

# Chat application

## Conception orientée objet

Rédigé par :

Chalhoub Sandro & Mousset Sarah  
Groupe 4IR - A2

- Rapport du 21 novembre 2022 -

# Sommaire

<b>Introduction</b>	<b>1</b>
<b>1 Diagramme des cas d'utilisation</b>	<b>2</b>
<b>2 Diagramme des classes</b>	<b>2</b>
<b>3 Diagrammes de séquence</b>	<b>4</b>
3.1 Connexion et déconnexion . . . . .	4
3.2 Changement de pseudonyme . . . . .	5
3.3 Envoi d'un message . . . . .	6
<b>4 Diagramme de structure composite</b>	<b>7</b>
<b>5 Diagramme relationnel de la base de données</b>	<b>8</b>
<b>6 Maquettes des GUI</b>	<b>9</b>
6.1 Login . . . . .	9
6.2 Application . . . . .	9
<b>Conclusion</b>	<b>10</b>

# Introduction

Le but du projet Chat Application est de concevoir un système de communication permettant à des individus d'échanger entre eux. Le système doit pouvoir s'adapter à différents scénarios et pouvoir être déployé sur divers environnements.

Pour débiter ce projet, en nous appuyant sur le cahier des charges, nous nous sommes penchés sur la conception du système. Nous avons utilisé plusieurs approches, et donc créé différents diagrammes, afin de mieux comprendre les interactions entre les objets que nous allons créer dans le cadre du projet, ainsi qu'identifier les limites et difficultés du projet.

Une fois les diagrammes établis, nous pourrions commencer à mettre en place le squelette de l'application et les principales classes du projet.

# 1 Diagramme des cas d'utilisation

Le premier diagramme que nous avons fait est un diagramme des cas d'utilisation, permettant d'avoir une meilleure idée de comment et pourquoi une personne pourrait utiliser notre application.

