INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Conception d'un Système de Chat distribué

Conception Orienté Objet

Rédigé par : Walid EL-ASSIMY & Ghizlane BADAOUI

Groupe 4A IR B1

- Rapport du 2022/2023 -

Sommaire

1 Introduction

"Chat System" est une application qui permet aux utilisateurs appartenant au même réseau local d'échanger des messages textuels.

** Si les diagrammes ne sont pas assez clairs, vous les trouverez dans le dossier "/rendu/diagrams" sous format pdf.*

2 Acteurs

2.1 Acteurs primaires

- User : il représente l'utilisateur principal du système, et il joue le rôle d'un agent au sein de l'application.

2.2 Acteurs secondaire

- Administrator : il s'occupe de la configuration et le déploiement de l'application dans les différents supports informatiques.
- User : est l'utilisateur qu'échange avec l'utilisateur principal "User" et qui interagit avec lui. Il a les mêmes propriétés et peut faire les mêmes fonctionnalités que le "User" principal.

3 Diagramme des cas d'utilisation

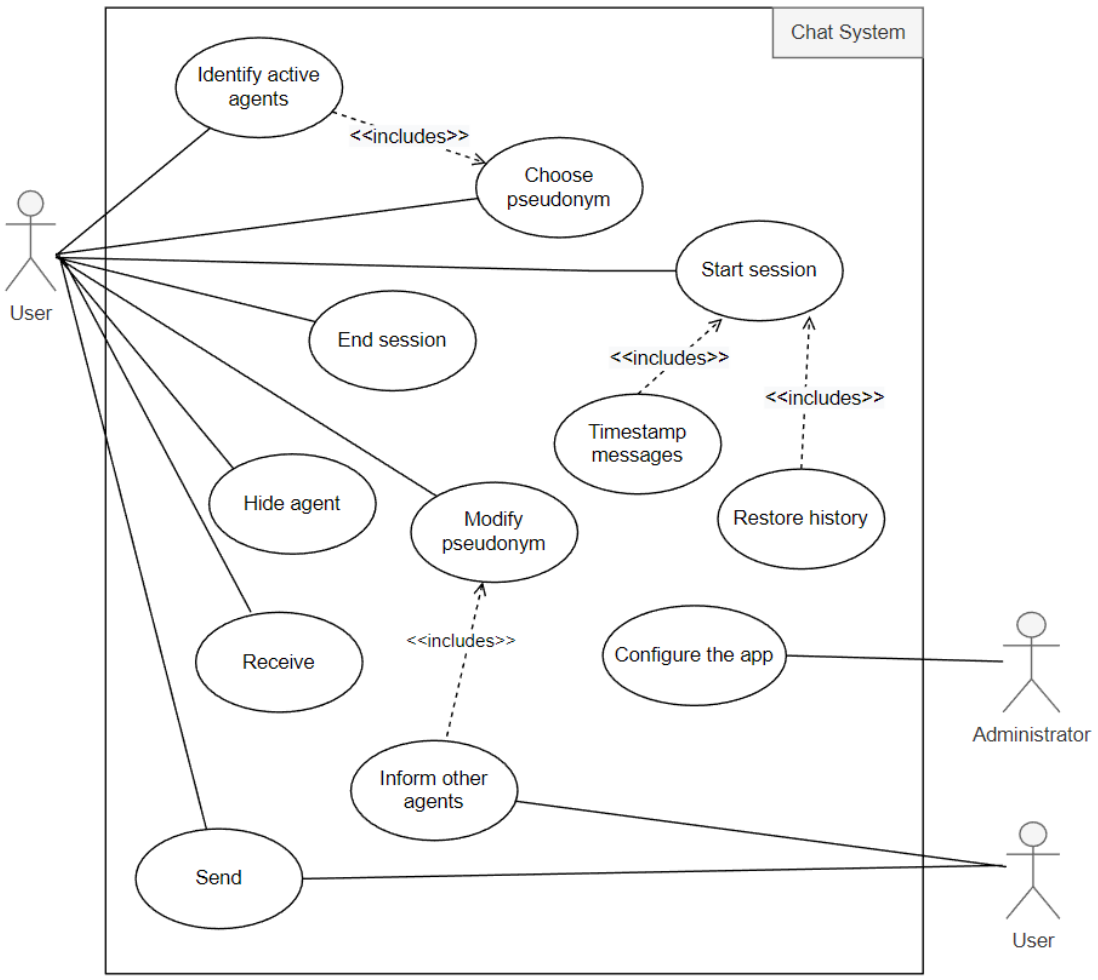
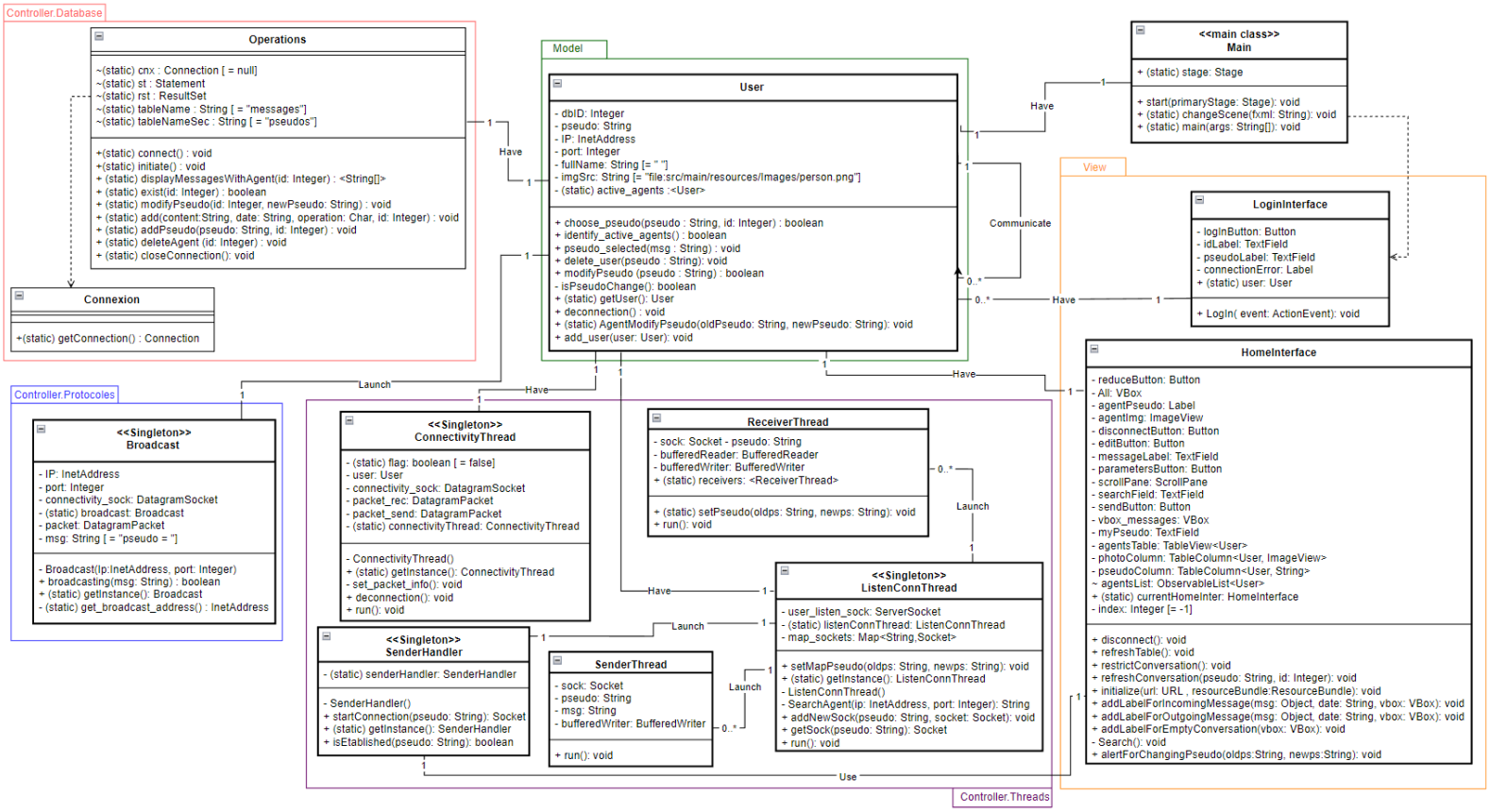


