INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
TOULOUSE

Département de Génie Électrique & Informatique

Rapport Projet Clavardage

Programmation Orientée Objet

14 février 2021

Étudiants :

Auriane LARTIGUE alartigu@etud.insa-toulouse.fr
Nabil MOUKHLIS moukhlis@etud.insa-toulouse.fr

Enseignant :

Sami YANGUI

Table des matières

Introduction

Lors du processus de création de toute application, et même de tout projet de manière générale, deux étapes au moins sont nécessaires : la conception et l'implémentation. La conception de notre application de clavardage a été détaillée dans le rapport de COO. Nous allons ici plus nous concentrer sur la partie Implémentation de l'application.

Ce rapport s'articule sur plusieurs axes. Dans un premier temps nous détaillerons toutes les manipulations nécessaires pour mettre en place et comprendre notre application et son code.
Dans un deuxième temps, nous dresserons un manuel utilisateur pour permettre à tout utilisateur d'utiliser correctement toutes les fonctionnalités de l'application.
Dans un troisième temps, le but est d'expliquer tous nos choix d'implémentation et ainsi pouvoir faire comprendre au lecteur quelles étaient nos motivations.
Dans un quatrième temps, nous listerons les tests que nous avons effectués sur notre application, les résultats et les problèmes survenus à la suite de cette batterie de tests.
Enfin, dans un dernier temps , nous mettrons en lumière quelques améliorations possibles à notre projet.

Concernant la gestion de notre projet, nous avons utilisé Trello pour gérer notre progression dans le temps et nous répartir les tâches et git pour pouvoir implémenter de nouvelles fonctionnalités de manière parallèle sans affecter le code de l'autre. Aussi, pour développer le projet, nous avons utilisé l'IDE Intellij IDEA Ultimate, gradle pour la gestion des dépendances et JavaFX pour l'interface graphique.

1 Guide Administrateur

1.1 Obtenir les ressources sur GIT

Tout d'abord, il est nécessaire de récupérer les codes source en effectuant la commande git suivante sur un terminal. `git clone https ://git.etud.insa-toulouse.fr/alartigu/ChatApp-AL-NM.git`
Vous avez alors en votre possession différents dossiers :

- Application → L'application (format .jar) et son fichier de configuration
- Implémentation → Code source de l'application
- Javadoc
- Rapport → Le rapport POO, le rapport COO et les diagrammes de conception
- Serveur → Le code source du serveur de présence ainsi que le fichier .war associé
- Readme.md → Document plus détaillé

1.2 Observer la Base de Données

Notre projet nécessitait une base de données. Nous avons donc utilisé un serveur MySQL déployé au département du GEI. Pour y accéder nous vous recommandons les commandes suivantes : `mysql -h srv-bdens.insa-toulouse.fr -u tp_servlet_006 -p` (Un accès au VPN est primordial).

Le système vous demandera alors de rentrer le mot de passe correspondant. Pour utiliser notre base de données, effectuez la commande : `USE tp_servlet_006`
Pour accéder aux différentes tables présentes dans notre base de données vous pouvez utiliser `SHOW TABLES ;`
Et pour finir, si vous souhaitez observer les éléments présents dans une table précise vous pouvez faire : `SELECT * FROM 'Nom de la table' ;`

1.3 Mettre en marche le Serveur de Présence

Notre projet comprend aussi l'installation d'un serveur de présence permettant aux personnes externes de se connecter au réseau. Nous avons donc développé un code à part entière pour celui-ci. Il est possible de le lancer de deux façons.

Utiliser TOMCAT sur votre ordinateur grâce à un IDE : Il est possible de tester localement le serveur sans passer par un gestionnaire d'application web. Il suffit pour cela d'utiliser un IDE qui intègre l'option de déployer un serveur. Nous pouvons citer Eclipse JEE ou IntelliJ IDEA Ultimate. C'est le deuxième que nous avons utilisé lors de ce projet pour effectuer les tests.

Utiliser le Gestionnaire d'applications WEB Tomcat de l'INSA : Le GEI a mis en place un gestionnaire d'application WEB où nous avons pu déposer le code du serveur sous format .war. Pour le lancer, il suffit d'appuyer sur démarrer et de vérifier que la colonne Fonctionnelle est bien à 'true'.

