

SAE R.309 et R.302

Développer une Application Communicante

Zabchat

Projet réalisé par :

MARTIN Jean Christophe


Sommaire

Table des matières

I.	Installation du Serveur.....	3
1.	Installation pour Windows.....	3
2.	Installation pour Debian/Ubuntu	4
II.	Installation du Client.....	6
III.	Documentation Serveur	7
1.	Classe	7
2.	Fonctions.....	8
3.	Commandes.....	12
4.	Base de données	18
5.	Droits d'accès	21
IV.	Documentation Client	22
1.	Interface Utilisateur	22
2.	Fonctions.....	26
V.	Document Réponse.....	31
1.	Les limites du programme	31
2.	La Sécurité	31
3.	Confidentialité.....	31
4.	Maintenance	31
5.	Mots de la Rédaction	32

I. Installation du Serveur

1. Installation pour Windows

- 1) Installer Python pour Windows
- 2) Exécutez le fichier *client_install.bat*  `client_install.bat`
- 3) Installez MySQL serveur
- 4) Importer le fichier *zabchat.sql* sur le serveur MySQL
- 5) Lancez le script *server-bdd.py*

Optionnel : Pour aller plus loin, je vous propose d'utiliser phpMyAdmin afin de faciliter la modération du chat

2. Installation pour Debian/Ubuntu

1) Installer Python

Installation des dépendances

```
sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev libnss3-dev libssl-dev libreadline-dev libffi-dev libsqlite3-dev wget libbz2-dev
```

Installation de Python

```
sudo apt install python3 -y  
sudo apt install python3-pip -y
```

2) Exécutez les commandes suivantes afin d'installer les modules python

```
pip3 install mysql-connector-python  
pip3 install cryptocode  
pip3 install datetime
```

3) Installer Mysql

```
sudo apt update  
sudo apt upgrade  
sudo apt install https://dev.mysql.com/get/mysql-apt-config_0.8.29-1_all.deb
```

2 boites de dialogues vont s'ouvrir, sélectionnez OK

```
sudo apt update  
sudo apt install mysql-server  
sudo mysql_secure_installation
```

Entrez le nouveau mot de passe d'administration

En cas d'erreur voir la page : <https://www.nixcraft.com/t/mysql-failed-error-set-password-has-no-significance-for-user-root-localhost-as-the-authentication-method-used-doesnt-store-authentication-data-in-the-mysql-server-please-consider-using-alter-user/4233>

4) Clonez le répertoire GitHub du projet

```
gh repo clone JacquesCentue/TP-exception
```

Ou copier les fichiers dans le répertoire de travail

5) Déployez la base de données d'administration

```
cd ./TP-exception/SAE/Server  
mysql -u root -p < zabchat.sql
```

Note : il se peut que les paramètres de sécurité de mysql entraine une erreur dut au mot de passe de l'utilisateur « zabchat » crée afin d'administrer la base de données : <https://www.tecmint.com/fix-mysql-error-1819-hy000/>

6) Lancez le programme python

```
cd <répertoire ou se trouve le fichier server-bdd.py>  
python3 server-bdd.py
```

7) Installation terminée

II. Installation du Client

- 1) Installez Python 3
- 2) Copiez les fichiers clients
- 3) Exécutez le fichier client_install.bat
- 4) Modifiez le fichier client-bdd-gui.py afin de modifier l'adresse IP du serveur

ATTENTION : il faut modifier l'IP a 2 endroits dans le code :

- Dans la classe ***AuthWindow***
- Dans la classe ***ChatWindow***

Sous l'attribut *self.HOST*

- 5) Le client est prêt à l'emploi

Pour aller plus loin vous pouvez aussi transformer le fichier python en exécutable via ce programme : <https://pypi.org/project/auto-py-to-exe/>

Comme cela, le programme n'est plus consultable par l'utilisateur

III. Documentation Serveur

Le programme possède plusieurs modules permettant l'ajout de fonctionnalités :

- Socket : permet d'établir la communication entre le client distant et le serveur
- Threading : permet d'effectuer plusieurs fonctions en même temps
- mysql.connector : permet la connexion du serveur a la base de données pour administration
- cryptocode : permet de mettre en place un processus de chiffrement des messages entre le client et le serveur
- datetime : permet d'avoir la date et l'heure, utile lors du Timeout ou Kick
- hashlib : permet de hasher les mots de passe afin de ne pas les stocker tels quel dans la base de donnée et lors de l'authentification du root

1. Classe

Le programme du serveur est composé de 1 classe qui comporte 10 attributs :

- [self.HOST](#) : adresse IP du serveur
- [self.PORT](#) : port de connexion cette valeur doit être identique aux clients
- [self.serverstatus](#) : statut du serveur
- [self.db_connection](#) : ce sont les informations de connexion au serveur de base de données
- [self.clients](#) : liste de tous les clients connectés
- [self.ban](#) : liste des utilisateur/IP bannis
- [self.mdpRoot](#) : hash du mot de passe du root (par défaut : root)

Lors de l'appel du constructeur, les threads sont alors démarrés afin que le programme puisse faire plusieurs choses à la fois

2. Fonctions

Dans le programme du serveur, il y a 6 fonctions existantes :

1. Le constructeur

2. listen_connections

Cette fonction est démarrée dans un thread lors du démarrage du programme cela permet alors d'écouter si un nouveau client souhaite se connecter.

Si un client souhaite se connecter, alors la fonction va créer un thread qui s'occupera alors de gérer les clients en envoyant les arguments nécessaires tel que son adresse IP et son port.

3. handle_client

Cette fonction est appelée par le thread listen_connections il va d'abord effectuer des tests si l'utilisateur existe ou si l'utilisateur n'est pas banni. Pour cela on va faire appel à la base de données qui renvoie toute sa liste des comptes et IP bannies puis on les stock dans self.ban

```
# creation de la liste des utilisateurs bannis
cursor = self.db_connection.cursor(buffered=True)
cursor.execute("SELECT ban FROM ban WHERE dateUnban > NOW()")
# Cela permet le formatage de la liste a partir des identifiants, on enleve les parentheses
self.ban = [user[0] for user in cursor.fetchall()]
```

Figure 1 mise en place de la liste des utilisateurs bannis

ensuite on teste si les paramètres du client correspond a une entrée de la liste.

```
for i in range(len(self.ban)):
    if username != self.ban[i] and client_address[0] != self.ban[i]:
        continu=1
    else:
        client_socket.send(cryptocode.encrypt("BANNED", "uhgkO48G5SETzgs54e/ù").encode('utf-8'))
        print(f"un utilisateur banni a tenté une connexion : {self.ban[i]}")
        # Fermer la connexion du client
        username = ""
        client_socket.close()
```

Figure 2 test si l'utilisateur est banni

Si ce test revient positif, on va alors envoyer un message d'erreur au client et fermer sa connexion. Si il revient négatif, on peut continuer les test afin de déterminer ses droits d'accès afin d'afficher les canaux auquel l'utilisateur est autorisé a lire, pour cela, on va lui donner une séquence qui sera analysé par le client.

```
userid, username, access_rights = user
# Envoyer l'autorisation au client avec le numéro d'utilisateur et les droits d'accès
#print(f"AUTHORIZED,{userid},{username},{access_rights}")
client_socket.send(cryptocode.encrypt(f"AUTHORIZED,{userid},{username},{access_rights}\n", "uhgkO4SG5SETzgs54e/ù").encode('utf-8'))
time.sleep(0.5)
self.broadcast(f"{username} s'est connecté", "0.0.0.0")
```

Figure 3 : Réponse du serveur au client avec les droits d'utilisateur

Après cela, le serveur se met en attente de message du client

Le serveur gère les canaux de chat par le biais d'un préfixe avant chaque messages (/General, /Blabla,...) cela permet au serveur de bien identifier a quel canal le message est destiné et cela permet alors de sauvegarder le message dans la table de la base de donnée correspondante.

