INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Université Fédérale Toulouse Midi-Pyrénées

Département de Génie Électrique & Informatique

Rapport MessageApp: COO/POO

Message App

February 5, 2020

Students :

Rafael	ACHEGA	achega@etud.insa-toulouse.fr
Matthieu	LACOTE	lacote@etud.insa-toulouse.fr
Lucas	PERARD	lucas.perard@etud.insa-toulouse.fr

Contents

1 Conception (COO)

1.1 Diagramme de cas d'utilisation

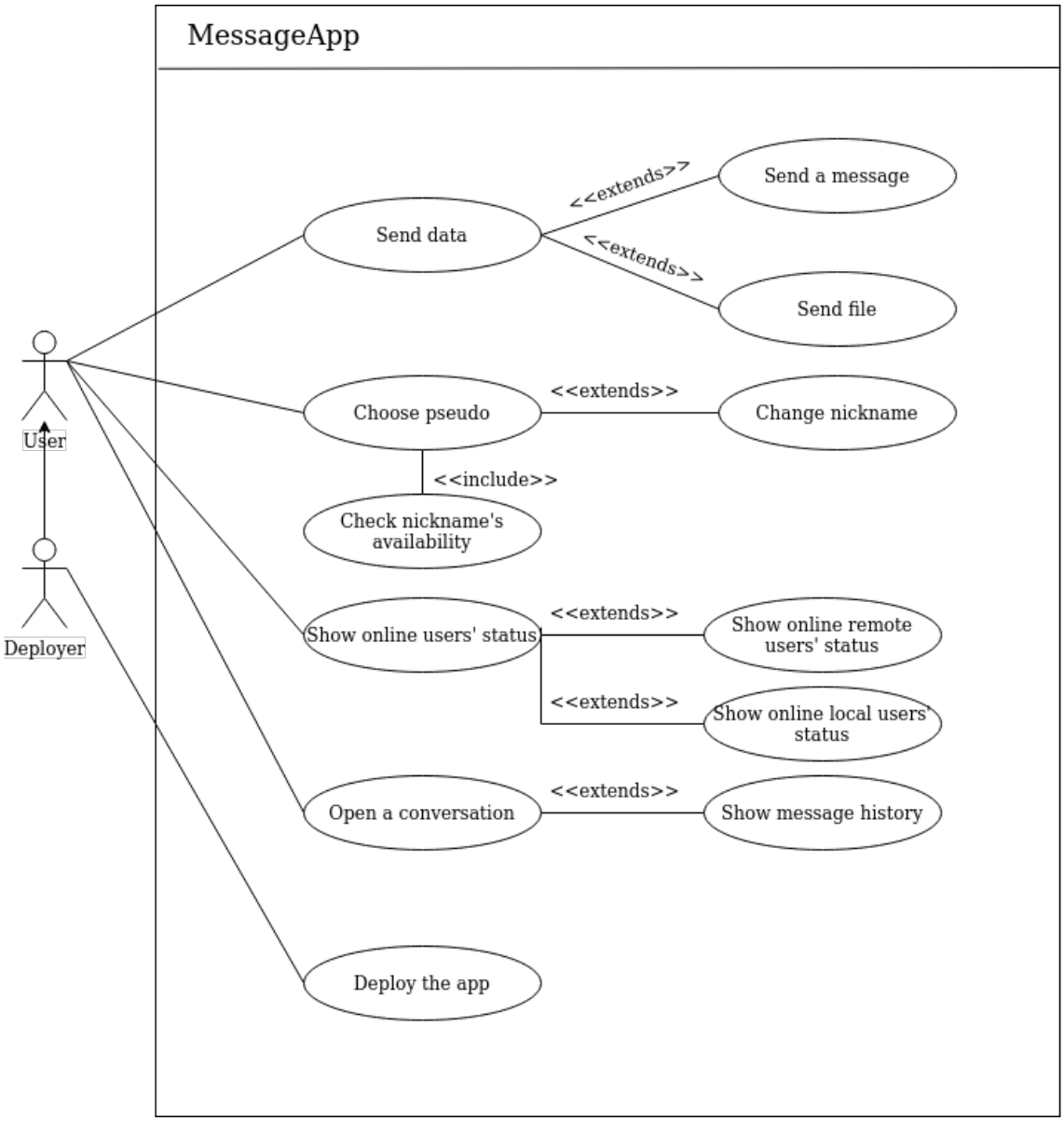

Figure 1: Diagramme de cas d'utilisation

1.2 Diagramme de classes

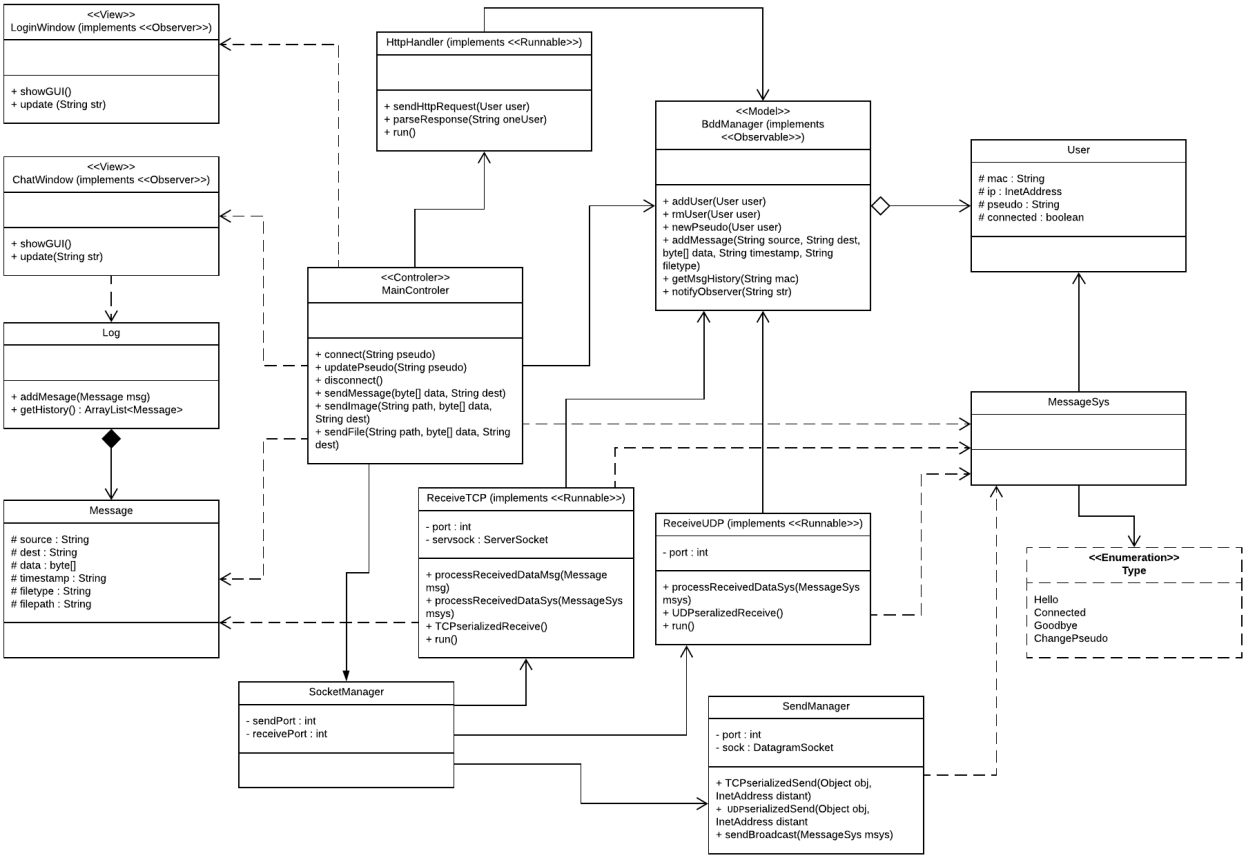

Figure 2: Diagramme de classes

1.3 Diagrammes de séquence

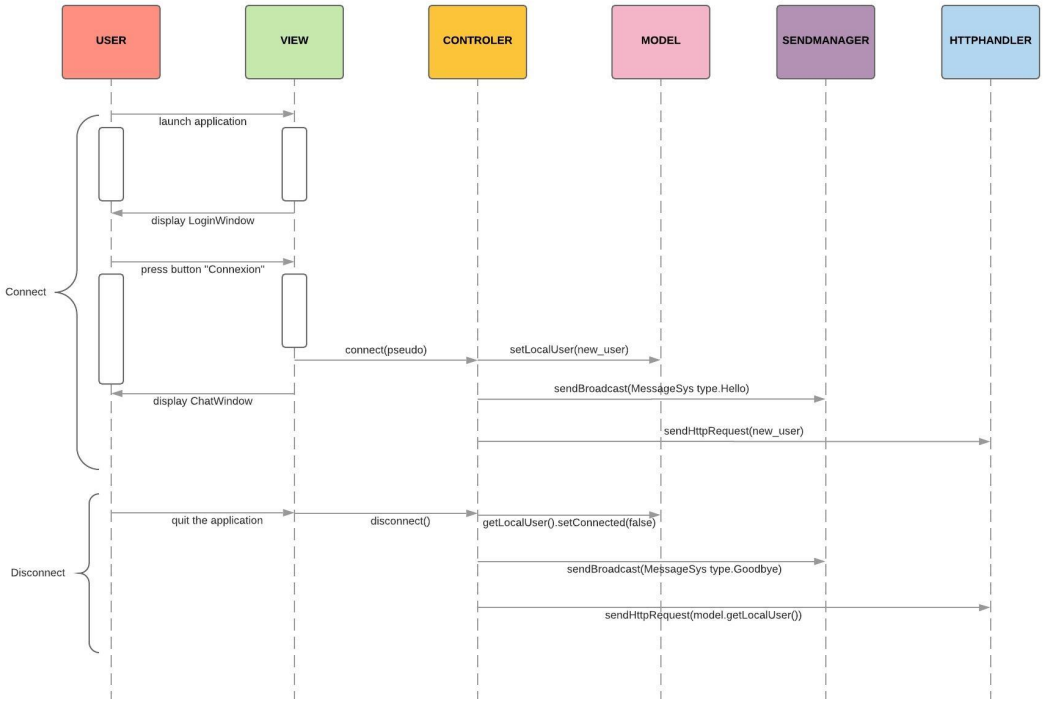

Figure 3: Diagramme de séquence: connexion

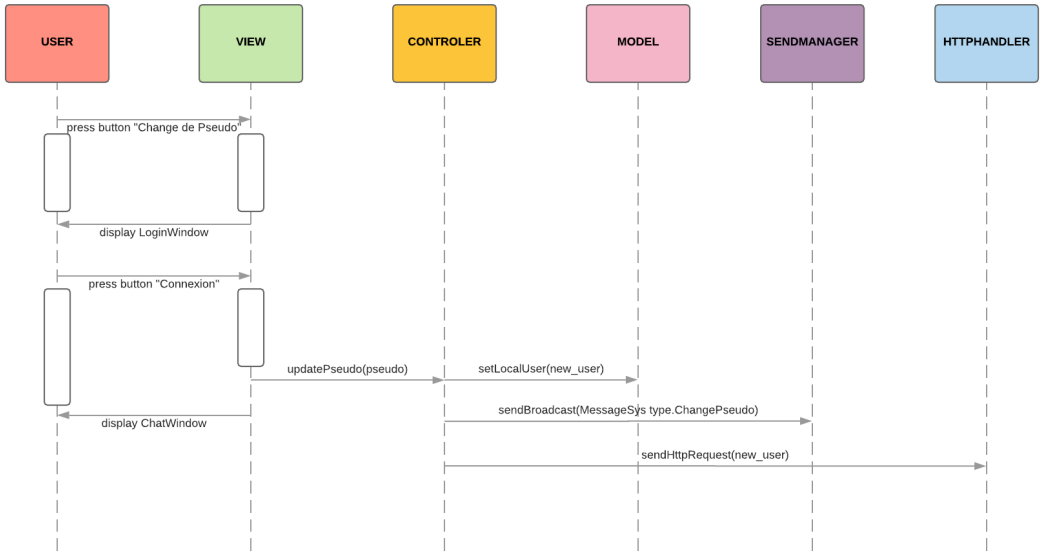

Figure 4: Diagramme de séquence: changement de pseudo

