INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport de projet – UML/PDLA

Système de clavardage

MANSY Tizziana

KOUTSODIMA Georgia

4IR-A2 promotion 58

23 janvier 2024

Rapport de projet – UML/PDLA

Système de clavardage

MANSY Tizziana

KOUTSODIMA Georgia

4IR-A2 promotion 58

23 janvier 2024

Table des matières

Introduction

Dans le cadre du projet de la création d'un système de clavardage distribué interactif multi-utilisateur en temps réel (Chat system), nous devons appliquer nos connaissances sur le Processus de Développement Logiciel Automatisé (PDLA) ainsi que sur la Conception Orienté Objet. Dans un premier temps, nous analyserons notre travail sur la conception UML (centrée sur l'utilisateur et sur la structure interne du projet), puis nous allons voir comment on a appliqué PDLA sur ce projet, et enfin nous ferons un bilan sur le travail de groupe.

1 Conception UML centrée sur l'utilisateur

1.1. Acteurs et hypothèses

- Acteurs primaires

Dans notre application, les acteurs primaires sont les utilisateurs. En effet, notre application est adressée aux utilisateurs qui vont vouloir se connecter, envoyer et recevoir des messages à travers notre application. Ainsi, leurs interactions avec le système sont directes et essentielles pour la fonctionnalité de notre système.

Les techniciens sont également des acteurs primaires, car ils permettent la maintenance du service, en réglant d'éventuels dysfonctionnements et en surveillant les performances du système.

Les fournisseurs, quant à eux, sont chargés des tests et vérifications sur le système, ainsi que de fournir les plans de développement du système et d'Assurance Qualité.

- Acteurs secondaires

Nous avons quelques entités qui ont des interactions indirectes sans être les utilisateurs principaux participant directement aux conversations.

En effet, pour la deuxième phase du projet (TCP, échange de messages), nous avons utilisé un serveur de base de données afin de stocker les utilisateurs ainsi que les messages échangés. Ceci nous a permis entre autres d'assurer l'indépendance entre les conversations qui ont lieu entre des utilisateurs différents ainsi que l'affichage de l'historique des messages. Même si ce serveur n'est pas directement impliqué dans les conversations et l'interaction avec les utilisateurs, il reste essentiel pour le fonctionnement correct de notre application.

- Hypothèses

  - Concernant l'authentification, les utilisateurs n'ont pas besoin de s'authentifier en utilisant un login et un mot de passe car ceci ne fait pas partie des fonctionnalités

principales de notre application. En effet, les utilisateurs se connectent en choisissant un nickname qui doit être unique. Cela signifie qu'un utilisateur ne peut pas sélectionner un nickname déjà utilisé par un autre utilisateur déjà connecté. De plus, l'application est faite pour échanger seulement entre les machines du même réseau, ce qui signifie que l'identification de la personne se fait à partir de l'adresse IP de sa machine.

- Le système est capable de supporter la connexion simultanée de toutes les personnes au sein de l'organisme. Ainsi, le système est toujours fiable même étant en capacité maximale d'utilisateurs.

- On peut rajouter des nouvelles fonctionnalités à notre application sans avoir un impact négatif sur les performances du système. Par exemple, on peut rajouter l'apparition de notifications lors de la réception d'un message. En effet, nous cherchions lors de la création de l'application, à avoir un faible couplage et une haute cohésion.

1.2. Diagramme de cas d'utilisation (cf. Annexe 1)

Le diagramme des cas d'utilisation est construit à partir du cahier des charges. Les acteurs sont les utilisateurs de l'application. L'administrateur est responsable de la gestion globale de l'application (par exemple il installe les agents) et de l'administration des utilisateurs (par exemple il valide la création des comptes). Le fournisseur peut fournir des services supplémentaires à l'application. Le technicien est responsable de la surveillance de la continuité du service et de la résolution de problèmes techniques, garantissant son bon fonctionnement. La base de données permet le stockage des données de l'application.

Ce diagramme nous a permis de représenter les interactions entre les utilisateurs et notre système, décrivant les fonctionnalités disponibles.

1.3. Diagramme de séquence (cf. Annexe 2)

Notre diagramme de séquence est composé de 4 instances de classes : l'utilisateur, le chatsystem, l'administrateur et le technicien. L'utilisateur commence par s'identifier. Ensuite, on a créé une condition if else : si le système fonctionne correctement alors l'utilisateur peut utiliser l'application (envoyer des messages, changer son nickname, etc.), sinon il doit notifier le dysfonctionnement afin que le technicien puisse intervenir. Une session est représentée par une boucle infinie : tant que la session est ouverte, l'utilisateur voit l'historique des messages échangés avec la personne et peut envoyer et recevoir des messages.

Le diagramme de séquence illustre les messages échangés entre les différents acteurs. On peut suivre le déroulement des fonctionnalités de l'application de façon visuelle.

