Documentation Serveur

Application de chat 'CHandler'

Rédigé par O.Pichot
RT231 UHA

SOMMAIRE

1. Introduction

a. Synopsis de la documentation

Bienvenue sur la documentation du côté serveur, ce document à pour but de montrer les options disponibles sur l'application de chat 'CHandler'.

Ce document passera également en revu le fonctionnement de certains échanges entre le client et le serveur notamment durant l'envoi d'informations en plusieurs étapes par le biais de 'paquets'.

Nous verrons également comment effectuer certaines actions sur le serveur en tant qu'admin sous la forme de tutoriaux avec des exemples et prises d'écrans du logiciel.

Ce document n'a pas pour but de décrire avec grande précision le fonctionnement des classes, méthodes, et fonctions en tout genre que le programme utilise.
Pour ceci, nous vous invitons à vous tourner vers les documentations techniques mises à votre disposition sous la forme de documents web interactifs*.

Ces documentations techniques permettent notamment la recherche de modules, fonctions, classes, etc.. par des mots-clefs, la visualisation du code source et la description exhaustive du fonctionnement de chaque partie du code.

Sur ce, ne perdons pas plus de temps et commençons avec le premier point de cette documentation.

**Vous pouvez retrouver ces documents à l'adresse suivante :*
https://github.com/Wewenito/SAE302/tree/main/DOCUMENTS/Documentations%20Docstring

2. Base de données de CHandler

a. Organisation de la base de données

Ci-dessous un schéma représentatif de la base de données utilisée par l'application :

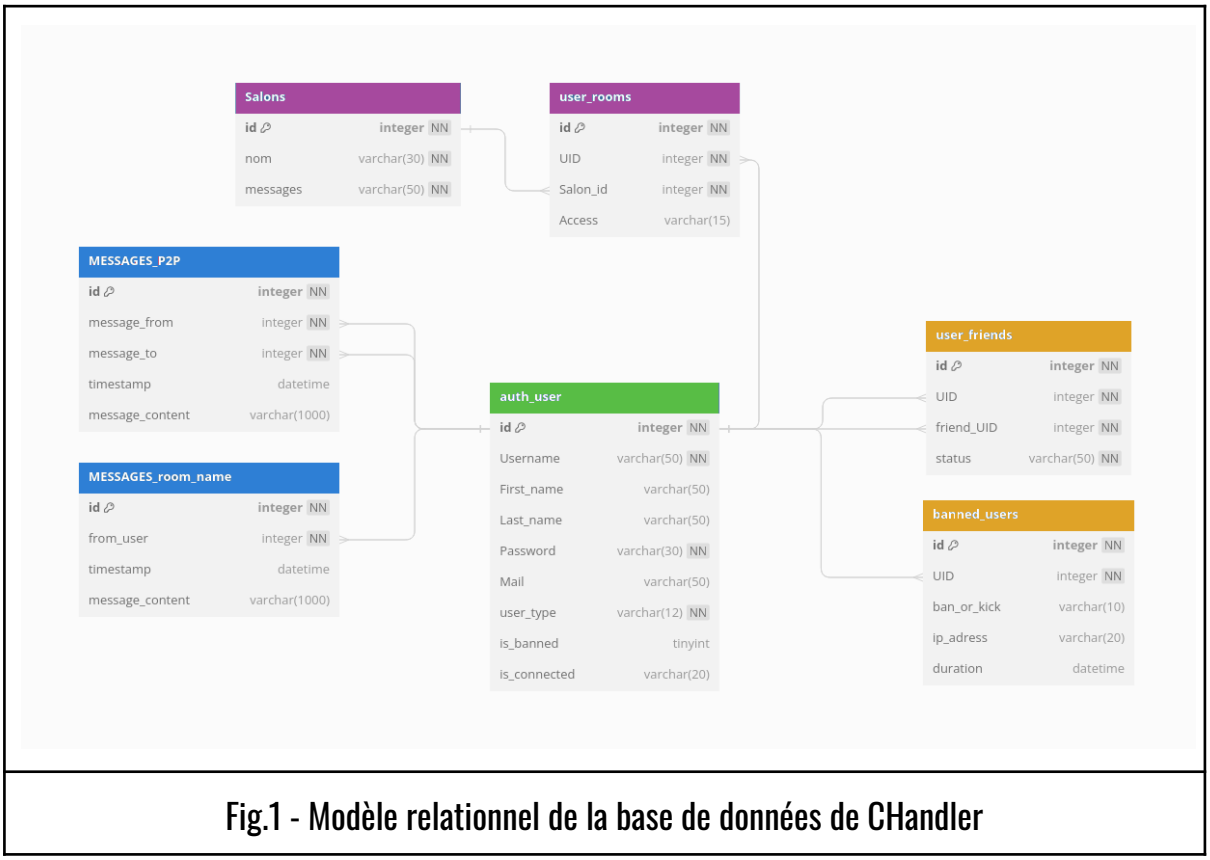


Fig.1 - Modèle relationnel de la base de données de CHandler

**Le document est disponible sur le repository github ici pour l'afficher en plein écran*

Chaque table de la bdd appartient à une catégorie de données bien spécifique :

- **Informations principale d'un utilisateur (auth_user)**
- **Informations supplémentaires liées à l'utilisateur (user_friends & banned_users)**
- **Informations relatives aux salons (Salons & user_rooms)**
- **Informations relatives aux messages (MESSAGES_P2P & MESSAGES_room_name)**

Vous trouverez ci-dessous un tableau décrivant les informations contenues au sein de chaque table ainsi qu'une description de celles-ci.

