Petites Annonces

Protocoles des Services Internet

Guillaume Roumage Natacha Gillaizeau-Simonian

2020

Sommaire

1 But du projet

On souhaite réaliser une application de petites annonces pour la vente entre particuliers. On réalise tout d'abord une implémentation permettant la création, consultation, mise à jour et suppression d'annonces en ligne, présentée dans ce rapport. Dans un second temps, nous implémenterons une communication pair à pair entre clients.

Un utilisateur peut proposer des objets à la vente par le biais d'une annonce visible par les autres utilisateurs. Une annonce possède un titre, un domaine (type voiture, téléphone...), un descriptif et un prix.

En plus de poster des annonces, un utilisateur peut mettre à jour ou supprimer une de ses propres annonces uniquement.

Chaque utilisateur peut aussi consulter les annonces correspondant à un domaine.

2 Protocole

Le diagramme du protocole est représenté ci-dessous. Le détails de chaque requête et réponse du serveur, notamment le nombre et le type d'arguments transmis, sont détaillé en sous-section 2.1. .

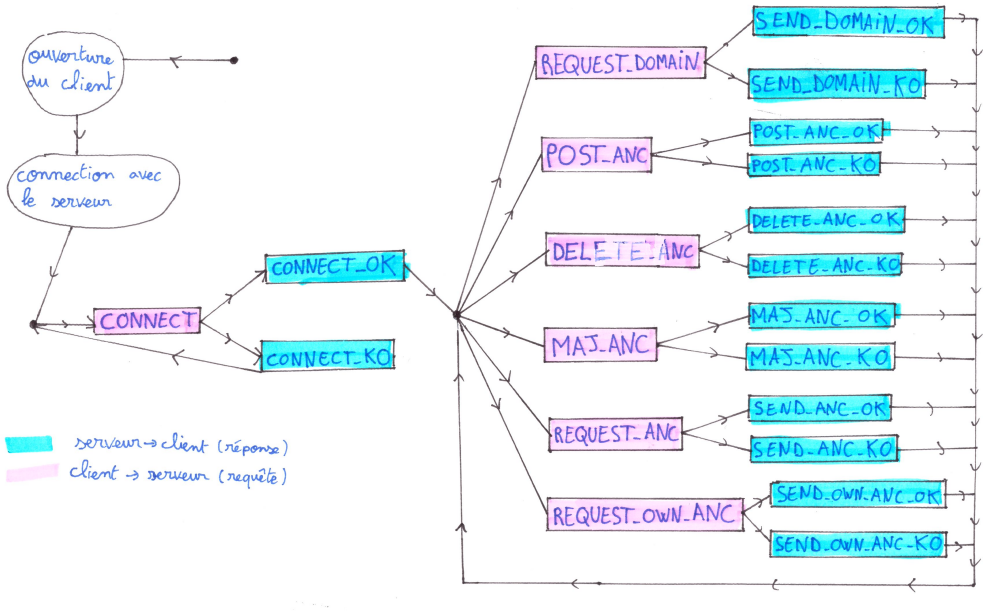

Figure 1: Schéma protocole

Chaque requête client → serveur ou serveur → client est de la forme :

```
<EN_TÊTE>
<arg_1>
...
<arg_n>
.
```

On peut noter le fait que chaque requête se termine par un point, signalant la fin de transmission des arguments.

2.1 Connexion

La connexion au serveur se fait par le client avec la requête suivante :

```
CONNECT
<utilisateur> ou <token>
.
```

Si l'utilisateur s'est déjà connecté au serveur par le passé, il peut se reconnecter avec son token.

Si c'est sa première connexion, il entre un nom d'utilisateur et le serveur va lui renvoyer un token pour ses connexions futures.