figure 1 : Diagramme de cas d'utilisation de "Chat System"

4 Diagramme des classes

4.1 Schéma



** Les classes actives ont une bordure en gras (les seules classes passives sont Connexion et Operations)*
** Les attributs/méthodes de classes ont une mention "(static)" (l'outil utilisé ne permet pas le soulignement)*
** Les getters et les setters n'ont pas été ajoutés à ce diagramme.*

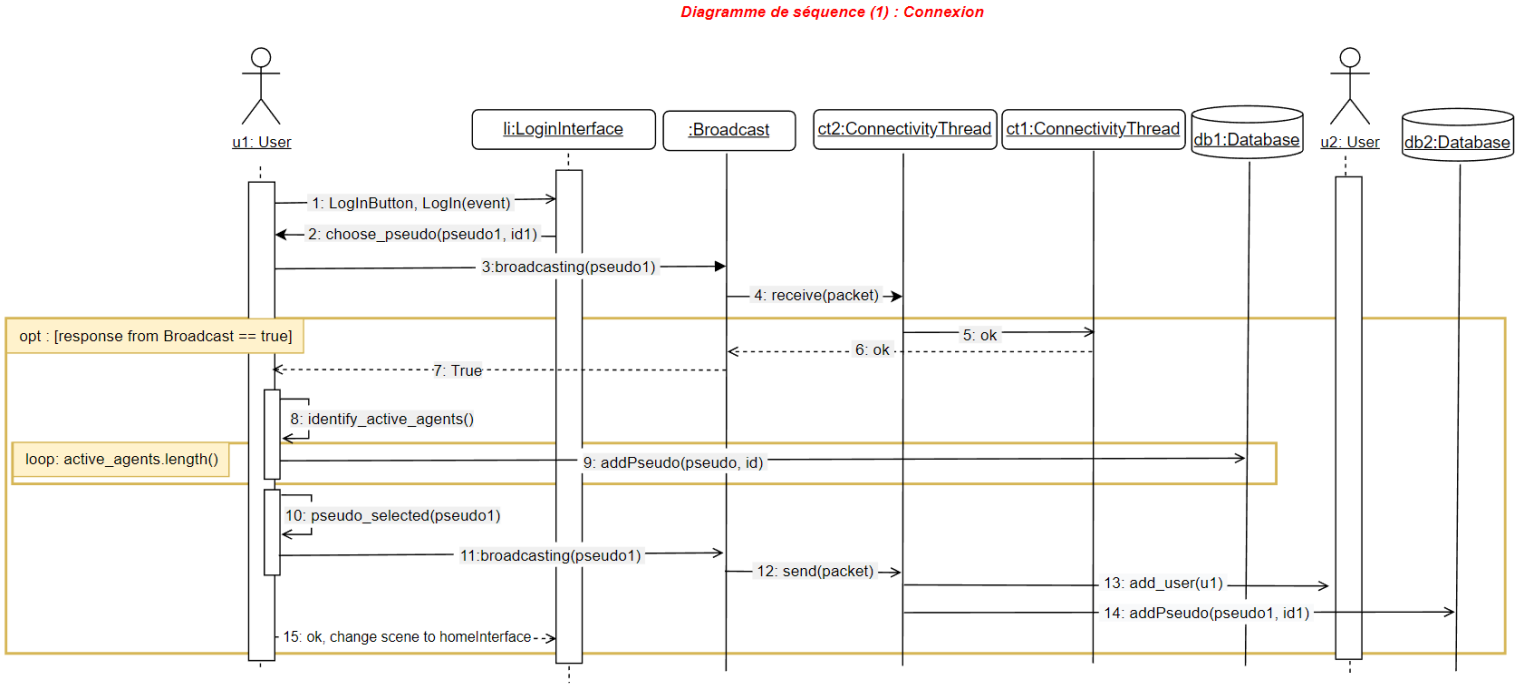
figure 2 : Diagramme de classe du "Chat System"