TABLE	COLONNE	COMMENTAIRE
auth_user	id	Id unique de chaque utilisateur
	Username	Nom d'utilisateur
	First_name	Prénom de l'utilisateur
	Last_name	Nom de l'utilisateur
	Password	Mot de passe de l'utilisateur
	Mail	Adresse mail de l'utilisateur
	user_type	Type d'utilisateur (admin ou employé)
	is_banned	L'utilisateur est-il banni ou non
	is_connected	L'utilisateur est-il connecté ou non
user_friends	UID	ID d'un utilisateur X
	friend_UID	ID de l'un des amis de l'utilisateur X
	status	Statut de la relation entre ces deux utilisateurs. (Amis, demande en cour, etc..)
banned_users	UID	ID de l'utilisateur concerné
	ban_or_kick	L'utilisateur est-il exclu ou banni
	ip_adress	Adresse IP associée à l'utilisateur si celui-ci est banni
	duration	Durée de l'exclusion si celui-ci est exclu
Salons	id	ID du salon
	nom	Nom du salon
	messages	Nom de la table contenant les messages de ce salon (ex : MESSAGES_general)

user_rooms	UID	ID de l'utilisateur en question
	Salon_id	ID du salon en question
	Access	L'utilisateur a-t-il accès au salon (YES, NO, PENDING, etc..)
MESSAGES_P2P	id	ID du message
	message_from	ID de l'utilisateur ayant envoyé le message
	message_to	ID de l'utilisateur ayant reçu le message
	timestamp	Heure d'envoi du message (heure & date)
	message_content	Contenu du message
MESSAGES_room_name	id	ID du message
	from_user	ID de l'utilisateur ayant envoyé le message
	timestamp	Heure d'envoi du message (heure & date)
	message_content	Contenu du message

Petite précision concernant les tables stockant les messages de salons :
Chaque salon possède sa propre table de stockage, chacune de ces tables est nommée selon la syntaxe suivante : MESSAGES_ + Nom de la table.

Nous avons donc, pour l'instant, 5 tables de stockages de messages de salons :

- MESSAGES_general
- MESSAGES_blabla
- MESSAGES_comptabilite
- MESSAGES_informatique
- MESSAGES_marketing

Ceci permet de séparer proprement les données de chaque salon et limiter les requêtes à une même table au même moment.

b. Accès à la base de données

Pour accéder à la base de données, un utilisateur est présent sur la machine servant l'application aux clients.

Celui-ci est créé sur le serveur Mariadb / MySQL (au choix) durant la procédure d'installation du serveur et possède les crédentiels suivant :

- Nom d'utilisateur : admin
- Mot de passe : admin

Seul cet utilisateur à accès à la base de données (SAE302) et c'est ce même utilisateur qui est utilisé au sein du fichier 'sql_Handler.py' pour créer un objet de connexion Mysql. Cet objet permet par la suite d'ajouter, supprimer, lire et modifier des données sur la bdd.

N.B : Pour des raisons de sécurité, nous vous conseillons fortement de modifier le mot de passe de l'utilisateur de la base de données. Il faudra alors également changer le mot de passe renseigné au sein du fichier "sql_Handler.py' dans la fonction 'establish_connection'.

Comme convenu dans le cahier des charges que vous nous aviez confié, seul le serveur de l'application peut avoir accès à la base de données, c'est pourquoi nous verrons dans la prochaine section de cette documentation comment se déroulent les échanges d'informations entre le serveur et le client.

c. Exemple de requête client/serveur pour obtenir des infos provenant de la BDD

> Un client souhaite obtenir l'historique des messages envoyés au sein d'un salon X :

