INSA
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
TOULOUSE

Rapport de projet COO-POO
Système de clavardage

4IR-SI, groupe A2

Martin EMERY
Marie PONTALIER

TABLE DES MATIÈRES

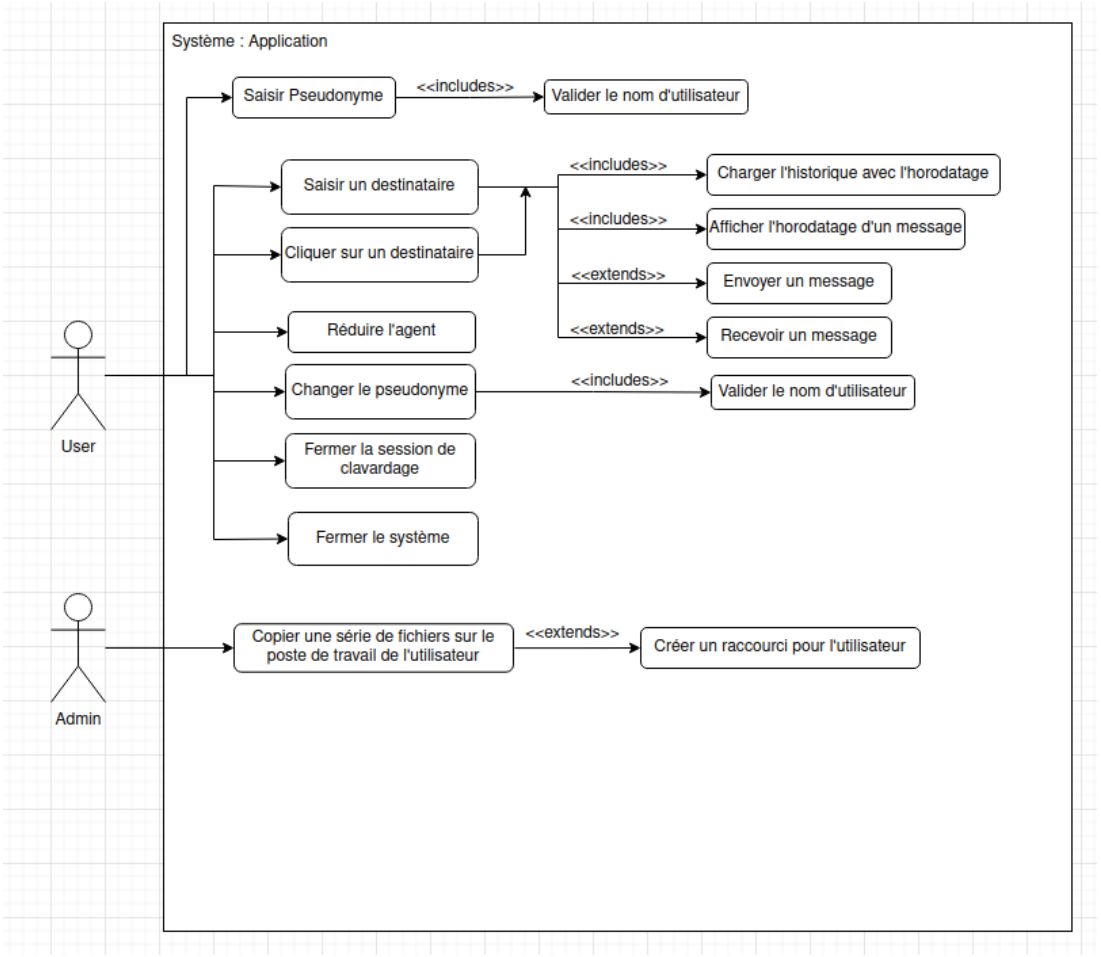
Introduction

Tout au long du semestre, nous avons travaillé sur un projet de système de clavardage. L'objectif de ce projet était de mettre en application nos compétences en programmation orientée objet ainsi qu'en conception UML.
Nous avons ainsi suivi un cahier des charges détaillé afin de concevoir un système de chat permettant à différents utilisateurs de communiquer via une interface graphique simple.
Ce rapport permet d'expliquer comment nous avons conçu et programmé notre système d'après le cahier des charges. Nous faisons notamment l'hypothèse qu'une personne n'utilise qu'un ordinateur et qu'un ordinateur n'a qu'un seul utilisateur.
Nous avons utilisé le logiciel Visual Studio Code pour la programmation en Java et Draw.io pour la conception UML.