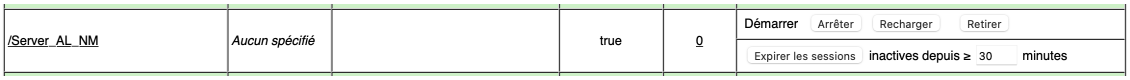

FIGURE 1 – Gestionnaire d'applications WEB Tomcat

Lorsque vous cliquez sur `'/Server_AL_NM'` ou tapez cette URL dans un nouvel onglet, la fenêtre suivante devrait s'ouvrir. Le serveur est en fonctionnement.

FIGURE 2 – Serveur est en fonctionnement

1.4 Fichier de Configuration

Etant donné que notre projet nécessite un serveur de présence ainsi que d'une base de données, différentes variables sont associées à ces éléments notamment une URL. Nous avons donc décidé de créer un fichier de configuration 'config.json'. Celui-ci doit toujours être dans le même répertoire que le code-source au format .jar. Il vous est donc possible de modifier les variables comprises à l'intérieur.

FIGURE 3 – Fichier de configuration JSON

1.5 Documentation : La JAVADOC

Une description du code sera disponible dans la partie 'Choix d'implémentation'. Il est aussi possible d'obtenir des détails supplémentaires grâce à la Javadoc. Notre GIT contient un dossier Javadoc, pour obtenir la liste des packages vous pouvez cliquer sur le document 'index-all.html' qui vous ouvrira un onglet avec la liste suivante :

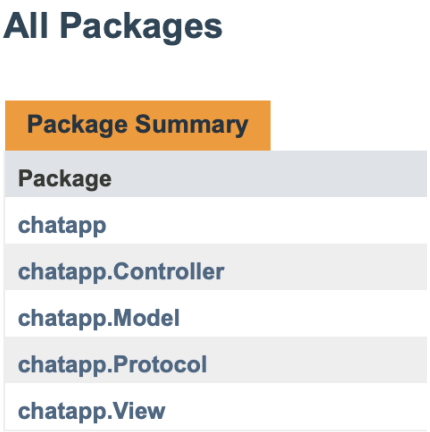

FIGURE 4 – Packages Javadoc

Notre architecture suit le motif 'Modèle-vue-contrôleur', d'où les noms des packages.

2 Manuel Utilisateur

2.1 Lancement de l'application

Sur MacOS ou Windows, deux possibilités s'offrent à vous pour lancer l'application. La première consiste à taper la commande suivante sur un terminal : `java -jar chatapp-1.0-SNAPSHOT-all.jar`
Il est possible de cliquer directement sur le .jar .
Le serveur de présence doit être actif , cf section "Mettre en marche le Serveur de Présence" pour l'activer, et une connexion vpn doit être établie.

2.2 Connexion

Au lancement de l'application, une fenêtre s'ouvre. Il vous est demandé de renseigner un pseudonyme. Pour le valider il vous suffit d'appuyer sur le bouton connexion ou bien d'utiliser le raccourci clavier entrée. Au bout d'un laps de temps défini à 2 secondes vous serez redirigé vers le menu si votre pseudonyme est bon.

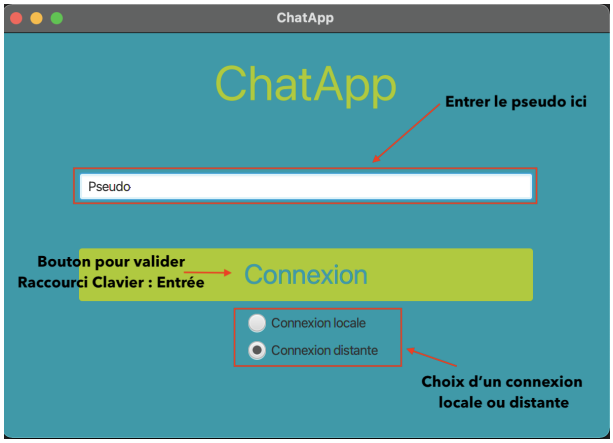

FIGURE 5 – Ecran de connexion

Dans le cas où le pseudonyme est déjà utilisé, vous recevrez un popup d'erreur. Celui-ci indiquera que le pseudonyme est déjà utilisé. Vous aurez la possibilité de rentrer un nouveau pseudonyme par la suite.

