

# Rapport du Projet Réseaux

# **Banque de données**

---

FILLEUL Mika

STAVRIDIS Adonis



## Sommaire

<b>Sommaire</b>	<b>1</b>
<b>Architecture</b>	<b>2</b>
Application / Client	2
Serveur d'accès	2
Serveurs de données	3
Diagramme de la banque	4
<b>Echange de messages</b>	<b>5</b>
Description des types	5
Diagramme des échanges	6
<b>Documentation utilisateur</b>	<b>6</b>

## Architecture

Dans ce projet d'algorithmes des réseaux, il faut construire une banque de données distribuée avec des notions de respect de la qualité privée des données et étant tolérante aux pannes en même temps. Pour cela, il faut une application utilisée par un client, un serveur d'accès et des serveurs de données: ces serveurs forment l'ensemble de la banque de données sécurisée.


### Application / Client

Tout d'abord, on a besoin d'une application que l'utilisateur pourra lancer afin d'accéder à cette base de données. Cette application permettra à cet utilisateur de s'authentifier avec ses coordonnées, c'est-à-dire son nom d'utilisateur et son mot de passe (non hashé). Après s'être authentifié avec succès, l'utilisateur pourra écrire des instructions pour manipuler ses données dans la banque de données: il pourra lire, écrire ou supprimer des données. Après avoir effectué tout ce dont il souhaite, il se désauthentifie et peut soit s'authentifier avec un utilisateur différent (s'il possède plusieurs comptes par exemple) ou directement quitter l'application. Cette application cliente, va à son lancement essayer de se connecter au serveur pour vérifier s'il est en ligne. S'il se connecte avec succès, alors l'application peut continuer, sinon elle se ferme, puisque l'utilisateur ne peut rien faire avec cette application, si la connexion avec le serveur ne peut pas être établie. Si le client ne reçoit pas de réponse du serveur, après un envoi de paquet, alors le client va supposer que le serveur d'accès s'est déconnecté, et donc l'application se ferme.

L'application cliente écoute sur une socket et est représentée par un seul port.

### Serveur d'accès

Le serveur d'accès est le point central de la banque de données. Il permet de créer toutes les connexions nécessaires afin de mettre en place le bon fonctionnement du système. Ce serveur d'accès tourne tout le temps normalement, et c'est le premier processus à être lancé pour mettre en place la banque: s'il n'est pas fonctionnel lors du lancement des serveurs de données ou des applications clientes, ces derniers ne pourront pas se connecter: ils vont supposer que le serveur d'accès est hors ligne puisqu'il ne répond pas, et donc ces processus ferment. Ce serveur d'accès répond deux demandes essentielles: permettre la connexion et authentification des clients et des serveurs de données, mais aussi répondre aux instructions des clients et des serveurs de données. Le serveur d'accès est composé alors de deux threads, chacun effectuant l'une des fonctionnalités précisées ci-dessus. Lors de la connexion première



avec un processus, il reçoit des informations relatives à ce dernier, qu'il stocke dans une liste pour pouvoir reconnaître les processus connectés, et par la suite, il doit pouvoir traiter les instructions / demandes des clients et serveurs de données. Pour cela, dès que le serveur reçoit une instruction (lire, écrire, supprimer ou synchroniser pour les serveurs de données), il la transforme en demande, qu'il envoie aux serveurs de données correspondants. Ceux-ci lui répondent en ayant traité la demande, et donc le serveur véhicule cette réponse au processus ayant envoyé l'instruction initiale. Ainsi, il est possible pour les clients et les serveurs de données d'envoyer des instructions aux serveurs de données, ce qui permet une mise en place d'un système de synchronisation entre les données stockées dans les serveurs. Il faut noter que si l'instruction reçue est de lire, il n'est pas nécessaire de communiquer avec tous les serveurs de données: il suffit de recevoir la réponse que d'un serveur de données du bon champ. Cependant, pour écrire ou supprimer des données, le serveur d'accès va faire en sorte d'envoyer cette demande à tous les serveurs de données correspondant aux bons champs, afin que tous les serveurs aient les mêmes informations. Pour synchroniser les données entre les serveurs de données, il faut uniquement deux serveurs de même champ qui communiquent entre eux via le serveur d'accès, car tous les serveurs de données connectés au serveur d'accès possèdent les mêmes informations.

