



# Création d'une application de vente entre particuliers par petites annonces

---

Natacha Gillaizeau-Simonian  
Guillaume Roumage

Protocole des service internet 2020/2021

# Sommaire

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Protocole client/serveur et pair-à-pair</b>	<b>3</b>
2.1	Diagramme . . . . .	3
2.2	Protocole des annonces . . . . .	5
2.2.1	Connexion . . . . .	5
2.2.2	Déconnexion . . . . .	6
2.2.3	Publication d'annonce . . . . .	6
2.2.4	Mise à jour d'annonces . . . . .	6
2.2.5	Suppression d'annonce . . . . .	7
2.2.6	Demander la liste des domaines disponible . . . . .	7
2.2.7	Consulter les annonces d'un domaine . . . . .	8
2.2.8	Consulter ses propres annonces . . . . .	8
2.2.9	Récupérer l'adresse IP d'un tier connecté . . . . .	9
2.2.10	Réception d'un message inconnu par le serveur . . . . .	9
2.3	Protocole de communication pair-à-pair . . . . .	10
2.3.1	Envoi d'un message texte . . . . .	10
2.3.2	Envoi d'un message d'acquittement . . . . .	10
<b>3</b>	<b>Conception et architecture de l'application</b>	<b>10</b>
3.1	Architecture client/serveur . . . . .	10
3.2	Architecture pair-à-pair . . . . .	11
3.3	Utilisation du token . . . . .	11
3.4	Mécanisme d'acquittement des messages entre utilisateurs . . . . .	11
3.5	Historique des messages . . . . .	11
<b>4</b>	<b>Mode d'emploi</b>	<b>12</b>
4.1	Côté serveur . . . . .	12
4.2	Côté client . . . . .	12
4.2.1	Connexion et déconnexion . . . . .	12
4.2.2	Les domaines existants . . . . .	12
4.2.3	Poster une annonce . . . . .	13
4.2.4	Consulter les annonces existantes . . . . .	13
4.2.5	Mettre à jour et supprimer une annonce . . . . .	13
4.2.6	Contacteur le vendeur d'une annonce . . . . .	13
4.2.7	Requêtes personnalisées . . . . .	14
<b>5</b>	<b>Vers une application sécurisée</b>	<b>15</b>
5.1	Amélioration de l'utilisation des token . . . . .	15
5.2	Chiffrement de la communication . . . . .	15
<b>6</b>	<b>Conclusion</b>	<b>16</b>
6.1	Limites du protocole . . . . .	16
6.2	Pistes d'amélioration . . . . .	16
<b>A</b>	<b>Diagramme UML</b>	<b>17</b>
A.1	Diagramme package <code>client</code> . . . . .	17
A.2	Diagramme package <code>server</code> . . . . .	17

# 1 Introduction

Ce rapport présente la réalisation d'une application de vente entre particulier par petites annonces à l'aide d'un serveur gestionnaire. Lorsqu'un utilisateur est intéressé par une annonce, il doit pouvoir correspondre directement avec l'utilisateur ayant proposé l'annonce.

Le rôle du gestionnaire est de collecter les annonces, de les diffuser aux clients, de les mettre à jour et de donner aux utilisateurs les informations pour correspondre avec les autres utilisateurs. Chaque client doit pouvoir proposer des objets à la vente par le biais d'une annonce visible par les autres utilisateurs. En plus de poster des annonces, un utilisateur peut mettre à jour ou supprimer une de ses propres annonces. Naturellement, chaque utilisateur peut consulter les annonces disponibles dans un domaine de vente spécifique.