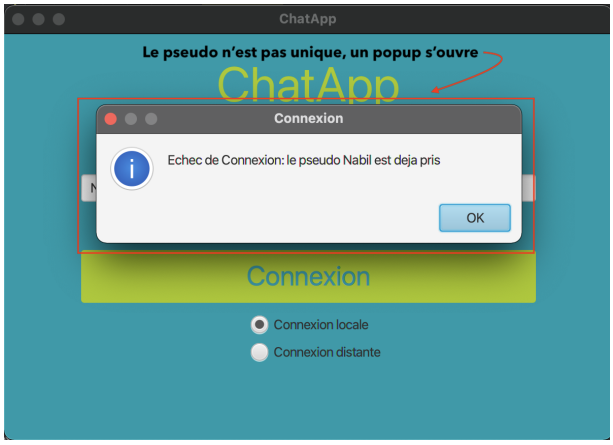

FIGURE 6 – Popup : Erreur dans le choix du pseudonyme

La taille maximale d'un pseudonyme est de 10 caractères, un popup s'affiche si cette limite n'est pas respectée.

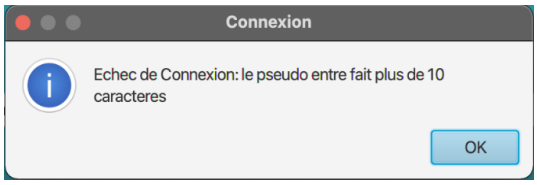

FIGURE 7 – Popup : Erreur dans la taille du pseudonyme

2.3 Menu

Dans le menu principal, un message de bienvenue vous indiquera votre pseudonyme actuel. Vous aurez alors accès à une barre de menu en haut à gauche vous permettant d'effectuer les actions suivantes :

— Connaître les utilisateurs actuellement actifs
— Démarrer une session de clavardage avec l'un d'entre eux
— Changer votre pseudonyme
— Vous déconnecter

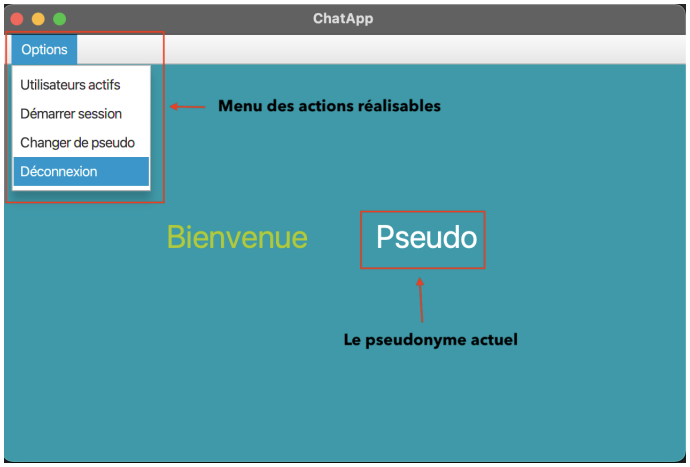

FIGURE 8 – Ecran Principal

2.4 Déconnexion

Un utilisateur souhaitant se déconnecter possède deux options :

— Soit cliquer sur la croix rouge en haut à gauche pour les MACs, ou à un autre emplacement suivant le système d'exploitation.
— Ou bien appuyer sur l'option déconnexion dans la barre de menu.

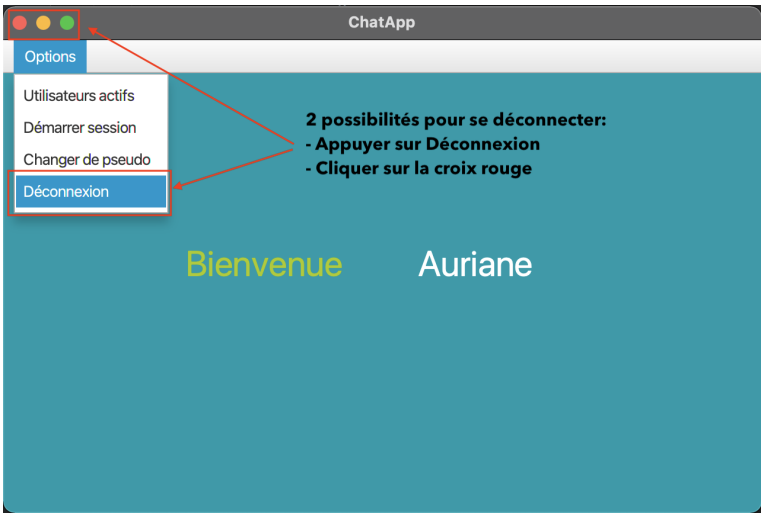

FIGURE 9 – Ecran Principal : Déconnexion

2.5 Changement de Pseudonyme

En choisissant l'option Changement de Pseudo, cette fenêtre apparaît. Une zone d'insertion de texte vous permet de rentrer un nouveau pseudo désiré. L'ancien est affiché en haut. Pour valider, vous pouvez appuyer sur le bouton valider ou utiliser le raccourci clavier entrée. A tout moment, il vous est possible de revenir au menu principal en cliquant en bas à droite sur le bouton "Retour au menu".