2 Conception UML centrée sur la structure interne du projet

2.1 Diagramme de classes

Nous avons utilisé le modèle MVC afin de faciliter la séparation des responsabilités, la réutilisabilité du code ainsi que l'ajout de nouvelles fonctionnalités. Les classes du Modèle sont en turquoise, du Contrôleur en jaune et de la Vue en rouge. De plus, nous avons souligné les classes statiques.

Les diagrammes de classes que nous avons créés servent à décrire les relations entre les différentes classes afin d'avoir une vue globale des entités du système. Ceci nous a permis de mieux comprendre et organiser notre code.

- Découverte des contacts (cf. Annexe 3)

Nous avons utilisé une relation de composition pour relier Receive et Broadcast car la classe Receive est implémentée dans la classe Broadcast. Elle est responsable de la réception des broadcasts et des messages unicasts. Le 0..1000 correspond au nombre de sessions de clavardage qui peuvent être ouvertes simultanément par un utilisateur. Une session est ouverte par un utilisateur (ce qui est représenté par le chiffre 1 sur la relation d'association). Par ailleurs, pour la valeur du port dans le broadcast, on a choisi de faire un {readOnly} car elle ne peut pas être modifiée après son initialisation (les clients peuvent lire la valeur, mais ne peuvent pas la modifier directement). On a choisi une valeur appartenant à l'intervalle [1024 ; 49151] afin d'éviter les ports qui sont réservés.

- ChatSystem entier (cf. Annexe 4)

Toutes les classes du View sont connectées par des agrégations, indiquant ainsi la navigation ou le passage d'une interface à une autre lorsque l'utilisateur appuie sur le bouton correspondant.

AppData est une classe passive car elle contient des méthodes qui sont utilisées par d'autres classes.

Nous avons rajouté des couleurs sur les relations de dépendance afin de rendre le diagramme plus facile à lire.

Concernant le modèle "Observers", dans notre diagramme, l'interface Observer est implémentée dans ClientsList.Observer, le "Concrete Observer" correspond à la classe StartEverything, héritage de "Observer", et la subject class est "ClientHandler".

2.2 Schéma de la base de données (cf. Annexe 5)

Le database schema est une représentation de la base de données lors de l'utilisation de notre application. Il définit la structure logique de notre base de données.

Nous avons donc une table "Users" où sont stockées les adresses IP des contacts de l'utilisateur. Elles sont uniques et servent donc d'identifiant, ce qui fait qu'elles correspondent à la clé primaire.

Nous avons également créé une table pour stocker les messages par contact, contenant la date et l'heure du message, l'adresse IP de la personne qui l'a envoyé et le contenu du message. Ainsi, le nom de cette table sera "Messages_100_12_25_15" pour la machine ayant l'adresse

100.12.25.15. Nous l'avons représenté en une seule table car le nombre dépend du nombre de contacts.

Les tables "Messages" correspondent à une table "Users" et la table "Users" correspond à plusieurs tables "Messages".

2.3 Architerture overview (cf. Annexe 6)

Nous avons choisi de représenter notre architecture par un diagramme de séquence afin de représenter le système de communication et la base de données. Nous avons représenté l'ensemble des échanges d'information lors de l'utilisation des protocoles UDP et TCP, ainsi que le stockage des informations dans la base de données au fur et à mesure de ces échanges.

Nous avons choisi de ne pas représenter les échanges avec l'interface utilisateur (View) afin de simplifier le diagramme et mettre en évidence la logique de communication entre les différentes parties du système (communication, base de données).

Ainsi, ce diagramme nous a permis d'avoir une vue claire sur l'ensemble de l'architecture et sur le fonctionnement général de notre système.

Par ailleurs, pour les interfaces (View), nous avons utilisé Swing car il s'agit d'une technologie Java et donc les classes et les objets Swing peuvent être utilisés en conjonction avec d'autres classes et objets Java. Il comporte toutes les fonctionnalités (bouton cliquable, saisie de texte, etc.) dont nous avons besoin pour notre application.

Nous avons choisi d'utiliser SQLite pour la base de données (représentée par "database" dans le schéma) en raison de sa simplicité et de son efficacité pour des applications légères et locales. En effet, SQLite stocke seulement les bases de données dans un fichier, sans utiliser de serveur, simplifiant ainsi l'utilisation de la base de données.

3 Processus de Développement Logiciel Agile (PDLA)

3.1 Méthode Agile

Pour la gestion de notre projet, nous avons utilisé Jira.

Nous avons commencé par analyser les différentes fonctionnalités de notre application (données dans le cahier des charges) afin d'avoir une idée globale du travail à faire. Ainsi nous avons pu créer les user stories correspondant aux différentes fonctionnalités de notre application. Pour créer les tickets enfants, nous avons dû rentrer plus dans les détails d'implémentation de l'application.

