

DOCUMENTATION SERVEUR

Talkie

Sommaire

A) Base de données.....	Page 2
I) Schéma relationnel.....	Page 2
II) Fonction des tables.....	Page 3-6
B) Première utilisation.....	Page 7
I) Lancement.....	Page 7
II) Utilisation.....	Page 8-11
C) Fonctionnement.....	Page 12
I) Communication.....	Page 12
II) Gestion des messages.....	Page 13
III) Connexion.....	Page 14
IV) Inscription.....	Page 15
V) Profil utilisateur.....	Page 16
VI) Demandes.....	Page 17-19
VII) Amis.....	Page 20
VIII) Messages privés.....	Page 21
IX) Commandes.....	Page 22-23
Annexe : Codes d'erreur.....	Page 24



A) Base de données

Consultez les fichiers et la procédure d'installation (dossier Documentation) sur le [Github du projet "Talkie"](#) pour pouvoir commencer l'utilisation.

I) Schéma relationnel

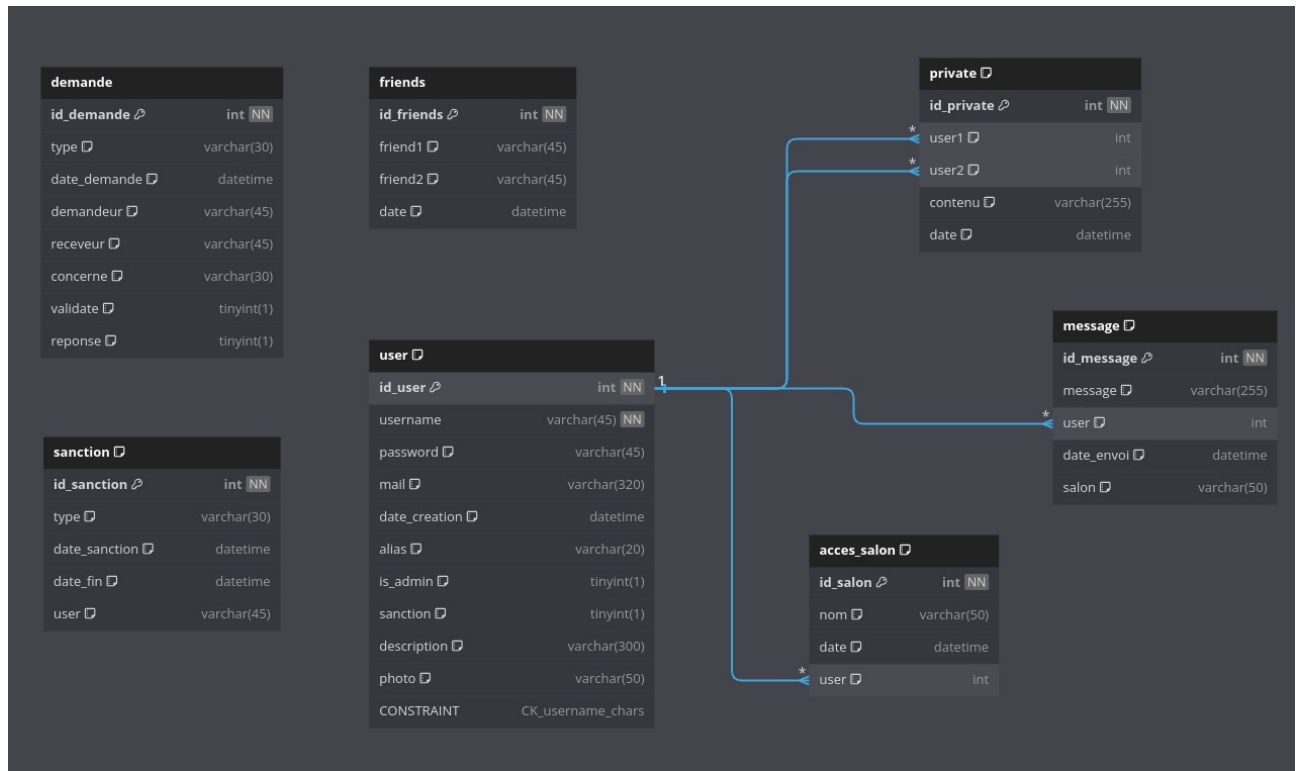


Illustration 1: cf. [savemysql.sql](#)



II) Fonction des tables

Pour gérer l'application, il est nécessaire d'utiliser une Base de données (ici mysql) pour sauvegarder toute les données.

1. Table user

Cette table répertorie la listes des utilisateurs existants.

- username

Identifiant utilisateur unique

- password

Mot de passe de l'utilisateur

- mail

Mail de l'utilisateur

- date_creation

Date de création du compte

- alias

Pseudonyme non unique de l'utilisateur

- is_admin

Booléen pour vérifier si l'utilisateur a les droits admin

- sanction

Booléen pour vérifier si l'utilisateur a une sanction

- description

Description destinée au profil de l'utilisateur personnalisable

- photo

La valeur string associé à l'image de profil de l'utilisateur

- CK_username_chars

Contrainte d'unicité pour interdire les caractères [%#'&] pour le username.

2. Table message

Cette table stocke l'historique des messages dans le serveur.