Le serveur d'accès écoute sur deux sockets différentes, et est donc représenté par deux numéros de ports différents.

## Serveurs de données

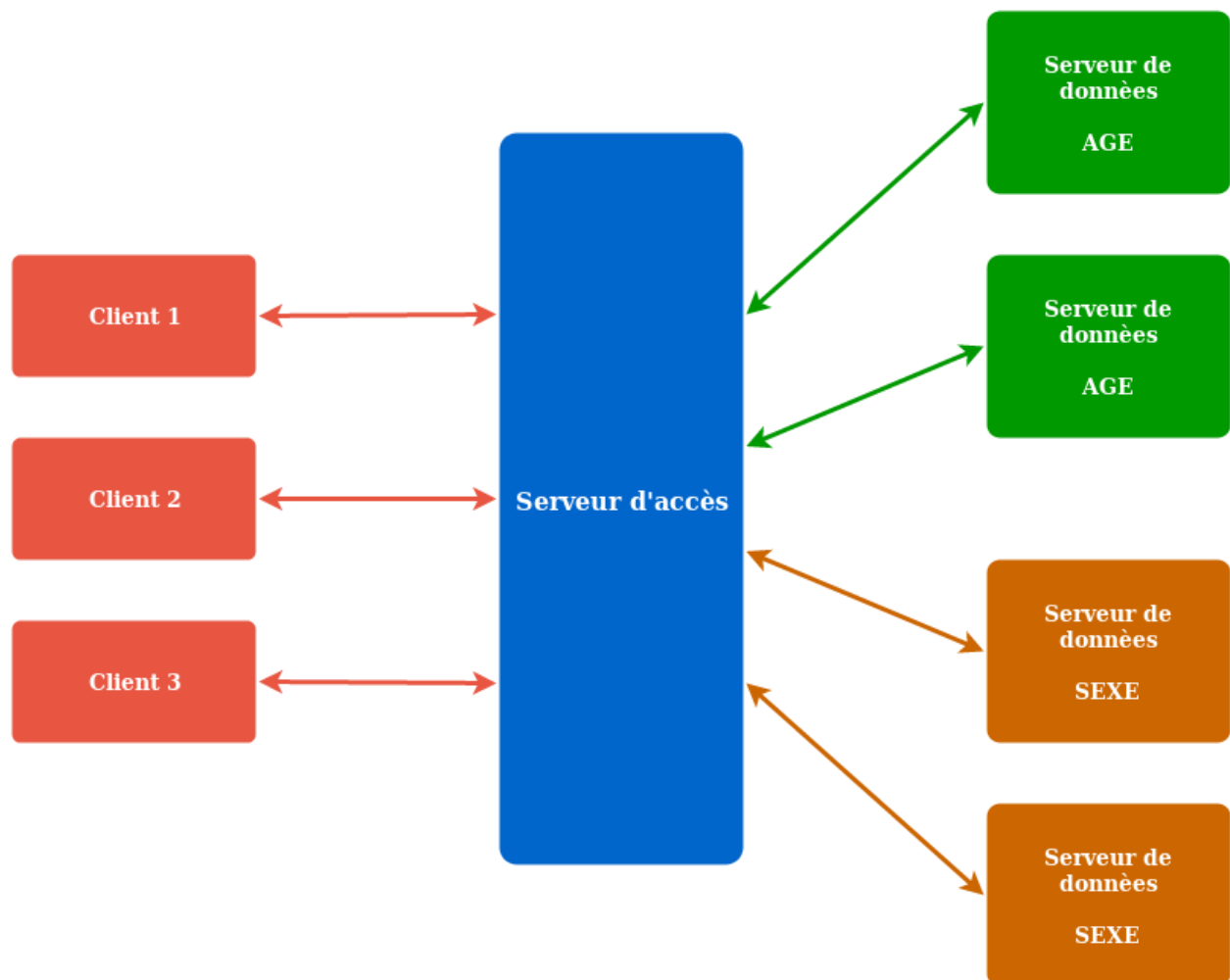
Les serveurs de données permettent de stocker en mémoire toutes les informations des utilisateurs. Chaque serveur ne stocke les valeurs que d'un seul champ, mais peut enregistrer les valeurs pour ce champ pour plusieurs utilisateurs. Afin de mettre en place un système tolérant aux pannes, il faut que les données soient dupliquées: il faut donc au moins deux serveurs de même champ, puisque si l'un des serveurs tombe en panne alors le deuxième serveur du même champ sera toujours fonctionnel et n'aura pas perdu ses informations. C'est pourquoi, une demande d'écriture de la part du client est envoyée à tous les serveurs concernés, pour qu'ils possèdent tous les mêmes informations (le serveur d'accès traite lui-même l'envoi des demandes aux différents serveurs de données). Ces serveurs de données reçoivent donc une demande du serveur d'accès, la traitent, et puis envoient une réponse en fonction de celle-ci. Les informations sur les utilisateurs sont stockés dans une liste chaînée, qui est manipulée à chaque demande reçue. Un problème qui pourrait se poser est le suivant: si un serveur de données de champ âge est déjà connecté au serveur d'accès et qu'un deuxième serveur de ce même champ est lancé, alors toutes les données sur le premier serveur ne seront pas présentes sur le deuxième. Il faut donc mettre en place une fonctionnalité de synchronisation des données entre les serveurs de même champ. Pour cela, si un serveur de données se connecte au serveur