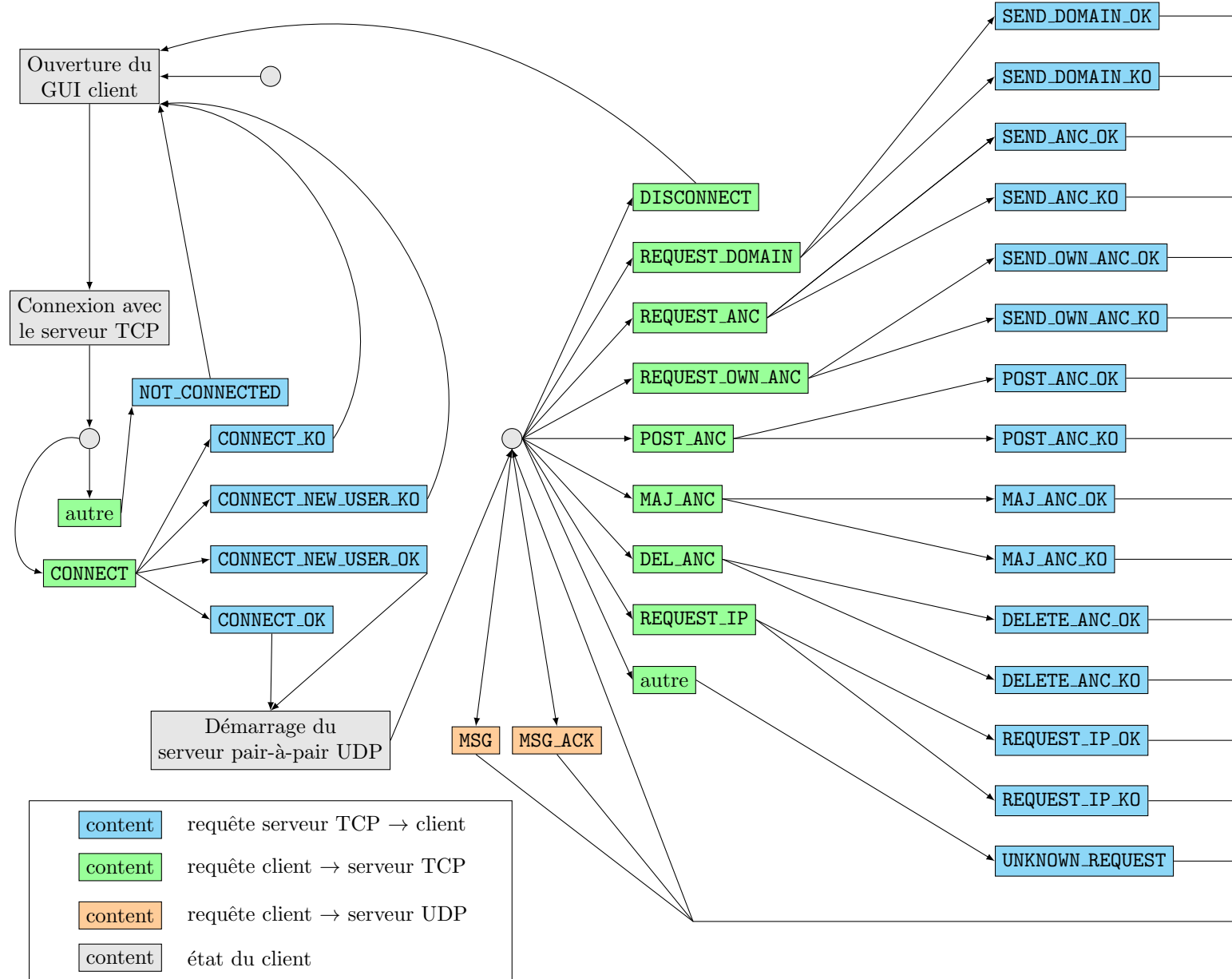
Ce projet a été effectué dans le cadre de l'UE *Protocoles des services internet*, avec un premier rendu implémentant la consultation et modification d'annonce uniquement, puis un second rendu avec toutes les fonctionnalités requises.

La version de java utilisée pour ce projet est la version 11.

Le reste du document est organisé de la manière suivante. En section 2 est présenté le protocole de communication client/serveur et pair-à-pair. Nous poursuivrons en section 4 par un mode d'emploi de l'application. La section 5 propose des pistes d'amélioration afin de tendre vers une application sécurisée. La section 6 conclue ce document, en évoquant notamment les limites du protocoles ainsi que des pistes d'améliorations.

## 2 Protocole client/serveur et pair-à-pair

### 2.1 Diagramme



## 2.2 Protocole des annonces

Chaque requête client ↔ serveur est de la forme suivante :

```
EN_TÊTE
<arg_1>
...
<arg_n>
.
```

L'entête indique l'action à effectuer ou l'information reçue lors de la réception de la requête. Chaque requête se termine par un point, signalant la fin de transmission des arguments. Les arguments sont envoyés sous forme de chaînes de caractères séparés par des sauts de ligne. Ces arguments sont ensuite transformés par l'application côté client ou le gestionnaire en d'autres types (`int`, `double` etc.) si besoin.

Le choix a été fait de choisir le port **1027** comme port de référence pour la connexion au serveur gestionnaire. Le protocole utilisé est le protocole TCP. On réfère le lecteur à la section 3.1 pour plus de détails concernant ce choix du protocole TCP.

### 2.2.1 Connexion

Lorsque le serveur reçoit la demande de connexion d'un nouvel utilisateur, il ajoute une entrée dans la structure de données permettant de gérer les connexions et les requêtes client ↔ serveur (dans notre contexte, c'est la classe `ClientHandler` qui se charge de cette tâche) et passe le statut de l'utilisateur à **connecté**. Lorsque l'utilisateur s'est déjà enregistré, seul ce changement de statut de connexion est effectué.

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

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

Lors de la première connexion, l'utilisateur saisit un nom d'utilisateur. Le serveur lui renverra alors un token qui devra être conservé secrètement et utilisé pour des connexions futures.

Dans le cas où l'utilisateur s'est déjà connecté au moins une fois, il peut se reconnecter avec son token. Une utilisation différente du token pourra être faite, notamment pour tendre vers une application sécurisée. Nous détaillerons cela dans la section 5.