Après le message sauvegardé, le message est alors transmis a la fonction *broadcast*.

4. broadcast

Cette fonction va alors recevoir le message du client et le retransmettre à tous les clients connectés y compris de client destinataire, cela permet au destinataire de savoir si la connexion est établie.

```
message = cryptocode.encrypt(message, "uhgkO4SG5SETzgs54e/ù")
#print(f"message crypté: {message}")
# Diffusion du message à tous les clients
for client in self.clients:
    try:
        # Envoi du message à chaque client meme a l'expéditeur afin de voir si la connexion au serveur est active
        client[0].send(message.encode('utf-8'))
    except:
        # En cas d'erreur, fermer la connexion du client
        self.remove_client(client[0])
```

Figure 4 : Fonction Broadcast afin de diffuser les messages

5. remove_client

Cette fonction permet de supprimer les clients de la liste des clients si une exception est levée par une autre fonction.

La fonction va parcourir la liste et supprimer l'index correspondant au client à supprimer

```
# Retirer un client de la liste

for client in self.clients:
    # si le client correspond au socket de la liste client, on ferme son socket
    if client[0] == client_socket:
        self.clients.remove(client)
```

Figure 5 : Fonction remove_client

6. commande

Cette fonction permet l'administration complète de l'application, cette fonction est gérée dans un thread, ce qui permet l'exécution des commandes en même temps que l'écoute des connexions et des messages.

Les commandes suivantes sont gérées :

- /help permet d'afficher la liste des commandes disponibles
- /ban <arg1> <arg2>(optionnel) permet de bannir un utilisateur et son adresse IP
- /unban <arg1> permet de gracier un utilisateur banni
- /send /<Canal>(optionnel) <message> permet d'envoyer un message via le serveur
- /kick <utilisateur> <jour(s)> permet de bannir temporairement un utilisateur
- /droit <numéro de droit> <utilisateur> permet de changer les droits de l'utilisateur -> droit se fait par rapport au numéro : voir readme
- /createuser <utilisateur> <mot de passe> <numéro de droit> permet de créer un utilisateur via la console
- /deluser <utilisateur> permet de supprimer un utilisateur via la console
- /mdp <utilisateur> <mot de passe> permet à l'administrateur de changer le mot de passe depuis la console")

Lorsque l'administrateur envoie la commande, celle-ci est traitée par le serveur : le serveur va analyser le préfixe (ou premier mot) de la chaîne de caractère et en fonction du résultat, la commande correspondante est exécutée.

```
serverStatus = 1
mdp=hashlib.sha256(input("Mot de Passe d'administration :").encode("utf-8")).hexdigest()
if mdp==self.mdpRoot:

    print(f"/help ou /? pour afficher l'aide")
    while serverStatus==1:
        commandPrompt = input("root : ")
        try :
            command= commandPrompt.split()[0]
```

Figure 6 : premières lignes de la fonction commandes

NB : `command= commandPrompt.split()[0]` permet de sauvegarder le préfixe de la commande afin d'effectuer les tests qui s'ensuivent.

3. Commandes

Ici nous allons expliquer plus en détail la liste des 10 commandes disponibles.

NB : toutes les commandes se basent sur les mêmes principes

1. /shutdown

Comme son nom l'indique cette commande entraine l'arrêt du serveur au bout de 60 secondes

On va alors envoyer un message dans tous les canaux afin de prévenir les utilisateur de l'arrêt du serveur

Puis a bout du temps imparti, le serveur va alors fermer les sockets des clients les un après les autres puis quitter le programme

```
if commandPrompt==" /shutdown":
    #on est obligé de faire un delai entre chaque message sinon tout les message se retrouvent dans Blabla
    self.broadcast("vous allez etre déconnecté dans 1 minute (le serveur va s'arreter)", "0.0.0.0")
    time.sleep(0.5)
    self.broadcast("/Blabla vous allez etre déconnecté dans 1 minute (le serveur va s'arreter)", "0.0.0.0")
    time.sleep(0.5)
    self.broadcast("/Informatique vous allez etre déconnecté dans 1 minute (le serveur va s'arreter)", "0.0.0.0")
    time.sleep(0.5)
    self.broadcast("/Marketing vous allez etre déconnecté dans 1 minute (le serveur va s'arreter)", "0.0.0.0")
    time.sleep(0.5)
    self.broadcast("/Comptabilite vous allez etre déconnecté dans 1 minute (le serveur va s'arreter)", "0.0.0.0")
    time.sleep(60)

    print("Arrêt du serveur en cours...")
    # Fermer tous les sockets clients
    for client_socket, _ in self.clients:
        try:
            client_socket.send(cryptocode.encrypt("QUIT", "uhgk04SG58ETzgs54e/ù").encode('utf-8'))
            client_socket.shutdown(socket.SHUT_RDWR)
            client_socket.close()
        except Exception as e:
            print(f"Erreur lors du shutdown : {e}")

    # Fermer le socket serveur
    try:
        self.server_socket.shutdown(socket.SHUT_RDWR)
        self.server_socket.close()
        print("Serveur déconnecté")
    except Exception as e:
        print(f"Erreur lors de l'extinction du serveur {e}")
    os.close(1)

    # Arrêter le thread de commande
    self.serverstatus = 0
    quit()

    print("Serveur arrêté")
```

Figure 7 : commande d'extinction du serveur

2. /ban

La commande ban permet de bannir un utilisateur ou son adresse IP le seul problème que j'ai avec cette commande c'est que l'utilisateur n'est pas expulsé de l'application.

