

Rendu Projet Java

**AKKAR ZAIBEL Anouar
ALAMI EL MEJJATI Amine**

4IR B2

Sommaire

Introduction	3
I. Conception	4
a. Use Case	4
b. Diagrammes de séquence	4
c. Diagrammes de classe	4
II. Architecture du projet	5
a. Packages	5
b. Broadcast	5
c. Échange TCP	5
III. Manuel Utilisateur	6
a. Installation	6
b. Page de connexion	6
c. Page principale(Generale)	6
IV. Choix d'implémentation	7
a. Base de données	7
b. Swing	7
c. Maven	7
V. Test réalisés	8
BDD	8
TCP	8
UDP	8
VI. Gestion de projet	9
a. Jira	9
b. Github	9

Introduction

En tant qu'étudiants en 4ème année en informatique et réseau à l'INSA de Toulouse, nous avons été chargés de créer un système de clavardage distribué multi-utilisateur en temps réel dans le cadre de notre formation. L'objectif de ce projet était de développer une application de clavardage fonctionnelle qui respecte un cahier des charges établissant un ensemble de fonctionnalités.

Au cours du développement de cette application, nous avons suivi les étapes classiques de la conception et de l'implémentation.

Ce rapport décrit notre projet, notre processus de conception, nos choix d'implémentation ainsi qu'un manuel d'utilisation pour faciliter la prise en main de notre projet Clavardage.

I. Conception

a. Use Case

Afin de comprendre les exigences et les besoins du projet, nous avons utilisé un diagramme de cas d'utilisation. Ce diagramme a permis une première analyse des demandes du client en se basant sur les spécifications énoncées dans le cahier des charges. Il nous a aidé à identifier les différentes fonctionnalités nécessaires pour répondre aux besoins de notre projet de système de clavardage distribué multi-utilisateur en temps réel.

b. Diagrammes de séquence

Les diagrammes de séquence que nous avons créés ont permis de représenter de manière graphique les interactions et les échanges entre les différents composants du système au cours du temps. Ils ont facilité la compréhension du fonctionnement des différentes phases rencontrées en visualisant les flux d'échanges entre les différents éléments du système. Ainsi, ils ont été d'une grande utilité pour comprendre la logique de fonctionnement de notre projet de système de clavardage distribué multi-utilisateur en temps réel.

c. Diagrammes de classe

Pour notifier notre interface des différents changements nous avons utilisé des éléments de type observable et une interface ConnectionListener ainsi nos contrôleur (WelcomeControler , mainwindowControler) sont notifiés des différents changements que ce soit sur la liste des utilisateurs ou celle des messages. D'autre part nos contrôleurs communiquent aussi avec la base données tout cela pour afficher les échanges de messages , la connexion de nouveaux utilisateurs, déconnexion ...

II. Architecture du projet

a. Packages

Nous avons séparé le projet en 4 parties Connexion(UDP),Clavardage(TCP),DataBase et Interface mise à part l'interface les autres packages doivent être totalement indépendants et fonctionner correctement.

b. Broadcast

Cette fonctionnalité est implémentée dans le package Connexion et on y trouve 6 classes :

Ecoute : cette classe contient une InetAddress (celle de la personne qui envoie le message) , un buffer (ou est stocké le message envoyé par les autres utilisateurs) , un objet de la classe Connexion . Cette classe lance un thread qui écoute en permanence sur un port prédéfinie et attends un message udp il réagit en fonction du message soit un pseudo et ainsi se fait la vérification si cet id n'est pas présent dans sa liste il l'ajoute et la notifie de cette ajout , si le pseudo est déjà utilisée elle envoie un message pour notifier de ceci . D'autre part cette classe permet aussi au personne souhaitant se connecter de savoir si le nom d'utilisateur est déjà utilisé ou d'ajouter les personnes déjà connectées à sa liste.

ConnexionListener : cette interface sert à notifier la classe welcomeController de l'acceptation ou le rejet d'un nom d'utilisateur elle contient les méthodes isValide et invalid .