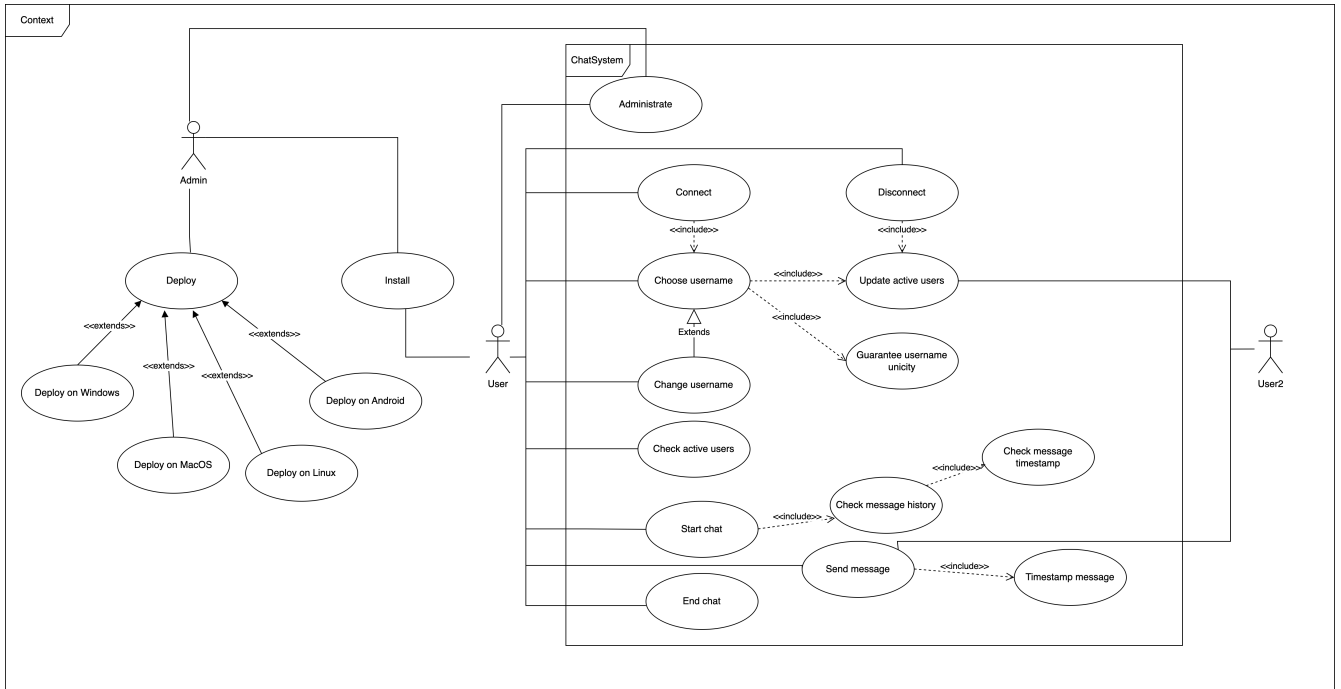


FIGURE 1 – Diagramme de cas d'utilisation

Nous avons identifié deux acteurs principaux : l'administrateur du système d'une part, et d'une autre part l'utilisateur principal en train d'utiliser l'application. Nous avons également identifié un acteur secondaire, représentant un autre utilisateur connecté. L'administrateur permet de gérer la compatibilité de l'agent avec le système d'exploitation dans lequel il est déployé, tandis que l'utilisateur principal est celui qui interagit avec l'agent. Dans cette vision précise de l'application, l'autre utilisateur est considéré inactif : il reçoit simplement un message ou met à jour ses connaissances sur les acteurs actifs. Il prendrait effectivement le rôle d'utilisateur principal s'il effectuait une action à son tour.

## 2 Diagramme des classes

Le diagramme de classes suivant permet de représenter les dépendances et différentes interactions existantes entre les classes du projet. Nous avons choisi de suivre une architecture de type MVC (*Model View Controller*), séparant les classes selon leur responsabilités, entre la logique et l'affichage de l'agent.