d'accès et il n'est pas premier de son champ, alors il envoie une instruction au serveur d'accès pour commencer la fonctionnalité de synchronisation, qui permet de copier toute la liste de données du premier serveur du même champ à ce deuxième serveur récemment connecté. Ainsi, les données sont synchronisées dès le lancement des serveurs.

Un serveur de données écoute sur une seule socket et est représentée par un seul port.

## Diagramme de la banque

Voici un diagramme représentant les différentes connexions possibles de cette banque de données:



## Echange de messages

### Description des types

Afin de communiquer entre eux, les clients et serveurs s'envoient des messages contenant des informations essentielles pour effectuer une ou plusieurs instructions. Dans notre banque de données, les clients et serveurs s'envoient des structures de données. Puisque cette banque est utilisée dans un environnement local et codée entièrement en langage C, alors il n'y a pas vraiment de risque à utiliser les structures de données pour l'envoi de messages. Cependant, si la banque était utilisée non-localement, alors il aurait été nécessaire de transformer ces structures en des messages compatibles avec les normes de réseau (transformer les structures en un buffer).

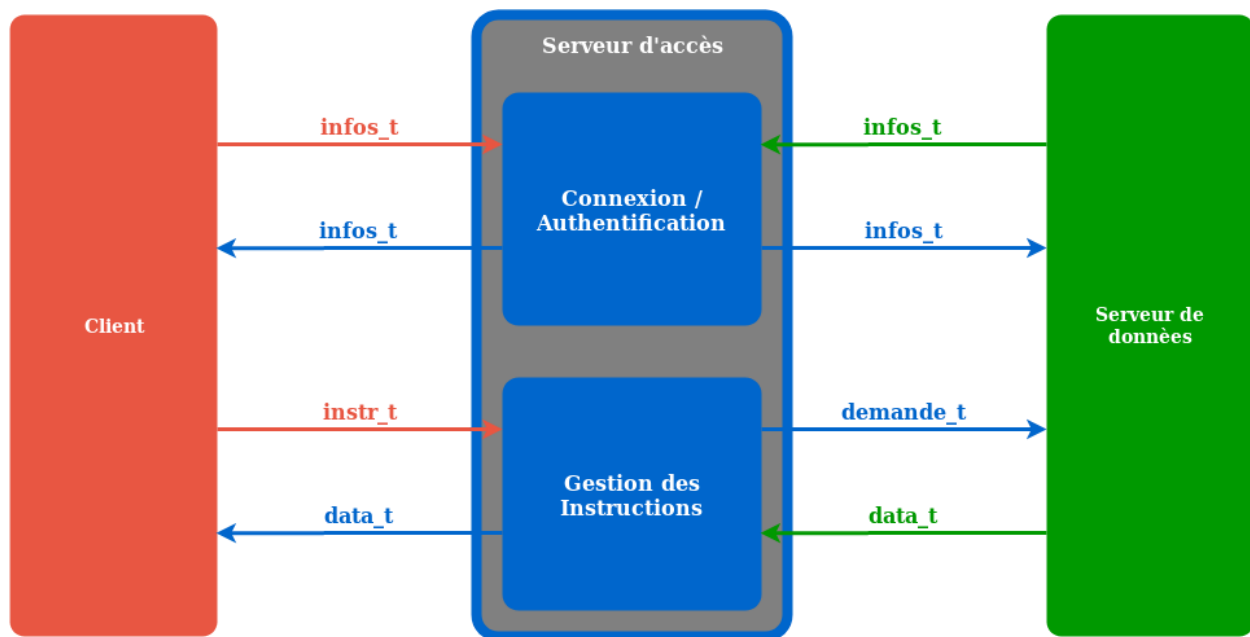
Ainsi, nous avons définis plusieurs nouveaux types ou structures, se trouvant dans le fichier `/include/types.h` du projet. Voici une liste de tous les types, utilisés pour l'envoi de messages:

- **infos\_t**: cette structure contient toutes les informations nécessaires pour permettre d'identifier un processus qui veut se connecter ou s'authentifier au serveur d'accès. Elle permet donc la connexion et l'authentification d'un client, mais aussi la connexion d'un serveur de données.