Lorsqu'un enregistrement sur le serveur lors d'une première connexion est un succès, le serveur renvoie :

```
CONNECT_NEW_USER_OK
<token>
.
```

Si une première connexion échoue, le serveur renvoie :

```
CONNECT_NEW_USER_KO
.
```

En cas de succès lorsque le client est déjà enregistré sur le serveur, celui-ci renvoie :

```
CONNECT_OK
<token>
<utilisateur>
.
```

En cas d'échec, le serveur renvoie :

CONNECT\_KO

.

Dans le cas de futurs échanges avec le serveur, si le client n'est pas connecté, le serveur envoie :

NOT\_CONNECTED

.

### 2.2.2 Déconnexion

Lorsqu'un client souhaite se déconnecter, il envoie la requête suivante :

DISCONNECT

.

Le serveur ferme alors la connexion avec cet utilisateur.

### 2.2.3 Publication d'annonce

Comme nous l'avons vu en section 3, une annonce est représentée par cinq attributs : un domaine, un titre, une description, un prix et un id. Un utilisateur souhaitant créer une annonce envoie au serveur les quatre premiers attributs. Le serveur a la tâche de gérer le cinquième.

POST\_ANC

<domaine>

<titre>

<description>

<prix>

.

Lorsque la création d'annonce est un succès, le serveur renvoie :

POST\_ANC\_OK

<id\_annonce>

.

Le cas échéant, lorsqu'un échec de création d'annonce survient, le serveur renvoie :

POST\_ANC\_KO

.

### 2.2.4 Mise à jour d'annonces

Un utilisateur peut mettre à jour une de ses annonces avec la requête suivante :

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

En cas de réussite, le serveur renvoie :

```
MAJ_ANC_OK
<id_annonce>
.
```

Sinon, en cas d'échec :

```
MAJ_ANC_KO
.
```

### 2.2.5 Suppression d'annonce

Un utilisateur 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
.
```

### 2.2.6 Demander la liste des domaines disponible

Un utilisateur peut demander au serveur quels sont les domaines existants avec la requête suivante :

```
REQUEST_DOMAIN
<domaine>
.
```

En cas de réussite, le serveur renvoie :

```
SEND_DOMAIN_OK
<domaine_1>
[...]
<domaine_n>
.
```

En cas d'échec :

```
SEND_DOMAIN_KO
.
```

### 2.2.7 Consulter les annonces d'un domaine

Un client peut demander les annonces d'un domaine spécifique par la requête suivante :

```
REQUEST_ANC
<domaine>
.
```

En cas de réussite, le serveur renvoie :

```
SEND_ANC_OK
<id_annonce_1>
<domaine_1>
<titre_1>
<description_1>
<prix_1>
[...]
<id_annonce_n>
<domaine_n>
<titre_n>
<description_n>
<prix_n>
.
```

En cas d'échec :

```
SEND_ANC_KO
.
```

### 2.2.8 Consulter ses propres annonces

Un utilisateur peut consulter ses propres annonces avec la requête suivante :

```
REQUEST_OWN_ANC
.
```

En cas de réussite, le serveur renvoie :



```
SEND_OWN_ANC_OK
<id_annonce_1>
<domaine_1>
<titre_1>
<description_1>
<prix1>
[...]
<id_annonce_n>
<domaine_n>
<titre_n>
<description_n>
<prix_n>
.
```

Sinon, en cas d'échec :

```
SEND_OWN_ANC_KO
.
```

### 2.2.9 Récupérer l'adresse IP d'un tier connecté

Un utilisateur peut obtenir l'adresse IP et le nom du propriétaire d'une annonce par la requête :

```
REQUEST_IP
<id_annonce>
.
```

En cas de succès :

```
REQUEST_IP_OK
<ip_utilisateur>
<nom_utilisateur>
.
```

En cas d'échec :

```
REQUEST_IP_KO
.
```

### 2.2.10 Réception d'un message inconnu par le serveur

Si un en-tête inconnu est réceptionné par le serveur, celui-ci renvoie la commande :

```
UNKNOWN_REQUEST
.
```

## 2.3 Protocole de communication pair-à-pair

Pour la connexion entre les pairs, les utilisateurs utilisent le protocole UDP et écoutent sur le port **7021**. Tous les datagrammes auront une taille maximale de 1024 octets. Le point final sera utilisé pour séparer le contenu de la requête de la partie qui assure l'authenticité grâce à un HMAC. Un message est renvoyé N fois tant qu'aucun acquittement n'a été reçu, où le nombre N est laissé au choix de l'implémentation. Nous prendrons la valeur  $N = 5$ .

### 2.3.1 Envoi d'un message texte

Pour envoyer un message, un utilisateur envoie sur le serveur voulu la requête suivante :

```
MSG
<nom_émetteur>
<timestamp>
<msg>
.
```

Avec `timestamp` le temps UNIX en millisecondes.

### 2.3.2 Envoi d'un message d'acquittement

Pour acquitter la réception d'un message, l'autre pair renvoie l'acquittement suivant :

