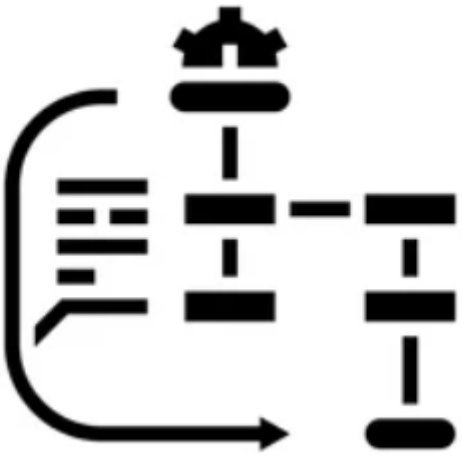
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Projet JAVA Chatsystem : UML & PDLA



MIRANVILLE Florent
HENRIET Baptiste

Tuteurs:
Sami Yangui
Nawal Guermouche

24 Janvier 2024

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Sommaire

I. UML Designs

Le langage de modélisation unifié (UML) fournit une méthode visuelle standardisée pour représenter les concepts et les interactions dans le développement logiciel. La communication entre les membres de l'équipe est simplifiée grâce l'utilisation de différents diagrammes, facilitant ainsi la compréhension, la documentation et la gestion du cycle de vie du projet. Ce rapport explore divers diagrammes UML utilisés pour modéliser notre système.