4.2 Détails et typage

- Les classes Broadcast, ConnectivityThread, ListenConnThread et SenderHandler représentent des design pattern de type Singleton, et elles sont présentes dans tout le projet en une seule instance pour un utilisateur.
- Les types des variables, des paramètres d'entrée et des valeurs de retour des méthodes de chaque classe, sont cités dans le diagramme de manière explicite.
- Les classes sont organisées dans 3 packets : **M**odel, **V**iew et **C**ontroller (cf. Architecture du système)
- Quelques détails supplémentaires :
    - Toute variable de type <TYPE> est une "liste de TYPE".
    - Toute variable de type InetAddress est une adresse IP.
    - La valeur entre crochets devant certaines variables est une valeur initiale (par défaut) de cette variable.
    - Les attributs de LoginInterface et HomeInterface sont des objets graphiques de JavaFX.

5 Diagrammes de séquence

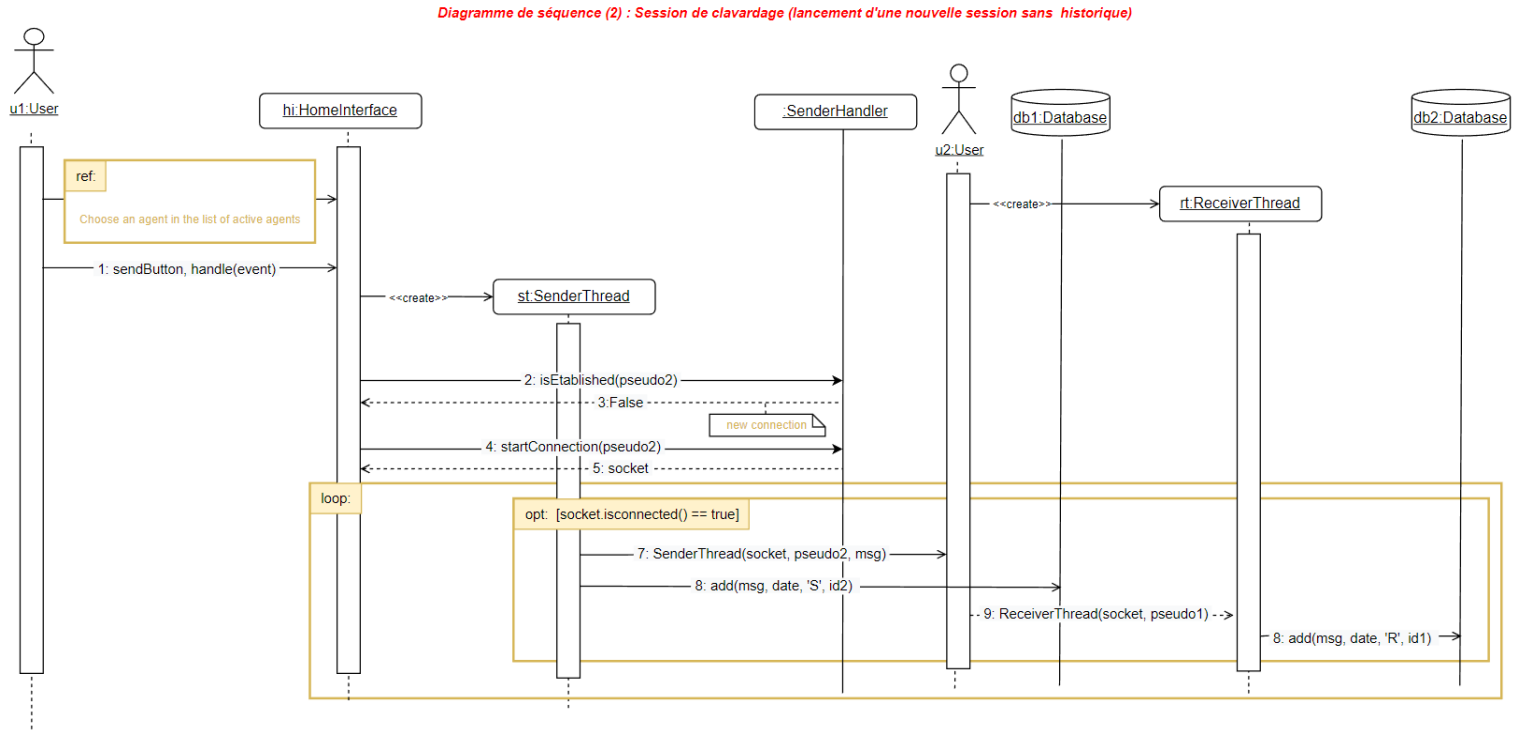
5.1 Diagramme 1 : Connexion



** db1 (Database), ct1 (ConnectivityThread), id1 et pseudo1 concernent l'utilisateur u1, et db2 et ct2 concernent l'agent u2.*
** La boucle (étape 9) ajoute les agents de la liste active_agents un par un à la base de données db1.*

figure 3 : Diagramme de séquence pour une connexion d'un utilisateur

5.2 Diagramme 2 : Envoi et réception d'un message



** Le diagramme illustre une nouvelle session pour un tout premier échange entre deux agents.*
** Pas de restauration de l'historique de la conversation (puisque c'est un premier échange entre les deux agents).*