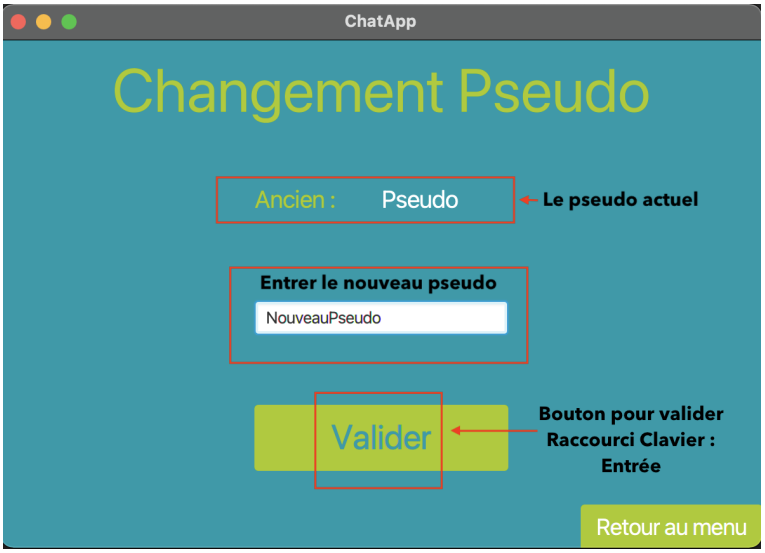

FIGURE 10 – Ecran Modification de Pseudonyme

De nouveau, une vérification d'unicité du pseudonyme est effectuée. Si celui-ci n'est pas unique, c'est-à-dire qu'un autre utilisateur l'utilise, un popup apparaît pour vous avertir. Il est alors à nouveau possible d'entrer un autre pseudonyme.

FIGURE 11 – Popup : Erreur d'unicité du pseudo

2.6 Connaître les utilisateurs actifs

En choisissant l'option "Utilisateurs actifs", la fenêtre suivante s'affichera. Vous observerez alors une liste des utilisateurs actuellement actifs sur le réseau, utilisateurs aussi bien locaux que distants". Le premier pseudonyme de la liste correspond au vôtre. Encore une fois, il vous est possible à tout moment de revenir au menu principal en cliquant en bas à droite sur le bouton "Retour au menu".

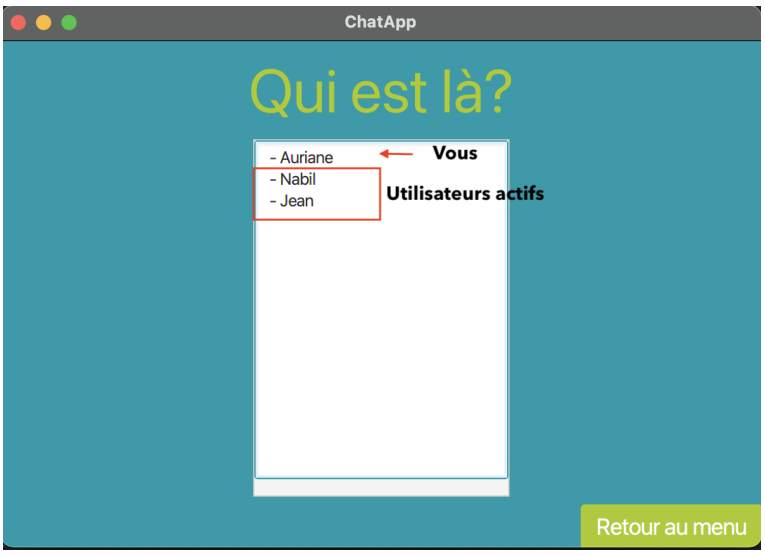

FIGURE 12 – Ecran pour Connaître les utilisateurs actifs

2.7 Clavarder

2.7.1 Choix des utilisateurs

En choisissant l'option "Démarrer Session", une première fenêtre apparaît. Celle-ci vous permet de sélectionner avec quel utilisateur vous souhaitez clavarder. Il vous est possible de clavarder avec vous même, même si cela n'a pas grande utilité. Après votre choix, une nouvelle fenêtre apparaît

en parallèle chez vous ainsi que chez l'utilisateur que vous avez sélectionné. Son nom est retiré de la liste de choix d'un contact.