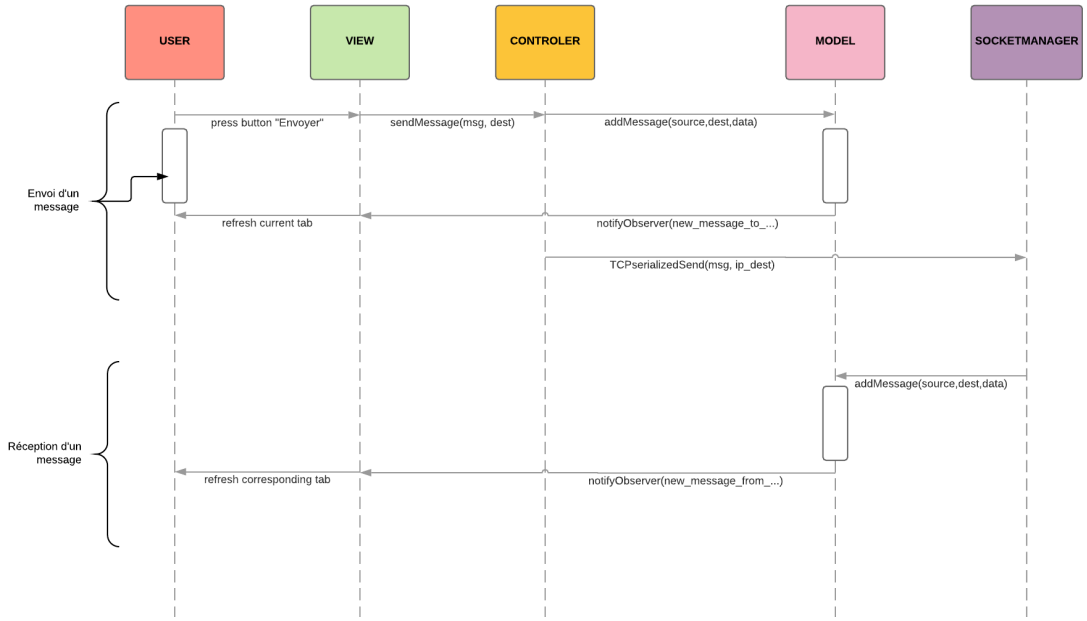

Figure 5: Diagramme de séquence: envoi de message

2 Implémentation (POO)

2.1 Explication de nos choix de conception

2.1.1 Identification par IP et utilisation d'une base de donnée locale

Après lecture du cahier des charges, nous avons interprété l'exigence de décentralisation comme affectant l'ensemble de notre application. Ainsi nous avons décidé de ne pas se baser sur un serveur central pour l'identification et la base de donnée, et donc de tout gérer en local.

Dans ce sens, nous avons choisi d'identifier nos utilisateurs par l'adresse MAC de leur machine sur le réseau local. Nous avons privilégié l'adresse MAC par rapport à l'adresse IP du fait qu'un utilisateur se connectant sur un réseau Wifi où l'adressage est dynamique aurait