- **instr\_t**: cette structure est une traduction des commandes lire, écrire ou supprimer, entrées par l'utilisateur dans un processus client. Elle est aussi utilisée pour donner le départ pour la fonctionnalité de synchronisation entre deux serveurs de données de même champ.
- **demande\_t**: cette structure est une traduction d'une instruction de type `instr_t` mais pour un seul champ, car une instruction peut être exécutée sur plusieurs champs. Cette demande contient toutes les informations nécessaires à un serveur de données pour effectuer l'équivalent de l'instruction entrée par l'utilisateur.
- **data\_t**: cette structure est envoyée par les serveurs de données, contenant une chaîne de caractère représentant la réponse à une demande.

Tous ces types possèdent des arguments qui peuvent représenter un état d'un processus (connecté, authentifié, etc...) ou d'une instruction (valide, non valide, erreur, etc...). Ainsi en vérifiant les états, il est possible de plus facilement reconnaître et gérer les erreurs qui pourraient survenir. En général, toute structure reçue après un envoi, servira d'acquittement.

## Diagramme des échanges

Voici un diagramme représentant ce que chaque processus envoie ou reçoit:



Il faut noter que lors de la synchronisation, le serveur de données qui envoie les données, donc le serveur qui est déjà connecté au serveur d'accès, va être considéré comme un client. Il est donc possible, lors de la synchronisation que les serveurs de données envoient des **instr\_t** et reçoivent des **data\_t**, qui ces-derniers serviront d'acquittement.

## Documentation utilisateur

Pour la documentation utilisateur se référer au fichier *README.md* du projet.