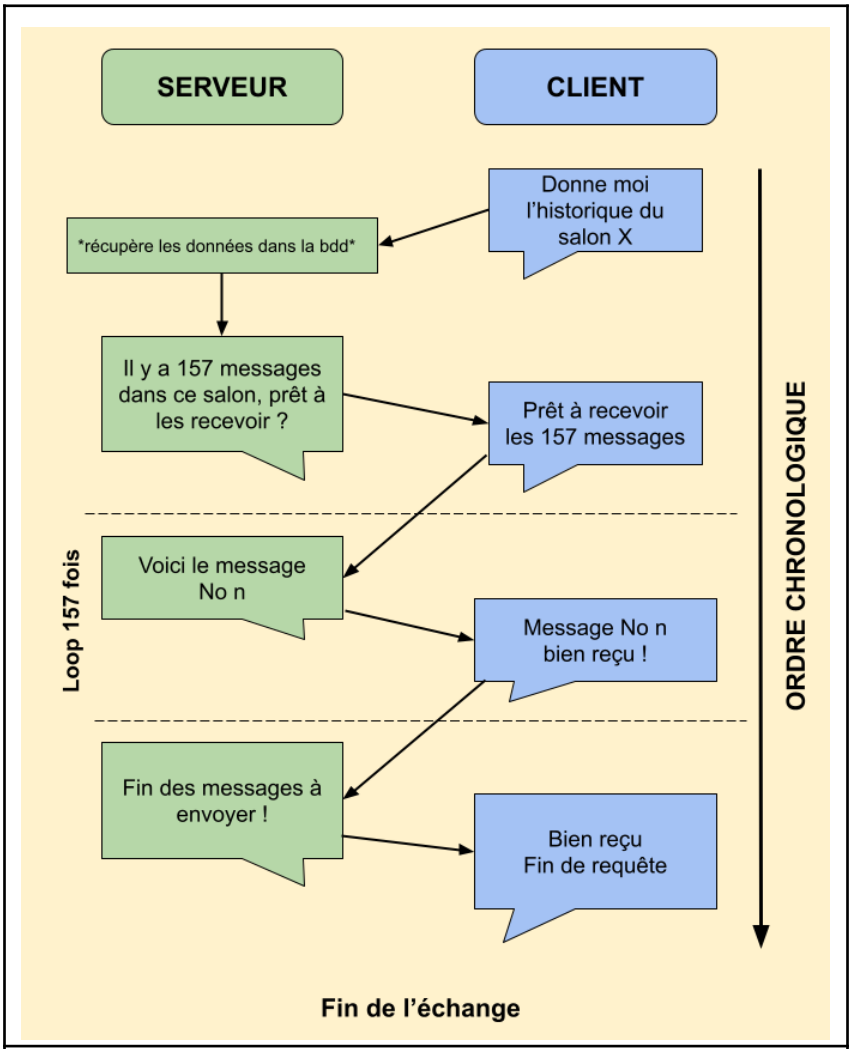


Fig.2 - Exemple d'échange client-serveur pour l'obtention de données

Lors de l'envoi de beaucoup de données, le serveur divise le tout en un nombre n de paquets puis les envoie un par un afin d'éviter la perte de données.

Dans l'exemple ci-dessus, chaque paquet est composé de la sorte :

"No de paquet | Nom de l'utilisateur | Date & Heure d'envoi | Contenu du message"

3. Organisation du code de l'application

L'application CHandler est divisée en plusieurs fichiers, ceux-ci sont au nombre de quatres et ont chacun leur propre utilité.
La division des fichiers du programme permet une meilleure lisibilité du code et une segmentation propre des tâches que nous allons voir ci-dessous.

main.py :

Ce fichier représente l'exécutable de l'application, il est la racine du programme et se charge de faire appel à chacun des autres fichiers lors de son exécution.
Il va notamment instancier une classe de serveur et lier celui-ci à un socket pour le rendre disponible sur le réseau.
Ce fichier va également gérer les interactions qu'un admin peut avoir avec le terminal serveur pour y envoyer des commandes et se connecter.

server.py :

Ce fichier contient la classe Server_handle, cette classe représente le serveur et toutes ses méthodes. Vous pouvez retrouver le détail complet de chacune de ses méthodes et variables au sein de la documentation technique du serveur ICI.

sql_Handler.py :

Ce fichier contient toutes les fonctions pouvant être appelées par l'objet serveur permettant des échanges avec la base de données. La seule condition pour appeler l'une de ces fonctions est d'initier un objet de connexion 'mydb' à passer en paramètre à ces fonctions.
Vous pouvez également retrouver le fonctionnement de ces fonctions au sein de la documentation technique.

sys_x.py :

Ce fichier gère les interactions visuelles du terminal, notamment : l'affichage de code d'erreurs et le nettoyage du terminal.

4. Utilisation du terminal serveur

a. Authentification

Lors du lancement du serveur, l'interaction avec celui-ci ne sera pas possible tant qu'un administrateur ne sera pas authentifié auprès de celui-ci :

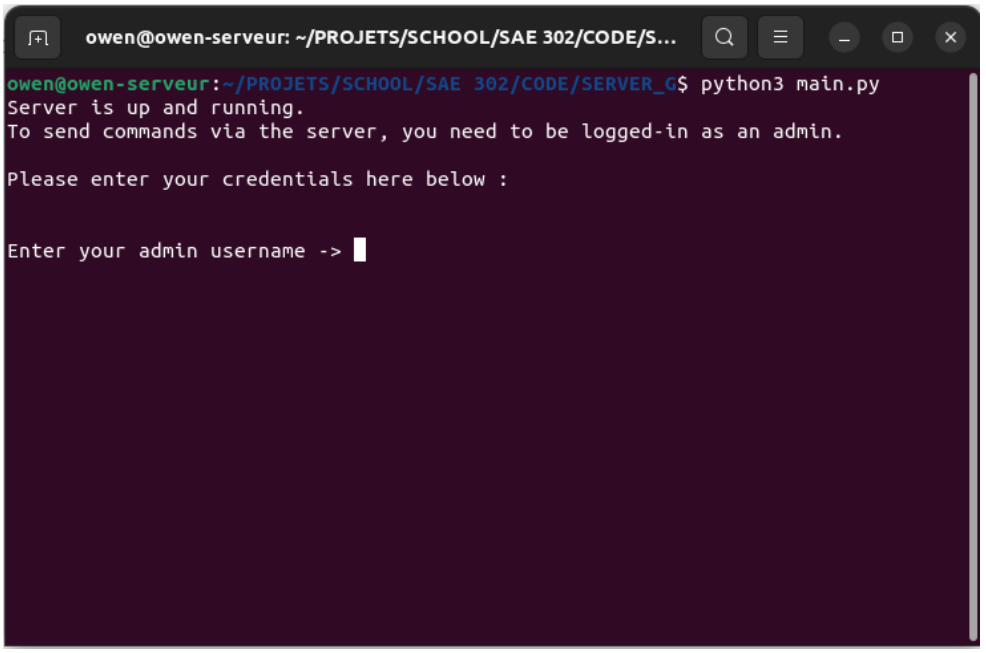


Fig.3 - Terminal du serveur après le lancement de 'main.py'

Vous pouvez voir ici (Fig.3) que le serveur est bien lancé et que celui-ci demande à l'utilisateur de se connecter sur un compte admin.
N.B : Même si personne ne se connecte sur le terminal, le serveur est d'ores et déjà opérationnel, les clients peuvent se connecter sans problème, cependant, il ne sera pas possible de taper quelconque commande sur le terminal tant qu'un admin n'est pas connecté.