```
MSG_ACK
<émetteur_du_message_reçu>
<timestamp_du_message_reçu>
.
```

Le timestamp étant unique car en milliseconde (on considère qu'un utilisateur peut envoyer au maximum un message par milliseconde), on s'assure que cela acquitte un unique message.

## 3 Conception et architecture de l'application

Une annonce est composée de cinq éléments suivants : un domaine, un titre, une description, un prix et un identifiant, que nous désignerons par id. Un gestionnaire est chargé de gérer toutes les opérations faites par les clients relatives à ces annonces tels que l'ajout, la suppression, la mise à jour, la collection et la diffusion. Dans la suite, nous appellerons le gestionnaire : serveur ou serveur gestionnaire. Une architecture naturelle pour dans ce contexte est l'architecture client/serveur, que nous décrivons dans la sous-section suivante.

De plus, nous avons posé le principe qu'un utilisateur ne peut pas consulter toutes les annonces du serveur en même temps. Il peut seulement demander les annonces d'un domaine spécifiques. Nous avons fait ce choix pour limiter le nombre d'annonces envoyées à un utilisateur, en partant du principe qu'un utilisateur sait ce qu'il veut acheter.

Les domaines sont définis dès le début au sein d'une énumération dans le serveur. Il n'y a pas la possibilité d'ajouter des domaines en cours d'utilisation : cela peut faire l'objet d'une amélioration.

Nous avons également établi un diagramme UML du projet, voir annexe (*A Diagramme UML*).

### 3.1 Architecture client/serveur

Dans une architecture client/serveur, un ou plusieurs clients ont la possibilité de se connecter à un serveur. Dans notre projet, nous considérons qu'il existe un unique serveur, qui fait donc office de gestionnaire des annonces. Ce serveur centralise la quasi totalité des informations concernant les annonces et les clients : c'est

une architecture centralisée. Les clients communiquent avec le serveur (selon le protocole défini en section 2) pour effectuer des opérations sur les annonces.

Nous avons fait le choix d'utiliser le protocole TCP. Le protocole TCP est orienté connexion, cela nous assure que les paquets envoyés du client vers le serveur, et inversement. On est également assuré de l'intégrité des données, ce qui est primordiale afin de s'assurer que l'on poste ou que l'on réceptionne une annonce correctement.

Le serveur gestionnaire est capable de gérer plusieurs connexions simultanément, par la création d'un thread de la classe `ClientHandler`. Cette classe a pour but de gérer la connexion entre un utilisateur et le serveur gestionnaire. Chaque utilisateur interagira avec le serveur via son `ClientHandler`.

## 3.2 Architecture pair-à-pair

Pour permettre aux utilisateurs de correspondre directement entre eux, nous pouvons utiliser une architecture pair-à-pair. Cela signifie que chaque utilisateur est à la fois client et serveur : c'est une architecture distribuée. De plus, la communication directe entre utilisateurs permet de ne pas surcharger le gestionnaire avec les messages envoyés entre utilisateurs.

