Rapport Java
Tchat multi-utilisateur

A. Bento da Silva
A. Gillioz

3 juin 2017

1 Structure

Nous avons définis notre logiciel en six classes, trois pour la partie client et trois pour la partie serveur.

Client :
Classe main du client, fait appel à l'interface graphique ClientViewController.

ClientServices :
ClientServices contient la majorités des fonctions du client, notamment la connexion, l'envoi de message, l'envoi de fichier, etc...

ClientViewController :
Représente l'interface graphique, du client qui donne accès aux différentes options. Et permet aussi de récupérer les actions des utilisateurs

Server :
La classe main du serveur, qui va attendre de recevoir de nouveaux sockets, donc de nouveaux clients.

ClientHandler :
Représente les clients côté serveur, chaque instances de ClientHandler possède la liste de l'ensemble des instances, quand un message est envoyé depuis une de ces instances, il est renvoyé à toutes les autres.

Trame des fichiers :
La trame pour l'envoie de fichiers se décomposent de la manière suivantes :
- 1 byte pour la taille du nom de fichier
- n bytes pour le nom de fichier
- m bytes pour le fichier en lui même

Ce qui va nous permettre, à la réception de la trame de pouvoir récupérer les méta-données (nom du fichier) avant de réécrire le fichier.

2 Diagramme de classe

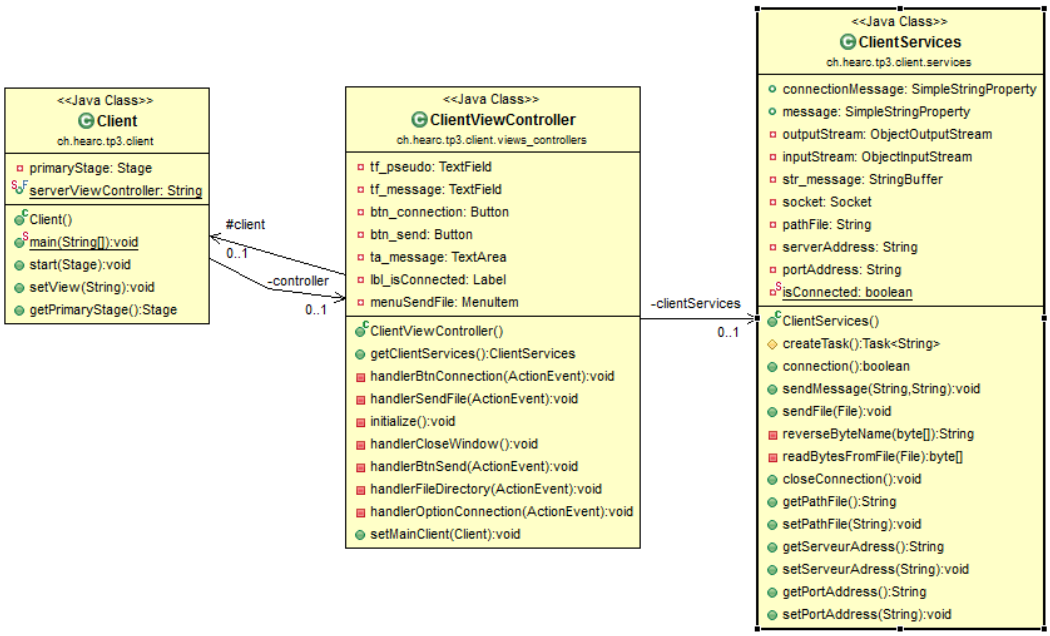


Diagramme de classe côté serveur

Comme on peu le voir la classe Client fait appel à l'interface graphique ClientViewController avec setView.
Cette interface qui pendant l'évènement handlerBtnConnection, appel ClientServices pour faire la connection avec le serveur.

