

# RAPPORT DE PROJET

---

**Julien Ferry**

**Quentin Genoud**

**4IR-A**

Ce dossier contient :

- L'ensemble des composants de l'application client de clavardage (les fichiers .java)
- Un dossier compressé (.zip) contenant les fichiers utiles au déploiement du serveur de présence
- Un driver JDBC (au format .jar)
- Un fichier SQL permettant la mise en place de la base de données utilisée (sur MySQL)

*Note : Tous les diagrammes de conception sont accessibles à l'adresse suivante :*

<https://drive.google.com/open?id=1A7HP-Bph0FLwvJ8UnAnUXOjCFPdM0w4S>

## APPLICATION CLIENT

---

*Un utilisateur correspond à une machine, on ne peut donc lancer qu'une seule session de l'application sur un ordinateur. Une conversation correspond au couple (pseudo de l'utilisateur distant, date de la conversation).*

Pour **compiler les fichiers sources de l'application client**, il suffit d'exécuter la commande suivante :

```
| javac *.java
```

**Installation :**

- En fonction du mode de découverte des utilisateurs en ligne choisi : Avoir installé un conteneur de Servlet, et avoir placé correctement le dossier de la servlet fournie (voir la section "Serveur de présence").
- En fonction du mode de persistance des données choisi : Créer un dossier "histories" au même niveau que le dossier clavardage, ou avoir installé et configuré mySQL (voir la section "Historique et Persistance des données").

Pour **lancer le programme** :

Se placer dans le dossier au dessus des dossiers clavardage et histories et exécuter la commande :

```
| java clavardage/Run
```

**Résumé des fonctionnalités implémentées par notre application de clavardage :**

- Découverte des utilisateurs en ligne par l'utilisation d'un serveur de présence ou via une architecture distribuée (au choix)
- Stockage (automatique) et consultation des historiques de conversation dans une base de données ou sous forme de fichiers (au choix)
- Discussion entre utilisateurs via des sockets TCP
- Possibilité pour un utilisateur de changer son pseudo une fois connecté
- Interface graphique sobre et fonctionnelle

## Traitement des erreurs :

Les erreurs non critiques (c'est à dire que l'application peut continuer à fonctionner normalement après leur occurrence) sont traitées dans des blocs try/catch, affichées dans la console (indications permettant de localiser leur occurrence dans le programme + affichage de la trace) et, si elles sont liées à une action de l'utilisateur, elles donnent lieu à l'affichage d'un message d'erreur dans la fenêtre de l'application.

Certaines erreurs critiques sont partiellement, ou pas gérées. Par exemple, au cours de la connexion, si le serveur de présence spécifié par l'utilisateur n'est pas joignable, un message d'erreur est affiché au bout d'un certain délai pendant lequel l'application n'est pas utilisable. Des erreurs encore plus graves, comme par exemple l'impossibilité d'ouvrir le DatagramSocket servant à la découverte du réseau en UDP, donnent lieu à un affichage dans la console, mais leur occurrence empêche évidemment le bon fonctionnement de l'application.

## Présentation rapide du parcours de l'utilisateur dans notre application :

L'interface graphique de notre application a été implémentée grâce à l'API Swing. Pour des soucis d'ergonomie, le binding de certaines touches a été réalisé (par exemple la touche Entrée pour envoyer le message saisi), de même que le binding avec les éléments graphiques de la fenêtre (par exemple la croix pour fermer proprement la fenêtre ou l'application). La validité des différentes saisies de l'utilisateur est toujours vérifiée avant qu'elles ne donnent lieu à des modifications.

### 1) Fenêtre de connexion :

Welcome. Enter your pseudo please.

Julien

Validate Pseudo

---- Network discovery mode selection : ----

☒ Local Mode (UDP-Based)

☐ Distant Mode (HTTP-Based)

Enter HTTP presence Server Address :

localhost

Enter HTTP presence Server Port :

8080

---- History saving mode selection ----

☐ History save in a database (requires mySQL to be installed)