- message

Contient le contenu des messages

- user

Clé étrangère de user.id_user pour connaître l'expéditeur des messages

- date_envoi

Date d'envoi des messages

- salon

Salon dans lequel est envoyé le message

3. Table acces_salon

Chaque ligne de cette table déclare les droits d'accès d'un utilisateur à un salon.

- nom

Nom du salon

- date

Date à laquelle l'utilisateur a eu les droits d'accès à un salon

- user

Clé étrangère de user.id_user de l'utilisateur ayant accès au salon

4. Table private

Cette table stocke tous les messages privés.

- user1

Clé étrangère de user.id_user d'un utilisateur de la conversation privé

- user2

Clé étrangère de user.id_user d'un utilisateur de la conversation privé

- contenu

Contenu du message privé

- date

Date d'envoi du message

5. Table sanction

Contient la liste des sanctions des utilisateurs.

- type

Type de sanction (kick ou ban)

- date_sanction

Date de la condamnation

- date_fin

Date de fin de la sanction (si temporaire sinon nulle)

- user

Identifiant de l'utilisateur sanctionné.

6. Table demande

Cette table contient la liste des demandes des utilisateur.

- date_demande

Date de la demande

- type

Type de la demande (Salon, Admin, Ami)

- demandeur

Identifiant de l'utilisateur qui effectue une demande

- receveur

Identifiant de l'utilisateur qui reçoit une demande

- concerne

En quoi concerne la demande (ex : nom du salon)

- Validate

Si la demande est une réponse, savoir si c'est un refus ou non

- Reponse

Booléen pour savoir si la demande est une réponse

7. Table friends

Cette table contient la liste des relations amicales

- friend1

Identifiant d'un utilisateur ami à un autre

- friend2

Identifiant d'un utilisateur ami à un autre

- date

Date du début de la relation.

B) Première utilisation

Consultez les fichiers et la procédure d'installation (dossier Documentation) sur le [Github du projet "Talkie"](#) pour pouvoir commencer l'utilisation.

I) Lancement

1) Placez-vous au bon endroit dans le terminal

Pour lancer le programme ouvrez un terminal.
Ensuite situez-vous au même emplacement dans l'arborescence de fichiers de votre machine que le dossier **Server**.

```
cd /home/moi/Bureau/Server
```

2) Exécutez le fichier

Ensuite pour lancer le Serveur effectuez la commande

```
python3 Server.py
```

ou

```
python Server.py
```

ou (sur Linux uniquement)

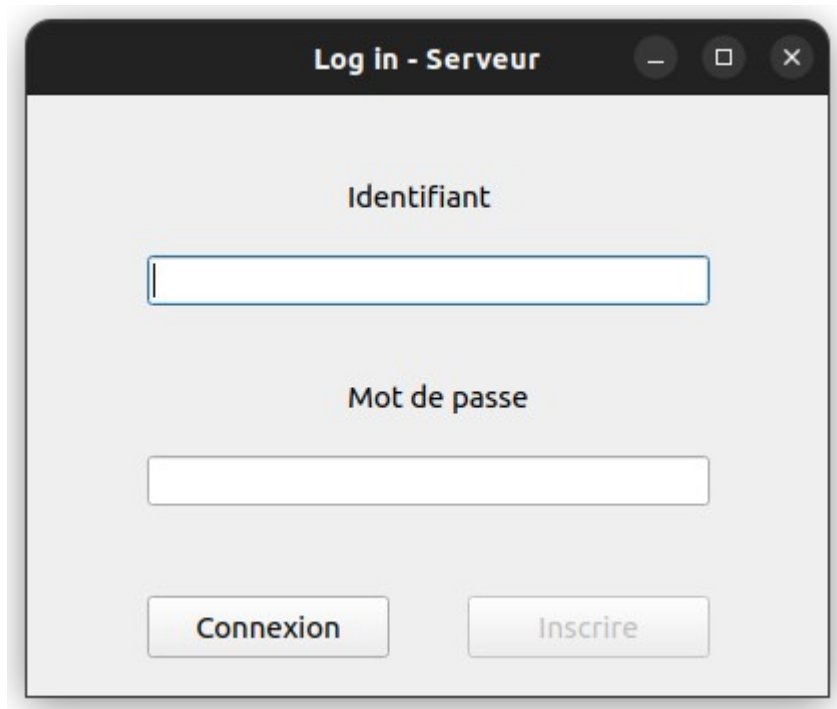
```
/bin/python Server.py
```

Sur Windows vous pouvez également cliquer 2 fois sur le fichier **Server.py** si vous avez un interpréteur correct. Vous pourrez également créer un raccourci si vous le souhaitez.

II) Utilisation

1) Fenêtre de connexion

Voici ce qui devrait normalement s'afficher.



À présent pour vous connecter et lancer le serveur vous devez écrire :

- Dans la case Identifiant : « **root** »
- Dans la case Mot de passe: « **toor** »

Ce sont les valeurs par défaut. Utilisez le script **admin_update.py** afin de modifier les identifiants par défaut par les valeurs que vous souhaitez.

