Projet UDP / TCP

INFO0503 - Modélisation client-serveur et programmation Web avancée

Juliete Halal Rossie

Antinéa Pintau-Loiseaux

Intervenant : Cyril Rabat

Introduction :

Dans le cadre de la matière INFO0503, il nous a été demandé de réaliser une application de tracking sportif. Cette application permet au client d'envoyer périodiquement les positions GPS au serveur qui les stockera. Les client doit saisir des données manuellement.

Travail réalisé :

Implémentation 1 :

Pour l'implémentation 1 l'application est en mode client / serveur à l'aide de sockets en mode non :connecté. On a donc implémenté l'application en UDP mais il fallait faire quelques changement au fonctionnement d'UDP pour avoir une application en mode client / serveur.

Tout d'abord UDP est un mode non connecté, mais le mode dans le mode client / serveur le client devra pouvoir ce connecter, pour cela on a choisit de rajouter une attribut privée HashMasp au serveur. Dès qu'un client envoie une message de connexion (message id 0) le serveur utilise le gestionnaire d'utilisateurs pour vérifier l'identité du client, et si le client à un compte, le serveur enregistre son adresse et son pseudo dans la map. Donc la map contient HashMap<InetAdress, String>. Si le serveur reçoie un message d'un autre type (par exemple message id 2 pour commencer une activité), avant d'accepter le message, le serveur vérifie que l'utilisateur est connecté en recherchant son adresse dans la map.

Deuxièmement dans le mode client / serveur le serveur devra toujours répondre au client. Pour faire cela il est possible de créer un DatagramPacket dans le serveur, avec l'adresse du client en reprenant l'adresse et le port du message (DatagramPacket) du client avec la méthode get. Ceci permet d'établir la connexion et serveur / client et le client pourra choisir ces prochaines actions selon le message du serveur.

Implémentation 2 :

Dans l'implémentation 2, on a recréer la même application mais en mode communication par messages, à l'aide de sockets en mode connecté. Ainsi on a implémenté notre application en TCP, mais le serveur n'envoie pas toujours des accusés de réception au client. Dans notre implémentation, le seul message envoyé du serveur au client est celui pour indiquer si le pseudo et mot de passe sont corrects.

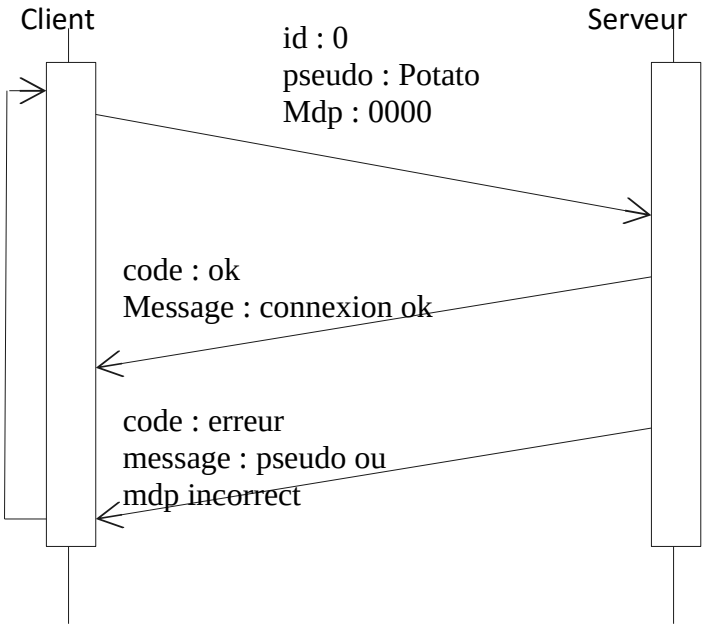
En mode communication par message, le client n'est pas forcement connecté mais pour l'application on a besoin de savoir le pseudo de l'utilisateur, c'est pour cela qu'on a choisit de mettre l'attribut user qui nous permet d'enregistrer le pseudo de l'utilisateur, et d'utiliser ce pseudo lors de l'enregistrement d'une activité.

L'implémentation 2 a été faite en mode multi-thread, pour permettre à plusieurs utilisateurs de se connecter au même temps.

