nouveaux comptes utilisateurs
- S'authentifier
- Rédiger des tweets
- Follower d'autres utilisateurs

Afin d'approfondir mes connaissances sur « Flask », j'ai donc décidé d'ajouter les deux fonctionnalités suivantes à « Minitwit » :

- Générer un classement des 10 utilisateurs twittant le plus
- Gérer le concept des hashtags

Ajout de la fonctionnalité 'Classement des twittos' :

Tout d'abord, j'ai élaboré la requête SQL suivante me permettant de récupérer le top 10 des twittos en nombre de messages. Je me suis servi du plugin Firefox « Sqlite Manager » pour rédiger et tester mes requêtes SQL.

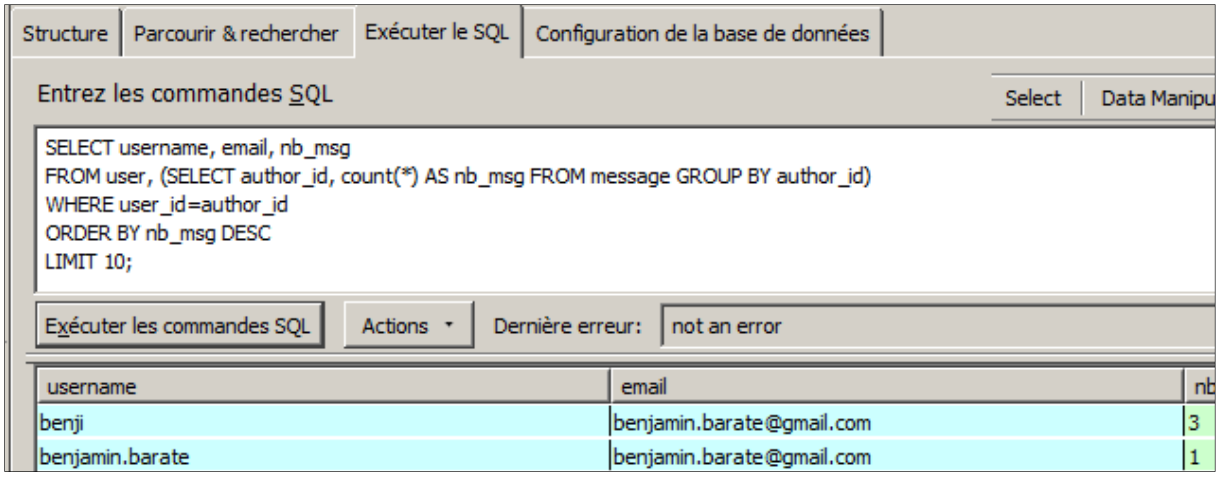

Figure 2: Requête SQL générant le TOP 10

Ensuite j'ai ajouté une route dans le fichier minitwitt.py pour gérer cette nouvelle fonctionnalité. Je dis bien route car le contrôleur de l'application « Minitwit » réalise très peu de contrôle et je le conçois donc plus comme un dispatcher. Pour l'instant ma nouvelle route renvoie sur le template « login.html ». Cela me permet de tester si j'ai instancié ma route correctement.

```
@app.route('/top')
def top_twittos():
    """Displays the top 10 twittos"""
    return render_template('login.html', error=None)
```

Figure 3: Nouvelle route

Il est intéressant de remarquer que nous ne sommes pas obligé de redémarrer notre serveur pour que les modifications du code source soient prises en compte. Voici les logs affichés dans la console :

```
 * Detected change in 'minitwit.py', reloading
 * Restarting with reloader
```

Figure 4: Console

Afin que la nouvelle route soit accessible via le menu de navigation, j'ai inséré le code suivant dans le fichier « layout.html » :

```
    <a href="{{ url_for('top_twittos') }}">top twittos</a> |
```

Figure 5 : Modification du menu de navigation

Ensuite, j'ai testé la nouvelle route en atteignant l'adresse http://localhost:5000 et en cliquant sur « top twittos ». J'étais dirigé sur la page d'authentification, tout était donc bien mis en place.

Figure 6: Page d'authentification

J'ai donc pu créer un nouveau template html qui affiche le top 10 des twittos. Je l'ai appelé top.html et voici son contenu :

```
{% extends "layout.html" %}
{% block title %}Top twittos{% endblock %}
{% block body %}
    <h2>Top twittos</h2>
    <ul class="twittos">
        {% for user in users %}
        <li><img src="{{ user.email|gravatar(size=48) }}"><p>
        <strong>
            <a href="{{ url_for('user_timeline', username=user.username)}}">
                {{ user.username }}
            </a>
        </strong>
        wrote {{ user.nb_msg }} message(s).
  {% else %}
    <li><em>There's no users so far.</em>
  {% endfor %}
  </ul>
{% endblock %}
```

