Floriane Geoffroy - Lucas Radureau

Manuel utilisateur - Projet de clavardage

I. Qu'est ce que ce système fait ?

Le logiciel permet à plusieurs personnes personnes connectés sur des ordinateurs différents, dans une entreprise par exemple, de communiquer facilement en s'envoyant des messages avec une interface agréable et simple d'utilisation. Chaque ordinateur correspond à un compte différent, il est donc adapté en entreprise où pour la plupart du temps chaque employé à son propre ordinateur.

1) Partie discussion

- Afficher une fenêtre avec les utilisateurs connectés
- Actualiser la liste des utilisateurs connectés en temps réel
- Cliquer sur un utilisateur pour ouvrir une conversation et chatter avec lui
- Envoyer des messages
- Recevoir des messages
- Afficher les anciens messages envoyés/reçus avec une personne avec qui l'on vient de commencer à chatter
- Gérer plusieurs conversations en même temps avec des utilisateurs différents
- Réduire les fenêtres
- Quitter une conversation

2) Partie connexion/gestion

- Ouvrir une fenêtre de connexion au lancement du logiciel
- Se connecter de manière **sécurisée** avec un couple (login/MDP)
- Fermer fenêtre de connexion et ouvrir la fenêtre des utilisateurs connectés si les login et MDP sont bons.
- Gérer l'unicité des pseudos
- Choisir son pseudo (sous réserve de disponibilité)
    - Durant la phase de connexion
    - A tout moment (à condition d'être connecté)

II. Quels sont les procédés utilisés pour utiliser ces fonctionnalités

1) Partie discussion

Envoyer/Recevoir message	Envoi en TCP de messages entre 2 utilisateurs sur un PC différent, fenêtre de chat affichant les messages envoyés et reçus avec un utilisateur, avec également une zone de texte dans la fenêtre pour envoyer les messages.
Afficher fenêtre utilisateurs connectés	Fenêtre utilisant la liste des utilisateurs connectés connus par l'utilisateur actuel.
Actualisation en temps réel	Information de connexion / déconnexion envoyé par les autres utilisateurs en UDP et qui est tenu en compte dans la fenêtre des utilisateurs connectés
Cliquer sur un utilisateur pour chatter avec lui	Bouton dans la fenêtre des utilisateurs connectés ouvrant une fenêtre de chat
Afficher les derniers messages envoyés / reçus avec un utilisateur lorsque l'on ouvre une conversation avec lui	Stockage de tous les messages envoyés / reçus avec une date, une source et un destinataire associé. Récupération au moment de l'ouverture de la fenêtre de chat des 10 messages les plus récents stockés faisant partie de la conversation entre les 2 utilisateurs concernés. Affichage de ces messages dans la fenêtre avec distinction des messages reçus et envoyés et triés chronologiquement.
Gérer plusieurs conversations en même temps avec des utilisateurs différents	Utilisation du multithreading permettant d'avoir un programme pour chaque conversation et qu'ils puissent s'exécuter tous en même temps. On a une fenêtre de chat pour chaque utilisateur avec lequel on discute.
Réduire les fenêtres	Les fenêtres possèdent toutes un bouton pour qu'elle soit réduites (installé de base sur les fenêtres JAVA SWING)
Quitter une conversation	Fermeture de la fenêtre de chat, fermeture des sockets TCP de réception de message et d'envoi de messages et arrêt du programme associé à la conversation. Affichage pour l'autre utilisateur que son interlocuteur a quitté la conversation.