## 2.1 Schéma

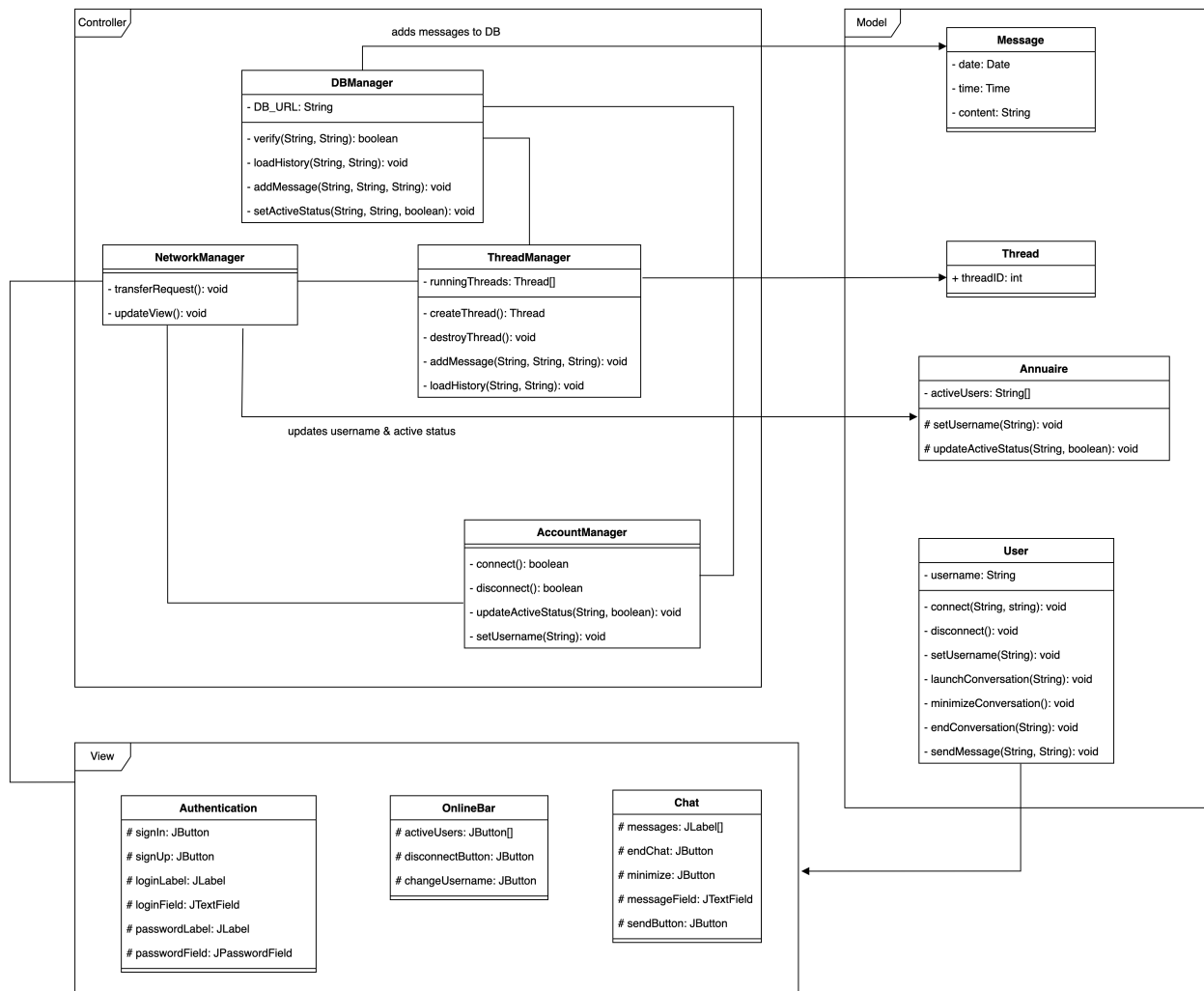


FIGURE 2 – Diagramme des classes

## 2.2 Détails

- Émetteur et récepteur ne figurent pas parmi les attributs de la classe *Message* parce que cet aspect du message est géré par la base de données (cf. schéma relationnel de la base de données).
- L’ID et le mot de passe de l’utilisateur ne figurent pas parmi les attributs de la classe *User* pour la même raison.
- Le fonctionnement global de l’application est le suivant : *User* communique par le biais des classes de type *View* qui, elles, transmettent ses requêtes au *NetworkManager*. Ce dernier permet tout simplement d’acheminer les requêtes vers la bonne classe de contrôle, puis de correctement mettre à jour les vues selon le retour du contrôleur appelé.
- Le *ThreadManager* permet de créer et détruire des *Threads*, sachant qu’il existe un *Thread* par conversation active.
- Le *DBManager* permet la gestion de la base de données des conversations (ajout de messages, chargement de l’historique), mais permet également de vérifier que l’identifiant et le mot de passe entrés par l’utilisateur lors de la phase de connexion sont corrects.

### 3 Diagrammes de séquence

Les diagrammes de séquence ci-dessous permettent de comprendre plus en détails les échanges ayant lieu entre les différentes classes lors d'une action effectuée par l'utilisateur. Nous avons choisi de faire abstraction des classes de type *View* pour nos diagrammes de séquence, afin de montrer au mieux le fonctionnement du système lui-même.

#### 3.1 Connexion et déconnexion

Lorsque l'utilisateur essaie de se connecter, le *NetworkManager* transfère la requête au contrôleur *AccountManager* qui vérifie les identifiants auprès de *DBManager* et le connecte ou le déconnecte. *AccountManager* met aussi à jour l'annuaire qui contient la liste des utilisateurs connectés.