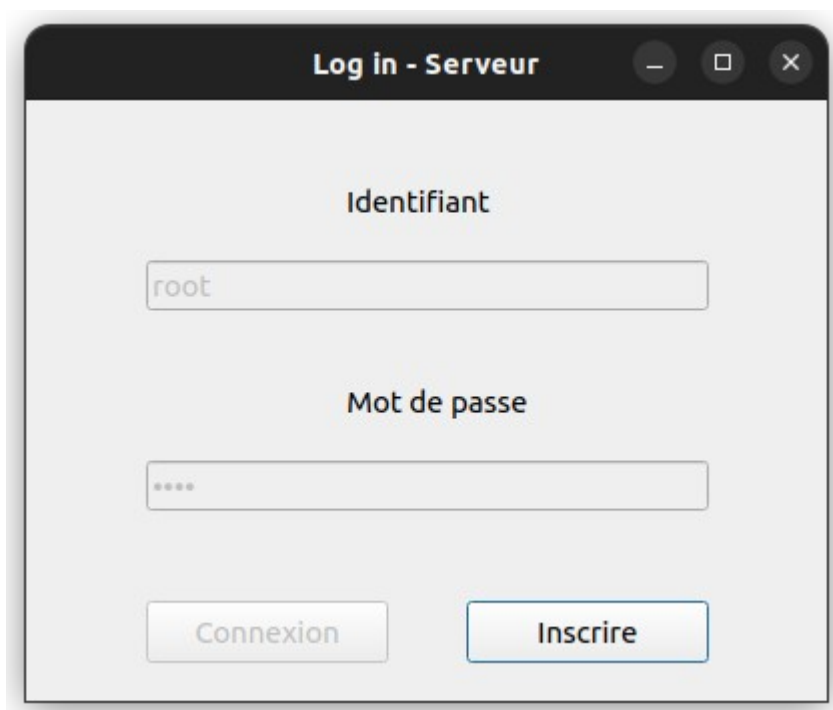
Attention votre choix est **définitif** (ou presque) et par sécurité **vous devrez détruire le programme**, après l'unique utilisation pour ne pas risquer d'endommager votre script Server dans des cas particuliers.

Si vous modifiez **Server.py** via **admin_update.py** vous devrez relancer le programme pour que les modifications s'appliquent.

Si nécessaire, vous pouvez les changer manuellement dans le programme **Server.py** en recherchant ces lignes.

```
if username == 'root':  
    if password == "root":  
        self.username.setEnabled(False)  
        self.password.setEnabled(False)  
        self.connect_button2.setEnabled(False)  
        self.signup_button.setEnabled(True)  
        self.start_server.emit()  
        window.show()
```

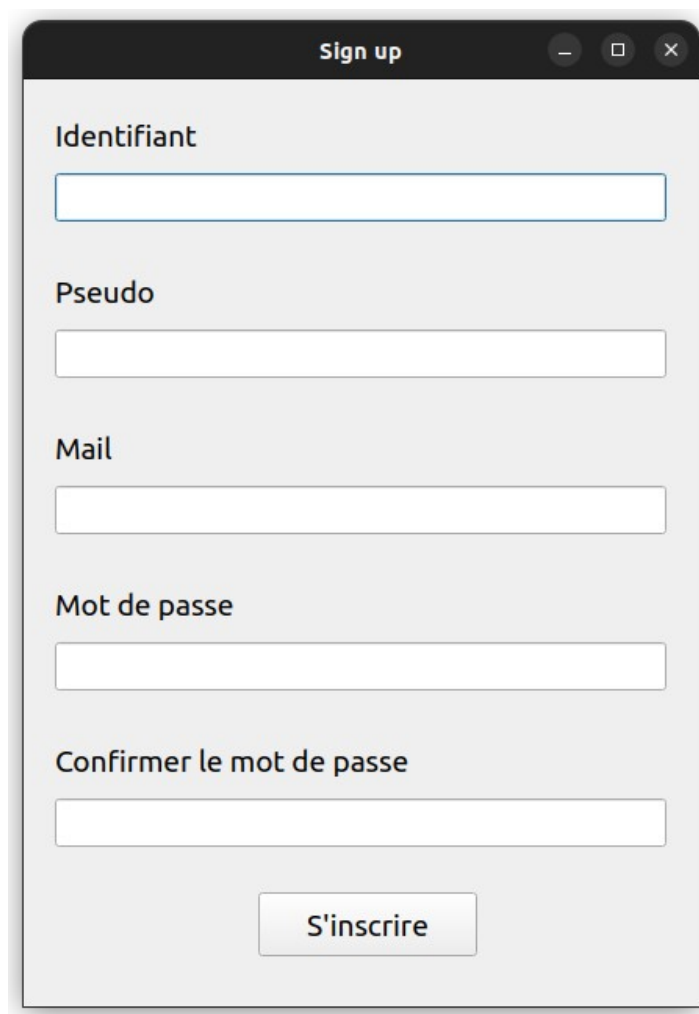
Si vous êtes bien connecté, votre fenêtre ressemblera à ceci :



De plus, la fenêtre de gestion du serveur apparaîtra en parallèle et vous aurez accès à la fenêtre d'inscription.

Conseil : vous pouvez utiliser le bouton « Entrer » de votre clavier au lieu de cliquer.

2) Fenêtre d'inscription



The image shows a 'Sign up' window with a dark title bar containing the text 'Sign up' and standard window control buttons (minimize, maximize, close). The window has a light gray background and contains the following elements from top to bottom:

- A label 'Identifiant' followed by a white rectangular input field.
- A label 'Pseudo' followed by a white rectangular input field.
- A label 'Mail' followed by a white rectangular input field.
- A label 'Mot de passe' followed by a white rectangular input field.
- A label 'Confirmer le mot de passe' followed by a white rectangular input field.
- A button labeled 'S'inscrire' centered at the bottom.