A. Use Case Diagram

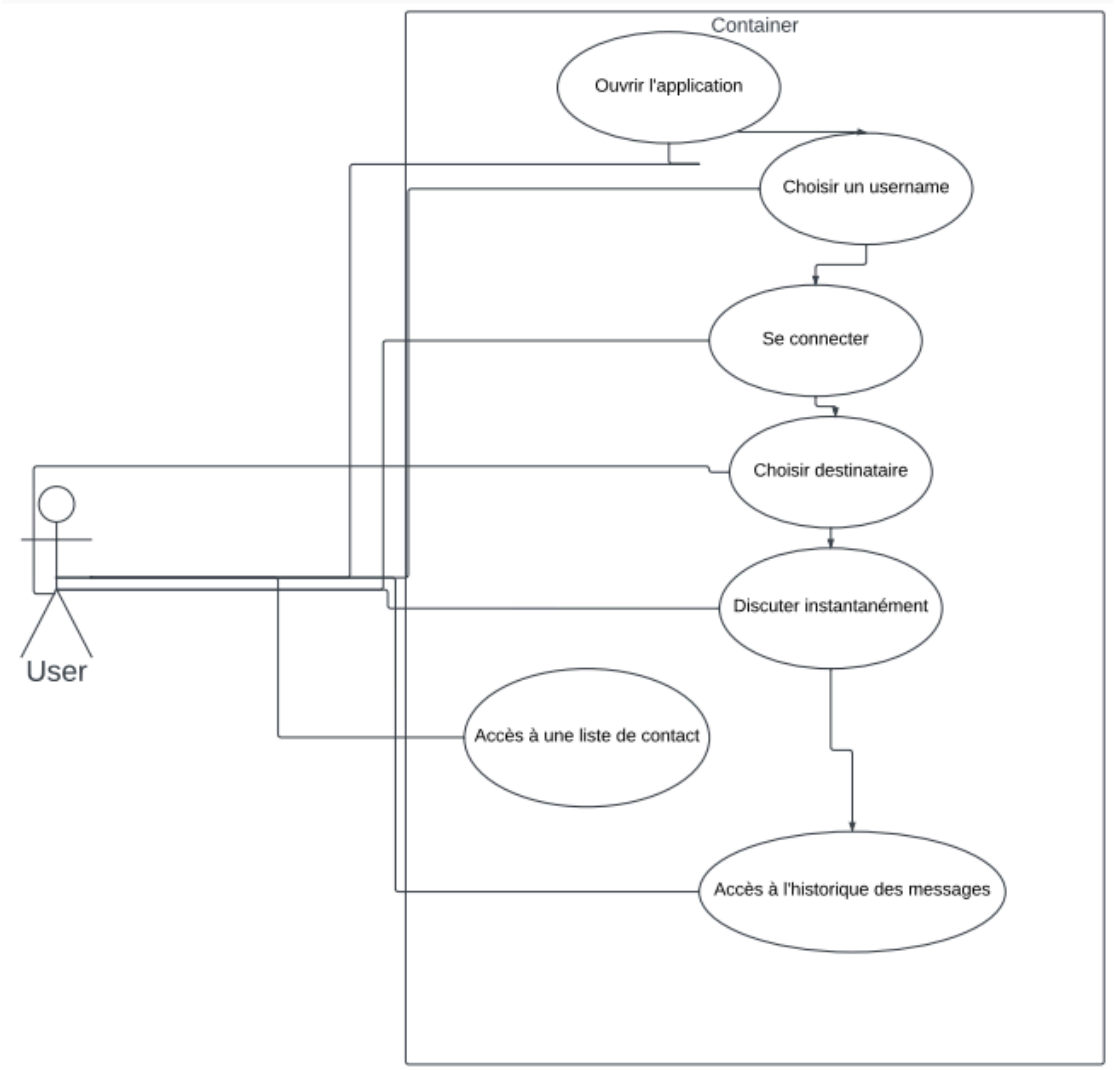


Diagramme de scénario du projet ChatSystem

Ce diagramme nous a permis d'avoir une idée globale des fonctionnalités phares du projet. Notre projet permet une discussion instantanée entre des utilisateurs distants via une interface graphique. La fonctionnalité principale étant la communication directe, et le fait que d'autres assets soient automatiques et cachées de l'utilisateur font que les différents scénarios sont étroitement liés et suivent un ordre particulier. En effet, pour engager une discussion, l'utilisateur doit impérativement se connecter, en ouvrant l'application et en choisissant un nom d'utilisateur, et choisir son destinataire qui se trouvera dans une liste de contact dont il aura accès dès sa connexion. Discuter avec un destinataire permettra la création d'un historique dont l'utilisateur aura accès.

B. Diagramme de classe & Vue de l'architecture

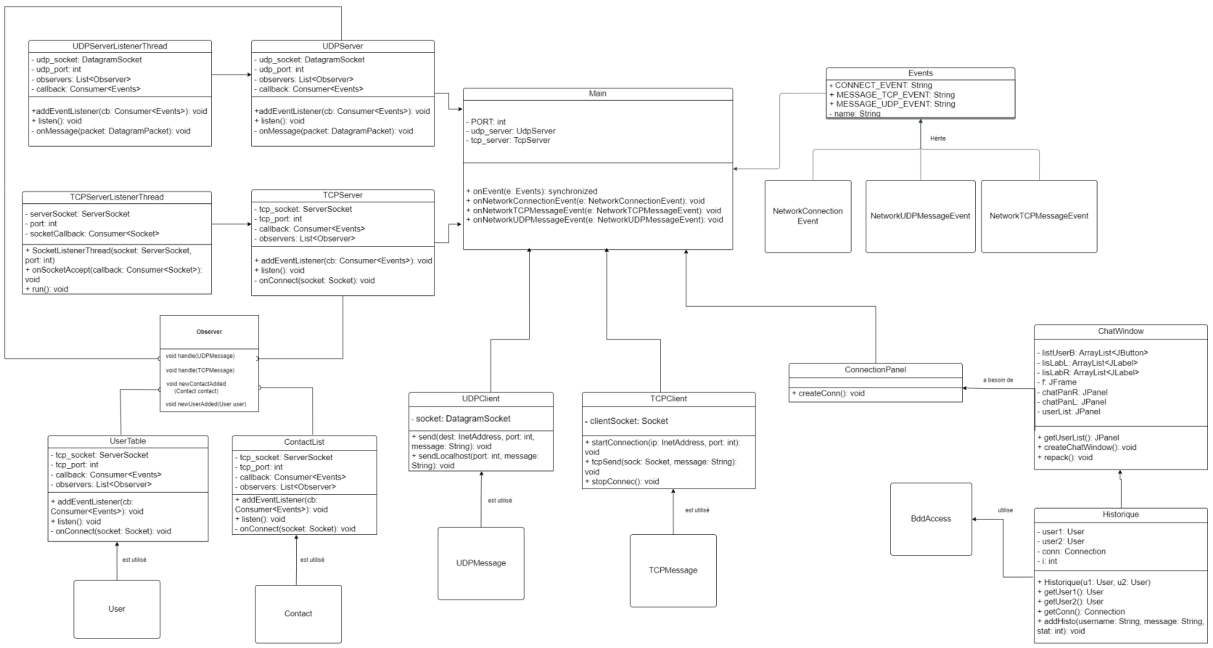


Diagramme de classe du projet ChatSystem

L'architecture du projet Chatsystem se compose comme suit : la partie supérieure centrale représente la classe Main, c'est-à-dire celle qui va lancer l'application et lier ses fonctionnalités qui sont disposées autour.

