Moot

Une application inspirée de twitter

Share and have Fun !

Auteur :
Olivia Bruce I4

Sommaire

I.Introduction....3
II.Projet 3
II.1 Les technos utilisées:....3
a.Redis....3
b.Scala....3
c.Play Framework....4
d.AngularJS....4
II.2 La structure du projet :....4
II.3La structure redis :....5
III.Présentation du contexte et du projet global :....6
III.1Les outils utilisés :....6
IV.Conclusion :....7
V.Linkographie :....7

I. Introduction

Twitter est un réseau social qui permet à des millions de personnes de connecter en même temps. Il nécessite donc de grandes performances. Nous avons vu en cours plusieurs outils permettant le stockage de données de manière efficace.

Le but de ce projet est de proposé une application similaire à twitter qui utilisera la base de données Redis.

II. Projet

Lors de la présentation du projet plusieurs technologies nous étaient possible. Il y avait les technologies classiques java, C# qui sont bien sûre des mastodontes de l'informatique.
Cependant mon choix, j'ai souhaité le porter sur des technologies plus jeune. J'ai donc décidé de réaliser le projet en scala bien que je n'y connaisse rien. C'était donc l'occasion de se lancer et heureusement qu'on a eu du temps parce ce n'est pas un langage comme les autres. Son approche tantôt fonctionnel tantôt POO est parfois un peu déroutant.

II.1 Les technos utilisées:

a. Redis

Redis (de l'anglais REmote DIctionary Server) est un système de gestion de base de données clef-valeur scalable, très hautes performances
Redis conserve l'intégralité des données en RAM. Cela permet d'obtenir d'excellentes performances en évitant les accès disques, particulièrement coûteux.
Pour la communication entre Scala et Redis, le choix s'est porté sur scredis, qui est une librairie non bloquante et bien documentée (voir lien en annexe).

b. Scala

Scala est un langage de programmation multi-paradigme, pour exprimer les modèles de programmation courants dans une forme concise et élégante. Son nom vient de l'anglais *Scalable language* qui signifie à peu près « langage adaptable » ou « langage qui peut être mis à l'échelle ». Il peut en effet être vu comme un métalangage. -Wikipedia

Il utilise des éléments de la programmation orientée objet et de la programmation fonctionnelle. Cela le rend très performant mais aussi complexe à comprendre au premier abord.

c. Play Framework

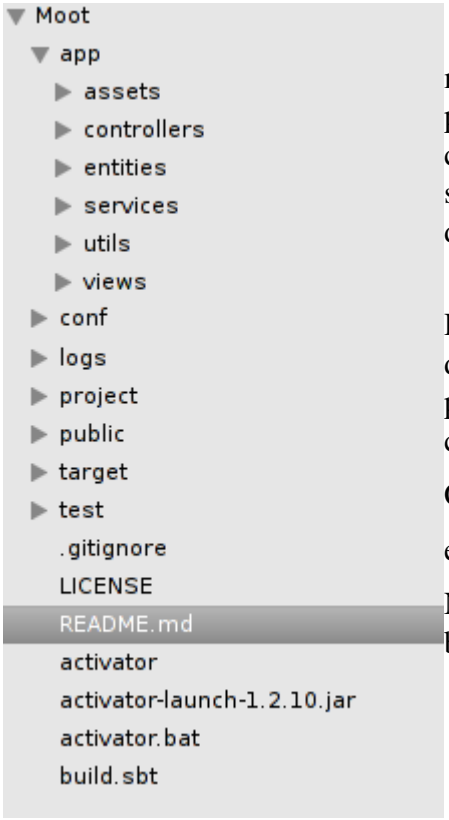
Play Framework est un framework web open source qui permet d'écrire rapidement des applications web pour le Java et le Scala. Il se base sur la modèle MVC et est RESTFull.

d. AngularJS

Angularjs est un framework libre et open-source développé par google qui permet d'augmenter du code HTML pour rendre des vues dynamiques très facilement. Il a pour but de simplifier l'utilisation du javascript et d'y ajouter des fonctionnalités.

