INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport de projet Chat System : Gestion de projet, PDLA et UML

COSSOUL Lucile

RAMOUCHE Josua

4IR

2023 – 2024

Rapport de projet Chat System : Gestion de projet, PDLA et UML

COSSOUL Lucile

RAMOUCHE Josua

4IR

2023 – 2024

Table des matières

Introduction

Ce rapport offre une perspective approfondie sur le développement de notre projet de système de chat, mettant en lumière la gestion de projet selon les principes de la méthodologie Agile et l'application du Processus de Développement Logiciel Automatisé (PDLA) jusqu'à l'intégration continue. Une section distincte est réservée à la gestion de projet et au PDLA, complétée par un rapport UML détaillé.

En ce qui concerne la gestion de projet, nous examinerons en détail l'impact positif des différentes itérations, des réunions quotidiennes de Scrum, et de la collaboration étroite avec les parties prenantes sur le succès global du projet. Pour ce qui est de la mise en œuvre du processus de développement logiciel automatisé, nous décrirons comment, dès la création du projet, nous avons automatisé les tests, augmentant ainsi l'efficacité du développement et la qualité du code produit. La collaboration au sein de notre équipe s'est articulée autour de la plateforme GitHub, où nous détaillerons la structuration de notre espace de travail, la gestion des problèmes, et les pratiques de synchronisation régulières visant à éviter les conflits.

La seconde partie du rapport se consacre à l'analyse UML, englobant les diagrammes des cas d'utilisations, de classes, de séquences, de composants, ainsi qu'une représentation schématique de la base de données.

Ce rapport vise à fournir une documentation exhaustive du projet, détaillant à la fois le processus de gestion de projet et l'aspect technique du développement à travers les outils et méthodologies utilisés.

1. Gestion de Projet et Processus de Développement Logiciel Automatisé (PDLA)

a. Gestion de Projet

Dans le cadre de notre projet de chat, l'adoption de la méthodologie Agile a grandement facilité notre progression. L'utilisation de l'outil Jira a été déterminante pour la répartition des tâches et le suivi de notre avancement. Les réunions régulières ont joué un rôle essentiel dans la communication des problèmes rencontrés, la révision croisée du code au sein du binôme, et la vérification des fonctionnalités implémentées. Cette approche a notoirement accéléré notre avancée vers la conclusion du projet par rapport à nos méthodes antérieures. Depuis le terme de notre précédent projet PDLA début décembre, nous avons effectué près de six sprints, comme illustré dans la chronologie JIRA (Figure 1).