Lorsque vous êtes connecté au serveur, vous avez la possibilité de créer des utilisateurs clients depuis le serveur (qui sont inscrits dans la BDD). Ces derniers auront forcément les droits super utilisateurs alors agissez avec précaution.

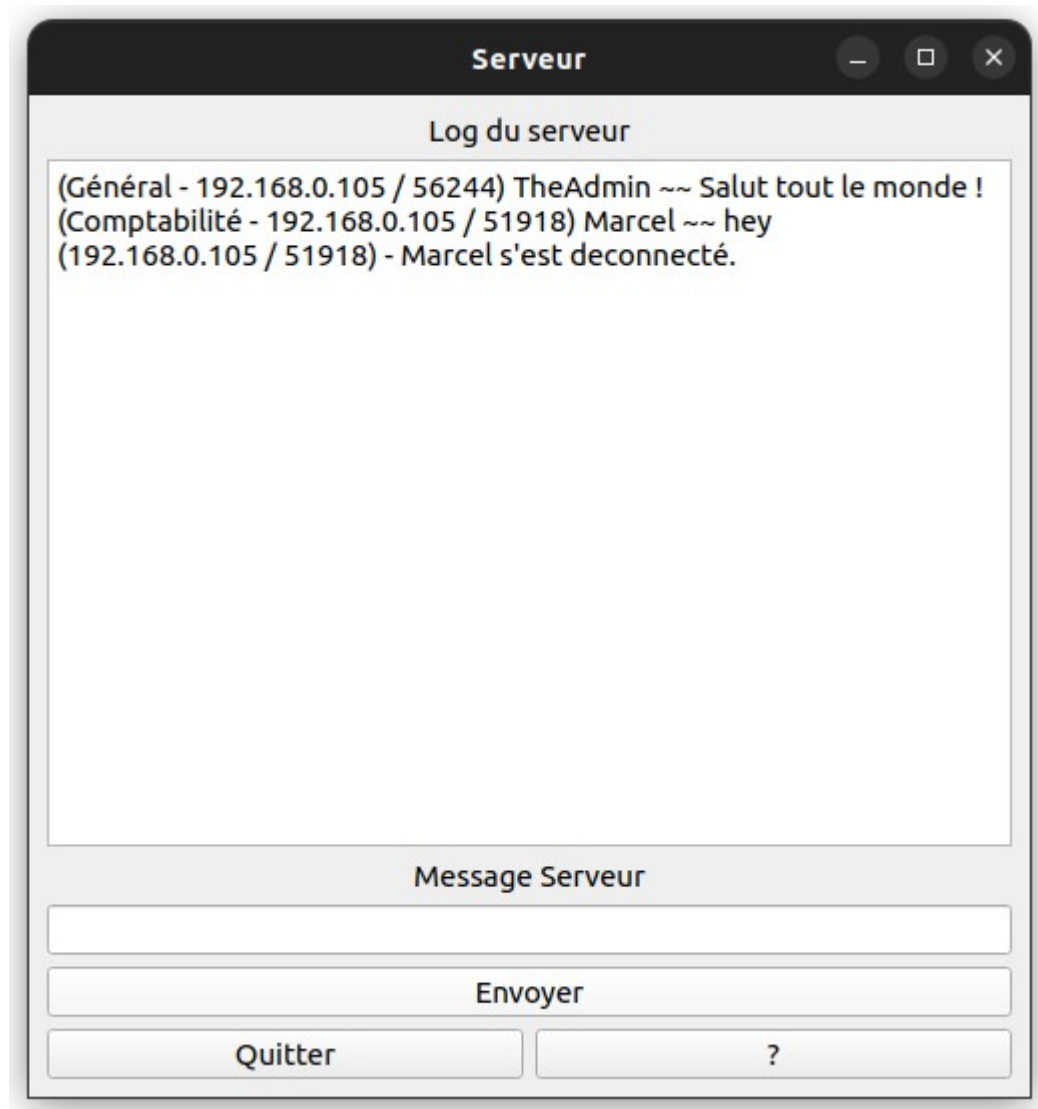
Vous devez respecter les règles suivantes :

- Identifiant unique
- Identifiant sans caractères interdit
- Identifiant ne s'appelant pas « Admin » ou « admin » (interdits pour des raisons pratiques de fonctionnement).
- Respect du format de mail.
- Mots de passe identiques.

3) Fenêtre de gestion du serveur

La fenêtre affiche tous les logs du serveur avec le détail de l'adresse IP, de l'utilisateur, du salon, et le contenu du message.

Lorsque un client se déconnecte cela est indiqué dans les logs.



L'interface permet également au serveur d'envoyer des messages qui apparaîtront dans tous les salons chez les utilisateurs, néanmoins les messages Serveur ne sont pas enregistrés dans la BDD.

Le bouton « Quitter » arrête le Serveur et déconnecte tous les clients, tandis que le bouton « ? » indique une description utile.

C) Fonctionnement

Cette partie se veut plus détaillée sur le fonctionnement technique du Serveur.