En effet, ces derniers étant plus spécifiques, ils demandent d'avoir une idée plus précise de comment mettre en place les fonctionnalités. Nous avons d'abord débriefé sur comment fonctionnerait l'implémentation du code concrètement avant de les créer. Nous avons également ajouté des tickets enfant au fur et à mesure que nous travaillions car nous avons réalisé que nous n'avions pas pensé à tout ce qu'il faudrait mettre en place.

Ensuite, nous avons classé nos user stories par ordre de priorité afin de pouvoir les assigner à des sprints, notre but étant de traiter en premier les fonctionnalités les plus critiques pour les utilisateurs. On travaillait toutes les deux sur le même sprint afin de pouvoir communiquer plus

facilement et afin de pouvoir s'entraider si besoin. Nous avons donc effectué une partie assez importante du travail ensemble (particulièrement les parties TCP et UDP).

Chaque sprint représentait une phase différente du projet.

En effet, tout le premier sprint était dédié à la découverte de contacts et donc à la connexion (et déconnexion) d'un utilisateur à l'application. Ensuite, lors du deuxième sprint, nous nous sommes concentrées sur l'échange de messages (avec TCP), l'ajout des données utilisateurs et l'affichage de l'historique (avec la base de données). Il s'agissait de la partie du travail la plus importante. Le troisième sprint est beaucoup plus court car il s'agit d'une fonctionnalité moins prioritaire : le changement de nickname.

Nous avons mis des délais assez longs pour ces sprints car le travail à faire était assez conséquent (un à deux mois pour les premiers sprints et cinq jours pour le dernier). Nous les avons terminés à temps, même si nous ne les avons pas terminés directement sur Jira car il aurait fallu les rouvrir pour y avoir accès de nouveau. Nous avons donc terminé les sprints plus tard, quand nous savions que nous n'aurions plus besoin de les voir.

L'utilisation de la méthode Agile nous a permis de mieux communiquer et collaborer ainsi que de répondre de manière flexible aux changements tout le long du projet. Particulièrement pour la mise en place de priorités sur les fonctionnalités à mettre en place et dans notre façon de travailler, Jira nous a aidé à structurer notre travail.

3.2 Contrôle de version

Nous avons utilisé GitHub.

Nous avons travaillé sur la même branche (main) afin de voir en temps réel les modifications apportées par l'autre membre du groupe. En effet, comme notre groupe était constitué seulement de 2 personnes, la communication était facile et nous n'avons pas jugé nécessaire de créer plusieurs branches. De plus, nous étant réparti le travail au préalable, nous savions que, si nous travaillions sur deux fonctionnalités séparées, cela ne poserait pas de problème car il n'y aurait de collision. Nous faisions plusieurs commits, notamment quand on travaillait séparément, afin que l'autre puisse savoir précisément l'avancement du projet. Ainsi, nos commits fournissent un historique clair des changements apportés au code.

Nous avons également utilisé GitHub pour l'automatisation des tests unitaires, pour pouvoir détecter les erreurs de notre code rapidement. De plus, le fait de rendre les tests automatiques nous permettait de voir si une fonctionnalité de notre application créée précédemment était altérée lors de l'avancement du projet, qui a demandé à plusieurs reprises de changer des éléments déjà créés pour mettre en place les éléments suivants. Nous pouvions voir directement les résultats attendus, et s'ils étaient atteints ou non.

3.3 Gestion des erreurs

Pour nos tests, nous avons créé des tests unitaires en utilisant des assertions et la méthode suite(), qui fournit une suite de tests à exécuter et renvoie une instance de TestSuite. Nos classes héritent de TestCase et utilisent ses méthodes.

Pour la découverte de contacts, nous avons créé une classe dans laquelle on a testé toutes les fonctionnalités concernant la gestion de la réception et de la liste de contact (par exemple, la réception d'un nickname déjà pris, l'ajout d'un utilisateur dans la liste de contact, etc.).

Pour la base de données, nous avons commencé par tester la connexion ainsi que la création de tables. Ensuite, nous avons créé tous les tests concernant la gestion des messages (leur présence dans la base de données, la réception, etc.). Nous nous sommes aussi concentrés sur toutes les fonctionnalités de la base de données concernant les utilisateurs. Ainsi, nous avons par exemple l'ajout d'un utilisateur dans la base de données, le changement de son nickname, etc.

Nous n'avons pas testé tout ce qui correspond à la partie réseau (broadcast, client/serveur etc…), car il n'est pas possible de tester ces fonctionnalités seulement grâce à des tests unitaires.

Par ailleurs, nous avons utilisé des loggers pour le suivi du débogage de notre code. Par exemple, dans la classe Broadcast, nous avons utilisé un logger pour l'enregistrement d'un message d'information (logger.info("Receive socket closed.");) et d'un message d'erreur (logger.log(Level.SEVERE, "IOException: " + e.getMessage());). Nous avons utilisé le niveau sévère pour les problèmes critiques. Le message contient des informations détaillées sur l'erreur, y compris le type d'erreur et le message associé, ce qui nous a permis de comprendre et examiner l'erreur plus facilement. On a pu distinguer les événements importants et résoudre les problèmes de notre application.

