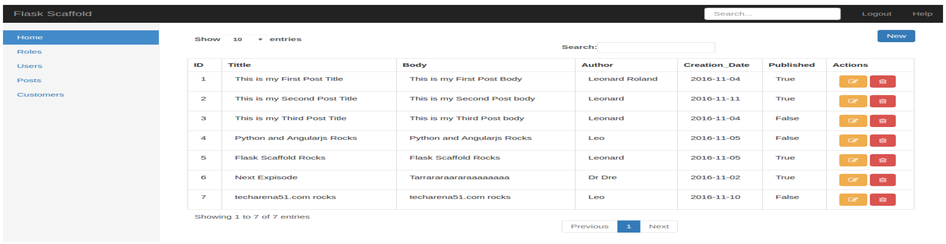

Guión de Prácticas 4: Bases de Datos NoSQL, CRUD

S. Alonso (zerjioi@ugr.es) y J.M. Guirao (jmguirao@ugr.es)

Entrega: 29 - noviembre

Resumen

En esta práctica veremos como utilizar una base de datos NoSQL, que pueden ser de gran utilidad en una aplicación web.

Bases de Datos No SQL: mongoDB

MongoDB es una base de datos NoSQL potente que nos permitirá almacenar cualquier información que nuestra aplicación web necesite. Desde Python podemos acceder a la base de datos MongoDB usando el cliente Pymongo.

En esta práctica vamos a usar alguno datos de prueba para mongoDB (ver siguiente apartado). La idea es crear varias páginas web que nos permitan consultar información sobre algunos de estos datos, hacer alguna búsqueda sencilla, insertar nuevos elementos, modificarlos, etc.

Instalación del servicio mongoDB y de bases de datos de prueba

Usaremos una imagen de mongo para montar nuestro servicio de base de datos, de manera que el archivo **docker-compose.yml** queda como:

```
# docker-compose.yml

version: '3.7'

services:
      flask:
         build: .
         depends_on:
            - mongo
         ports:
            - 5000:5000
         volumes:
            - ./app:/app

      mongo:
         image: mongo:latest
         ports:
            - 27017:27017
         volumes:
            - ./dump:/dump              # los datos de prueba
            - ./datos_db:/data/db       # almacenamiento en el host
```

Crearemos los directorios **dump**, para los datos de prueba, y **datos_db**, para el almacenamiento en el host de los datos de la DB en el host, ya que si no la hecmos así, los perderiamos en cada build

Además tenemos que añadir un nuevo requisito(pymongo) a requirements.txt:

```
# requirements.txt
Flask
pymongo
```

Nos aseguramos que la imagen de Flask se reconstruye con la nueva librería:

```
> docker-compose build
```

En este punto ya podríamos manejar la base de datos desde nuestra aplicación Flask usando pymongo. Sin embargo no tenemos ningún dato sobre el que trabajar. Vamos a alimentar la base de datos con unas colecciones descargadas de Internet.

En la carpeta **dump**, creamos una subcarpeta **SampleCollections** para una BD con este nombre, y dentro de ella las collecciones que elijamos de SampleCollections. Para restaurar la BD ponenmos a funcionar los servicios:

```
> docker-compose up
```

y en otra terminal, abrimos una sesión de bash en el contenedor de mongo:

```
#> docker-compose exec mongo /bin/bash
```

y en la terminal del contenedor:

```
#> mongorestore --drop dump
```

que restaurará la base de datos 'SampleCollections' con las collecciones que tenga.

Aqui podemos comprobar que los datos están usando el shell de mongo. En http://www.diegocalvo.es/tutorial-de-mongodb-con-ejemplos/ tenemos una chuletas para usarlo.

Si todo ha ido bien ya podríamos hacer que nuestra aplicación web muestre información de nuestra base de datos. Por ejemplo: el título de un capítulo de la serie Friends:

```
#./app/app.py
from pymongo import MongoClient

client = MongoClient("mongo", 27017) # Conectar al servicio (docker) "mongo" en su puerto estandar
db = client.SampleCollections        # Elegimos la base de datos de ejemplo

...

@app.route('/mongo')
def mongo():
    # Encontramos los documentos de la coleccion "samples_friends"
    episodios = db.samples_friends.find() # devuelve un cursor(*), no una lista ni un iterador

    lista_episodios = []
    for episodio in episodios:
        app.logger.debug(episodio) # salida consola
        lista_episodios.append(episodio)

    # a los templates de Jinja hay que pasarle una lista, no el cursor
    return render_template('lista.html', episodios=lista_episodios)
```

(*) find()

Hacer una búsqueda simple con un formualrio para collección elegida. Pymongo Tutorial sería la documentación a consultar para las búsquedas. También puede ser util la búsqueda con expresiones regulares

Para nota

Incluir todo el CRUD con la modificación parcial, borrado, y adicción de nuevos documentos