** Si ce n'était pas un 1er échange, la socket d'échange est restaurée depuis ListenConnThread où elle a été stockée au préalable.*
** Si ce n'était pas un 1er échange, une restauration de l'historique de la conversation est effectuée depuis la base de données au début de l'échange.*
** db1, id1 et pseudo1 concernent l'utilisateur u1, et db2, id2 et pseudo2 concernent l'agent u2.*
** ReceiverThread de u1 (qui reçoit les messages depuis u2) est créé au moment de la création de SenderThread de u1, et est ajouté à la liste statique "receivers" dans ReceiverThread.*

figure 4 : Diagramme de séquence illustrant le cycle d'une conversation

5.3 Diagramme 3 : Déconnexion

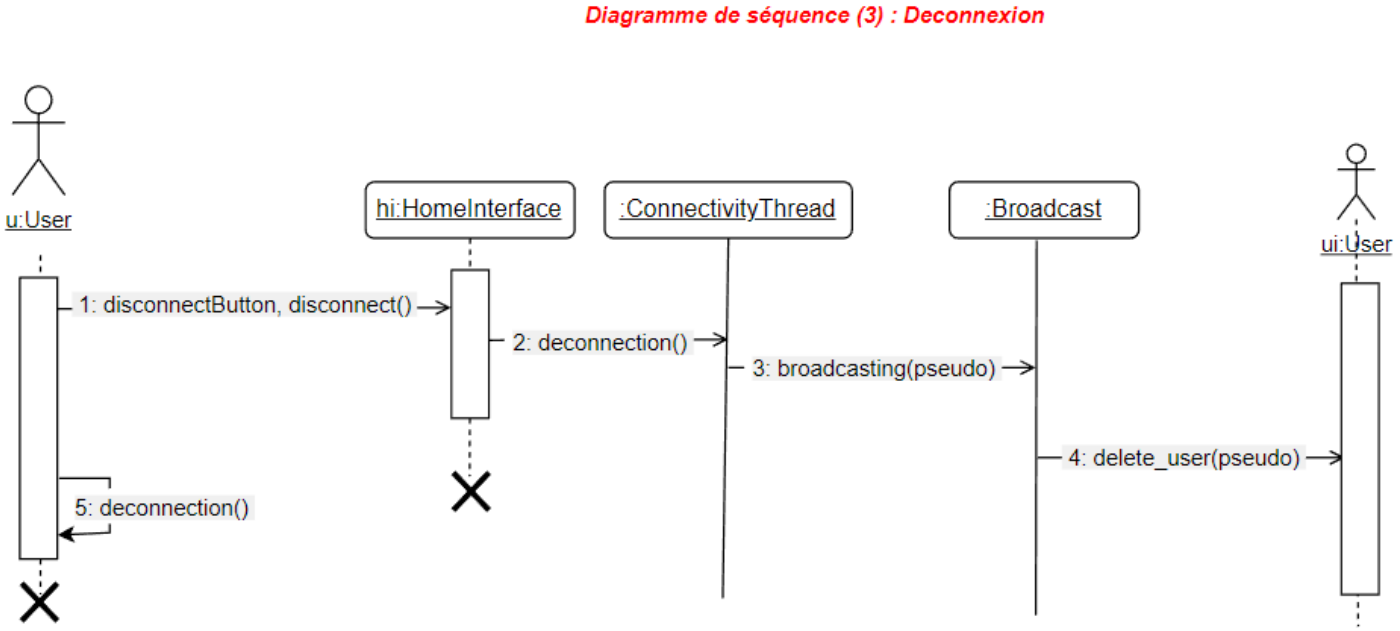


figure 5 : Diagramme de séquence pour une déconnexion d'un utilisateur

6 Diagramme de composite

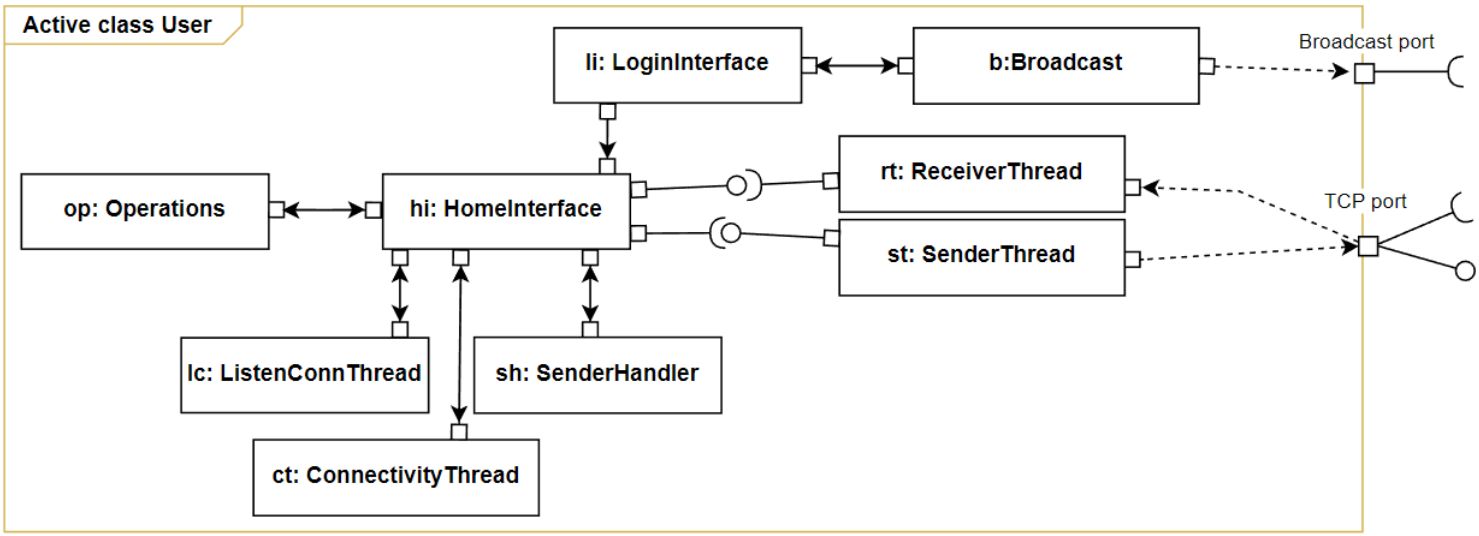


figure 6 : Diagramme de structure composite pour la classe active "User"

