

# ***RAPPORT DE BUREAU D'ETUDES COO/POO***

SYSTEME DE CLAVARDAGE DISTRIBUÉ INTERACTIF  
MULTI-UTILISATEUR TEMPS REEL

---

BENAZZOUZ Abir  
BONNEAU Clémentine  
HOK Jean-Rémy

Année 2020-2021  
Date de rendu du rapport : 15 février 2021

INSA Toulouse  
Promotion 55  
4<sup>ème</sup> année IR-SI-A1

## SOMMAIRE

<b>I – PRÉSENTATION GÉNÉRALE DU PROJET .....</b>	<b>3</b>
<b>II – NOTRE MODELISATION UML .....</b>	<b>4</b>
1) DIAGRAMME DE CAS D’UTILISATION.....	4
2) DIAGRAMMES DE SEQUENCE.....	4
3) DIAGRAMME DE CLASSES .....	8
<b>III – NOS CHOIX TECHNOLOGIQUES ET CONCEPTUELS .....</b>	<b>12</b>
1) SEPARATION EN PACKAGES .....	12
2) PROPERTIESREADER : CONFIGURATION SIMPLE ET RAPIDE.....	13
3) L’INTERFACE GRAPHIQUE HOMME-MACHINE.....	13
4) L’AUTHENTIFICATION ET LA GESTION DES LOGIN .....	14
5) LA COMMUNICATION RESEAU .....	14
6) LA DECOUVERTE DES AUTRES UTILISATEURS.....	16
7) BASE DE DONNEES : GESTION DE L’HISTORIQUE DES MESSAGES ET STOCKAGE DES UTILISATEURS ET DE LEURS INFORMATIONS .....	16
8) DECONNEXION ET RECONNEXION.....	17
9) LOGS.....	17
10) SERVLET (SERVEUR DE PRESENCE).....	17
11) DETECTION D’UNE AUTRE INSTANCE DE CLAVARDAGE.....	18
12) GESTION DES FICHIERS ET IMAGES.....	18
<b>IV – PROCEDURES DE TESTS ET DE VALIDATION .....</b>	<b>19</b>
1) UTILISATION DE DOCKERS .....	19
2) PREMIERS TESTS AVEC INTERFACE « CONSOLE » .....	19
3) IMPLEMENTATION DE L’INTERFACE GRAPHIQUE.....	19
4) VIDEOS DE DEMONSTRATION .....	21
<b>V – MANUEL D’UTILISATION DE NOTRE PRODUIT .....</b>	<b>22</b>
1) TELECHARGER L’APPLICATION .....	22
2) CONFIGURER L’APPLICATION .....	22
3) CONFIGURER ET INSTALLER LE SERVEUR DE PRESENCE SUR LE SERVEUR TOMCAT .....	22
4) UTILISER LES DIFFERENTES FONCTIONNALITES .....	23

## I – Présentation générale du projet

Ce projet vise à mettre en place un système de clavardage distribué interactif, permettant à plusieurs utilisateurs de communiquer en temps réel.

Il s'est découpé en plusieurs étapes. En premier lieu, nous avons effectué la conception de notre système, en le modélisant par des diagrammes UML (diagramme de cas d'utilisation, diagrammes de séquence, diagramme de classes, diagramme de structure composite et diagramme de machine à état).

Par la suite, nous avons implémenté notre système de clavardage, en répondant aux attentes du client fournies dans le cahier des charges. Les plus conséquentes étant que chaque utilisateur doit être identifiable via un pseudonyme personnel et unique. Chaque utilisateur doit de plus avoir accès à l'ensemble des utilisateurs connectés au système de clavardage sur réseau local, ainsi que sur réseau externe. Chaque utilisateur doit être à même d'engager une session de clavardage avec un autre utilisateur en ligne, et de voir l'historique de toutes ses précédents messages. Enfin, les messages envoyés peuvent être du texte, mais également des images ou des fichiers.

Les outils utilisés afin de mener à bien sa réalisation sont le langage de programmation java, associé à l'environnement de développement eclipse. Pour la partie interface, nous avons utilisé la bibliothèque graphique swing. Le code final se trouve sur un dépôt github à l'adresse suivante : <https://github.com/PiKouri/4a-projet-oo>.

## II – Notre modélisation UML

Tous nos diagrammes sont accessibles sur notre git à l'adresse suivante : <https://github.com/PiKouri/4a-projet-oo/tree/main/img/Nouveaux%20Diagrammes>.

### 1) Diagramme de cas d'utilisation

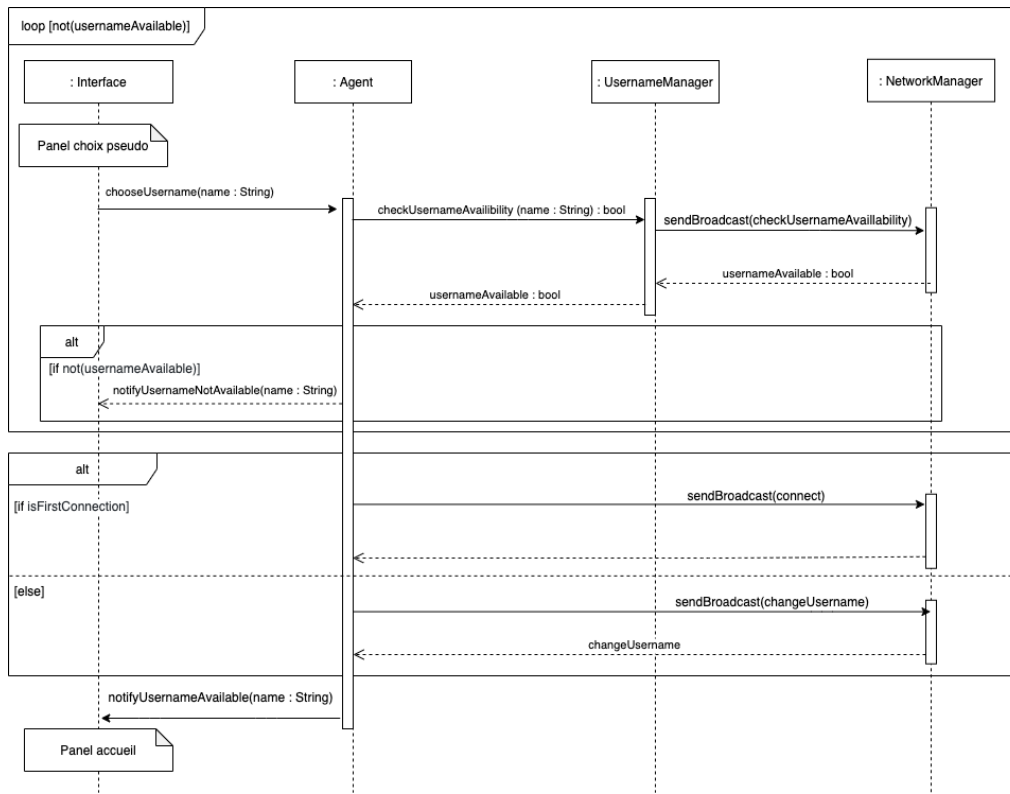


Diagramme de cas d'utilisation

### 2) Diagrammes de séquence

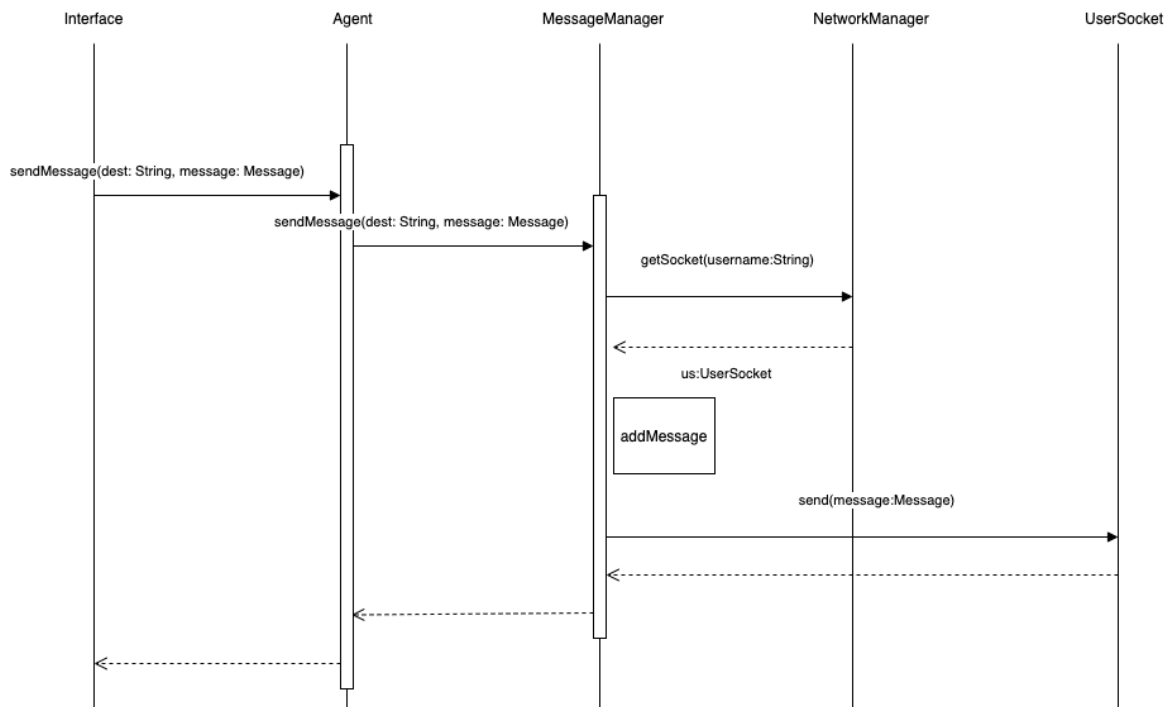
Les diagrammes ont été faits pour les communications dans le réseau interne, mais les utilisateurs externes utilisent environ les mêmes étapes à la différence qu'ils envoient les notifications (connect, checkUsernameAvailability, etc.) au serveur de présence via requête POST.