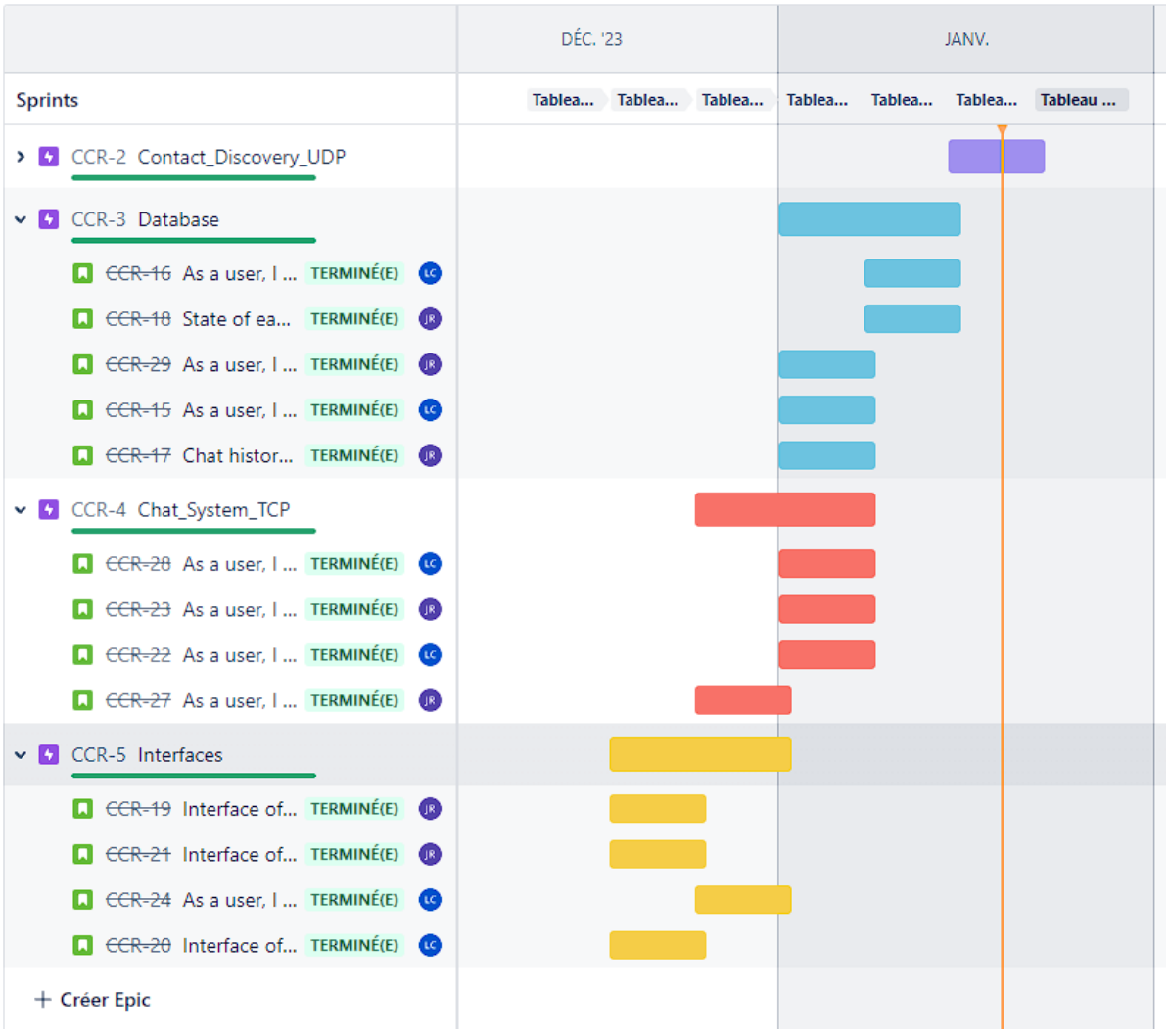

Figure 1: Chronologie JIRA (Gestion de Projet)

L'adoption de la méthodologie Agile, en particulier lors des phases 2 et 3 du projet, a permis une clarification significative des tâches à accomplir, favorisant une meilleure cohésion au sein du binôme. La durée d'une semaine pour chaque sprint s'est révélée optimale, offrant une cadence régulière de travail sur le projet et facilitant la consolidation du code.

b. Processus de Développement Logiciel Automatisé (PDLA)

Le Processus de Développement Logiciel Automatisé (PDLA) a suivi une démarche similaire à celle de la méthodologie Agile. Dès la conclusion du volet PDLA précédent, nous avons entrepris l'automatisation des tests pour notre projet. Cette approche a permis une vérification continue de l'intégrité de nos tests à chaque envoi de code sur GitHub.

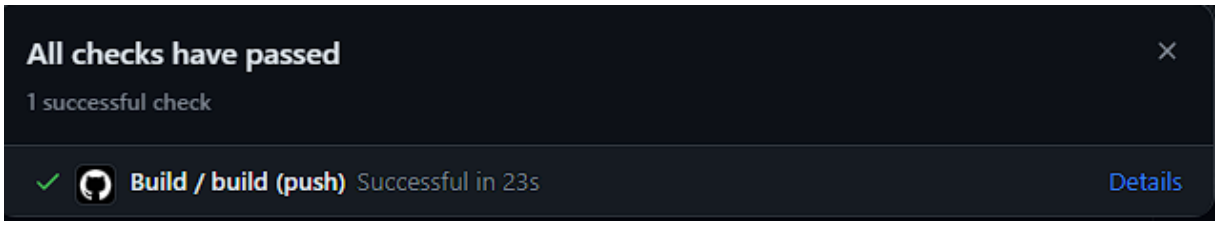

Figure 2: Build Maven automatisé GitHub

Figure 3: Tests automatisés GitHub

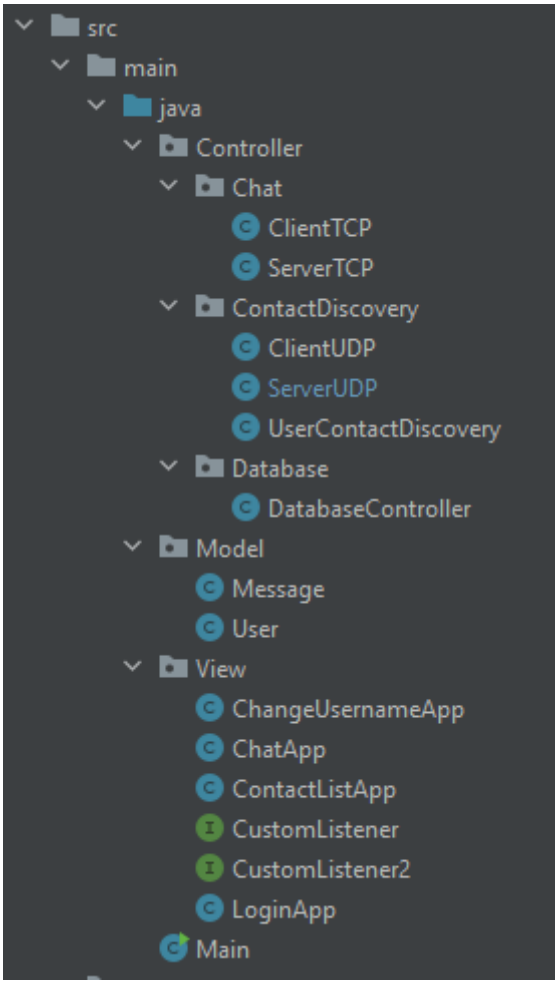

Figure 4: Organisation du projet MVC

L'organisation méticuleuse de notre projet selon le modèle Model View Controller (MVC), divisé en parties distinctes pour le TCP, l'UDP, et la base de données locale, a revêtu une importance cruciale. La structuration des tests dans un dossier test/java sous /src a suivi la même logique, éliminant ainsi toute confusion potentielle et assurant une compréhension constante des dépendances entre les différentes classes.