Le serveur va alors séparer la commande entrée par l'administrateur et va dissocier si 1 ou 2 arguments sont précisés. Il va alors écrire dans la base de données le nom d'utilisateur et l'argument 2

Si 1 seul argument est précisé, il va alors insérer seulement 1 seul argument. Cette commande est une relique : c'est la première commande que j'ai inséré dans mon programme, du temps où j'avais séparé le bannissement via IP du bannissement via nom d'utilisateur.

```
elif command=="/ban":
    username = commandPrompt.split()[1]

    try:
        arg2 = commandPrompt.split()[2] # Prend le deuxième argument

        # Ajoute l'utilisateur à la liste des bannis avec le deuxième argument
        cursor = self.db_connection.cursor()
        cursor.execute("INSERT INTO ban(ban) VALUES (%s)", (username,))
        cursor.execute("INSERT INTO ban(ban) VALUES (%s)", (arg2,))
        self.db_connection.commit()
        cursor.close()
        print(f"{username} et {arg2} on été bannis")

    except IndexError:
        # Si aucun deuxième argument n'est fourni
        cursor = self.db_connection.cursor()
        cursor.execute("INSERT INTO ban(ban) VALUES (%s)", (username,))
        self.db_connection.commit()
        cursor.close()
        print(f"Bannissement de {username}")
    except Exception as e:
        print(f"Erreur lors du bannissement : {e}")
```

Figure 8 : commande /ban : la relique venue d'un temps ancien

3. /unban

Cette commande est plus récente, elle permet à l'administrateur de gracier un utilisateur banni sans passer par la console MySQL.

Pour cela, le même principe est utilisé, on va alors utiliser le deuxième mot de la chaîne de la commande et effectuer une requête DELETE sur la table ban sur le champ ban.ban

```
elif command=="/unban":
    username = commandPrompt.split()[1]

    try:
        # Supprime l'utilisateur de la liste des bannis
        cursor = self.db_connection.cursor()
        cursor.execute("DELETE FROM ban WHERE ban.ban =%s", (username,))
        self.db_connection.commit()
        cursor.close()
        print(f"l'utilisateur {username} a été gracié")

    except Exception as e:
        print(f"Erreur lors du traitement de la commande : {e}")
```

Figure 9 : Commande /unban

4. /send

Comme son nom l'indique cette commande permet d'envoyer les messages de la part du serveur.

Conseil : l'administrateur peut spécifier le canal d'envoi du message en précisant le préfixe :

- /General
- /Blabla
- /Informatique
- /Marketing
- /Comptabilite

NB : l'administrateur peut aussi forcer la déconnexion de tout les clients en utilisant /send QUIT

```
elif command=="/send":
    try:
        message=commandPrompt.split()
        message = ' '.join(message[1:])
        self.broadcast(message,"0.0.0.0")
    except Exception as e:
```

Figure 10 : commande /send

5. /kick

La commande /kick est une commande /ban modernisée, elle admet le même principe mais cette fois ci on y ajoute la notion de durée : l'administrateur pourra ainsi définir la durée de bannissement d'un utilisateur si l'administrateur ne spécifie pas la durée du bannissement, alors l'utilisateur est banni pour 1 jour sinon il est banni pour le nombre de jours voulu.

Vous l'avez peut être remarqué mais cette fonction utilise le module datetime ce qui lui permet d'insérer une date dans la base de donnée.

```
elif command==" /kick":
    username = commandPrompt.split()[1]

    try:
        try :
            arg2 = int(commandPrompt.split()[2]) # Prend le deuxième argument
            dateUnban = datetime.datetime.now() + datetime.timedelta(days=arg2)
            print(dateUnban)

            # Ajoute l'utilisateur à la liste des bannis avec le deuxième argument
            cursor = self.db_connection.cursor()
            cursor.execute("INSERT INTO ban(ban, dateban, dateUnban) VALUES (%s, NOW(), %s)",(username, dateUnban))
            self.db_connection.commit()
            cursor.close()
            print(f"L'utilisateur {username} a été privé de communication pour {arg2} jours jusqu'au {dateUnban}")
        except Exception as e:
            # nous sommes obligés de faire une update au cas ou l'utilisateur est déjà enregistré dans la base de donnée car c'est
            arg2 = int(commandPrompt.split()[2]) # Prend le deuxième argument
            dateUnban = datetime.datetime.now() + datetime.timedelta(days=arg2)

            cursor = self.db_connection.cursor()
            cursor.execute("UPDATE `ban` SET `dateban` = NOW(), `dateUnban` = %s WHERE `ban`.`ban` = %s", (dateUnban, username))
            self.db_connection.commit()
            cursor.close()
            print(
                f"L'utilisateur {username} a été privé de communication pour {arg2} jours jusqu'au {dateUnban}")

    except IndexError:
        try :
            dateUnban = datetime.datetime.now() + datetime.timedelta(days=1)
            # Si aucun deuxième argument n'est fourni
            cursor = self.db_connection.cursor()
            cursor.execute("INSERT INTO ban(ban, dateban, dateUnban) VALUES (%s, NOW(), %s)",(username, dateUnban ))
            self.db_connection.commit()
            cursor.close()
            print(f"L'utilisateur {username} a été privé de communication pour 1 jour")
        except Exception as e:
            print("l'utilisateur est déjà présent dans la table, modification de la ligne")
            dateUnban = datetime.datetime.now() + datetime.timedelta(days=1)
            # Si aucun deuxième argument n'est fourni et que l'utilisateur est déjà présent dans la table de bannissement
            cursor = self.db_connection.cursor()
            cursor.execute("UPDATE `ban` SET `dateban` = NOW(), `dateUnban` = %s WHERE `ban`.`ban` = %s", (username, dateUnban))
            self.db_connection.commit()
            cursor.close()
            print(f"L'utilisateur {username} a été privé de communication pour 1 jour")

    except Exception as e:
        print(f"Erreur lors du Timeout : {e}")
```

Figure 11 : commande /kick

6. /help

Commande basique d'affichage de l'aide

7. /droit

Permet à l'administrateur de changer les droits des utilisateurs a la volée.

Cette commande sépare la chaine en 3 parties : Préfixe, numéro de droit, nom d'utilisateur.

Puis elle effectue une requête UPDATE sur la table *user* sur la colonne *rights* pour l'utilisateur précisé

```
elif command=="/droit":
    try:
        droits=commandPrompt.split()[1]
        username = commandPrompt.split()[2]
        cursor = self.db_connection.cursor()
        cursor.execute("UPDATE `user` SET `rights` = %s WHERE `user`.`username` = %s", (droits, username))
        self.db_connection.commit()
        cursor.close()
    except Exception as e:
        print(f"Erreur lors du changement des droits de l'utilisateur : {e}")
```

Figure 12 : commande /droit

NB : pour que l'utilisateur ai accès aux nouveaux canaux, celui-ci doit redémarrer son application.

8. /createuser

Commande permettant de créer un nouvel utilisateur sans devoir passer par la console MySQL.

Pour cela, l'administrateur va devoir renseigner 3 arguments dans la commande qui sera ensuite traitée par le serveur et inséré dans la table.

```
elif command=="/createuser":
    try :
        droits = commandPrompt.split()[3]
        username = commandPrompt.split()[1]
        password = commandPrompt.split()[2]
        password=hashlib.sha256(password.encode("utf-8")).hexdigest()
        print(password)
        cursor = self.db_connection.cursor()
        cursor.execute("INSERT INTO user(username, password, rights) VALUES (%s, %s, %s)", (username,password,droits))
        self.db_connection.commit()
        cursor.close()
        print(f"L'utilisateur {username} est maintenant disponible")

    except Exception as e:
        print(f"erreur lors de la création de l'utilisateur : {e}")
```

Figure 13 : commande /createuser

NB : le mot de passe est hashé avant de faire son apparition dans la BDD pour des raisons de sécurité, aucun mot de passe est visible en clair.