RemoteUser : cette class représente un utilisateur quelconque celui ci a un String pour pseudo et une InetAddress . Cette classe contient un constructeur et les différents guetteurs nécessaires .

UDP : Cette classe a un Socket et un port prédéfini . Elle a une seule méthode , broadcast qui prends un String (le message à envoyer) et une address ip (celle à qui on veut envoyer le message en udp)

UserList : cette classe représente une liste de personnes connectées, elle contient une ObservableList (nous avons choisi ce type pour pouvoir notifier la classe MainWindowController et ainsi afficher les différents utilisateurs connectés) . Cette classe contient les méthodes delUser , addUser , getUserByID ...

Connexion : Cette classe contient un String ,le pseudo .Elle contient les méthodes verifyID qui envoie le pseudo donnée en entrée en broadcast à toutes les personnes connectés , il contient aussi la méthode change pseudo .

c. Échange TCP

Dans ce package on retrouve 8 classes :

ClientTCP : cette classe contient un socket, un PrintWriter et un BufferedReader ainsi que les méthodes startConnexion permettant de lancer une connexion vers un serveur, getConnexion permettant d'instancier une connexion à partir d'un socket, sendMessage, rcvMessage qui permettent l'envoie et la réception de String, finalement stopConnexion.

ServeurTCP : cette classe crée un ServerSocket qui est en accept en boucle, après un accept() il lance un thread ClientHandler qui récupère le socket et crée un ClientTCP et se met en écoute.

StartSession : cette classe permet de se connecter à un serveur en créant un ClientTCP et se met en écoute.

Message : C'est un type qui contient (String userName, String Texte, Date Date), et les getter.

Les 4 classes ListOfClients, ListOfHandlers, ListOfMessages et ListOfSessions sont des listes qui contiennent des ClientTCP, ClientHandler, Messages et StartSessions respectivement ainsi que des getters.

III. Manuel Utilisateur

a. Page de connexion

Cette page contient un champ où l'utilisateur peut entrer son pseudo. Ensuite en cliquant sur le bouton connexion, la vérification du nom d'utilisateur se fait. Si celui-ci est déjà pris, un message s'affiche pour demander à l'utilisateur de réessayer avec un nouveau pseudo. Si le nom d'utilisateur n'est pas pris alors une nouvelle page s'affiche, la fenêtre principale. Pour éviter des problèmes avec la création des tables dans la base de données nous vérifions aussi que le pseudo ne contient pas de caractères spéciaux et ne commence pas par un chiffre.

b. Page principale(Generale)

Cette page contient la liste des utilisateurs connectés si on veut communiquer avec une personne il faut qu'elle soit déjà connecté et donc qu'elle soit présente dans la liste il suffit de cliquer sur la personne dans la liste . Tous les anciens messages sont chargés si on veut envoyer un message il suffit de le rentrer dans la zone texte réservée et d'appuyer sur le bouton Send . Il y a aussi un bouton déconnexion pour se déconnecter et si on veut changer de pseudo il suffit d'entrer le nouveau pseudo dans la zone réservée et d'appuyer sur le bouton changerPseudo .

IV. Choix d'implémentation

a. Base de données

Afin d'enregistrer les messages et pouvoir les récupérer lors d'une autre connexion il faut une base de données. Nous avons choisi de faire une BDD locale, ce projet étant décentralisé il nous paraissait plus logique de faire ainsi, de plus cela faciliterait un peu l'implémentation de la BDD. Nous avons choisi d'utiliser SQLite car il y a énormément d'informations disponibles sur Internet sur SQLite. Afin d'utiliser SQLite dans notre projet il faut ajouter une dépendance dans le pom.xml .

Nous avons choisi de mettre trois champs dans la table (pseudo,message,date) étant les trois des clés primaires, autrement on ne pourrait pas insérer des messages avec le même pseudo ou des pseudo avec les mêmes messages. Pour nommer les tables nous avons décidé d'utiliser le pseudo et l'adresse ip de l'autre personne (pseudo127_0_0_0).

b. Java fx

Nous avons opté pour javafx car c'est une bibliothèque populaire pour la création d'interfaces graphiques pour les applications Java, elle offre une grande flexibilité et des fonctionnalités avancées. Elle permet de créer des interfaces graphiques multiplateformes qui s'affichent de manière identique sur différents systèmes d'exploitation. En utilisant SceneBuilder en combinaison avec JavaFX, vous pouvez faciliter la mise en place des différentes classes de la vue de l'application en utilisant une approche visuelle pour concevoir les scènes graphiques.

V. Test réalisés

BDD

Afin de pouvoir tester le bon fonctionnement des méthodes nous avons fait un test JUnit qui crée une nouvelle base de données test, par la suite on crée une table test, on insère un messages de test pour finalement faire un select et récupérer le contenu de la table et le comparer avec assertEquals au message qu'on a insérer. Ceci permet de vérifier le bon fonctionnement de toutes les méthodes utilisées dans le projet.

TCP

Dans le clienthandler du serveur lorsqu'il reçoit "Hello dude" il renvoie "Hi mate" cela permet de faciliter le test. Je crée un Serveurtcp et plusieurs clientTCP qui vont se connecter sur ce serveur et qui vont envoyer "Hello dude" , puis attendent un message de la part du serveur, finalement je compare ce message avec "Hi mate" en utilisant assertEquals.

Après j'envoie "end1" avec chaque client en attendant à chaque fois que la liste de clients contient le bon nombre de clients.

Ce test permet donc de vérifier que plusieurs clients peuvent se connecter au serveur et que les clients peuvent bien envoyer des messages et recevoir des messages ainsi que le serveur. ça permet aussi de vérifier que la déconnexion marche correctement.

UDP

Pour commencer nous avons testé si le message envoyé est le même que le message reçu de la même manière que décrit précédemment . Nous avons également testé la vérification du pseudo et l'ajout des utilisateurs dans la liste en lançant l'application sur 2 machines et plus . En ce qui concerne les méthodes relatives à la liste comme ajouter , enlever des utilisateurs , récupérer les pseudos ... toutes ces fonctionnalités ont été testées grâce à des tests unitaire comme vous pourrez le voir sur le git.

VI. Gestion de projet

a. Jira

Jira étant un outil de gestion de projet, son utilisation nous a permis d'organiser correctement le projet. Tout d'abord nous avons divisé notre projet en 4 grandes parties : UDP,TCP,BDD,Interface. Nous avons par la suite créé différentes tâches (sprints) plus précises (Pouvoir se connecter via un socket client vers un socket serveur, pouvoir envoyer des messages depuis le client vers le serveur, vice-versa etc.). En utilisant Jira , nous avons également pu suivre les progrès de chaque sprint et identifier les éventuels problèmes en temps réel, ce qui nous a permis d'ajuster notre plan d'actions en conséquence. Enfin, Jira nous a permis de maintenir une communication efficace entre les différents membres de l'équipe en permettant de partager des commentaires, des fichiers et des mises à jour en temps réel, ce qui a considérablement amélioré la qualité de notre projet.

b. Github

Github est un outil qui permet de gérer efficacement le code source de leurs projets. Il offre une grande variété de fonctionnalités telles que la gestion des versions, la collaboration en temps réel, la gestion des bugs et des demandes de fonctionnalités. En utilisant GitHub, on peut travailler sur des projets en équipe et partager facilement notre code avec les membres de l'équipe. Malheureusement nous avons eu un problème et nos push étaient sur des répertoires différents, par conséquent nos commits ne sont pas dans le même répertoire. Finalement afin de vérifier automatiquement que notre projet fonctionne correctement nous avons mis en place sur Jenkins un trigger qui permet de lancer la compilation du projet ainsi que les tests Junit lorsqu'un push est fait dans le répertoire.



INSA Toulouse

135, avenue de Rangueil
31077 Toulouse Cedex 4 - France
www.insa-toulouse.fr



MINISTÈRE
DE L'ÉDUCATION NATIONALE,
DE L'ENSEIGNEMENT SUPÉRIEUR
ET DE LA RECHERCHE