Table des matières

[Documentation Client 1](#_Toc154928245)

[Introduction 1](#_Toc154928246)

[1. Connexion et Communication 1](#_Toc154928247)

[2.Fonctionnalités Clés 2](#_Toc154928248)

[3.Architecture et Technologies 2](#_Toc154928249)

[Conclusion 2](#_Toc154928250)

[Documentation Serveur 3](#_Toc154928251)

[Introduction 3](#_Toc154928252)

[1.Classe et méthode 3](#_Toc154928253)

[2.Fonctionnalités Clés 4](#_Toc154928254)

[3.Panneau de Contrôle du Serveur : 5](#_Toc154928255)

[Conclusion 6](#_Toc154928256)

# Documentation Client

## Introduction

Le client de mon application de chat est conçu pour fournir une expérience utilisateur fluide et interactive. Il permet aux clients/utilisateur de se connecter au serveur, de rejoindre des salons de discussion, d'envoyer et de recevoir des messages de plusieurs clients dans un salon mais aussi en privé et demander à rejoindre un salon. Avec une interface graphique intuitive construite à l'aide de la bibliothèque PyQt5, les utilisateurs peuvent facilement naviguer à travers l'application, rendant la communication en ligne plus agréable.

### 1. Connexion et Communication

#### 1.1Connexion au Serveur :

• Les utilisateurs débutent par se connecter au serveur en fournissant l'adresse et le port. Une fois connectés, ils peuvent s'authentifier ou s'inscrire pour accéder aux fonctionnalités du chat.

*La classe ServerConnectionWindow gère la connexion initiale au serveur.*

class ServerConnectionWindow(QWidget):

    """

    Fenêtre pour la connexion au serveur de l'application de chat.

    Permet à l'utilisateur d'entrer l'adresse et le port du serveur auquel se connecter,

    puis tente d'établir une connexion. En cas de succès, passe à l'interface d'authentification.

    """

#### 1.2Envoi et Réception de Messages :

• Les utilisateurs peuvent envoyer des messages textuels dans différents salons de chat auxquels ils ont adhéré. Ils peuvent également recevoir des messages d'autres utilisateurs en temps réel via la classe ChatClient et un ClientThread dédié.

class ClientThread(QThread):

    """

    Un thread client pour gérer la réception des messages du serveur.

    Cette classe hérite de QThread pour gérer la réception des messages

    de manière asynchrone. Elle émet un signal chaque fois qu'un message est reçu.

    """

class ChatClient(QWidget):

    """

    Interface client pour l'application de chat.

    Cette classe gère la fenêtre principale de l'application client de chat.

    Elle initialise l'interface utilisateur, gère la connexion au serveur et

    l'affichage des messages reçus.

    """

### 2.Fonctionnalités Clés

#### 2.1Interface Utilisateur :

• La classe ChatClient représente l'interface principale du client. Elle offre une zone de texte pour afficher les messages des salons, une zone pour saisir les nouveaux messages, et des boutons et menus pour différentes actions comme envoyer des messages, rejoindre et quitter des salons.

#### 2.2Gestion des Salons :

• Les utilisateurs peuvent rejoindre différents salons disponibles ou en quitter, via la méthode def join\_salon et leave\_salon dans la classe ChatClient.

#### 2.3Messages Privés :

• En plus des salons publics, les utilisateurs ont la possibilité d'envoyer des messages privés à d'autres utilisateurs, permettant des conversations plus personnelles et directes avec la méthode send\_private\_message dans la classe ChatClient.

#### 2.4Notifications et Alertes :

• La méthode display\_message dans ChatClient s'occupe de l'affichage des notifications et des alertes.

    def display\_message(self, message):

        """

        Affiche les messages reçus du serveur.

        Cette méthode gère l'affichage des messages normaux, des avertissements,

        et des notifications administratives. Elle gère aussi la fermeture

        de la session en cas de kick ou ban.

**"""**

### 3.Architecture et Technologies

ClientThread : Ce thread s'occupe de recevoir les messages du serveur de manière asynchrone pour éviter de bloquer l'interface utilisateur.

PyQt5 : Utilisé pour construire l'interface graphique, permettant une expérience utilisateur interactive et agréable.

## Conclusion

La partie client de l'application de chat est essentielle pour une expérience utilisateur réussie. Elle offre une interface intuitive, supporte des communications fluides et sécurisées, et permet une interaction facile avec le serveur et les autres utilisateurs. En résumé, le client rend l'utilisation de l'application de chat accessible, engageante, et sécurisée pour tous.