7 Diagramme de déploiement

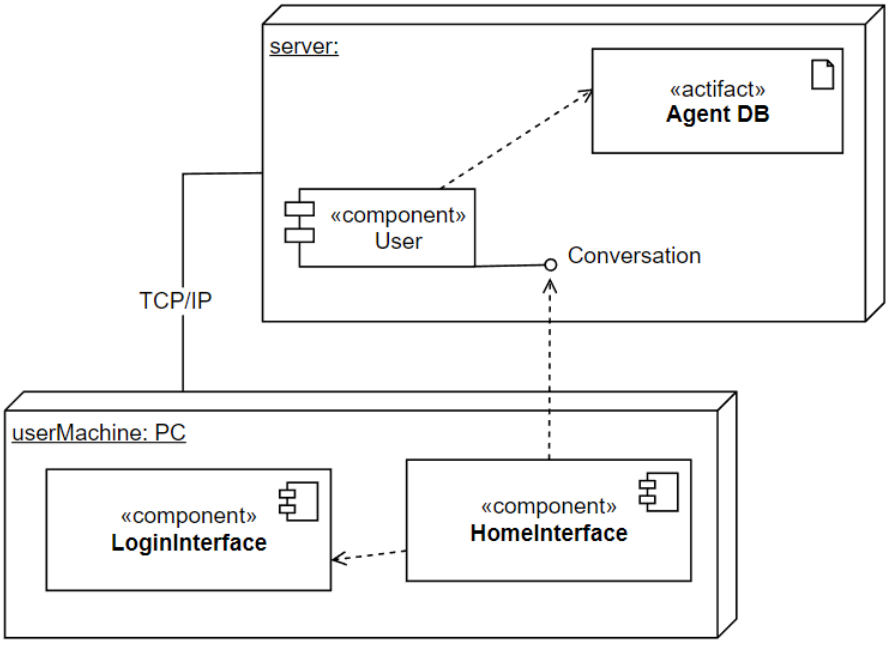


figure 7 : Diagramme de déploiement

8 Diagramme machine d'états

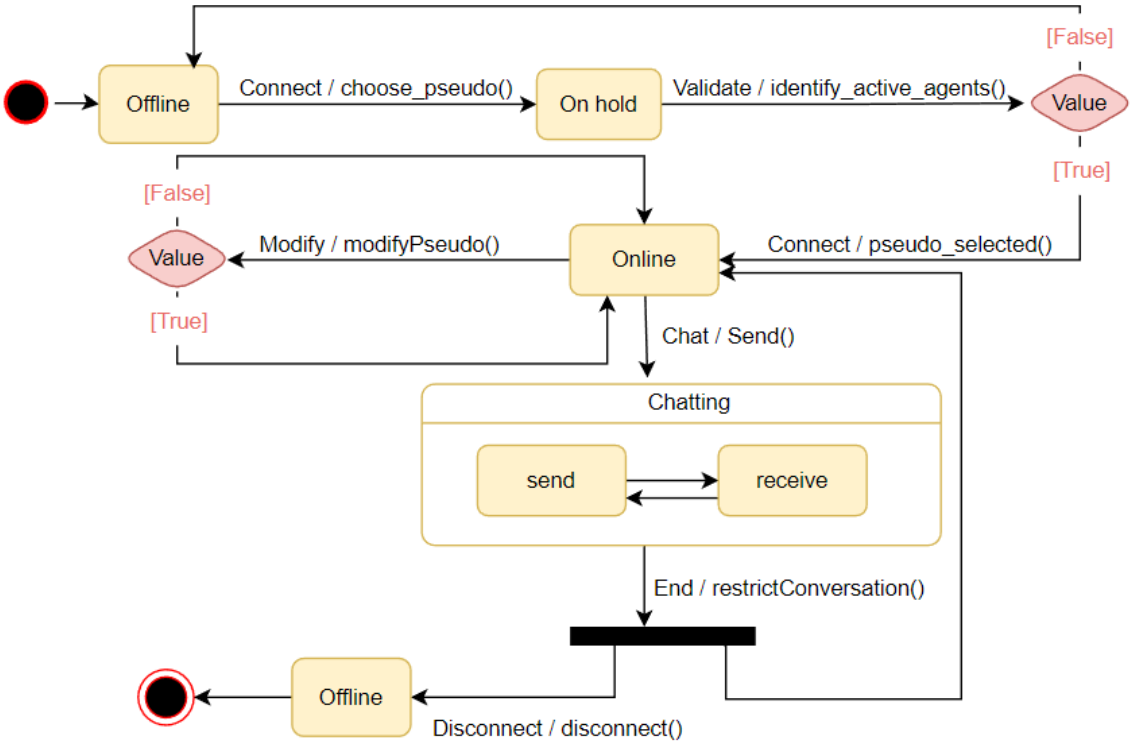
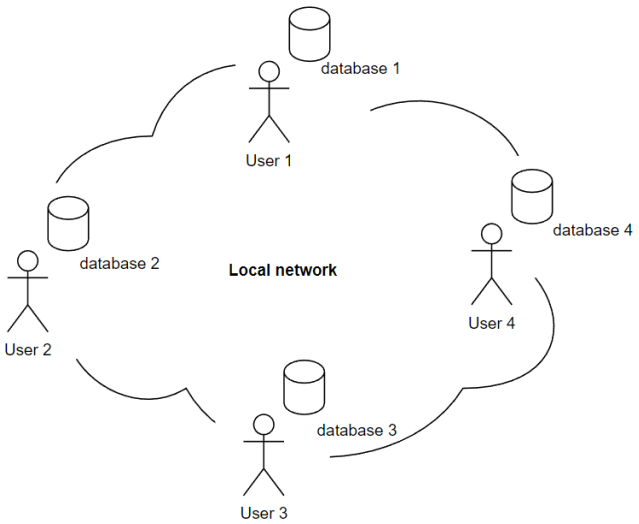


figure 8 : Diagramme machine d'état partant de la connexion "User" jusqu'à sa déconnexion