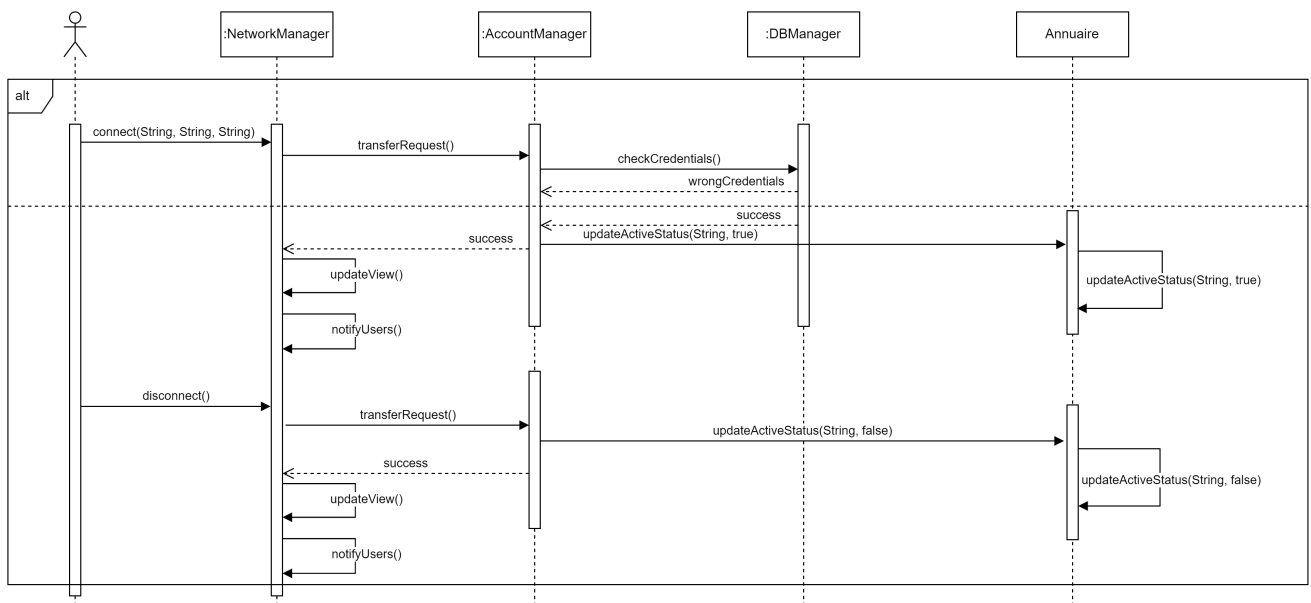


FIGURE 3 – Diagramme de séquence de connexion et déconnexion

## 3.2 Changement de pseudonyme

Lorsqu'un utilisateur souhaite modifier son pseudonyme, le *NetworkManager* transfère la requête à la classe *AccountManager* qui va demander au *DBManager* de modifier le pseudonyme si c'est possible. À la fin, dans le cas où le pseudonyme demandé n'était pas déjà pris, le *NetworkManager* met à jour la vue de l'utilisateur et notifie tous les autres utilisateurs connectés du changement de pseudonyme.

