INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport de Projet
UML

HAMDAN Célian Hilal, 4IR-SC
DA COSTA BENTO Marie-Line, 4IR-SC

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport de Projet

UML

HAMDAN Célian Hilal, 4IR-SC
DA COSTA BENTO Marie-Line, 4IR-SC

INSA TOULOUSE

Tables des Matières

I - DIAGRAMME DE CAS D'UTILISATION 3

II - DIAGRAMMES DE SÉQUENCE 4

A - Inscription et découverte des utilisateurs actifs 4

B - Changement de nickname 5

C - Client TCP : Connexion et envoi de message 6

D - Serveur TCP : Connexions entrantes et réception de messages 7

II - DIAGRAMME DE CLASSES 8

I - DIAGRAMME DE CAS D'UTILISATION

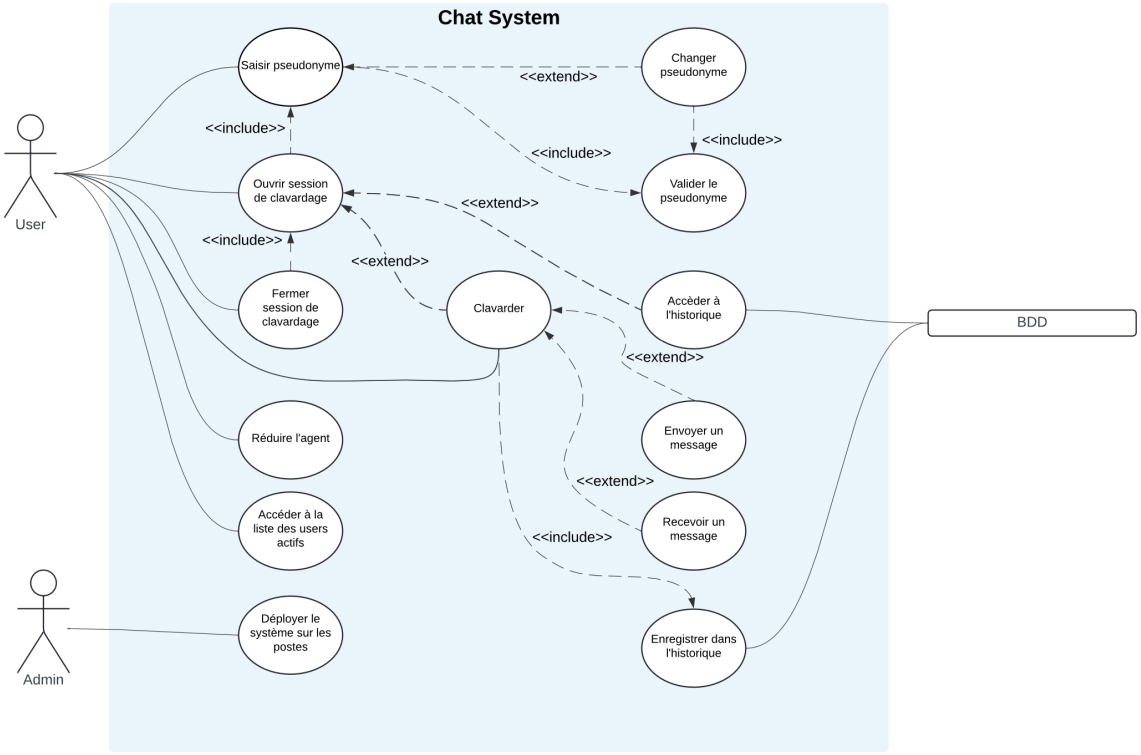

Figure 1 : Diagramme de cas d'utilisation : fonctionnalités principales de l'application

Ce diagramme de cas d'utilisation représente un système de clavardage, où deux acteurs principaux, l'utilisateur (« User ») et l'administrateur (« Admin »), interagissent avec différentes fonctionnalités du système. L'utilisateur peut effectuer des actions comme saisir un pseudonyme, ouvrir ou fermer une session de clavardage, envoyer et recevoir des messages, accéder à l'historique des conversations, et changer son pseudonyme. Ces actions incluent ou étendent des sous-fonctions spécifiques, comme la validation du pseudonyme ou l'enregistrement des messages dans l'historique, avec une interaction directe avec la base de données (« BDD »). L'administrateur, de son côté, est chargé de déployer le système sur les postes.

