Architecture client

L'architecture est simple car le client est en bloquant.
La classe ClientTCP contient le client avec la gestion de la ligne de commande et utilise les classes Readers et Writters du package clientProtocol pour communiquer avec le serveur.

La classe Readers permet de lire les paquets envoyé par le serveur. On a décidé de déconnecté le client si le serveur envoyé un mauvais paquet.

La classe Writters permet d'envoyer des demandes au serveur en respectant le format de la RFC.

Architecture serveur

La classe DataPacketRead permet de stocker les données lors d'une lecture d'une demande d'un client.

L'interface Reader permet d'implémenter plusieurs type de readers pour lire tous les formats combiner avec nos readers.

On a donc des classes ReadersInt, ReaderString et ReaderLong qui lise le format donné et remplisse l'objet DataPacketRead.
Il y a plusieurs états de lecture pour gérer le tcp en non bloquant.
Des énums sont utilisé pour représenter les différents types de packets et les états du serveur.

Fonctionnalité :

Toutes les fonctionnalités de base marchent, envoyer des messages, connexion privé, message privé, envoie de fichiers…

Pour les cas particuliers, il est possible que nous n'avons pas tout détaillé car il y en a énormément.

Exemples de cas particulier gérés :

-Toutes les déconnexions clients ou serveur, déconnexion privé…Une déconnexion d'un client ne dérange pas le serveur ni celle d'un autre client qui été connecter à lui. Le client pourra continuer de discuter sur le chat.
-Mauvaise trames comme une acceptation de connexion privé alors que le client n'avait fais aucune demande.
-Absence de réponse d'un client pour une connexion privé.
-Pseudo vide ou avec espace car cela cause des problèmes pour faire des invitations.
-Demande de connexion ou acceptation de connexion quand on a déjà atteint le maximum de connexion
-Unicité des pseudos géré par le serveur.

-Gestion de l'usurpation d'identité, car il est nécessaire d'envoyer un long unique généré par le serveur et donnée au client lors de sa connexion.
-Envoi de fichier inexistant
-Connexion privé à soi même refusé.
-Connexion privé à une personne qui n'existe pas.
-Nombre de connexion privé limité, pour fixer le nombre de Thread.
-Demande de connexion privé à quelqu'un déjà connecté.
-Acceptation/refus à une personne alors qu'on est déjà connecté à elle.
-Message privé non bloqué par envoi de fichier et envoi de fichier possible à plusieurs client en même temps. Impossible d'envoyer plusieurs fichiers à un même client par contre.
-Réception d'un fichier qui a le même nom qu'un fichier qu'on a déjà. Le fichier en question n'est pas supprimé ou remplacé, un autre fichier avec un entier devant est crée à la place.

Exemples de cas non gérer :
-Taille pseudo, message.
-Client inactif sur le serveur. Ils ne sont pas déconnecté.

Difficultés rencontrés :

La plus grosse difficulté rencontré a été la gestion des cas particuliers pour obtenir un serveur et un client robuste sans restreindre les fonctionnalités du chat.
La seconde difficulté majeure a été la gestion des états du serveur et l'implémentation en non bloquant en général.

Nous avons mis dans ce répertoire nos automates qui nous ont aidé à gérer ces états.