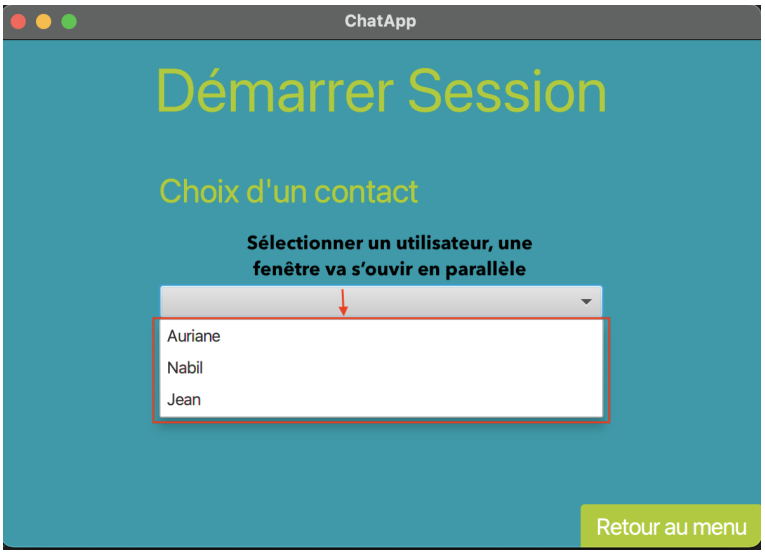

FIGURE 13 – Ecran de démarrage d'une session

2.7.2 Envoyer/Recevoir des messages

Une fenêtre de session de clavardage vous permet d'envoyer et recevoir des messages avec un utilisateur. Il vous est possible de récupérer l'historique des messages avec cet utilisateur en appuyant sur le bouton vert +. La base de données étant stockée sur un serveur distant, une connexion au vpn de l'insa est, nous le rappelons, indispensable. Une zone d'insertion de texte est présente en bas de l'écran. Pour envoyer le message il suffit d'appuyer sur "envoyer" ou d'utiliser le raccourci clavier entrée.

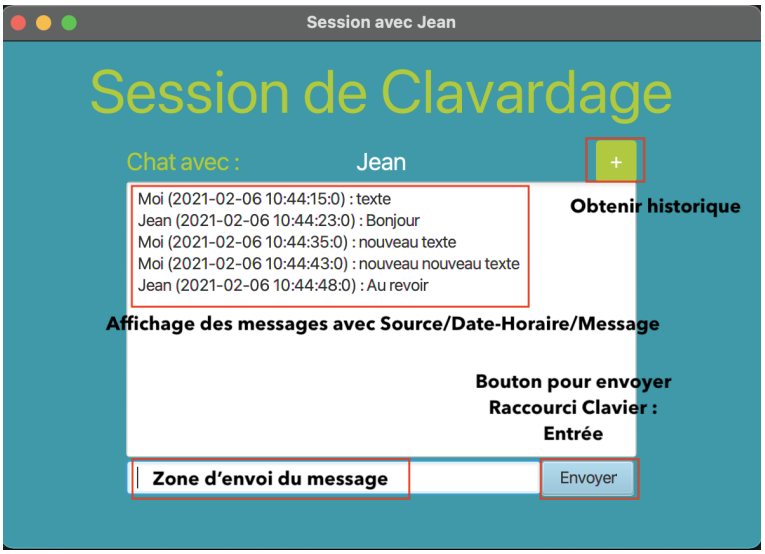

FIGURE 14 – Ecran de Clavardage

3 Choix d'implémentation

3.1 ChatAPP : MVC + P

Pour l'implémentation de notre application, nous avons choisi de suivre le modèle **MVC** traditionnel (Model View Controller) auquel nous avons ajouté un package Protocol.