En outre, l'intégration de Maven dans notre processus de développement a joué un rôle clé. Maven a été utilisé pour automatiser la construction du projet, simplifiant ainsi la gestion des dépendances et l'exécution des tests. Son utilisation a contribué à une plus grande efficacité dans la gestion du cycle de vie du logiciel, garantissant une cohérence dans le processus de déploiement du code.

2. Langage de Modélisation Unifié (UML)

Pour avoir une vision globale de notre projet, il était nécessaire que nous concevions divers diagrammes représentant d'une part le comportement, d'autre part la structure de notre projet.

Ces diagrammes ont évolué tout au long de notre projet, les premiers nous permettant d'avoir une idée de ce que nous devions implémenter (à retrouver sur notre dépôt Git, dans UML Diagrams → Conception Diagrams), et les derniers représentant l'état final de notre système (à retrouver sur notre dépôt Git, dans UML Diagrams → End of Project Diagrams).

a. Acteurs et hypothèses

Avant même de se pencher sur les diagrammes, il est important d'identifier les acteurs, et les fonctionnalités principales de notre système via la spécification fournie. Pour cela, il faut pouvoir répondre à trois questions fondamentales :

Qu'est-ce que c'est ? A quoi sert le système ? Sur quoi agit-il ?

Nous devions concevoir un système de communication capable de connecter plusieurs utilisateurs sur un même réseau. Notre dispositif serait particulièrement utile en entreprise, où dans le cas d'une communication locale.

C'est dans la partie des exigences fonctionnelles que nous avons pu avoir plus de détails sur les services que notre système devait être capable de fournir.

Pour la question du sujet de l'impact de notre système, nous avons pu identifier deux acteurs principaux, à savoir l'utilisateur, pouvant faire usage de toutes les fonctionnalités de notre application, et l'administrateur qui aurait, en plus, la charge d'installer et de mettre en place le système sur les machines.

Enfin, nous devions avoir une application permettant de se connecter, d'associer un nom d'utilisateur à une adresse IP, et fournir à l'utilisateur un moyen de retrouver ses contacts et les messages qu'il a échangés. Nous en avons déduit qu'une base de données serait indispensable pour pouvoir garder en mémoire les utilisateurs de l'application ainsi que l'historique de leurs messages.

A partir de toutes ces informations, nous avions de bonnes bases pour commencer à concevoir nos diagrammes. Dans la suite, nous allons voir les diagrammes finaux de notre projet, conçus lors des dernières étapes de l'implémentation.

b. Diagramme des cas d'utilisations

Le diagramme des cas d'utilisations reprend directement les éléments présents dans la spécification, pour regrouper sur un seul schéma l'entièreté des fonctionnalités que notre système doit pouvoir fournir.