2) Partie connexion/gestion

Ouvrir une fenêtre de connexion au lancement du logiciel	Affichage d'une fenêtre de connexion au démarrage comportant un champ pour rentrer son login, un autre champ pour rentrer son MDP et un dernier champ pour rentrer son pseudo que l'on veut utiliser pour cette connexion + un bouton pour se connecter une fois les différents champs remplis par l'utilisateur.
Se connecter de manière sécurisée avec un couple login/MDP	Enregistrement lors de la 1ère utilisation du logiciel du login et MDP entrés dans la fenêtre de connexion dans une BDD propre au PC. Vérification lors des utilisations suivantes que le couple login/MDP entré est bien le même que celui rentré la première fois et qui est stocké dans la BDD. Si ce n'est pas le cas alors les champs de texte login et MDP sont vidés et il faut recommencer l'opération en rentrant les bons identifiants.
Fermer fenêtre de connexion et ouvrir la fenêtre des utilisateurs connectés si les login et MDP sont bons.	Après validation des identifiants entrés, fermeture de la fenêtre, envoi en UDP à tout le réseau que l'on vient de se connecter puis attente de réception en TCP de la liste des utilisateurs connectés avec leurs adresses IP respectives. Cette liste est envoyée par le dernier utilisateur qui s'est connecté avant nous. Une fois la liste reçu, ouverture de la fenêtre des utilisateurs connectés qui utilise cette liste. Si on ne reçoit pas la liste au bout d'un certain temps cela veut dire que l'on est seul sur le réseau. Dans ce cas là on ouvre quand même la fenêtre des utilisateurs connectés, on aura simplement personne à qui parler pour le moment.
Choisir, Modifier pseudo	Dans la fenêtre de connexion ouverte au lancement du programme l'utilisateur peut choisir son pseudo. Celui-ci est ensuite envoyé à tous les autres utilisateurs connectés. Une fois connecté, l'utilisateur peut changer de pseudo en appuyant sur un bouton dans la fenêtre des utilisateurs connectés, ce qui lui ouvre une pop-up avec une zone de texte pour entrer le nouveau pseudo souhaité, qui est ensuite envoyé à tous les autres utilisateurs.
Gérer unicité des pseudos	Lors de la connexion d'un nouvel utilisateur, quand on envoie en UDP à tout le monde que l'on vient de se connecter on envoie son pseudo que l'on vient de choisir. Chaque utilisateur déjà connecté qui reçoit ce pseudo va vérifier qu'il n'est pas déjà pris, et si c'est le cas il ne l'ajoute pas à sa liste d'utilisateurs connectés. L'avant-dernier à s'être connecté va en plus de cela renvoyer en TCP une information au nouvel utilisateur lui indiquant que son pseudo est déjà pris. Cela se traduit du côté nouvel utilisateur par l'apparition d'un pop-up l'invitant à entrer un nouveau pseudo. Une fois connecté et donc avec la liste des utilisateurs connectés à jour en permanence, si l'utilisateur veut changer de pseudo il vérifie juste dans sa table des utilisateurs connectés si le pseudo qu'il veut prendre et déjà utilisé. Si ce n'est pas le cas on laisse la pop-up de changement de pseudo ouverte pour que l'utilisateur choisisse un pseudo correct.

III. Technologies utilisées

L'ensemble du système a été codé en JAVA sous Eclipse.

Il a été nécessaire de faire du multithreading via la classe *Thread* pour par exemple pouvoir gérer plusieurs conversations en même temps ou encore pouvoir discuter avec quelqu'un tout en surveillant si quelqu'un vient de se connecter ou se déconnecter.

Pour échanger des données entre utilisateurs du réseau, nous avons utilisé des sockets, UDP et TCP, qui font le lien entre la couche applicative et la couche réseau. C'est la classe *java.net* qui nous a permis de créer des sockets et les gérer.

Pour la partie interface, nous avons utilisé la classe *JAVA Swing* pour créer des fenêtre comprenant des labels, des champs pour entrer du texte, des bouts etc.

Pour stocker le login et mdp de l'utilisateur ainsi que les messages envoyés aux différents utilisateurs nous avons utilisé un moteur de base de données relationnelle : *Apache Derby*. Celui-ci est écrit en langage JAVA et pouvait être embarqué dans notre programme JAVA. En plus de cela il est de très petite taille (2MB), ce qui nous permet de respecter la contrainte de taille du logiciel issu du cahier des charges.
Pour gérer la base de donnée locale on utilise *JDBC* ainsi que *SQL* pour écrire les requêtes.

IV. Architecture du système

Le programme se lance via la classe *lancement.java .*

De cette classe, on crée directement une fenêtre de connexion invitant l'utilisateur à entrer son login, son MDP et son pseudo.

Puis une fois que l'utilisateur à entrer ses identifiants, il faut les vérifier.

Pour cela, on crée un objet de type *database*, qui représente la BDD avec les messages et les identifiants au préalable installé.

On exécute la methode *verification()* de *database* qui consiste à récupérer les identifiants stockés dans la BDD avec une requête SQL et de les comparer à ceux entrés par l'utilisateur. La méthode retourne 1 si les identifiants concordent, 0 sinon. Si la BDD est vide, c'est à dire que c'est la première fois qu'on se sert du logiciel sur ce PC, on crée une table *Authentification* et on y insère le login et le MDP que l'utilisateur vient de rentrer puis on retourne 1.