9. /deluser

Identique a /createuser sauf qu'il ne prend que 1 argument : le nom d'utilisateur.

Cette fois ci il va utiliser la requête DELETE afin de supprimer un utilisateur.

```
elif command=="/deluser":
    try:
        username = commandPrompt.split()[1]
        cursor = self.db_connection.cursor()
        cursor.execute("DELETE FROM user WHERE `user`.`username` = %s", (username,))
        self.db_connection.commit()
        cursor.close()
        print(f"L'utilisateur {username} est maintenant supprimé")
    except Exception as e:
        print(f"Erreur lors de la suppression du compte : {e}")
```

Figure 14 : commande /deluser

NB : lors de la suppression d'un utilisateur, tous les messages envoyés par l'utilisateur seront supprimés cela permet de mettre en place le droit à l'oubli du RGPD

10./mdp

Cette commande permet quant a elle de changer le mot de passe d'un utilisateur sans passer par la console MySQL.

Il faut alors lui fournir un nouveau mot de passe et le nom d'utilisateur et en fonction de cela, le serveur va modifier le mod de passe de l'utilisateur concerné avec la requête UPDATE sur le champ user.password.

```
elif command=="/mdp":
    try :
        username = commandPrompt.split()[1]
        password = commandPrompt.split()[2]
        password = hashlib.sha256(password.encode("utf-8")).hexdigest()
        cursor = self.db_connection.cursor()
        cursor.execute("UPDATE `user` SET `password` = %s WHERE `user`.`username` = %s", (password, username))
        self.db_connection.commit()
        cursor.close()
    except Exception as e:
        print(f"Erreur lors du changement de mot de passe : {e}")

else:
    print("commande inconnue")
```

Figure 15 : commande /mdp

4. Base de données

Nous allons maintenant nous intéresser à la structure de la Base de Données.

La base de données possède 7 tables

- 5 tables stockant chacune les messages de chaque canal
- 2 tables permettant au fonctionnement de la modération

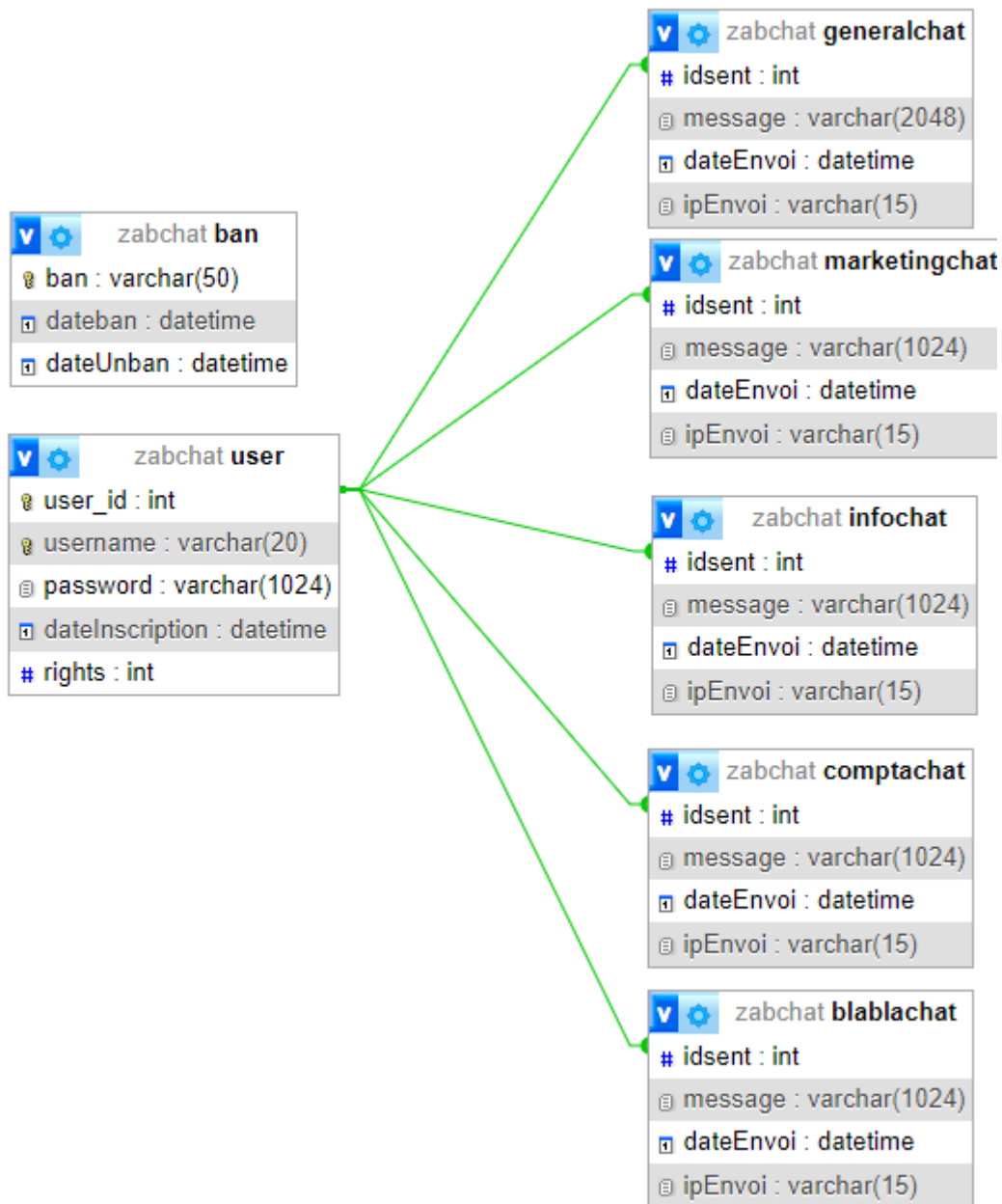


Figure 16 : structure de la base de données

1. Table user

Cette Table stocke l'ensemble des utilisateurs pouvant se connecter à l'application de chat

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 user_id	int			Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer Plus
<input type="checkbox"/>	2 username	varchar(20)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	3 password	varchar(1024)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	4 dateInscription	datetime			Non	CURRENT_TIMESTAMP		DEFAULT_GENERATED	Modifier Supprimer Plus
<input type="checkbox"/>	5 rights	int			Non	1			Modifier Supprimer Plus

Figure 17 : structure de la table user

- User_id est la clé primaire de la table, elle est unique
- Username est une clé unique à la table
- Password stocke les chaînes de caractère jusqu'à 1024 caractères
- dateInscription montre la date de la création du compte
- rights stocke les droits d'accès de l'utilisateur

2. Table ban

Cette table stocke l'ensemble des utilisateurs/ IP bannis en se connectant au serveur, le serveur va lire l'ensemble de la colonne ban afin d'en faire une liste et déterminer si l'utilisateur a le droit de se connecter ou pas

NB : cette table a une hiérarchie supérieure à la table user dans le programme, si un l'on crée un utilisateur figurant dans la table ban, celui-ci ne pourra pas se connecter

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1 ban	varchar(50)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer Plus
<input type="checkbox"/>	2 dateban	datetime			Non	CURRENT_TIMESTAMP		DEFAULT_GENERATED	Modifier Supprimer Plus
<input type="checkbox"/>	3 dateUnban	datetime			Non	5000-01-01 00:00:00			Modifier Supprimer Plus

- Ban clé primaire et unique
- dateban date à laquelle l'utilisateur s'est fait bannir (par défaut NOW())
- dateUnban date à laquelle l'utilisateur est sensé récupérer son pouvoir

NB : la date de unban est fixée à l'an 5000 cela est dû au fait que nous utilisons la même table pour bannir que pour le kick (ce n'est pas dû au fait que l'an 5000 est une amnistie totale).