9 Schéma de la base de données



On a opté pour **une base de données distribuée**; Chaque utilisateur dispose d'un serveur qui héberge sa propre BDD, il aura au moment de sa première connexion une BDD vide propre à lui, et elle permet de sauvegarder tout l'historique de ses échanges avec les autres agents. Cet historique peut être restauré à la reconnexion avec le même id et le même pseudo utilisés à la 1ère connexion.
Pendant l'utilisation de l'application, un seul accès à la BDD est fait (au lancement de l'application), et la BDD se ferme une fois l'application est fermée.
Par convention, chaque utilisateur a un id unique (qu'il ne faut pas modifier ou changer) différent des id des autres agents. La modification du pseudo à la connexion (dans LoginInterface) est interdite; Si l'utilisateur veut modifier son pseudo, il peut le faire depuis son interface d'accueil pour que les autres agents soient informés de cette modification.
La BDD de chaque utilisateur est constituée de deux tables:

- Table "messages" dont les champs sont :
  - messageID : clé primaire de la table qui identifie d'une manière unique un message (généré automatiquement).
  - content : le contenu d'un message
  - date : la date et l'heure d'envoi ou de réception d'un message en format Timestamp.
  - operation : égal à 'R' si message reçu, ou 'S' si message envoyé.
  - id : l'id de l'agent récepteur/émetteur du message, (clé étrangère référençant pseudoID de la table pseudos).
- Table "pseudos" dont les champs sont :
  - pseudoID: clé primaire de la table qui identifie d'une manière unique un agent.
  - unPseudo : le pseudo de l'agent.

10 Architecture du système et choix technologiques

10.1 Architecture du système

On a choisi de développer l'application avec une architecture **MVC** (Modèle-Vue-Contrôleur).
Elle consiste à séparer trois entités distinctes qui sont, le modèle, la vue et le contrôleur ayant chacun un rôle précis dans l'interface. **Le modèle** contient la classe User, elle contient toutes les actions que peut faire un utilisateur. **La vue** regroupe les contrôleurs des deux interfaces qui composent l'application, ils affichent d'une part les données qu'ils récupèrent auprès du modèle et de la BDD et reçoivent d'autre part toutes les actions faites par l'utilisateur. Les différents événements sont envoyés au **contrôleur** qui est chargé de la synchronisation du modèle et de la vue. Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer (il regroupe les threads, les protocoles et les opérations d'accès et de modification de la BDD).
L'architecture MVC nous garantit une bonne ergonomie de développement. En effet, nous sommes

deux à devoir coder donc il est simple et naturel de diviser le travail en fonction de cette architecture.

10.2 Choix technologiques

- La partie **backend** est développée en Java (JDK 18) sous format de projet maven (version 3.8.7) à l'aide d'IntelliJ IDEA.
- La partie **frontend** est faite avec JavaFX dans Scene Builder.
- Vu qu'on a choisi de faire une **base de données** distribuée, on a opté pour SQLite comme un moteur de BDD relationnelle (JDBC SQLite version 3.20.1). On a utilisé le plugin "Database Tool" de IntelliJ pour visualiser les données.
- La partie **testing** est faite à l'aide de JUnit 5.
- Le **déploiement** du projet est fait à l'aide de GitHub Actions.
- On a suivi la méthode Agile Scrum pour la **gestion du projet**, et on a utilisé Jira. Globalement, on a fait 6 sprints d'une semaine chacun.

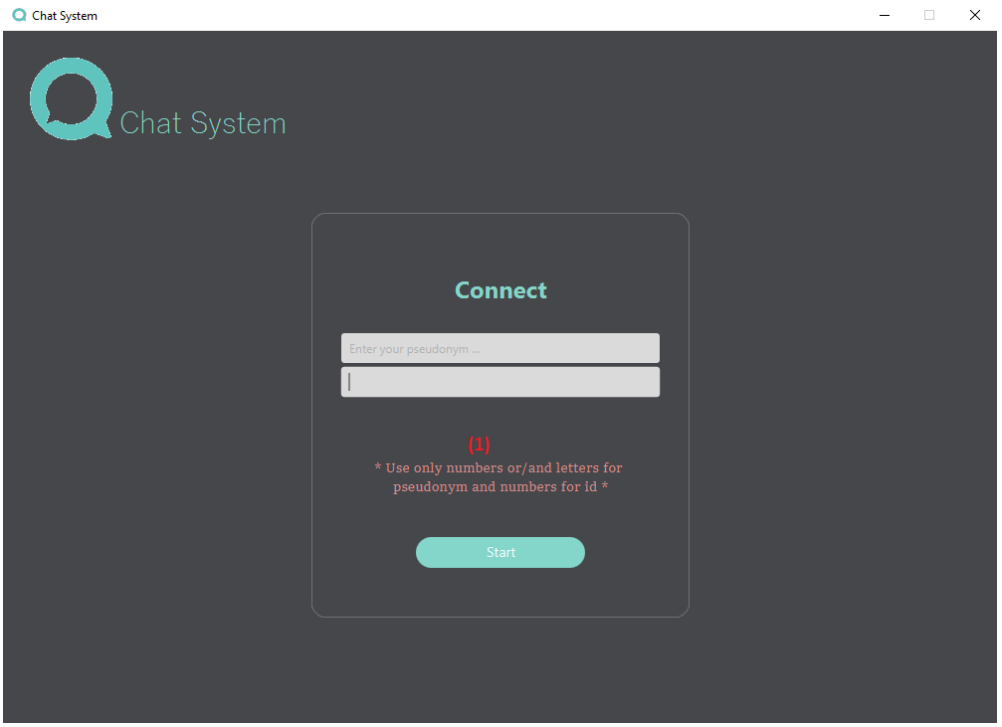
** Les fichiers .jar utilisés sont dans le dossier "/rendu/jar".*