II.2 La structure du projet :

Les technologies décidées, il ne restaient plus qu'a se lancer. J'ai donc décidé de me baser sur la documentation de play et de scala pour organiser mon code selon les bonnes pratiques.



Le code est donc organisé au sein d'une app qui contient les modèles, les vues, les controllers et les services. Elle correspond plus ou moins à la logique coté Back End. Cependant la construction de celui pourrait être un peu plus modulaire et s'organiser avec le client d'un côté (angular, html) et l'application de l'autre.

C'est au cœur des services que se trouve les appels à la base Redis. A l'aide de la librairie scredis qui est une librairie très bien documenté sur l'usage que l'on peut en faire, c'et d'ailleurs en partie pour cela que je l'ai choisie. Mais aussi parce qu'elle permet de créer une apllication play de manière non bloquante.

C'est à dire qu'elle permet de ne pas bloquer l'application si une exécution est longue ou bloquée.

Notamment avec l'utilisation des futures (qui m'ont donné des boutons).

II.3 La structure du schéma redis :

Pour les utilisateurs un HMSET il contient le nom d'utilisateur, le mot de passe et le token d'authentification.
Key : username:$username:uid – User unique identifier

Field	Value1
Username	String
Password	String
token	String

La clé de chaque utilisateur (uid) est unique, elle permet ainsi d'éviter l'auto-increment.

HSET
Key : user

Field	Value1
Username	UID

Pour éviter d'avoir à parcourir tous les utilisateurs pour retrouver l'uid à partir d'un username .
On fait appel à un HSET qui pour un nom d'utilisateur renvoie son identifiant

Followers/Followin → SortedSet
Key : user:$uid:followers ou user:$uid:following

Field	Score
UID	Time : long

On stock les followers/ followin pour un utilisateur, ranger par date de follow.

Post HMSET car les postes contiennent plusieurs éléments
Key : post:$postId

Field	Value1
PostId	String
Message	String
author	String

Posts → Liste de poste : globalfeed, usertimeline, hometimeline
Key : posts:UID ou globaltimeline

IndexN	…	1
PostID		PostID

Lorsque un poste est commité il faut le distribué à tous les abonnés.

Auth → hash d'authenficateur
Key : token:$token:uid

Field	Value1
authString	UID

Pour les hashtag → Liste d'identifiant de post
Key : hashtag

II.4 Implémentation :

L'application n'est pas complète, les interfaces sont faites, et les services Redis aussi cependant la communication entre les deux n'est pas encore complète.

Le fait de vouloir faire une application asynchrone n'est pas tache facile. ScRedis que j'utilise pour me connecter et discuter avec la base Redis, ne retourne quasiment que des futures qui ne sont pas encore totalement une notion acquise. J'ai donc eu un peu de mal à travailler dessus, notamment lorsqu'il faut renvoyer les éléments en CRUD en asynchrone. J'ai un peu déblayer le terrain avec les for comprehensions mais je manque encore de recule pour connaître les bonnes pratiques.

III. Conclusion :

Cette expérience a été très enrichissante pour moi, j'ai pu travailler avec des technologies que je connaissais pas du tout scala, play, redis. Et bien que je n'ai pas encore tout compris de la bonne manière de les utiliser, je pense avoir beaucoup appris.
Je me découvre vraiment un intérêt pour le Scala qui est un langage qui me parle. Ayant fais de l'Ocamel (et ayant survécu), j'ai réussi à retrouver ce qui semble manquer dans les langages plus traditionnels.

Je suis juste un peu déçue de ne pas avoir pu aller plus loin, ce sera la v0,9. Je souhaitais notamment mettre l'application sur heroku mais par manque de temps, elle n'est qu'en local (code sur github). Je souhaite continuer à travailler dessus pour la rendre meilleur.

IV. Linkographie :

Quelques liens intéressants les technologies et techniques utilisées dans le projet :

Play :

https://www.playframework.com/

Scala :

Site officiel

Neophyte's guide – document très intéressant sur des bonnes pratiques et tips

Redis :

http://redis.io/

https://github.com/Livestream/scredis

Twitter - Redis :

How twitter use redis – Un article sur la mise en place de redis chez twitter

Pour la suite du projet :

https://github.com/ojbruce/Moot