Nous avons fait le choix d'utiliser le protocole UDP pour l'architecture pair-à-pair principalement pour sa légèreté. Ce protocole utilise un mode de transmission sans connexion. Contrairement à TCP, nous ne pouvons donc pas nous assurer de la réception et de l'intégrité des messages. Cependant, si un utilisateur reçoit un message incohérent, il peut explicitement demander à l'autre utilisateur le renvoi de ce message. De plus, l'utilisation de UDP permet de supporter la montée en charge. En effet, contrairement à TCP, il n'est pas nécessaire de créer une socket pour chaque conversation. Autrement dit, TCP est un protocole 1 à 1 alors que UDP est un protocole 1 à n. Enfin, ce projet étant un projet étudiant, l'utilisation du protocole UDP pour la communication pair-à-pair nous permet de manier deux principaux protocoles de transmission : TCP pour l'architecture client/serveur et UDP pour l'architecture pair-à-pair.

## 3.3 Utilisation du token

Dans la présente version du projet, le token reçu à la première connexion d'un utilisateur permet de se connecter au serveur (celui-ci est saisi à la place du nom d'utilisateur). Il fait donc office de login et de mot de passe, dans le sens où le token de chaque utilisateur est connu uniquement de cet utilisateur. Dans une version plus sécurisée de l'application que nous développerons en section 5, le token ne sera plus utilisé par le serveur. Il servira à chiffrer les messages envoyés au serveur à l'aide d'un HMAC.

## 3.4 Mécanisme d'acquiescement des messages entre utilisateurs

Après le premier envoi d'un message inter-utilisateurs, celui-ci est renvoyé au maximum cinq fois avant d'être considéré comme perdu. S'il n'y a pas eu d'acquiescement après au moins 2 secondes ( $2^1$ ) secondes, alors on effectue un premier renvoi. On attend alors au moins 4 secondes ( $2^2$ ) avant d'effectuer un deuxième renvoi, puis au moins  $2^3$  secondes avant le troisième renvoi et ainsi de suite jusqu'au cinquième renvoi. Techniquement, les messages que les clients envoient sont stockés dans une structure spécifique : une `HashMap`, associant un message à son identifiant (celui-ci étant le nom de l'émetteur suivi du timestamp d'émission du message). Tant qu'un message n'a pas été acquiescé, il est conservé dans cette structure. Un message acquiescé est supprimé de cette structure. Dans le but de mettre à jour cette `HashMap`, l'application du client vérifie chaque seconde cette structure et fait les opérations nécessaires pour chaque messages (renvoie de message, attente ou suppression).

## 3.5 Historique des messages

Les messages échangés entre client ne sont pas stockés. En effet, ceux-ci sont écrits les uns en dessous des autres dans l'application du client et constitue un historique des messages. Lorsque le client se déconnecte, tous les messages sont effacés. On pourrait améliorer l'application pour que les messages soient conservés