I) Communication

1. Utilisation des threads

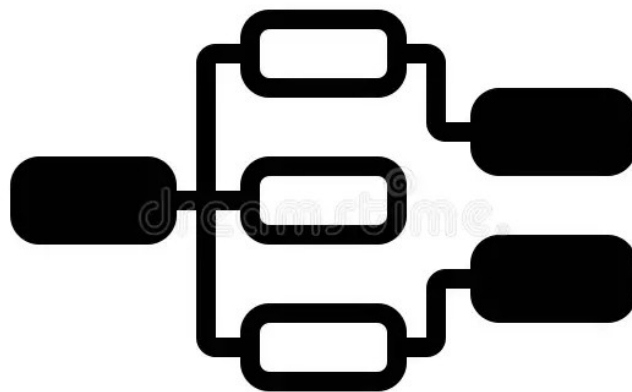
Le serveur utilise des sockets pour communiquer avec les clients. Il crée un socket, le lie à une adresse IP et un port spécifiques, puis écoute les connexions entrantes.

Pour chaque client qui se connecte, le serveur crée un nouveau thread. Cela permet au serveur de gérer plusieurs clients simultanément, chaque thread s'occupant de la communication avec un client spécifique.

2. Récupération des connexions

Lorsqu'un client se connecte, le serveur accepte la connexion, créant ainsi un nouveau socket pour communiquer avec ce client. Le serveur récupère également l'adresse du client.

Le serveur utilise un dictionnaire pour stocker les informations sur les clients. La clé est le socket client et la valeur est l'adresse du client. Cela permet au serveur de garder une trace de tous les clients connectés et de communiquer avec eux individuellement.



II) Gestion des messages

1. Réceptions des messages avec système d'entêtes

Le serveur reçoit les messages des clients avec un système d'entêtes. L'entête contient des informations sur le type du message, ce qui permet au serveur de savoir comment gérer les données. Par exemple si le message a pour but d'inscrire un utilisateur (SIGNUP), ou de connecter un utilisateur (LOGIN), ou encore d'enregistrer un message d'un salon spécifique (Général)

2. Enregistrement dans la base de données

Une fois le message reçu, le serveur peut l'enregistrer dans une base de données. Cela permet de conserver un historique des messages et peut être utile pour des fonctionnalités comme la récupération de l'historique des chats chez le client.

3. Affichage chez le client via envoi des données du serveur

Après avoir reçu et traité le message, le serveur l'envoie aux autres clients connectés. Ces clients peuvent alors afficher le message dans leur interface utilisateur. Cela permet une communication en temps réel entre les clients via le serveur.



III) Connexion

1. Enregistrement des connexions

Lorsqu'un utilisateur se connecte, le serveur l'enregistre dans un dictionnaire. La clé est généralement le socket client et la valeur peut être des informations sur l'utilisateur, comme son nom d'utilisateur. Cela permet au serveur de garder une trace de tous les utilisateurs connectés.

2. Déconnexions des clients

Lorsqu'un client se déconnecte, le serveur le supprime du dictionnaire des utilisateurs connectés. Le serveur a également avoir une liste des connexions actives, et le socket client est supprimé de cette liste lors de la déconnexion. Cela permet de libérer les ressources et de garder à jour la liste des utilisateurs connectés.

3. Login côté serveur

Le serveur a une classe **Login()** qui gère le processus de connexion. Cette classe possède des méthodes pour vérifier les identifiants de l'utilisateur et le mot de passe inscrits en dur.



IV) Inscription

1. Inscription des utilisateurs

Lorsqu'un utilisateur souhaite s'inscrire, il envoie un message au serveur avec le préfixe "SIGNUP". Ce message contient des informations telles que le nom d'utilisateur et le mot de passe souhaité.

Le serveur reçoit ce message et appelle la fonction **create_user()** pour créer un nouvel utilisateur. Cette fonction vérifie si les conditions d'inscription sont respectées (par exemple, si le nom d'utilisateur n'est pas déjà pris) et, si c'est le cas, elle inscrit l'utilisateur dans la base de données.

2. Envoi des nouveaux utilisateurs aux clients

Lorsqu'un nouvel utilisateur est créé, le serveur peut envoyer un message à tous les autres clients pour les informer de cette nouvelle connexion. Ce message aura une entête spéciale, comme "NEW_USER", pour indiquer qu'il s'agit d'un nouvel utilisateur.

3. Inscription côté serveur (classe Sign_up)

Le serveur possède une classe Sign_up() qui gère le processus d'inscription. Cette classe a des méthodes permettant l'inscription d'un super-utilisateur. Elle vérifie également les conditions d'inscription, et inscrit les nouveaux super-utilisateurs dans la base de données.

V) Profil utilisateur

1. Envoi des informations du profil

Lorsqu'un utilisateur demande à voir son profil sur l'interface graphique, le serveur a envoyé préalablement lors de sa connexion les informations du profil de l'utilisateur au client via la fonction **profil()**. Ces informations sont préfixées par "PROFIL" pour indiquer qu'il s'agit d'informations de profil.

2. Fonctionnement de la fonction **profil** et **user_profil** pour envoyer les données profil au client

La fonction **user_profil()** peut être utilisée pour récupérer les informations de profil d'un utilisateur à partir de la base de données. Ces informations peuvent ensuite être envoyées au client.

3. Modification du profil

Lorsqu'un utilisateur souhaite modifier son profil, il peut envoyer une demande de modification au serveur. Le serveur peut alors utiliser la fonction **update_profil()** pour mettre à jour les informations de profil de l'utilisateur dans la base de données.

