

DJEBAR Loïc
TISSOT Evan

Manuel d'utilisation Système de Clavardage

SOMMAIRE

I / L'architecture du système développé

II / Les fonctionnalités supportées par le système

III / Scénario d'utilisation type

IV / Requirements

V / Tests unitaires

INTRODUCTION

L'application que nous avons développée est un système de clavardage. Il permet entre autre d'échanger des messages instantanément avec d'autres utilisateurs connectés.

Ce document est un guide de prise en main de l'application de clavardage que nous avons développée. Il a pour but d'expliquer comment notre application est organisée tout en ne rentrant pas dans des détails très techniques. Il est également voué à expliquer comment utiliser notre application, à décrire les fonctionnalités présentes et à donner le cadre dans lequel l'application fonctionne.

I/ L'ARCHITECTURE DU SYSTÈME DÉVELOPPÉ :

Notre système se basant sur des caractéristiques à la fois de systèmes dits « clients » mais aussi « serveurs », il a fallu mettre en place une architecture tenant compte de ces deux « modes » de fonctionnement. En effet, notre système peut être client dans l'interaction avec un système A et dans le même temps, il peut être serveur dans une autre interaction avec un système B.

Voici donc un schéma simple de l'architecture du système :

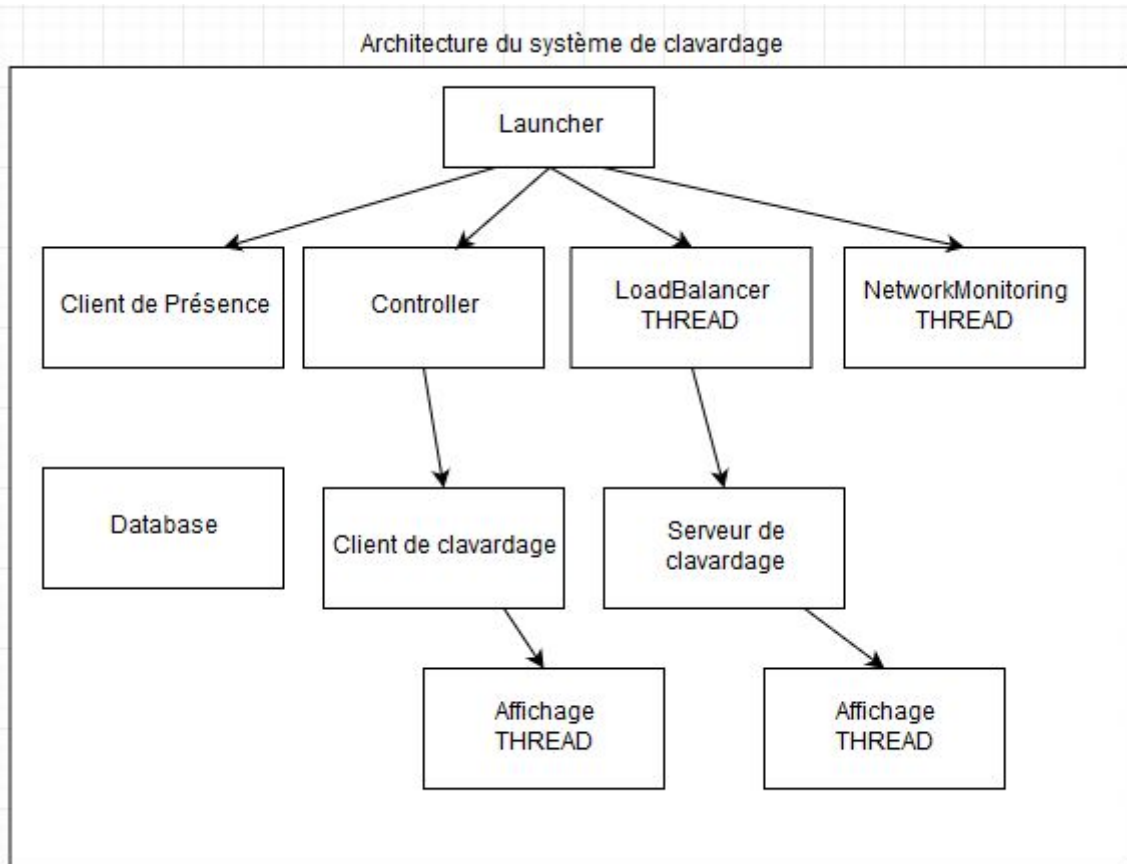


Figure 1 – Architecture du système de clavardage

Cette modélisation de l'architecture est volontairement simpliste pour que l'on puisse comprendre l'idée générale du fonctionnement du système. L'interface graphique et tout ce qui est « objets » nécessaires au fonctionnement du système ne sont pas représenté ici.