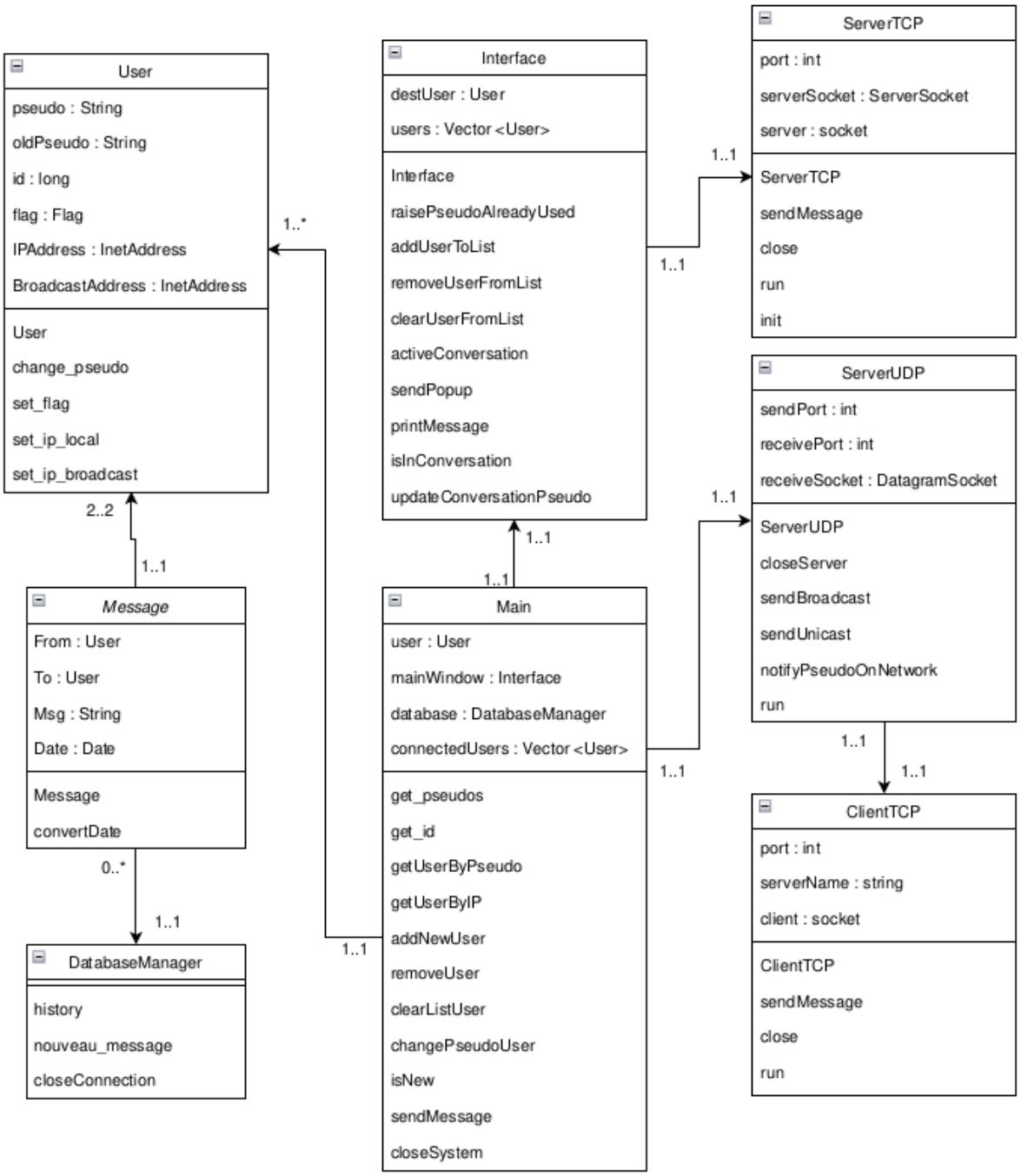
I – Conception Orientée Objet

Nous avons réalisé la conception UML sur le logiciel gratuit en ligne draw.io.
Nous avions construit les premiers diagrammes avant de commencer à programmer, lors des cours dédiés à cela. Cependant, après avoir codé le projet, nos diagrammes n'étaient plus à jour (sauf le diagramme des cas d'utilisation) et nous avons préféré tout recommencer.
Pour les diagrammes de séquence nous avons choisi de modéliser 4 actions qui nous paraissaient pertinentes : la connexion de l'utilisateur (qui reprend aussi le changement de pseudo), le changement de destinataire, l'envoi d'un message et la déconnexion de l'utilisateur.