Cette fonction prend les nouvelles informations de profil, soit la description, soit la photo de profil, les valide, puis met à jour la base de données.



VI) Demandes

1. Envoi de la liste des demandes

La fonction **demande()** récupère la liste des demandes depuis la base de données en attente d'un utilisateur spécifique, que le client peut voir les ayant envoyés.

2. Envoi de la liste des notifications

La fonction **notifications()** semble être utilisée pour récupérer la liste des notifications pour un utilisateur spécifique, que le client peut consulter et gérer.

3. Gestion des demandes de salon

En ce qui concerne la gestion des salons, les fonctions telles que `demande_salon`, `checkroom2`, `new_salon`, `send_new_salon`, et `send_demande_salon` sont responsables de la gestion des demandes de création de nouveaux salons de discussion.

Par exemple, `demande_salon` permet de recevoir des demandes de création de nouveaux salons, tandis que `new_salon` est utilisée pour créer effectivement ces nouveaux salons, et les fonctions de notification `send_new_salon` et `send_demande_salon` informent les autres utilisateurs de la création du nouveau salon.

4. Gestion des demandes admin

Concernant les demandes administratives, les fonctions `demande_admin`, `check_admin`, et `send_demande_admin` semblent jouer un rôle dans la gestion des demandes liées aux droits d'administration.

Par exemple, `demande_admin` est utilisée pour recevoir des demandes des droits administrateurs, `check_admin` vérifie si un utilisateur possède des droits administratifs, et `send_demande_admin` informe les autres super-utilisateurs de la demande d'administration.

5. Gestion des demandes d'amis ou friends

En ce qui concerne les demandes d'amis, les fonctions telles que `demande_ami`, `check_user_exists`, `check_ami`, `send_demande_ami`, et `check_demande_friend` sont responsables de la gestion des demandes d'amitié entre utilisateurs.

Ces fonctions permettent de recevoir des demandes d'ami, de vérifier si les utilisateurs sont déjà amis, d'informer les utilisateurs concernés des nouvelles demandes, et de vérifier l'existence de demandes d'amis en attente pour ne pas créer de doublons.

6. Validation des demandes

La fonction **`check_salon_accept()`** est utilisée pour vérifier si l'utilisateur a accepté une demande de création de salon. Elle joue un rôle crucial dans la gestion des demandes de salons en permettant de confirmer si l'utilisateur a approuvé la création d'un nouveau salon.

La fonction **`send_new_salon()`** a pour objectif d'informer tous les utilisateurs concernés de la création d'un nouveau salon. Elle intervient après que l'utilisateur ait accepté la demande de création de salon et joue un rôle de notification au sein de la plateforme.

La fonction **`new_salon2()`** est responsable de la création effective du nouveau salon. Elle prend en compte les informations de l'utilisateur initiateur de la demande ainsi que les détails du salon à créer.

Les fonctions **`accept()`** et **`refuse()`** sont utilisées pour respectivement approuver et décliner la demande de création de salon. Ces fonctions sont déclenchées en fonction de la décision de l'utilisateur, offrant ainsi un mécanisme interactif pour la gestion des demandes de salons.

En ce qui concerne la gestion des demandes administratives, la fonction **`check_admin_accept()`** est employée pour vérifier si l'utilisateur a accepté une demande d'administration. Elle s'inscrit dans le processus de validation des demandes liées aux droits administratifs.

La fonction **`send_new_admin()`** a pour rôle d'informer tous les utilisateurs de la plateforme de la nomination d'un nouvel administrateur. Elle intervient suite à l'acceptation d'une demande administrative par un utilisateur.

La fonction **add_admin()** est utilisée pour ajouter un utilisateur à la liste des administrateurs. Elle est déclenchée une fois que l'utilisateur a accepté une demande d'administration, lui conférant ainsi des droits spécifiques au sein de la plateforme.

Les fonctions **accept()** et **refuse()** sont également employées dans le contexte de la gestion des demandes administratives, permettant à l'utilisateur d'approuver ou de refuser une demande d'administration en fonction de sa décision.

En ce qui concerne les demandes d'amitié, la fonction **check_ami_accept()** est utilisée pour vérifier si l'utilisateur a accepté une demande d'amitié. Elle intervient dans le processus de confirmation des relations d'amitié entre utilisateurs.

La fonction **send_new_friend()** a pour objectif d'informer les utilisateurs pertinents de la demande d'amitié. Elle est déclenchée après l'acceptation d'une demande d'amitié par un utilisateur, permettant ainsi de notifier les parties concernées.

La fonction **new_friend()** est utilisée pour ajouter un nouvel ami à la liste d'amis de l'utilisateur. Elle intervient après que l'utilisateur ait accepté une demande d'amitié, consolidant ainsi la connexion entre les deux utilisateurs.

La fonction **delete_demande()** joue un rôle similaire aux contextes précédents en retirant la demande d'amitié de la liste des demandes en attente après qu'une décision a été prise par l'utilisateur.

Les fonctions **accept_friend()** et **refuse_friend()** sont employées pour accepter ou refuser une demande d'amitié, offrant à l'utilisateur un contrôle sur la construction de ses relations au sein de la plateforme.

La fonction **delete_demande()** est utilisée pour chaque type de demande après gestion. Elle permet de retirer la demande de la base de données. En effet il est essentiel de nettoyer la file d'attente des demandes pour maintenir la clarté et l'efficacité du système, et éviter les doublons.