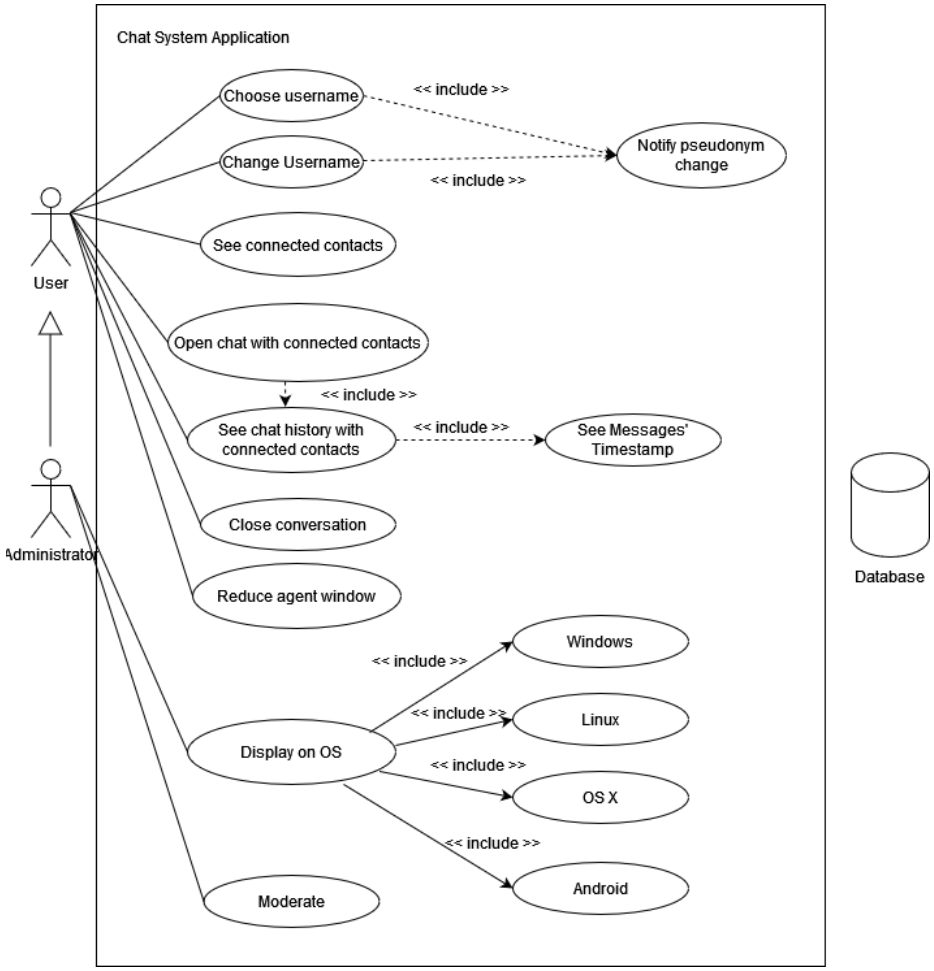

Figure 5: Diagramme des cas d'utilisations

Nous retrouvons les acteurs identifiés dans la partie précédente ainsi que la base de données qui est un composant interne au système. L'utilisateur peut effectuer diverses actions. Certaines d'entre elles incluent d'en faire une autre (pour se connecter, il faut choisir un pseudonyme).

L'administrateur hérite de l'utilisateur. Il peut dont effectuer les mêmes actions qu'un utilisateur, mais aussi s'occuper de la modération et du déploiement de l'application sur différents OS.

Ce diagramme, le premier que nous avons conçu avant l'implémentation de notre système, est le seul à ne pas avoir évolué au fil de notre projet, étant donné que les fonctionnalités restent les mêmes. Seule la façon de parvenir à ce résultat peut changer pendant l'implémentation du système.