a. chooseUsername



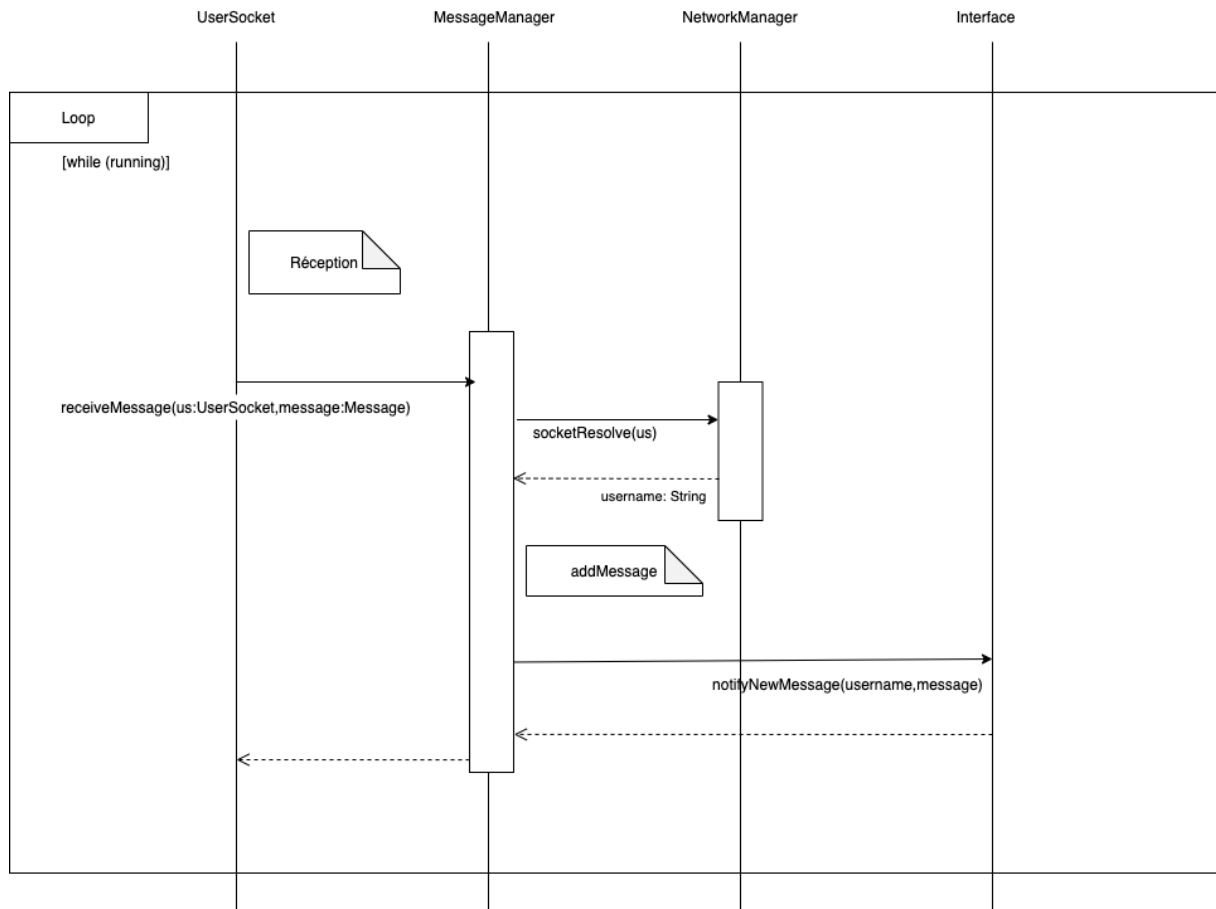
*Diagramme de séquence pour choisir son username*

b. sendMessage



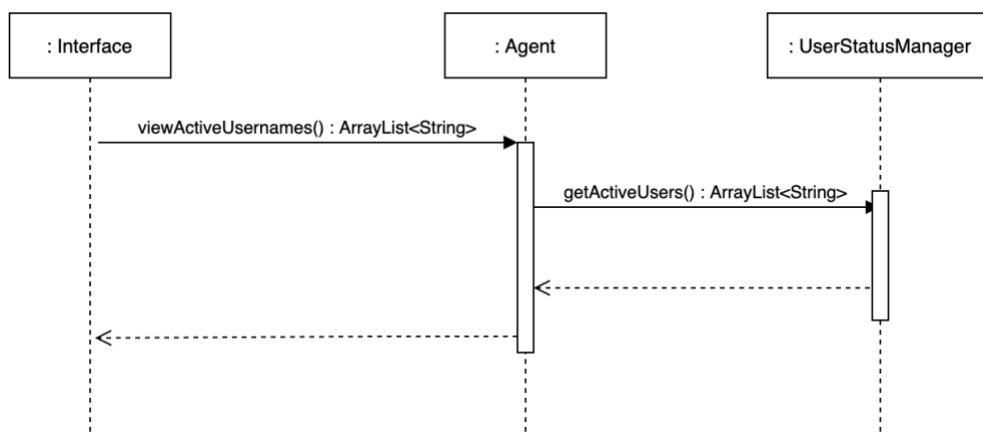
*Diagramme de séquence d'envoi de messages*

### c. receiveMessage



*Diagramme de séquence de la réception de messages*

### d. viewActiveUsernames



*Diagramme de séquence pour voir les utilisateurs actifs*

Le fonctionnement est identique pour voir les utilisateurs déconnectés.