En cas de réussite, le serveur renvoie :

```
CONNECT_OK
.
```

Et sinon :

```
CONNECT_KO
.
```

2.2 Domaines existants

Le client peut demander au serveur quels sont les domaines existants avec la requête suivante :

```
REQUEST_DOMAIN
.
```

En cas de réussite, le serveur renvoie :

```
SEND_DOMAIN_OK
<domaine 1>
<domaine 2>
[...]
.
```

Et en cas d'échec :

```
SEND_DOMAIN_KO
.
```

2.3 Poster une annonce

Un client peut poster une annonce avec la requête suivante :

```
POST_ANC
<domaine>
<titre>
<descriptif>
<prix>
.
```

En cas de réussite, le serveur renvoie :

```
POST_ANC_OK
<id_annonce>
.
```

Sinon

```
POST_ANC_KO
.
```

2.4 Mise à jour d'annonce

Le client peut mettre à jour une de ses annonces avec la requête suivante :

```
MAJ_ANC
<id_annonce>
<domaine>
<titre>
<descriptif>
<prix>
.
```

Si l'un des champs (id_annonce exclus) est égale à "null" alors l'ancien champs sera conservé.
En cas de réussite, le serveur renvoie :

```
MAJ_ANC_OK
<id_annonce>
.
```

Sinon

```
MAJ_ANC_KO
.
```

2.5 Suppression d'annonce

Le client peut supprimer une de ses annonces avec la requête suivante :

```
DELETE_ANC
<id_annonce>
.
```

En cas de réussite, le serveur renvoie :

```
DELETE_ANC_OK
<id_annonce>
.
```

Sinon

```
DELETE_ANC_KO
.
```

2.6 Voir les annonces

Les clients peuvent voir les annonces correspondants à un certain domaine avec la requête suivante :

```
REQUEST_ANC
<domaine>
.
```

En cas de réussite, le serveur renvoie :

```
SEND_ANC_OK
<id_annonce1>
<domaine1>
<titre1>
<prix1>
<id_annonce2>
[...]
.
```

Sinon :

```
SEND_ANC_KO
.
```

2.7 Voir ses propres annonces

Un client peut voir ses propres annonces avec la requête suivante :

```
RESQUEST_OWN_ANC
.
```

En cas de réussite, le serveur renvoie :

```
SEND_OWN_ANC_OK
<id_annonce1>
<domaine1>
<titre1>
<prix1>
<id_annonce2>
[...]
.
```

Sinon :

```
SEND_OWN_ANC_KO
.
```

3 Choix d'architecture et d'implémentation

Pour ce projet, nous avons utilisé l'architecture client-serveur. Le principe est d'avoir un serveur au sein duquel est centralisé la quasi-totalité des données et du traitement des informations nécessaires au bon fonctionnement de l'application.

Pour ce projet, nous avons un serveur d'actif à la fois, d'où le choix de passer la majorité des éléments concernant le serveur en `static`.

Plusieurs clients peuvent se connecter simultanément (par la création de thread de `ClientHandler` comme nous le verrons en sous-section 3.1) au serveur pour faires des requêtes (demande des domaines d'annonces disponible, publication d'une annonce...). Le serveur attend les requêtes et, en réponse à ces requêtes, il envoie les informations demandées au client (la liste des domaines, la confirmation ou non de la création d'une annonce ...).

La classe `Domain` représente les domaines présents sur le serveur. Il n'y a pas de possibilité d'ajout ou de suppression de domaines lorsque le serveur est en cours de fonctionnement. Cela pourra faire l'objet d'une amélioration.

La classe `Annonce` permet de simplement représenter une annonce avec tous les attributs qui lui sont associés.

3.1 La classe `ClientHandler`

Le `ClientHandler` a pour fonction de gérer la connexion entre un client et le serveur. C'est un thread qui est lancé à chaque demande de connexion sur le serveur par un client. De cette manière, le serveur peut gérer plusieurs clients simultanément (chaque thread de `ClientHandler` s'occupera alors d'un client). Chaque client connecté sur le serveur interagira donc avec le serveur via son `ClientHandler`. Le `ClientHandler` traite les requêtes du client et envoie au client la réponse adéquate, selon les traitements faits par le serveur.

3.2 La classe `ClientServer`

Le `ClientServer` gère notamment l'ajout ou la suppression des annonces. Cette classe permet au serveur de garder une trace de chaque client passé sur le serveur, ainsi que les annonces qu'ils ont créées, même si ces clients se déconnectent. C'est la représentation d'un client du point de vue du serveur.

3.3 La classe `Client`

La classe `Client` représente un client. Elle permet d'interagir avec le serveur (en envoyant des requêtes) et d'interpréter les réponses du serveur (les réponses aux requêtes). Dans cette première partie du projet, elle est constituée d'une part de toutes les informations nécessaires à l'identification (nom d'utilisateur, token) et à la connexion (socket, input et ouput stream ...) du client, et d'autre part de deux threads. Un thread `outputCmd` dont le rôle est d'envoyer les requêtes du client au serveur. En parallèle, le thread `inputCmd` a pour rôle de recevoir les réponses du serveur.