II - DIAGRAMMES DE SÉQUENCE

A - Inscription et découverte des utilisateurs actifs

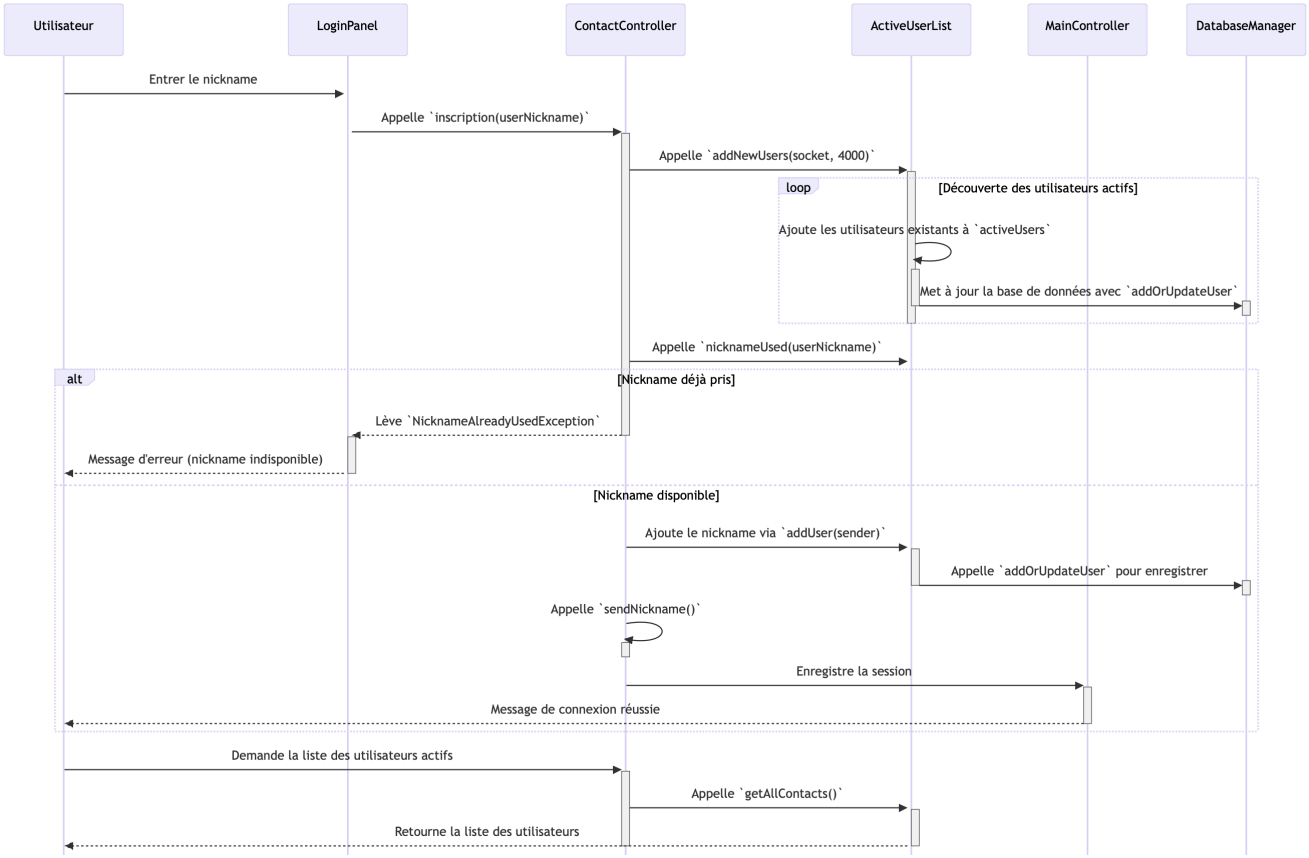

Figure 2 : Diagramme de séquence : processus d'inscription et découverte des utilisateurs actifs

