Vincent AGULLO - Jonathan CRÉTÉ

Projet ChatHack : Client/Serveur

Détails d'implémentations

Dans ce document nous allons détailler ce qui a été implémenté dans le projet ChatHack.

1. Détails d'implémentations

Notre programme implémente le design-pattern des visiteurs. Nous avons une interface ***Frame*** décrivant ce que doit implémenter une trame en l'occurrence les données de cette trame sous forme de byteBuffer ainsi qu'un ***frameVisitor*** pour indiquer l'entité qui appelle cette trame (*le client ou le serveur*).
Du point de vue implémentation chaque trame que nous avons réalisé implémente l'interface ***Frame***. Pour chaque trame nous avons une classe reader implémentant l'interface Reader (i*nterface permettant lire les données d'une trame*) qui lui est associé. Lors de la lecture d'un paquet, la classe ***FrameReader*** regarde l'opérande du paquet et appelle et lit la bonne trame en appelant le reader de la trame associé.

2. Client

Pour démarrer un client nous mettons le login et le mot de passe (s'il y en a) en paramètre de commande. Dès le démarrage d'un client, une trame de connexion est envoyé directement au serveur, afin de vérifier l'identité du client. Si la connexion du client est approuvé par le serveur, alors un message lui sera envoyé lui signifiant de quel type de connexion il s'agit (*connexion identifié via mot de passe, ou connexion anonyme*). Dans le cas où le client s'identifie avec un login ou un mot de passe erroné ou si le client en connexion anonyme utilise un login déjà enregistré dans la base de donnée alors le client verra sa connexion se fermée.

Lorsqu'un client envoie un message public à tous les autres clients connectés le serveur reçoit la trame envoyé par le client et le ***FrameReader*** lit l'opérande 14. Le ***ServerFrameVisitor*** appelle la méthode visit() et appelle la méthode ***broadcast***() pour diffuser le message. Pour toutes les clés enregistrés auprès du sélecteur, nous récupérons le contexte de chaque clients sur lequel nous devons envoyer le message. Le ***ClientFrameVisitor*** appelle lui aussi la méthode visit() mais lit le message avec la classe ***AuthentificationMessageReader.***

Lors de la demande d'une connexion privée par un client, une trame ***RequestPrivateConnection*** est envoyé au serveur. Cette trame est ensuite envoyé par le serveur au client destinataire. Le client destinataire reçoit cette trame et a le choix de soit accepter la demande ou soit de la refuser. Si le client destinataire refuse la connexion, une trame ***RefusePrivateConnexion*** est renvoyé au serveur qui va le transmettre à l'expéditeur de la demande. Dans le cas où le client destinataire accepte la demande. A la différence le client envoie une trame ***AcceptPrivateConnexion*** passant par le serveur pour envoyer une ***Socket Address*** (*adresse IP port*) ainsi qu'une clé unique pour se connecter. Pour l'envoi du premier message, le client ChatHack maintient une **table de hachage** contenant les pseudonymes des clients en attente de validation de connexion Les valeurs de la hashmap sont les messages (pour ne pas perdre le 1er message)

Un contexte spécifique (***ClientPrivateContext***) de manière à envoyer un message d'un client à un autre. Les messages transites par se nouveau contexte.
Nous avons commencé le développement de la fonctionnalité d'échange de fichier client-client (*qui n'est possible uniquement si une connexion privée est établie*) en codant un ***reader FileReader***.

3. Serveur

Nous avons implémenter la fonctionnalité d'authentification. Dès que le client va s'authentifier, on va vérifier son identité auprès de la base de données.Un **contexte spécifique DBContext** pour la base de donnée est alors attaché au sélecteur. Le client va alors envoyer une trame, le serveur en recevant cette trame va, en fonction du type d'authentification (*avec ou sans mot de passe*) envoyer une trame spécifique à la base de données. Si le client a envoyé une demande d'authentification avec un pseudonyme et un mot de passe, on va envoyer une trame à la base de données simulée pour vérifier que le pseudonyme et le mot de passe concordent. S'ils concordent, le serveur renverra un message au client par le biais de la méthode ***sendToClientResponseOfDB()*** pour lui signaler qu'il est à présent connecté. Dans le cas où celui-ci ne possède pas de mot de passe, l'utilisateur est alors connecté de manière anonyme. Si l'utilisateur s'authentifie de manière anonyme avec un pseudonyme déjà existant dans la base de donnée, le client ne sera pas connecté.

Aussi, nous avons développé la fonctionnalité de demande de connexion privée. Nous avons choisi d'implémenter le type nommé ***RequestPrivateConnection*** donnant lieu au reader

RequestPrivateConnectionReader représent permettant la lecture de la requête de connexion privée. Dès qu'un client en fait la demande, une trame est envoyée par celui-ci, elle transite par le serveur et est envoyé au client destinataire. Le rôle du serveur est alors, à partir de la trame de connexion privée, d'identifier à quel client le message est destiné, avec la méthode ***getLoginFromId(),*** et d'envoyer le message.