Pour implémenter les connexion pair à pair, une piste est de créer deux threads supplémentaires : le premier se chargera de l'envoie des messages, tandis que le second se chargera de la réception des messages.

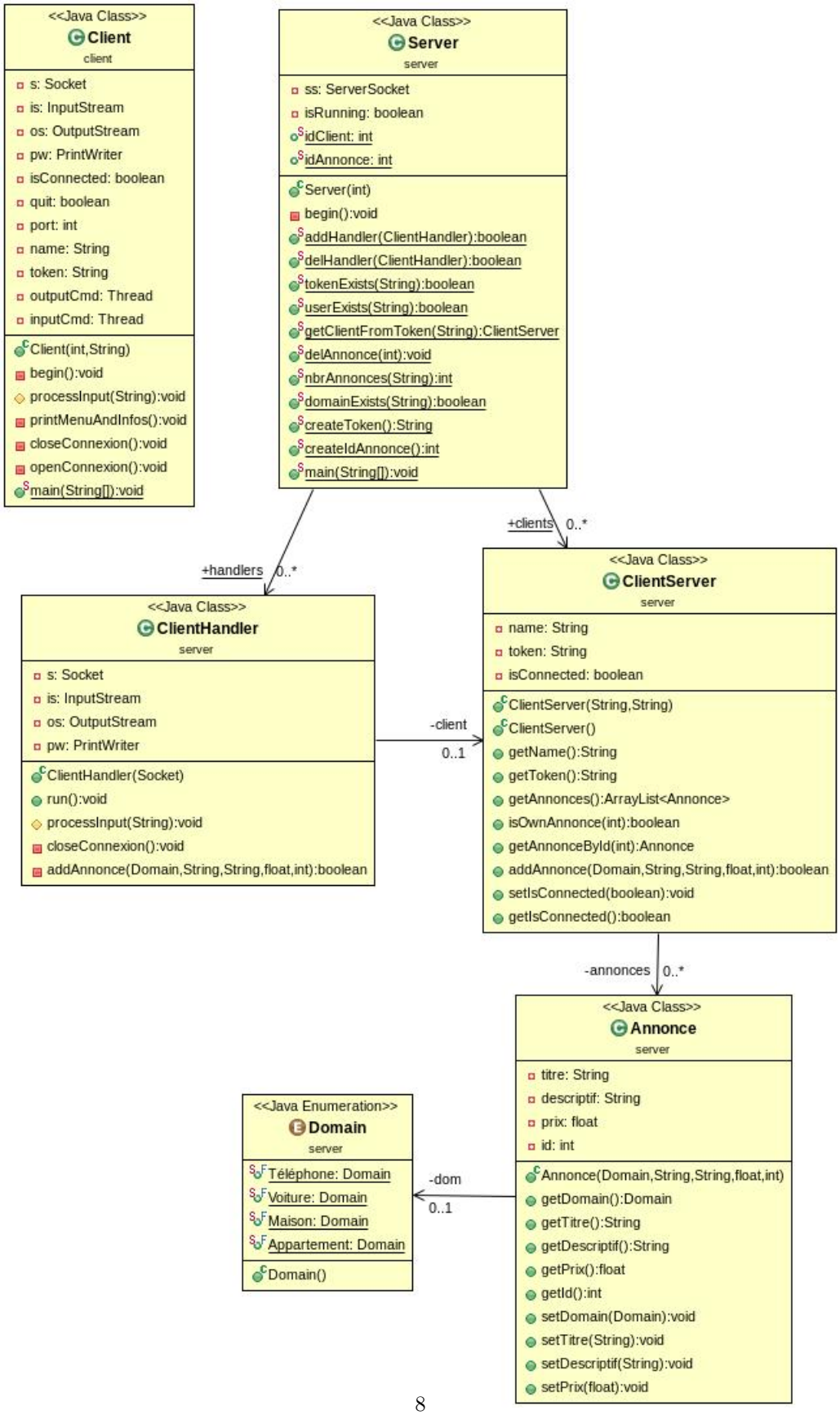

Figure 2: Diagramme UML

4 Mode d'emploi

4.1 Principe d'utilisation et de fonctionnement

4.1.1 Connexion et token

Une fois le serveur lancé, aucune action n'est nécessaire sur celui-ci. Tout comme le client, il y a un principe d'historique des commandes. A chaque requête reçue, une ligne d'informations est affichée dans la console.