3.4 Gestion automatisée des dépendances

Nous avons choisi d'utiliser Maven comme outil permettant de gérer automatiquement les dépendances du projet.

Nous avons donc dû configurer un POM déclarant les dépendances nécessaires au projet, dans notre cas sqlite pour la base de données, junit pour les tests et slf4j pour les loggers. C'est grâce à ce fichier que nous avons pu utiliser et identifier les bibliothèques externes nécessaires à notre projet, dans leurs bonnes versions compatibles avec le projet et notre version de Intelliji. Ainsi, nous n'avons pas eu à télécharger manuellement les différentes bibliothèques employées, d'autant plus puisque Maven utilise des dépôts centralisés où se trouvent les bibliothèques de Java utiles. De plus, l'utilisation de Maven et du pom.xml permet au projet d'être cloné facilement puisque toutes les dépendances seront téléchargées automatiquement.

3.5 Bilan sur le travail de groupe

La répartition du travail était équilibrée. Nous travaillions souvent ensemble, particulièrement pour les parties concernant l'échange de messages (TCP) et la découverte des contacts (UDP). De même pour la partie UML : puisque nous avons réfléchi à l'architecture et à la conception de l'application ensemble, nous avons fait la majorité des diagrammes ensemble. Nous avons donc autant travaillé l'une que l'autre sur le projet, et la communication au sein du binôme s'est bien passée.

Nous avons avancé durant le semestre chez nous pour pouvoir poser nos questions au fur et à mesure des séances de TP. Ainsi, nous n'avons pas fini de façon précipitée et avons bien géré notre temps.

Conclusion

En conclusion, la réalisation de notre système de clavardage a été guidée par la méthode Agile. Cette approche a favorisé une adaptation constante aux besoins changeants du projet,

permettant ainsi une création efficace et évolutive de l'application. L'utilisation du modèle MVC a permis d’avoir une logique claire, facilitant la séparation des différentes préoccupations de façon organisée. De plus, les diagrammes UML nous ont permis de concevoir la structure et le comportement de notre système ainsi que de faciliter la communication entre nous concernant l’organisation de nos classes. Les tests unitaires nous ont permis de vérifier le bon fonctionnement des composants individuels en simplifiant la gestion des erreurs et le processus de débogage. Enfin, GitHub était utile pour notre collaboration, le suivi des modifications et l’automatisation des processus de développement logiciel.