c. Diagrammes de classes

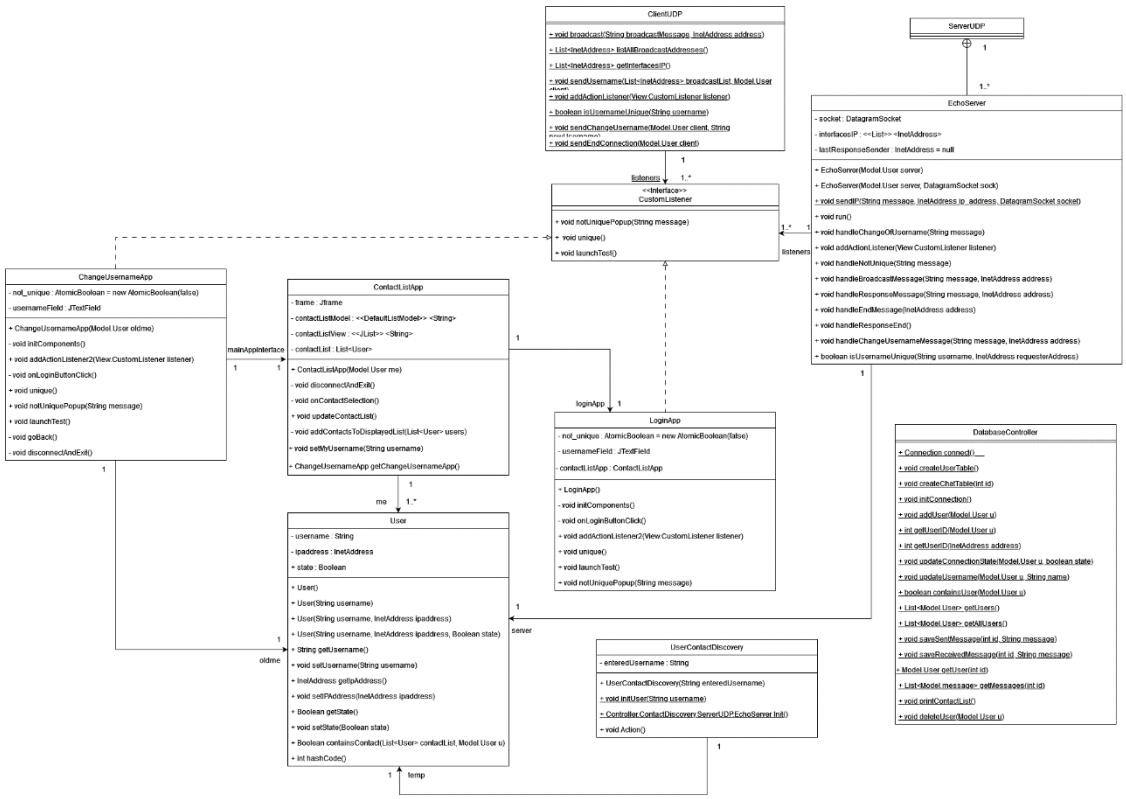

Figure 6: Diagramme de classes UDP

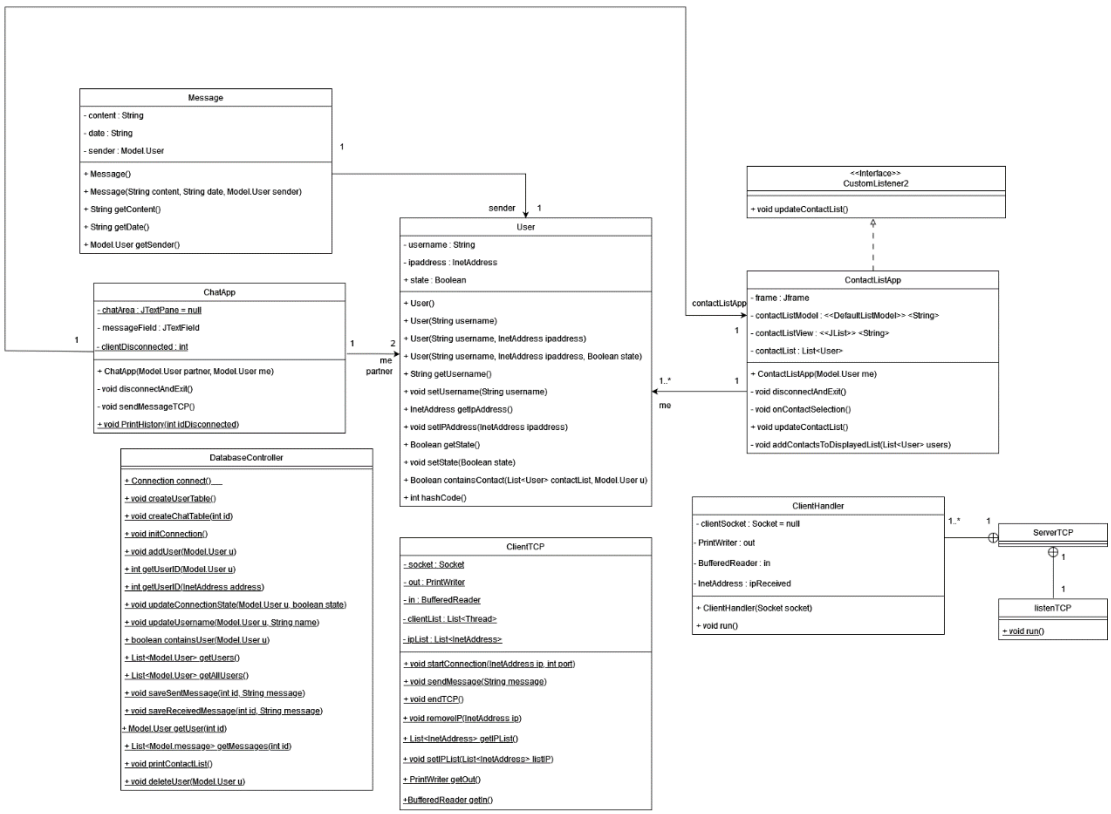

Figure 7: Diagramme de classes TCP

Les diagrammes de classes ont été séparés en deux parties, une pour la partie UDP (phase de connexion) et une pour la partie TCP (discussion par chat).

Nous avons décidé d'implémenter sur ce projet le schéma de Model View Controller (MVC).

Les classes avec la dénomination « App » correspondent aux Views de notre projet. Les classes User et Message sont des Models et le reste représente des Controllers.

Les attributs, relations et méthodes ont toutes été symbolisées sur ces diagrammes dans le but d'expliciter le fonctionnement de notre application et le lien entre les différentes classes.

La base de données joue un rôle déterminant dans notre implémentation puisqu'elle lie la phase UDP à la phase TCP via les interfaces et nous permet de mettre en place la quasi-totalité de nos fonctionnalités.

Nous avons implémenté à deux reprises le design pattern de l'observer dans notre projet.

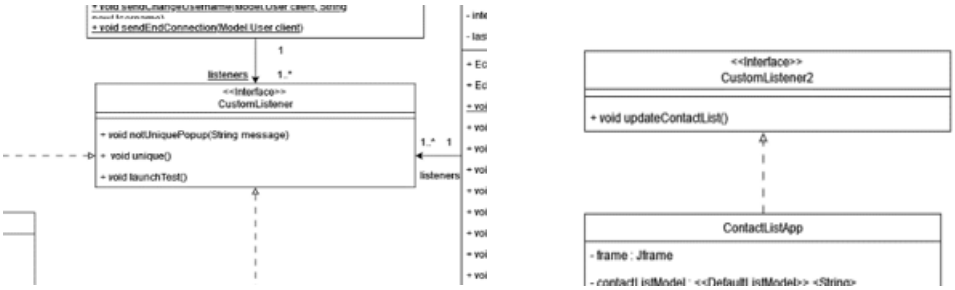

Figure 8: CustomListeners

Le CustomListener lie les classes de ServerUDP et ClientUDP (publishers) à la LoginApp et la ChangeUsernameApp (subscribers). Ici, elle propose des méthodes implémentées dans les classes subscribers.

Ce listener permet principalement de vérifier l'unicité des pseudonymes et vérifie les réponses du serveur (base de données des autres utilisateurs) ainsi que dans le client (vérification dans notre base de données).