I.1 – Diagramme des cas d'utilisation

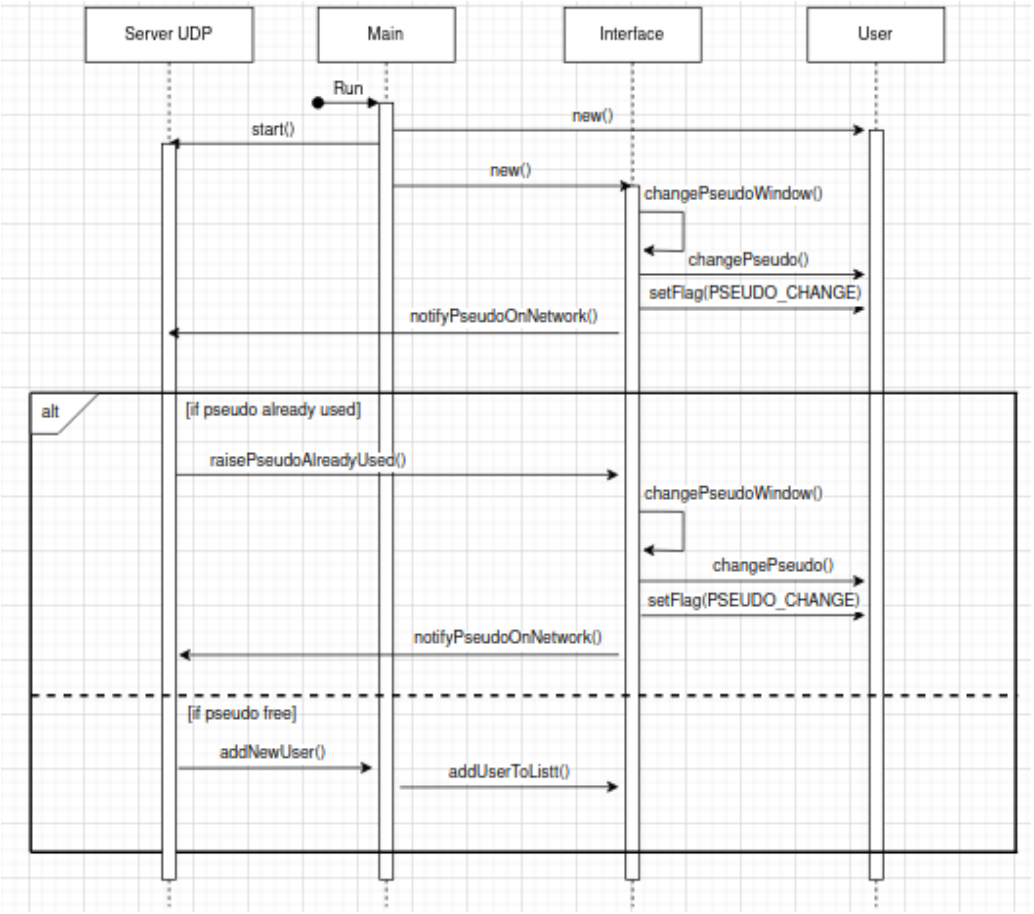


I.2 – Diagramme de classes

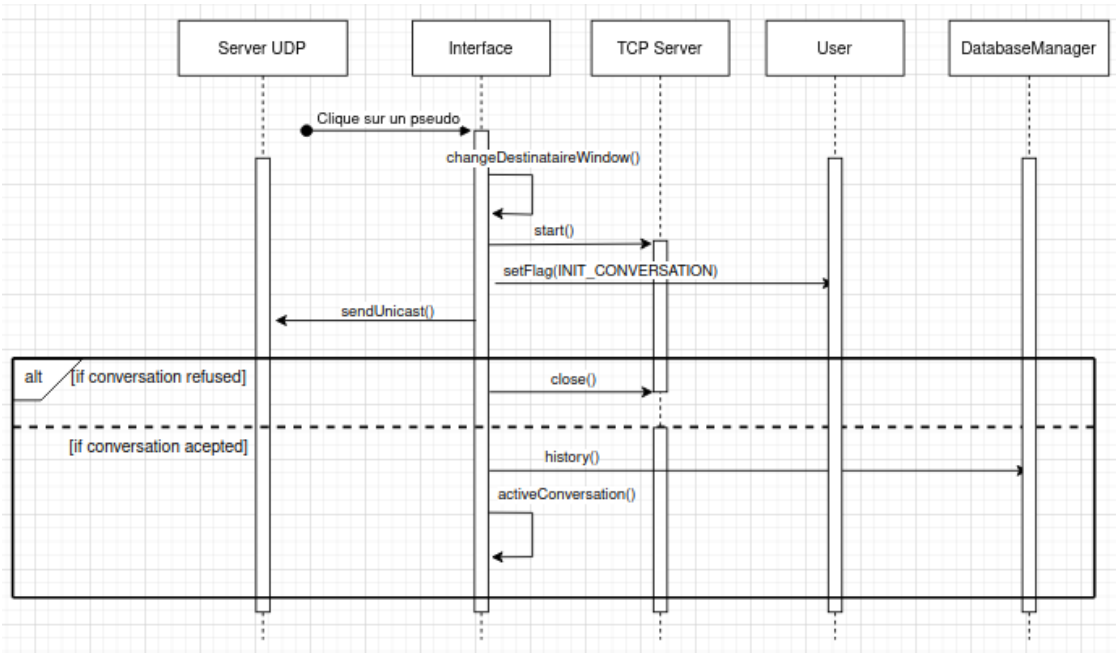


I.3 – Diagrammes de séquence

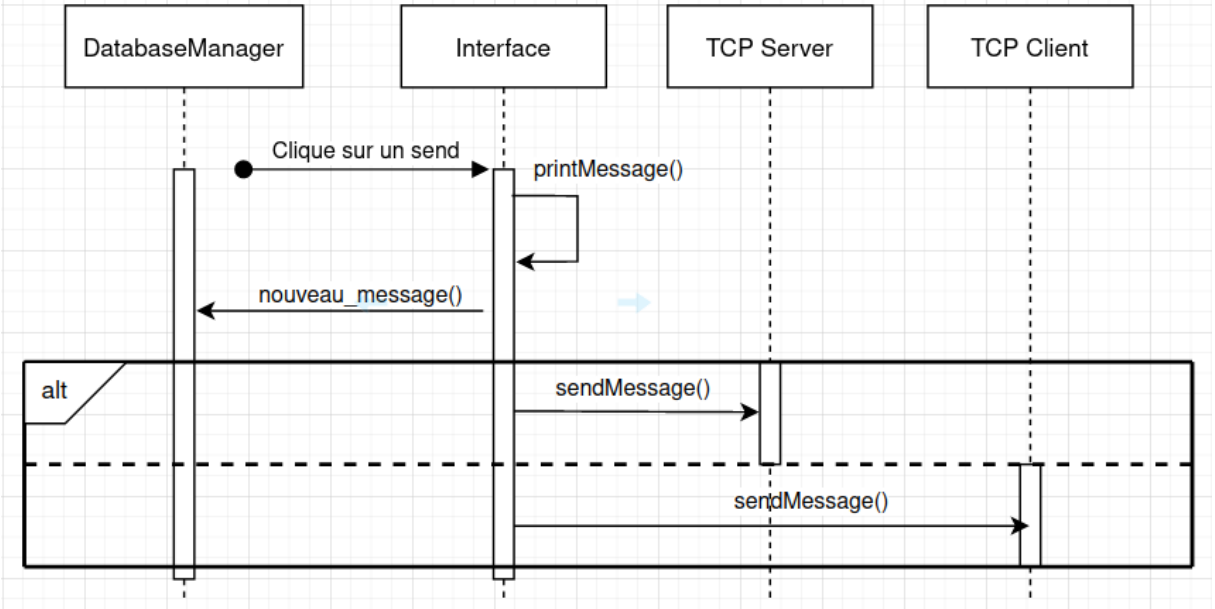
I.3.1 – Connexion



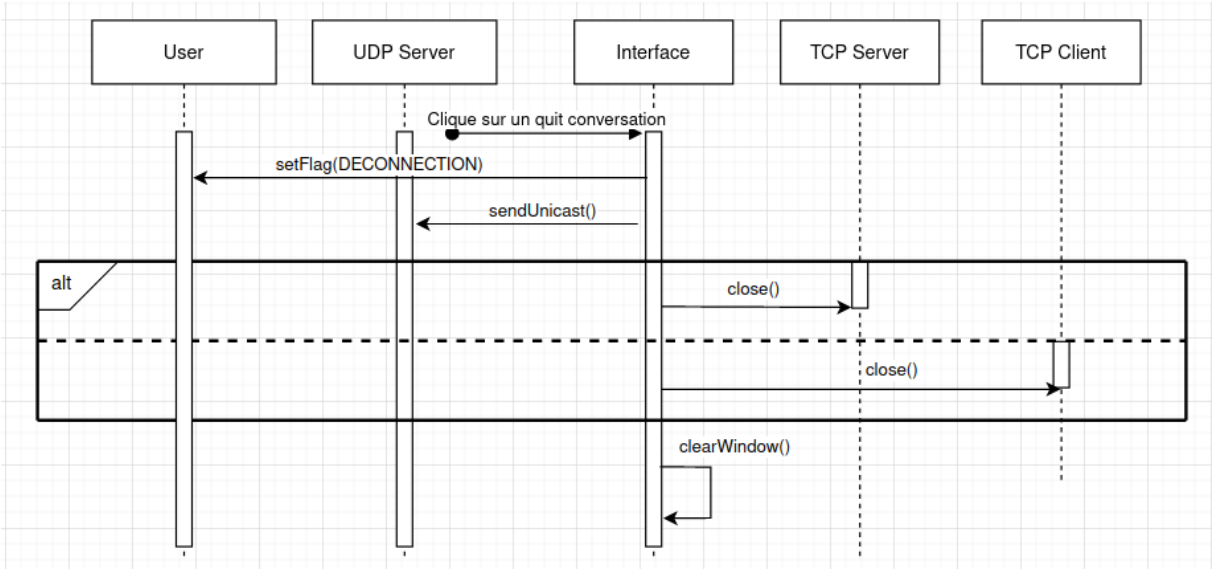
I.3.2 – Changement de destinataire



I.3.3 – Envoi d'un message



I.3.4 – Déconnexion



II – Programmation Orientée Objet

Dans cette partie nous allons détailler les classes utiles pour comprendre le fonctionnement général du système.

II.1 – L'utilisateur

Chaque utilisateur est identifié par plusieurs attributs comme un pseudo, un identifiant unique ou une adresse IP.
