INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Rapport de projet Clavardage

Année 2022-2023

Sacha Cauli - Marco Ribeiro Badejo

Rapport de projet Clavardage

Sommaire

Introduction

Dans le cadre des cours de Conception Orientée Objet, de Programmation avancée en Java, et de Conduite de projet, nous avons été amené à réaliser une application de messagerie en temps réel. Cette dernière avait pour objectif de permettre l'échange de messages entre différents utilisateurs appartenant à un même réseau local, et devait remplir un cahier des charges fourni lors de la première séance.

Dans un premier temps, il nous a donc fallu analyser ce cahier des charges, afin d'en retirer les besoins et attentes du client, puis d'entamer la réflexion concernant la phase de conception du projet. Cette phase ne sera pas détaillée dans ce rapport, puisqu'elle a été traduite sous la forme de diagrammes UML, qu'il est possible de trouver sur le dépôt Github (dossier Conception).

Nous avons ensuite abordé la phase d'implémentation, lors de laquelle de nombreux choix techniques ont dû être effectués, imposant des corrections sur certaines décisions prises lors de la phase de conception. Cette phase sera décrite dans une première partie, avant d'expliciter les tests implémentés dans une seconde partie. Les outils utilisés pour la réalisation du projet, ainsi que la façon dont celui-ci a été géré (notamment en ce qui concerne le travail collaboratif) seront abordés dans une troisième partie. Enfin, une quatrième et dernière partie expliquera succinctement le fonctionnement de l'application, du point de vue utilisateur.

Notons que du fait des circonstances dans lesquelles ce projet a été réalisé (l'un des étudiants étant en semestre d'échange), il n'a pas été possible de tester le projet autrement qu'en local, et certaines fonctionnalités n'ont pas pu être complètement implémentées.

Implémentation et choix techniques

Lors de la phase d'implémentation, nous nous sommes rapidement aperçu que le diagramme de classe produit lors de la phase de conception était loin d'être parfait. S'agissant d'un projet à but éducatif, il s'agit là d'un problème que nous pensions rencontrer durant cette phase. Nous avons par conséquent estimé que nos diagrammes seraient amenés à évoluer en fonction des choix faits lors de l'implémentation, certainement à l'inverse de la manière dont se déroule le développement d'un logiciel en entreprise.

Par conséquent, un diagramme de classe de la version finale de notre projet est disponible en annexe de ce rapport, ainsi que dans le dossier Conception du dépôt Github (dans le cas où l'annexe de serait pas suffisamment lisible).

A. Environnement de travail

Pour réaliser ce projet, nous avons utilisé Maven (version 3.8.1) ainsi que Java (version 11). Nous avons fait le choix d'utiliser l'environnement de développement intégré IntelliJ pour sa simplicité d'utilisation.
Dans un premier temps, nous avons réalisé ce projet dans les salles de TP, ce qui permettait de tester notre programme sur plusieurs machines à la fois (notamment lors de l'envoie de trames TCP ou UDP).
Dans un second temps (à compter des vacances de Noël), l'un des étudiants partant en semestre d'échange, il a fallu envisager une façon de continuer le développement du projet en ne pouvant effectuer nos tests qu'en "local". Par conséquent, une large partie du projet a été modifiée, pour fonctionner en local et pouvoir valider l'avancement du projet par des tests, ce qui a pu conduire à certains choix d'implémentation qui pourrait être remis en cause.

B. Messages de service (UDP)

Nous avons utilisé le protocole UDP afin d'envoyer des messages de service, assurant le bon fonctionnement de l'application. Ces messages pouvaient être envoyés en broadcast ou en unicast, le choix étant basé sur une volonté de réduire la charge du réseau (les messages ne sont donc envoyés en broadcast que lorsque cela est réellement nécessaire).

Cinq types de messages UDP ont été créés, et sont identifiés par un code numérique allant de 001 à 005.

Code	Utilité
001	Message de découverte du réseau et de

	partage du pseudo choisi, envoyé en broadcast. L'expéditeur attend une réponse des utilisateurs qui ont reçu ce message, afin de les ajouter dans une base de données des utilisateurs connectés. Ce message n'est envoyé qu'une fois, au démarrage de l'application.
002	Message de réponse à une trame 001 ou 004, permettant de prévenir le destinataire que l'on est connecté, mais que le pseudo choisi par ce dernier est déjà utilisé.
003	Similaire au message 002, mais le pseudo choisi par le destinataire est cette fois valide.
004	Message permettant de partager un nouveau pseudo (lors du changement, fonctionnalité partiellement implémentée et non disponible). Le message contient l'ancien pseudo, ainsi que le nouveau.
005	Message envoyé en broadcast permettant de prévenir les destinataires que l'expéditeur se déconnecte. Fonctionnalité non implémentée.