Figure 7: top.html

J'ai également instancié la classe CSS « twittos » dans le fichier « static/style.css » afin que la liste à puce ressemble à celle de la timeline. A noter que le répertoire « static » contient les fichiers statiques fourni par le serveur Web interne de « Flask ».

```css
div.page ul.twittos {
    list-style: none;
    margin: 0;
    padding: 0;
}

div.page ul.twittos li {
    margin: 10px 0;
    padding: 5px;
    background: #F0FAF9;
    border: 1px solid #DBF3F1;
    -moz-border-radius: 5px;
    -webkit-border-radius: 5px;
    min-height: 48px;
}

div.page ul.twittos p {
    margin: 0;
}

div.page ul.twittos li img {
    float: left;
    padding: 0 10px 0 0;
}
```

Figure 8: Modification de style.css

Enfin j'ai modifié la route pour qu'elle exécute la requête SQL récupérant le top 10 des twittos, injecte son résultat dans le template « top.html » et renvoie ce dernier via HTTP :

```python
@app.route('/top')
def top_twittos():
    """Displays the top 10 twittos"""
    return render_template('top.html', users=query_db('''
        SELECT username, email, nb_msg
        FROM user, (
            SELECT author_id, count(*) AS nb_msg
            FROM message
            GROUP BY author_id
        )
        WHERE user_id=author_id
        ORDER BY nb_msg DESC
        LIMIT 10'''))
```

Figure 9: définition de la fonction top_twittos()

Enfin le résultat (cf. figure 10) affiche le top 3 des utilisateurs. En effet, la page n'en affiche pas 10 car l'application minitwitt ne contient que trois utilisateurs :

- benji ayant pour mot de passe 'mini42'
- benjamin.barate ayant pour mot de passe 'mini42'
- chuck_norris ayant pour mot de passe 'chuck'

Figure 10: Classement des twittos gazouillant le plus

Ajout de la gestion des hashtags :

L'application « Minitwit » de base ne gère pas les hashtags. J'ai donc décidé d'intégrer leur gestion en :

- créant une table « hashtag » dans la base de données sqlite (cf. figure 11)
- modifiant la fonction « add_message() » (cf. figure 12) afin qu'elle insère en base de données les hashtags contenus dans chaque tweet
- créant le template « hashtag.html » (cf. figure 13) qui liste l'ensemble des hastags
- créant la fonction « hashtag_listing() » (cf. figure 14) renvoyant une liste des hashtags contenus en base de données

A noter que cette nouvelle fonctionnalité est plus que rudimentaire car elle n'empêche pas le doublon des hashtags en base de données et ne garde aucun lien entre les hashtags et les tweets les contenant.

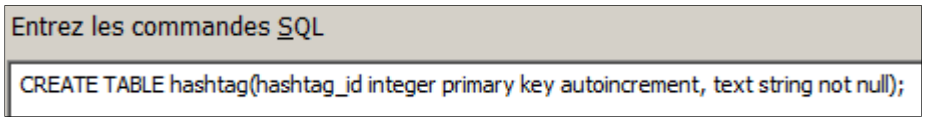

Figure 11: Création de la table hashtag

```
if request.form['text']:
    for tag in request.form['text'].split():
        if tag.startswith("#"):
            tag = tag.strip("#")
            g.db.execute('''insert into hashtag(text)
                values (?)''', [tag])
            g.db.commit()
```

Figure 12: Modification de add_message()

```
{% extends "layout.html" %}
{% block title %}Hashtags listing{% endblock %}
{% block body %}
    <h2>Hashtags listing</h2>
    <ul class="hashtag">
        {% for hashtag in hashtags %}
        <li>{{ hashtag.text }}
  {% else %}
    <li><em>There's no hashtag so far.</em>
  {% endfor %}
  </ul>
{% endblock %}
```

Figure 13: Template « hashtag.html »

```
@app.route('/hashtag')
def hashtag_listing():
        """Displays all the hashtags"""
        return render_template('hashtag.html',
                hashtags=query_db('''
                select distinct * from hashtag'''))
```

Figure 14: code de la fonction hashtag_listing()

Afin d'alléger ce rapport, j'ai volontairement survolé la mise en place de la gestion des hashtags. En effet, celle-ci respecte la même procédure que la mise en place du top 10 des twittos. Le résultat est le suivant :

Figure 15: Hashtag listing

Comme on peut le voir j'en ai profité également pour modifier le style CSS pour que l'ensemble soit plus agréable visuellement.

Conclusion :

En conclusion, grâce à « Flask », en à peine quelques heures, j'ai ajouté deux fonctionnalités à une application MVC déjà existante. Sachant que je n'avais jamais utilisé le langage Python avant ce projet, ce délai très court montre que « Flask » est framework puissant et bien documenté.

Certes, il ne propose sûrement pas l'ensemble des fonctionnalités proposées par un framework comme Spring mais il a l'avantage d'être mis en place rapidement et permet la production de code en très peu de temps.