L'application que nous vous avons fourni vient avec un fichier template de base de données, disponible ICI, cette base de données possède un utilisateur admin défaut dont les crédentiels sont "admin/admin".

/!\ Nous recommandons fortement de changer le mdp de cet admin après votre 1ère connexion ! /!\

Une fois l'utilisateur admin connecté sur le terminal, vous aurez alors accès à chacune des commandes proposées par l'application (nous verrons ces commandes plus en détail un peu plus loin dans ce document).

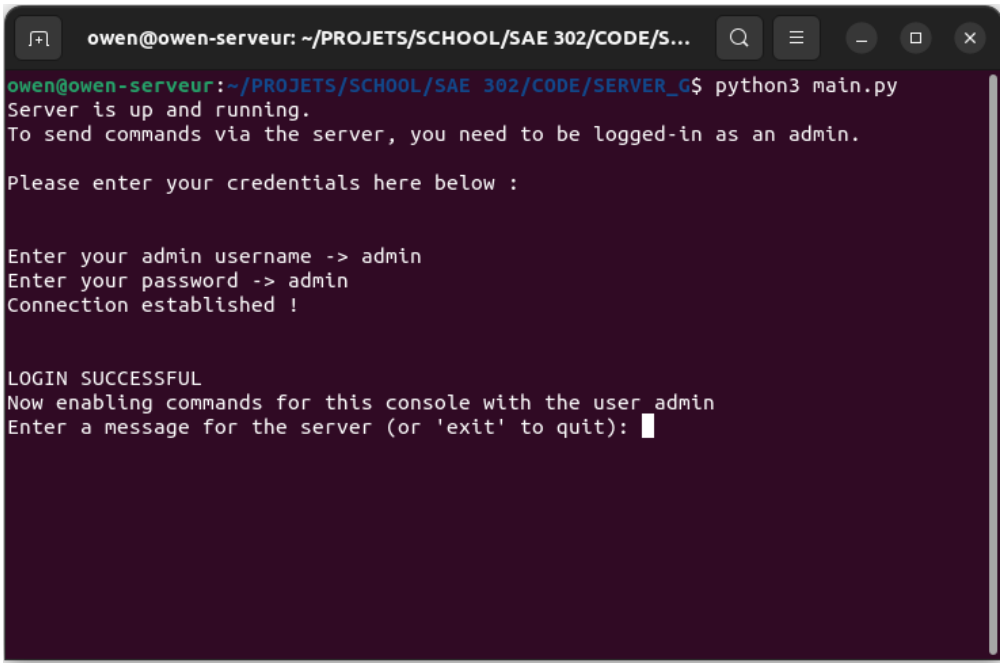


Fig.4 - Terminal du serveur après la connection d'un administrateur

Comme vous pouvez le voir ci-dessus, une fois l'administrateur connecté, une nouvelle boucle se lance (provenant de 'main.py'), celle-ci écoute constamment un potentiel input texte de la part de l'administrateur et analyse ensuite l'action à réaliser.

Nous allons maintenant voir la liste de ces fameuses commandes auxquelles l'admin à accès.

b. Commandes du serveur

L'application côté serveur met à disposition plusieurs commandes utilisables par l'admin, nous allons voir dans cette section chacune de ces commandes, leur syntaxe d'utilisation et le résultat attendu / l'objectif de celles-ci.

• Commande de broadcast :

DESCRIPTION :

Cette commande permet l'envoi d'un message à chacun des clients actuellement connecté au serveur. Ceci est fait en effectuant une loop au sein des objets de connexion sauvegarder dans le serveur sous la variable 'current threads'.

SYNTAXE :

> /broadcast <message>

PARAMÈTRES :

message → Message à envoyer aux clients, pas de limite de taille mais il est conseillé de ne pas dépasser les 1000 caractères.

EXEMPLE D'UTILISATION & RÉSULTAT :

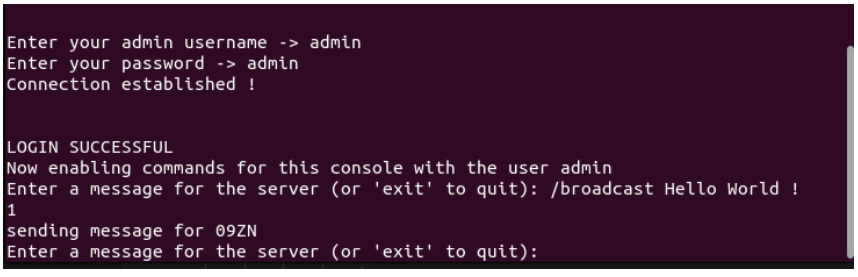
SERVEUR	
CLIENT	

Fig.5 - Envoie d'un message broadcast depuis le serveur & réception d'un client.

- ## Affichage des clients :

DESCRIPTION :

Cette commande permet d'afficher chacun des clients actuellement connectés au serveur, si un client est authentifié, alors ses informations d'utilisateur seront également affichées, sinon, seul son ID unique de connexion (4 caractères) sera affiché.