Le CustomListener2 lie les classes LoginApp et ChangeUsernameApp (publishers) à la ContactListApp (subscriber). Ce listener permet de rafraîchir la liste de contacts via la base de données.

d. Diagrammes de séquence

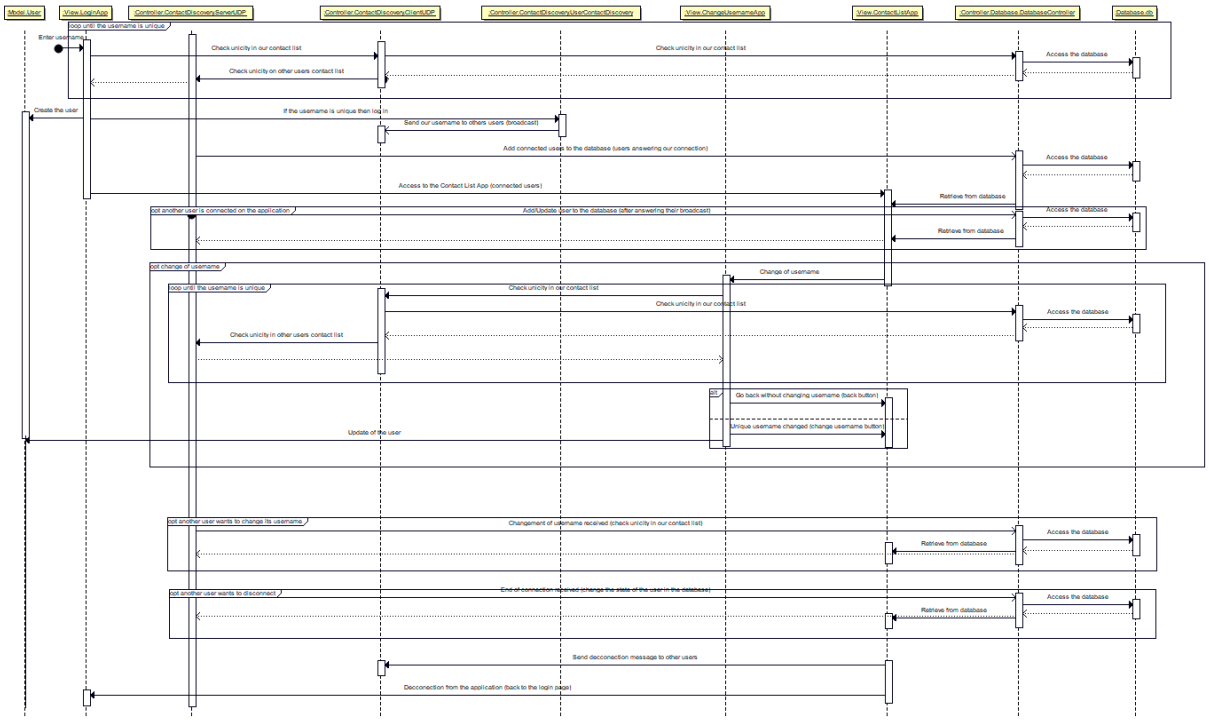

Figure 9: Diagramme de séquences UDP

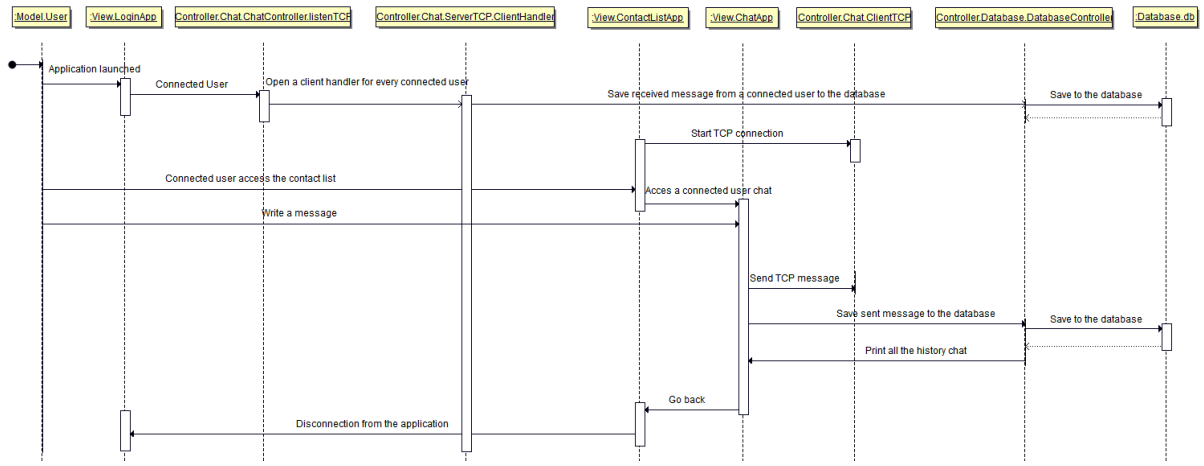

Figure 10: Diagramme de séquences TCP

Pour optimiser la compréhension de notre projet, nous avons choisi de présenter deux diagrammes de séquences afin de garantir une lisibilité accrue.

