POLYTECHNIQUE
MONTRÉAL

INF3405
Réseaux informatiques
Groupe 1

Travail pratique 1

Communication Serveur-Client

Soumis à : Kadi, Mehdi

Soumis par :
Boukaftane, Hamza - 2183376
El Harami, Mehdi - 2113402
Benzekri Omar 2244082

26 mai 2023

Dans un contexte marqué par les préoccupations la protection de la vie privée, il est devenu essentiel de développer des solutions de communication sécurisées. Ainsi, nous avons décidé de créer une application de clavardage Java sécuritaire respectant la vie privée des utilisateurs. Nous avons développé un système de communication basé sur l'architecture Serveur-Client permettant l'échange de messages entre usagers. Le serveur doit être en mesure d'authentifier ou de créer des comptes utilisateurs, de recevoir et de transmettre des messages et de maintenir un historique des messages. Le client doit être en mesure de se connecter, de quitter ou de créer un compte et d'envoyer ou de recevoir les messages. Enfin, l'objectif de ce travail est de se familiariser avec les concepts d'échanges entre des clients et un serveur, d'interfaces de connexion (sockets) et de fils d'exécution (thread). Dans ce rapport, il sera question d'expliquer le système produit.

Dans un premier temps, nous avons décomposé le système en trois différents projets Java dans l'environnement de développement intégré Eclipse : serveur, inputValidation et client. Pour ce qui est du projet inputValidation, il est constitué d'une classe contenant deux méthodes statiques permettant de valider le format d'une adresse IP et le numéro de port. Elle contient également un attribut privé de type Pattern correspondant au format des adresses IP accepté. La méthode validant l'adresse IP prend une chaîne de caractères en argument et la compare à l'attribut privée à l'aide d'un objet de type Matcher. L'autre méthode prend en argument un nombre (int), vérifie sa correspondance à l'intervalle 5000 à 5050. Dans les deux cas, les méthodes retournent un booléen représentant le résultat de la comparaison. Enfin, la classe InputValidation permet d'encapsuler les méthodes de validation afin de les utiliser dans les autres projets de l'application.

Par ailleurs, le projet serveur contient les classes Server, pour l'établissement d'un chat room particulier caractérisé par une adresse IP et un port, et ClientHandler, le fil d'exécution de la gestion par le serveur d'un client en particulier. En ce qui concerne la classe Server, elle encapsule les attributs suivants : le numéro de port, l'adresse IP, les noms des fichiers textes contenant les informations de connexion des utilisateurs et le l'historique complet des messages, un objet ServerSocket afin d'écouter les connexions des clients et différents Concurrent HashMap pour conserver un registre des clients connectés en temps réel, un registre des nom d'utilisateur et de mot de passe et un registre contenant un maximum des 15 derniers messages. L'assignation et le remplissage des attributs sont effectuée lors de la configuration du serveur. L'utilisation des Concurrents HashMap est justifié plus amplement à la dernière page du rapport. La classe contient une méthode main qui est le point d'entrée principale de l'application serveur. Lors de son appel, une instance de Serveur est générée et le processus de configuration est entamé. Ainsi, l'application serveur demande à l'utilisateur de saisir une adresse IP et numéro de port valides. Puis, un objet ServerSocket est instancié et lié à l'adresse IP et le numéro de port saisie. Par la suite, les base de données sous forme de fichiers textes (.txt) sont créés dans le répertoire du projet serveur s'il n'existe pas. Sinon, les base de données existantes sont lues et les HashMap des informations de connexion d'utilisateurs et des messages les plus récents sont chargés avec les données lues. De

plus, l'application serveur affiche également à cette étape les 15 messages les plus récents sur la console s'il y a lieu. La gestion des fichiers se font à l'aide des objets Files, Path, Paths, FileWriter, FileReader, BufferedWriter et BufferedReader de Java. Enfin, une fois le serveur configuré, le programme rentre dans une boucle infinie à l'intérieur de laquelle sont instanciées les différents fils d'exécution de gestion de client particulier lorsque l'objet ServerSocket identifie une tentative de connexion d'un client. Le constructeur de ClientHandler est alors appelé et l'instance Serveur lui passe en référence un socket disponible, les noms des fichiers de base de données, les Concurrent HashMap contenant les messages les plus récents, la liste des utilisateurs connectés et la liste des informations de connexion de tous les utilisateurs. Après l'instanciation, le fil d'exécution est immédiatement démarré.