Voyons plus en détails les éléments qui composent la figure 1 :

- **Le Launcher**

C'est l'élément qui permet de démarrer le système de clavardage. Il lance les threads LoadBalancer et NetworkMonitoring ainsi que les fonctions d'initialisation du Controller. Il affiche également la fenêtre de connexion permettant de récupérer le login de l'utilisateur et créer la Database associée.

- **Le Controller**

Cet élément contient les fonctions d'annonce sur le réseau :

L'annonce du nouvel utilisateur sur le réseau via le broadcast UDP du pseudo.

L'annonce d'une déconnection d'un utilisateur.

L'annonce d'un changement de pseudo.

- **Le NetworkMonitoring**

Ce thread récupère sur le port 1235 toutes les annonces UDP sur le réseau faites par les fonctions du Controller des autres utilisateurs présents sur le réseau. Il récupère les pseudos reçus en broadcast, les compare au sien, puis les ajoute à sa liste d'utilisateurs connectés s'ils sont différents. Il renvoie un message au destinataire si le pseudo reçu est identique au sien. Il met à jour sa liste d'utilisateur lorsqu'il reçoit une déconnexion ou un changement de pseudo.

- **Le LoadBalancer**

Ce thread récupère (accepte) toutes les connexions TCP sur le port 1234 qui correspondent à des demandes de « clavardage » et renvoie à la source un numéro de port supérieur à 1300 tout en lançant un serveur de clavardage en attente d'acceptation sur ce port. Ainsi le client qui reçoit ce nouveau numéro de port ferme sa connexion TCP actuelle et en ouvre une nouvelle sur le port reçu.

- **Le serveur de Clavardage**

Est lancé sur un port précis par le LoadBalancer et accepte la première connexion reçue pour démarrer l'affichage de la fenêtre de discussion, et l'exécution du THREAD d'affichage.

- **Le client de Clavardage**

Établi une connexion TCP sur le port 1234 du destinataire afin de recevoir un numéro de port, pour ensuite terminer la session TCP actuelle et se connecter toujours en TCP sur le numéro de port reçu. Une fois cela effectué, comme pour le serveur, on lance la fenêtre de chat ainsi que le THREAD d'affichage.

- **La Database**

Celle-ci est en SQL. Un fichier de base de données (*.db) est créé pour chaque utilisateur qui se connecte sur une machine. La base de données est constituée d'autant de tables que d'utilisateurs avec qui notre utilisateur principal a dialogué. Chaque table est constituée de 3 différents champs :

Temps de type VARCHAR(20) qui désigne la date d'émission du message

Pseudo de type VARCHAR(20) qui désigne l'émetteur du message

Message de type VARCHAR(150) qui désigne le contenu du message

- **Le client de présence**

Le client de présence se signale lors d'une phase d'initialisation au serveur de présence. Il envoie une requête HTTP avec comme argument son username, et son adresse IP. Il envoie par la suite régulièrement des requêtes HTTP au serveur de

présence (toutes les 5 secondes) pour récupérer la userList actualisée. Lors de la déconnexion de l'utilisateur, le client signale sa déconnexion au serveur.

- **Le serveur de présence**

Cette partie est à part car elle n'est pas incluse dans l'application. Le serveur, constitué par un servlet doit tourner sur un serveur HTTP.

Le serveur de présence reçoit des requêtes de la part des clients et renvoie différentes informations en retour. Il possède notamment une userList mise à jour en fonction des requêtes et qu'il transmet aux clients.

