****

**Projet de Programmation**

**Orientée Objet**

**Chat System**

**Emma Baudoint / Matthieu Jacques**

4IR C1 | 2019/2020

Table des matières

[Guide d’installation pour l’administrateur 3](#_Toc63851029)

[1 - Récupérer le code 3](#_Toc63851030)

[2 - Installer la bonne version 3](#_Toc63851031)

[Guide d’utilisation pour l’utilisateur 3](#_Toc63851032)

[1 - Exécuter le programme 3](#_Toc63851033)

[2 - S’identifier 3](#_Toc63851034)

[3 - Choisir son pseudo pour la session 4](#_Toc63851035)

[4 - Se servir du chat 4](#_Toc63851036)

[Choix d’implémentation 5](#_Toc63851037)

[1 - Plages de port 5](#_Toc63851038)

[2 – Identifiants 6](#_Toc63851039)

[3 - Fonctionnement du serveur 6](#_Toc63851040)

[5- Systèmes d'exploitations 7](#_Toc63851041)

[Procédure de test 7](#_Toc63851042)

[1- Nouvelle connexion 7](#_Toc63851043)

[2- Base de données et communication 7](#_Toc63851044)

[3- Servlet 7](#_Toc63851045)

[Future Implémentation 8](#_Toc63851046)

[1- Régler le problème d’adresse 8](#_Toc63851047)

[2- Ajouter une barre de scroll pour les messages 8](#_Toc63851048)

[3- Ajout d’un transfert de fichier 8](#_Toc63851049)

# Guide d’installation pour l’administrateur

## 1 - Récupérer le code

Pour commencer l’installation, il faut récupérer les sources sur le lien suivant :

<https://github.com/Emma-bau/poo.git>

ou avec la ligne de commande suivante :

git clone <https://github.com/Emma-bau/poo.git>

## 2 - Installer la bonne version

Si l’ordinateur de l’utilisateur est dans l’entreprise, on garde sur son ordinateur la version chatInterne.jar. Sinon il faudra qu’il utilise la version chatExterne.jar. C’est .jar sont présents dans le dossier git. Par mesure de sécurité, il faudra donc installer seulement celui utilisé. Si les .jar ne fonctionnent pas, il faudra passer par éclipse pour lancer l’application à chaque utilisation.

Il faut aussi faire attention à la version de java installée sur l’ordinateur afin de pouvoir ouvrir le fichier, si l’ordinateur n’arrive pas à ouvrir l’application, installée la version de java disponible ici :

<https://www.oracle.com/java/technologies/javase-jdk15-downloads.html?fbclid=IwAR2IR9jrm3pQ4h5fP5clFQUTPj_sv6fEOXw7e__ZhYrAtTpi_4NXIJmgSIo>

Ces versions n’ont été testées que sur des systèmes d’exploitation Windows et Linux.

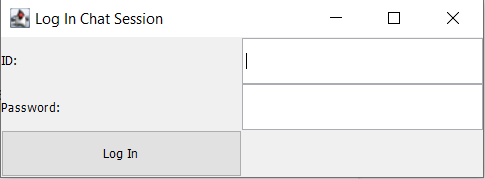
# Guide d’utilisation pour l’utilisateur

## 1 - Exécuter le programme

Lancer le fichier .jar

## 2 - S’identifier

Au lancement, la fenêtre suivante doit apparaître en haut à gauche de l’écran.



4 comptes sont disponibles:

ID: 1   /  Password: toto

ID: 2   /  Password: toto

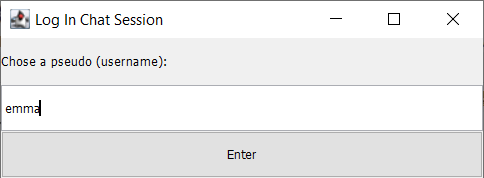
ID: 3   /  Password: toto

ID: 4   /  Password: toto

Cliquez sur Log In pour ouvrir la fenêtre suivante. Un même compte ne doit être utilisé que sur une instance du programme à la fois

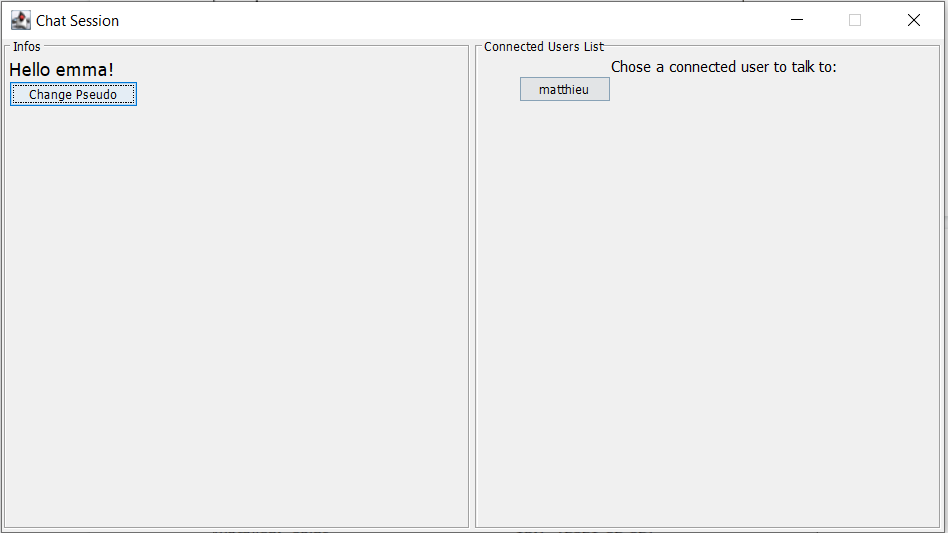
## 3 - Choisir son pseudo pour la session

La fenêtre suivante vous permet d’entrer votre pseudo pour la session. Il peut être modifié plus tard

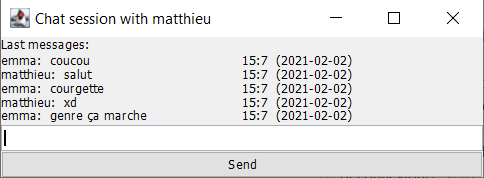


## 4 - Se servir du chat

A gauche s’affiche votre pseudo et le bouton pour le changer. A droite se trouve la liste dynamique des utilisateurs connectés. Cliquez sur un nom pour ouvrir une session de clavardage avec ce dernier.