une nouvelle adresse IP à chaque utilisation ce qui l'empêcherait de s'identifier. En revanche la limite de ce choix réside dans le fait que l'utilisateur doit toujours utiliser la même machine pour se connecter au système de messagerie. Nous avons cependant estimé que dans un contexte d'entreprise où chaque employé possède sa propre machine cette contrainte reste raisonnable et de plus aucune indication du cahier des charges ne va à l'encontre de ce choix.

Pour la base de donnée, nous avons là aussi fait le choix d'une base de donnée locale, basée sur SQLite. Utiliser une base de donnée locale est cohérent avec nos choix au niveau de l'identification. En effet si un utilisateur utilise toujours la même machine pour se connecter, il n'est pas gênant d'avoir une base de donnée sous forme de fichier en local (dans notre cas directement dans le dossier racine de notre application). On pourrait même argumenter sur l'aspect sécurisant de la chose, un attaquant devrait attaquer chaque client pour récupérer l'intégralité des conversations de l'application alors que dans le cas d'un serveur centralisé, si celui-ci est compromis c'est l'ensemble des utilisateurs qui est affecté (que ce soit par du vol de données, par un déni de service, etc. . .).

2.1.2 Utilisation et implémentation du design pattern MVC

Pour assurer la fonctionnalité de notre application et le lien dynamique entre notre base de donnée et notre interface nous avons choisi d'implémenter le design pattern MVC (Model View Controler). Ce pattern a l'avantage de faciliter grandement la modification de l'interface à chaque