II/ LES FONCTIONNALITÉS SUPPORTÉES PAR LE SYSTÈME :

Afin de bien comprendre l'intérêt de notre système, listons toutes les fonctionnalités qu'il supporte :

Découverte des utilisateurs

Chaque utilisateur nouvellement connecté broadcast en UDP ses informations sur le réseau, sur le port 1235, les informations sont alors captées par le THREAD NetworkMonitoring puis ajoutées à la liste des utilisateurs connectés. Le THREAD renvoie dans le même temps les informations du user local en unicast UDP.

Notification de la déconnexion

Lorsque l'on quitte l'application, une notification est envoyée en broadcast afin que tous les autres utilisateurs savent que l'on s'est déconnecté.

Passage en mode invisible

Le passage en mode invisible permet de continuer à voir les personnes qui sont connectées tout en étant non visible dans la liste des utilisateurs des autres utilisateurs.

Changement de pseudo

Il est possible de changer de pseudo lorsque l'application est déjà lancée. Il suffit d'écrire son nouveau pseudo dans la boîte de dialogue et d'appuyer sur change.

Historique des conversations

Un historique des conversations est gardé en local. L'application interroge la base de données SQL pour afficher l'historique des conversations lorsqu'une discussion est lancée.

Vérification de l'unicité du pseudo

A chaque ouverture de l'application (juste après la saisie du pseudo), notre identité est envoyée en broadcast sur le réseau. Si un utilisateur possède le même Username que celui que nous avons renseigné, nous sommes automatiquement déconnecté et un message d'erreur nous avertira que le pseudo n'est pas unique. Il faudra alors saisir un nouveau username unique (commande change) pour voir s'afficher les autres utilisateurs connectés.

Serveur de présence centralisé

Le serveur de présence sert à garder une liste des utilisateurs mise à jour régulièrement et centralisée. Le serveur communique cette liste à tous ses clients.

Ouverture et fermeture automatique des fenêtres de discussion

Quand l'un des deux utilisateurs ouvre une fenêtre de discussion de son côté, celle-ci est également ouverte du côté du destinataire. De même lorsque l'un des deux utilisateurs ferme sa fenêtre de chat, un message informant de la déconnexion apparaît dans la fenêtre de l'utilisateur restant, puis celle-ci se ferme au bout de deux secondes.

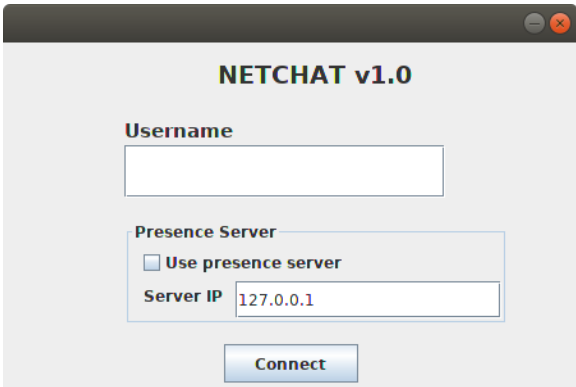
AutoScroll de la fenêtre de discussion

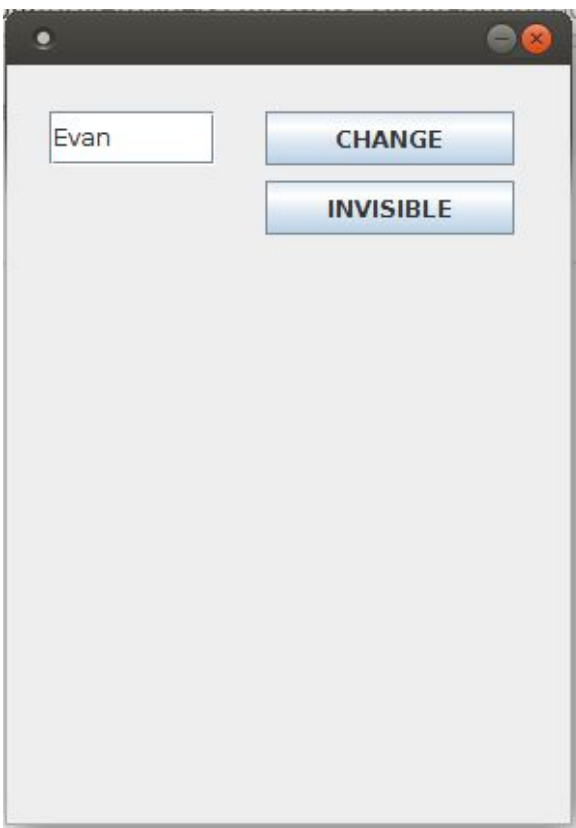
Lors de la réception ou de l'émission d'un message, la fenêtre de discussion va toujours faire en sorte que le dernier message affiché soit bien visible sans aucune intervention de l'utilisateur.