e. getMessageHistory

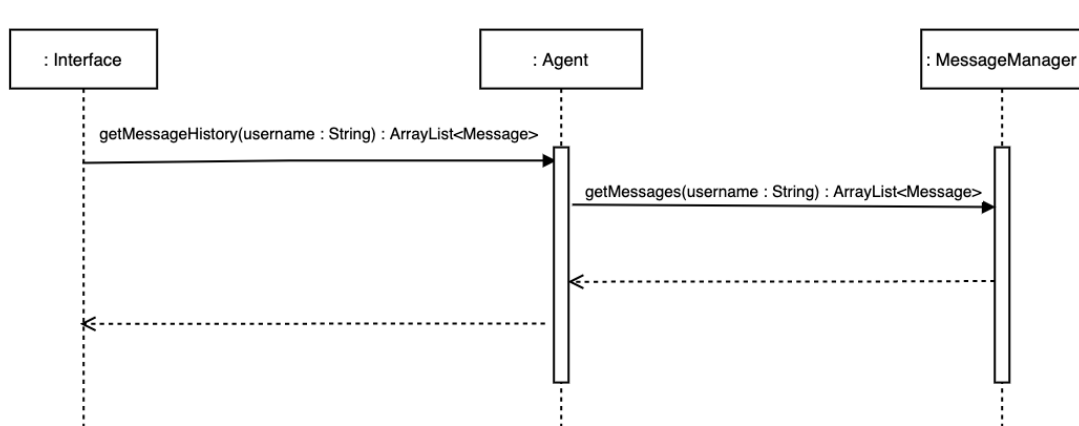


Diagramme de séquence pour voir l'historique des messages

f. disconnect

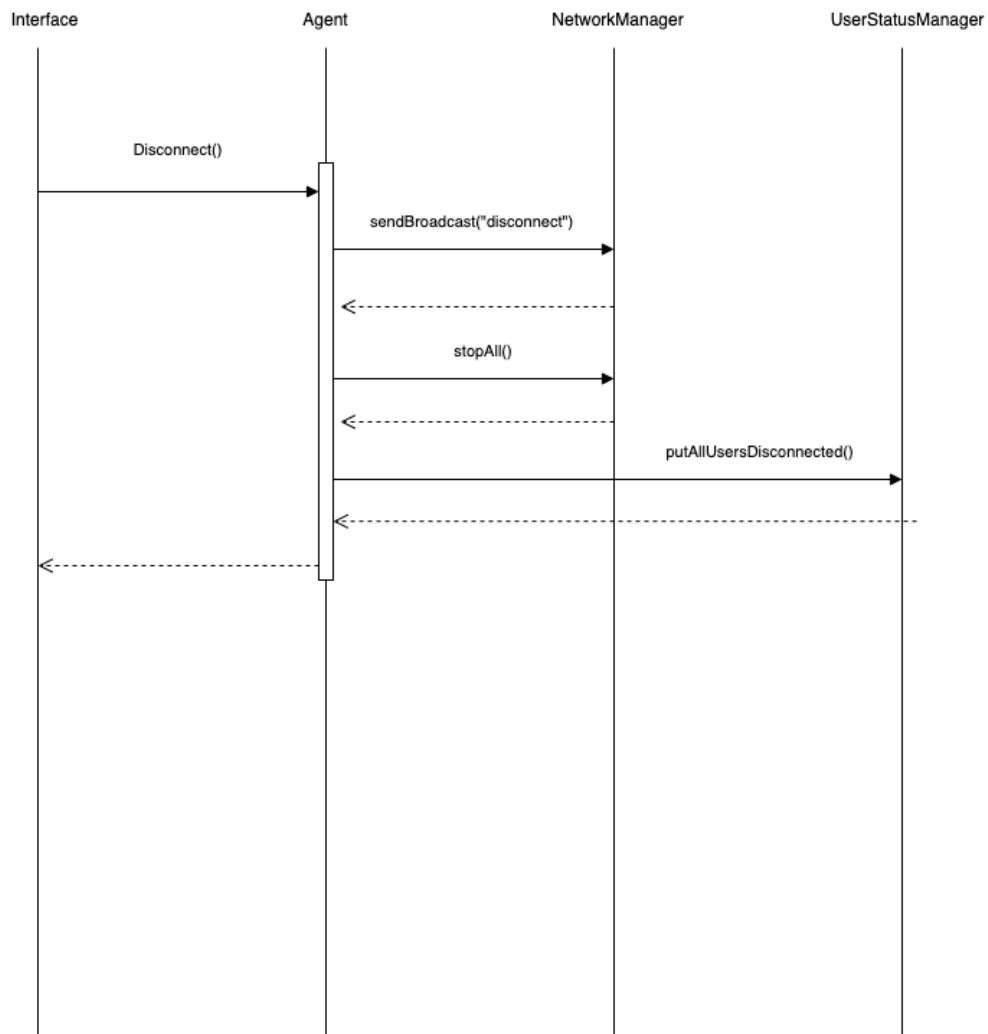
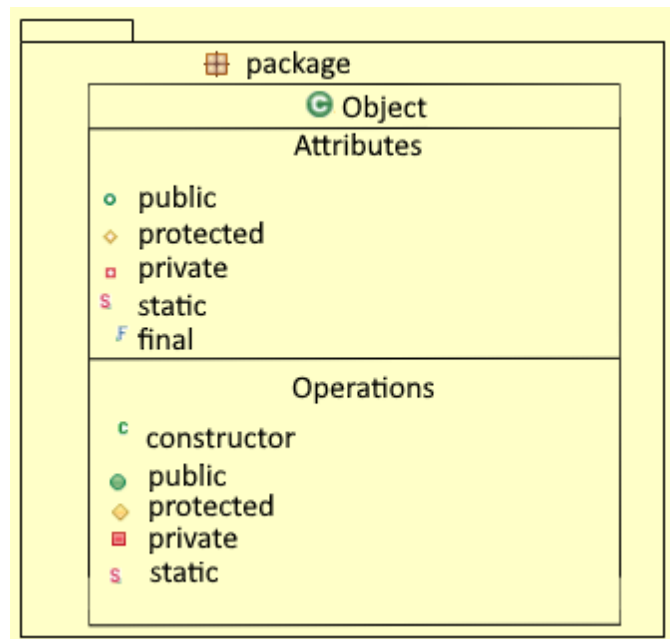


Diagramme de séquence de la déconnection

### 3) Diagramme de classes

#### a. Légende des diagrammes suivants



Légende des diagrammes de classe

#### b. Package datatypes

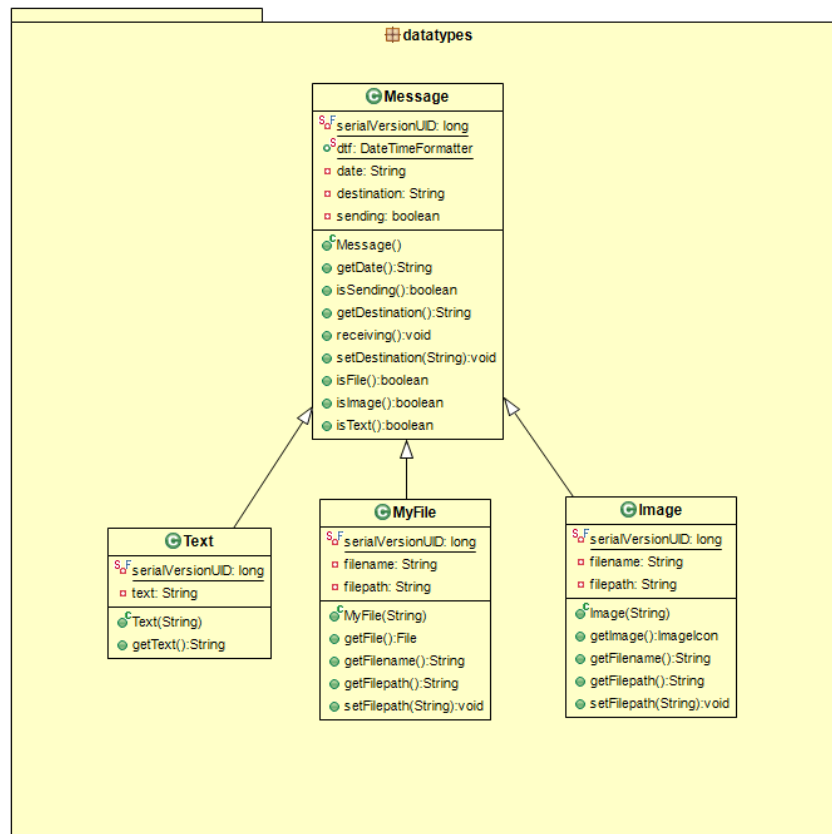
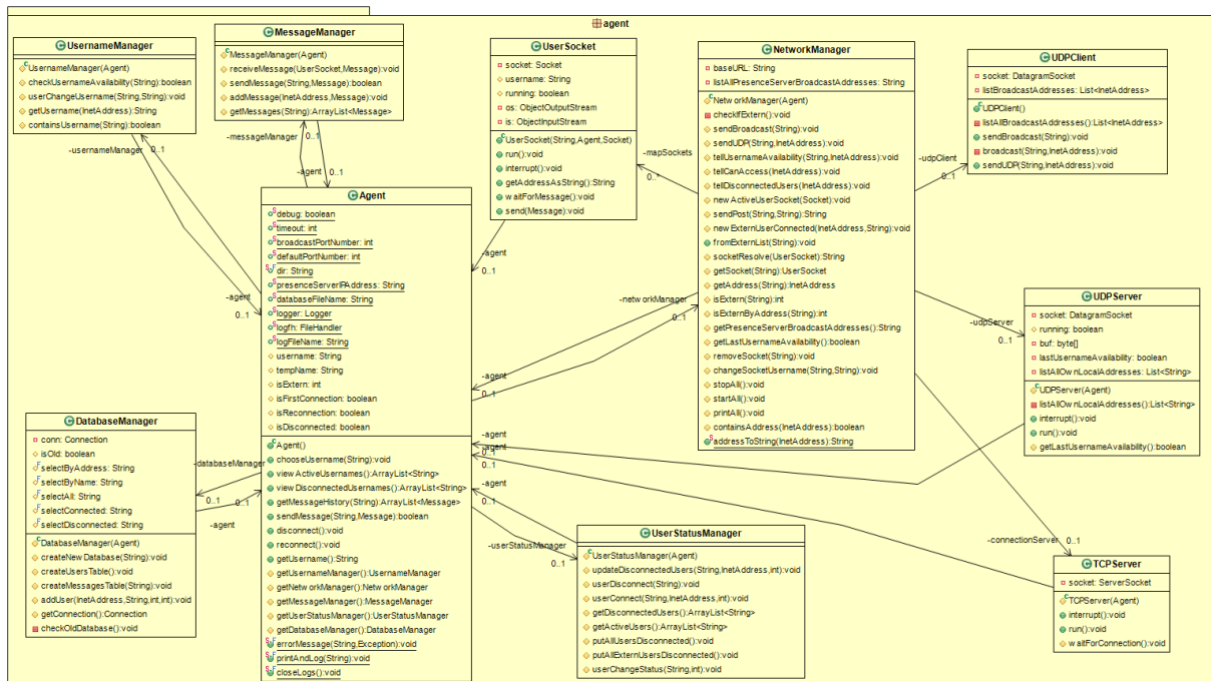