La méthode est rappelé tant que l'utilisateur n'entre pas les bons identifiants.

Une fois cette étape passé, la fenêtre est fermée, on retourne dans *lancement* et on crée un objet ***utilisateur*** initialisé avec le pseudo entré.

Un *utilisateur* est caractérisé par : *un pseudo, une adresse IP, une adresse IP de broadcast, une liste des pseudos connectes, une liste des adresses IP connectes.*

Le constructeur d'*utilisateur* initialise l'adresse IP et l'adresse IP de broadcast d'*utilisateur*.

Tout au long du programme les 2 listes sont étroitement liées. En effet, le pseudo numéro ***i*** de la liste des pseudos correspond toujours à l'adresse ***i*** de la liste des adresses.

Une fois l'utilisateur créé, il doit se préparer à recevoir éventuellement des messages de la part d'autres utilisateurs.
Pour cela, les méthodes *demarrerserveurudp()* et *demarrerserveurtcp()* sont exécutés.
Elle consiste à créer un objet *Threadtcp* et un objet *Threadudp*, chacun caractérisé par un *utilisateur*.

Le thread TCP crée un socket sur un port bien défini et le met en état d'accept(). Ainsi dès que quelqu'un souhaitera démarrer une conversation il se connectera en TCP à ce socket, ce qui engendrera la création d'un objet *Thread* qui correspond à une conversation, que l'on expliquera plus tard.
Puis le thread TCP retourne en état d'accept et ainsi de suite, il gère donc les demandes de créations de conversation.

Le thread UDP sert lui à gérer les phases de connexions, déconnexions, changement de pseudos. A son lancement il crée un socket UDP qui se met en écoute sur un port bien défini. Selon les types de paquets reçus il va mettre à jour les listes des pseudos connectés et adresses connectés de l'utilisateur en modifiant un pseudo (changement de pseudo d'un utilisateur du réseau) , ajoutant un élément à chaque liste (nouvel utilisateur) ou en en supprimant un (deconnexion).

Une fois qu'il est prêt et que les *Threads* sont lancés, l'utlisateur doit signaler sa présence sur le réseau et récupérer la liste des pseudos connectés et l'adresse IP de chaque utilisateur associé à ses pseudos.

Pour cela la méthode *connexion* d'utilisateur est executée. Elle crée un socket UDP et envoie en broadcast son pseudo sur le port UDP d'écoute de *ThreadUDP.*

Les autres utilisateurs regarde le pseudo reçu en UDP et si il est nouveau et provient d'une adresse jusqu'a présent méconnue à leur yeux, ils mettent à jour leurs 2 listes avec le pseudo envoyé et l'adresse source du paquet. L'avant dernier utilisateur à s'être connecté va en plus lui renvoyer au nouvel utilisateur sa liste des pseudos connectés ainsi que la liste des adresses connectés, en TCP.

Le nouvel utilisateur quant à lui après avoir envoyé son pseudo en UDP à directement ouvert un socket TCP pour recevoir les listes à jour par le dernier utilisateur à s'être connecté avant lui.

Si les autres utilisateurs reçoivent un pseudo en UDP et que ce dernier est déjà utilisé alors ils n'en tiennent pas compte et ne mettent pas à jour leur listes.
Le dernier utilisateur à s'être connecté avant le nouvel utilisateur va quant à lui envoyer en TCP à ce dernier une information comme quoi son pseudo est déjà pris plutôt que lui envoyer les listes à jour. En réception le nouvel utilisateur va être invité dans une pop-up à retaper son pseudo et l'opération sera répétée jusqu'à avoir un pseudo valide.

Une fois les listes de l'utilisateur à jour, on crée un objet *fenetredesutilisateursconnectes*. Cette fenêtre est initialisée avec la liste des pseudos connectés. Elle affiche un bouton pour chaque élément de la liste des pseudos connectés avec marqué sur chacun le pseudo en question.
Il y aussi dans cette fenêtre un bouton pour changer de pseudo. La méthode utilisée pour changer de pseudo à ce moment-là est sensiblement la même que pour changer de pseudo au départ, sauf que ce coup-ci l'avant dernier utilisateur connecté ne renvoie pas sa liste des utilisateurs connectés à celui qui vient de changer de pseudo.