SYNTAXE :

> /show_clients

PARAMÈTRES :

Aucun

EXEMPLE D'UTILISATION & RÉSULTAT :

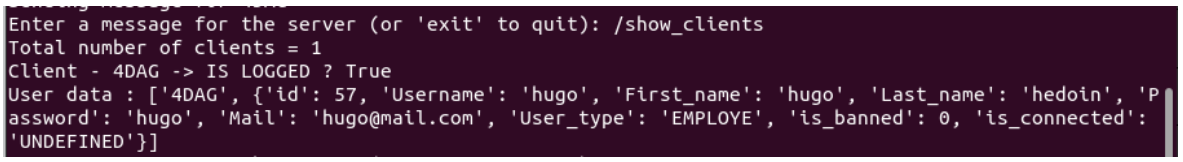


Fig.6 - Utilisation de la commande /show_clients sur le terminal serveur

On peut voir dans l'exemple ci-dessus que seul un client est connecté au serveur :

"Total number of clients = 1"

Que celui-ci est bien authentifié auprès du serveur :

"Client - 4DAG → IS LOGGED ? True"

Et on retrouve enfin ses informations d'utilisateurs sur les lignes restantes.

N.B : Le code '4DAG' représente un identifiant unique de connexion. Chaque client se connectant au serveur s'en voit assigné un et celui-ci sert au serveur pour faire le lien entre les informations d'un utilisateur et son objet de connexion au socket attitré.

• Modifier le mot de passe d'un utilisateur :

DESCRIPTION :

Cette commande permet de modifier le mot de passe d'un utilisateur. Une fois la commande entrée, il suffit de taper une première fois le nouveau mot de passe à assigner à l'utilisateur puis de confirmer ce nouveau mot de passe.

SYNTAXE :

> /change_password <nom_utilisateur>
> Entrer le nouveau mot de passe
> Confirmer le nouveau mot de passe

PARAMÈTRE :

nom_utilisateur → Ce paramètre spécifie le nom de l'utilisateur dont on souhaite changer le mot de passe.

EXEMPLE D'UTILISATION & RÉSULTAT :

```
Enter a message for the server (or 'exit' to quit): /change_password admin
Please enter the new password for this user -> azerty
Confirm the password -> azerty
User password has been succesfully changed !
Enter a message for the server (or 'exit' to quit):
```

Fig.7 - Utilisation de la commande /change_password sur le terminal serveur

Dans l'exemple ci-dessus (Fig.7), on change le mot de passe de l'utilisateur 'admin', le nouveau mot de passe assigné à cet utilisateur sera désormais 'azerty'.

```
mysql> select Username, Password from auth_user where Username = 'admin';
+----------+----------+
| Username | Password |
+----------+----------+
| admin    | azerty   |
+----------+----------+
1 row in set (0,00 sec)
```

Fig.8 - Affichage du mot de passe modifié au sein de la base de données

Si l'on vérifie au sein de la base de données les informations de l'utilisateur 'admin', on constate que son mot de passe a bel et bien été modifié en 'azerty'.

- ## Changer le statut d'un utilisateur :

DESCRIPTION :

Cette commande permet de passer un utilisateur du statut lambda (Employé) à un statut 'admin'.

⚠ **Attention, cette action n'est pas à prendre à la légère, un administrateur à de grands pouvoirs sur le fonctionnement de l'application** ⚠

SYNTAXE :

> /make_admin <nom_utilisateur>

PARAMÈTRE :

nom_utilisateur → Ce paramètre spécifie le nom de l'utilisateur que l'on souhaite passer en statut d'administrateur.

EXEMPLE D'UTILISATION & RÉSULTAT :

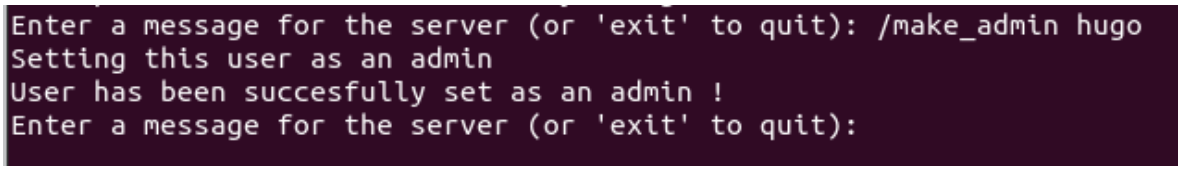


Fig.9 - Utilisation de la commande /make_admin sur le terminal serveur

Dans l'exemple ci-dessus (Fig.9), on passe l'utilisateur 'hugo' au statut d'administrateur.

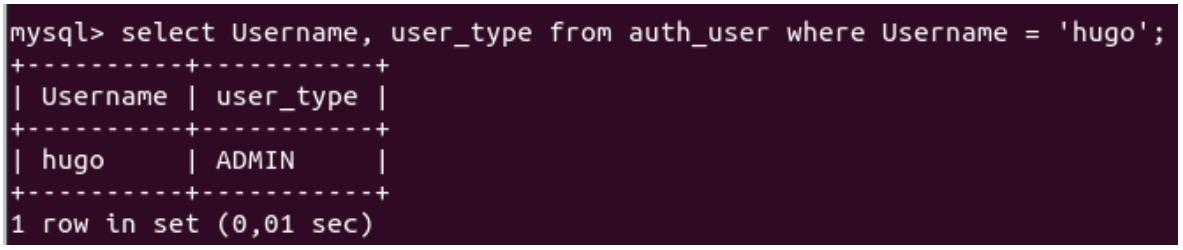


Fig.10 - Affichage du statut utilisateur modifié au sein de la base de données

Si l'on vérifie au sein de la base de données le statut de l'utilisateur 'hugo', on peut constater que l'utilisateur est désormais bel et bien un administrateur.

• Extinction du serveur :

DESCRIPTION :

Cette commande permet de couper entièrement le serveur, ce faisant, un message d'alerte sera envoyé à chacun des clients (via la commande /broadcast) et le drapeau d'arrêt sera alors levé. Pour plus d'information sur le fonctionnement du drapeau et de la commande, voir la documentation technique accessible ICI.