ou même que le l'utilisateur puisse en recevoir alors qu'il n'est pas connecté et consulter ses messages à sa reconnexion.

## 4 Mode d'emploi

### 4.1 Côté serveur

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.

### 4.2 Côté client

Le client peut interagir avec le serveur via une interface graphique. L'interface possède une console dans laquelle s'affiche les message du serveur et autre messages d'informations.

#### 4.2.1 Connexion et déconnexion

Pour se connecter, le client doit simplement taper son nom d'utilisateur (si c'est sa première connexion) ou son token (pour une authentification). Il doit aussi taper l'adresse IP du serveur auquel il souhaite ce connecter. Une fois les informations entrées, il appuie sur le bouton *Connect*

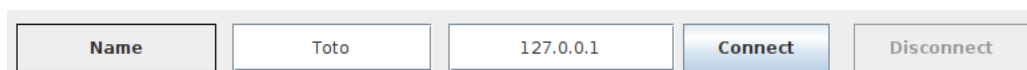
A screenshot of a graphical user interface for a client. It features a horizontal row of five elements: a text input field labeled 'Name' containing the text 'Toto', another text input field containing the IP address '127.0.0.1', a blue button labeled 'Connect', and a grey button labeled 'Disconnect'.

Figure 1: Exemple : Toto se connecte pour la première fois au serveur à l'adresse 127.0.0.1

A tout moment, l'utilisateur peut se déconnecter en appuyant sur le bouton *Disconnect*

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 naturels 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.

#### 4.2.2 Les domaines existants

Pour demander au serveur de lui envoyer la liste des domaines existants, l'utilisateur doit simplement appuyé sur le bouton *Ask Domains*. Il a alors un menu déroulant qui lui permet de consulter les domaines existants.

Il est à noter qu'avant de pouvoir de créer une annonce, l'utilisateur doit d'abord demander au serveur la liste des domaines existants.

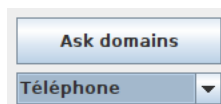
A screenshot of a graphical user interface element. It consists of a blue button labeled 'Ask domains' positioned above a dropdown menu. The dropdown menu has a blue header labeled 'Téléphone' and a small downward-pointing arrow on its right side.

Figure 2: Exemple de requête de domaines

Chaque annonce est associée à un unique identifiant. Ces identifiants sont transmis avec le contenu des annonces, et permettent de désigner l'annonce avec laquelle on veut agir (fait automatiquement avec l'interface). L'incrémentation des identifiants d'annonces se fait de la même manière que l'incrémentation des token : la première à l'identifiant 1, la deuxième à l'identifiant 2 et ainsi de suite.

### 4.2.3 Poster une annonce

Pour poster une annonce, un utilisateur connecté doit cocher la case *Create anc.* Une fois les informations rentrées, il pourra valider avec le bouton *Post anc.*

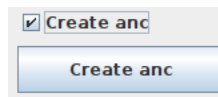


Figure 3: Boutons de création d'annonce

Le posteur de l'annonce peut ensuite entrer les informations requises : Titre, domaine, description, prix.