Diagramme de classe du package datatypes

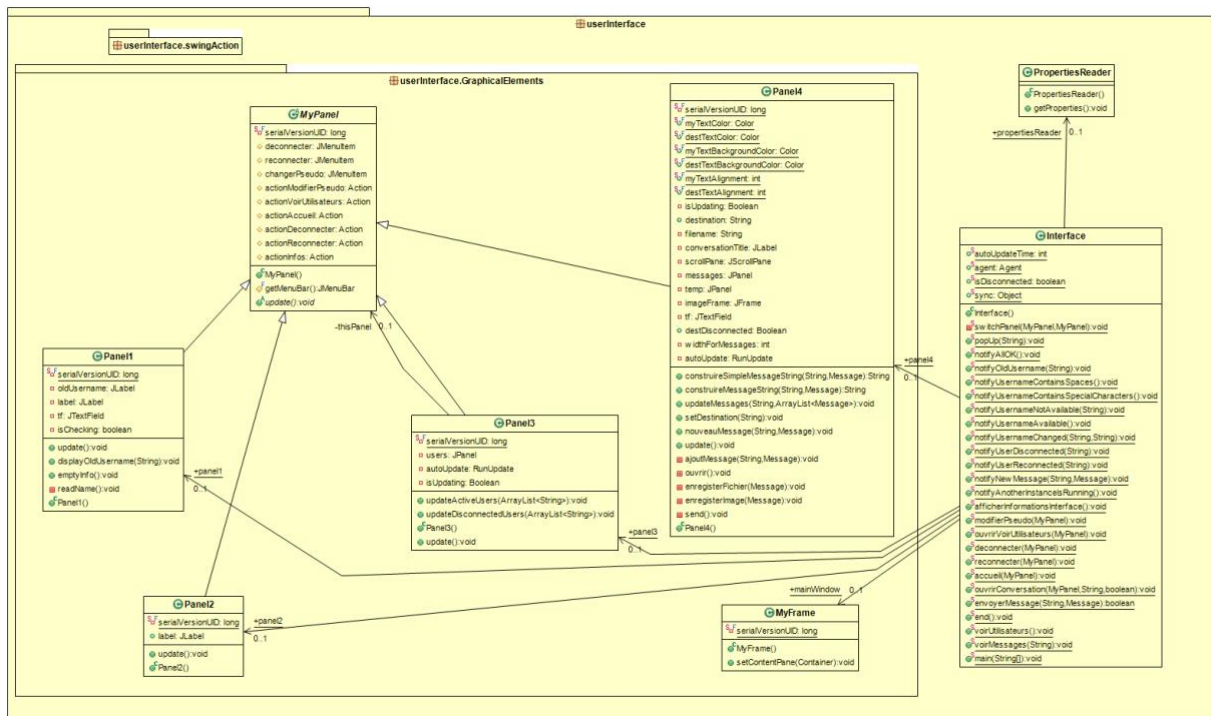


### c. Package agent



*Diagramme de classe du package agent*

### d. Package userInterface



*Diagramme de classe du package userInterface*

## e. Résumé clavardage

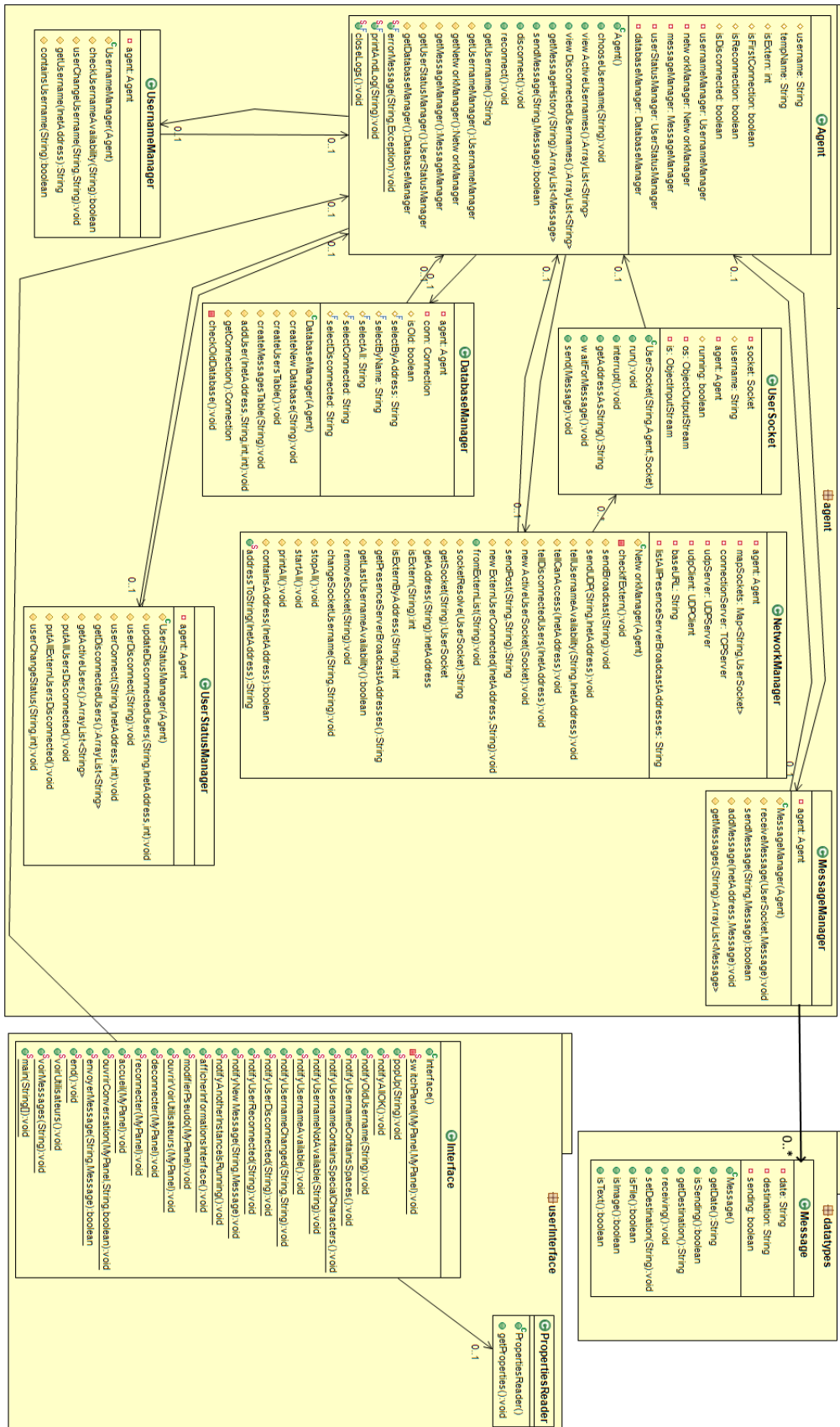
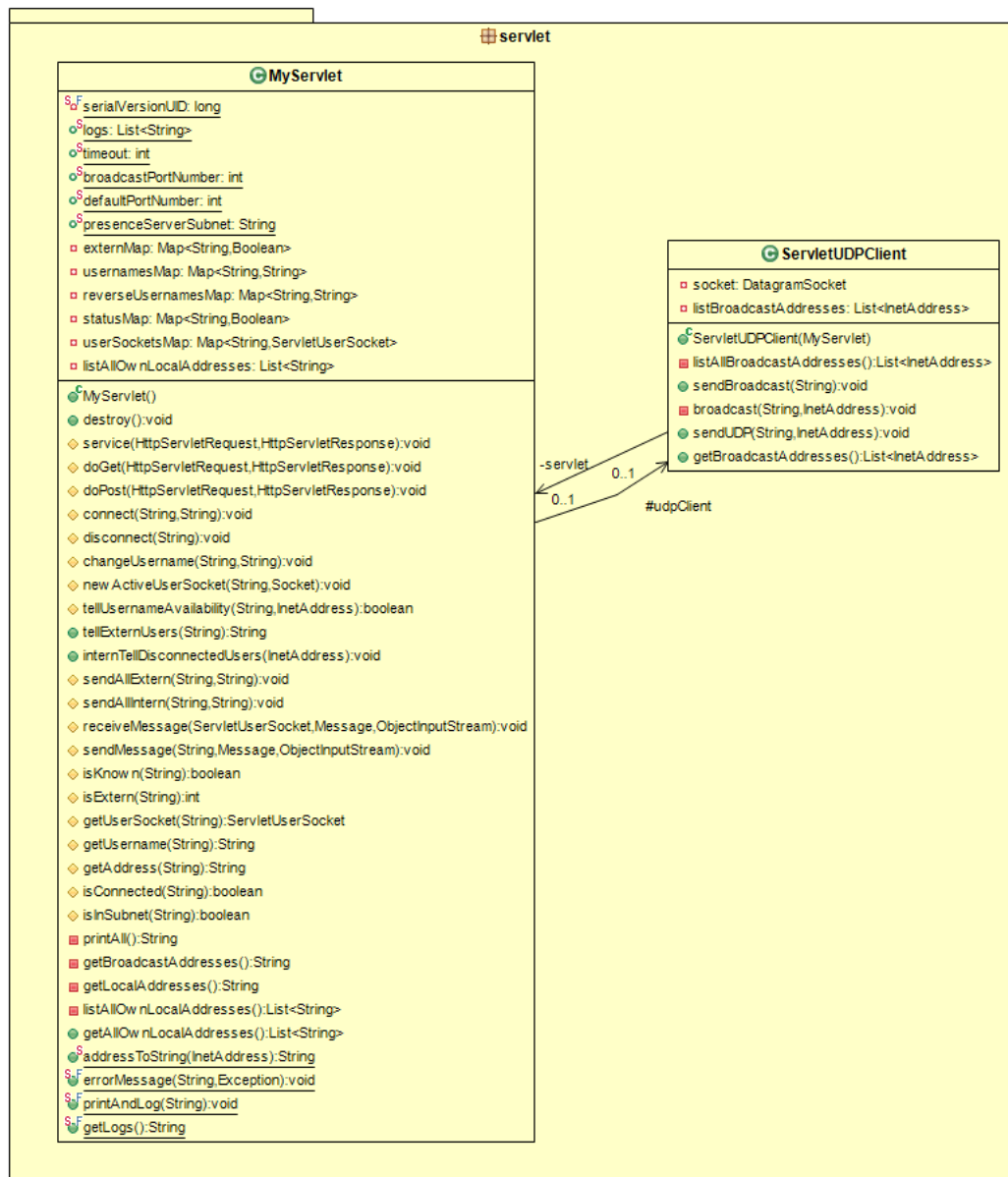


Diagramme de classe du système de clavardage

f. Servlet



*Diagramme de classe du servlet*

### III – Nos choix technologiques et conceptuels

#### 1) Séparation en packages

Pour plus de lisibilité du projet, nous avons choisi de séparer les différentes classes dans des paquetages (package) spécifiques :

- **agent** : paquetage regroupant les différents « gestionnaires / managers » et classes qui gèrent les principales fonctionnalités de l'agent, à savoir *chooseUsername*, *sendMessage*, *receiveMessage*, *viewActiveUsernames*, *getMessageHistory*, *disconnect*, dont la classe principale est **Agent**.
- **datatypes** : paquetage regroupant les classes « types » que l'on a utilisé pour les messages dont notamment la classe principale **Message** dont hérite *MyFile*, *Text* et *Image*.
- **userInterface** : paquetage regroupant les classes utilisées pour l'interface graphique Homme-Machine dont la classe principale est **Interface**.