SYNTAXE :

> /kill

PARAMÈTRE :

Aucun

EXEMPLE D'UTILISATION & RÉSULTAT :

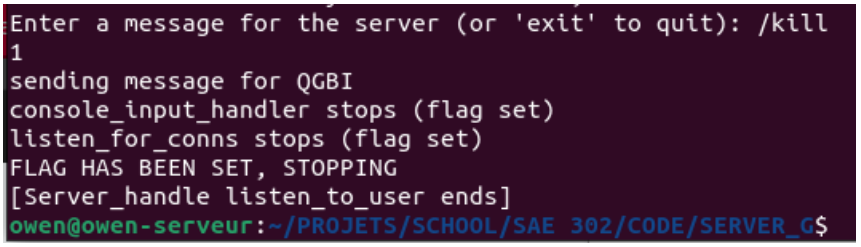
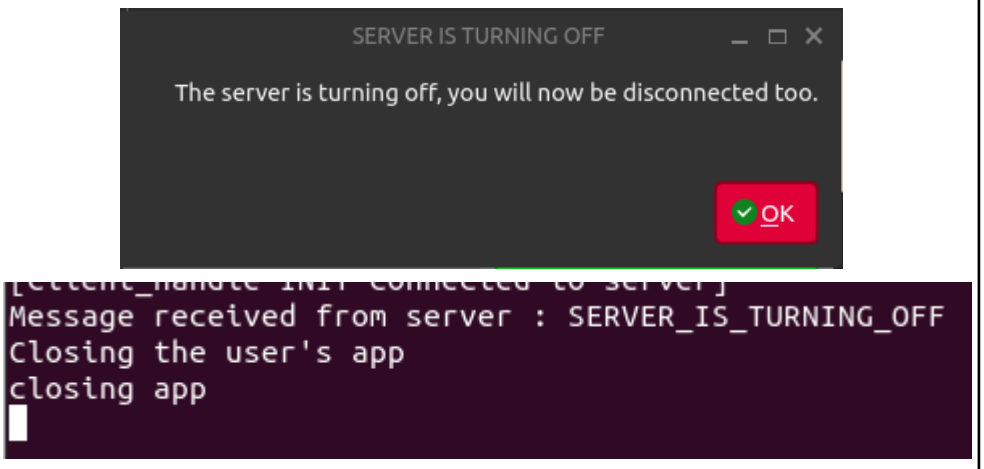
SERVEUR	
CLIENT	

Fig.11 - Résultat de la commande /kill sur le terminal serveur & client ainsi que l'UI du client

On constate que la commande /kill a non seulement arrêté le serveur mais aussi envoyé un message sur le terminal client ce qui génère par la suite un popup pour prévenir le client de la fermeture de l'application.

- ## Exclure un utilisateur :

DESCRIPTION :

Cette commande permet d'exclure temporairement un utilisateur, la durée de l'exclusion est à spécifier en secondes. Si l'utilisateur est connecté lors de son exclusion, alors un popup lui sera affiché et son application sera ensuite fermée afin de lui couper l'accès à l'application. Il ne pourra ensuite plus se reconnecter tant que la date de fin d'exclusion à été passée.

SYNTAXE :

> /kick <nom_utilisateur> <durée_exclusion>

PARAMÈTRE :

nom_utilisateur → Paramètre spécifiant le nom de l'utilisateur à exclure.
durée_exclusion → Paramètre spécifiant la durée de l'exclusion en

EXEMPLE D'UTILISATION & RÉSULTAT :

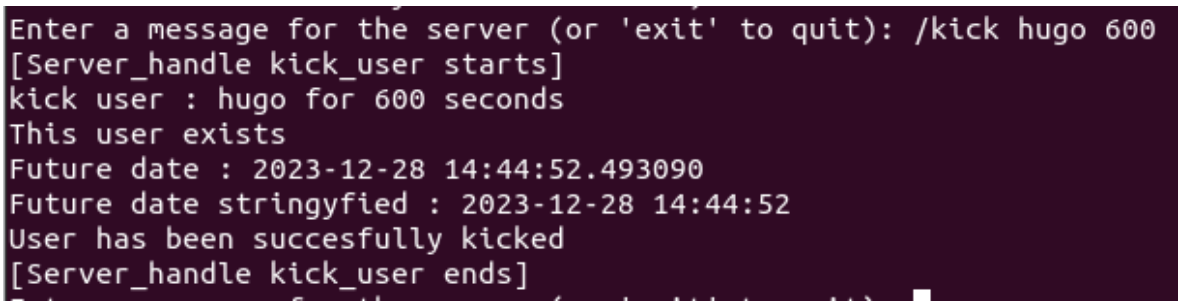


Fig.12 - Exécution de la commande /kick sur l'utilisateur 'hugo'

Dans cet exemple, on peut voir que la procédure d'exclusion d'un utilisateur s'effectue en plusieurs temps, le serveur vérifie notamment si l'utilisateur que l'on souhaite exclure existe ou non dans la base de données (et si celui-ci n'est pas déjà banni / exclu).

Ensuite, le serveur prépare la date et l'heure à laquelle l'utilisateur pourra de nouveau rejoindre le serveur, étant donné qu'ici on exclut l'utilisateur pendant 600 secondes (10 minutes) à 14h34, il pourra alors rejoindre de nouveau le serveur à 14h44.

Voyons maintenant le résultat de cette exclusion côté client, celui-ci étant connecté au moment de l'exclusion.