11 Maquettes des GUI et manuel d'utilisation

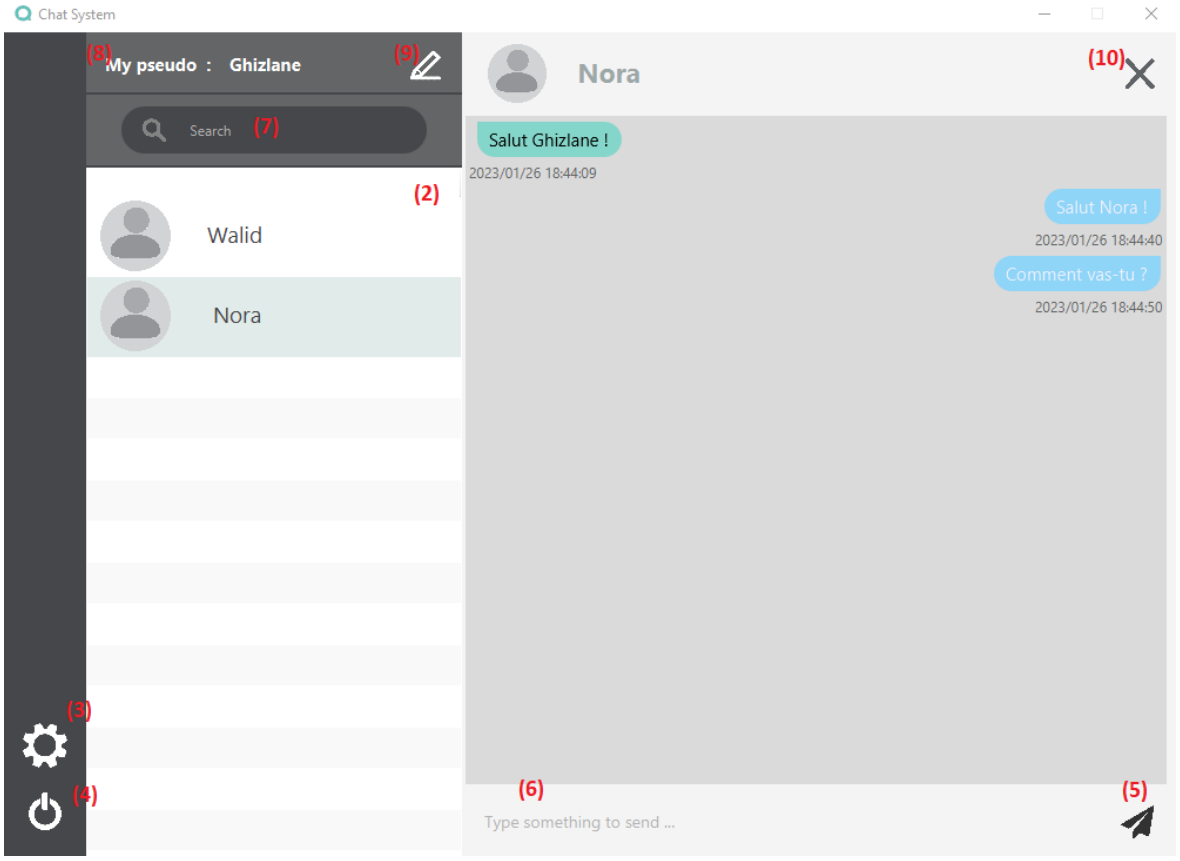
11.1 Interface Login

➔ Pour se connecter, on saisit le pseudo et l'id et on clique sur le bouton Start :
- L'utilisateur doit saisir un pseudonyme et son unique id.
- Les deux champs doivent être remplis.
- Le pseudonyme ne doit pas contenir ":" ou "@" pour des raisons de traitement des messages envoyés en UDP, et l'id doit être obligatoirement un nombre pour que les contraintes liées à la BDD soient respectées.
- Cette interface de connexion traite toutes ces conditions, et un message d'information s'affiche dans (1) si une de ces conditions n'est pas respectée.

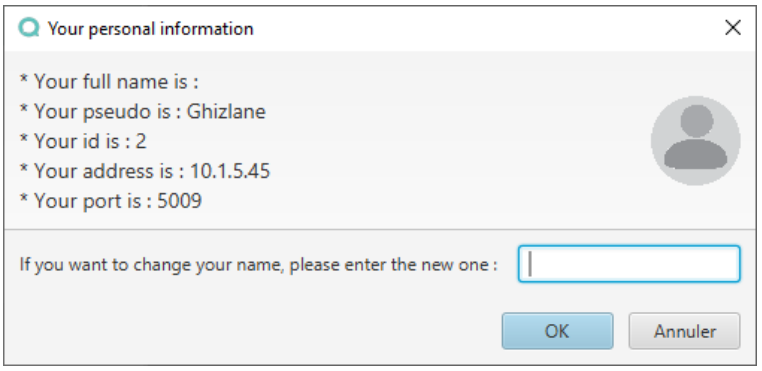
➔ Le bouton Start permet de lancer un broadcast vers toutes les machines du réseau :
- Si le pseudo n'est pas bon et qu'il est utilisé par un autre agent, un message d'information s'affiche dans (1).



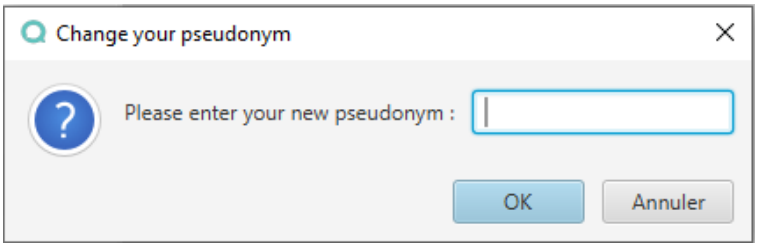
11.2 Interface Accueil



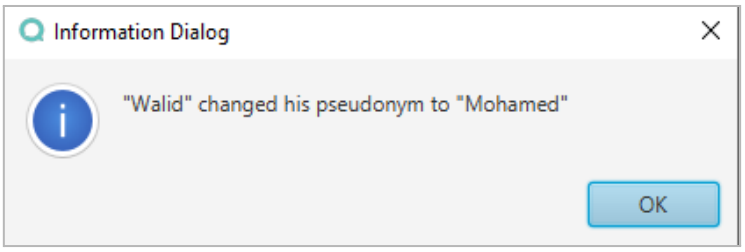
- ➔ Une fois connecté, votre liste des agents actifs, s'ils existent, s'affiche (2). Si la liste des agents actifs est longue, l'utilisateur peut chercher dans (7) sans avoir besoin de parcourir toute la liste.
- ➔ (10) sert à réduire un agent/une conversation.
- ➔ Les messages envoyés sont en bleu et les messages reçus sont en vert.
- ➔ La date et l'heure de l'envoi ou de réception sont marqués au-dessous des bulles de message.
- ➔ (4) permet de se déconnecter, et de retourner vers l'interface de connexion.
- ➔ (3) permet d'afficher une boîte de dialogue contenant les informations de l'utilisateur. Toutes ces informations à l'exception du nom ne sont pas modifiables.