Chronogrammes d'échanges :

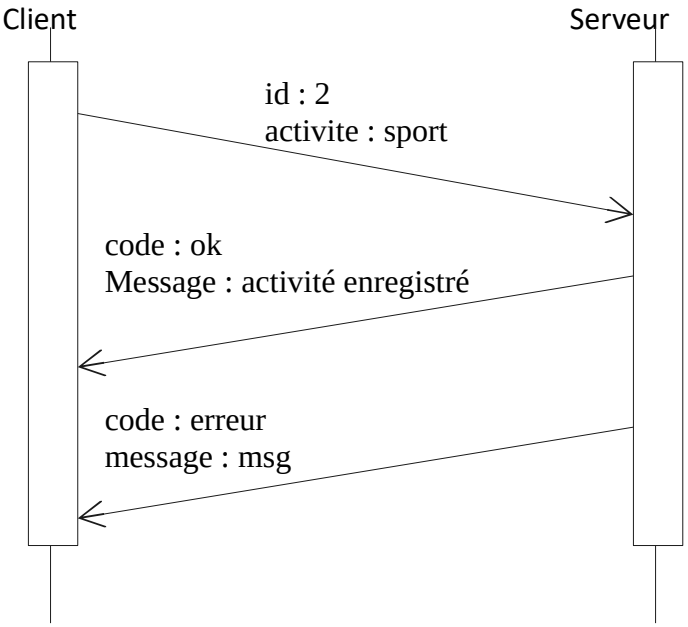
Connexion :

Pour la connexion, le chronogramme est le même pour les deux implémentations.



Commencer une activité :

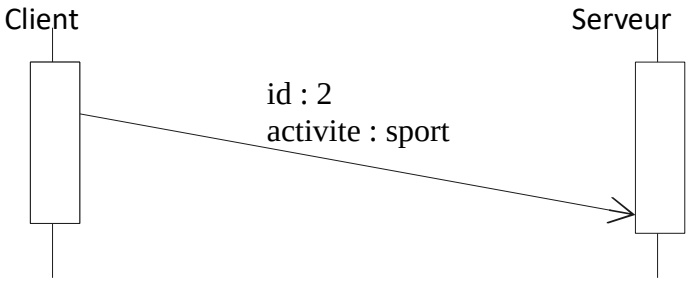
Implémentation 1 :



Erreurs possible :

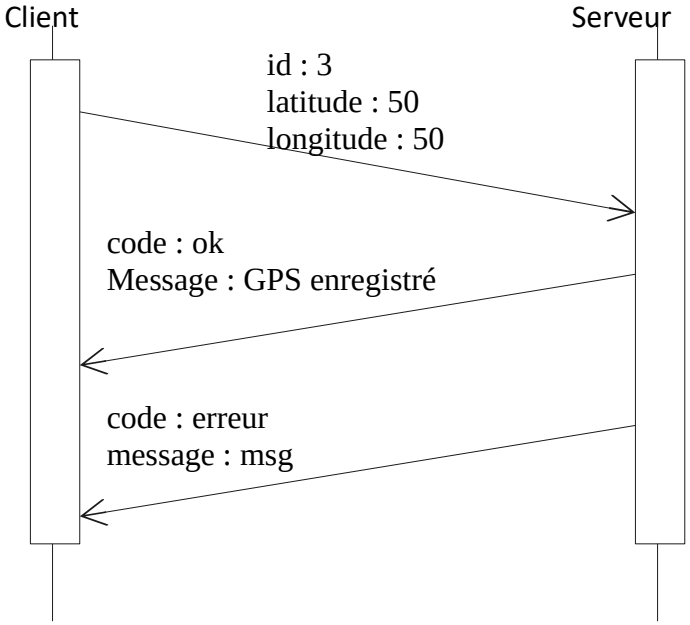
- Client non connecté
- Erreur lors de la création de l'activité

Implémentation 2 :



Entrer GPS :

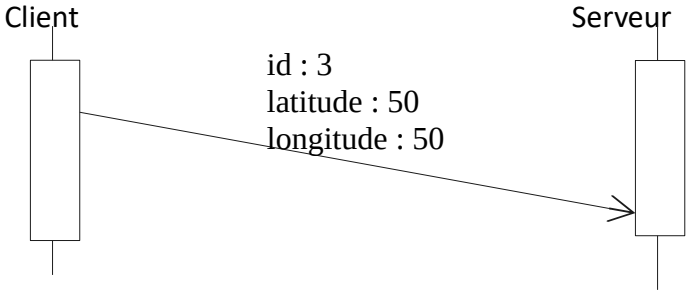
Implémentation 1 :



Erreurs possible :