Le diagramme de séquence de la *Figure 2* illustre le processus d'inscription et de découverte des utilisateurs actifs. L'**Utilisateur** commence par saisir son nickname dans le LoginPanel, qui envoie une requête d'inscription à **ContactController** via la méthode `inscription(userNickname)`. Ensuite, **ContactController** demande à **ActiveUserList** d'ajouter de nouveaux utilisateurs en appelant `addNewUsers(socket, port)`. Une boucle met à jour la liste des utilisateurs actifs et synchronise ces données avec la base de données via **DatabaseManager** grâce à la méthode `addOrUpdateUser`.

Si le nickname est déjà utilisé, une exception `NicknameAlreadyUsedException` est levée, renvoyant un message d'erreur à l'**Utilisateur.** Sinon, le nickname est ajouté, enregistré dans la base de données, et une session utilisateur est créée dans **MainController**, qui confirme la connexion réussie. Enfin, lorsque l'**Utilisateur** demande la liste des contacts actifs, **ContactController** interroge **ActiveUserList** via `getAllContacts()`, qui renvoie cette liste à l'**Utilisateur**.

B - Changement de nickname

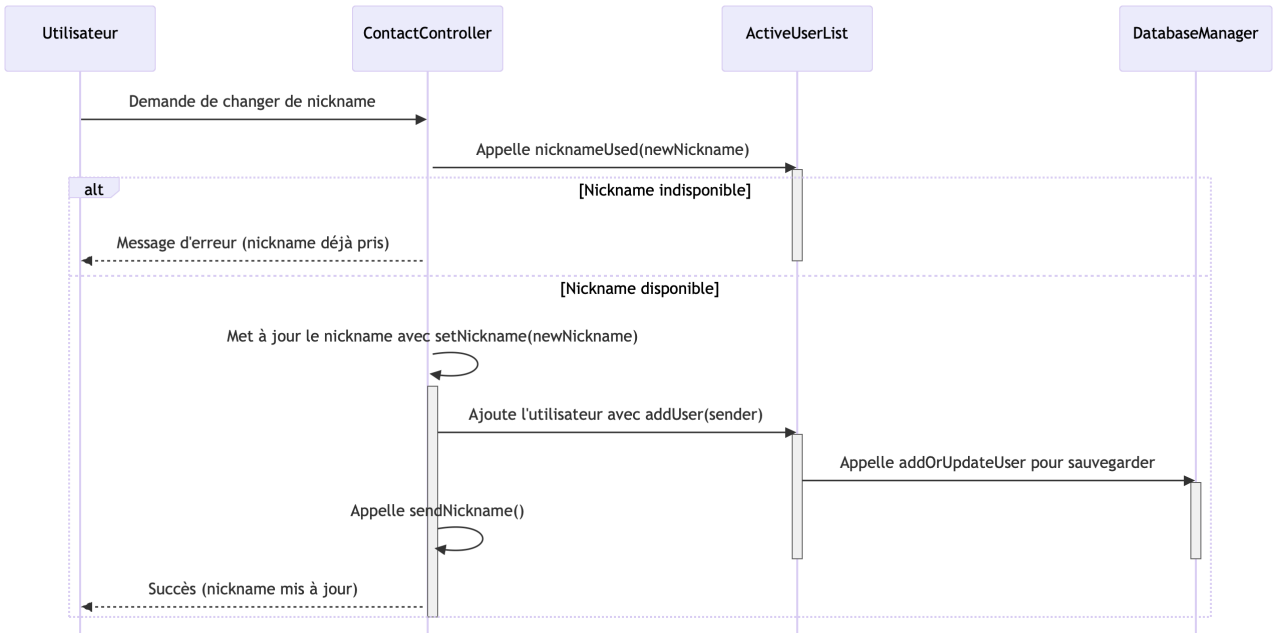

Figure 3 : Diagramme de séquence : processus de changement de nickname

Le diagramme de séquence de la *Figure 3* décrit le processus de changement de nickname. L'Utilisateur initie la demande en envoyant une requête au **ContactController** pour modifier son pseudo. **ContactController** vérifie la disponibilité du nouveau nickname via la méthode `nicknameUsed(newNickname)` de **ActiveUserList**. Si le pseudo a déjà été utilisé, un message d'erreur est renvoyé à l'Utilisateur. Dans le cas contraire, le nickname est mis à jour localement via `setNickname(newNickname)`. **ContactController** enregistre ensuite le nouvel utilisateur dans la liste active en appelant `addUser(sender)` de **ActiveUserList**, qui synchronise les données avec la base de données via la méthode `addOrUpdateUser` de **DatabaseManager**. Enfin, **ContactController** transmet le nouveau nickname via `sendNickname()` et informe l'**Utilisateur** du succès de l'opération.

