# Rendu Intermédiaire Projet MicroBlogAMU

Samson Mannon
Verniol Baptiste

Notre serveur fonctionne grâce au pool de Thread voleur. Nous avons choisi ce pool car en faisant nos test lors des Tps précédents nous nous sommes apperçus qu'il était le plus performant.
Lorsqu'un client se connecte au serveur pour envoyer une requête, l'Executor se charge de créer un nouveau Thread.

Ce Thread execute la classe SocketHandler et prendra alors entièrement en charge les requêtes du client, il possède plusieurs méthodes :

- Les méthodes pour reconnaître une requête :
    - void run() : la première méthode appelée qui appelle ensuite recoRequest() et requestManager()
    - Request recoRequest(), void requestManager() : Ces deux méthodes servent a reconnaître la requête et appeler les méthodes adéquates pour chaque requêtes.
- Les méthodes pour le mode requête/réponse :
    - void publish() : Ecrit le message dans la base de données et envoie la réponse « OK » ou « ERROR ». Vérifie dans la base de données si des utilisateurs sont abonné a l'author du message ou aux tags qu'ils contient, si il y en a, soit l'envoie directement si le client est connecté, soit le met en attente. Il le recevra quand il se connectera.
    - void rcv_ids() : Envoie les MSG_IDS trouvés en réponse ou « ERROR » si elle n'y arrive pas ou n'en trouve pas.
    - void rcv_msg() : Envoie une réponse « MSG » contenant le message recherché.
    - void reply() : Fonctionne comme publish() mais spécifie dans la base de données que le message est une réponse a un autre. « reply_to_id:id » dans l'en-tête du message.
    - void republish() : Fonctionne comme publish() mais spécifie dans la base de données que le message est une re-publication d'un autre. « republished:true » dans l'en-tête du message.
- Les méthodes pour le mode flux :
    - void connect() : Ajoute le socketHandler a la liste des clients connectés puis verifie si cet utilisateur a des messages en attente auquel il est abonné, si oui les envoie a la suite.
    - void subscribe() : Ajoute a la base de données l'information que l'utilisateur connecté est maintenant abonné à l'author ou au tag spécifié dans la requête SUBSCRIBE
    - void unsubscribe() : Supprime de la base de donnée l'abonnement spécifié dans la requête UNSUBSCRIBE pour l'auteur connecté.

Nous stockons toute les informations du serveur dans une base de donnée représentée par 3 fichiers.

MsgDB, le fichier qui contient tout les messages. Un message est représenté sous la forme d'une ligne :
author:@author msg_id:id | corps

SubscriptionDB, le fichier qui contient tout les abonnements. Une ligne représente :
@author @author1 @author2 …
OU
#tag @author1 @author2…
ou @author1 et @author2 sont abonnés a @author et à #tag

WaitingMsgDB, le fichier qui contient tout les messages en attente pour chaque utilisateur. Sous la forme :
@author id1 id2 id3 …
ou les messages dont les id sont id1, id2, id3 sont en attente pour @author, quand il se connectera il recevra tout les messages en attente et la ligne se supprimera.

Quand un utilisateur se connecte en envoyant une requête CONNECT, il est ajouté dans une structure de donnée de type :
HashTable(K, V)
- K : le nom d'utilisateur
- V la liste de SocketHandler pour cet utilisateur

De cette manière un client peut se connecter plusieurs fois, sur plusieurs appareils par exemple, et ses Socket seront stockés dans sa liste

9.
La communication se fait de manière asynchrone car elle se fait depuis le SocketHandler qui est un Thread et un Thread est asynchrone.

Les Clients :

Pour implémenter les clients, on a créé une classe abstraite Client qui traite la connexion avec le serveur et dont hérite toutes les autres classes de clients. Il y a aussi la classe abstraite ClientWithIdentification qui hérite de Client et qui permet l'identification de l'utilisateur en lui demandant son pseudo.

Ensuite, on a créé des classes pour les différentes requêtes et leurs traitements pour aider à une implémentation plus propre des clients et éviter ainsi la répétition. Les requêtes créent les requêtes adéquates et les traitements entre autre les envoies au serveur.

Il y a ainsi les classes abstraites Request et RequestWithBody dont hériteront les autres requêtes, pareil pour la classe abstraite Treatment et les classes treatment.