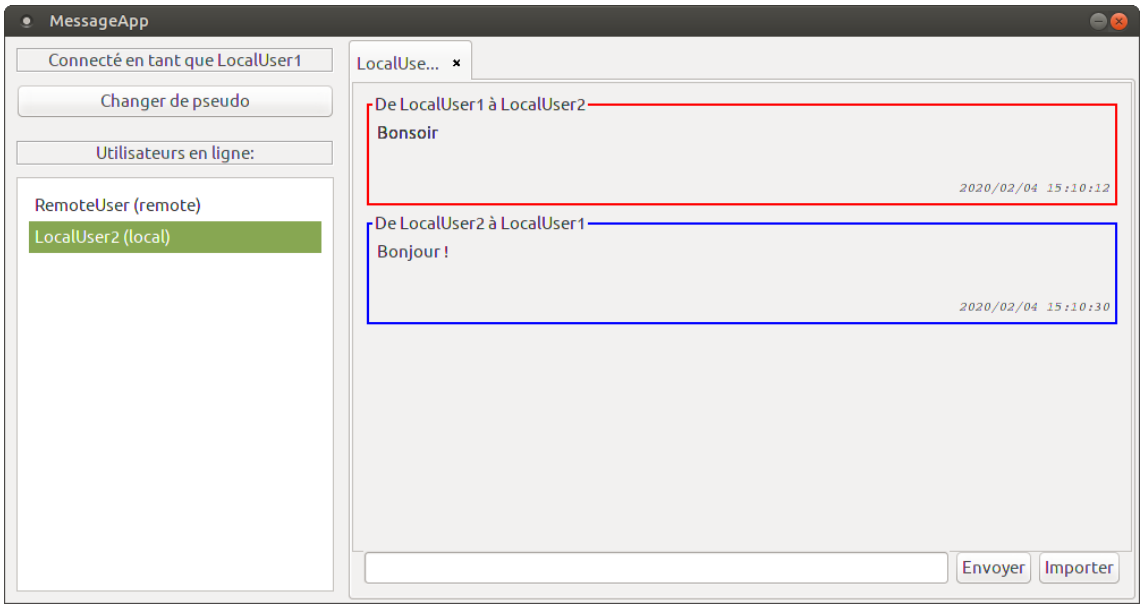

Figure 6: Session de discution avec historique

nouvelle entrée dans la base de donnée et la mise à jour de la base de donnée à chaque interaction de l'utilisateur avec l'interface, tout ça passant par un contrôleur qui fait office d'intermédiaire.

En Java, l'implémentation du MVC se fait grâce à 2 interfaces : Observable et Observer. Ainsi le Model (la classe gérant la base de donnée) implémente Observable et la View (les classes gérant l'interface) implémentent Observer. A la création des instances du Model et de la View, la View s'abonne en tant qu'Observer au Model. Ainsi, chaque changement dans le Model sera notifié à la View grâce à la méthode "notifyObserver(string str)" du Model. Ensuite en fonction du contenu de "str" qui sera vérifié par la méthode "update(string str)" de la View, celle pourra adopter un comportement différent et modifier l'interface en conséquence. Typiquement les changements possibles dans notre cas sont la connexion/déconnexion d'un client qui doit entraîner la mise à jour de la liste des cliens connectés, la réception / l'envoi d'un nouveau message qui, après avoir été ajouté à la base de donnée, doit apparaître sur l'historique des messages avec la source / le destinataire.

2.1.3 Choix de la gestion des protocoles de transport

Pour envoyer des messages et des fichiers entre 2 utilisateurs, nous utilisons TCP. Par souci de praticité et de rapidité d'implémentation, nous ne stockons pas les sockets associés à chaque conversation, ce qui n'est pas forcément une bonne pratique. A chaque envoie de donnée, on crée un socket, on l'utilise, puis on le fans ce sens, nous avons choisi d'identifier nos utilisateurs par l'adresse MAC de leur machine sur le réseau local. Nous avons privilégié l'adresse MAC par rapport à l'adresse IP du fait qu'un utilisateur se connectant sur un réseau Wifi où l'adressage est dynamique aurait une nouvelle adresse IP à chaque utilisation ce qui l'empêcherait de s'identifier. En revanche la limite de ce choix réside dans le fait que l'utilisateur doit toujours utiliser la même machine pour se connecter au système de messagerie. Nous avons cependant estimé que dans un contexte d'entreprise où chaque employé possède sa propre machine cette contrainte reste raisonnable et de plus aucune indication du cahier des charges ne va à l'encontre de ce choix. Pour la base de donnée, nous avons là aussi fait le choix d'une base de donnée locale, basée sur SQLite. Utiliser une base de donnée locale est cohérent avec nos choix au niveau de l'identification. En effet si un utilisateur utilise toujours la même machine pour se connecter, il n'est pas gênant d'avoir une base de donnée sous forme de fichier en local (dans notre cas directement dans le dossier racine de notre application). On pourrait même argumenter sur l'aspect sécurisant de la chose, un attaquant devrait attaquer chaque client pour récupérer l'intégralité des conversations de erme. Cette méthode nous a permi

de mieux avancer certaines autres fonctionnalités du logiciel. De plus, cette manière, bien que peu élégante, fonctionne.

2.1.4 Gestion de la découverte des utilisateurs sur le réseau local

Pour découvrir les utilisateurs connectés sur le réseau local, on utilise des messages système. Il en existe 4 sortes:

- lorsque l'on se connecte, avant de rentrer son pseudo, on envoie un message "Hello", permettant de récupérer la liste des utilisateurs connectés, et des pseudos utilisés.
- lorsque l'on rentre son pseudo et qu'il est validé, on envoie un message "Connect", indiquant notre pseudo à tout le monde.
- pour changer de pseudo, on envoie un message système "Change pseudo", permettant de notifier le changement à tous le monde.
- lors de la déconnexion, on envoie un message "disconnect".

Ces 4 types de messages sont envoyés en broadcast en UDP. Ils nous permettent de garder à jour la liste des utilisateurs connectés ainsi que leur pseudos, et de notifier tous les utilisateurs d'un quelconque changement.