☒ Save history in File System

Quit

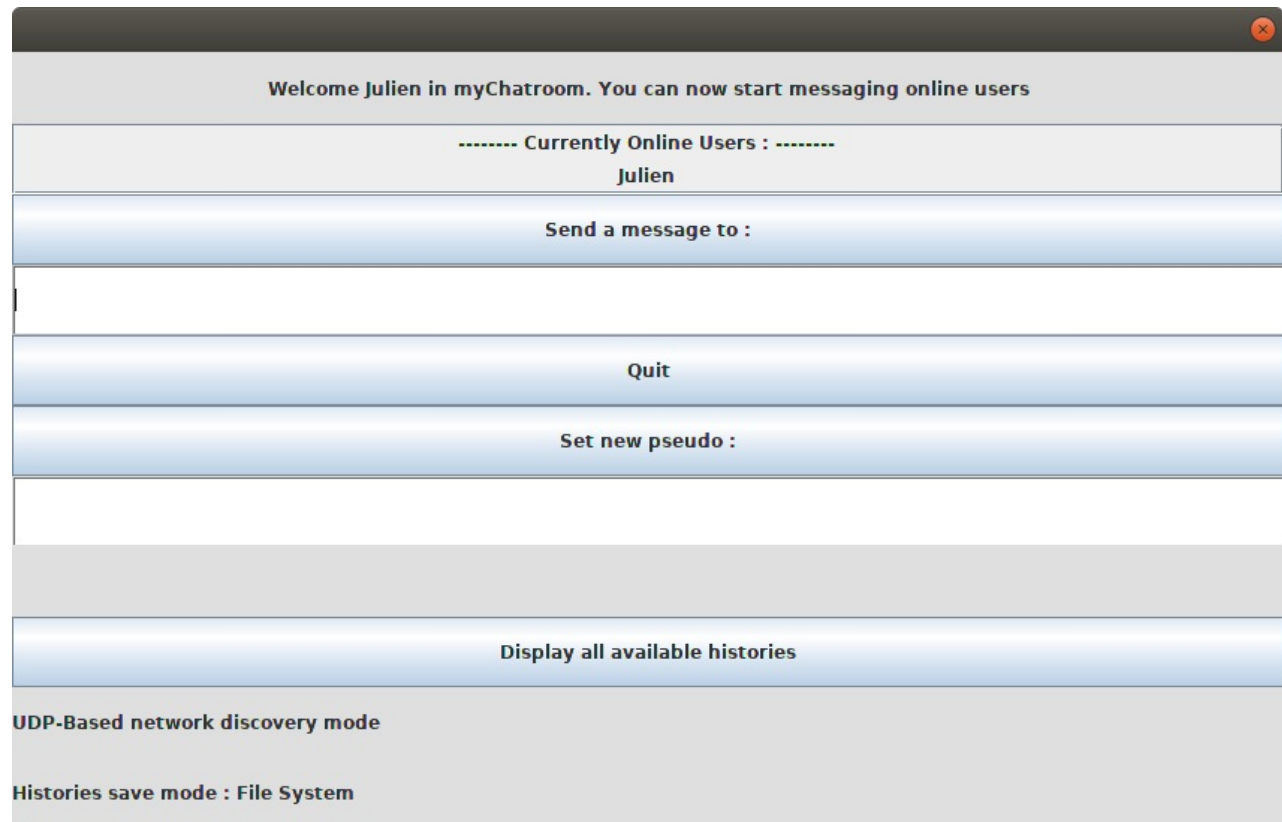
Avant de se connecter à l'application de Chat, **l'utilisateur doit choisir** :

- *le mode de découverte du réseau* : soit **basée sur UDP** (ce qui requiert que les différents utilisateurs soient sur le même domaine de broadcast et limite donc l'utilisation de ce mode aux réseaux locaux), soit **basée sur l'utilisation d'un serveur de présence HTTP** (ce qui requiert qu'un serveur de présence HTTP soit déployé et accessible, à une adresse et un numéro de port connus et à renseigner dans l'application).

- *le mode de stockage des historiques* : soit **dans des fichiers** (requiert seulement la présence d'un dossier "histories" au même niveau que le dossier "clavardage"), soit **dans une base de données** (requiert une installation d'un logiciel de base de données, et la création de tables, comme décrit dans la section dédiée de ce README).

Au moment de se connecter (après le clic sur le bouton dédié), **l'application vérifie que le pseudo rentré par l'utilisateur est non vide et libre** (en fonction du mode de découverte du réseau choisi, soit avec une requête GET au serveur de présence soit par broadcast UDP et analyse des réponses).

## 2) Fenêtre principale :



The screenshot shows a window titled "Welcome Julien in myChatroom. You can now start messaging online users". Below the title bar, there is a section for "Currently Online Users : ....." which lists "Julien". Below this, there is a button labeled "Send a message to :". Underneath the button is a text input field. Below the input field is a button labeled "Quit". Below the "Quit" button is another button labeled "Set new pseudo :". Underneath this button is another text input field. Below the input field is a button labeled "Display all available histories". At the bottom of the window, there is a status bar with two lines of text: "UDP-Based network discovery mode" and "Histories save mode : File System".