ANNEXE 1 : Diagramme de cas d'utilisation

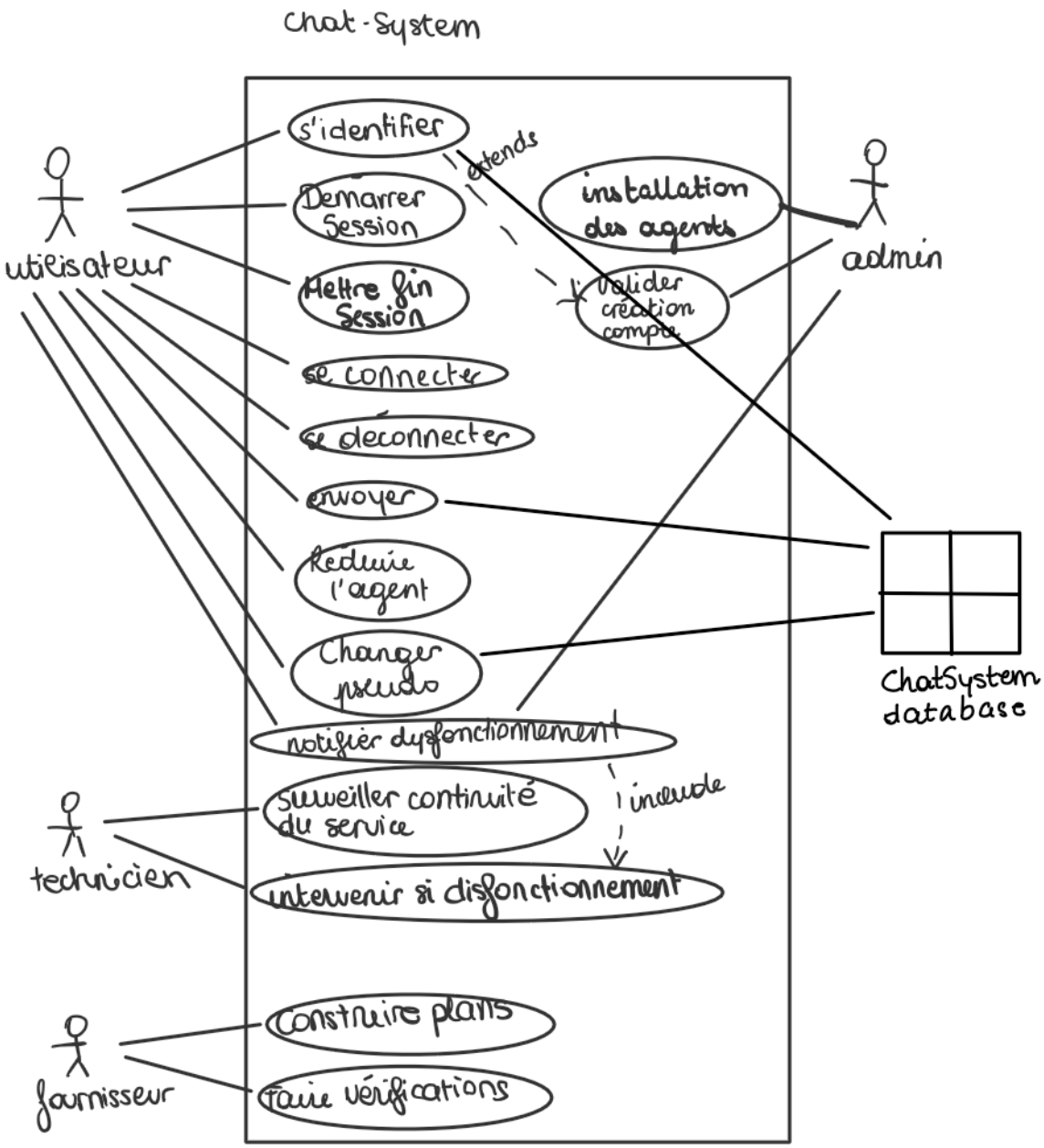

Diagramme de cas d'utilisation du système de clavardage

ANNEXE 2 : Diagramme de séquence

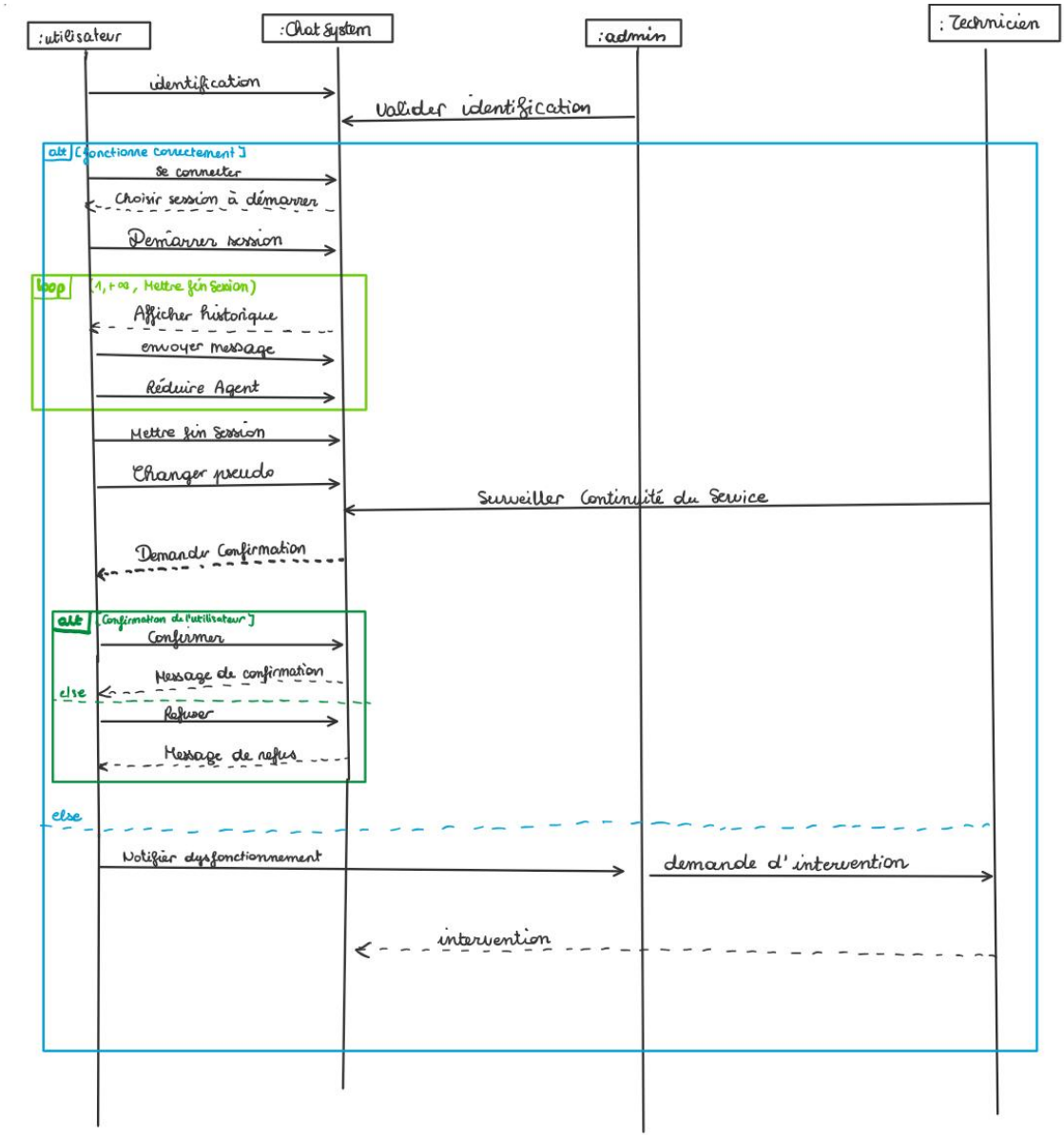

Diagramme de séquence d'utilisateur du système de clavardage