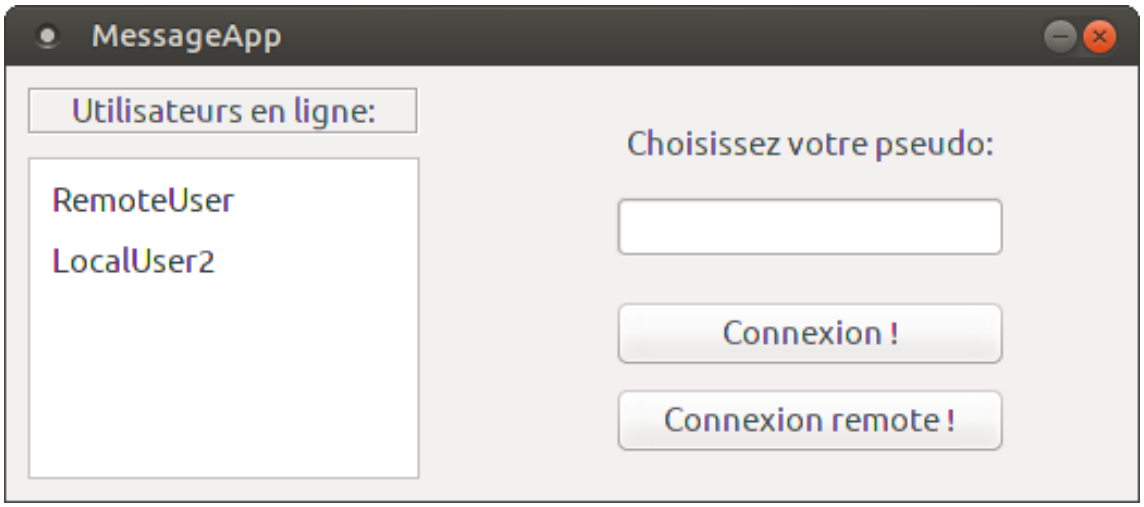

Figure 7: Ecran de connexion

2.1.5 Gestion de la découverte des utilisateurs hors du réseau local

La découverte des personnes hors du réseau local se fait par l'utilisation d'un servlet. Chaque client, lorsqu'il lance le l'application, envoie ses informations (adresse, mac, ip, pseudo) au servlet, et récupère en retour la liste de tous les utilisateurs connectés. Tous les 3 secondes, on renvoie une requête contenant nos informations au servlet pour qu'il actualise ses données, et on re-récupère la liste des utilisateurs.

Deux cas se présentent ici:

- si on est sur le réseau local de l'entreprise, le broadcast de message système sert déjà à récupérer les utilisateurs locaux. Ainsi, une partie des données du servlet ne doit pas être traiter, sinon il y aurait des doublons. Les changements de pseudo sont immédiatement visible pour les utilisateurs locaux, car cela se passe avec un broadcast.

- si on est en dehors du réseau, l'actualisation de nos données se fait toutes les 3 secondes.

Le servlet, lors de son initialisation, crée une liste vide. Chaque utilisateur qui se connecte est alors rentré dans la liste. A chaque requête, on envoie toutes la liste des utilisateurs. Un utilisateur qui se déconnecte envoie une requête permettant de l'enlever de la liste du servlet.