- A gauche du Main se trouve les classes des serveurs UDP et TCP et leurs threads respectifs, qui vont permettre la réception des messages en continu et les acceptations de connexions entrantes.

- Toujours à gauche du main, en dessous des serveurs, se trouvent les classes ContactList et UserTable, qui vont permettre le stockage des utilisateurs connectés

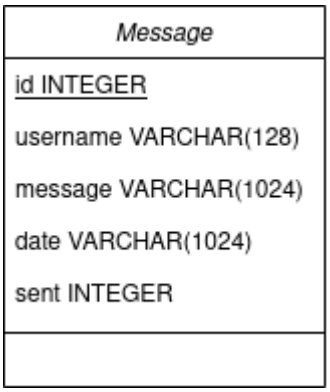
utilisant l'application. On remarque que les serveurs et ces classes sont lié à une interface qui leur est nécessaire, c'est-à-dire l'interface Observer, qui va monitorer les changements d'état d'une ou plusieurs composantes des classes.

- En bas du Main se trouvent les classes UDPClient et TCPClient qui vont permettre l'envoi de messages de type éponyme, en utilisant respectivement les classes de données UDPMessage et TCP Message

- A droite du Main se trouve la classe mère Events et ses classes filles représentant les différents types d'évènements qui peuvent survenir lors de la réception de messages, et la manière dont le programme Main va réagir.

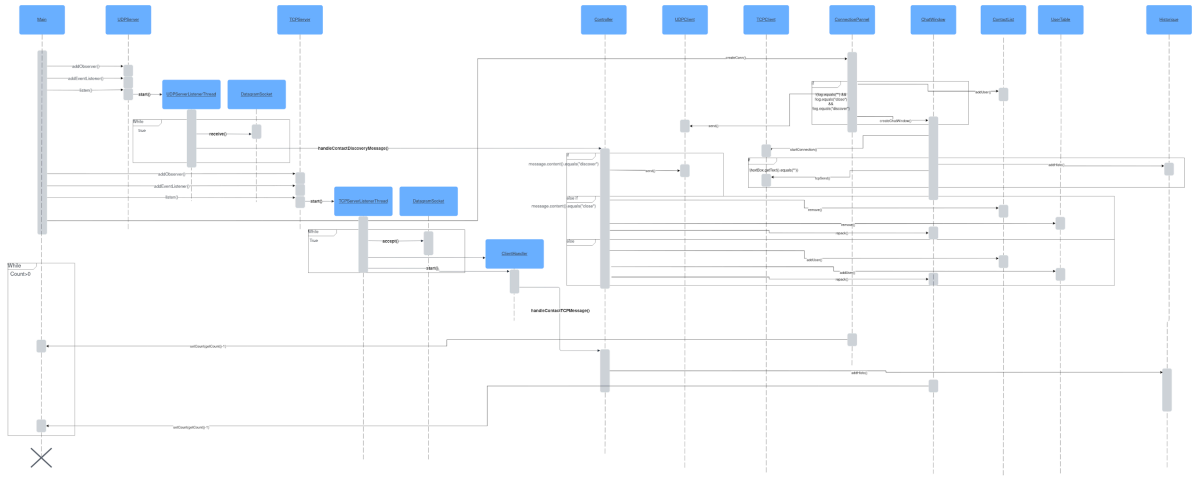
- Toujours à droite du Main, en dessous des Events se trouve la partie Interface du projet avec les classes GUI ConnectionPanel et ChatWindow, qui vont offrir à l'utilisateur les fonctionnalités de l'application sous forme d'interface graphique. On y trouve également les classes Historique et BddAccess pour la gestion de l'historique des messages reçus et envoyés.