##### a. Le package agent

Dans le package agent, la classe principale **Agent** représente le cœur de l'application comme précisé précédemment. Les autres classes « gestionnaires / managers » orbitent autour de cette dernière :

- Le **UsernameManager** gère les pseudos : la disponibilité des pseudos, les changements de pseudo et la récupération du pseudo associé à une adresse IP.
- Le **UserStatusManager** gère l'état des utilisateurs : la connexion, et la déconnexion des utilisateurs, et la gestion des listes des utilisateurs connectés/déconnectés.
- Le **MessageManager** gère les messages : la réception et l'envoi des messages et la gestion de l'historique des messages.
- Le **DatabaseManager** gère la base de données stockée localement : la création de la base de données si elle n'existe pas déjà, la création de la table des informations sur les utilisateurs et les tables des messages pour chaque utilisateur.
- Le **NetworkManager** gère les communications réseaux de l'application : il utilise la classe *UDPCClient* pour envoyer des messages UDP, gère les messages UDP entrants avec la classe *UDPServer* et les connexions TCP entrantes avec la classe *TCPServer*.

Enfin la classe **UserSocket** permet l'envoi et la réception des messages avec un utilisateur donné. A noter que les classes *UserSocket*, *TCPServer* et *UDPServer* sont des *Thread*.

##### b. Le package datatypes

Dans le package datatypes, la classe principale **Message** modélise, comme son nom l'indique, les messages : on y stocke la *Date* d'envoi et le destinataire. Cette classe implémente également les fonctions *isFile()*, *isImage()* et *isText()* permettant de reconnaître les classes filles utilisées.

La classe **Text** ajoute un attribut *text* : le texte du message.

La classe **MyFile** ajoute les attributs *filename* et *filepath* représentant respectivement le nom et le chemin du fichier.

Enfin la classe **Image** est similaire à la classe *MyFile* mais retourne une *ImageIcon* au lieu d'un *File*.

### c. Le package `userInterface`

Dans le package `userInterface`, la classe principale ***Interface***, contenant le `main()` de l'application, modélise l'interface graphique. Ce package contient également les classes ***MyFrame*** représentant la fenêtre principale de l'application et ***MyPanel*** qui montre un onglet type de l'application (dont hérite les classes *Panel1*, *Panel2*, *Panel3*, et *Panel4*). Enfin la classe ***PropertiesReader*** permet de lire le fichier `config.properties` et de configurer l'application.

#### 2) `PropertiesReader` : configuration simple et rapide

La classe ***PropertiesReader*** lit le fichier `config.properties` grâce à `java.util.Properties`, ainsi nous récupérons la valeur des différentes propriétés que l'utilisateur a pu modifier pour ensuite les assigner à l'**Agent**. Voir le manuel utilisateur section Configuration de l'application pour plus de précisions à propos de chaque propriété.

#### 3) L'interface graphique Homme-Machine

Pour l'interface graphique, nous avons principalement utilisé Java SWING. Nous utilisons une `JFrame` (plus précisément une `MyFrame`) dans laquelle nous changeons le `JPanel` (plus précisément `MyPanel`) affiché : ce qui permet des changements d'onglets rapides et sans mouvement de la fenêtre principale.

Des images de l'application sont disponibles dans le manuel utilisateur dans la section Utilisation des boutons.

Le *Panel1* est l'onglet « Choix de pseudo », premier onglet ouvert qui demande à l'utilisateur d'entrer un pseudo. Le *Panel2*, quant à lui, est l'Accueil, avec un message de bienvenue. Le *Panel3*, affiche la liste des utilisateurs connectés ((+) `connectedUser`) et la liste des utilisateurs déconnectés ((-) `disconnectedUser`). A noter que cet onglet implémente un `Thread` pour afficher la liste des utilisateurs, sans encombre pour l'utilisateur. Sur cet onglet, l'utilisateur peut choisir d'ouvrir la discussion avec un utilisateur connecté, cela amène à l'onglet *Panel4*. Ce dernier affiche l'historique des messages échangés avec l'utilisateur donné : en bleu sont affichés les messages envoyés, tandis que les messages reçus sont affichés en gris. Sur cet onglet, l'utilisateur peut choisir d'écrire du texte et de l'envoyer ou d'ouvrir un explorateur de fichier pour sélectionner un fichier ou une image à envoyer. A noter que cet onglet implémente également un `Thread` pour afficher l'historique des messages et pour l'ajout de nouveaux messages reçus/envoyés, sans encombre pour l'utilisateur. L'envoi de messages se fait par clic sur le bouton *Envoyer* ou par appui sur la touche clavier *Entrer*.

Il est important de noter que l'utilisateur peut changer de pseudo ou se déconnecter ou afficher les utilisateurs connectés via la barre de menu en haut de la fenêtre.

L'utilisateur reçoit également un pop-up lui notifiant la réception d'un message ou du changement du pseudo d'un utilisateur s'il n'est pas sur la conversation en question.

Enfin dans la classe ***Interface***, des méthodes `notify()` permettent à l'Agent et aux différents `MyPanel` de notifier l'interface graphique de divers changements. Typiquement : le pseudo est déjà pris, le pseudo est disponible, un utilisateur a changé de pseudo, un utilisateur s'est connecté/déconnecté, un message a été reçu/envoyé.



#### 4) L'authentification et la gestion des login

##### a. Côté application

Dans cette partie les communications broadcast UDP sont utilisées exclusivement pour les indoor users tandis que les requêtes POST sont utilisées par les outdoor users et les indoor users lorsque le serveur de présence est accessible.

Le cahier des charges spécifie que chaque utilisateur doit être identifiable via un pseudonyme unique et personnel. Ainsi, lorsqu'un nouvel utilisateur se connecte au système, il doit choisir son pseudo, et il doit y avoir une vérification du fait que ce dernier ne soit pas déjà utilisé. Pour ce faire, dès qu'il y a une nouvelle connexion, le nouvel utilisateur envoie automatiquement une requête en broadcast UDP (ou une requête POST au serveur de présence pour un utilisateur externe). Ainsi, tous les utilisateurs déjà présents et le serveur de présence reçoivent le pseudo, et indiquent au nouveau connecté s'ils valident le pseudo, ou bien si ce dernier leur est déjà attribué, évitant ainsi les doublons.

Le premier utilisateur ne reçoit aucun message de réponse à sa requête, et sait ainsi qu'il n'y a pas encore d'autres personnes connectées et que son pseudo est donc libre d'accès. Après le choix du pseudo, l'Agent envoie un message en broadcast UDP (ou une requête POST au serveur de présence pour un utilisateur externe) afin de demander aux autres utilisateurs de se connecter à son serveur TCP. A la réception de sa requête UDP, les autres utilisateurs lui envoient leur adresse IP (envoi via UDP pour récupérer l'adresse faisant partie du sous-réseau interne) et leur pseudo correspondant. Dans le cas de la requête POST, le serveur de présence répond avec la liste des utilisateurs et leurs informations (pseudo, adresse IP, statut, et externe ou non). Enfin, s'il s'agit d'une reconnexion suivant une déconnexion, les autres utilisateurs sont informés -si le pseudo rentré est nouveau- de cette modification.

Lors d'un changement de pseudo, nous utilisons directement notre table des pseudos locale pour vérifier la disponibilité du pseudo.

##### b. Côté servlet (serveur de presence)

Dans le cadre de l'utilisation du serveur de présence, lorsque ce dernier reçoit une requête POST demandant si l'utilisateur concerné est interne ou externe, il vérifie dans sa table et répond à cette requête avec un 1 pour utilisateur externe et un 0 pour utilisateur interne.

S'il reçoit une demande de la disponibilité d'un pseudo, il vérifie dans sa table et répond à cette requête avec *true* lorsque le pseudo est disponible ou *false* sinon.

Enfin s'il reçoit une notification de connexion/de déconnexion, il transmet ce message aux autres utilisateurs : si la source est interne, il le transfère seulement aux utilisateurs externes, tandis que si la source est externe, il le transfère à tout le monde (externes + internes).

#### 5) La communication réseau

##### a. Protocole UDP

Les communications UDP sont utilisées pour l'établissement des connexions et la notification de changement de pseudo ou de déconnexion. L'Agent dispose d'un *UDPClient* qui envoie les messages UDP et un *UDPServer* à l'écoute des messages à sa destination (broadcast ou dont l'adresse de destination appartient à sa liste d'adresses locales : *listAllOwnLocalAddresses*).

Message type	Utilité
<i>connect username isExtern</i>	Pour notifier notre arrivée et que les autres utilisateurs se connectent
<i>disconnect username</i>	Pour notifier notre déconnexion
<i>changeUsername oldUsername newUsername</i>	Pour notifier le changement de notre pseudo
<i>checkUsernameAvailability username</i>	Requête demandant la disponibilité d'un pseudo
<i>tellUsernameAvailability username true/false</i>	Réponse notifiant la disponibilité (ou non) du pseudo
<i>canAccess username</i>	Après avoir reçu un « connect », on renvoie un « canAccess » pour notifier notre adresse IP (dans le réseau interne) au nouvel arrivant
<i>updateDisconnectedUsers username address isExtern</i>	Après avoir reçu un « connect », on envoie notre liste des utilisateurs déconnectés au nouvel arrivant

### b. Protocole TCP

Les communications TCP sont utilisées pour l'échange de messages entre les utilisateurs. L'Agent dispose d'un *TCPServer* à l'écoute des connexions entrantes sur le port *defaultPortNumber* (voir manuel d'utilisation partie configuration de l'application). Lorsqu'une connexion arrive, le *TCPServer* envoie le socket associé au *NetworkManager* qui va créer une association (*Map*) entre ce socket, ou plus précisément le *UserSocket* créé à partir de ce socket, et le nom de l'utilisateur (reconnu grâce à son adresse IP).