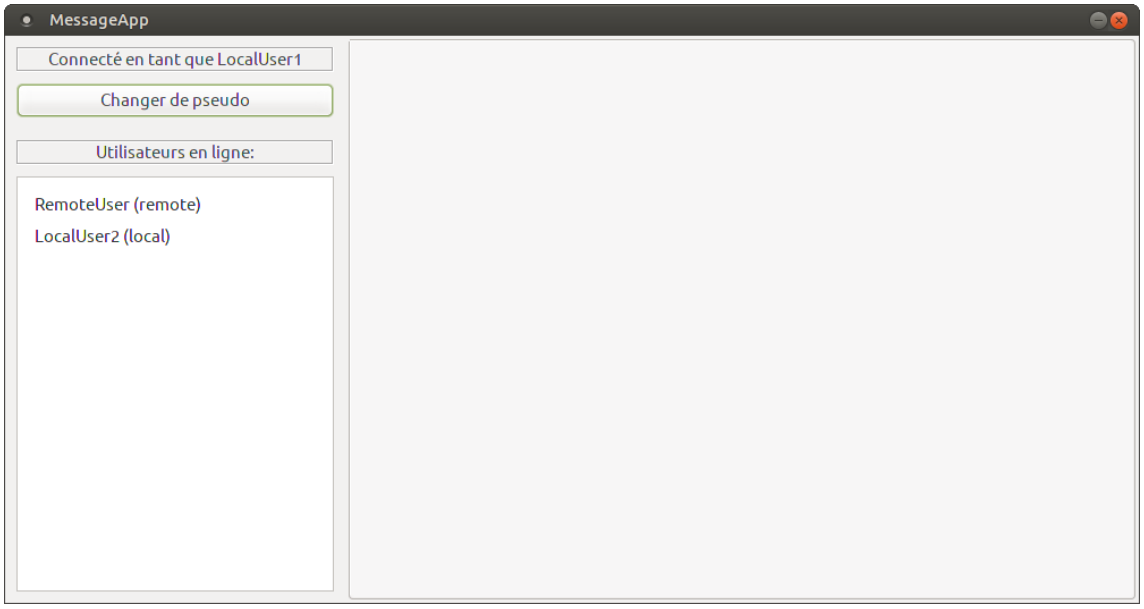

Figure 8: Découverte des utilisateurs

2.1.6 Gestion de l'envoi d'images et de fichiers

Concernant le partage d'images et de fichiers, la taille de ces objets en mémoire étant trop importante pour permettre leur stockage dans notre base de données SQLite. Ainsi nous avons décidé de stocker en "cache" toutes les images et les fichiers partagés par et avec le client dans un dossier à part "/tmp". Cet artifice permet d'assurer le fait que ce type de données soient toujours accessibles après s'être déconnecté et avoir relancé l'application.

Au niveau de l'affichage de l'historique, les images sont affichées telles quelles et un bouton permet de télécharger les fichiers à l'endroit de notre choix (ce qui revient donc dans notre cas à faire une copie du fichier en mémoire dans le dossier "/tmp").

2.2 Manuel d'utilisation

2.2.1 Lancer l'application

Pour lancer l'application de messagerie, il suffit d'exécuter le script bash MessageApp.sh (soit en lui donnant des droits d'exécution soit en se plaçant dans le dossier et en lançant la commande :

./MessageApp.sh

2.2.2 Se connecter

Au lancement de l'application, une première fenêtre s'ouvre. L'utilisateur est invité à entrer un pseudo et à appuyer sur un des deux boutons en fonction du réseau sur lequel il se trouve actuellement ("Connexion" s'il est sur le réseau local de l'INSA ou "Connexion remote" dans tous les autres cas) ou presser la touche ENTER du clavier (qui correspond à une connexion en local) pour se connecter avec ce pseudo.

Les contraintes sur le choix du pseudo sont :

- pas de pseudo vide
- pas de pseudo avec plus de 16 caractères
- pas de pseudo identique à l'un des pseudos déjà utilisé par les autres utilisateurs connectés

Si l'utilisateur ne respecte pas une de ces contraintes, une fenêtre de dialogue apparaît alors et lui indique son erreur. Une liste des pseudos des autres utilisateurs connectés se trouve sur la gauche de la fenêtre afin de faciliter le choix.

2.2.3 Ouvrir/Fermer une session de clavardage

Une fois le choix du pseudo fait et la connexion établie, une nouvelle fenêtre apparaît. Sur la gauche se trouve une liste des utilisateurs connectés.

Pour ouvrir une session de clavardage avec un utilisateur il suffit d'appuyer sur le pseudo de cet utilisateur dans cette même liste. Un onglet s'ouvrira alors sur la partie droite de la fenêtre, affichant l'historique de la conversation avec cet utilisateur et un champ pour lui envoyer des messages.

Le fonctionnement des onglets est assez intuitif, on peut changer d'onglet en cliquant sur un autre, on peut fermer un onglet en cliquant sur la petite croix à côté du pseudo de l'utilisateur au niveau du titre de l'onglet... On peut également changer d'onglet en cliquant sur un autre utilisateur dans la liste des utilisateurs connectés.

2.2.4 Envoyer un message

Pour envoyer un message il suffit de remplir le champ destiné à cet effet dans l'onglet correspondant au destinataire souhaité. Il faut alors appuyer sur le bouton "Envoyer" ou tout simplement presser la touche Enter du clavier.

Si on cherche à envoyer un message vide, une petite fenêtre de dialogue s'ouvrira, avertissant l'utilisateur et lui demandant de confirmer ou d'annuler son action. A noter également que des messages contenant des apostrophes et/ou guillemets passent sans problème grâce à notre utilisation de requêtes préparées au niveau de la base de données.

Au niveau de l'affichage d'un message, la couleur du bloc qui encadre le message diffère en fonction de si nous sommes la source ou le destinataire de celui-ci. Il y a également un petit système de notification : lorsqu'on reçoit un message un astérisque apparaît à côté du pseudo au niveau du titre de l'onglet et si aucun onglet ne correspond à l'expéditeur celui-ci est ouvert.