C. Network Manager

L'un des packages les plus volumineux de ce projet est le package NetworkManager, qui contient une classe homonyme, ainsi que les classes suivantes : ServerTCP, ClientTCP, ServerUDP, ClientUDP.

Les deux classes clientes possèdent des méthodes static permettant l'envoi de messages depuis le NetworkManager sans avoir à créer d'instance. Les deux classes serveurs, sont quant à elles créées par le Launcher (ServerUDP) ou par le ThreadManager (ServerTCP).

La classe NetworkManager était chargée du fonctionnement global de l'application. Elle était en charge du lien entre l'interface graphique, les bases de données, et l'échange de messages.
Lorsque l'un des serveurs (que ce soit TCP ou UDP) recevait un message, ce dernier était transmis à la classe NetworkManager, qui - dans le cas des trames TCP - l'insérait dans la base de données puis l'affichait ou bien - dans le cas des trames UDP - réalisait un traitement associé au contenu de la trame. Elle permettait également l'accès aux deux bases de données utilisées au sein de projet : l'une permettant de stocker les conversations (base de données classique), l'autre permettant de stocker les utilisateurs connectés (base de données constituée d'une HashMap).

Le diagramme suivant permet de voir le rôle que joue la classe NetworkManager lors de la réception d'un message sur un serveur TCP actif.

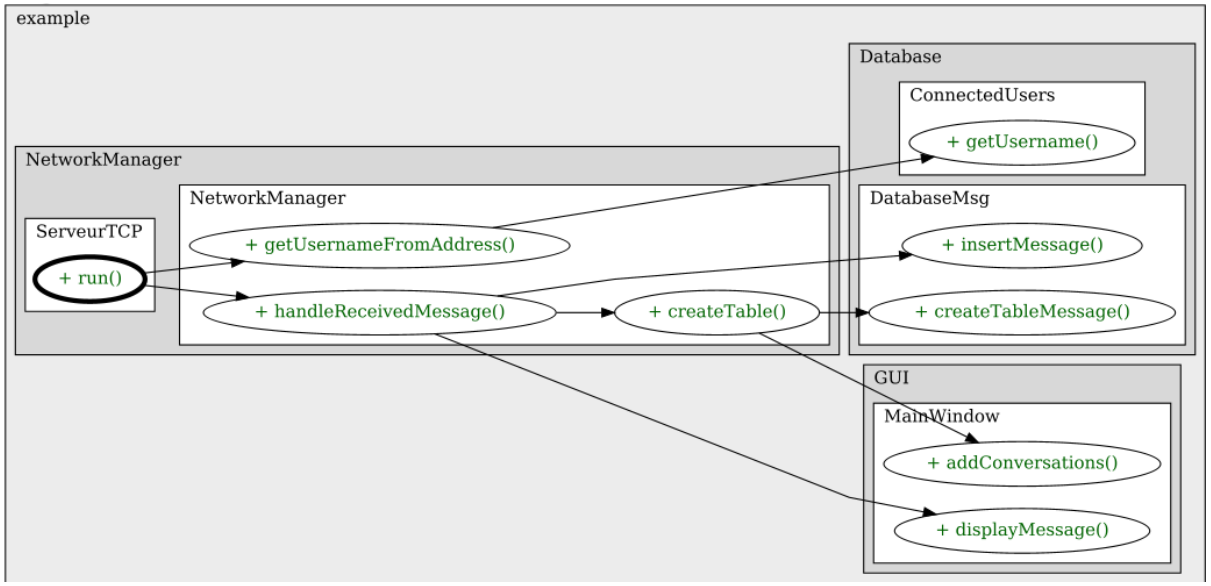

Diagramme 1 : Réception d'un message depuis un serveur TCP

D. Thread Manager

Cette classe possède un fonctionnement relativement simple, et a pour but de créer un nouveau serveur TCP à chaque fois qu'une demande de connexion est reçue sur le port utilisé, puis de lancer ce nouveau serveur dans un thread séparé. Ainsi, chaque conversation est associée à un Thread, ce qui permet de discuter avec plusieurs utilisateurs différents en même temps (puisqu'il n'est pas possible de connecter plusieurs clients à un même serveur).