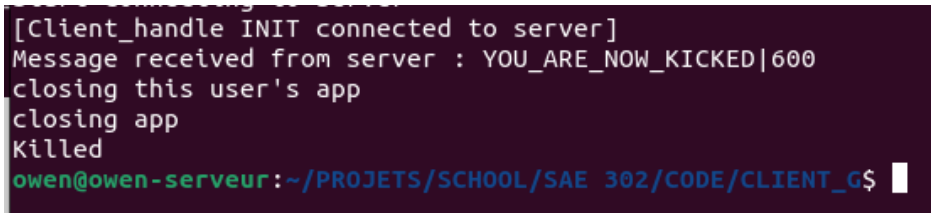
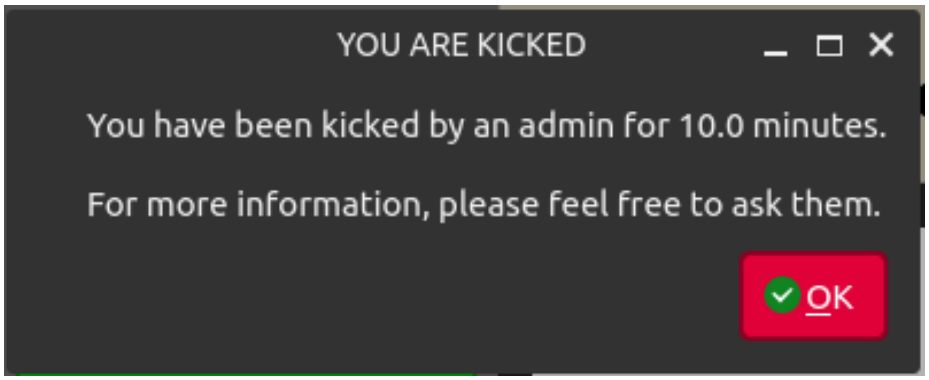
Terminal client	
Interface client	

Fig.13 - Résultat d'une exclusion côté client (terminal et UI)

On constate que le client à bien été prévenu de son exclusion, une fois la fenêtre popup fermée, l'application est alors complètement terminée et le client ne pourra plus se reconnecter avant la fin de sa période d'exclusion (colonne 'duration' (Fig.14)).

```
mysql> select id, Username, is_banned from auth_user where Username = 'hugo';
+----+----------+-----------+
| id | Username | is_banned |
+----+----------+-----------+
| 57 | hugo     |         1 |
+----+----------+-----------+
1 row in set (0,00 sec)

mysql> select * from banned_users where UID = 57;
+----+-----+-------------+-----------+---------------------+
| id | UID | ban_or_kick | ip_adress | duration            |
+----+-----+-------------+-----------+---------------------+
| 13 |  57 | KICK        | NULL      | 2023-12-28 14:44:52 |
+----+-----+-------------+-----------+---------------------+
1 row in set (0,00 sec)
```

Fig.14 - Résultat de l'exclusion de l'utilisateur au sein de la base de données.

N.B : L'adresse IP n'est pas associée à l'exclusion, ceci est utilisé dans le cas d'un ban.

- ## Bannir un utilisateur :

DESCRIPTION :

Cette commande permet de bannir un utilisateur, une fois l'utilisateur banni, celui-ci ne sera plus en mesure de se connecter à l'application. A la différence d'une exclusion, le bannissement est définitif et l'adresse IP de l'utilisateur sera associée au ban, soit immédiatement si l'utilisateur est connecté au moment du ban, soit lorsqu'il essayera de se reconnecter pour la première fois après avoir été banni.

SYNTAXE :

> /ban <nom_utilisateur>

PARAMÈTRE :

nom_utilisateur → Nom de l'utilisateur que l'on souhaite bannir de l'application.

EXEMPLE D'UTILISATION & RÉSULTAT :

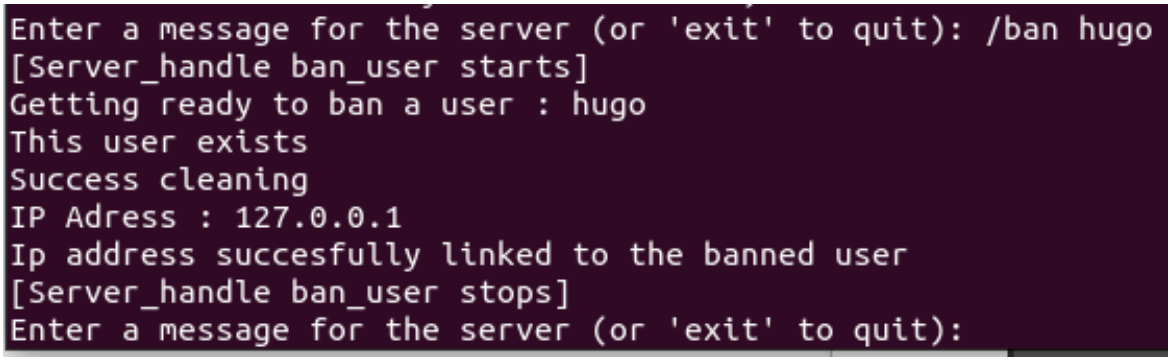


Fig.15 - Exécution de la commande /ban sur l'utilisateur 'hugo'

Dans l'exemple ci-dessus (Fig.15) un administrateur décide de bannir un utilisateur (hugo). On peut voir que le serveur commence d'abord par vérifier que cet utilisateur existe bien. Puis, le serveur supprime ses accès aux salons, passe son compte au statut banni et, puisque l'utilisateur était connecté lors du ban, fini par lié l'adresse IP de la machine que hugo utilise au ban de celui-ci (voir Fig.17).