Le premier se concentre sur la partie UDP, couvrant la connexion à l'application et la récupération des utilisateurs connectés dans la liste de contacts.

Lorsqu'un utilisateur se connecte à l'application, nous effectuons une vérification pour garantir que le pseudonyme saisi ne soit pas déjà présent dans notre liste de contacts. Ensuite, nous diffusons en mode "broadcast" notre pseudonyme aux autres utilisateurs connectés sur le réseau. Ces derniers réceptionnent le pseudonyme et vérifient son unicité avant de nous

répondre en nous transmettant le leur si notre pseudonyme est unique. Dans le cas contraire, ils nous notifient de l'impossibilité d'utiliser notre pseudonyme.

Les fonctionnalités telles que le changement de pseudonyme, qui fonctionne également en garantissant l'unicité, ainsi que la déconnexion sont également intégrées dans ce processus. Cette approche vise à simplifier les interactions entre les utilisateurs, tout en assurant une gestion fluide des pseudonymes et des connexions dans notre application de chat.

Le deuxième diagramme de séquences se focalise sur la partie TCP, détaillant le fonctionnement des échanges de messages entre deux utilisateurs au sein de notre application de chat. Lorsqu'un utilisateur est connecté à l'application, deux scénarios se présentent : soit un autre utilisateur connecté prend l'initiative de nous écrire un message, soit nous prenons l'initiative de lui écrire.

Dans le premier cas, lors de la réception d'une demande de connexion TCP d'un utilisateur, nous mettons en place un thread d'écoute dédié entre cet utilisateur et nous. Ce thread permet de stocker tous les messages reçus dans la base de données, avec un thread distinct créé pour chaque connexion TCP établie.

En cas d'envoi d'un message de notre part, le processus est inversé. Nous lançons une connexion TCP avec l'utilisateur ciblé, tout en archivant les messages envoyés dans la base de données. Cette approche garantit la constitution d'un historique complet des messages échangés, contribuant ainsi à la traçabilité des conversations dans notre application.

e. Diagramme de composants et architecture

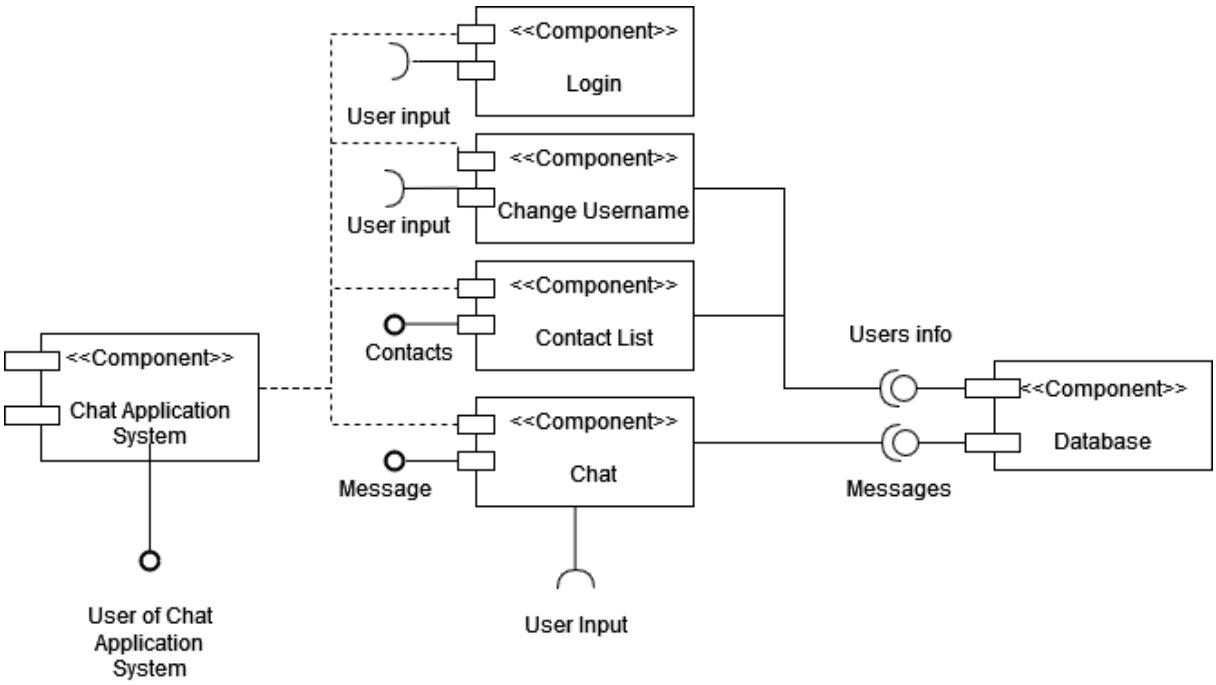

Figure 11: Diagramme de composants

En ce qui concerne l'architecture de notre projet de ChatSystem, nous avons adopté le schéma du Model View Controller (MVC) pour structurer nos classes. Au sein de cette architecture MVC, nous avons réalisé une subdivision en deux parties distinctes : d'une part, la section UDP du projet, et d'autre part, la section TCP.