3. Table chat

Les tables Chats sont tous identiques les uns par rapport aux autres.

	#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
<input type="checkbox"/>	1	idsent 	int			Non	Aucun(e)			 Modifier  Supprimer Plus
<input type="checkbox"/>	2	message	varchar(2048)	utf8mb4_general_ci		Non	Aucun(e)			 Modifier  Supprimer Plus
<input type="checkbox"/>	3	dateEnvoi	datetime			Non	CURRENT_TIMESTAMP		DEFAULT_GENERATED	 Modifier  Supprimer Plus
<input type="checkbox"/>	4	ipEnvoi	varchar(15)	utf8mb4_general_ci		Non	Aucun(e)			 Modifier  Supprimer Plus

Figure 18 : Table de stockage des messages

- Idsent est une clé étrangère qui permet d'associer à un message un expéditeur
- Message est la chaîne de caractère du message
- DateEnvoi stocke la date d'envoi du message
- IpEnvoi stocke l'IP de l'expéditeur

NB : ici la clé étrangère est configurée comme étant une cascade de user_id c'est-à-dire que si il y a un changement de l'user_id, celui-ci sera répercuté dans les tables de chat. Si l'on supprime un utilisateur, tout ses messages seront supprimés.

5. Droits d'accès

Pour la définition des droits, je me suis inspiré des droits linux afin de simplifier la gestion des droits.

Ce qui donne ce tableau

	INFO	MARKET	COMPTA	GENERAL/BLABLA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	0	1
4	0	1	1	1
5	1	0	0	1
6	1	0	1	1
7	1	1	0	1
8	1	1	1	1

Figure 19 : tableau des droits (1 signifie accès au canal, 0 refus)

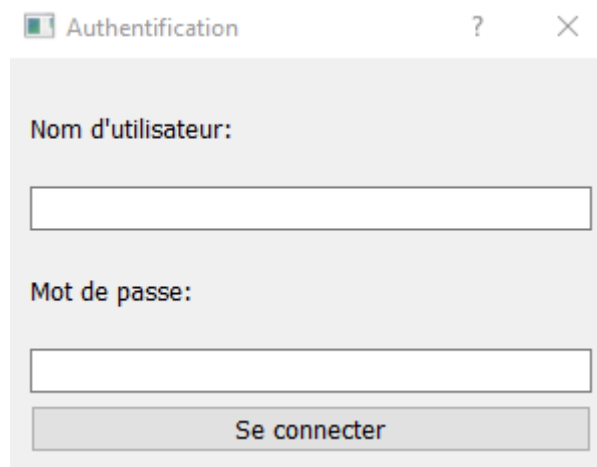
IV. Documentation Client

1. Interface Utilisateur

Chaque fenêtres sont placées dans leurs classes respectives, cela signifie que le programme possède donc 3 classes :

1. Fenêtre d'authentification

La fenêtre d'authentification est basique

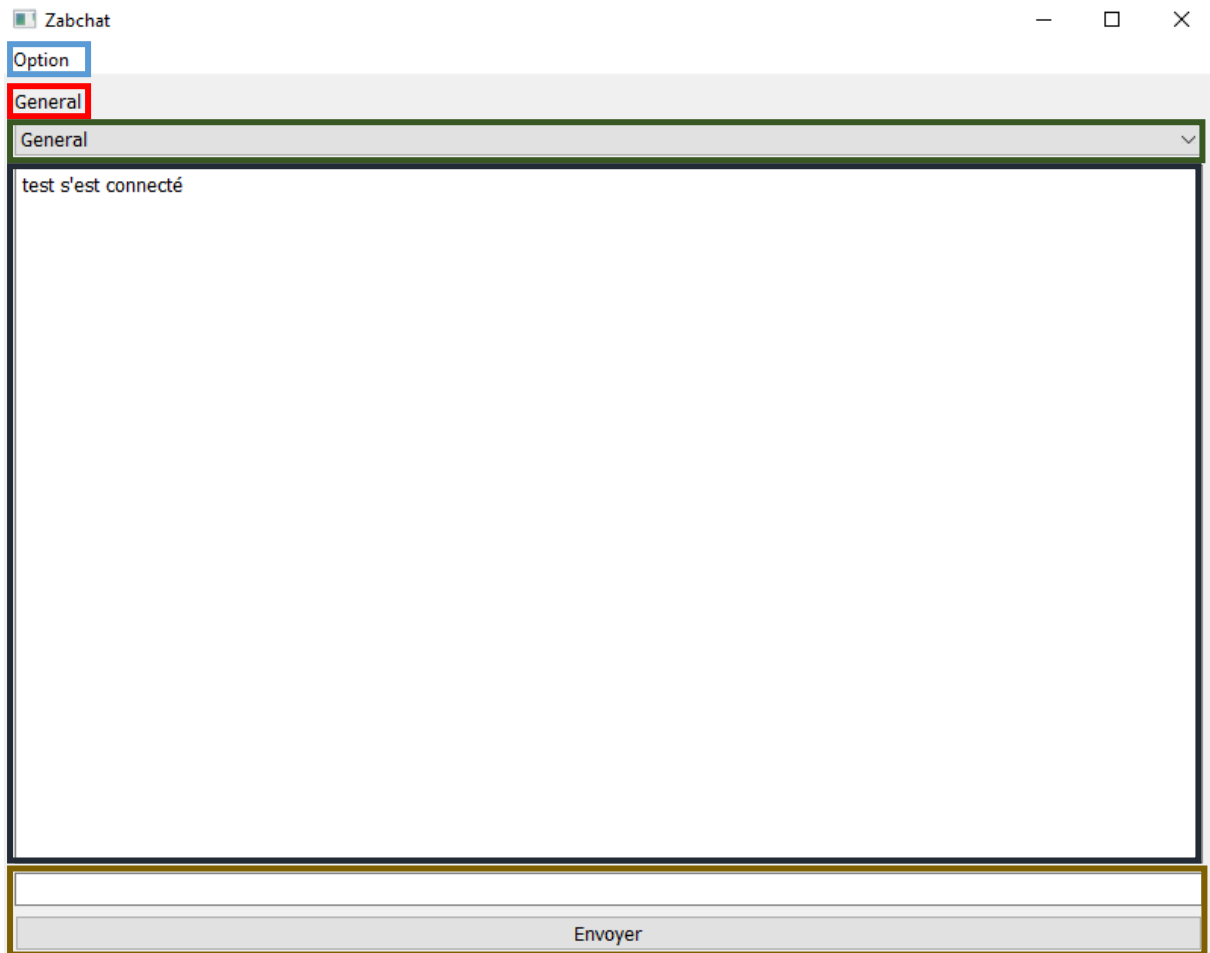


```
def initUI(self):  
    self.setGeometry(300, 300, 300, 200)  
    self.setWindowTitle('Authentification')  
  
    self.lblUsername = QLabel('Nom d\'utilisateur:', self)  
    self.lblPassword = QLabel('Mot de passe:', self)  
  
    self.tbxUsername = QLineEdit(self)  
    self.tbxPassword = QLineEdit(self)  
    self.tbxPassword.setEchoMode(QLineEdit.Password)  
  
    self.btnLogin = QPushButton('Se connecter', self)  
    self.btnLogin.clicked.connect(self.authenticate_user)  
  
    layout = QVBoxLayout()  
    layout.addWidget(self.lblUsername)  
    layout.addWidget(self.tbxUsername)  
    layout.addWidget(self.lblPassword)  
    layout.addWidget(self.tbxPassword)  
    layout.addWidget(self.btnLogin)  
  
    self.setLayout(layout)
```