Après connection, l'utilisateur a, sur cette fenêtre principale, la possibilité de :

- *voir la liste des utilisateurs en ligne* (seulement l'utilisateur Julien sur la capture d'écran par exemple)
- *changer son pseudo*. Si le changement n'est pas possible (pseudo déjà pris par exemple), l'application le signale à l'utilisateur et n'effectue pas le changement.
- *lancer une conversation avec un utilisateur* (si une conversation n'est pas déjà lancée avec cet utilisateur, une nouvelle fenêtre s'ouvre alors, chez l'utilisateur distant et chez l'utilisateur local ; si une conversation avec cet utilisateur a déjà été créée, un message s'affiche pour l'indiquer à l'utilisateur).
- *afficher la liste des historiques stockés sur cette machine* (une nouvelle fenêtre s'ouvre, et l'utilisateur peut alors consulter l'historique de conversation de son choix (les historiques disponibles seront ceux correspondant au mode de stockage spécifié lors de la connection, tandis que ceux stockés dans l'autre mode ne seront pas visibles)).
- *se déconnecter*

Notes :

- Dans tous les cas, les historiques de conversation sont sauvegardés automatiquement par l'application, au cours des conversations.
- La réception d'un message dans une conversation déjà ouverte entraîne la mise au premier plan de la fenêtre de conversation concernée.
- Les modes de découverte des utilisateurs en ligne et de stockage des historiques choisis lors de la connection sont rappelés en bas de la fenêtre.
- La liste des utilisateurs en ligne est mise à jour automatiquement grâce à une tâche périodique programmée sur un Timer (de cette manière on évite de surcharger le seueur de présence inutilement). Dans le cas de l'utilisation d'un serveur de présence, les requêtes sont effectuées périodiquement, tandis que dans le cas de l'architecture distribuée, les différents messages reçus permettent de maintenir à jour la liste des utilisateurs en ligne.

## DECOUVERTE DES UTILISATEURS EN LIGNE

Deux modes de découverte du réseau sont possibles. Le choix est laissé à l'utilisateur, comme décrit dans la section "Application Client". Les contraintes et prérequis de chaque mode sont également détaillés dans la section "Application Client".

### 1) Découverte du réseau par broadcast UDP

Ce mode de découverte du réseau n'implique aucune installation particulière. Il correspond simplement à **l'échange de messages UDP d'une syntaxe particulière, analysés et exploités par tous les clients appartenant au domaine de broadcast correspondant.**

La syntaxe générale de ces messages émis en broadcast est la suivante :

- Demande des pseudos en ligne : [000]
- Se déclarer en ligne : [001]\_Pseudo
- Réponse à une déclaration en ligne : [011]\_Pseudo
- Déclaration d'un nouveau pseudo : [021]\_NouveauPseudo (*Ancien pseudo déterminé par l'adresse*)
- Se déconnecter : [002]\_Pseudo

L'inconvénient majeur de ce mode de découverte est la possibilité de pertes (difficilement détectables et corrigibles) de paquets (UDP n'offrant aucune garantie de service).

### 2) Serveur de présence

**Pour déployer le serveur de présence contenu dans le fichier "presenceserver.zip", l'installation d'un conteneur de servlet est un prérequis. Le dossier proposé ici a été constitué pour fonctionner avec le logiciel libre TomCat. Voici son arborescence :**

```
presenceserver
|_WEB_INF
|   |_classes
|       |_presenceServer
|           |_ClavardageServlet.class
|   |_src
|       |_presenceServer
|           |_ClavardageServlet.java
|_web.xml
```

C'est l'arborescence classique d'une servlet Java, pour TomCat. **Le répertoire "classes" contient le résultat de la compilation de la servlet, tandis que le répertoire "src" contient le code source de la servlet.**

**Le fichier "web.xml" est le "descripteur de déploiement".** Il contient, au format xml, les paramètres de déploiement de la servlet sur le serveur de TomCat. Par exemple, c'est ce fichier qui définit l'adresse URL par laquelle la servlet est accessible. Nous l'avons déjà rempli, et il n'est pas nécessaire de le modifier. Le serveur de présence est accessible à l'adresse :

```
localhost:8080/presenceserver/connect
```

(si le serveur TomCat est déployé localement et sur le port 8080 (le choix du port de déploiement est défini dans votre installation TomCat, et modifiable dans le fichier "server.xml" du répertoire "conf" dans le dossier d'installation de TomCat)).