Le package **Model** contient toutes les classes nécessaires à la gestion des données. On retrouve ainsi des classes pour modéliser les utilisateurs, les listes d'utilisateurs et les messages horodatés mais aussi une classe pour gérer la base de données. Il est à noter qu'un utilisateur dispose d'un ID unique. Celui-ci étant basé sur l'adresse IP de l'utilisateur, nous supposons que chaque utilisateur n'a qu'une seule adresse IP et que chaque adresse IP n'est utilisée que par un seul utilisateur. La base de données étant distante, cette classe a pour but de formuler les requêtes nécessaires pour mettre à jour la base de données ou récupérer des données.

Le package **View** contient toutes les classes nécessaires à l'interface graphique. Pour l'interface graphique, nous avons choisi d'utiliser JavaFX. Cette technologie utilise deux types de fichier : des fichiers XML qui s'occupent de gérer la partie purement graphique et esthétique de chaque fenêtre et des fichiers Java qui permettent de faire le lien entre l'interface graphique et le reste de l'application. Ce sont ces derniers fichiers que l'on retrouve dans le package View, les fichiers XML étant à part dans un dossier "ressources" distinct.

Le package **Controller** a pour but de faire le lien entre tous les packages. Il n'est composé que d'une unique classe ChatApp qui va se charger de lancer les différents threads et consulter les différents éléments du Model.

Le package **Protocol** que nous avons rajouté rassemble toutes les classes permettant à notre application de communiquer avec le réseau. La classe UDPEchange gère les communications en UDP, cela correspond à tous les échanges sur le réseau local quand il s'agit de se connecter au réseau, de modifier son pseudo ou de se déconnecter. Elle dispose d'un thread qui va écouter en permanence sur le réseau et ainsi permettre aux différents membres du réseau de répondre aux requêtes des uns et des autres.
La classe **RunnerEcouteTCP** va gérer la réception des requêtes TCP. Les communications TCP sont utilisées pour les sessions de clavardage. Lorsque le thread reçoit une requête, il crée une instance de la classe SessionClavardage et les deux utilisateurs peuvent alors communiquer librement entre eux jusqu'à ce que l'un des membres décide de mettre fin à la communication.
La classe **HttpEchange** permet aux utilisateurs distants de communiquer avec le serveur de présence pour effectuer les tâches de connexion, déconnexion et changement de pseudonyme.

3.2 Serveur de présence : Proxy

Le serveur de présence que nous avons implémenté est un servlet Java. Il permet aux utilisateurs externes au réseau local de pouvoir communiquer avec d'autres utilisateurs. Les utilisateurs distants peuvent se connecter au serveur de présence (et donc au réseau local par extension) à l'aide de requêtes HTTP. Une requête HTTP POST permet aux utilisateurs distants d'effectuer toutes les tâches qui auraient nécessité un envoi en broadcast.
Le serveur est aussi connecté au réseau local, il s'agit d'ailleurs du tout premier utilisateur connecté sur le réseau.
Le serveur peut alors tenir une liste de tous les utilisateurs connectés localement en même temps. Il a alors deux listes : une des utilisateurs distants (càd ceux s'étant connectés à l'aide de la méthode POST) et une des utilisateurs locaux (càd ceux s'étant connectés à l'aide d'un message UDP en broadcast).

Lorsqu'un nouvel utilisateur se connecte, qu'il soit distant ou local, le serveur va prévenir les

autres membres connectés. Si c'est un utilisateur distant qui se connecte, tous les autres membres sont prévenus de son arrivée par un message UDP (unicast pour les membres distants et broadcast pour ceux sur le réseau). Si c'est un utilisateur local qui se connecte, le message n'est relayé qu'aux membres distants.

Un utilisateur distant A ne peut communiquer directement en TCP avec un autre utilisateur, puisqu'ils ne seront pas sur le même réseau. C'est alors que le serveur rentre en jeu en faisant office de proxy. C'est-à-dire que lorsque A veut communiquer avec B les messages vont transiter par le serveur qui transmettra ensuite le message à B. Ce mécanisme est réalisable du fait que le serveur aura au préalable indiqué son adresse IP à la place de celle des utilisateurs. En d'autres termes, du point de vue d'un utilisateur distant, tous les utilisateurs (qu'ils soient distants ou locaux) auront pour adresse IP celle du serveur. Du point de vue des membres locaux, seuls les utilisateurs distants auront comme adresse IP celle du serveur.

3.3 Base de Données

Notre base de données permet de stocker une liste des utilisateurs du réseau, ainsi que les historiques des échanges qu'ils ont entre eux.