Figure 21 : code de l'interface utilisateur de la fenêtre d'authentification

2. Fenêtre principale

La fenêtre principale est composée de plusieurs Widgets :



- Un menu option qui permet a l'utilisateur de changer de mot de passe
- Un label qui permet de savoir dans quel canal on se trouve
- Une combobox qui permet à l'utilisateur de changer de mot de passe
- La textbox de tous les messages reçus du canal spécifique
- La textbox de saisie du message à envoyer

```

def initUI(self):
    self.setGeometry(100, 100, 800, 600)
    self.setWindowTitle('Zabchat')

    self.lblChat = QLabel("General")
    self.tbxGeneralChat = QTextEdit(self)
    self.tbxGeneralChat.setReadOnly(True)
    self.tbxGeneralChat.setStyleSheet("background-color: rgb(255, 250, 205);")
    self.tbxGeneralChat.setStyleSheet("color: black;")
    self.tbxGeneralChat.hide()

    self.tbxBlablaChat = QTextEdit(self)
    self.tbxBlablaChat.setReadOnly(True)
    self.tbxBlablaChat.setStyleSheet("background-color: rgb(255, 250, 205);")
    self.tbxBlablaChat.setStyleSheet("color: blue;")
    self.tbxBlablaChat.hide()

    self.tbxInformatiqueChat = QTextEdit(self)
    self.tbxInformatiqueChat.setReadOnly(True)
    self.tbxInformatiqueChat.setStyleSheet("background-color: rgb(255, 250, 205);")
    self.tbxInformatiqueChat.setStyleSheet("color: green;")
    self.tbxInformatiqueChat.hide()

    self.tbxMarketingChat = QTextEdit(self)
    self.tbxMarketingChat.setReadOnly(True)
    self.tbxMarketingChat.setStyleSheet("background-color: rgb(255, 250, 205);")
    self.tbxMarketingChat.setStyleSheet("color: pink;")
    self.tbxMarketingChat.hide()

    self.tbxComptabiliteChat = QTextEdit(self)
    self.tbxComptabiliteChat.setReadOnly(True)
    self.tbxComptabiliteChat.setStyleSheet("background-color: rgb(255, 250, 205);")
    self.tbxComptabiliteChat.setStyleSheet("color: orange;")
    self.tbxComptabiliteChat.hide()

    self.input_line = QLineEdit(self)
    # Envoi du message si l'utilisateur appuie sur la touche entrer
    self.input_line.returnPressed.connect(self.send_message)

    self.send_button = QPushButton('Envoyer', self)
    self.send_button.clicked.connect(self.send_message)

    # definition de la combobox des différents cannaux
    self.cbxCanaux = QComboBox(self)

    self.selectChat("General")
    self.cbxCanaux.addItem("General")
    self.selectChat(self.lblChat.text())
    self.cbxCanaux.addItem("Blabla")
    if self.droits >= 5:
        self.cbxCanaux.addItem("Informatique")
    if self.droits == 3 or self.droits == 4 or self.droits == 7 or self.droits == 8:
        self.cbxCanaux.addItem("Marketing")
    if self.droits == 2 or self.droits == 4 or self.droits == 6 or self.droits == 8:
        self.cbxCanaux.addItem("Comptabilite")

    self.cbxCanaux.currentTextChanged.connect(self.selectChat)

```

Figure 22 : code de l'interface utilisateur de la fenêtre principale

Lors de l'appel de l'initialisation de la fenêtre, le code va vérifier des droits de l'utilisateur via le retour du serveur et va alors ajouter les éléments dans la liste déroulante.

3. Fenêtre de changement de mot de passe

L'utilisateur a la possibilité de changer de mot de passe à n'importe quel moment tant qu'il est connecté, pour cela l'utilisateur a 2 manières de faire :