Ces fonctions créent des fonctions de type « réponse » qui seront affichées dans les notifications des utilisateurs concernés

VII) Amis

1. Envoi de la liste d'amis aux clients

La fonction `friends` est utilisée pour récupérer la liste d'amis d'un utilisateur spécifique à partir de la base de données. Cette liste peut ensuite être envoyée au client lorsqu'il se connecte, lui permettant de voir qui sont ses amis.

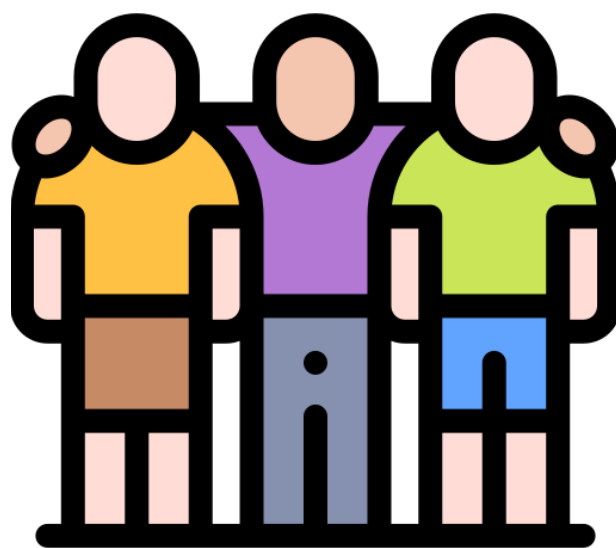
2. Ajout d'ami

La fonction `send_new_friend()` est utilisée pour envoyer une notification à un utilisateur qu'une nouvelle amitié a été établie. Cette notification est préfixée par "NEW_FRIEND" pour indiquer qu'il s'agit d'une nouvelle amitié.

La fonction `new_friend()` est probablement utilisée pour ajouter la nouvelle amitié à la base de données des deux utilisateurs concernés.

3. Suppression d'amitié

La fonction `delete_friend()` est utilisée pour supprimer une amitié de la base de données. Un message avec le préfixe "DELETE_FRIEND" est envoyé aux clients concernés pour leur indiquer qu'ils doivent mettre à jour leur liste d'amis.



VIII) Messages privés

1. Envoi de l'ensemble des messages privés lors de la connexion d'un utilisateur via la fonction **private()**

La fonction **private()** est utilisée pour récupérer tous les messages privés d'un utilisateur spécifique à partir de la base de données. Ces messages sont envoyés à l'utilisateur lorsqu'il se connecte, lui permettant de voir tous ses messages privés précédents.

2. Réception d'un message privé

Si un message entrant a le préfixe "PRIVATE", cela signifie qu'il s'agit d'un message privé, et le contenu du message sera donc ensuite redirigé vers les fonctions gérant les messages privés.

La fonction **private_message()** est utilisée pour enregistrer le message privé dans la base de données avec les utilisateurs concernés, et ainsi la fonction **send_private()** sert à envoyer le message privé aux utilisateurs concernés.



IX) Commandes

1. /kick et /ban

Après vérification de si le message fait partie du tuple « cmd » contenant la liste des commandes, et de si l'utilisateur a les droits de sanction avec **check_admin()**, le programme vérifiera également si l'utilisateur a déjà une sanction avec la fonction **check_sanction()**. En effet, il est impossible de mettre plusieurs sanctions sur un utilisateur pour des raisons pratiques.

La commande kick permet d'exclure un utilisateur temporairement. Dans cette commande on indiquera la durée du bannissement qui se base sur la base actuelle (par exemple 30 SECOND, 10 MINUTE, 3 HOUR...). La date de fin est ensuite automatiquement enregistrée dans la base de données sur ce principe.

La commande ban exclut définitivement un utilisateur. Ce dernier ne pourra en aucun cas se reconnecter. En effet dans la base de données la valeur de la date de fin sera nulle concernant cette sanction.

Ces commandes sont utilisables avec différents arguments. On peut sanctionner un utilisateur via son adresse ip (argument -a), ou via son identifiant (argument -p).

2. Unban et /unban

Quand l'utilisateur va tenter de se connecter, si il a une sanction la fonction **sanction_type()** va vérifier si la sanction est un « KICK ». Dans ce cas il va vérifier si la date de fin de la sanction est dépassée, dans ce cas la fonction **unban()** supprimer la sanction de l'utilisateur et le connecter.

Si un super-utilisateur effectue la commande /unban suivit de l'identifiant de l'utilisateur sanctionné, la fonction **unban_cmd()** est utilisée pour lever une sanction attribuée à un utilisateur. Elle met ensuite à jour la base de données concernant la sanction dans la table « user » et la table « sanction » pour retirer la sanction.

3. /get-ip

Cette commande, lorsqu'elle est exécutée par un super-utilisateur, permet de récupérer l'adresse IP associée à un utilisateur spécifique en saisissant son identifiant. Elle renvoie également le code unique correspondant à cet utilisateur en recherchant ces informations dans un dictionnaire partagé qui est indexé de manière identique.