***N.B** : L'adresse IP de l'exemple est une adresse locale car ce test a été effectué sur une seule et même machine via l'adresse de loopback, en situation réelle, l'adresse sera celle de la carte réseau connecté au réseau de l'entreprise.*

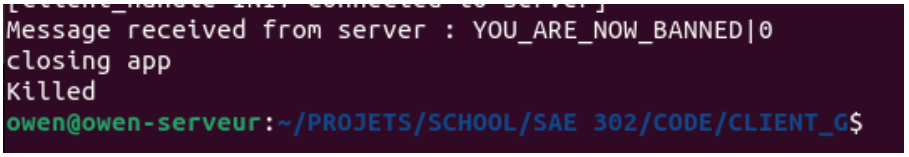
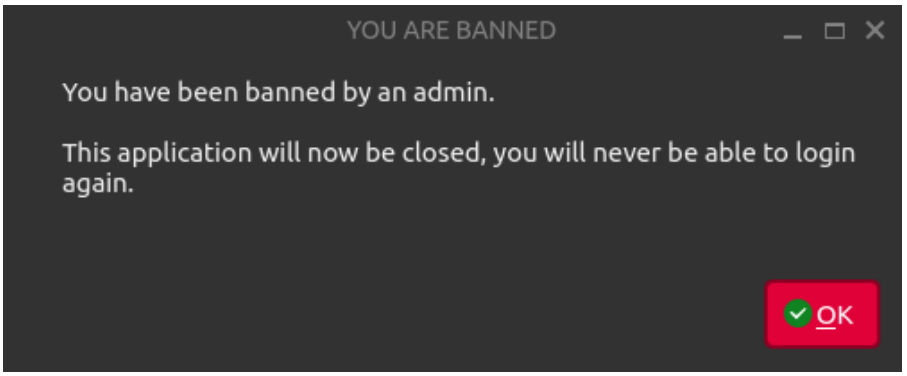
Terminal client	
Interface client	

Fig.16 - Résultat du ban coté client (terminal et UI)

On peut voir dans l'exemple ci-dessus (Fig.16) que l'utilisateur à été prévenu du ban à son encontre et qu'une fenêtre popup lui indique que l'accès à l'application lui sera désormais refusé.

```
mysql> select id, Username, is_banned from auth_user where Username = 'hugo';
+----+----------+-----------+
| id | Username | is_banned |
+----+----------+-----------+
| 57 | hugo     |         1 |
+----+----------+-----------+
1 row in set (0,00 sec)

mysql> select * from banned_users where UID = 57;
+----+-----+-------------+-----------+----------+
| id | UID | ban_or_kick | ip_adress | duration |
+----+-----+-------------+-----------+----------+
| 14 |  57 | BAN         | 127.0.0.1 | NULL     |
+----+-----+-------------+-----------+----------+
1 row in set (0,00 sec)
```

Fig.17 - Résultat du ban de l'utilisateur au sein de la base de données

Notez que contrairement à une exclusion, la colonne 'ban_or_kick' à pour valeur 'BAN' et que l'adresse IP de la machine utilisée par l'utilisateur banni à été associée à lui. De plus, la colonne 'duration' est Null car l'utilisateur ne pourra jamais revenir sur l'application.

- ## Affichage des threads actifs du serveur :

DESCRIPTION :

Cette commande permet d'afficher la liste complète des threads actifs du serveur.

SYNTAXE :

> /threads

PARAMÈTRE :

Aucun

EXEMPLE D'UTILISATION & RÉSULTAT :

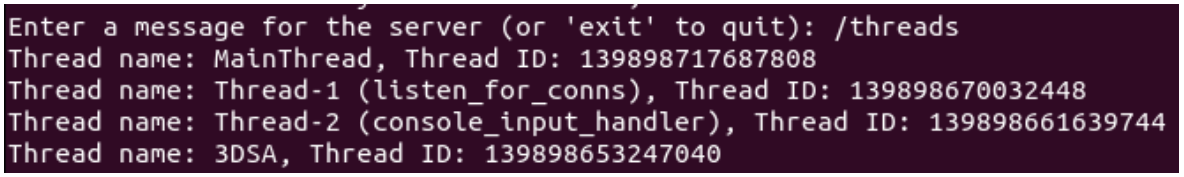


Fig.18 - Résultat de la commande /threads sur le terminal serveur

On peut voir que suite à l'exécution de la commande, l'entièreté des threads actif du serveur sont imprimés dans le terminal, passons en revue leur signification par rapport à leur nom :

- '**MainThread**' → Thread d'exécution à la source du programme (main.py)
- '**Thread-1**' → Thread d'écoute de potentielles nouvelles connexions clients.
- '**Thread-2**' → Thread de réception des commandes sur le terminal serveur
- '**3DSA**' → Thread d'écoute d'un client déjà connecté au serveur.

5. Conclusion

Ceci conclut la documentation de l'application côté serveur.

A l'avenir, il serait intéressant d'implémenter une interface graphique pour le serveur afin de rendre l'utilisation de celui-ci plus intuitive et accessible.

Nous profitons de cette section pour vous rappeler que le détail technique de chacune des fonctions abordées dans ce document est disponible ICI.

Nous vous remercions pour votre attention et espérons que cette documentation de l'utilisation du côté serveur de l'application CHandler aura été assez claire et concise à vos goûts.

Vous êtes bien sûr les bienvenus à contacter nos équipes en cas de questions supplémentaires sur le fonctionnement de l'application ou bien en cas de comportement anormal.