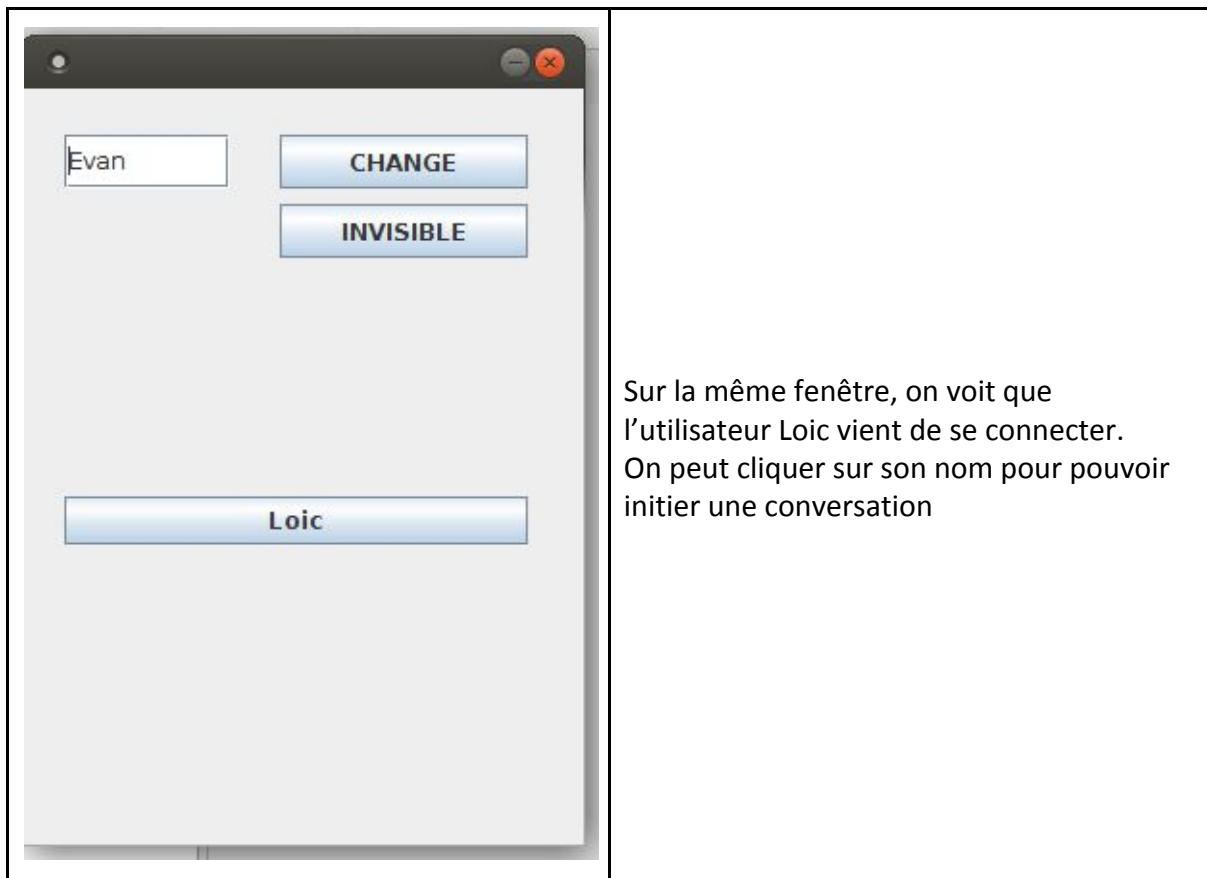
Un mode de fonctionnement avec ou sans serveur de présence