E. Base de données

Comme mentionné dans la partie C., plusieurs bases de données sont utilisées pour assurer le bon fonctionnement de l'application.
La première, qui correspond à la classe DatabaseMsg, permet de stocker les messages échangés. Pour cela, une table est créée pour chaque conversation initiée ou reçue par l'utilisateur (si la table n'existe pas déjà). Le nom de la table correspond au pseudo de l'utilisateur distant. A l'intérieur de chaque table est stocké le contenu des messages, l'heure de réception ou d'envoi (sous forme de chaîne de caractères) ainsi qu'un entier permettant de déterminer s'il s'agit d'un message reçu ou envoyé. Une méthode permet de mettre à jour le nom d'une table lorsqu'un utilisateur désire changer son pseudo.

La seconde base de données stocke les correspondances adresses IP et pseudo pour chaque utilisateur connecté. Des méthodes sont disponibles afin de récupérer l'adresse IP

correspondant à un pseudo et vice versa (nécessaire pour l'envoi de message). Lors de la déconnexion d'un utilisateur, l'entrée correspondante est supprimée.

F. Difficultés

Malheureusement, nous avons accumulé les retards lors de la réalisation de ce projet, ce qui a eu un impact sur son avancement. Certaines fonctionnalités n'ont pas pu être implémentés, et d'autres ne sont que partiellement fonctionnelles.

Ainsi, la fonctionnalité de choix du pseudo au démarrage ne permet pas de choisir un pseudo contenant des espaces ou des caractères spéciaux, et ne gère pas le cas des conflits (cas où un utilisateur choisit un pseudo déjà utilisé). Il n'est pas possible de modifier son pseudo une fois une session démarrée. Enfin, la déconnexion d'un utilisateur n'est pas gérée. Le code permettant d'implémenter ces fonctionnalités est partiellement présent sur le dépôt Github, mais a été commenté afin d'éviter tout problème de fonctionnement.

Le fait de réaliser ce projet en ne pouvant que le tester en local à eu de grosse répercussion sur son avancement, et nous a poussé à faire des choix parfois discutables. À titre d'exemple, la classe NetworkManager n'avait pas un rôle aussi centrale dans les premières versions de notre projet, et une partie du code a été migré dans celle-ci afin d'assurer une indépendance du programme vis à vis du Launcher (puisque nous avons utilisé deux Launchers durant la phase de développement, il était nécessaire que le code n'est pas besoin d'une instance de l'un des launcher pour s'exécuter normalement).

Tests unitaires

Pour tester les différents composants de notre projet, nous avons eu recours à des tests unitaires (JUnit). Nous possédons deux classes de test, l'une permettant de tester le bon fonctionnement des échanges UDP, qu'il s'agisse de l'envoie d'une broadcast ou d'une trame à destination d'un utilisateur en particulier, et l'autre permettant de tester le comportement de la base de données chargée de stocker les conversations d'un utilisateur donné.
Les tests des interfaces graphiques ont été réalisés "à la main", en exécutant le programme et en vérifiant son bon comportement.

A. Tests UDP

La classe UDPTests contient deux méthodes de test, dont le comportement est similaire : un serveur est démarré dans un thread séparé, puis un client envoie une trame (en broadcast dans un cas, en unicast dans l'autre). Ne sachant pas comment effectuer des tests unitaires sur un programme utilisant plusieurs thread, nous avons - sur conseil de notre professeur - créé un handler dans le serveur. Ce handler joue un rôle de setter, et assigne à un attribut de la classe de test la valeur reçue par le serveur. Puisque ce setter est exécuté depuis un thread secondaire, nous endormons le thread principal le temps que le setter remplisse sa mission. Pour éviter les attentes trop longues, l'appel à la méthode Thread.sleep() est placé dans une boucle while bornée en temps, et dont on sort lorsque la modification apportée par le setter est effective.

B. Tests Base de données

La classe DatabaseTests contient quant à elle 3 tests, liés aux requêtes sql utilisées par le logiciel. Ainsi, ces tests permettent de tester l'insertion de données dans une table, la création d'une table, ainsi que la modification de l'une des tables. Ce dernier test est nécessaire car la fonctionnalité de modification du pseudo d'un utilisateur distant implique une mise à jour de la table associée à cet utilisateur (si elle existe) chez chacun des autres utilisateurs du système.

C. Tests à implémenter

Il serait nécessaire d'ajouter certains tests à notre projet pour vérifier d'autres comportements, tels que l'échange de trame TCP, ou le stockage des utilisateurs connectés. Cependant, par manque de temps, ces tests n'ont pas pu être développés avant l'échéance du projet, fixée au 27 janvier.