Le schéma ci-dessus illustre les divers composants de notre projet, comprenant les interfaces de connexion (Login), de modification du nom d'utilisateur (Change Username), de la liste de contacts (Contact List), ainsi que des discussions (Chats). Pour la conception de ces interfaces, nous avons fait appel à la technologie Swing.

Chaque composant est lié à une base de données locale propre à chaque utilisateur. Dans le cadre de ce projet de ChatSystem, nous avons choisi d'utiliser SQLite, permettant ainsi la création d'un fichier .db lors de l'utilisation de l'application. Cette approche nous offre la possibilité de sauvegarder les informations relatives aux utilisateurs, ainsi que l'historique des messages échangés avec ces derniers.

f. Schéma de la base de données

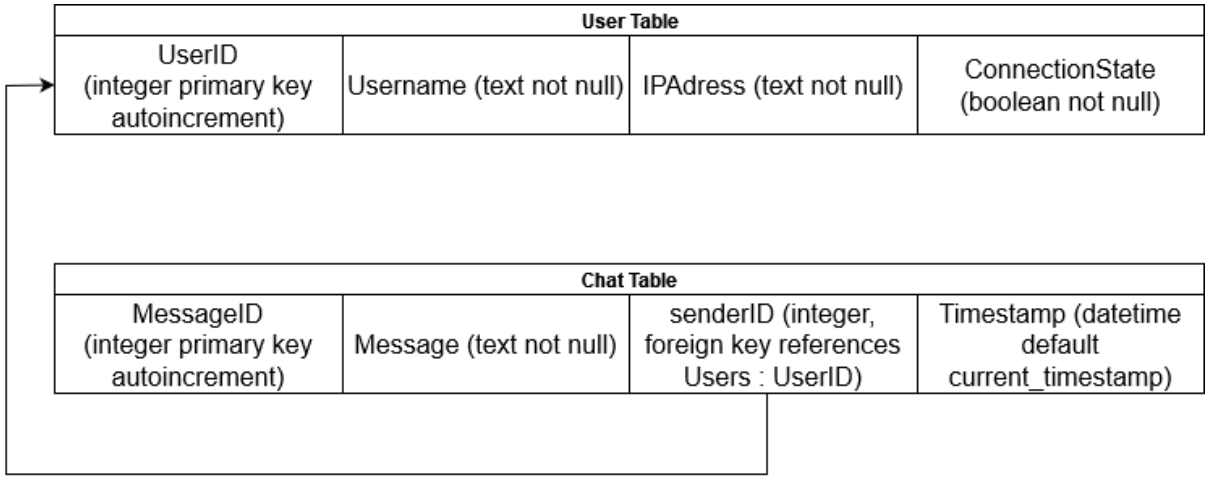

Figure 12: Schéma de la base de données

Le diagramme ci-dessus illustre la configuration de notre base de données locale. Dans le cadre de ce projet de chat, nous avons mis en place deux structures de table distinctes.

La première, nommée "User Table", est dédiée à la conservation des informations sur les utilisateurs, qu'ils soient connectés ou non. Chaque utilisateur dispose d'une unique instance de cette table.

La seconde table, nommée "Chat Table", est créée en autant d'exemplaires qu'il y a de connexions TCP établies avec des utilisateurs connectés. En d'autres termes, une table distincte est dédiée à chaque utilisateur connecté à l'application et désirant échanger des messages avec nous. Chaque "Chat Table" est spécifiquement reliée à un utilisateur connecté avec lequel nous échangeons des messages, permettant ainsi une gestion individualisée de chaque conversation.

Conclusion

En conclusion, ce rapport reflète notre parcours passionnant dans le développement d'un système de chat. Nous avons navigué avec succès à travers les rouages complexes de la gestion de projet Agile, mettant en évidence les retombées positives des réunions de Scrum quotidiennes. L'implémentation du Processus de Développement Logiciel Automatisé dès les premières étapes a considérablement renforcé notre efficacité de développement, tout en élevant la qualité du code généré.

La cohésion au sein de notre équipe, facilitée par GitHub, a été un pilier essentiel. Nous avons instauré une structure de travail harmonieuse, résolu les problèmes de manière proactive, et maintenu des synchronisations régulières pour prévenir les conflits.

La seconde partie du rapport dévoile une analyse UML détaillée, offrant un éclairage approfondi à travers divers diagrammes. Cette approche permet une compréhension approfondie de la dimension technique du développement.

Pour conclure, ce rapport constitue une synthèse complète, fusionnant la perspective de gestion de projet avec les détails techniques du développement. Le chemin parcouru témoigne de notre détermination à créer un produit de qualité, résultat d'une collaboration dynamique et d'une approche méthodique.

Annexes

Afin d'optimiser la lisibilité, l'ensemble des diagrammes mentionnés dans ce rapport se trouve regroupé dans le répertoire "UML Diagrams/End of project Diagrams" sur GitHub.

Les diagrammes initialement élaborés avant le début du projet, durant la phase de conception, sont également accessibles dans le dossier "UML Diagrams/Conception Diagrams".