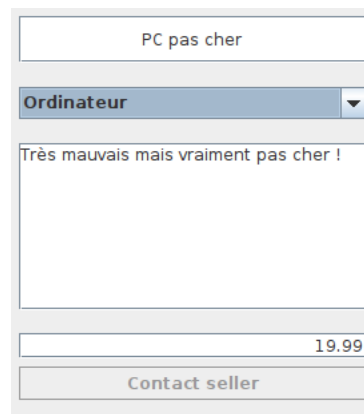


Figure 4: Exemple de l'annonce "PC pas cher" dans le domaine "Ordinateur" au prix de 19.99€

Noter qu'avant de poster une annonce, l'utilisateur doit tout d'abord demander la liste des domaines existants au serveur.

Une fois que l'utilisateur a créé toutes les annonces qu'il souhaite, il doit décocher la case *Create anc.*

### 4.2.4 Consulter les annonces existantes

Pour consulter les annonces dans un domaine, il suffit à l'utilisateur de choisir un domaine dans le menu déroulant, et toutes les annonces correspondantes s'afficheront. Il pourra ensuite regarder les détails d'une annonce en cliquant dessus.

Pour consulter ses propres annonces, le fonctionnement est le même à une exception près. Au lieu de choisir un domaine dans le menu déroulant, l'utilisateur doit cliquer sur le bouton *My anc.*

### 4.2.5 Mettre à jour et supprimer une annonce

Pour mettre à jour une de ses annonces, l'utilisateur doit cocher la case *Update anc.* Les détails de son annonce vont alors s'afficher et il pourra modifier ce qu'il souhaite puis confirmer avec le bouton *Update anc.*

S'il veut supprimer l'annonce, l'utilisateur peut, une fois la case *Update anc* cochée et l'annonce sélectionnée, appuyé sur le bouton *Delete anc.*

Lorsque l'utilisateur a fini de mettre à jour et supprimer ses annonces, il doit alors décocher la case *Update anc.*

### 4.2.6 Contacter le vendeur d'une annonce

L'utilisateur doit sélectionner l'annonce qui l'intéresse (cf 4.2.4 Consulter les annonces existantes) puis appuyer sur le bouton *Contact seller.*

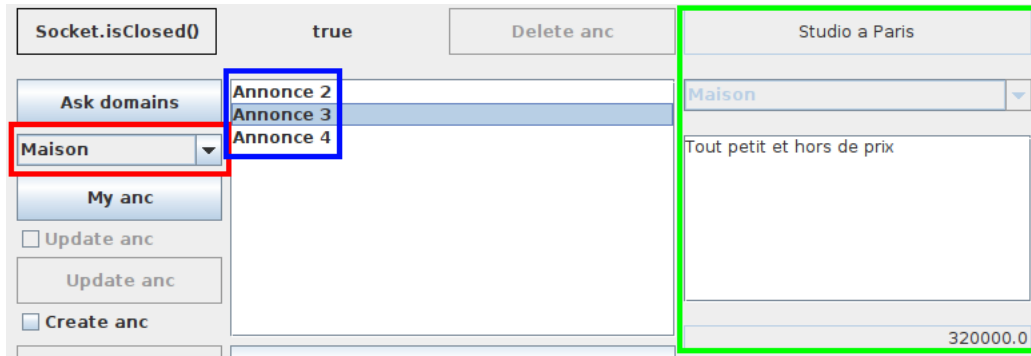


Figure 5: En rouge, le menu déroulant de sélection de domaine. En bleu les annonces correspondantes au domaine demandé. En vert, les détails de l'annonce sélectionnée

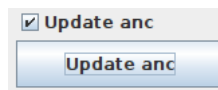


Figure 6: Case et bouton *Update Anc*

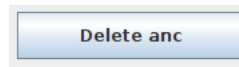


Figure 7: Bouton *Delete anc*

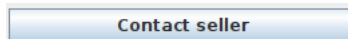


Figure 8: Bouton *Contact seller*

Une fenêtre de discussion s'ouvre alors.

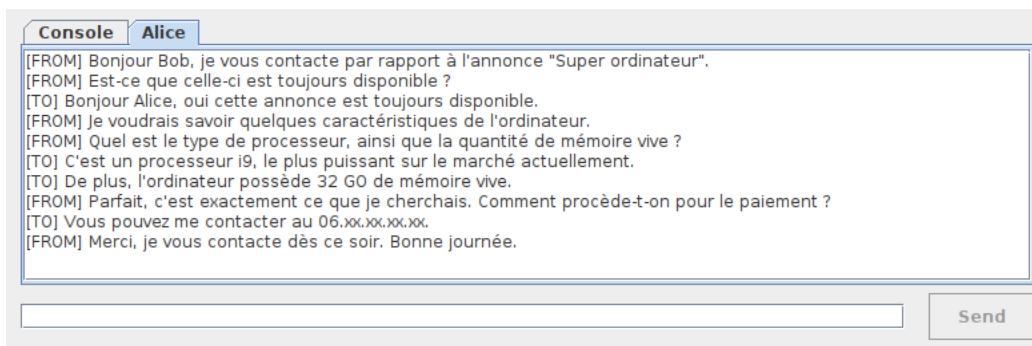


Figure 9: Fenêtre de chat avec Alice

#### 4.2.7 Requêtes personnalisées

Afin de tester notre application, il y a la possibilité d'envoyer au serveur des requêtes à la main avec le bouton *Custom commande*.

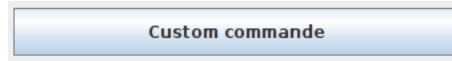


Figure 10: Bouton *Custom commande*

Une requête personnalisée sera envoyée vers le serveur avec un point '.' final. Ce choix a été fait car le serveur a été implémenté de façon à détecter une fin de requête lorsqu'un '.' est lu. Si une requête personnalisée était envoyée sans point final, cela pourrait bloquer ou modifier le comportement du serveur quant à son comportement concernant les requêtes suivant la requête personnalisée.

## 5 Vers une application sécurisée

### 5.1 Amélioration de l'utilisation des token

Un point important à sécuriser est l'authentification des utilisateurs. Pour l'instant, notre application génère des tokens permettant aux utilisateurs de se reconnecter avec ce token (qu'ils doivent donc garder secret).