Conduite de projet

A. Github

Afin de simplifier le travail collaboratif durant le développement de ce projet, nous avons eu recours à un outil largement utilisé par la communauté de développeurs, à savoir GIT (et plus particulièrement Github). Notre dépôt, accessible à l'adresse https://github.com/VassiliKan/Clavardage, a été créé dès la première séance, et nous a permis d'une part de travailler à deux personnes sur un même projet, et d'autre part de s'assurer d'avoir une sauvegarde de la version la plus avancée du projet. Notons qu'IntelliJ simplifie l'utilisation de GIT, en permettant d'effectuer un commit, un push ou un pull en un seul clic.

B. Jenkins

Conformément à ce que nous avons appris durant les cours de conduite de projet, nous avons utilisé un outil d'intégration continue tout au long du développement du projet Clavardage. Nous avons créé deux jobs sous Jenkins, un premier en charge de réaliser un build du projet lorsqu'un push était réalisé sur le dépôt Github, et un second en charge de réaliser des builds du projet présent en local sur les ordinateurs, à intervalle de temps régulier (toutes les 5 minutes).

Jenkins semble être un outil très intéressant, mais nous ne sommes pas certains de l'intérêt que son utilisation présentait durant la réalisation de ce projet (générer des exécutables n'était ici pas utile).

C. Agile

L'objectif de ce projet étant également l'utilisation des méthodes de développement Agile, nous avons eu recours à l'outil en ligne Jira. Cet outil nous a permis de suivre l'évolution du projet, et d'éviter de "s'éparpiller". Cependant, nous avons rapidement réalisé que lors de la phase d'évaluation du poids à accorder à certaines user story, nous avons eu tendance à en surestimer certaines.

En condition normale, Jira est un outil qui semble très utile. Cependant, les problèmes apparus lors du passage de l'environnement de développement disponible à l'INSA à celui disponible en n'utilisant qu'une seule machine (en local) ont eu pour conséquence d'amoindrir l'utilité de la planification Agile. En effet, il a été nécessaire de revenir sans cesse sur certaines user story qui étaient censées être achevées, et de retarder certaines

qui se trouvaient dans le prochain sprint. De plus, le fait de terminer un sprint et de transférer les users story restantes dans le sprint suivant ne démarre pas automatiquement ce dernier. Il est donc fréquemment arrivé de réaliser que le sprint en cours n'était finalement pas démarré dans Jira, ce qui a certainement impacté la timeline du projet (sans importance mais cela reste regrettable car il est par conséquent difficile de voir l'évolution du projet).

Notre projet Jira est accessible à l'adresse suivante : https://badecau.atlassian.net/jira/software/projects/CLAV/boards/1/backlog.

Manuel simplifié

Lors du lancement du programme, la fenêtre suivante apparaît et attends la saisie d'un nom d'utilisateur.

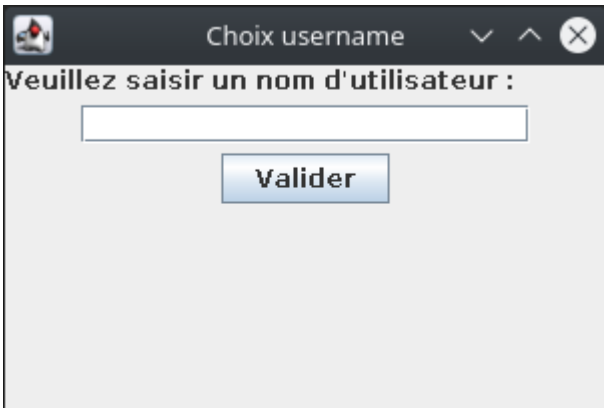

Image 1 : Fenêtre permettant de choisir un nom d'utilisateur

Le programme n'étant pas finalisé, il est nécessaire de choisir un nom d'utilisateur ne contenant ni espaces (contrairement à ce qui est visible sur l'image 2) ni caractères spéciaux. Une simple fonction de filtrage permettra de régler ce problème dans une future version du logiciel. Une fois le nom d'utilisateur choisi, il suffit de cliquer sur le bouton "Valider" pour que la fenêtre principale s'ouvre.