2.2.5 Envoyer une image ou un fichier

Si l'on souhaite envoyer un image ou un fichier, le processus est sensiblement le même. Il suffit d'appuyer sur le bouton "Importer" en bas à droite de la fenêtre. Une fenêtre de sélection apparaît alors, nous permettant de parcourir notre PC à la recherche du fichier que l'on souhaite expédier.

Au niveau de l'affichage, une image apparaîtra de la même manière qu'un message mais pour un fichier, un bouton apparaît et offre à l'utilisateur de l'enregistrer à l'endroit de son choix s'il clique dessus.

2.2.6 Changer de pseudo

Pour changer de pseudo, il suffit de cliquer sur le bouton en haut à gauche de la fenêtre. On revient alors à la première fenêtre d'identification et l'on peut choisir à nouveau son pseudo. A noter que l'on est toujours connecté à ce moment là et donc que les autres utilisateurs peuvent

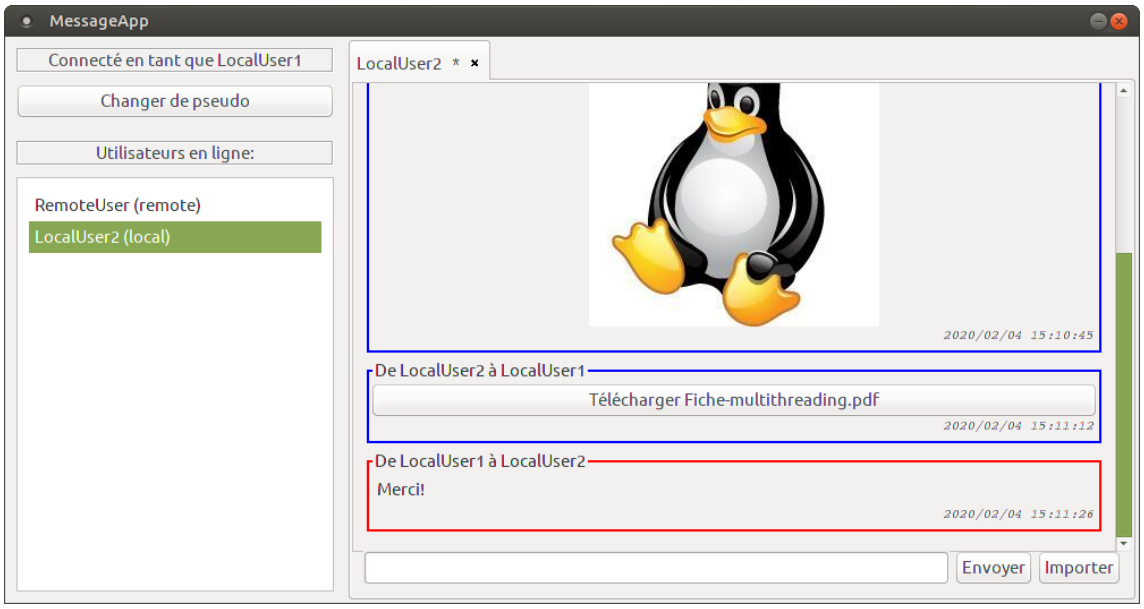

Figure 9: Historique après envoi d'une image et d'un fichier

toujours nous envoyer des messages. De plus, lors du changement de pseudo, la liste des utilisateurs connectés et les éventuels onglets concernant l'utilisateur ayant changé de pseudo seront modifiés en conséquence.

2.2.7 Se déconnecter

Pour se déconnecter il suffit d'appuyer sur la croix en haut à droite de la fenêtre.

2.3 Procédures de test et de validation

Le respect des exigences du cahier des charges est vérifiable lors d'une utilisation basique de notre application (se connecter, changer de pseudo, envoyer des messages...) à quelques exceptions près qui requièrent un peu plus d'explications :

- **[CdC-Bs-22]** Le système doit permettre la mise en place de 1000 sessions de clavardage simultanées au sein de celui-ci => nos sessions de clavardage ne nécessitant pas l'établissement d'une connexion constante quelconque, cette exigence est très facilement respectée
- **[CdC-Bs-23]** L'agent doit permettre la mise en place de 50 sessions de clavardage simultanées => idem
- **[CdC-Bs-26]** Le système doit permettre un usage simultané par au minimum 100 000 utilisateurs => Difficilement vérifiable à notre échelle, nous avons pu essayer notre application avec une centaine de "fausses" personnes connectées mais nous n'avons pas eu l'occasion d'essayer notre système avec 100 000 "vrais" utilisateurs connectées à la fois.
- **[CdC-Bs-30]** Le système doit garantir une intégrité des messages supérieure à 99,999% => Comme nous utilisons TCP pour l'envoi et la réception des messages, nous bénéficions de la garantie de l'intégrité de nos messages.