Protocoles des Services Internet – TP

Annonce Manager

(TCP Porotocol using SSL for security)

Réalisé par :
Hamza OUADHDHAFE
Hajar TAJMOUATI

Dans le groupe :
Ke Lyu et Chuanhao Zhang
Ghada Chtioui

Introduction

Ce document définira l'architecture et la structure du protocole d'une application d'annonce.

Cette application simple permet aux clients de s’adresser à un serveur pour envoyer des annonces, les ajouter, les modifier et même les supprimer, ces mêmes annonces seront publiées ultérieurement si tel est le souhait du client. Le travail du serveur est de renvoyer à tous les clients toutes les annonces qui ont été enregistrées et répondre bien sûr à toutes les autres commandes concernant la gestion d’un compte aussi.

Il est important de considérer que l'annonce publiée par le serveur sera « mise à jour », ce qui signifie que les annonces provenant de clients hors ligne et obsolètes doivent être discriminatoires.

1- Architecture adoptée

Dans le but de modéliser cette application, l'application démarre par une communication Client/Serveur, lorsque chaque endpoint (client) est initialisé, il se connecte à un serveur (adresse IP et port connus). Ce serveur gère la quantité de clients connectés et leurs annonces.

Ce serveur représente la centrale qui stocke, met à jour et redistribue les annonces (diffusées à tous les points de terminaison). Chaque fois qu'un client souhaite publier un message, supprimer un message ou se déconnecter de l'application, il communiquera directement avec le serveur.

Type de communication : Client / Server

Type de Socket: TCP
Server IP: localhost
Server Port: 1027

Pour notre solution nous avons opté pour le protocole TCP qui est basé sur la communication Client/serveur d'égale à égale entre les applications, communication réalisée par dialogue entre deux processus.

En ce qui concerne la version sécurisée, le but était de rendre la communication « sure » en veillant à ce qu'aucune tierce personne puisse s'immiscer entre les utilisateurs, pour ça nous avons opté pour SSL (Secure socket layer) qui est un complément à TCP/IP et qui permet de sécuriser n'importe quel protocole ou programme utilisant TCP/IP, ce dernier assure trois points essentiels voulus :

- **Confidentialité :** il est impossible d'espionner les informations échangées.
- **Intégrité :** il est impossible de truquer les informations échangées.
- **Authentification :** il permet de s'assurer de l'identité du Client avec lequel on communique.