Le pseudo est visible par l'ensemble des utilisateurs, tandis que l'id et l'adresse IP ne sont utilisés qu'en interne. Cette dernière sert au routage dans le réseau alors que l'id sert à stocker les messages dans la base de données. L'adresse IP est récupérée depuis le fichier config.xml qui doit être configuré par l'administrateur de l'entreprise. L'adresse de broadcast est la même pour tous les utilisateurs d'un même réseau et doit également être configurée dans le fichier config.xml.
L'id est généré lors de la première exécution du programme et de manière aléatoire dans le fichier caché userData. Lors des autres exécutions, cet id sera juste récupéré dans ce fichier, afin de reconnaître l'utilisateur.
Chaque utilisateur dispose également d'un flag qui permet de signaler différentes actions :

```
PSEUDO_CHANGE,          // Signale un changement de pseudo en cours
PSEUDO_NOT_AVAILABLE,   // Signale que le pseudo demandé est déjà pris
CONNECTED,              // Signale que l'utilisateur envoyé est connecté et a un pseudo unique
DISCONNECTION,          // Signale qu'un utilisateur quitte le réseau
INIT_CONVERSATION,      // Signale la volontée de démarrer une conversation
REFUSE_CONVERSATION,    // Signale le refus de démarrer une conversation
CLOSE_CONVERSATION      // Signale que l'interlocuteur a fermé la conversation
```

II.2 - Message

Un message est un paquet composé de deux utilisateurs (émetteur et destinataire), du message en lui-même (au format texte uniquement) et de la date à laquelle il a été envoyé.
Ce message sera échangé sur le réseau.