Pour **déployer le serveur de présence**, il faut copier/coller le dossier presenceserver (décompressé) dans le répertoire webapps de votre installation TomCat.

*Note : la servlet est ici déjà compilée, mais pour la (re)compiler, il est nécessaire d'inclure l'API Servlet (qui fait partie de JEE). Un moyen de le faire est de copier/coller la version de cette API fournie par TomCat (qui se trouve dans le répertoire d'installation de TomCat, dans le dossier "lib" (servlet-api.jar)) dans le répertoire d'installation de Java (sous jre/lib/ext). La commande à exécuter pour compiler la servlet (en se plaçant dans le dossier WEB\_INF) et stocker le résultat dans le répertoire adéquat est alors :*

```
javac -d classes src/presenceServer/ClavardageServlet.java
```

Une fois le dossier décompressé copié collé dans le repertoire webapps de TomCat, il suffit de **lancer le serveur TomCat** avec la commande :

```
sudo $CATALINA_HOME/bin/startup.sh (où $CATALINA_HOME est le répertoire d'installation de TomCat)
```

Pour **arrêter le serveur**, la commande suivante peut être utilisée :

```
sudo $CATALINA_HOME/bin/shutdown.sh
```

**Le format général des requêtes (GET) que notre serveur de présence va traiter** est le suivant :

```
adresseDéploiementServeurTomcat:portDéploiement/presenceserver/connect?display="unBooléen"&pseudo="unPseudo"&type="unType"
```

#### **Explication des paramètres :**

- *unBooléen* peut être soit true (le serveur renvoie alors un affichage en HTML de plusieurs paramètres, notamment la liste des utilisateurs connectés, le nombre de requêtes reçues...etc), soit autre chose (auquel cas le serveur renvoie des données dans un format exploitable par l'application client).
- *unPseudo* est le pseudo de l'utilisateur qui effectue la requête
- *unType* est le type de requête que l'utilisateur effectue. Ce peut être :
  - *connect* : l'utilisateur effectue cette requête pour **signaler au serveur de présence qu'il est en ligne.**
  - *disconnect* : l'utilisateur effectue cette requête pour **signaler au serveur de présence qu'il n'est plus en ligne.**
  - *change* : l'utilisateur effectue cette requête pour **signaler au serveur de présence qu'il change de pseudo.** Le paramètre *unPseudo* contient alors le nouveau pseudo, et c'est

l'adresse du client qui est utilisée par le serveur pour l'identifier.

- *info* : l'utilisateur effectue cette requête pour **demander au serveur la liste des utilisateurs connectés**.
- *isfree* : l'utilisateur effectue cette requête pour **demander au serveur si le pseudo renseigné dans le champ *unPseudo* est libre (ou s'il est déjà pris)**.

Les différentes actions associées aux requêtes listées ci-dessus sont effectuées indépendamment de la valeur du paramètre display. Seul le format de la réponse du serveur change.

### Affichage dans un navigateur web des informations du serveur

Requête utilisée :

```
localhost:8080/presenceserver/connect?
display="true"&type="info"&pseudo="julien"
```



### Presence Server Graphical Display

Server address: julien-VirtualBox/127.0.1.1

Remote Address: 127.0.0.1

type = info

pseudo = julien

nbRequests = 8

OnlineUsers:

- Julien is online (@10.0.2.15)
- Quentin is online (@10.0.2.17)

## HISTORIQUE ET PERSISTANCE DES DONNEES :

La persistance de données que nous avons choisi d'implémenter permet de **sauvegarder l'historique en local**. C'est-à-dire que vous ne pouvez voir que les historiques des conversations que vous avez eues sur l'ordinateur utilisé. Par exemple, si vous prenez le pseudo « Antoine7 » sur un ordinateur A et que plus tard vous prenez ce même pseudo « Antoine7 » sur un ordinateur B, vous ne pourrez pas accéder à l'historique des messages que vous avez eu en utilisant l'ordinateur A. Nous avons fait ce choix pour des questions de sécurité et confidentialité.

De plus, il faut savoir que **nous avons choisi de laisser le choix à deux modes de persistance de données différents**. Le premier mode nécessite l'installation de MySQL et l'utilisation de JDBC tandis que le deuxième n'a pas besoin d'utiliser MySQL et enregistre l'historique dans des fichiers texte. Nous avons fait ce choix car par exemple sur les ordinateurs de l'INSA, on ne peut pas installer MySQL (car nous n'avons pas les droits suffisants sur ces ordinateurs) donc l'historique ne serait pas disponible.