- **Structure du projet**

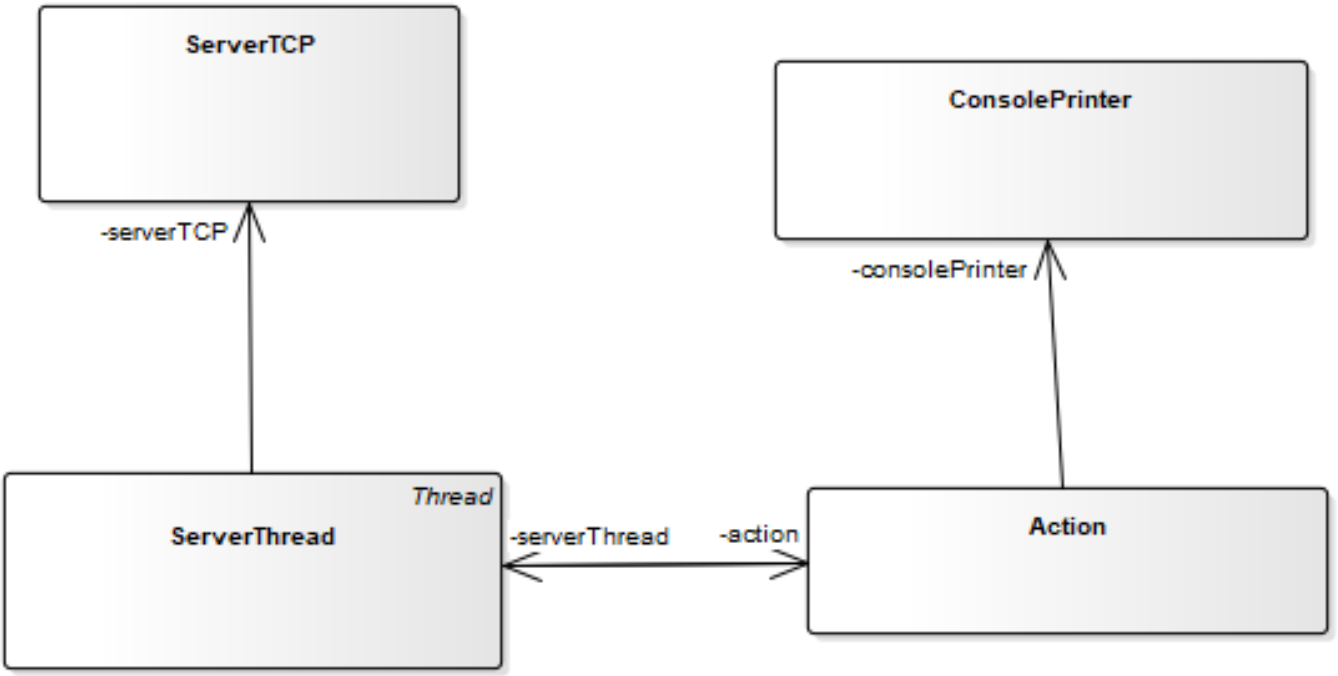

Diagramme de Classes de notre Serveur

Détails de la structure du projet

- **ServerTCP :** Cette classe nous créé un socket qui sert à construire un serveur, dans un premier temps le serveur est en mode écoute en attendant une demande de connexion des clients.
- **ServerThread :** Cette classe permet de gérer les requêtes des clients en parallèle en donnant des réponses pour chacune des requêtes.
- **Action :** Permet de traiter la requête demander et l'exécute.
- **ConsolePrinter :** Génère la réponse envoyée au client esthétiquement parlant.

Cette architecture nous permet de séparer les responsabilités et chaque objet fait son rôle. Dans un premier temps quand le serveur reçoit les requêtes des clients **ServerThread** envoi à l'objet **Action** pour traiter la commande et exécute la commande, par la suite **ConsolePrinter** génère la réponse et l'envoie à **ServerThread** qui s'occupe de répondre le client concerné.

2- Description du protocole :

1ére étape : Le client se connecte avec le serveur

2éme étape : après le handshake, le client entre son identifiant pour se connecter avec la commande (ex. « connect user1 :user1 »)

3éme étape : si l'identifiant du client est correct il peut consulter, modifier, supprimer et ajouter les annonces.

Premier point très essentiel, parce que les réponses reçues par le serveur contiennent un séparateur '|' qui sépare les lignes, nous avons envisagés d'implémenter un mécanisme utilisé pour délimiter les messages en remplaçant celui-ci avec '**\n** ' (retour à la ligne) avec :

```
response.replace(("|", System.getProperty("line.separator")));
```

Tous les échanges de messages entre les clients et le serveur seront envoyés en format String construit par notre objet **ConsolePrinter**.

La récupération des réponses du serveur est possible grâce à l'utilisation de l'objet **BufferedReader** avec :

```
BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
```

Pour lancer l'application Serveur/Client

Le lancement du programme peut se faire de 2 façons :

Ouvrir un terminal, se positionner dans le projet et exécuter la commande choisie suivant la façon choisie.

1ère façon : Si vous voulez compiler via le serveur, voici la commande :

>> make

A présent le serveur est à l'écoute, vous devriez normalement recevoir une réponse qui ressemblerait à ça :

Server start listening on port (1027)

Connection (1) established with client from /127.0.0.1: 52413

2ème façon : Si vous voulez utiliser les commandes java :

>> javac *.java

>> java ServerTCP

Maintenant pour lancer le client, on ouvre un autre terminal et on se met dans le projet toujours, puis on écrit les commandes suivantes :

>> javac ClientTCP.java

>> java ClientTCP

Vous devriez normalement avoir comme résultat ceci :

Voulez-vous vous connecter ? la ligne de commande est :

```
>> connect <username>:<password>
```

Voulez-vous créer un compte ? la ligne de commande est :

```
>> create user <username>:<password>:<phone>:<email>
```

Vous pouvez à présent entrer les commandes souhaitées détaillées par la suite.

Attention : dans ces 2 cas et après que vous lancez le Serveur, vous lancez notre Client avec les commandes :

>> javac ClientTCP.java

>> java ClientTCP

Le client communique avec le serveur grâce aux différentes commandes envoyées par celui-ci, nous allons décrire en détail ce qui se passe lors de cette communication pour mieux comprendre.

Supposons que le client s'est connecté au serveur, il envoie une commande quelconque au serveur, on choisit :

>> annonce <price>:<description>:<domain>

Comme exemple pour ajouter une annonce

Du côté du serveur la commande est reçue exactement de la sorte, elle est envoyée à l'objet « **Action** » pour être traitée. Le traitement est rapide et conçue de façon optimiale car ça teste si la commande est correcte de base et suit la norme de la commande qui devrait être écrite, si non le serveur renvoie un message d'erreur, si oui la requête est tout de suite envoyée à l'objet « **Action** » qui fait tout le travail du traitement et prend le soin de retourner une réponse correcte à la demande, ce travail consiste à récupérer le string pour déjà tester de quel type de commande ça s'agit et **spliter** chaque partie de la requête dans un tableau de string puis renvoyer le traitement à l'objet « ConsolePrinter » qui se charge de fabriquer le String qui va être affiché au final au client.