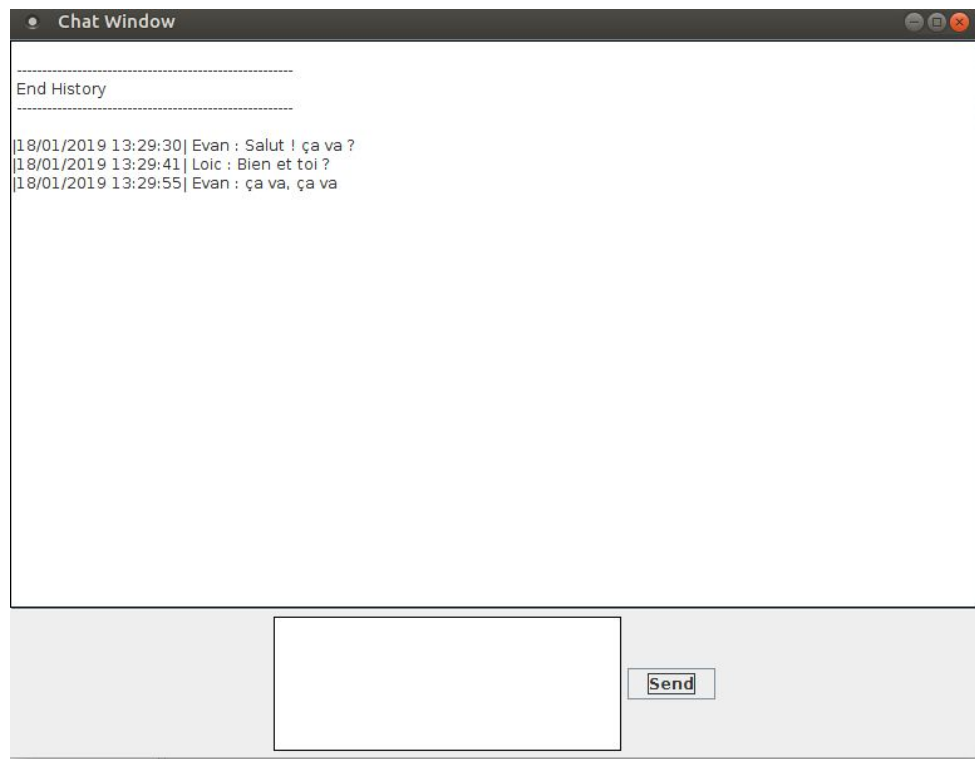
Lors de la connexion, l'utilisateur a la possibilité de sélectionner s'il souhaite n'utiliser que la fonction de découverte des utilisateurs dans son réseau local ou s'il souhaite également activer le serveur de présence qui fonctionnera alors en parallèle de la fonction précédemment énoncée.

III - SCÉNARIO D'UTILISATION TYPE

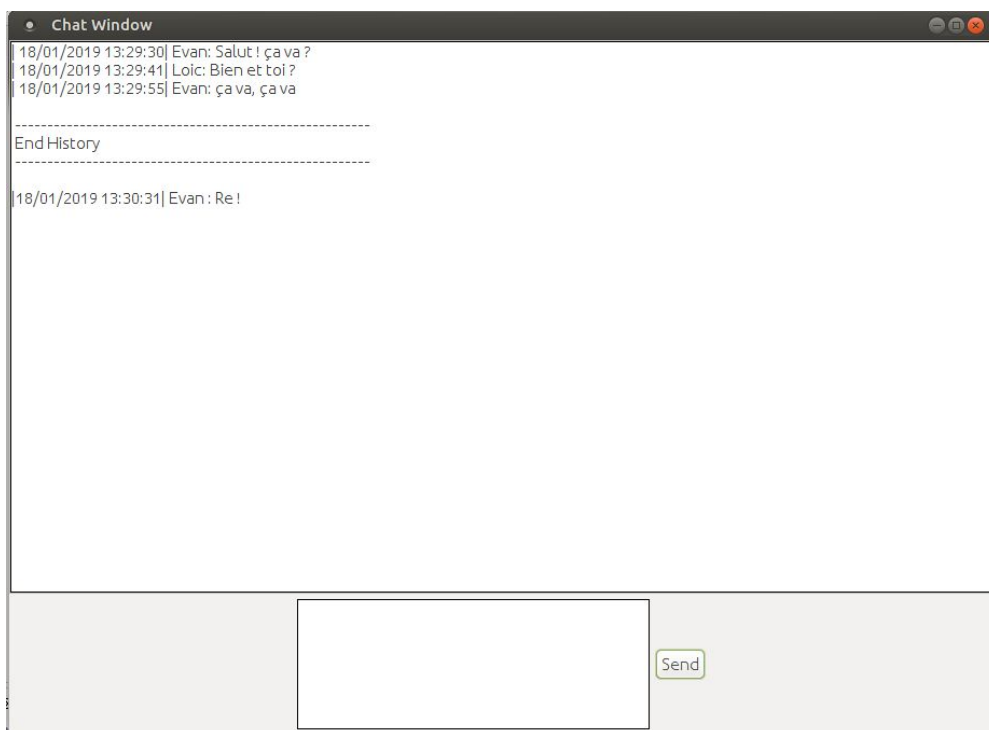
| | |
|---|---|
|  | <p>Lorsque l'utilisateur lance notre application, cette fenêtre apparaît. Il suffit de renseigner son pseudo et d'appuyer sur <i>Connect</i>.</p> |
|---|---|