ANNEXE 3 : Diagramme de classes de la 1ère phase

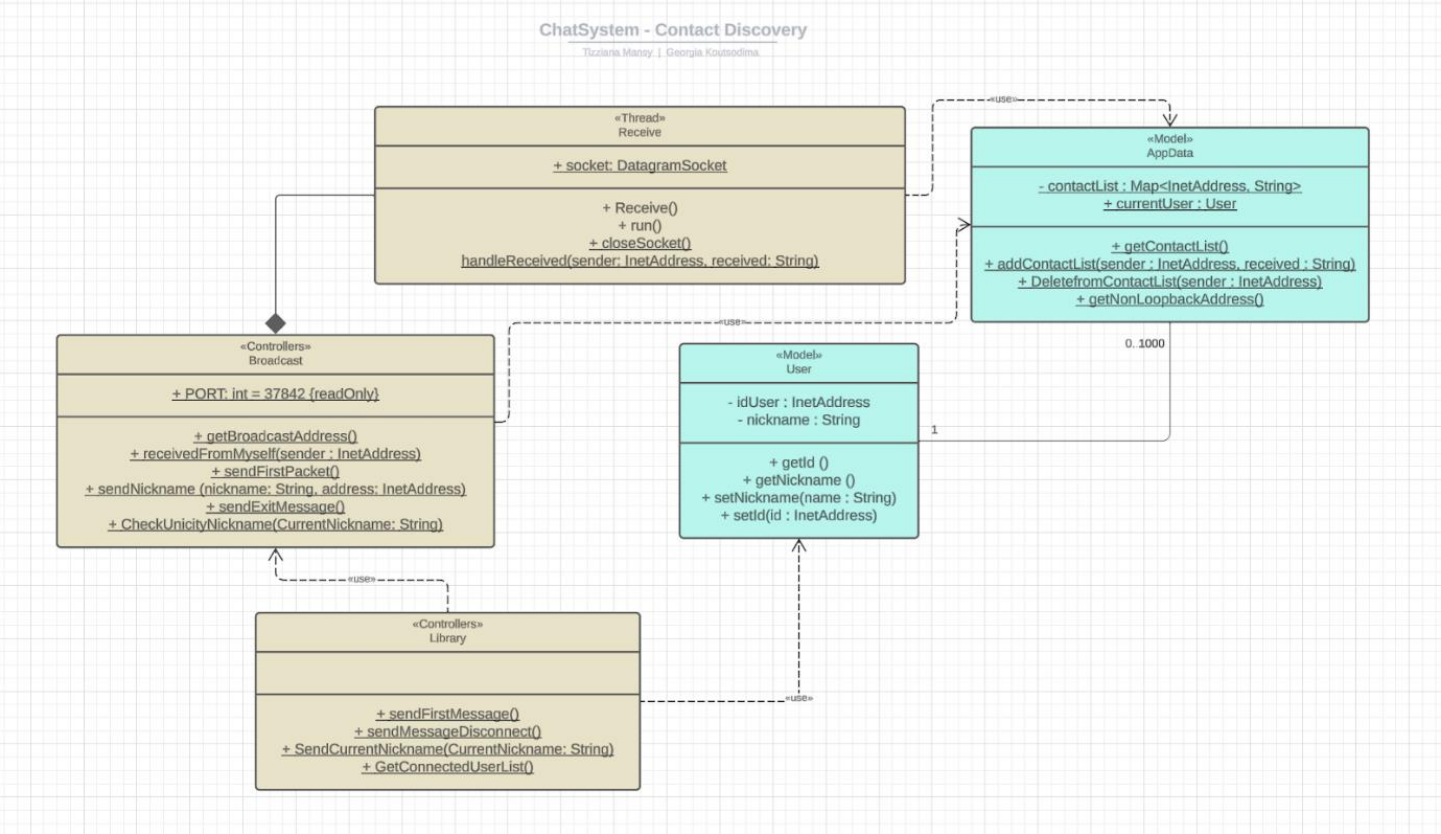

Diagramme de classes de la découverte des contacts

ANNEXE 4 : Diagramme de classes de la 2ème phase

https://drive.google.com/drive/folders/1bxNDRHFl4_TuPZ081Sl51naXiAL1oLbb?usp=drive_link [LIEN VERS DIAGRAMME EN JPG]