Nous avons choisi de créer une classe *UserSocket* (qui étend la classe *Thread*) qui est associée à un socket (donc à la connexion avec un utilisateur) et qui permet la réception et l'envoi de *Message* à cet utilisateur. Ce *Thread* utilise *waitForMessage()* pour vérifier continuellement l'arrivée d'un nouveau message, auquel cas il est envoyé à l'Agent et plus précisément au *MessageManager*. L'Agent maintient une connexion par utilisateur pour faciliter et la réception et l'envoi des messages.

### c. Interaction avec le serveur de présence : requêtes POST

Lorsque l'utilisateur lance l'application, l'Agent va faire une requête POST au serveur de présence pour savoir s'il est accessible et pour savoir si l'utilisateur est interne ou externe au sous-réseau du serveur de présence (192.168.1.0/24, voir manuel d'utilisation partie configuration du serveur de présence). Si le serveur de présence n'est pas accessible, l'utilisateur est considéré comme interne et l'Agent n'utilisera que la communication broadcast UDP.

Les utilisateurs externes utilisent exclusivement les requêtes POST (vers le serveur de présence) pour communiquer.

Les attributs d'une requête POST type	Utilité
action=checkIfExtern	Demande si l'utilisateur est externe (n'appartient pas au sous-réseau du serveur de présence) ou interne
action=connect & username=username	Demande de notifier aux autres utilisateurs notre connexion
action=disconnect & username=username	Demande de notifier aux autres utilisateurs notre déconnexion
action=changeUsername & username=oldUsername & 3rdArgument=newUsername	Demande de notifier aux autres utilisateurs un changement de notre pseudo

## 6) La découverte des autres utilisateurs

Le cahier des charges spécifie également que chaque utilisateur doit être capable d'identifier simplement l'ensemble des utilisateurs pour lesquels l'agent est actif sur le réseau. Comme indiqué précédemment, dans le cas des indoor users, une requête en broadcast UDP est envoyée à tous les utilisateurs connectés lors de la connexion. Ainsi, les autres utilisateurs sont notifiés. De même dans le cas du serveur de présence, il transmet les informations de connexion et de déconnexion, ainsi tout le monde est notifié.

Une fonctionnalité ajoutée est *updateDisconnectedUsers* : lors de l'utilisation de l'application dans un réseau interne, lorsqu'un nouvel utilisateur arrive, les autres utilisateurs lui notifient les utilisateurs déconnectés qu'ils connaissent, pour que le nouvel arrivant puisse actualiser sa table locale.

## 7) Base de données : gestion de l'historique des messages et stockage des utilisateurs et de leurs informations

Nous avons choisi de stocker localement les informations (messages, et informations sur les autres utilisateurs) dans une base de données locale : *database.db* (voir manuel d'utilisation partie configuration de l'application, pour changer le nom du fichier utilisé).

Nous avons un tableau des utilisateurs nommé « **users** » avec les informations suivantes : adresse IP (sous forme de *String*), pseudo, status (0, déconnecté ou 1, connecté), *isExtern* (0, utilisateur interne ou 1, utilisateur externe).

Nous utilisons ensuite des tableaux de messages pour chaque utilisateur nommé de la façon suivante : « pseudo\_messages » qui stockent chacun des messages sous formes de **BLOB** (car nous stockons des Objets *Message*)

Quand un nouvel utilisateur se connecte, nous ajoutons une ligne dans le tableau « users » et nous créons un nouveau tableau pour stocker l'échange de messages.

Quand un utilisateur change de pseudo, nous changeons son pseudo dans « users » et nous changeons le nom du tableau des messages associé.

Quand nous envoyons ou recevons un message, nous l'ajoutons dans le tableau des messages associé à l'utilisateur.

Enfin, l'utilisation de la base de données facilite le « filtrage » des données : par exemple, pour obtenir la liste des utilisateurs connectés nous utilisons la requête "*SELECT address, username,*



*status, isExtern FROM users WHERE status = 1*". Ceci s'applique également pour récupérer les informations associées à un pseudo donné ou une adresse IP donnée.

## 8) Déconnexion et reconnexion

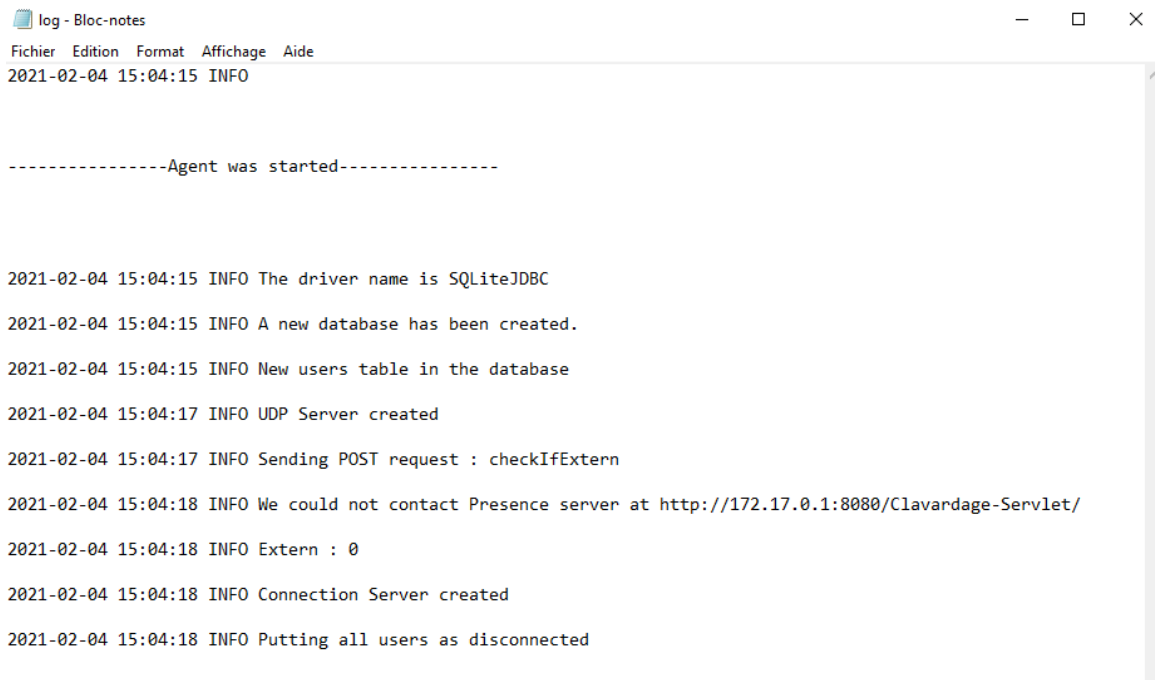
Lorsque l'utilisateur se déconnecte, les serveurs UDP et TCP sont arrêtés, nous fermons également toutes les connexions (*UserSocket*) et nous mettons tous les utilisateurs comme déconnectés.

A la reconnexion, nous redémarrons les serveurs UDP et TCP et nous laissons l'utilisateur choisir son pseudo (nous le notifions de son ancien pseudo pour plus de facilité à l'utilisation). L'utilisateur peut choisir de fermer l'application, ce qui le déconnectera et stoppera les serveurs UDP et TCP, etc. Il pourra ensuite relancer l'application et récupérer l'historique des messages et des utilisateurs étant donné que ces informations ont été stockées dans la base de données *database.db*.

A noter que notre statut est affiché dans le titre de la fenêtre principale.

## 9) Logs

Un système de logger a été mis en place : toutes les informations de debug sont affichées (si *debug=true*, voir manuel d'utilisation partie configuration de l'application) et stockées dans un fichier **log.log** (voir manuel d'utilisation partie configuration de l'application, pour changer le nom du fichier utilisé).



```

log - Bloc-notes
Fichier  Edition  Format  Affichage  Aide
2021-02-04 15:04:15 INFO

-----Agent was started-----

2021-02-04 15:04:15 INFO The driver name is SQLiteJDBC
2021-02-04 15:04:15 INFO A new database has been created.
2021-02-04 15:04:15 INFO New users table in the database
2021-02-04 15:04:17 INFO UDP Server created
2021-02-04 15:04:17 INFO Sending POST request : checkIfExtern
2021-02-04 15:04:18 INFO We could not contact Presence server at http://172.17.0.1:8080/Clavardage-Servlet/
2021-02-04 15:04:18 INFO Extern : 0
2021-02-04 15:04:18 INFO Connection Server created
2021-02-04 15:04:18 INFO Putting all users as disconnected
  
```

Exemple d'un fichier log.log

## 10) Servlet (serveur de présence)

Pour le serveur de présence, nous avons implémenté un Servlet qui gère les requêtes POST et également l'envoi d'informations via UDP broadcast afin de communiquer avec les indoor users. Le *Servlet* stocke localement les informations des utilisateurs dans des *Map*. Il associe à une adresse IP : un pseudo, un statut connecté ou déconnecté, et un statut externe ou

interne. Pour déterminer si un utilisateur est externe ou interne, nous vérifions si l'utilisateur appartient au sous-réseau interne du servlet : **192.168.1.0/24** (voir manuel d'utilisation partie configuration du serveur de présence).