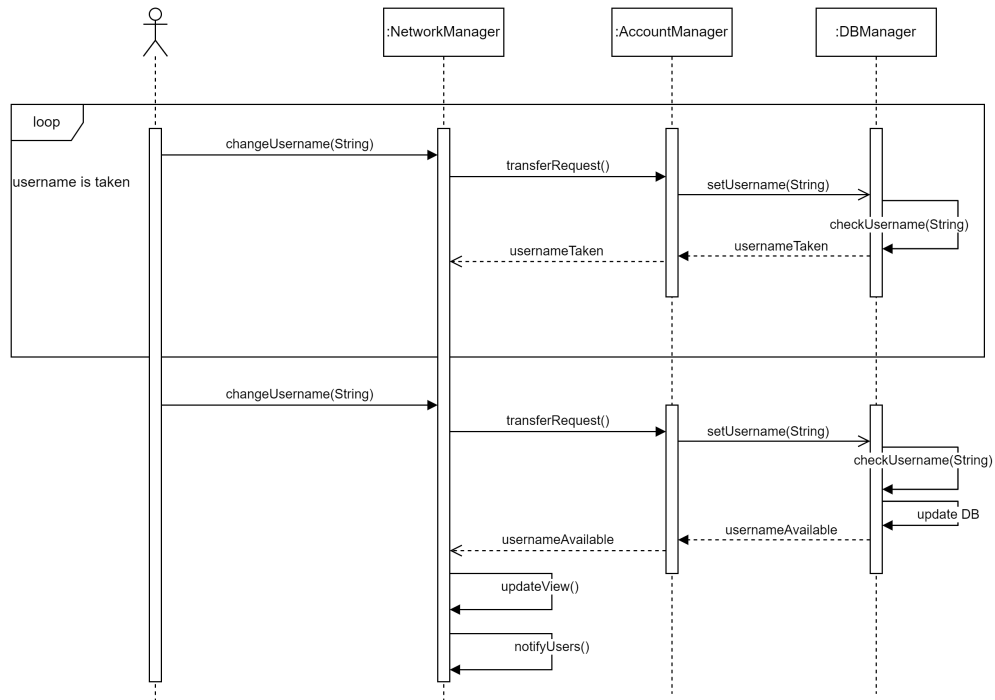


FIGURE 4 – Diagramme de séquence de modification de pseudonyme

### 3.3 Envoi d'un message

Lorsqu'un utilisateur veut envoyer un message à un autre utilisateur, le déroulé est le suivant. Tout d'abord, l'utilisateur destinataire est forcément en ligne, sinon l'utilisateur principal ne serait pas en mesure de le contacter car il n'apparaîtrait pas dans la liste d'utilisateurs actifs avec lesquels il pourrait potentiellement avoir une conversation. Le *NetworkManager* va tout d'abord transférer la requête au *ThreadManager* qui créera un thread pour la conversation et demandera à *DBManager* d'afficher l'historique de conversation ; puis, lorsque l'utilisateur enverra un message, demandera à *DBManager* d'ajouter ledit message dans la base de données.

Pour l'exemple ci-dessous, l'utilisateur décide d'abord de commencer une conversation avec un autre utilisateur connecté, puis il envoie un message au destinataire, et enfin ce-dernier décide de mettre fin à la conversation.