4. /stop

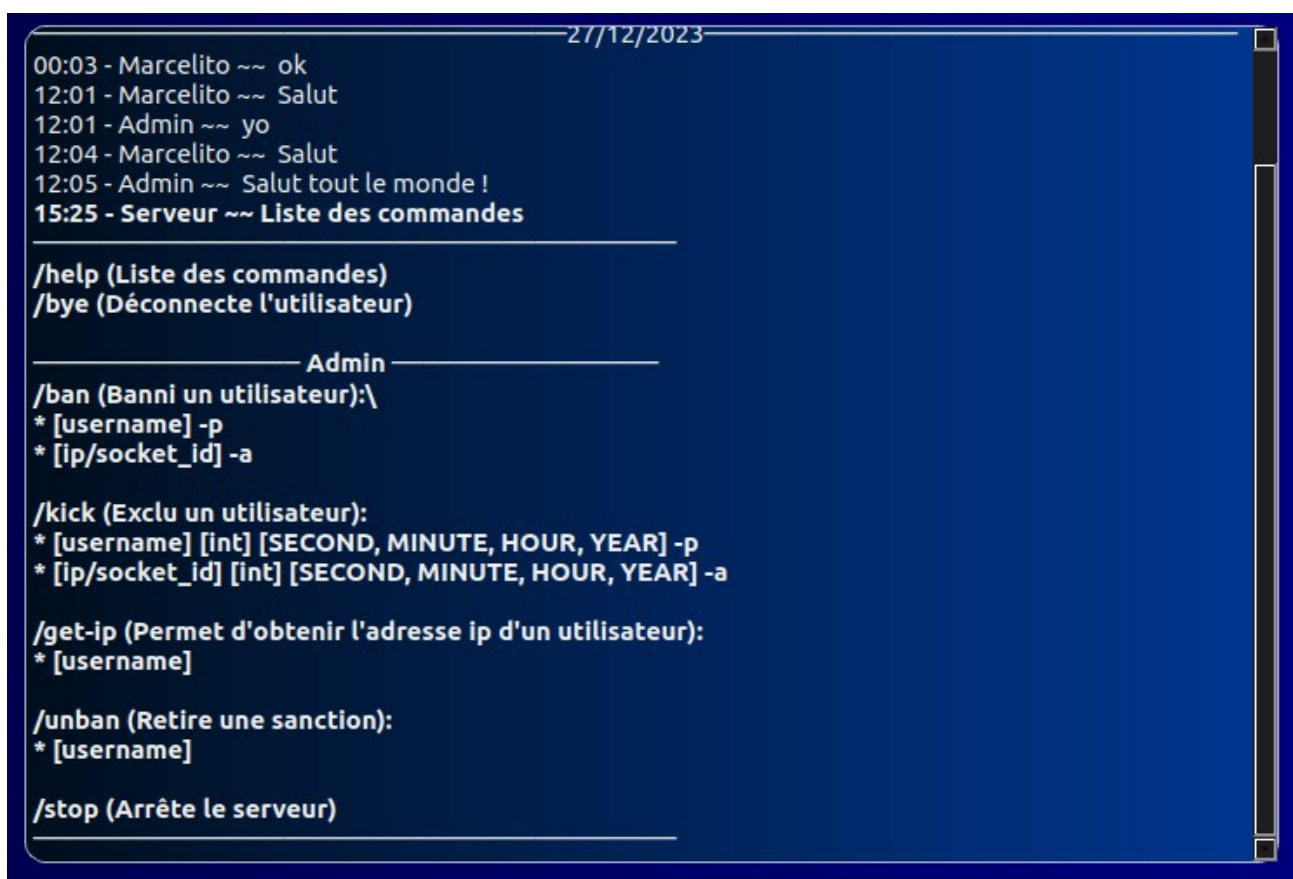
Permet à super-utilisateur d'arrêter le serveur dans les 10 secondes après l'exécution de la commande. Le chat est néanmoins toujours fonctionnel durant ce délai.

5. /bye

Permet à un utilisateur de se déconnecter. Cela ferme également le programme de son côté, et sa déconnexion sera affichée dans les logs.

6. /help

Envoie un message destiné à l'utilisateur ayant effectué la commande, avec la liste de toutes les commandes existantes et leur description. Le contenu ne sera pas le même en fonction du grade de l'utilisateur l'ayant exécutée.



```
27/12/2023
00:03 - Marcelito ~~ ok
12:01 - Marcelito ~~ Salut
12:01 - Admin ~~ yo
12:04 - Marcelito ~~ Salut
12:05 - Admin ~~ Salut tout le monde !
15:25 - Serveur ~~ Liste des commandes

/help (Liste des commandes)
/bye (Déconnecte l'utilisateur)

----- Admin -----
/ban (Banni un utilisateur):\
* [username] -p
* [ip/socket_id] -a

/kick (Exclu un utilisateur):
* [username] [int] [SECOND, MINUTE, HOUR, YEAR] -p
* [ip/socket_id] [int] [SECOND, MINUTE, HOUR, YEAR] -a

/get-ip (Permet d'obtenir l'adresse ip d'un utilisateur):
* [username]

/unban (Retire une sanction):
* [username]

/stop (Arrête le serveur)
```

Annexe : Codes d'erreur

- **A** : Succès lors de l'inscription
- **B** : Mot de passe incorrect (connexion)
- **C** : Mots de passe différents (inscription)
- **D** : Nom d'utilisateur non unique ou interdits (inscription)
- **E** : Nom d'utilisateur incorrect ou interdit (connexion)
- **F** : Format de mail non respecté (inscription)
- **G** : Caractères non autorisés (inscription)