| | |
|--|--|
|  | <p>Une nouvelle fenêtre apparaît. C'est la fenêtre centrale de notre application. Les utilisateurs connectés seront visibles sur cette fenêtre.</p> <p>Le bouton change permet de changer de Username comme expliqué précédemment.</p> <p>Le bouton invisible permet de passer à l'état invisible comme expliqué précédemment.</p> |
|--|--|





On voit le déroulement d'une discussion. On peut quitter à tout moment une discussion en fermant la fenêtre. L'utilisateur avec qui on discutait sera notifié du fait que l'on a quitté la conversation et sa fenêtre sera automatiquement fermée au bout de deux secondes.



Si on ouvre à nouveau une discussion, on voit l'historique qui s'affiche tout en haut de la conversation.

IV - REQUIREMENTS

Adressage

Pour savoir quelle adresse IP il faut utiliser, l'application passe en revue toutes les interfaces réseau et sélectionne la première adresse qui commence par 192.168.* ou 10.1.*. Cela implique qu'il faut régler au préalable l'adresse IP sous cette forme là sur l'interface qu'on utilise.

Si toutefois vous ne voulez ou pouvez pas adapter votre adresse IP et si vous avez accès au code source de l'application, vous pouvez changer ce comportement dans la méthode `getIP()` de la classe `Adressage`.

SQLite

L'application utilise SQLite pour pouvoir gérer l'historique. Il faut s'assurer que le driver SQLite est présent.

L'application génère un fichier *.db dans le dossier de l'application. Ce fichier représente une base de données. Il faut donc que l'application possède les droits d'écriture pour créer un fichier à cet emplacement.

Serveur de présence

Le serveur de présence est réalisé par un servlet. Cette servlet doit tourner sur un serveur HTTP. Durant toute la phase de programmation et de test, nous avons utilisé Tomcat 8. Notre serveur de présence est donc fonctionnel avec ce programme. Il se peut qu'il fonctionne avec d'autres programmes mais nous ne pouvons rien garantir car nous n'avons pas testé.

V - LES TESTS UNITAIRES

Afin d'évaluer le bon fonctionnement de notre système de clavardage et d'optimiser au mieux les fonctionnalités, il a fallu s'appuyer sur des tests.

Nous avons choisi de réaliser tous ces test manuellement, en effet, tout d'abord nos applications utilisant des port spécifiques sur chaque machine, on ne peut lancer qu'une instance de l'application (Hors pour vérifier le bon fonctionnement, il faut deux instances différentes). Il faudrait donc créer des tests clients/serveurs à exécuter sur chaque machine dans le bon ordre. Enfin, comme les informations sont recueillies via l'interface graphique, il faudrait modifier le code pour automatiser l'envoi de messages.

Les tests que nous avons réalisé manuellement consistent donc en l'exécution sur plusieurs machines d'un même réseau (généralement deux ou trois en simultané) de nos programmes de clavardage. Une fois l'application déployée, nous avons pu vérifier la conformité des fonctionnalités à chaque étape du développement en réalisant un suivi durant toute la conception.