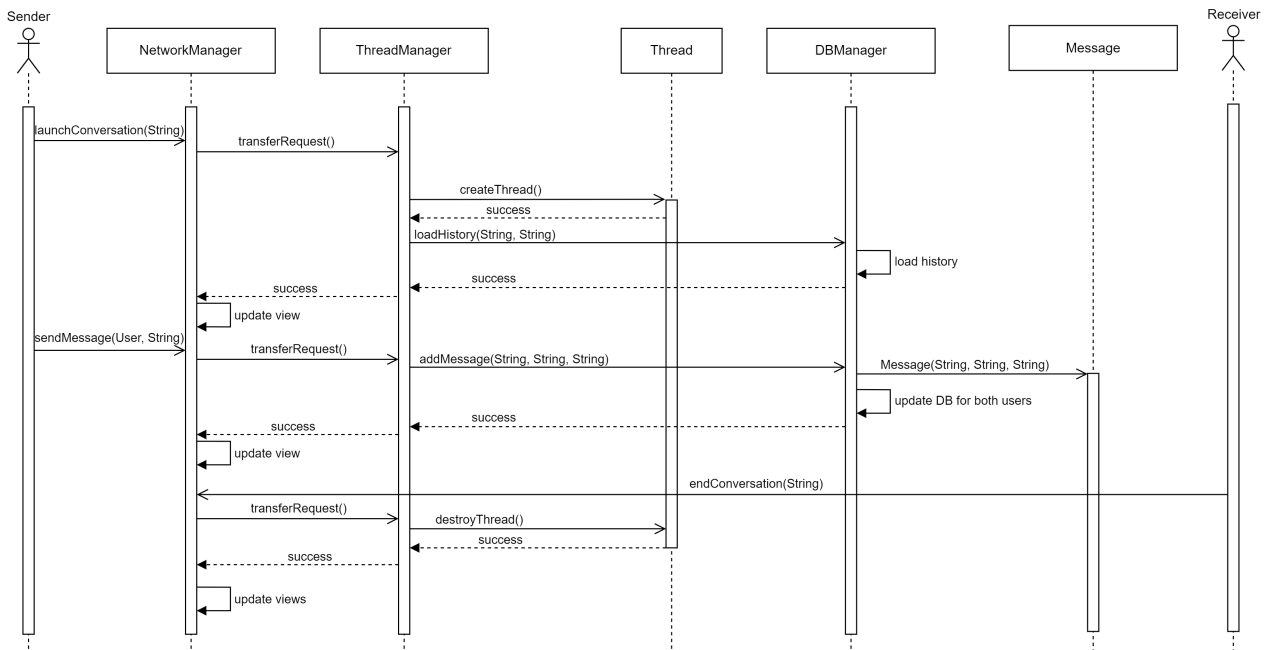


FIGURE 5 – Diagramme de séquence d'envoi d'un message



## 4 Diagramme de structure composite

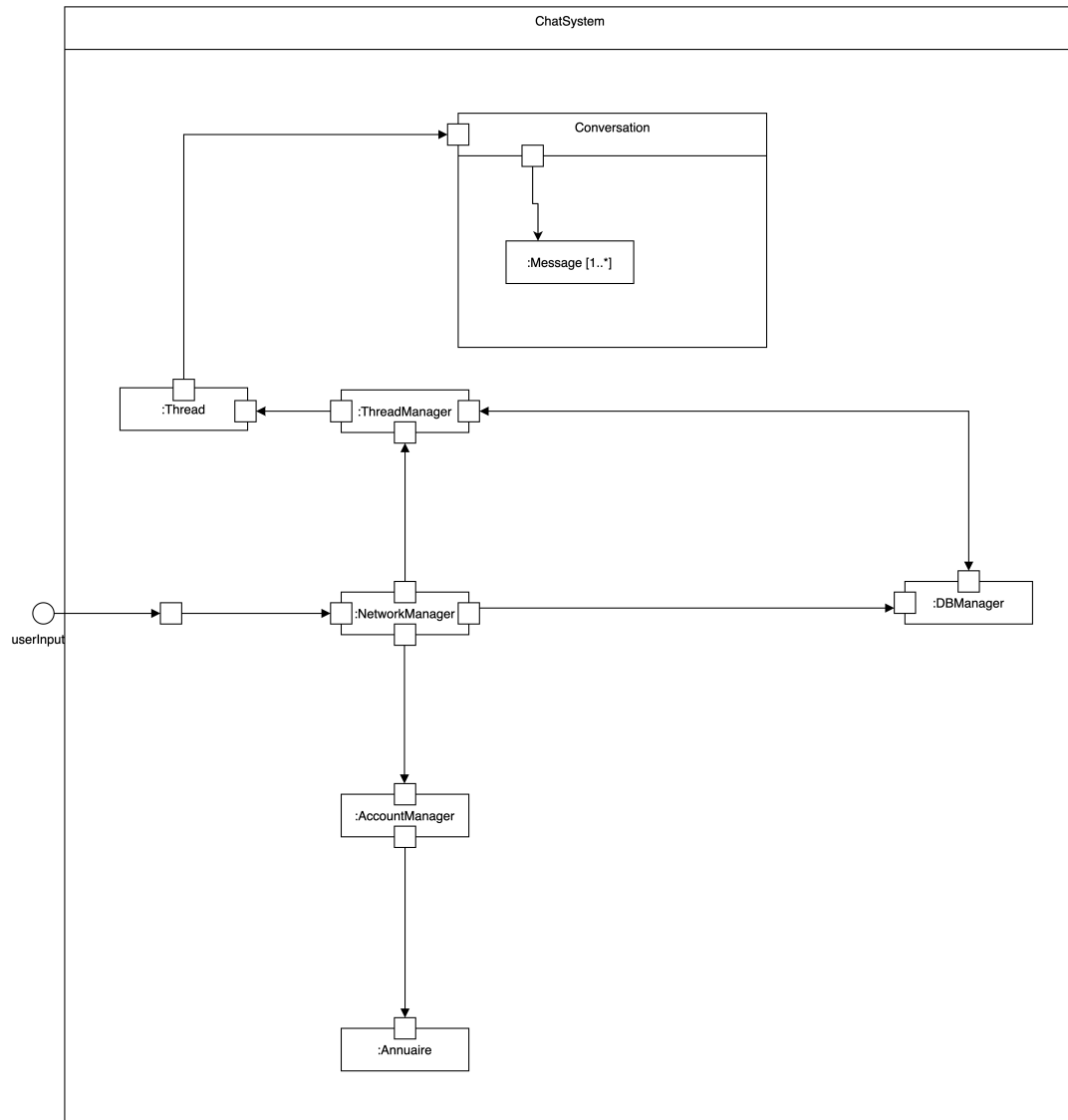


FIGURE 6 – Diagramme de structure composite

## 5 Diagramme relationnel de la base de données

Dans le cadre de ce projet, la base de données décentralisée a deux fonctions principales : stocker les informations liées aux comptes de l'utilisateur pour que ce dernier puisse se connecter (il s'agit donc d'un tableau à une seule entrée), et stocker les messages de toutes ses conversations, de façon à pouvoir restituer les historiques correctement. Évidemment, deux utilisateurs ont une conversation unique, mais un utilisateur peut discuter avec plusieurs autres utilisateurs. De même, une conversation peut contenir plusieurs messages, mais un message n'appartient qu'à une seule et unique conversation. De ce fait, notre diagramme contient deux entités, *Message* et *Conversation*. La relation "chat" se traduira, dans l'implémentation de la base de donnée, par la présence de la clé de *Conversation* en '*foreign key*' dans le tableau des messages, de sorte à pouvoir identifier l'appartenance de chacun des messages envoyés et reçus par l'utilisateur. L'attribut '*type*' de l'entité *Message* permet de déterminer si le message en question est un message reçu ou envoyé.