La classe ClientHandler encapsule le fil d'exécution de la gestion des différents clients et différents attributs aidant à la gestion du client (notamment les références aux attributs de l'instance serveur lors de la construction). En effet, chaque instance initialise, à partir de l'interface de connexion (Socket), deux canaux de communication entrant (DataInputStream) et sortant (DataOutputStream) uniques avec le client permettant d'envoyer et de recevoir des données. Dans un premier temps, le fil d'exécution valide les informations de connexion de l'utilisateur. On attend le nom d'utilisateur et le mot de passe de l'utilisateur provenant du client. Si, l'utilisateur n'est pas dans la liste des utilisateurs connues, alors un nouveau compte est créé et les informations sont ajoutées à la base de données et à la liste des utilisateurs connues. Sinon, dans le cas d'un mauvais mot de passe, l'instance refuse la connexion du client et elle attend la réception des bonnes informations de connexion. Dans le cas d'un bon mot de passe, l'instance accepte la connexion. Une fois la connexion ou la création du client validé, l'instance envoie un message de confirmation de connexion et les messages les plus récent au client. À cette étape du programme, le fil d'exécution entame une boucle dépendant de la valeur booléenne de l'activité du client qui est toujours assigné à vrai après instanciation du ClientHandler. Dans cette boucle, on attend les messages entrants. Les messages sont diffusés à l'ensemble des clients connectées au serveur et affichés sur la console de l'application serveur. Si le message provient d'un client, il est également sauvegardé dans la base de données de message et ajouté à la liste des messages les plus récents. Enfin, s'il y a rupture du canal de communication avec le client, la valeur d'activité du client devient fausse ce qui nous permet de sortir de la boucle et un message est diffusé à tous les clients connectés et affichée sur la console.

Finalement, le projet client contient la classe Client qui permet aux usagers de choisir un serveur de communication et d'échanger avec les autres utilisateurs. Elle encapsule les attributs suivants : le numéro de port, l'adresse IP, le nom d'utilisateur, le mot de passe, une interface de connexion (Socket) et les canaux de communication. L'assignation et le remplissage des attributs sont effectuée lors de la configuration du client. La classe contient une méthode main qui est le point d'entrée principale de l'application client. En effet, lorsque que cette dernière est appelée, une instance de client est générée et le processus de configuration est entamé. Ainsi, l'application client

demande à l'utilisateur de saisir une adresse IP et numéro de port valides. Puis, le processus de validation du client commence. L'utilisateur saisie son nom d'usager et son mot de passe. Le client instancie alors l'interface de connexion (Socket) lié à l'adresse IP et le numéro de port saisie et les canaux de communication entrants et sortants. Puis, les informations d'authentification qui ont été saisie par l'utilisateur sont envoyées au serveur pour validation. Si la réponse du serveur est négative, le processus d'authentification recommence. Sinon, l'instance client reçoit les messages les plus récents, les affichent sur la console et le client rejoint le chat room. À cette étape, un nouveau fil d'exécution permettant de recevoir et d'afficher les messages est initialisée en parallèle au programme qui entame une boucle dépendant de la valeur booléenne de l'activité du client qui est toujours assigné à vrai après instanciation du Client. Dans la boucle, l'utilisateur peut écrire des messages afin de les transmettre au server pour diffusion. Après la saisie, le message est traité. Si ce dernier est « quit », l'instance client sort de la boucle d'envoi de message, arrête le fil d'exécution parallèle de réception de message et l'application client est arrêter. Sinon, l'instance formatte le message en ajoutant une en-tête au message contenant le nom du client, l'adresse IP, le port local du client, la date et le temps d'envoie. La date et le temps sont ajouté à l'aide des objets LocalDateTime et DateTimeFormatter. Elle s'assure également que le message ne dépasse pas 200 caractères sans quoi elle coupe le message à la taille voulue. En ce qui concerne le fil d'exécution parallèle de réception, il est initialisé grâce à la classe ClientMessageReceiver contenant une référence au canal entrant de l'instance client. Dans une boucle du même style que celle du client, l'instance de ClientMessageReceiver reçoit les messages destinés à l'instance client et elle les affiche sur la console client.

La synchronisation des données entre une instance Serveur et les différentes instances de ClientHandler a été un défi rencontrer. En effet, comme l'enregistrement des utilisateurs connectées et des nouveaux messages se font aux niveaux de l'instance ClientHandler qui ne communique pas entre elle, la synchronisation des données causent un souci. C'est ici qu'interviennent les Concurrent HashMap. En effet, comme toutes les Concurrent HashMap des instances ClientHandler sont construit à partir d'une référence à un Concurrent HashMap de l'instance Server, tout changements sur un des HashMap se répercute sur l'ensemble des Concurrent HashMap. Cela permet la synchronisation des données entre les différentes instances. De plus, un autre problème rencontré était la nature bloquante de la méthode scanner.nexLine() pour l'envoie de message des instances clients. En effet, tant que l'utilisateur n'a pas compléter la saisie, le code est mis en attente et l'instance client ne peut recevoir de message. Donc, certains messages sont perdus. La solution à ce problème est l'initialisation d'un fil d'exécution parallèle gérant la réception de message et l'affichage de ceux-ci sur la console client.

Pour conclure, ce travail pratique nous a permis d'explorer différentes structures de données de Java, de nous familiariser avec les concepts de serveur et de clients, de mieux comprendre le fonctionnement des fils d'exécution et de produire des exécutables. Nous sommes vraiment satisfaits de ce travail pratique, car le fonctionnement et la théorie derrière ces concepts est extrêmement pertinent dans le cadre de notre profession.