II.3 – Interface graphique

L'interface graphique est découpée en 5 parties :
- La liste des utilisateurs connectés : Tous les utilisateurs actuellement connectés au système sont affichés dans cette zone. Si on clique sur un des pseudos, la conversation avec cette personne s'ouvre et elle est sélectionnée comme destinataire de nos messages.
- Notre pseudo : Il est affiché et modifié lorsqu'on clique sur le bouton *Change Pseudo*, ce dernier ouvrant une nouvelle fenêtre pour que l'on puisse entrer notre nouveau pseudo.
- Le destinataire : Permet de savoir quelle conversation est actuellement ouverte (à qui l'on parle). Le bouton *Change Recipient* a le même effet que lorsqu'on clique dans la liste des utilisateurs connectés à ceci près que celui-ci ouvre une fenêtre pour que l'on entre manuellement le nom du destinataire.
- La conversation : C'est là que les messages échangés s'affichent sous la forme
Date Emetteur : Message. Lorsqu'on a choisi le destinataire, l'historique de la conversation avec cette personne est chargé.
- Le message : Il suffit de taper son message dans la zone de texte et d'appuyer sur entrée ou sur le bouton *Send* pour que le message s'envoie (et apparaisse donc dans la zone conversation).

II.4 – Réseau

La programmation du réseau est répartie sur 3 fichiers (ClientTCP, ServerTCP et ServerUDP) et deux protocoles qui fonctionnent en simultané.

II.4.1 Le protocole UDP

Il permet d'échanger des informations entre les utilisateurs. Chaque paquet échangé est de type User et on utilise les flags pour interpréter le message. Lorsque l'on se connecte au réseau, on envoie son User en broadcast avec le flag PSEUDO_CHANGE. Si un utilisateur le reçoit et que son pseudo est différent du sien, il renvoie en unicast son User avec un flag CONNECTED, sinon il renvoie un flag PSEUDO_NOT_AVAILAIBLE.
Quand on se déconnecte du réseau, le flag DISCONNECTION est envoyé.
Enfin, lorsque l'on veut commencer une discussion, on envoie le flag INIT_CONVERSATION. Pour la refuser on répond REFUSE_CONVERSATION et pour l'accepter on se connecte au serveur TCP.

II.4.2 Le protocole TCP

Ce protocole est utilisé pour échanger les messages entre deux utilisateurs. Il nécessite en premier lieu une connexion entre eux. L'utilisateur voulant initier la conversation lance un serveur TCP, celui qui accepte de rejoindre la conversation lance un client qui se connecte au serveur. Une fois la connexion établie, les échanges de messages peuvent avoir lieu.
Si un des deux utilisateurs quitte la conversation, la connexion TCP est interrompue et le second utilisateur est informé. Il pourra continuer d'envoyer des messages qui seront stockés dans l'historique mais ne seront pas reçus.

II.5 – Base de données

La base de données est de type SQL, centralisée et permet de conserver les messages envoyés entre les utilisateurs. Les messages sont stockés dans la table ***msgTable*** avec les id de l'émetteur et du destinataire ainsi que leur horodatage.
La fonction ***nouveau_message*** permet d'ajouter un message à la table à chaque fois qu'un message est envoyé sur le réseau.
La fonction ***history*** récupère dans la base de données tous les messages envoyés par l'un ou l'autre des deux utilisateurs passés en argument en les classant par date d'envoi.

La fonction d'affichage de la base de données n'est utile que pour effectuer les tests et n'est donc pas utilisée lors du fonctionnement du système.

```
[Database] Table msgTable :
-----------------------------
|  Emetteur  |  Destinataire  |             Message                |              Date              |
-----------------------------
| 733686242  | 989163925      |    Salut !                         |  jeu. janv. 27 18:03:37 CET 2022    |
| 989163925  | 733686242      |    Hey ! Ca va ?                       |  jeu. janv. 27 18:03:49 CET 2022    |
| 733686242  | 989163925      |    Oui et toi ?                        |  jeu. janv. 27 18:03:55 CET 2022    |
| 989163925  | 733686242      |    Super !                         |  jeu. janv. 27 18:04:05 CET 2022    |
| 733686242  | 989163925      |    Bonne journée !                     |  jeu. janv. 27 18:04:25 CET 2022    |
| 989163925  | 733686242      |    Toi aussi !                         |  jeu. janv. 27 18:04:31 CET 2022    |
-----------------------------
```

III – Guide d'utilisation

Le système a été conçu sous Visual Studio Code, il est donc préférable d'utiliser cet IDE pour le lancer. Nous avions créé un fichier .jar et un script bash pour faciliter l'utilisation mais, comme une configuration est nécessaire, nous ne pouvons pas encore l'utiliser. Une amélioration du programme serait donc de rendre la configuration automatique.