Nous avons également implémenté une fonctionnalité de logger et l'affichage des informations sur les utilisateurs : les logs et les informations sont affichés lorsque nous accédons au serveur par requête GET, c'est-à-dire lorsque nous accédons via navigateur à l'adresse : [http://adresse\\_du\\_serveur:8080/Clavardage-Servlet](http://adresse_du_serveur:8080/Clavardage-Servlet).

A noter que les utilisateurs doivent configurer l'adresse IP du serveur présence manuellement (voir manuel d'utilisation partie configuration de l'application).

#### 11) Détection d'une autre instance de clavardage

Nous avons ajouté une fonctionnalité permettant de notifier l'utilisateur qu'une autre instance de l'application est déjà en cours (si c'est le cas bien entendu) et que nous ne pouvons donc pas démarrer l'application avant d'avoir fermé l'autre instance.

#### 12) Gestion des fichiers et images

Pour l'envoi et la réception des fichiers (et images), nous avons choisi de les transférer sous la forme de bytes afin de les « télécharger » dans le dossier correspondant : un dossier *file* et un dossier *image* sont créés et contiennent tous deux des sous-dossiers correspondant à chaque adresse IP des utilisateurs.

Nous avons également implémenté une fonctionnalité de survol pour les images afin d'en avoir un aperçu dans les conversations avec les autres utilisateurs.

Enfin, les fichiers et images sont téléchargeables en cliquant sur le message associé, cela ouvrira un explorateur de fichier pour sélectionner l'emplacement où l'on veut enregistrer.

## IV – Procédures de tests et de validation

### 1) Utilisation de dockers

Nous avons choisi d'utiliser des *dockers* pour pouvoir avoir une multitude de machines dans un même réseau local, et pouvoir déployer facilement et rapidement notre application. Différentes difficultés ont été rencontrées lors de la création du *Dockerfile*, permettant de créer une « image docker » depuis laquelle les différents dockers sont déployés : il a été assez difficile de mettre en place l'affichage de l'interface graphique pour chaque docker, et ne connaissant pas du tout cette technologie avant ce projet, cette étape a été très chronophage, mais primordiale pour la suite du débogage.

Nous avons également créé des petits scripts *bash* permettant de :

-préparer le Dockerfile, afin de mettre à jour la version du fichier Clavardage.jar dans l'image docker (*./preparedocker.sh*)

-lancer rapidement un nombre donné de dockers (*./multirundocker.sh nb*)

-stopper l'exécution d'un docker (*./killdocker.sh*)

De plus, après l'implémentation du *logger*, il était important de faire un script permettant de récupérer facilement les différents fichiers *log* des dockers : notamment dans le cas d'un crash du docker à cause de l'application. (*./log.sh*)

### 2) Premiers tests avec interface « console »

Après avoir mis en place les dockers, nous avons pu tester le fonctionnement de notre application sur le réseau local des dockers.

Étant donné que nous n'avions pas encore implémenté l'interface graphique, la première étape de test a été d'utiliser une interface « console » avec des commandes pour interagir avec les autres dockers. Les commandes que nous avons mis en place étaient les suivantes : « *changeUsername | printAll | printActiveUsers | printDisconnectedUsers | getOwnIP | disconnect | reconnect | end* »

La commande *printAll* affichait les informations que l'application avait au sujet des autres utilisateurs.

Ainsi, après avoir pu tester les fonctionnalités réseaux de notre application, nous avons implémenté l'interface graphique.

### 3) Implémentation de l'interface graphique

L'intégration de l'interface graphique s'est faite assez facilement, étant donné le parallèle avec notre interface console et les différentes commandes correspondantes à des boutons de l'application.

Voici la liste des différentes fonctionnalités et les situations associées que nous avons testé.

#### a. Utilisation dans un réseau local

Gestion des pseudos :

- Choix du pseudo du premier utilisateur, seul dans le réseau

- Un nouvel arrivant choisi un pseudo déjà pris : il est notifié de l'impossibilité de ce choix
- Deux « premiers » utilisateurs essaient de prendre le même pseudo : le premier récupère le pseudo et indique au second qu'il n'est plus disponible.
- Un utilisateur change de pseudo, il notifie tous les autres utilisateurs du réseau de ce changement (bien entendu, nous vérifions au préalable dans notre table locale qu'aucun utilisateur n'avait choisi ce pseudo)
- Lorsque l'application reçoit la notification de changement du pseudo d'un utilisateur, il met à jour l'affichage des listes d'utilisateurs si nous sommes sur cet onglet, et met à jour le pseudo affiché sur la conversation si nous sommes sur l'onglet de discussion avec l'utilisateur en question
- Choix d'un pseudo non valide (comportant un espace, des caractères spéciaux, etc.) : notification d'échec

#### Gestion des utilisateurs connectés / déconnectés :

- Connexion / déconnexion d'un utilisateur pendant qu'un autre utilisateur est sur l'onglet où sont affichées les listes d'utilisateurs : les listes sont mises à jour en temps réel
- Pareil, mais l'utilisateur est sur un autre onglet : lorsqu'il revient sur l'affichage des listes d'utilisateurs, elles sont mises à jour
- De même que pour le changement de pseudo, lorsque nous sommes sur l'onglet de discussion avec un utilisateur, s'il se déconnecte, l'affichage est mis à jour

#### Gestion des messages :

- Envoi d'un message (fichier, texte et image) à un autre utilisateur connecté : il est notifié avec un pop-up de la réception s'il n'est pas sur l'onglet de discussion
- Test de l'enregistrement d'un fichier ou d'une image par clic sur le message reçu
- Test de l'affichage de l'aperçu d'une image envoyée ou reçue
- Essai d'envoyer un message à utilisateur déconnecté ou en étant déconnecté : notification d'échec

#### Reconnexion et relance de l'application :

- Lors de la reconnexion ou de la relance de l'application, l'ancien pseudo de l'utilisateur est affiché
- L'historique de messages est bien sauvegardé après fermeture de l'application
- La relance de l'application reprend l'écriture des logs à la fin du fichier

#### b. Ajout du serveur de présence

L'ajout du serveur de présence ne gêne pas les fonctionnalités testées précédemment dans le réseau local.

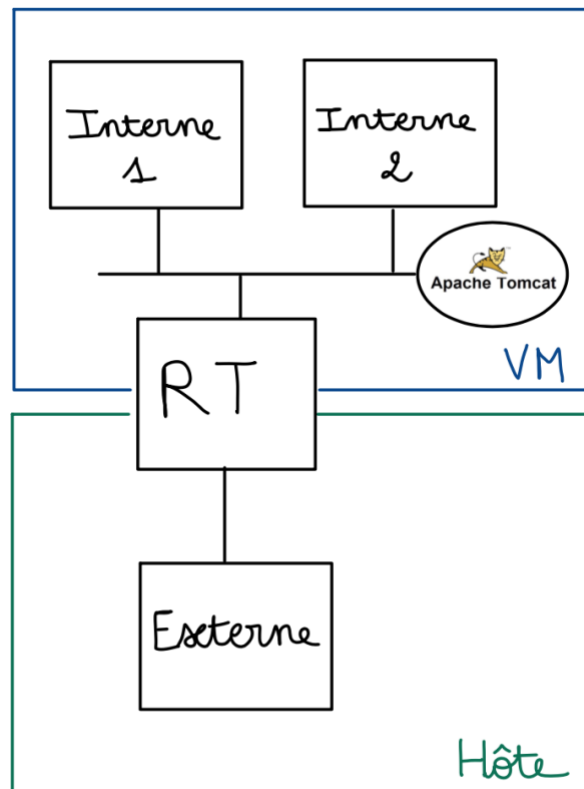
Nous avons testé le changement de pseudo, la connexion et la déconnexion d'un utilisateur externe puis d'un utilisateur interne pour vérifier que tous les utilisateurs reçoivent bien ces informations.

Lors des tests nous avons pu utiliser le système de logger que nous avons mis en place sur le *Servlet*.

#### 4) Vidéos de démonstration

Enfin, voici trois vidéos de démonstration de l'utilisation de notre application.

- Premier cas, utilisation dans un réseau local : <https://youtu.be/Ojc-RVkksYk>
- Second cas, utilisation du serveur de présence : <https://youtu.be/KS4yV88tEgw>



*Schémas réseau correspondant*

RT = routeur, mais le routage est direct : câble réseau virtuel entre la machine virtuelle VM et l'hôte.

- Autres fonctionnalités : <https://youtu.be/YL51-GbzIHA>

## V – Manuel d'utilisation de notre produit

### 1) Télécharger l'application

La première étape pour télécharger notre application est d'installer sur votre machine la version 11.0.6 de java.

Par la suite, veuillez trouver le lien vers notre application :

<https://github.com/PiKouri/4a-projet-oo/raw/main/Runnable/Clavardage.rar>

Veuillez télécharger et décompresser ce dossier.

### 2) Configurer l'application

À l'ouverture du dossier « Clavardage », vous trouverez le fichier « config.properties », qui vous permet de configurer l'application. Veuillez suivre les commentaires du fichier qui vous indiquent l'utilité des variables modifiables. Vous pouvez notamment :

- Désactiver le mode debug (activé par défaut). Pour ce faire, fixer la variable *debug* à *false*.
- Modifier la durée du timeout (via la variable *timeout*).
- Modifier le port utilisé pour les connexions en broadcast (via la variable *broadcastPortNumber*).
- Modifier le port utilisé pour les connexions TCP sortantes (via la variable *defaultPortNumber*).

La configuration essentielle qu'il vous faut effectuer est celle qui indique **l'adresse du serveur de présence**, via la variable *presenceServerIPAddress*. Afin de bien configurer cette dernière, veuillez à ce qu'elle soit en accord avec l'adresse à laquelle le serveur de présence est accessible.