C - Client TCP : Connexion et envoi de message

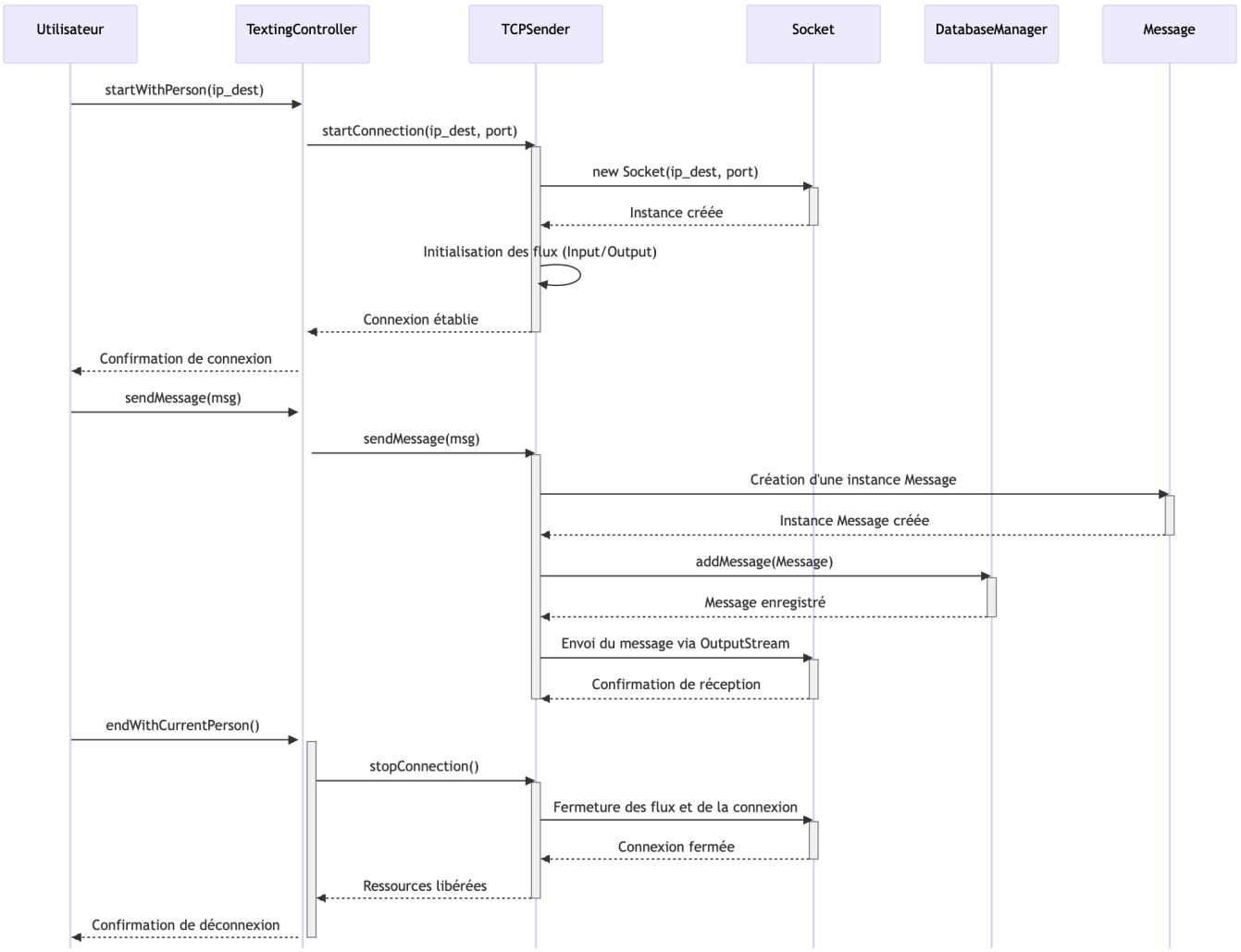

Figure 4 : Diagramme de séquence : connexion TCP et envoi de message

Le diagramme de séquence de la *Figure 4* illustre le processus de connexion, d'envoi de message et de déconnexion. L'**Utilisateur** commence par établir une connexion avec un destinataire en appelant `startWithPerson(ip_dest)` sur le **TextingController**, qui initie une connexion TCP via **TCPSender** et une instance de **Socket**, assurant l'initialisation des flux d'entrée et de sortie avant de confirmer la connexion à l'Utilisateur. Lors de l'envoi d'un message, l'**Utilisateur** appelle `sendMessage(msg)` sur le **TextingController**, qui passe la requête à **TCPSender** pour créer une instance de **Message**, l'enregistrer dans la base de données via **DatabaseManager**, puis transmettre le message au destinataire via le flux de sortie de **Socket**, avec une confirmation de réception. Enfin, lorsque l'**Utilisateur** souhaite mettre fin à la conversation avec `endWithCurrentPerson()`, le **TextingController** demande à **TCPSender** de fermer les flux et la connexion de **Socket**, libérant ainsi les ressources avant de confirmer la déconnexion à l'Utilisateur.