3.3.1 Une table Utilisateurs

Nous avons créé une table, nommée `'Utilisateurs'`, qui stocke les informations de chacun des utilisateurs. Lors de leur première inscription nous retenons leur pseudonyme, ainsi que leur ID (nom d'hôte déterminé à partir de l'adresse IP). Chaque ligne de la table est donc composée :

- d'un Pseudo unique
- d'un Identifiant unique
- d'une Date d'inscription (automatique lors de la première inscription)
- d'un Boolean "Actif" indiquant si il est actif ou non (mis à jour à chaque connexion/déconnexion)

Lorsqu'un utilisateur utilise notre application et possède la même adresse IP que lors de sa dernière connexion, nous mettons à jour son pseudonyme si celui-ci a été modifié. La date d'inscription nous était surtout utile pour implémenter nos tests, nous pouvions ainsi vérifier si les inscriptions étaient bien effectuées et les données modifiées par la suite. Il en va de même pour l'attribut Actif.

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut
1	**ID**	varchar(100)	utf8_general_ci		Non	*Aucun(e)*
2	**Pseudo**	varchar(100)	utf8_general_ci		Non	*Aucun(e)*
3	**Actif**	tinyint(1)			Non	*Aucun(e)*
4	**Inscription**	timestamp			Non	CURRENT_TIMESTAMP

FIGURE 15 – Table des Utilisateurs de l'application

3.3.2 Une table pour chaque échange

Le cahier des charges spécifie que chaque conversation entre utilisateurs doit être conservée et affichée à chaque nouvelle session de clavardage. Chaque utilisateur possède un ID, nous avons donc créé pour les sessions de clavardage entre deux utilisateurs, possédant comme identifiant respectivement Id1 et Id2, une table, `Chat_Id1_Id2`, pour stocker les échanges.
Chaque échange est composé de :

- l'Id du destinataire du message
- Id de celui qui envoie le message
- le corps du message envoyé

— une date à laquelle la Source a envoyé le message

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut
1	**Destinataire**	varchar(100)	utf8mb4_general_ci		Non	*Aucun(e)*
2	**Source**	varchar(100)	utf8mb4_general_ci		Non	*Aucun(e)*
3	**Envoi**	timestamp		on update CURRENT_TIMESTAMP	Non	CURRENT_TIMESTAMP
4	**Message**	text	utf8mb4_general_ci		Non	

FIGURE 16 – Table pour stocker l'historique des échanges

4 Tests réalisés et Problèmes rencontrées

Nous allons essayer dans cette partie de répertorier les cas d'utilisation les plus fréquents de notre application ainsi que d'expliquer les problèmes auxquels nous avons pu faire face. Lors de notre phase de tests, nous avons utilisé des configurations impliquant jusqu'à trois appareils. Pour plus de lisibilité, nous ne détaillons que les cas avec deux utilisateurs.

Connexion de 2 utilisateurs locaux : Pour cette partie, nous avons connecté sur un réseau local deux utilisateurs en même temps en mode "local". Nous avons testé la connexion de chacun, c'est-à-dire que nous avons vérifié si lorsqu'un utilisateur se connecte, l'autre utilisateur est prévenu de son arrivée. Nous avons aussi vérifié que si l'utilisateur A essaye de se connecter avec le pseudonyme de l'utilisateur B, une fenêtre d'avertissement apparaît.

Nous avons aussi testé le changement de pseudo lors d'une session de clavardage et en dehors de celle-ci. Nous avons ensuite tenté divers sessions de clavardages avec différentes configurations incluant des requêtes pour accéder à l'historique des messages.

Un des membres de notre binôme n'a pas réussi à lancer une session de clavardage depuis son ordinateur MacOS. L'erreur survenue n'apparaissait que lorsqu'une connexion sur le VPN était lancée, dans le cas contraire tout fonctionner normalement. Le problème était dû à l'expiration du timer lors de la création d'une socket. Nous avons supposé que le problème venait du pare-feu de l'appareil mais n'avons pas réussi à déboguer ce souci technique.

Connexion de 2 utilisateurs distants : Nous avons repris le même processus de tests que précédemment pour pouvoir tester toutes les fonctionnalités.

Lorsqu'un utilisateur se connecte en étant extérieur au réseau, il communique à l'aide d'une requête HTTP au serveur de présence pour le notifier de son arrivée. Le serveur lui répond en lui communiquant toujours à l'aide d'une réponse HTTP la liste des utilisateurs déjà actifs sur le réseau. Il transmet en unicast aux utilisateurs distants l'information de cette nouvelle arrivée et en broadcast aux utilisateurs locaux. Cette partie était fonctionnelle nous pouvions remarquer que le proxy avait bien mis son adresse IP à la place de celle de l'hôte distant.