- ➔ Pour envoyer un message, on choisit d'abord l'agent avec lequel on souhaite échanger, puis on insère le message dans (6) et puis on clique sur (5).
- ➔ Le pseudonyme de l'utilisateur connecté s'affiche dans (8) et il peut le modifier en cliquant sur (9), après la saisie du nouveau pseudonyme, une boîte de dialogue s'affiche pour informer si la modification est faite ou pas.

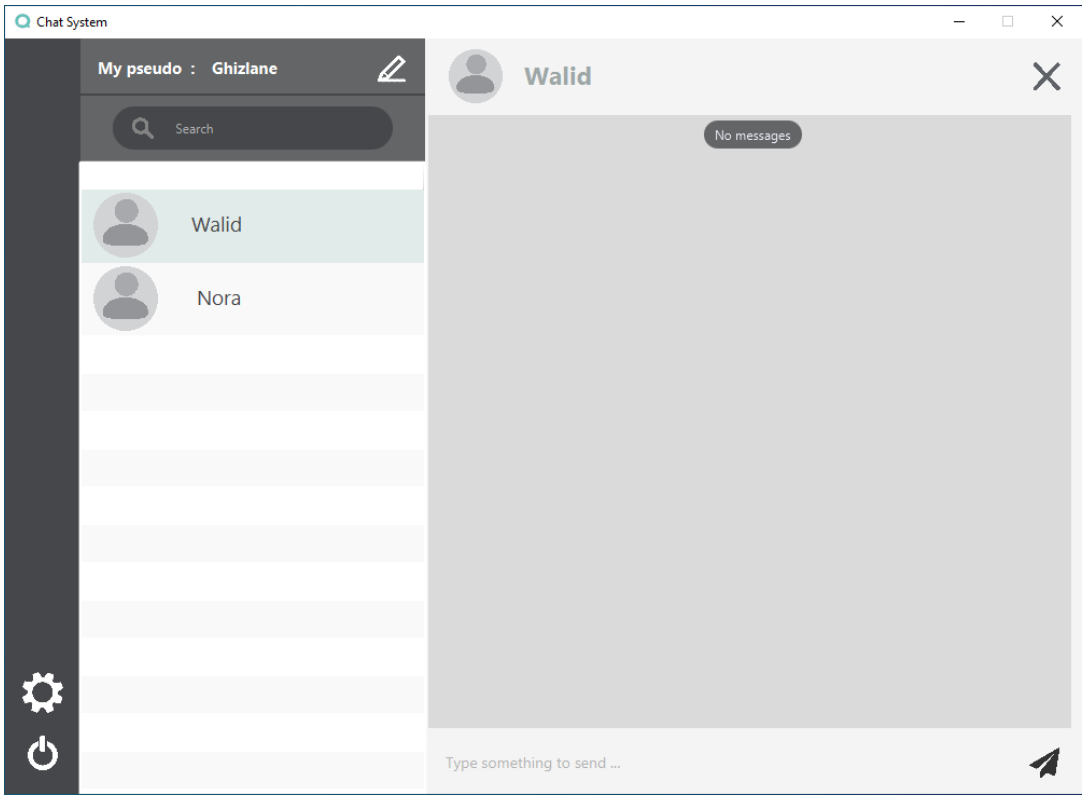


➔ Si un agent de la liste (2) a changé son pseudonyme, une fenêtre d'information temporaire s'affiche informant du nouveau pseudonyme de cet agent.



➔ Si un agent se déconnecte (se connecte), il disparaît (apparaît) automatiquement de (dans) (2).

➔ La conversation entre deux agents est restaurée si elle existe, sinon on a "No messages".



12 Procédure d'installation et de déploiement

Pour lancer l'application, il suffit d'avoir le dossier ChatSystem_jar situé dans "/out/artifacts/ChatSystem_jar" à condition d'avoir un JDK version 18 et puis exécuter dans un terminal la commande **java -jar ChatSystem.jar** ou tout simplement faire un double clic sur le fichier ChatSystem.jar. Le dossier ChatSystem_jar contient aussi la base de données à utiliser, et aucune configuration de cette dernière n'est demandée.

13 Procédures d'évaluation et de tests

Afin de vérifier son fonctionnement, on a lancé l'application dans 5 ordinateurs différents de l'INSA sous Windows, chacun peut communiquer avec les autres sans avoir une interférence entre les messages envoyés ou reçus.
Le code peut être exécuté depuis la classe _Main. Cette dernière a été ajoutée pour qu'on puisse générer un fichier .jar du projet, puisque on ne peut pas le faire avec la classe Main qui extends de Application.
Pour tester les différentes fonctionnalités de l'application, on a procéder pour des tests unitaires (dans src/test/java) pour toutes les classes sauf les classes HomeInterface (contient des objets graphiques), LoginInterface (contient des objets graphiques), Connexion (réalise des accès à la BDD), Operations (fait des modifications sur la BDD), ReceiverThread (contient que la méthode run()) et SenderThread (contient que la méthode run()).
Le fichier .github/workflows/maven.yml est configuré à lancer ces tests unitaires à chaque opération de push ou de pull.

14 Conclusion

Grâce à ce projet, on a pu réaliser toutes les étapes du développement d'un système de chat distribué et interactif en temps réel, en commençant par l'analyse des besoins, jusqu'à sa mise en service. Malgré les difficultés rencontrées, les objectifs définis au préalable ont été atteints.