Le client peut interagir avec le serveur via un menu de commande. Pour effectuer une commande, il faut saisir le chiffre associé et appuyé sur entrée.

Lorsque le client est lancé, la première action nécessaire est de saisir un nom d'utilisateur. Deux options s'offrent alors :

1. Pour intégrer le serveur en tant que nouvel utilisateur, il faut saisir un nom d'utilisateur et appuyer sur entrée. Si ce nom d'utilisateur n'est pas utilisé, le serveur renverra un token qu'il faudra impérativement conserver pour se reconnecter ultérieurement. Le cas échéant, si le nom d'utilisateur est déjà utilisé, le client ne peut se connecter au serveur. Le menu de commandes s'ouvre, mais il faut quitter et relancer l'application pour pouvoir saisir un nouveau nom d'utilisateur et intégrer le serveur sous un nom d'utilisateur non encore utilisé.

2. Saisir le token reçu à la première connexion. La connexion avec le serveur est alors établie avec les même droits qu'à la première connexion.

Dans notre implémentation, les tokens sont de la forme `#<nbr>`. La présence du suffixe dièse est indiqué dans le protocole. Celui-ci permet de différencier les chaînes de caractères naturel d'une chaîne de caractères représentant un token. L'argument `<nbr>` démarre à 1 et est incrémenté à chaque nouveau client géré par le serveur.

Une fois connecté avec un nouveau nom d'utilisateur ou un token, un menu s'ouvre, permettant d'effectuer des actions en saisissant au clavier l'entrée associer.

4.1.2 Présentation et contenu d'une annonce

Chaque annonce est associée à un unique id. Ces id sont transmis avec le contenu des annonces, et permettent de désigner l'annonce avec laquelle on veut agir (pour faire une mise à jour par exemple). L'incrémentation des id d'annonces se fait de la même manière que l'incrémentation des token : la première a l'id 1, la deuxième a l'id 2 et ainsi de suite.

4.1.3 Domaines des annonces

Les domaines présents dans notre implémentation sont `Téléphone`, `Voiture`, `Maison`, `Appartement`. Nous avons posé le principe qu'un client ne ne peut pas demander toutes les annonces du serveur. Il faut donc d'abord demander la liste des domaines, avant de demander les annonces d'un domaine spécifique.

4.1.4 Poster une annonce

Pour poster une annonce, il faut saisir un domaine, un titre, une description et un prix, dans cet ordre. La validité de ces données est vérifiée lorsqu'elles ont toutes été saisies. Par exemple, cela signifie par exemple qu'en cas de domaine non valide, le client en est informé après avoir saisie les quatre informations demandées et non directement après avoir saisi un mauvais domaine.

4.1.5 Mettre à jour une annonce

Pour mettre à jour une annonce, il faut saisir l'id de l'annonce (indiqué lors d'une demande d'annonces d'un domaine) puis saisir les mêmes informations, dans le même ordre, que lorsque l'on poste une annonce (domaine, titre, descriptif, prix). Si l'on ne souhaite pas changer une information, il suffit d'appuyer sur entrée lorsque l'information correspondante est demandée.

4.2 Protocole de test

Le protocole de test suivant indique, pour chaque étapes : `commande à effectuer` → `résultat attendu`. Ce protocole a été fait de tel sorte que chaque fonctionnalités soient testées.