Concernant les sessions de clavardage nous avions le droit à un crash total de l'ordinateur lorsque nous faisions nos tests finaux à l'INSA alors qu'il n'y avait aucun problème lorsque nous testions dans un autre réseau.

Connexion d'1 utilisateur local et 1 distant : Lorsqu'un utilisateur distant se connecte, le serveur de présence doit prévenir les membres locaux et distants de l'arrivée de celui-ci mais aussi indiquer à ce nouvel arrivant une liste des membres actuellement présents. Pour prévenir les utilisateurs, le serveur utilise le protocole UDP. Cependant une fois connectés en VPN, le serveur ne semble pas accessible en broadcast. C'est aussi le cas lorsque l'on est connecté au réseau grâce à une machine du GEI. Notre proxy n'est donc pas utilisable sur le réseau de l'INSA.

Nous avons effectué ces tests dans un réseau hors de l'INSA, sans VPN et donc de base de données. Les sessions de clavardage étaient possibles entre les appareils. Lorsque les messages transitent par le serveur nous avions fait en sorte qu'il rajoute un mot au corps du message à transmettre. Ainsi nous pouvions être sûrs durant nos tests que le proxy fonctionnait correctement.

Notre plus gros problème lors de notre processus de tests a été l'impossibilité de tester notre application à l'INSA. Comme nous n'avions pas effectué de séances de TP en présentiel et que nous ne disposions que d'un équipement sommaire, nous avons dû nous contenter d'une phase de tests improvisée dans les locaux de l'INSA entre deux cours en présentiel. Nous avions alors constaté beaucoup de problèmes qui n'étaient pas présents lors de nos tests chez nous et n'avons pas eu le temps de les corriger.

5 Améliorations Futures

Notre projet n'est qu'une première ébauche de ce que pourrait être une application de clavardage optimale bien qu'elle respecte le cahier des charges imposé. De nombreuses fonctionnalités pourraient être modifiées et d'autres pourraient même être ajoutées. La liste est longue mais en voici trois principales :

Changement de pseudonyme :
Dans une fenêtre de clavardage, le nom de celui avec qui nous communiquons est inscrit dans la partie‘ Chat avec : NOM ’ . Lors d'un changement de pseudo , si une session de clavardage est déjà ouverte le nom n'est pas modifié. Nous pourrions résoudre ce problème à l'aide d'un propertyChangeListener.
Il en est de même pour la zone de lecture des messages, celle-ci est de la forme ‘Nom (Date Heure) : Corps du message’, et le nom n'est toujours pas modifié.

Le serveur :
Au vue des problèmes rencontrés, le serveur semble être un point majeur à améliorer, notamment son déploiement sur le réseau de l'INSA. Avec des conditions sanitaires plus avantageuses, nous pourrions comprendre l'architecture réseau de l'INSA et mieux appréhender le système de broadcast/unicast en UDP qui fait actuellement défaut à notre application.

L'interface graphique :
L'interface graphique pourrait être plus attrayante et ergonomique. Notamment, l'impossibilité d'agrandir les fenêtres est un défaut qui mériterait d'être corrigé pour garantir une meilleure expérience utilisateur.

Conclusion

Pour conclure sur notre projet, nous sommes heureux d'avoir pu avoir la chance de travailler sur un tel projet dans le cadre de nos études. Cela a été pour nous l'occasion de travailler de manière concrète sur l'intégralité d'une application que nous avons nous-mêmes conçue. En terme de montée en compétences nous avons pu nous améliorer en gestion de projet et nous former à de nouveaux outils tels que JavaFX ou Gradle.

Nous regrettons cependant que le contexte sanitaire ne nous ait pas permis de pouvoir développer l'application dans les meilleures conditions possibles. Cela nous a handicapés en matière de temps et de moyens, ce qui a impacté notre projet sur sa dernière ligne droite.

Nous sommes tout de même satisfait d'avoir expérimenté un tel projet qui nous a beaucoup apporté en termes de professionnalisme et nous tenons à remercier nos tuteurs, Mme Meryem BILLAMI et M. Samy YANGUI qui nous ont accompagnés durant ce projet sur les parties conception et implémentation respectivement.

Table des figures