1. Guide pour créer votre client

Pour que votre Client puisse se connecter au Serveur il faut suivre les étapes suivantes :

1. Créer la socket sécurisée en utilisant le Store (fichier) client.jsk.

```
// for security we are using SSL
System.setProperty("javax.net.ssl.trustStore", "client.jsk");
System.setProperty("javax.net.ssl.trustStorePassword", "123456");
SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();

// sans oublier l'adresse IP du serveur et le port
SSLSocket socket = (SSLSocket) socketFactory.createSocket("localhost", 1027);
socket.setEnabledCipherSuites(socketFactory.getDefaultCipherSuites());
```

2. Pour la récupération des réponses du serveur il faut utiliser l'objet BufferedReader.

```
BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(socket.getInputStream()));
```

3. Lors de récupération de la réponse.

```
while (!request.equalsIgnoreCase("bye")) {
        response = bufferedReader.readLine();

        // parceque les reponse reçu de serveur contient un séparateur '|' qui sépare les
        lignes on remplace le séparateur avec « \n » ou System.getProperty("line.separator")
        reponse = response.replace(("|", System.getProperty("line.separator")));
        System.out.print(response);
        request = entryStream.readLine();
        printStream.println(request);
}
```

2. Protocole utilisé

a. *Pour lancer l'application Serveur/Client :*

+ java **ServerTCP.java**
+ java **ClientTCP.java** *(Vous pouvez lancer un ensemble de clients)*

Remarque 2 : *A propos de la solution sécurisée nous avons utilisé SSL en se basant sur les "KEYSTORE"*

***Prérequis** :*

Il faut générer 3 fichiers avant de pouvoir tester ce projet, un fichier .jsk pour le serveur, et un fichier .jsk pour le client généré é partir du certificat donné par le serveur (.crt)

Pour générer le serveur.jsk :

```
keytool -genkey -keystore server.jsk -alias server keyalg RSA
```

Pour générer le serveur.crt :

```
keytool -export -keystore server.jsk -alias server -file server.crt
```

Pour générer le client.crt :

```
keytool -import -alias server -file server.crt -keystore client.jsk
```

Les mots de passe demandé sont "123456"

PROTOCOLES ADOPTES PAR LES AUTRES MEMBRES DU GROUPE ET VERDICT APRES D'EVENTUELS TESTS :

Protocoles des autres membres du groupe :

- Avec **Ke Lyu et Chuanhao Zhang :**
  - Notre communication Client/Serveur dans les deux sens est bien fait, nous avons adapté presque le même protocole avec TCP et SSL pour la sécurité, avec un petit changement dans les lignes de commande.
- Avec **Ghada Chtioui**
  - Nos applications ne peuvent être communiqué en raison des problèmes suivants :
    - Nous avons utilisé **TCP** et elle a utilisé **UDP**.
    - Son serveur envoie des objets or notre Client a besoin d'une chaîne de caractères pour fonctionner.

b. *Gestion de commandes en détail :*

Dans un premier temps quand le client est connecté avec le serveur, ce dernier envoie au nouveau client un menu qui contient deux commandes

+ Pour créer un compte => create user <username>:<password>:<phone>:<email>

+ Pour se connecter => connect <username>:<password>

*username, password, phone, email sont choisi par le client et enregistré sur la plate-forme du serveur.

Si le client dispose déjà d'un compte il va exécuter la deuxième commande, sinon il peut créer son propre compte grâce à la première commande.

- Les autres lignes de commandes :

+ Pour ajouter une annonce => annonce <price>:<description>:<domain>
+ Pour lister vos annonces => myannonce
+ Pour lister toutes les annonces => annonce
+ Pour supprimer une annonce => delete <ID-annonce>

*ID-annonce : ID spécial donné à l'annonce par le client lui-même.

Attention !! : *dans les lignes de commandes si vous ajoutez le séparateur ":" dans les propriétés, le serveur envoie une erreur "BAD REQUEST" au client.*
Exemple : Dans la <description> si vous ajoutez ':' dans cette dernière rien ne va se passer à part l'affichage d'un message d'erreur.

c. Explications de chaque commande avec exemple :

Pour créer un compte => create user <username>:<password>:<phone>:<email>
```
>> create user test:test1:0615205151:test@gmail.com
```
+ La requête est envoyée au serveur, le serveur traite la commande, si elle est bonne alors l'utilisateur est ajouté dans la Base de données sinon le serveur envoie une erreur au client

---

Pour se connecter => connect <username>:<password>
```
>> connect test:test1
```
+ La requête est envoyée au serveur, le serveur traite la commande, si elle est bonne l'utilisateur est connecté à l'application et le serveur envoie la liste des annonces au client, sinon il envoie le menu d'authentification au client.

Pour ajouter une annonce => annonce <price>:<description>:<domain>
```
>> annonce 50:IphoneX:Téléphone
```
+ La requête est envoyée au serveur, le serveur traite la commande, si elle est bonne alors l'annonce est ajoutée dans la base données ensuite le serveur renvoie la liste des annonces et le menu au client.

Pour lister vos annonces => myannonce
```
>> myannonce
```
+ La requête est envoyée au serveur, le serveur traite la commande, si elle est bonne alors la liste des annonces du client connecté s'affiche.

Pour lister toutes les annonces => annonce
+ La requête est envoyée au serveur, le serveur traite la commande, si elle est bonne alors la liste de toutes les annonces est affichée.

Pour supprimer une de vos annonces => delete <ID-annonce>

```
>> delete 1
```

+ La requête est envoyée au serveur, le serveur traite la commande, si l’annonce choisie appartient au client ou existe alors elle va être supprimée sinon le serveur va envoyer une erreur.