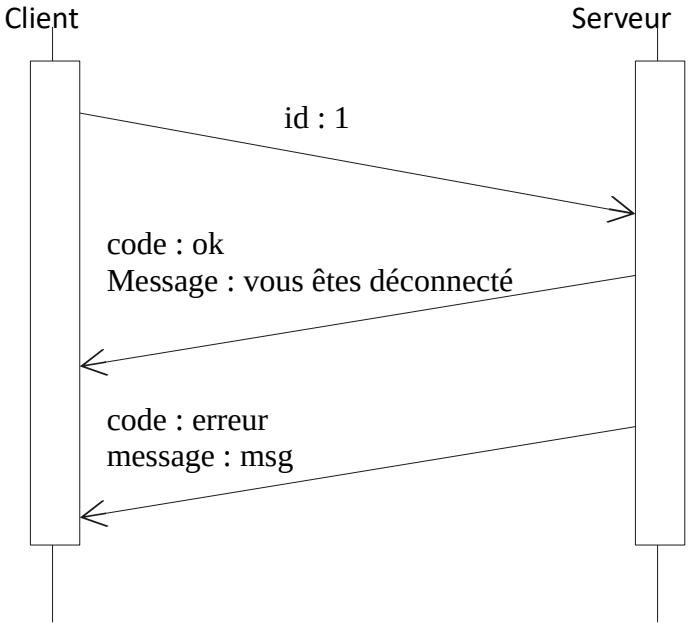
- Client non connecté
- Erreur lors de la création du GPS

Implémentation 2 :



Déconnexion :

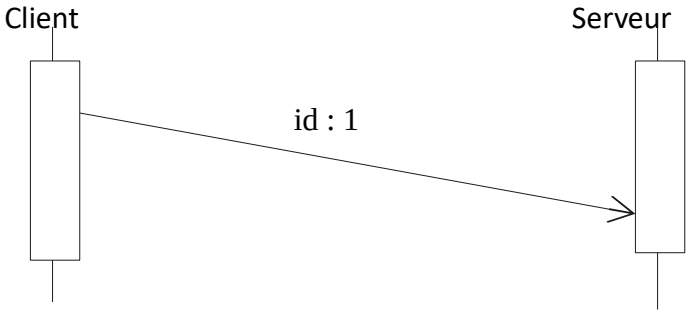
Implémentation 1 :



Erreurs possible :

- Client non connecté
- Erreur lors de la déconnexion

Implémentation 2 :



JSON :

Les messages :

Pour la communication entre client et serveur on utilise des message sous la forme JSON. Les messages envoyé par le client ont toujours un id, ainsi le serveur sait quel est l'intérêt de ce message, dans nos deux implémentations les id des messages sont :

- 0 → connexion
- 1 → déconnexion
- 2 → commencer une activité
- 3 → entrer ses coordonnées

Chacun des messages contient des données différentes par exemple le message avec un id 0 contient également un pseudo et un mot de passe, le message avec id 2 contient également le nom de l'activité, et le message avec id 3 contient également les coordonnées.

Les messages reçus par le client contiennent toujours un code « OK » si l'opération s'est déroulé sans erreur, et « erreur » sinon. Le message contient également un message envoyé par le serveur, ainsi en cas d'erreur on peut afficher le message du serveur au client.

Fichiers JSON :

Il y a plusieurs catégories de fichiers JSON stockés. Tout d'abord les fichiers dans le dossier config, qui permettent d'avoir les ports d'écoute du serveur et du client.

```
{
   "port": 1056,
   "adresse": "127.0.0.1"
}
```

On a également un fichier JSON contenant tout les utilisateur ayant un compte sur le serveur dans le dossier users.

```
{
    "users": [
        {
            "pseudo": "Potato",
            "mdp": "0000"
        },
       ...
]}
```

Finalement le dossier activite contient tous les activités stockés sur le serveur. Pour enregistrer les fichiers on utilise le pseudo de l'utilisateur avec le nom de l'activité qu'il a choisit.

```
{
   "heure_f": "19:56:49.025",
   "liste_GPS": [{
      "latitude": 2,
      "longitude": 3
   }],
   "heure_d": "19:56:43.988",
   "sportif": "Potato",
   "sport": "act"
}
```

Exécution :

Implémentation 1 : dans tp10/impl1

1. javac -cp .;../json/json.jar *.java
2. java -cp .;../json/json.jar Serveur
3. Dans un autre terminal : java -cp .;../json/json.jar Client

Implémentation 2 : dans tp10/impl2

1. javac -cp .;../json/json.jar *.java
2. java -cp .;../json/json.jar Serveur
3. Dans un autre terminal : java -cp .;../json/json.jar Client