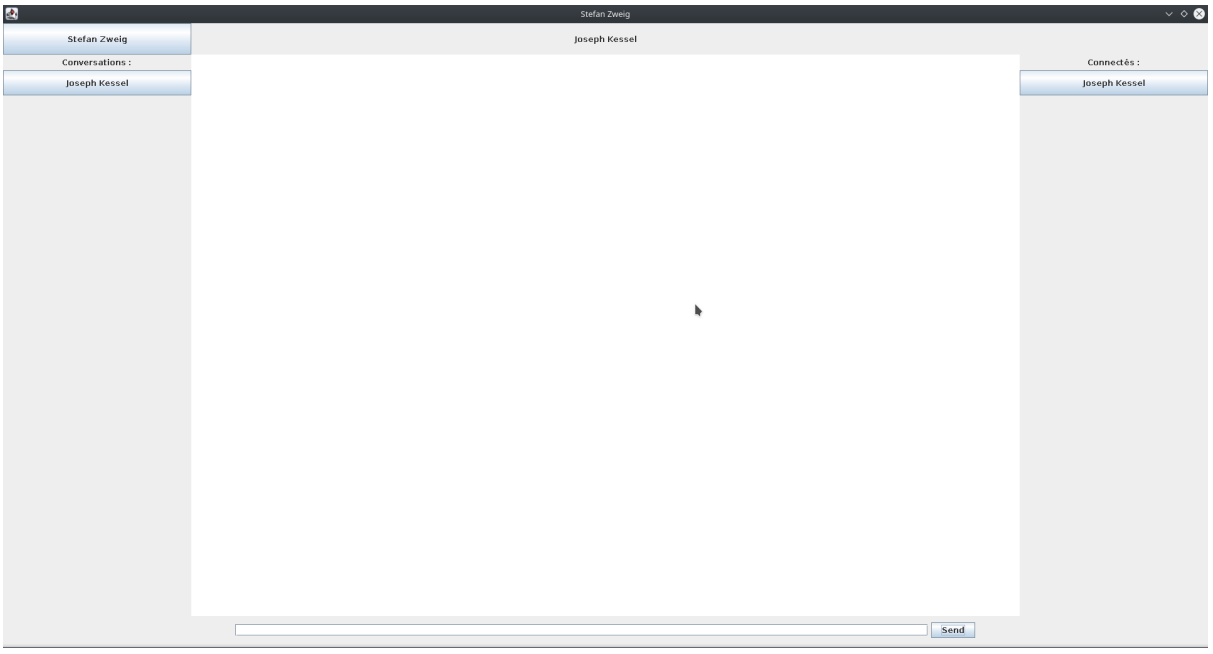

Image 1 : Fenêtre principale

La fenêtre principale permet d'effectuer l'intégralité des actions attendues de l'application. En haut à gauche se trouve un bouton ("Stefan Zweig"), qui correspond au nom d'utilisateur choisi pour la session actuelle. En cliquant dessus, il sera possible de modifier son pseudo (fonctionnalité qui n'est pour l'heure pas disponible).

Sur la gauche, en dessous du label “Conversations” se trouve une liste de boutons contenant l’historique des conversations. En cliquant sur l’un de ces boutons, la conversation correspondante s’affiche.

Sur la droite, en dessous du label “Connectés” se trouve également une liste de boutons, correspondants aux utilisateurs actuellement connectés.

En haut, se trouve un label correspondant au destinataire du message. Ce label est mis à jour lors d’un clic sur un bouton correspondant à une conversation ou à un utilisateur connecté. Pour envoyer un message à “Joseph Kessel”, il est par conséquent nécessaire de cliquer sur son nom (que ce soit dans la barre de gauche ou de droite), de taper un message dans le textField situé en base de la fenêtre, puis de cliquer sur le bouton “Send”.

Pour que Joseph Kessel voit ce message s’afficher dans son écran, il devra cliquer sur l’un des boutons situés à gauche ou à droite correspondant au nom de l’expéditeur (aucune fonctionnalité de notification n’est pour le moment disponible).

Conclusion

Ce projet nous a permis d'aborder l'intégralité des étapes du développement d'un logiciel, en utilisant des méthodes (Agile) et des outils (Github, Jenkins) très utilisés par les professionnels. Il a également permis de mettre en pratique certains enseignement dispensés dans d'autres modules, tels que le cours de réseau ou d'algorithmique répartie (ce dernier cours n'a pas été réellement mis en pratique mais nous avons envisagé d'utiliser certains algorithmes, avant de nous raviser pour nous tourner vers des solutions plus optimales pour la charge du réseau).

L'application que nous avons codée est loin d'être achevée, mais certaines difficultés mentionnées dans ce rapport ont retardé son développement. Beaucoup de travail est encore nécessaire afin de finir l'implémentation des fonctionnalités manquantes et de reprendre la gestion des erreurs qui est pour le moins incomplète. Cependant, le fonctionnement général est assez satisfaisant, et nous semble ergonomique et simple à déployer sur l'environnement visé.

Annexes

Annexe 1 : Diagramme de classe UML du projet en l'état

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