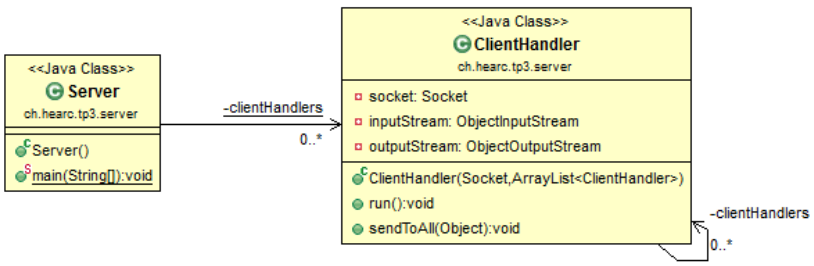


Diagramme de classe côté serveur

La classe Server crée le ServerSocket, et attends que ce dernier accepte des utilisateurs, pour chaque utilisateur accepté elle crée un ClienHandler dans un nouveau thread. Et c'est les ClientHandler qui gèrent les messages des utilisateurs

3 Motivation des choix

Nous avons dès le début, réfléchit à comment connecter plusieurs utilisateurs au tchat, nous avons compris qu'il était pour cela nécessaire d'avoir une représentation des clients côté serveur, ce qui nous as permis de créer la classe ClientHandler.

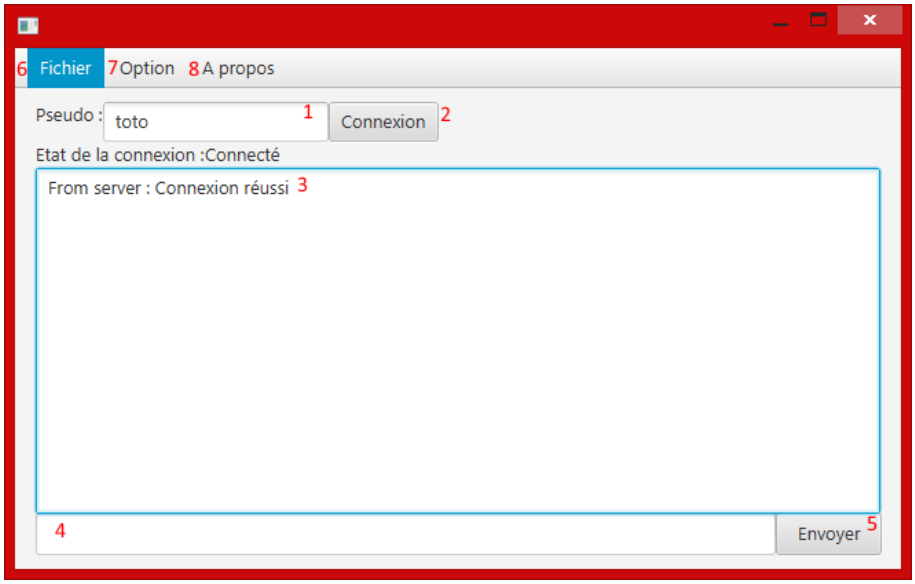
Nous avons également pensé à l'envoi de fichier, et pour cela nous avons décidé d'envoyer des variable de type objet, plus tôt que des strings. Ce qui permet au serveur de traiter les fichiers comme les messages, en renvoyant simplement l'objet à tous les utilisateur, et c'est à l'utilisateur de faire la différence entre les simples messages et les fichiers.

Un autre choix que nous avions fait à été de rendre les interactions du serveur les plus kiss possible, quand il reçoit un message il le renvoi et c'est tout. Il n'identifie pas ces utilisateurs, il leurs crée juste un socket et attends qu'il y ait du trafic sur ce socket. Ce qui veut également dire qu'un utilisateur qui envoi un fichier va également le recevoir et que les concepts de bannissement, de pseudo unique, etc... ne pourrait pas être implémenté sur ce serveur, ce qui dans notre cas ne pose pas de réel problème.

4 Manuel d'utilisation

La première chose à faire est de lancer le serveur.jar, ce qui va permettre aux utilisateurs de se connecter.

Après quoi vous pouvez lancer le client.jar, ce qui vous ouvrira une fenêtre tel que celle-ci.



Interface graphique du client

1. Cette textbox permet de rentrer votre pseudo.
2. Comme son nom l'indique ce bouton permet de faire la connexion vers le serveur, si cette connexion échoue une boîte de dialogue apparaît pour prévenir l'utilisateur.
3. Représente le tchat en tant que tel ou les différents messages sont affichés.
4. La textbox qui vous permet d'entrer des messages.

5. Permet d'envoyer le contenu de votre message au serveur.
6. Contient les outils d'envoi et les options de réception de fichier.
7. Contient les options réseaux, notamment l'adresse du serveur ainsi que le port utilisé pour s'y connecter.
8. Contient les coordonnées des développeurs.