D - Serveur TCP : Connexions entrantes et réception de messages

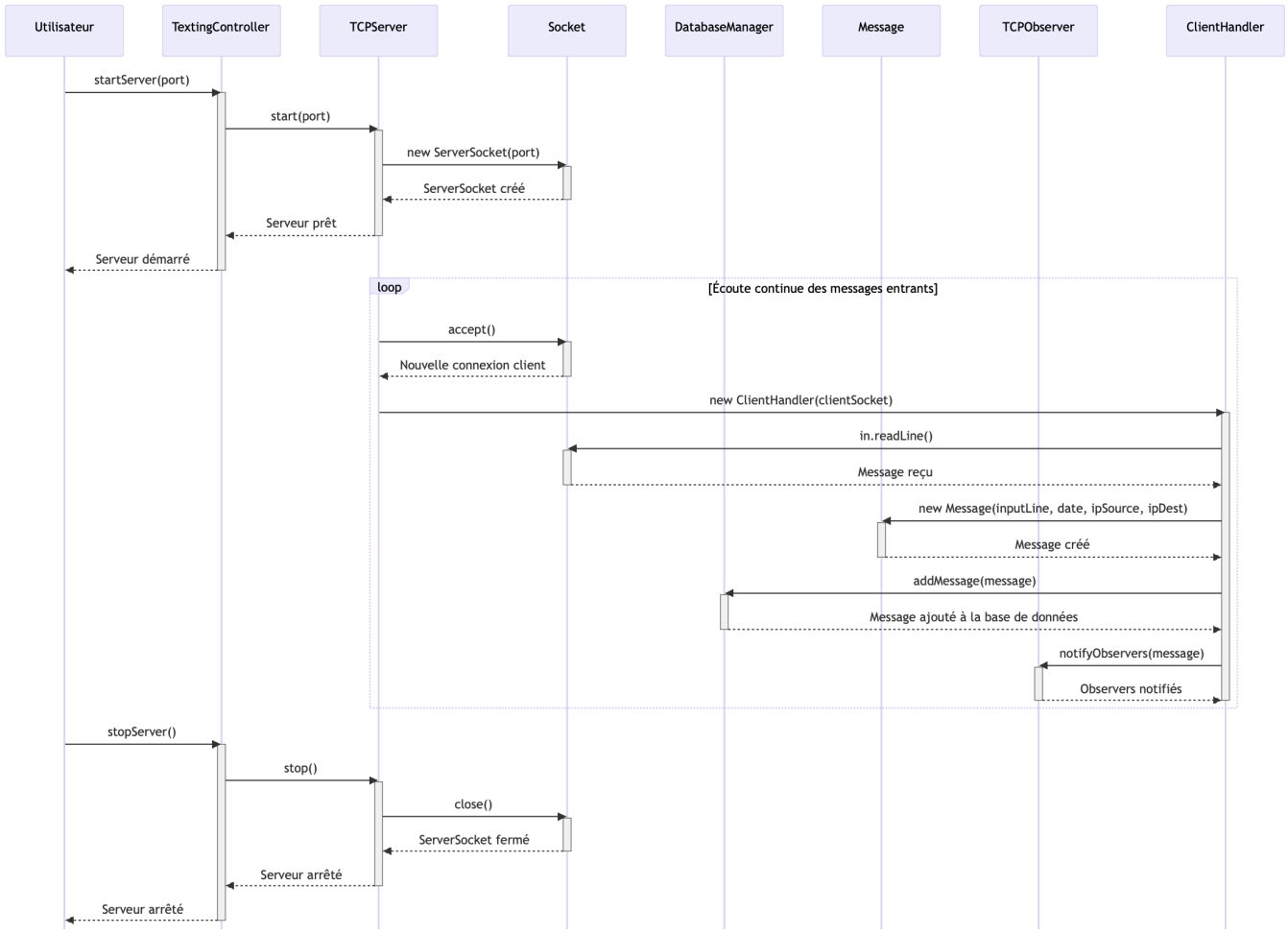

Figure 5 : Diagramme de séquence : fonctionnement d'un serveur TCP gérant connexions entrantes et réception de messages

Le diagramme de séquence de la *Figure 5* décrit le fonctionnement d'un serveur TCP gérant des connexions entrantes et la réception de messages. L'**Utilisateur** démarre le serveur en appelant `startServer(port)` sur le **TextingController**, qui initie un **TCPServer** en créant une instance **ServerSocket** pour écouter sur le port spécifié. Une fois le serveur prêt, une confirmation est renvoyée à l'Utilisateur. Ensuite, une boucle illustre l'écoute continue des connexions et des messages entrants. À chaque connexion, **TCPServer** accepte un client via `accept()` et crée un **ClientHandler** pour gérer cette session. Il va lire les données reçues via **Socket**, créer un objet **Message**, et le sauvegarde dans la base de données via **DatabaseManager**. Une fois le message enregistré, **TCPObserver** est notifié pour traiter les événements associés. Enfin, lorsque l'**Utilisateur** souhaite arrêter le serveur, il appelle `stopServer()`, ce qui entraîne la fermeture du **Socket** et la libération des ressources, avec une confirmation finale envoyée à l'**Utilisateur**.

II - DIAGRAMME DE CLASSES

Le diagramme de classes de la *Figure 6* modélise l'architecture de l'application en mettant en avant les interactions entre les classes principales. La classe **ActiveUserList** gère un ensemble d'objets **User**, permettant d'ajouter, supprimer et vérifier les utilisateurs actifs. **ContactController** interagit avec cette liste pour gérer les nouvelles connexions et inscriptions des utilisateurs via des messages UDP encapsulés dans des objets **UDPMessage**. La classe **DatabaseManager** assure la persistance des données d'utilisateurs et de messages, tandis que **MainController** orchestre les interactions entre les composants du système, notamment le serveur TCP (**TCPServer**) et le client TCP (**TCPSender**), qui permettent l'envoi et la réception de messages. La classe **TextingController** dépend du contrôleur principal pour gérer la communication directe entre utilisateurs. Enfin, des observateurs (**TCPObserver** et **UDPObserver**) sont utilisés pour notifier les composants concernés des changements d'état, tels que de nouvelles connexions ou la réception de messages, rendant le système extensible et réactif.