- Ctrl+p
- Menu option -> changer de mot de passe

Cela va alors ouvrir une nouvelle fenêtre

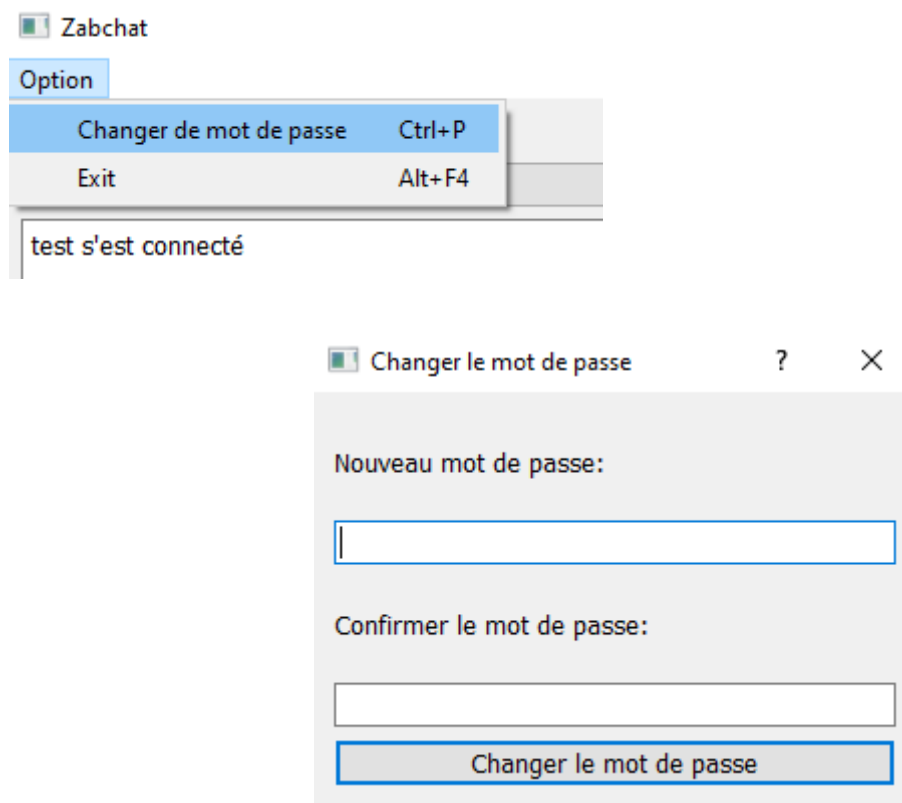


Figure 23 : fenêtre de changement de mot de passe

Le code pour afficher cette fenêtre est presque identique que pour l'authentification :

```
def initUI(self):
    """
    Fonction qui permet de définir les éléments de l'interface utilisateur
    :return: void
    """
    self.setGeometry(300, 300, 300, 200)
    self.setWindowTitle('Changer le mot de passe')

    self.lblNouvMDP = QLabel('Nouveau mot de passe:', self)
    self.tbxNouvMDP = QLineEdit(self)
    self.tbxNouvMDP.setEchoMode(QLineEdit.Password)

    self.lblConfirm = QLabel('Confirmer le mot de passe:', self)
    self.tbxConfirm = QLineEdit(self)
    self.tbxConfirm.setEchoMode(QLineEdit.Password)

    self.btnChangerMDP = QPushButton('Changer le mot de passe', self)
    self.btnChangerMDP.clicked.connect(self.change_password)

    layout = QVBoxLayout()
    layout.addWidget(self.lblNouvMDP)
    layout.addWidget(self.tbxNouvMDP)
    layout.addWidget(self.lblConfirm)
    layout.addWidget(self.tbxConfirm)
    layout.addWidget(self.btnChangerMDP)

    self.setLayout(layout)
```

Figure 24 : code de l'IU de la fenêtre de changement de mot de passe

2. Fonctions

Nous allons maintenant nous intéresser à quelques fonctions intéressantes :

1. authenticate_user

Cette fonction est appelée lorsque l'utilisateur clique sur le bouton de connexion de la fenêtre d'authentification

```

try:
    # Envoyer le nom d'utilisateur et le mot de passe au serveur pour l'authentification
    username = self.tbxUsername.text()
    password = self.tbxPassword.text()
    # password= cryptocode.encrypt(password, "zabchat")
    password = hashlib.sha256(password.encode("utf-8")).hexdigest()
    print(password)

    # Envoyer le nom d'utilisateur et le mot de passe au serveur
    self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.client_socket.connect((self.HOST, self.PORT))

    # Obligation de mettre un délai entre la connexion au socket et l'envoi des messages sinon la fenêtre est bloquée
    time.sleep(0.5)
    self.client_socket.send(username.encode('utf-8'))
    self.client_socket.send(password.encode('utf-8'))

    # Recevoir la réponse du serveur
    response = self.client_socket.recv(1024).decode('utf-8')
    response = cryptocode.decrypt(response, "uhgkO4SG5SETzgs54e/ù")

    # si le message commence avec "AUTHORIZED" on accepte la boîte de dialogue
    if response.startswith("AUTHORIZED"):
        # Analyser la réponse pour obtenir le numéro d'utilisateur et les droits d'accès, séparé par une virgule
        _, userid, utilisateur, droits = response.split(',')
        self.userid = userid
        self.utilisateur = utilisateur
        self.droits = int(droits)

        # Fermer la fenêtre d'authentification et afficher la fenêtre principale
        self.accept()
    elif response.startswith("BANNED"):
        QMessageBox.critical(self, 'Erreur d\'authentification',
                              'Accès refusé. Vous avez été banni veuillez contacter un administrateur')
        self.client_socket.close()
    else:
        # Afficher un message d'erreur en cas d'authentification échouée
        QMessageBox.critical(self, 'Erreur d\'authentification',
                              'Accès refusé. Veuillez vérifier vos informations.')
        self.client_socket.close()

except Exception as e:
    print(f"Erreur lors de l'authentification: {e}")

```

Figure 25 : code de la fonction d'authentification

en cliquant sur le bouton, le programme va alors hasher le mot de passe puis envoyer le mot de passe et le nom d'utilisateur puis attendre la réponse du serveur.

En fonction de la réponse de ce dernier, 3 événements sont possibles :

- utilisateur ou mot de passe erronés
- l'utilisateur a été banni
- authentification réussie

si l'authentification est réussie, alors la fenêtre va se fermer et envoyer tous les paramètres au programme principal afin d'ouvrir la fenêtre de chat.

Sinon un message d'erreur s'affiche en fonction de l'erreur rencontré

2. change_password

Cette fonction est assez simple mais reste quand même essentielle afin que l'utilisateur puisse changer de mot de passe afin de sécuriser son compte.

```
new_password = self.tbxNouvMDP.text()
confirm_password = self.tbxConfirm.text()

# Vérifiez que les mots de passe correspondent
if new_password == confirm_password:
    try:
        new_password = hashlib.sha256(new_password.encode("utf-8")).hexdigest()
        # Envoyez le nouvel utilisateur et le mot de passe au serveur pour la mise à jour
        message = f"/ChangePassword {self.utilisateur} {new_password}"
        message = cryptocode.encrypt(message, "earg45rsy72zerg")
        self.client_socket.send(message.encode('utf-8'))

        # Attendez la réponse du serveur
        response = self.client_socket.recv(1024).decode('utf-8')

        if response.startswith("/PasswordChanged"):
            # Mot de passe changé avec succès
            QMessageBox.information(self, 'Changement de mot de passe', 'Mot de passe changé avec succès!')
            self.accept()
        else:
            QMessageBox.critical(self, 'Erreur', 'Échec du changement de mot de passe.')
    except Exception as e:
        print(f"Erreur lors du changement de mot de passe: {e}")
else:
    QMessageBox.critical(self, 'Erreur', 'Les mots de passe ne correspondent pas.')
```

Figure 26 : code de changement de mot de passe par l'utilisateur

Tout d'abord le programme va aller vérifier si les 2 mots de passe correspondent sinon un message d'erreur apparaît.

Ensuite on va envoyer au serveur la requête de changement de mot de passe, le serveur va alors lire le préfixe du message et modifier les paramètres dans la base de données.

A la fin, la fenêtre va alors se fermer et l'utilisateur peut continuer à utiliser l'application.