Vous êtes à présent prêts pour lancer l'application en interne via « Clavardage.jar ».

### 3) Configurer et installer le serveur de présence sur le serveur Tomcat

Au préalable, il vous faut avoir installé le serveur Tomcat sur la machine qui vous servira de serveur de présence.

Veuillez trouver le lien vers notre servlet (serveur de présence) :

<https://github.com/PiKouri/4a-projet-oo/raw/main/Runnable/Clavardage-Servlet.rar>

Vous pouvez télécharger et décompresser ce dossier pour y accéder. Par la suite, déployez le fichier « Clavardage-Servlet.war » sur le serveur Tomcat.

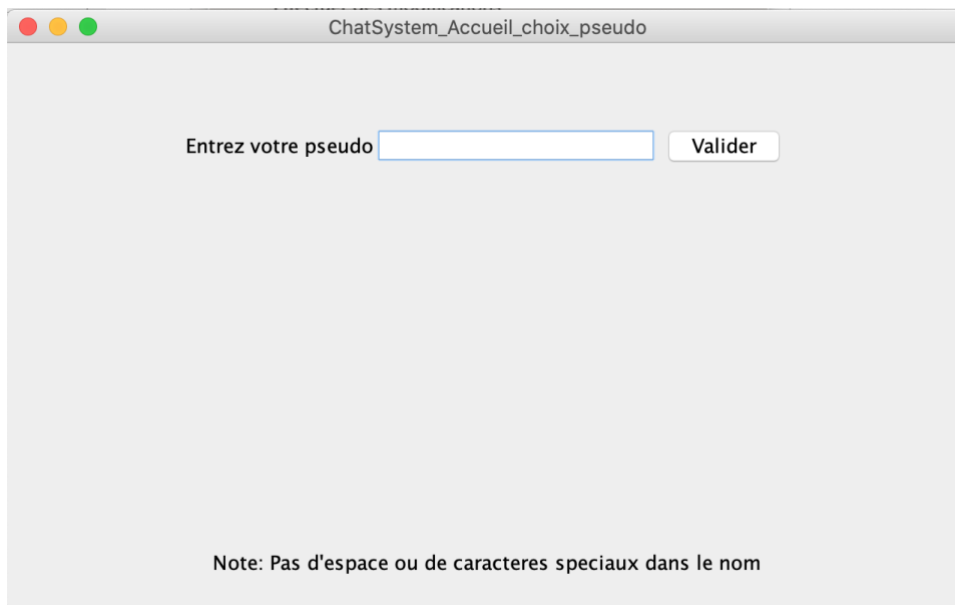
Veuillez noter que l'adresse du réseau interne doit être 192.168.1.0/24. Ainsi, le serveur de présence doit avoir une interface dans ce sous-réseau. De plus, les utilisateurs qui veulent

communiquer dans le réseau indoor doivent également avoir une adresse IP dans ce sous-réseau.

Vous pouvez accéder à l'interface de debug du servlet via le lien suivant : [http://adresse\\_du\\_serveur:8080/Clavardage-Servlet](http://adresse_du_serveur:8080/Clavardage-Servlet) (remplacer "adresse du serveur" par son adresse interne effective).

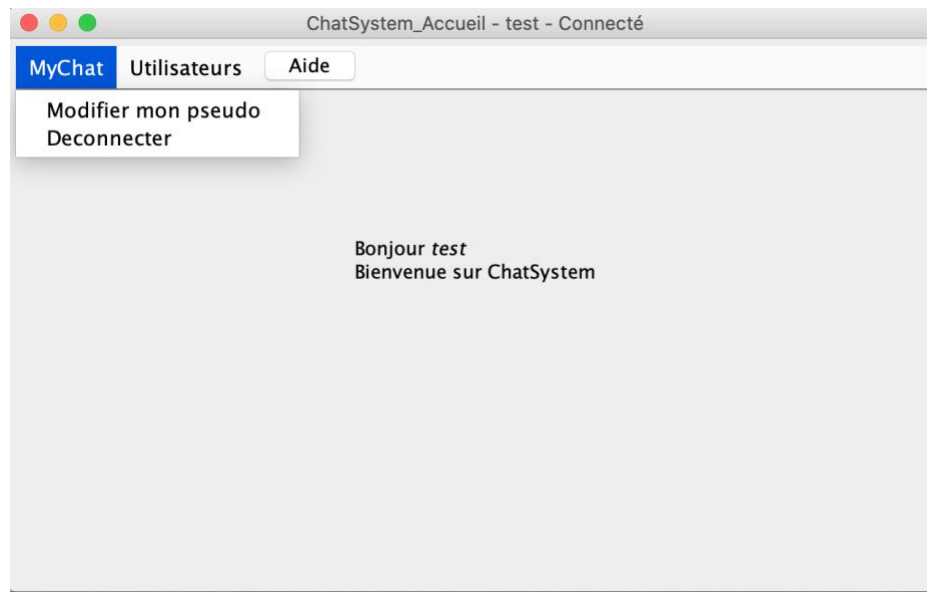
#### 4) Utiliser les différentes fonctionnalités

Au lancement de l'application, vous trouverez une page d'accueil vous proposant d'entrer un pseudo afin de vous connecter. Veuillez en choisir un, puis le valider via le bouton correspondant ou bien la touche Entrée de votre clavier.



#### Page de connexion de notre ChatSystem

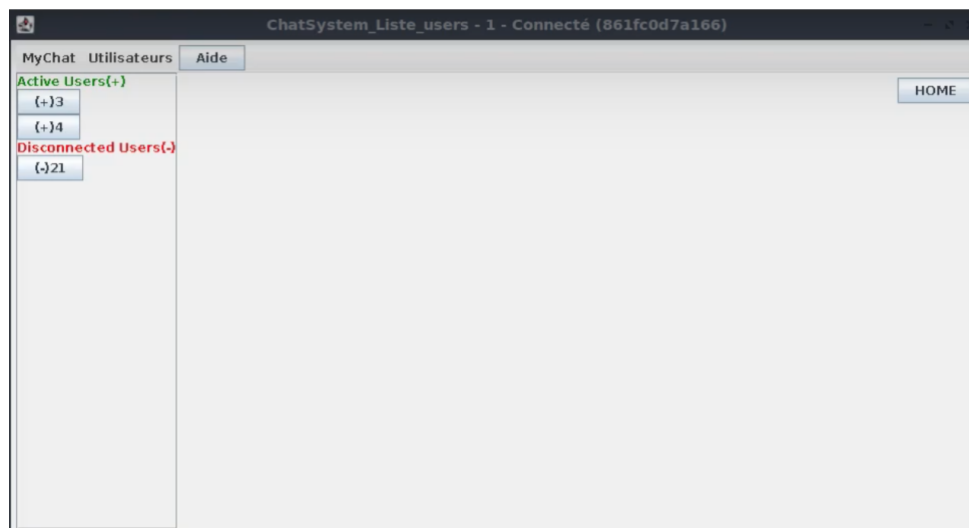
Si le pseudo n'est pas valide, veuillez en choisir un nouveau. Il ne doit pas contenir de caractères spéciaux, d'espaces, ni être vide, ou déjà utilisé par un autre utilisateur. Une fois ce dernier vérifié par l'application, vous serez connecté et arriverez sur notre page d'accueil.



Page d'accueil de notre ChatSystem

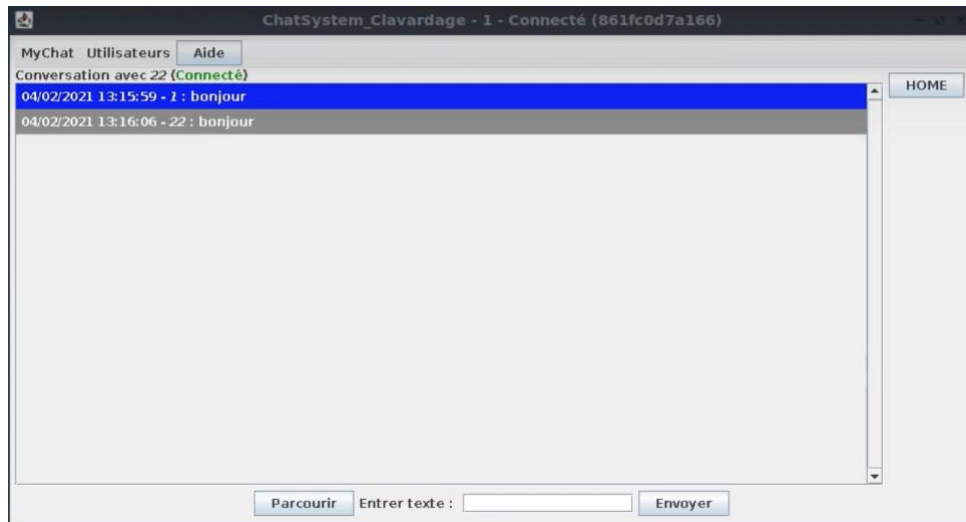
Vous disposez à présent de toutes les fonctionnalités de notre système de clavardage. Vous pouvez notamment Changer votre pseudo ou bien vous déconnecter via le bouton « MyChat » de la barre de menu.

Via le bouton « Utilisateurs », vous arrivez sur la liste des utilisateurs, séparés selon s'ils sont connectés ou non. En cliquant sur le pseudo d'une personne, l'historique des messages avec cette dernière s'affiche. Le bouton « HOME » vous permet à tout moment de retourner à la page d'accueil.



Vue des autres utilisateurs





### Historique d'une conversation

Enfin, le bouton d'aide vous apporte des informations complémentaires sur le fonctionnement des envois et réceptions d'images et de fichiers.