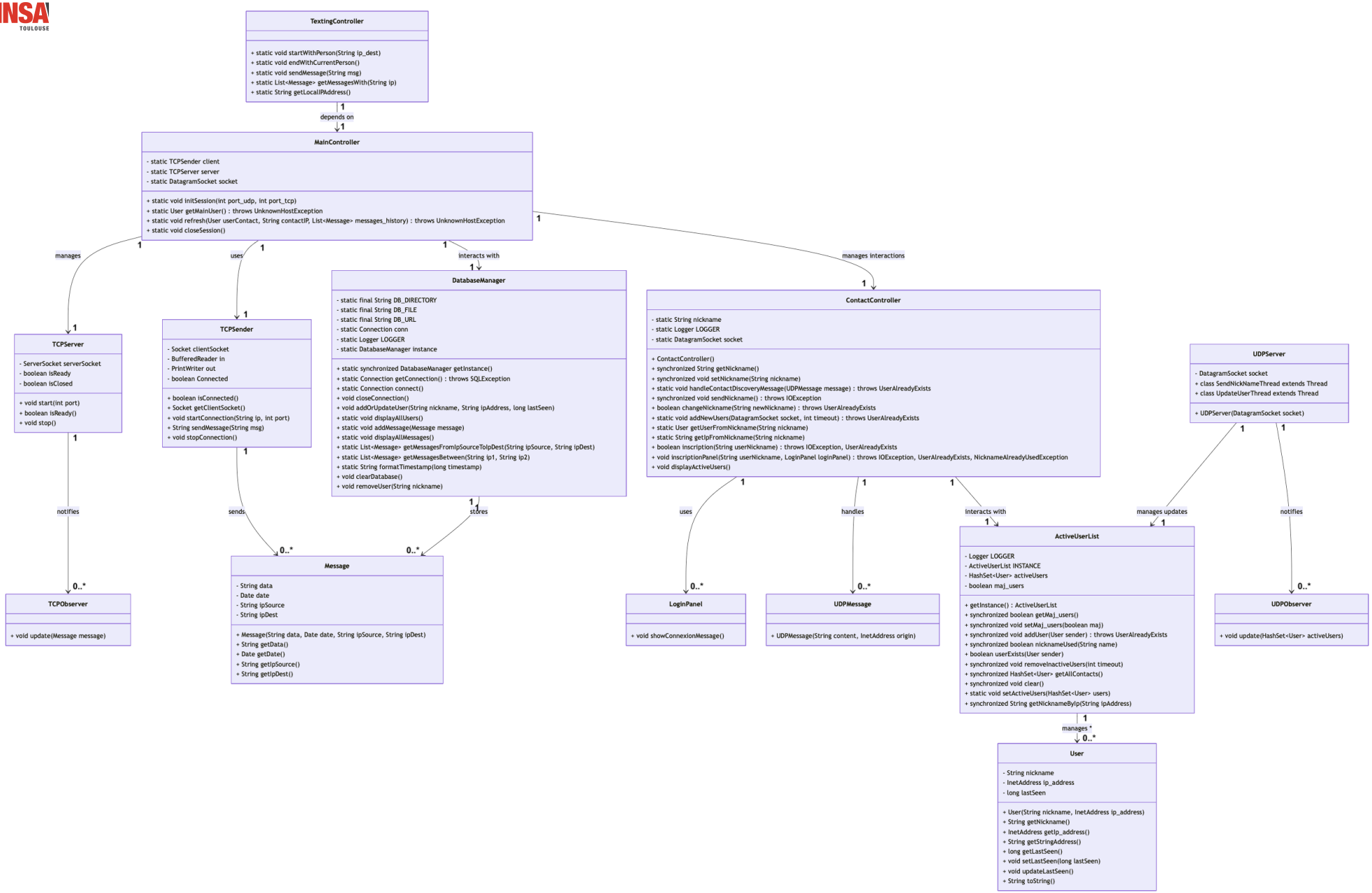

Figure 6 : Diagramme de classes