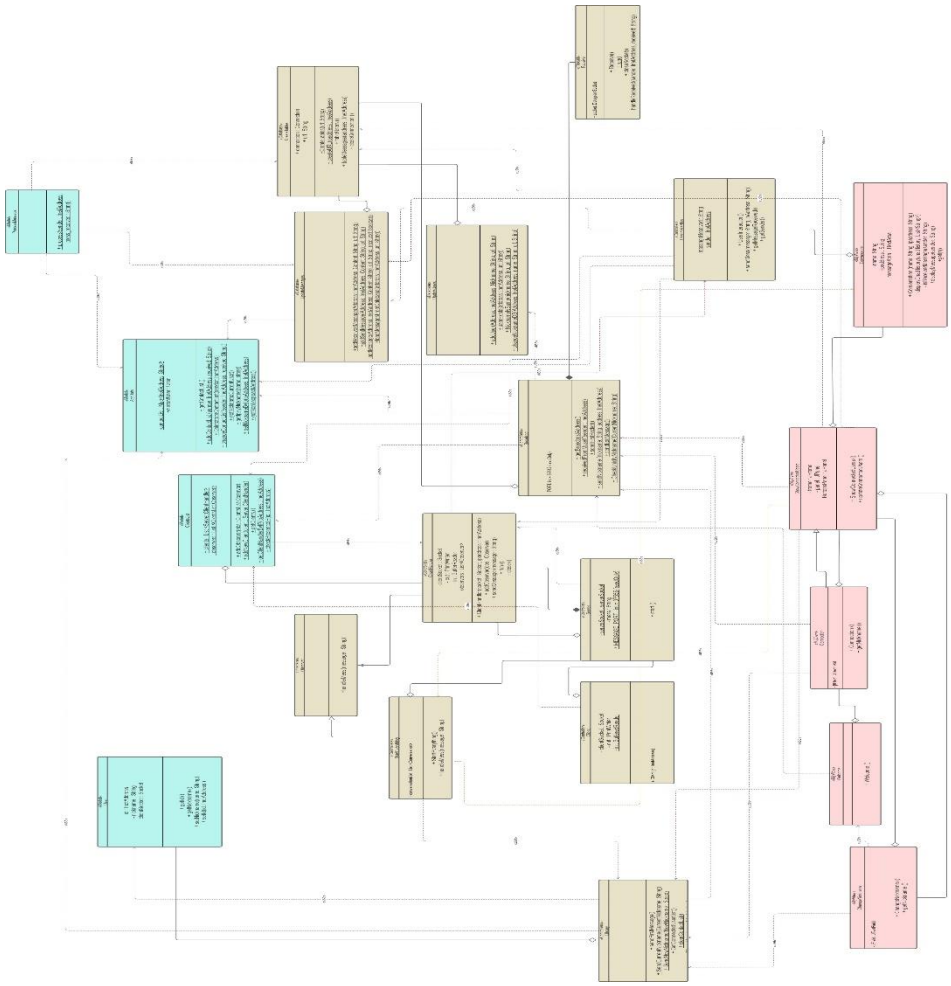

Diagramme de classes du système entier

ANNEXE 5 : Schéma de la base de données

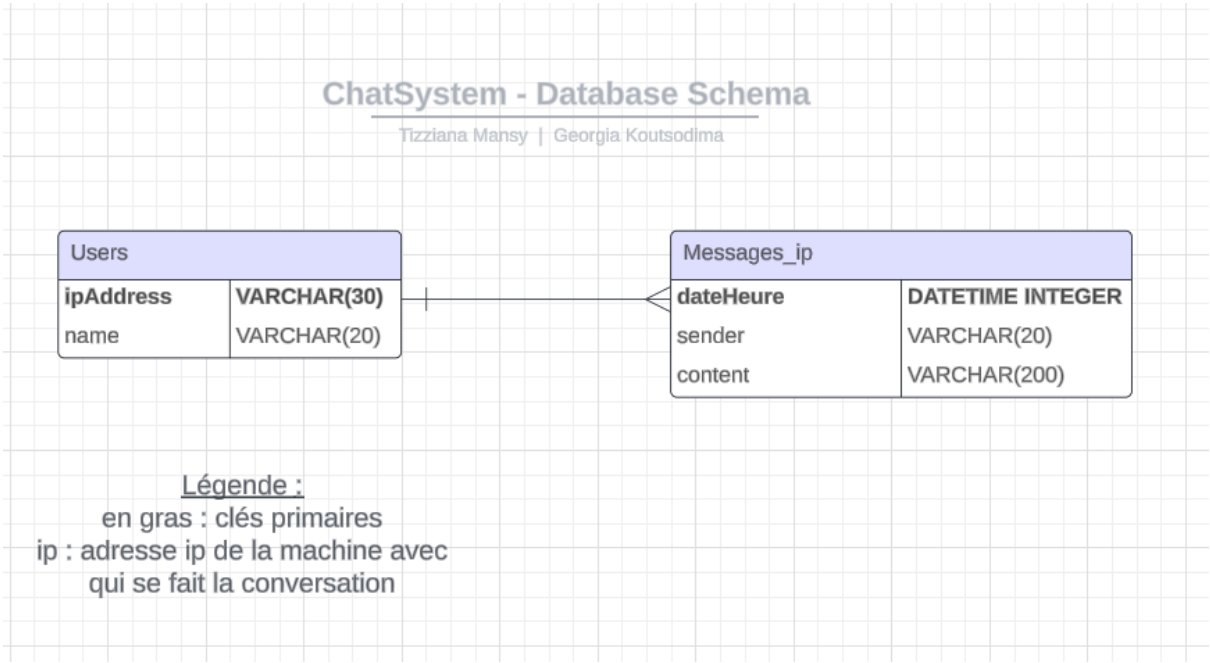

Schéma de la base de données utilisée pour l'application