1. Créer le serveur → le serveur démarre correctement.
2. Créer un client 1 nommé `C1` → `C1` est connecté au serveur, un token est renvoyée.
3. `C1` demande les domaines disponibles sur le serveur → les domaines sont renvoyés par le serveur à `C1`.
4. Le client `C1` demande les annonces d'un domaine inexistant (à déterminer en fonction du résultat du test précédent) → aucune annonce n'est renvoyée.
5. Dans la suite, nous supposerons que le domaine `Maison` existe sur le serveur. Le cas échéant, utiliser un domaine existant. Le client `C1` demande les annonces de `Maison` → aucune annonce n'est renvoyée.
6. Créer un client 2 nommé `C2` → `C2` et le connecter au serveur, un token est renvoyé.
7. Le client `C2` crée une première annonce dans le domaine `Maison` :

   ```
   Domaine    : Maison
   Titre      : Vente d'une grande maison.
   Descriptif : Au bord d'un lac et à trois pas d'une forêt.
   Prix       : 200000
   ```

   → cette annonce est créée.
8. Le client `C1` demande les annonces du domaine `Maison` → l'unique annonce créée est renvoyée.
9. Le client `C1` demande ses propres annonces → aucune annonce n'est renvoyée.
10. Le client `C1` crée une annonce correcte dans le domaine `Maison` :

    ```
    Domaine    : Maison
    Titre      : Vente d'une petite maison.
    Descriptif : En centre-ville.
    Prix       : 10000
    ```

    → cette annonce est créée.
11. Le client `C2` demande ses propres annonces → la première annonce créée est renvoyée.
12. Le client `C2` quitte l'application → il est correctement déconnecté.
13. Le client `C2` se reconnecte avec son token `#2` → il est correctement reconnecté.
14. Le client `C2` demande ses propres annonces → la première annonce est renvoyée.

15. Le client `C1` demande à supprimer la première annonce → requête refusée car il n'est pas propriétaire de cette annonce.

16. Le client `C2` demande à mettre à jour la première annonce :

```
Id         : <insérer_id_première_annonce>
Domaine    :
Titre      : Vente d'une super maison.
Descriptif :
Prix       : 1000
```

    → l'annonce est correctement mise à jour.

17. Le client `C1` demande à mettre à jour la première annonce :

```
Id         : <insérer_id_première_annonce>
Domaine    :
Titre      : Vente d'une gigantesque maison.
Descriptif :
Prix       : 999999
```

    → l'annonce n'est pas mise à jour car `C1` n'est pas propriétaire de la première annonce.

18. Le client `C1` demande à voir les annonces de `Maison` → toutes les annonces avec les bonnes modifications sont affichées.

19. Le client `C1` demande à supprimer la première annonce → l'annonce n'est pas supprimée car il n'en est pas propriétaire.

20. Le client `C2` demande à supprimer la première annonce → l'annonce est correctement supprimée.

21. Le client `C1` demande à voir les annonces de `Maison` → seul la deuxième annonce est renvoyée.

22. Le client `C2` est brutalement arrêté (`CTRL + C`) → le serveur prend en charge cet arrêt brutal.

23. Le serveur est brutalement arrêté (`CTRL + C`) → le client prend en charge cet arrêt brutal.

24. Quitter proprement le client `C1` → aucune erreur lors de la fermeture de l'application client.

5 Limites du protocole

- Ajouter des requêtes concernant les erreurs qui peuvent survenir afin de donner plus d'informations au client sur la nature de l'erreur. Par exemple, concernant l'intégration d'un nouvel utilisateur sur le serveur qui n'a pas pu aboutir, une requête indiquera que cela est dû au fait que le nom d'utilisateur soit déjà pris, tandis qu'une autre requête indiquera que le nom d'utilisateur est invalide (i.e. qui commence par un dièse).

6 Pistes d'amélioration

- Création d'une interface graphique client minimaliste. Cela sera notamment pertinent lorsque la communication pair à pair sera ajoutée.
- Ajouter une interface de commande sur le serveur pour pouvoir effectuer des requêtes (connaître le nombre de clients connectés en temps réel, le nombre d'annonces, stopper proprement l'exécution du serveur, ajouter des domaines en cours de fonctionnement etc).
- Ajouter la possibilité de retour chariot/paragraphes dans la description des annonces.
- Possibilité de créer des prix avec des espaces. Par exemple, `300 000` devra être valide.
- Mettre une génération de token aléatoire afin de ne pas pouvoir deviner le token en fonction du moment où l'on se connecte pour la première fois au serveur (étant donné qu'ils sont incrémentés de 1 en 1).
- Ajouter une connexion pair à pair entre clients souhaitant communiquer.
- Séparer connexion et log afin de donner la possibilité de consulter les annonces et les domaines sans avoir à créer un "compte" sur le serveur.