III.1 – Installer le système

L'installation du système nécessite seulement une configuration lors de la première utilisation.

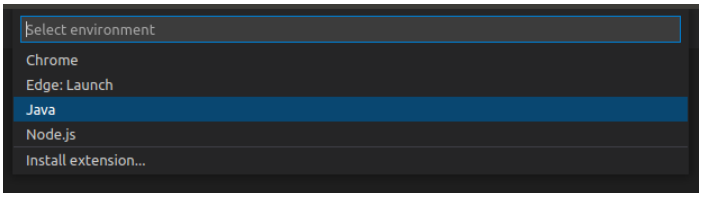
Télécharger tous les fichiers
- Aller à l'adresse https://github.com/martemr/Project_chat_system
- Cliquer sur Code – Download ZIP
- Extraire les dossiers et ouvrir le fichier config.xml

Mettre les adresses à jour
- Ouvrir un terminal (Ctrl+Alt+T)
- Taper la commande ifconfig
- Copier l'adresse inet (ip) de format X.X.X.X
- Dans le fichier config.xml, coller l'adresse inet entre les <ip_local>
- Retourner dans le terminal
- Copier l'adresse broadcast de format X.X.X.255
- Coller cette adresse entre les <ip_broadcast>

```
eth4: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.1.5.50  netmask 255.255.0.0  broadcast 10.1.255.255
```

```
config.xml
1  <user>
2      <ip_local>10.1.5.50</ip_local>
3      <ip_broadcast>10.1.255.255</ip_broadcast>
4  </user>
```

Lancer le programme
- Ouvrir le dossier Project_Chat_System avec Visual Studio Code (Commande : code Project_chat_system)
- Depuis le logiciel, lancer l'exécution : Run (F5)
- Sélectionner l'environnement java

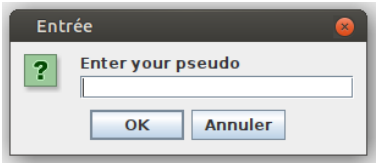


Le programme se lance. Bienvenue !

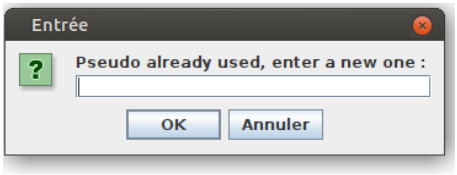
III.2 – Utiliser l'application

Changer de pseudo

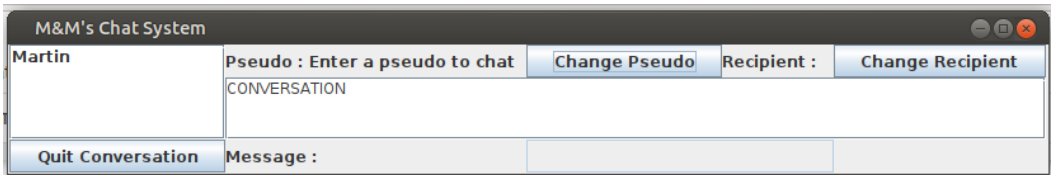
Cliquez sur le bouton Change Pseudo, une fenêtre s'ouvre. Entrez votre pseudo puis appuyez sur OK ou la touche entrée.



Si quelqu'un utilise déjà le pseudo que vous avez choisi, il vous est demandé de rentrer un nouveau pseudo.



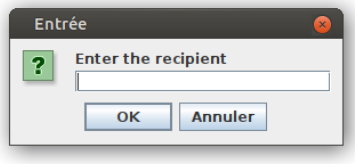
Si vous ne rentrez pas de pseudo vous ne pourrez pas envoyer de message.

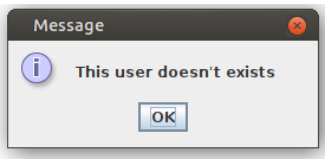


Changer de destinataire

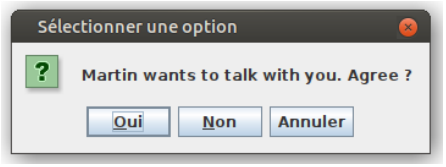
2 options sont proposées pour changer de destinataire :

- Cliquez sur un pseudo dans la liste des utilisateurs connectés.
- Cliquez sur le bouton Change Recipient, une fenêtre s'ouvre. Entrez le pseudo du destinataire puis appuyez sur OK ou la touche entrée. Si le destinataire n'existe pas, il vous faut recommencer.