**Ce choix de mode s'effectue sur la fenêtre de lancement de l'application** (en même temps que le choix entre le mode UDP et le serveur distant). **L'historique est accessible une fois connecté sur la fenêtre principale en cliquant sur le bouton « Display all available histories »**, mais on ne peut accéder à l'historique que dans le mode choisi (c'est-à-dire que soit on peut voir l'historique stocké dans les fichiers textes, soit l'historique stocké dans la BDD SQL mais pas les deux en même temps).

# 1) Historique avec base de données (MySQL et JDBC) :

Tout d'abord, **pour utiliser ce mode il faut avoir installé MySQL**. Puis, **il faut que le driver JDBC soit rajouté au classpath du projet** afin qu'il puisse être utilisé. Pour ce faire, dans notre git on retrouve un fichier.jar (mysql-connector-java-8.0.13.jar) qu'il suffit de rajouter au classpath du projet. Pour Eclipse, il faut faire un clic droit sur la base du projet Eclipse puis cliquer sur « Propriétés ». Dans le menu « Propriétés », il faut choisir « Java Build Path » puis sélectionner l'onglet « Libraries » puis « Add jars » pour pouvoir ajouter le fichier.jar au classpath.

Une fois le driver ajouté au classpath, il faut créer manuellement la base de données dans MySQL ainsi que l'utilisateur qui aura les droits sur cette base de données. Pour ce faire, il suffit de taper les commandes dans MySQL écrites dans le fichier « histo.sql » qui se trouve dans notre repository git ou alors d'exécuter le fichier (s'il se trouve dans le répertoire où a été lancé MySQL) avec la commande SQL « SOURCE histo.sql; ».

Une fois la base de données initialisée, il suffit de laisser tourner MySQL en fond puis on peut se servir de l'application avec ce mode de persistance.

## Exemple de table de données correspondant à une conversation passée (dans MySQL) :

```
mysql> SELECT * FROM HistConv;
```

pseudo_and_date_debut_conv	message
Julien_Tue Jan 15 14_34_41 CET 2019.txt	----- DEBUT CONVERSATION -----
Julien_Tue Jan 15 14_34_41 CET 2019.txt	Received: [Tue Jan 15 14:34:44 CET 2019] Bonjour
Julien_Tue Jan 15 14_34_41 CET 2019.txt	Sent: [Tue Jan 15 14:34:50 CET 2019] Salut Julien
Julien_Tue Jan 15 14_34_41 CET 2019.txt	Received: [Tue Jan 15 14:34:53 CET 2019] Comment vas-tu ?
Julien_Tue Jan 15 14_34_41 CET 2019.txt	Sent: [Tue Jan 15 14:35:08 CET 2019] Très bien, tu as fini le README ?
Julien_Tue Jan 15 14_34_41 CET 2019.txt	Sent: [Tue Jan 15 14:35:26 CET 2019] Impressionnant, quelle abnégation !
Julien_Tue Jan 15 14_34_41 CET 2019.txt	----- FIN CONVERSATION -----

# 2) Historique avec fichiers texte :

Pour utiliser ce mode de persistance de données, **il suffit de créer un dossier « histories » au même niveau que le dossier « clavardage »**. Dans ce dossier, les fichiers textes qui sauvegardent l'historique des messages seront automatiquement créés lors de l'utilisation de l'application.

## Exemple de fichier texte correspondant à une conversation passée :

```
Quentin_Tue Jan 15 14_34_42 CET 2019.txt
1 ----- Pseudo :Quentin-----
2 ----- Date de la session de conversation :Tue Jan 15 14:34:42 CET 2019-----
3 Sent: [Tue Jan 15 14:34:45 CET 2019] Bonjour
4 Received: [Tue Jan 15 14:34:51 CET 2019] Salut Julien
5 Sent: [Tue Jan 15 14:34:54 CET 2019] Comment vas-tu ?
6 Received: [Tue Jan 15 14:35:09 CET 2019] Très bien, tu as fini le README ?
7 Sent: [Tue Jan 15 14:35:16 CET 2019] Presque, j'ai travaillé toute la nuit dessus !
8 Received: [Tue Jan 15 14:35:27 CET 2019] Impressionnant, quelle abnégation !
9 ----- FIN session de conversation :Tue Jan 15 14:35:31 CET 2019-----
```