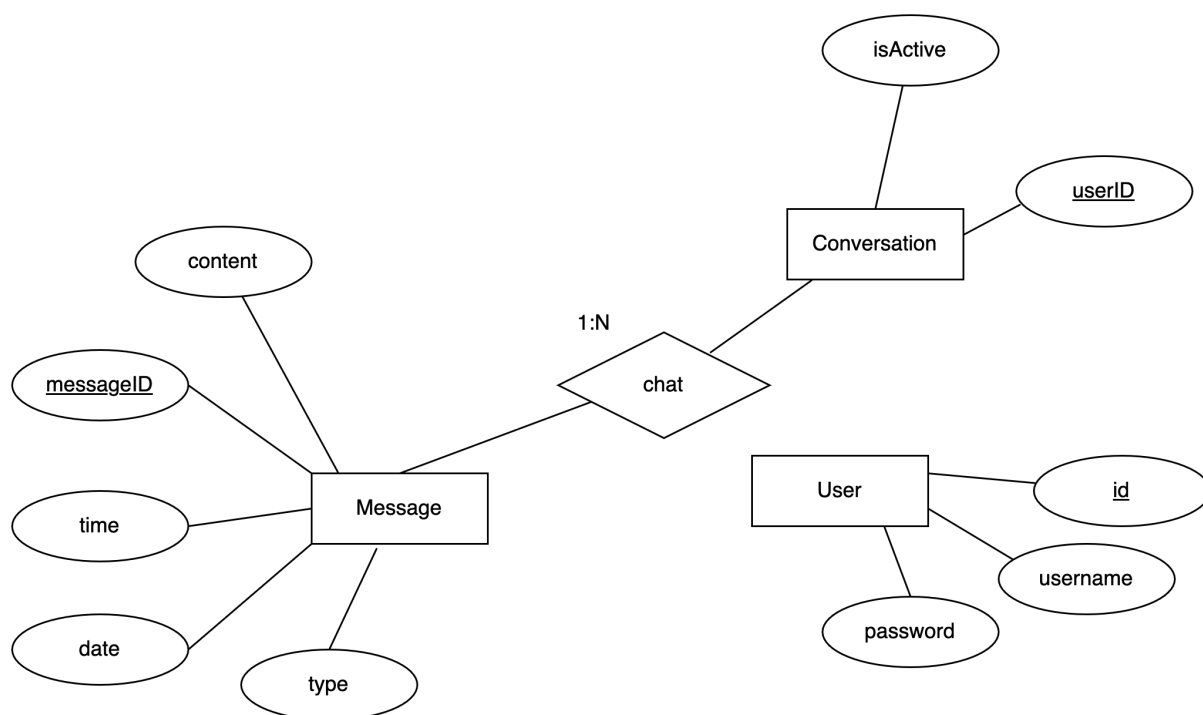


FIGURE 7 – Diagramme relationnel de la base de données

## 6 Maquettes des GUI

Les maquettes suivantes permettent d'avoir une meilleure idée de ce que nous avons en tête en ce qui concerne l'interface utilisateur. Nous avons choisi de ne représenter ici que les deux fenêtres principalement utilisées, soit celle de connexion et celle de clavardage. Cependant, il devra également exister une fenêtre pour le changement de pseudonyme, et une autre pour la création d'un compte, mais ces deux fenêtres seront assez similaires à la fenêtre de connexion.

### 6.1 Login



Don't have an account? [Create Account](#)

**Sign in**

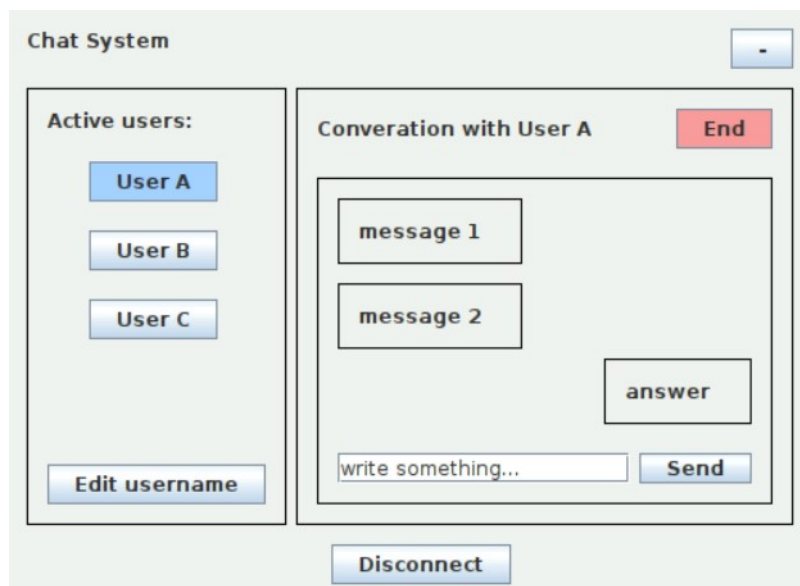
Login:

Password:

[OK](#)

FIGURE 8 – Maquette de la fenêtre de connexion

### 6.2 Application



**Chat System**

**Active users:**

[User A](#)

[User B](#)

[User C](#)

[Edit username](#)

**Conversation with User A** [End](#)

message 1

message 2

answer

write something... [Send](#)

[Disconnect](#)

FIGURE 9 – Maquette de la fenêtre de messagerie

## Conclusion

En conclusion, les diagrammes que nous avons réalisé nous ont permis de mieux comprendre comment nous pourrions implémenter notre application en prenant en compte ses différentes contraintes. Nous avons beaucoup appris grâce à cette conception, mais comprenons également qu'il y aura sans doute beaucoup de modifications à apporter une fois nous commençons l'implémentation du projet.

**INSA Toulouse**  
135, Avenue de Rangueil  
31077 Toulouse Cedex 4 - France  
[www.insa-toulouse.fr](http://www.insa-toulouse.fr)



MINISTÈRE  
DE L'ÉDUCATION NATIONALE,  
DE L'ENSEIGNEMENT SUPÉRIEUR  
ET DE LA RECHERCHE