3. receive_messages

cette fonction est essentielle dans une application de chat, elle permet d'écouter si le serveur envoie des messages

```

while client_status == 1:

    try:
        # print(self.droits)
        # Réception du message du serveur
        message = self.client_socket.recv(1024).decode('utf-8')
        # on décrypte le message avec le mot de passe
        message = cryptocode.decrypt(message, "uhgkO4SG5SETzgs54e/ù")

        # on determine a quel cannal le message apartient
        if message.startswith("/Blabla"):
            messagetransmi = message.split()
            messagetransmi = ' '.join(messagetransmi[1:])
            self.tbxBlablaChat.append(messagetransmi)
        elif message.startswith("/Informatique") and (
            self.droits == 5 or self.droits == 6 or self.droits == 7 or self.droits == 8):
            messagetransmi = message.split()
            messagetransmi = ' '.join(messagetransmi[1:])
            self.tbxInformatiqueChat.append(messagetransmi)
        elif message.startswith("/Marketing") and (
            self.droits == 3 or self.droits == 4 or self.droits == 7 or self.droits == 8):
            messagetransmi = message.split()
            messagetransmi = ' '.join(messagetransmi[1:])
            self.tbxMarketingChat.append(messagetransmi)
        elif message.startswith("/Comptabilite") and (
            self.droits == 2 or self.droits == 4 or self.droits == 6 or self.droits == 8):
            messagetransmi = message.split()
            messagetransmi = ' '.join(messagetransmi[1:])
            self.tbxComptabiliteChat.append(messagetransmi)
        elif message.startswith("/General"):
            messagetransmi = message.split()
            messagetransmi = ' '.join(messagetransmi[1:])
            self.tbxGeneralChat.append(messagetransmi)
        elif message.startswith("AUTHORIZED") or message.startswith("/ChangePassword"):

            print(message)
        elif message == "QUIT":
            self.client_socket.close()
            app.exit(0)
            sys.exit(0)

        else:
            self.tbxGeneralChat.append(message)

    except Exception as e:
        # En cas d'erreur, fermer la connexion du client
        print(f"Erreur lors de la réception du message. {e}")
        self.client_socket.close()

```

Figure 27 : code de la fonction de réception des messages

Cette fonction est exécutée dans un thread, ce qui permet au programme d'effectuer la réception et l'envoi de message de manière simultanée

Lors de la réception du message, le programme va alors décrypter le message puis va analyser son préfixe afin de déterminer le canal auquel il appartient.

4. send_message

Le client peut aussi envoyer des messages via cette fonction :

```
# Envoi du message au serveur
message = self.input_line.text().strip()
# verification du contenu du message (si il n'est pas vide)
if message != "":
    if message != "bye":
        message = (f"/{self.lblChat.text()} {self.utilisateur}> {self.input_line.text()}")
        # on chiffre le message afin qu'il soit cachés aux clients frauduleux
        message = cryptocode.encrypt(message, "earg45rsy72zerg")
        # print(message)
        self.client_socket.send(message.encode('utf-8'))
        # self.tbGeneralChat.append(f"vous> {message}")

        self.input_line.clear()
    else:
        message = ("bye")

        self.client_socket.send(message.encode('utf-8'))
        time.sleep(0.5)
        self.client_socket.close()
        app.quit()
```

Figure 28 : code d'envoi des messages

Lors de l'envoi du message (par appui sur le bouton ou en appuyant sur entrer), le message est alors transmis au serveur, on va alors formater le message en y ajoutant le préfixe lié au canal courant.

Si l'utilisateur envoie « bye », celui-ci sera envoyé au serveur et tous les utilisateurs recevront un message disant que l'utilisateur s'est déconnecté.

V. Document Réponse

1. Les limites du programme

Cette application de chat est conçue uniquement pour transférer des messages écrit, pour tout Transfer de fichier, mieux vaut opter pour un partage de fichier via Active Directory ou serveur FTPS voir par mail.

Les messages privés entre utilisateurs ne sont pas pris en compte dans le programme (j'aurais pu l'implémenter avec le système de préfixe mais je manque de temps).

2. La Sécurité

Actuellement le programme dispose d'une sécurité basique :

Tous les mots de passe ne circulent pas en clairs sur le réseau, ils sont hashé avant d'être transmis afin de masquer les mots de passe.

Tous les messages sont cryptés afin que si quelqu'un dispose d'un programme basique ne puisse pas lire les messages des différents canaux

Tous les canaux possèdent les mêmes mots de passe de chiffrement, pour plus de sécurité j'aurais pu opter pour 1 mot de passe de chiffrement par canaux

Le chiffrement est le même pour tous concernant l'authentification, j'aurais aussi pu faire un chiffrement personnalisé en fonction du nom d'utilisateur.

Il y a une protection contre les injections SQL

3. Confidentialité

Il en va de soi qu'une application de chat n'est pas faite pour envoyer des données confidentielles. Les emails sont plus aptes à protéger les données sensibles

L'avantage d'héberger ce service localement est d'avoir un contrôle total sur la confidentialité des informations et de ne pas laisser une entreprise extérieure avoir accès à certaines informations

4. Maintenance

Je recommande un redémarrage hebdomadaire du serveur car on n'est jamais à l'abri que certains clients soient connectés, utilisant des ressources

Un redémarrage ou une expulsion de tous les clients est requis lors d'un bannissement temporaire ou définitif afin que les changements soient pris en compte.

Afin de maintenir le service en état, cela rajoutera de la charge de travail supplémentaire au service informatique qui devra gérer la gestion et la modération du service : création de compte, bannissement, changement de mot de passe, ...

5. Mots de la Rédaction

J'ai réalisé le projet dans son intégralité malgré les hauts et les bas, ça n'a pas toujours été évident de mélanger projet et entreprise du fait que je suis en apprentissage et donc moins de temps a consacré à ce projet. Mais malgré cela, j'ai réalisé le projet dans son intégralité sauf pour la partie optionnelle des communications privées qui peuvent être remplacées par les mails. Les derniers tests ont été effectués avec succès le 29/12/2023.

J'avais déjà créé un programme qui interagit avec une base de données mais le langage n'était pas le même (C#), le principe reste le même avec quelques changements.

Je l'admets, le visuel de l'application laisse à désirer mais il faut faire avec ce qu'on a j'ai essayé à plusieurs reprises de modifier l'interface mais sans succès.

Merci de votre attention durant cette lecture de ce document si vous êtes arrivé ici, je vous souhaite une bonne année.

(Ce thème a été créé par moi-même j'ai juste repris les couleurs du cahier des charges, j'espère qu'il vous plait. Je l'utilise pour tous les documents)

n'imprimez ce document que si nécessaire, ça ne sert à rien de gâcher du papier.

Vous trouverez ci-dessous les documentations que j'ai utilisé dans ce projet qui m'ont grandement aidé :

Barre de menu :

https://koor.fr/Python/Tutoriel_PySide/pyside_menu_bar.wp

Interface :

<https://www.riverbankcomputing.com/static/Docs/PyQt5/>

Couleur (afin de différencier les chats) :

<https://stackoverflow.com/questions/27432456/python-qlineedit-text-color>

QIntValidator :

<https://www.programcreek.com/python/example/106684/PyQt5.QtGui.QIntValidator>

Combobox :

https://www.tutorialspoint.com/pyqt/pyqt_qcombobox_widget.htm

MySQL connector :

<https://dev.mysql.com/doc/connector-python/en/connector-python-examples.html>

cryptocode :

<https://www.delftstack.com/fr/howto/python/python-encrypt-string/>

hashlib :

<https://docs.python.org/3/library/hashlib.html>

FIN