L’historique des messages avec cet utilisateur (ID) est chargé et vous pouvez envoyer des messages.



# Choix d’implémentation

## 1 - Plages de port

Le tp étant réalisé en distanciel, nous avons créer une première partie de notre projet, qui est dans le dossier chat\_plage\_de\_port sur le git. Cette version de notre projet s’appuie sur l’utilisation de la messagerie sur un seul ordinateur. Afin de pouvoir ouvrir plusieurs serveurs tcp/ udp sur des ports différents d’un même ordinateur, nous avons fait le choix d’attribuer aléatoirement un port à chaque nouvelle connexion d’un utilisateur. On peut attribuer jusqu’à 1000 utilisateurs. Nous avons conservé cette implémentation de plage de port dans notre projet final dans la version chatInterne et chatExterne. Nous ne pouvons donc pas respecter la mention des 100 000 utilisateurs connectés en même temps, dû au nombre de ports disponibles.

## 2 – Identifiants

Etant donné que le cahier des charges stipule que chaque utilisateur doit avoir un compte sur une machine, nous avons simplement créé des comptes pré-faits dans le code. Cela nécessitera dans l’optique d’une prochaine amélioration de mettre en place une base de donnée cryptée avec les identifiants et les mots de passe.

## 3 - Fonctionnement du serveur

Concernant la mise en place de la reconnaissance des utilisateurs en ligne, de leurs informations et de leurs déconnexions, nous avons choisi d’utiliser l’UDP car nous n’avions pas besoin de réponse. Par contre, la communication et l'envoi de message entre utilisateur nécessitant une réponse, nous avons préféré utiliser le TCP pour cette partie. Le TCP fonctionne de la façon suivante :

Serveur TCP

Client TCP

Serveur TCP

Client TCP

Utilisateur 1

Utilisateur 2

Serveur TCP

Client TCP

Utilisateur 3

L’utilisateur 1 va passer par son client TCP pour dialoguer avec un utilisateur 2 lorsque c’est lui qui initie la connexion. Son client va dialoguer avec le serveur TCP de l’utilisateur 2. Si c’est l’utilisateur 3 qui initie la connexion avec notre utilisateur 1, il dialoguera avec le serveur de notre utilisateur 1.

## 5- Systèmes d'exploitations

Dû à la crise sanitaire, nous n’avons pas été en mesure de tester notre application sur un système d’exploitation différent de celui de Linus et de Windows. Nous ne pouvons donc pas garantir la compatibilité sur les autres systèmes d’exploitation.

# Procédure de test

## Nouvelle connexion

Afin de tester notre serveur UDP, il suffit de lancer plusieurs instances de notre application sur un même ordinateur, puis sur différents ordinateurs. On doit voir apparaître dans la liste des utilisateurs connectés le pseudo des autres instances connectés. Nous avons en priorité travailler sur un seul ordinateur et n’avons réaliser des tests sur plusieurs ordinateurs que bien plus tard.Nous avons parfois rencontré des problèmes au niveau du broadcast avec des paramètres de réseaux wifi qui bloquent celui-ci sur certains ordinateurs.

## Base de données et communication

Une fois ces instances lancées, il suffit de cliquer sur un pseudo afin d’établir la connexion TCP. A ce moment-là, le chargement de l’historique des messages se fait ( ce qui prouve que l’accès à la base de données est bien réalisé ). On peut alors envoyer un message à un autre utilisateur. Celui-ci est bien reçu par la personne, le transfert en TCP a donc bien fonctionné. Il apparaît aussi dans l’historique de la personne qui l’a envoyé, la mise à jour de la base donnée se fait donc correctement.

## Servlet

Pour le servlet, nous n’utilisons que des requêtes POST et GET. Il faut utiliser au moins une fois l’application chatExterne.jar. En ouvrant plusieurs autres instances avec n’importe laquelle de nos deux applications, on doit pouvoir voir dans la liste des personnes connectées l’utilisateur qui est passé par cette application. Nos requêtes fonctionnent donc correctement. Il reste néanmoins un problème de communication car nous récupérons grâce à ce serveur l’adresse privée de la personne et non son adresse publique ce qui nous bloque au niveau de la communication en TCP. Nous avions pensé à faire passer le message par notre servlet mais cela ne correspond pas à ce qui nous est demandé.

# Future Implémentation

## Régler le problème d’adresse

Il nous reste à régler le problème de récupération des adresses publiques. Une solution à ce problème pourrait être de faire passer les messages par le serveur mais cela semble peu cohérent avec la structure créée.

## 2- Ajouter une barre de scroll pour les messages

Afin de rendre les messages plus lisibles, il faut actuellement agrandir la fenêtre. On peut cependant envisager l'implémentation d’une barre de scroll afin d’avoir une meilleure lisibilité des anciens messages.

## Ajout d’un transfert de fichier

A l’avenir les utilisateurs devraient aussi pouvoir envoyer des fichiers et des images.