Si l'utilisateur clique sur le pseudo d'un autre utilisateur, cela correspond à une demande de connexion vers l'autre utilisateur. On a donc le *Threadtcp()* de l'autre utilisateur qui reçoit une demande de connexion et qui crée un *Thread()* qui va être dédié à la conversation créée. Un message en TCP de confirmation vers l'utilisateur qui vient de demander la connexion est envoyée depuis l'autre utilisateur. Ce message, envoyé depuis le nouveau socket crée permet à l'utilisateur qui le reçoit de savoir sur quel port il devra désormais "parler" pour communiquer avec l'utilisateur distant. A la réception de cette confirmation, l'utilisateur crée lui aussi un *Thread().* A la création d'un *Thread()*, on crée un socket TCP qui servira à recevoir et envoyer les messages liés à cette conversation. On crée aussi une *fenêtre()* avec marquée le pseudo de l'utilisateur distant dont on a récupéré les informations au moment de la demande de connexion et de la confirmation en TCP.

A l'initialisation de la fenêtre de conversation, on récupère les messages stockés dans la BDD qui ont pour adresse de destination l'adresse de l'utilisateur distant, et les messages qui ont pour source l'adresse de l'utilisateur distant et comme adresse de destination notre adresse, tout cela trié par date. Cette requête se fait en SQL. On affiche ensuite dans la fenêtre les messages dans l'ordre avec la date pour chacun.

Une zone de texte est présente dans la fenêtre avec un bouton "envoyer" à côté. A l'appui sur envoyer, le message entré est daté, stocké dans la BDD et envoyé grâce au socket TCP sur le socket TCP lié à la conversation de l'utilisateur distant.

Le principe est le même dans l'autre sens, le socket TCP lié à la conversation écoute continuellement si il reçoit des messages et si c'est le cas ces messages sont enregistrés dans la BDD de l'utilisateur qui reçoit le message puis affiché dans la conversation.

La fenêtre est scrollable grâce à la classe *JScrollPanel.*

Si un des 2 utilisateurs clique sur la croix de la fenêtre pour quitter, la fenêtre est fermée et donc les sockets sont fermés et le *Thread()* se termine.L'utilisateur distant, quand il essayera d'envoyer un message dans la fenêtre à l'utilisateur qui vient de quitter, verra s'afficher dans la conversation un message lui indiquant que son interlocuteur n'est plus là car une exception liée au socket distant qui vient d'être fermé et n'existe plus sera levée.

Si un utilisateur quitte la fenêtre d'affichage des utilisateurs connectés en cliquant sur la croix de cette fenêtre, un message est envoyé en broadcast à tout le monde en UDP contenant la chaîne de caractère suivante : "*deconnexion*". Le *Threadudp()* de chaque utilisateur va lire le message reçu en UDP et si il voit *"deconnexion"* il va supprimer de sa liste des adresses ip connectés l'adresse d'où le paquet a été émis, et selon l'index dans le tableau où se trouve cette adresse, il va pouvoir supprimer le pseudo associé qui est au même index dans la liste des pseudos connectés.

Dès qu'un changement est opéré dans la liste des pseudos connectés, la fonction *actualiser()* est utilisé, elle permet de rafraîchir la fenêtre des utilisateurs connectés en supprimant tous les boutons qui y figure et en les remettant tout d suite instantanément en fonction de la nouvelle liste d'utilisateurs connectés.

V. Améliorations possibles

Tout fonctionne au mieux dans notre logiciel. Toutes les fonctions qui étaient demandées dans le cahier des charges ont été traités, excepté l'envoi d'images et de documents dans les conversations. L'interface graphique aurait pu également être perfectionnée également.

Le seul problème restant est le fait que si un utilisateur quitte brusquement le programme (coupure d'électricité par exemple), aucun message de déconnexion sera envoyé aux autres utilisateurs et donc ils le verront toujours affiché dans la fenêtre des utilisateurs connectés.

Pour régler cela, il nous a été proposé de réaliser un serveur de présence, qui vérifierait si chaque utilisateur est toujours actif, ou en mode occupé par exemple, ou si il n'est plus du tout joignable.

La mise en place de ce serveur demandait beaucoup de travail pour régler un seul véritable problème, c'est pour cette raison que nous avons préféré nous concentrer sur cette version sans serveur, pour pouvoir proposer une version finale qui marche correctement, qui fait la très grande majorité de ce qui été demandé dans le cahier des charges et qui est relativement agréable, tout cela dans le temps imparti.