Dans notre version actuelle, on peut noter deux problèmes de sécurité.

Premièrement, pour simplifier nous avons choisi de générer les tokens en incrémentant un compteur. Cela signifie que le premier utilisateur à se connecter aura le token #1, le suivant le #2 etc... Si nous voulons passer à une application sécurisée, nous devons générer ces tokens de manière aléatoire.

Deuxièmement, le serveur stocke les tokens tels quels. Cela signifie que si une personne mal intentionnée réussit à récupérer les données du serveur, elle possède alors la liste des tokens et peut donc usurper l'identité de tous les utilisateurs. Pour améliorer la sécurité, il faudrait utiliser une fonction de hachage et stocker non les tokens mais les hachés des tokens.

Ainsi, lorsqu'un utilisateur tente de se connecter, il entre son token. Le serveur le reçoit, calcule le haché de ce token et vérifie qu'il appartient à la base de données. En revanche, si une personne tierce récupère les données du serveur, les hachés de tokens ne lui seront d'aucune utilité.

### 5.2 Chiffrement de la communication

Notons aussi qu'il faudrait aussi chiffrer la communication car si un individu parvient à intercepter le message du client vers le serveur contenant le token, il pourra alors usurper l'identité de ce client.

Le canal de communication entre clients devrait lui aussi être chiffré.

On pourrait par exemple imaginer un échange de clé avec le protocole Diffie-Hellman, puis un chiffrement des données avec AES.

## 6 Conclusion

Dans ce rapport, nous avons présenté la conception d'une applications de petites annonces, permettant aux clients de diffuser des annonces et de correspondre directement entre eux. Les annonces sont diffusées en utilisant le protocole TCP avec une architecture client/serveur. Le serveur, ou le gestionnaire, s'occupe de traiter toute demandes (ajout, suppression, modification etc.) d'annonce. Dans le but de communiquer de manière direct, on utilise le protocole UDP avec une architecture pair-à-pair. Les client sont sont à la fois client et serveur pour la transmissions des messages.

### 6.1 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.

### 6.2 Pistes d'amélioration

- 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 en oeuvre toutes les propositions faites dans la partie 5 *Vers une application sécurisée*.
- 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.
- Faire en sorte que les clients puissent recevoir des messages même s'ils ne sont pas connectés afin qu'ils puissent lire leurs messages lorsqu'ils se reconnecteront.



## A Diagramme UML

### A.1 Diagramme package client

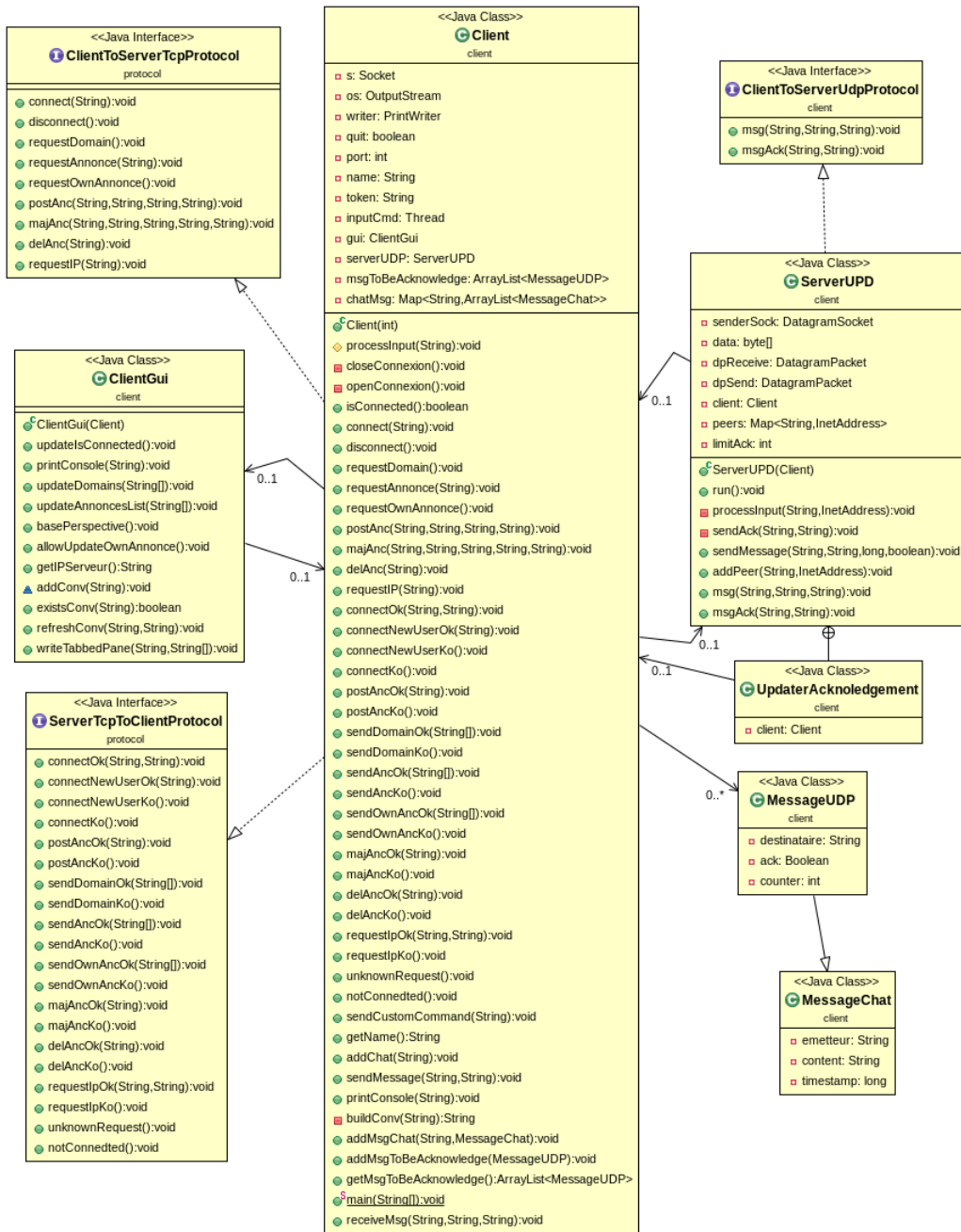


Figure 11: Diagramme UML du package Client

### A.2 Diagramme package server

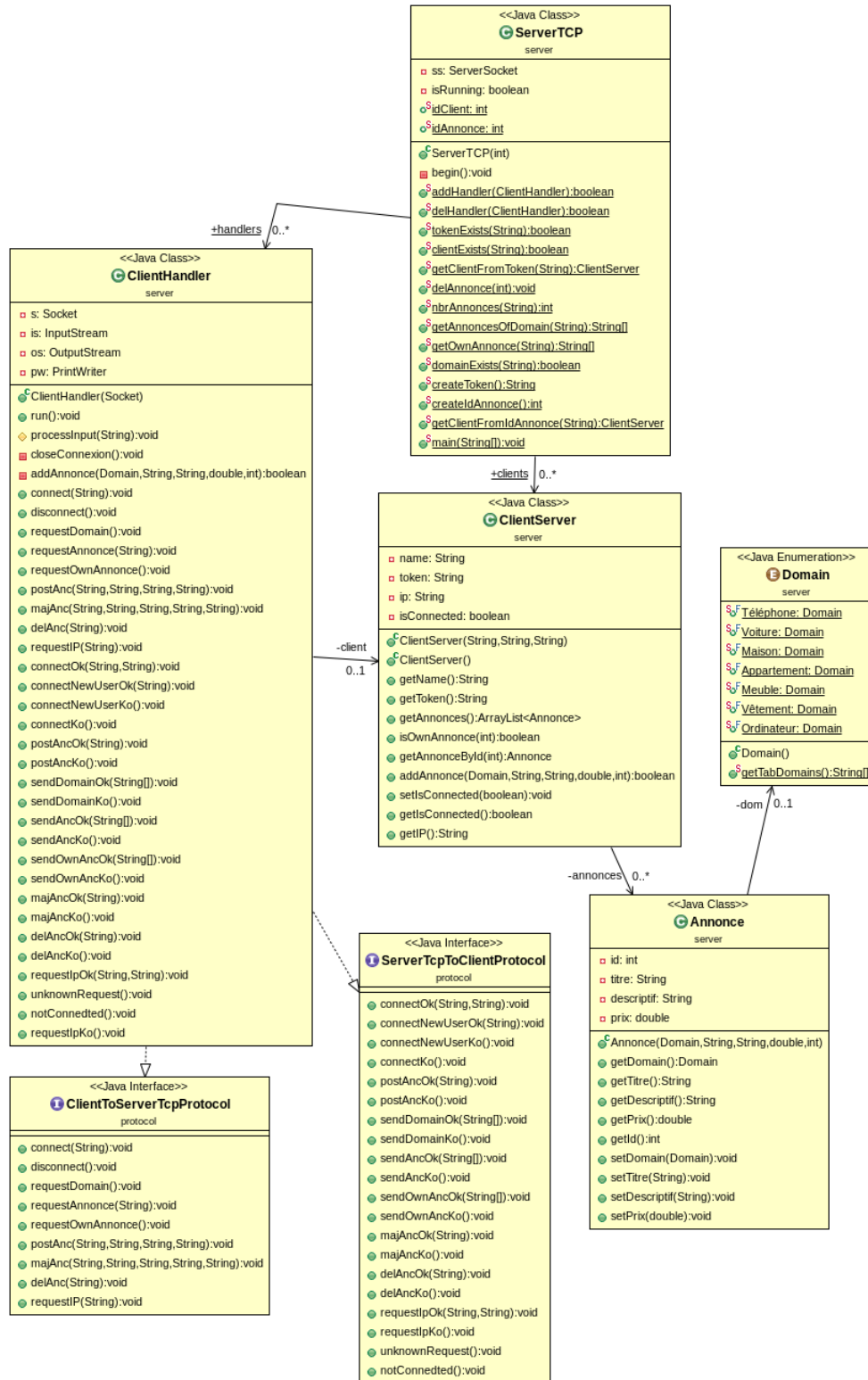


Figure 12: Diagramme UML du package Serveur