Votre destinataire reçoit une demande de connexion. S'il accepte, vous parlez en direct. S'il refuse, vous pouvez toujours lui envoyer un message mais il ne s'affiche pas sur son écran tant qu'il ne souhaite pas vous parler.



Envoyer un message

Pour réaliser cette étape il est nécessaire d'avoir effectué les 2 premières.

Tapez votre message dans la zone de texte puis cliquez sur le bouton Send ou sur la touche entrée.

Le message s'affiche dans la zone de conversation.

Quitter une conversation

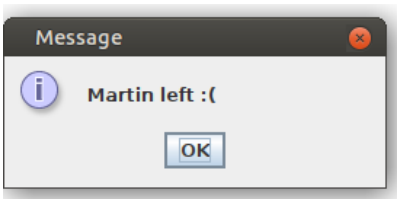
Cliquez sur le bouton Quit Conversation.

Votre interlocuteur en est informé via une popup. Il peut continuer à vous envoyer des messages mais vous ne les verrez pas tant que vous n'aurez pas décidé de rouvrir la conversation.

Se déconnecter

Cliquez sur la croix rouge en haut à droite de l'application.

Les autres utilisateurs sont informés de votre départ via une popup. Ils ne peuvent plus vous envoyer de messages.



IV – Processus de développement logiciel automatisé

Nous n'avions pas compris que l'utilisation de Jenkins était obligatoire donc nous ne l'avons pas utilisé.
Nous avons effectué les tests manuellement au fur et à mesure du projet.

V – Gestion de projet

Lien du git : https://github.com/martemr/Project_chat_system
Lien Jira : https://pontalier.atlassian.net/jira/software/projects/MMS/boards/2/backlog

La création du backlog et des sprints a été assez compliquée à réaliser. En effet, c'était la première fois que nous réalisions un projet d'une telle ampleur et nous n'avions pas bien conscience du travail qu'il nous faudrait effectuer.
Tout d'abord, nous ne savions pas par où commencer car c'était la première fois que nous partions de rien. Par la suite nous avons rencontré des problèmes auxquels nous n'avions pas pensé et au contraire certains points qui nous paraissaient très ardus se sont en fait révélés assez évidents. Nous avions donc fait des erreurs lors de l'estimation des difficultés avec la méthode poker.
Ci-dessous, quelques exemples de nos sprints.

Date	Événement	Ticket
Mon, Dec 06 2021, 8:42am	Sprint démarré	MMS-31 Create class Message MMS-20 Create a timestamp function MMS-30 Create class User MMS-38 Create a function for set thepseudo MMS-33 Create class Database MMS-18 Make requests on a SQL database MMS-21 Design a first version of the interface
Tue, Dec 14 2021, 6:11pm	Sprint démarré	MMS-28 Create a function that verify a username MMS-19 Create a function which searches in the database MMS-32 Create class Thread MMS-53 Create conversation field for printing messages MMS-54 Timestamp messages MMS-55 Print all messages send and receive MMS-52 Create message field MMS-47 Create functions for requests in database MMS-49 Create table pseudo MMS-51 Associate an unique id with an User MMS-22 Create a connected status MMS-61 Connect to Insa's database MMS-62 Create database
Tue, Jan 04 2022, 8:42am	Sprint démarré	MMS-40 Notify the pseudo changement MMS-27 Create a function that display a message MMS-64 Creer methode de vérification de l'existence d'un pseudo MMS-65 Créer une fenetre pop-up ou autre forme pour les erreurs (pseudos deja pris, etc.) MMS-66 Afficher la liste des utilisateurs MMS-67 Lier base de données à l'interface MMS-48 Make a link between program and database SQL MMS-50 Create table for messages (know which columns are necessary) MMS-69 Start connection tcp when clicking on the pseudo MMS-70 Make an exchange between two user MMS-71 List the connected users MMS-72 Click on a user to communicate with him MMS-73 Lire la date depuis la base de données

INSA Toulouse
135, avenue de Rangueil
31077 Toulouse Cedex 4 - France
www.insa-toulouse.fr

Université Fédérale Toulouse Midi-Pyrénées
Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
MINISTÈRE
DE L'ÉDUCATION NATIONALE,
DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE