GUESMI Mohamed Amine
RAZI Yassine

Rapport Final – Application de Chat Réseau (TCP)

Lien Github : https://github.com/GuesmiMedAmine/ProjetChatReseaux.git

1. Introduction

Ce projet consiste à développer une application de chat réseau en Java qui permet à plusieurs clients de communiquer entre eux via un serveur central. L'objectif principal est de mettre en œuvre les concepts de programmation réseau en utilisant les sockets TCP. Le serveur reçoit les connexions entrantes et gère chaque client au moyen d'un thread dédié, permettant ainsi la diffusion (broadcast) des messages entre tous les clients connectés.

2. Objectifs et Contexte

- **Objectifs pédagogiques :**
  - Apprendre à créer et gérer des connexions TCP en Java.
  - Mettre en place une architecture serveur/clients avec gestion multi-thread.
  - Implémenter un système de broadcast pour diffuser les messages de chaque client à tous les autres.
- **Contexte du projet :** Ce projet a été réalisé dans le cadre d'un TP réseaux et doit répondre à des exigences simples de communication entre plusieurs clients en utilisant une structure de répertoires bien organisée. La solution s'appuie sur des classes simples, adaptées à un contexte étudiant.

3. Architecture de l'Application

Le projet est organisé en deux parties principales :

3.1 Côté Serveur

La partie serveur est composée de trois classes regroupées dans le package **server** :

- **ServerApp.java :**
  Point d'entrée du serveur. Cette classe contient la méthode main() qui instancie et démarre le serveur sur un port défini (par exemple, 1234).

- **Server.java :**
  Cette classe gère la logique principale du serveur. Elle crée un objet ServerSocket et entre dans une boucle infinie pour accepter les connexions des clients. Pour chaque connexion établie, elle crée un objet ClientHandler (un thread dédié) afin de gérer la communication avec ce client. Une liste synchronisée permet de conserver la trace de tous les clients connectés pour diffuser les messages.

- **ClientHandler.java :**
  Pour chaque client connecté, cette classe (qui étend Thread) gère la lecture des messages envoyés par le client. Elle demande d'abord au client de s'identifier en envoyant son nom, puis diffuse les messages reçus à tous les autres clients via la méthode broadcast() du serveur. Lorsqu'un client se déconnecte (en envoyant la commande quit), le thread se termine et le client est retiré de la liste.

3.2 Côté Client

La partie client se compose de deux classes dans le package **client** :

- **ClientApp.java :**
  Point d'entrée du client. La méthode main() de cette classe crée une instance de la classe Client en spécifiant l'adresse et le port du serveur (ici, "localhost" et 1234) et démarre la communication.

- **Client.java :**
  Cette classe gère la connexion au serveur à l'aide d'un objet Socket. Elle met en place deux flux : un pour envoyer les messages (via un PrintWriter) et un pour les recevoir (via un BufferedReader). Pour éviter le blocage lors de la réception, un thread distinct est lancé afin de lire en continu les messages diffusés par le serveur. L'utilisateur peut alors saisir ses messages dans la console. Lorsque le client envoie la commande quit, il se déconnecte proprement du serveur.

4. Déroulement de la Communication

1. **Démarrage du Serveur :**
   - Le serveur démarre et se met en écoute sur le port 1234 à l'aide d'un ServerSocket.
   - Il affiche un message indiquant qu'il est opérationnel.
2. **Connexion des Clients :**
   - Lorsqu'un client se connecte, le serveur accepte la connexion et crée un nouveau thread (ClientHandler) pour gérer cette connexion.

- Le client reçoit d'abord une demande d'identification et envoie son nom.
- Le serveur diffuse ensuite un message d'accueil à tous les clients pour annoncer l'arrivée du nouveau participant.

3. **Échange de Messages :**
    - Chaque client peut envoyer des messages en tapant dans la console.
    - Le thread de lecture côté client affiche les messages diffusés par le serveur.
    - Le serveur, via le ClientHandler, lit chaque message et le transmet à l'ensemble des clients connectés (broadcast).
4. **Déconnexion :**
    - Si un client envoie le message quit, il reçoit un message d'adieu et se déconnecte.
    - Le serveur retire ce client de la liste des clients connectés et informe les autres participants de sa déconnexion.
    -

5. Conclusion

Ce projet de chat réseau en TCP permet de comprendre les mécanismes fondamentaux de la programmation réseau en Java. La répartition en modules clairs (côté serveur et côté client) facilite la compréhension et la maintenance du code.
Les points clés abordés sont :

- La gestion des connexions par un ServerSocket.
- L'utilisation de threads pour gérer simultanément plusieurs clients.
- La diffusion des messages via une méthode broadcast pour assurer la communication entre tous les participants.

Ce projet constitue une base solide pour étendre les fonctionnalités (par exemple, gestion des erreurs, interface graphique, etc.) et approfondir la compréhension des communications réseau.