ANNEXE 6 : Architecture Overview

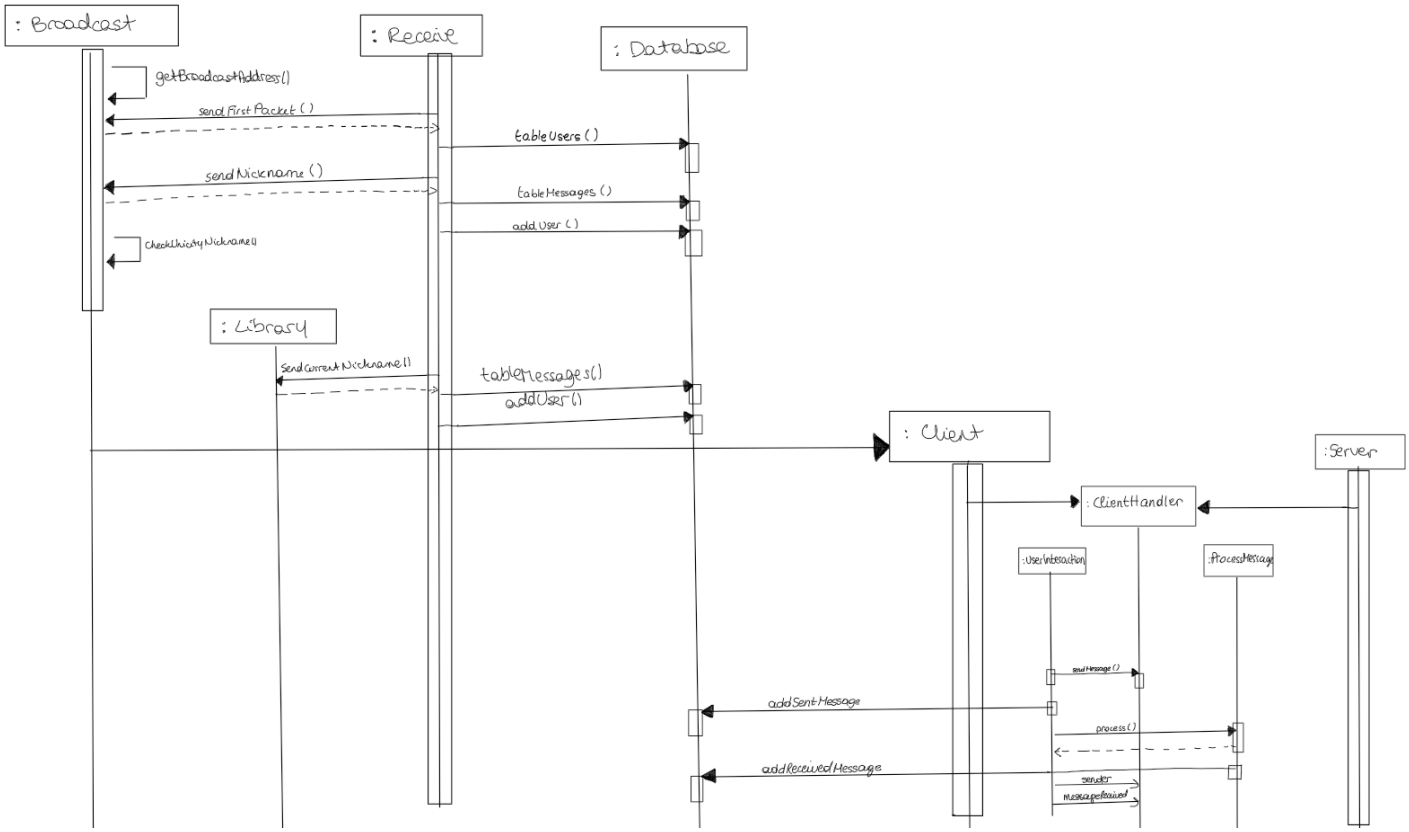

Diagramme de séquence de l'architecture globale du système

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