C. Schéma de la base de données



On a utilisé une petite base de données en SQLite pour générer l'historique des messages entre chaque utilisateur. Elle contient donc une seule template de base, et plusieurs en seront créées à chaque discussion entre utilisateurs.
Nous avons préféré limiter le nombre d'éléments dans la base de données afin d'avoir le moins d'accès à la base de données possible dans le programme, ne connaissant pas la complexité de cet accès.

D. Diagramme de séquences



Le Main lance les serveurs TCP et UDP avec les méthodes leur permettant d'écouter en lançant chacun un thread d'écoute. Lorsque ces threads détectent des messages UDP ou TCP ils appellent les méthodes du Controller correspondantes prévues. Le thread d'écoute TCP lance lui-même un nouveau thread à chaque connexion tcp, c'est ce thread qui gère l'arrivée des messages TCP. Le controller agis différemment selon qu'il ait reçu un message de discovery, de déconnexion ou un message contenant le pseudo d'un autre utilisateur.
Le Main lance également un message de discovery en broadcast et crée la fenêtre de connexion. Dans cette fenêtre, lorsque le bouton connexion est appuyé, on vérifie le contenu du champ de texte. S'il est vide ou qu'il contient un pseudo non autorisé, c'est-à-dire un pseudo déjà utilisé ou réservé aux échanges d'information comme le discovery ou la déconnexion, on affiche un message d'erreur, sinon, on enregistre le pseudo dans la liste des utilisateurs, on envoie un broadcast de ce pseudo afin d'informer les autres utilisateurs et on ouvre la fenêtre de chat.
Dans cette fenêtre, cliquer sur les boutons correspondant aux utilisateurs permet d'ouvrir une connexion TCP avec eux pour leur envoyer des messages tout en affichant en temps réel l'historique des messages avec eux. Le message est envoyé lorsque l'on clique sur le bouton "envoyer" et que le champ n'est pas vide. Une autre connexion TCP temporaire s'ouvre alors et se referme dès que le message est envoyé.
Lorsque l'on envoie ou reçoit un message en TCP, son contenu est ajouté à la BDD grâce à l'Historique et sa méthode addHisto(), on indique alors le pseudo de l'utilisateur correspondant, le message en lui-même et si le message a été reçu ou envoyé.
Après avoir ouvert la fenêtre de ConnectionPannel, le Main se bloque dans un while, attendant que son attribut Cont descende à 0. Il est initialisé à 1 et est incrémenté de 1 à chaque fenêtre ouverte et décrémenté de 1 quand on les ferme. Lorsque l'on ferme la fenêtre de ChatWindow, on envoie un message de déconnexion en broadcast prévenant les

autres utilisateurs. Après être sorti du while, le Main exécute un System.exit(0) et le programme se ferme.

II. PDLA

Pour mener à bien le projet ChatSystem, nous avons adopté un principe de conduite de projet, et également des concepts tels que l'intégration continue et le développement continu.

A. Motivation des choix et application des méthodes

Afin de mettre en place la méthode agile, nous aurions pu utiliser un outil prévu à cet effigie tel que Jira. Cependant, comme nous avions en plus des diagrammes UML à mettre en place tels que le diagramme de scénario et également le fait que nous ne soyons que deux pour ce projet, nous avons décider de se baser principalement sur des esquisses de diagramme de scénarios, faisant office de "User stories" si l'on veut choisir un équivalent dans la méthode agile, et les "sprints" ont été effectués à l'oral et au fur et à mesure au vu du nombre de séances qui ont été prévus pour réaliser le projet ChatSystem. Ensuite, nous avons appliqué une méthode d'intégration continue et automatisée en les outils Maven, Github actions et Junit permettant la compilation du projet et la vérification automatique des tests à chaque push sur le git.

Conclusion

Les diagrammes UML nous ont permis d'avoir une vue globale du projet et ont été mis à jour au fur et à mesure de l'avancement. Ils nous ont servi également de support pour la mise en place d'une méthode agile dans le cadre du processus de conduite de projet. Nous avions omis l'utilisation de JIra car nous avions déjà commencé avec les diagrammes, cependant il est indéniable que l'utilisation d'un tel outil facilite la mise en place des méthodes de conduite de projet.