# Documentation Serveur

## Introduction

Au début, ce projet de serveur de chat était contrôlé uniquement par des commandes textuelles dans le terminal. C'était fonctionnel, mais pas toujours pratique. Alors, j'ai ajouté une interface graphique pour rendre les choses plus simples et visuelles, tout en gardant la possibilité d'utiliser le terminal. Maintenant, vous avez le choix entre utiliser les commandes dans le terminal ou cliquer sur des boutons dans l'interface graphique (notification : dans l’interface graphique seules les notifications lors de l’utilisation des commandes administratives et gestions des salons apparait ; tandis que dans le terminal toute la notification apparait en plus des commandes administrative ex : client {alias} à rejoint/quitter le salon, Authentification de {alias} réussi…)

### 1.Classe et méthode

#### 1.1 Connexion à la Base de Données

Le serveur utilise une base de données MySQL pour enregistrer des infos importantes comme les utilisateurs, leurs messages, l’authentification des utilisateurs, et la gestion des sanctions administrative. Pour se connecter et interagir avec cette base de données, il y a une méthode spécifique :

Méthode de Connexion :

*# Utilisation de la méthode get\_db\_connection de l'objet Server pour obtenir une connexion à la base de données dans les méthodes qui ont besoins d’être connecter (par ex : voir la méthode login)*

def get\_db\_connection(self):

            """Établit une connexion à la base de données et la retourne."""

            try:

                connection = pymysql.connect(

                    host=self.db\_host,

                    user=self.db\_user,

                    password=self.db\_password,

                    db=self.db\_name,

                    charset='utf8mb4',

                    cursorclass=pymysql.cursors.DictCursor

                )

                return connection

            except Exception as e:

                print(f"Erreur lors de la connexion à la base de données : {e}")

                return None

#### 1.2Pour démarrer le serveur, méthode start dans la classe Server :

def start(self):

        """

        Démarre le serveur pour écouter les connexions entrantes.

        Lie le serveur à l'adresse et au port spécifiés et commence à écouter les connexions.

        Pour chaque nouvelle connexion, un thread client est démarré pour gérer la communication.

        """

#### 1.3La classe StartServerWindow pour l’interface de démarrage du serveur :

class StartServerWindow(QMainWindow):

    """

    Fenêtre pour démarrer le serveur.

    Cette interface permet à l'utilisateur de démarrer le serveur avant de passer à l'interface d'authentification.

    """

La méthode login qui se trouve dans la classe AuthentificationWindow pour l’authentification du serveur.

    def login(self):

        """

        Gère la tentative de connexion du serveur.

        """

### 2.Fonctionnalités Clés

#### 2.1Gestion des connexions :

Accepte les connexions des clients et crée un nouveau thread pour gérer chaque client.

while True:

                client\_socket, addr = self.server\_socket.accept()

                client\_thread = threading.Thread(target=self.handle\_client, args=(client\_socket, addr))

                client\_thread.start()

#### 2.2Gestion des salons :

Permet aux clients de rejoindre différents salons de chat (Général, Blabla, etc.) et de recevoir ou envoyer des messages au sein de ces salons.

#### 2.3Authentification et gestion des utilisateurs :

Méthode login qui se trouve dans la classe Server. Authentifie les utilisateurs en utilisant les informations stockées dans une base de données MySQL.

def login(self, alias, password):

        """

        Vérifie les informations de connexion d'un utilisateur et l'authentifie si elles sont correctes.

        Utilisez get\_db\_connection de l'objet Server pour obtenir une connexion à la base de données

        """

        connection = self.get\_db\_connection()

#### 2.4 Commandes Administratives

Au début, le serveur permettait la gestion via des commandes textuelles dans le terminal. Les administrateurs pouvaient utiliser des commandes telles que kick, ban, et kill pour modérer l'activité sur le serveur.

Administration du serveur :

Fournit des commandes administratives pour gérer les utilisateurs et le serveur.

def handle\_admin\_commands(self):

        """

        Gère les commandes administratives du serveur.

        Permet à l'administrateur d'exécuter des commandes dans le terminal pour gérer le serveur et les utilisateurs,

        telle que l'acceptation ou le refus de demandes de participation à des salons,

        le bannissement d'utilisateurs, l’exclusion et aussi l’arrêt du serveur.

        """

#### 2.5Exemple de Commandes dans le Terminal :

Tout d’abord, spécifier le nom d’utilisateur, ensuite le mot de passe pour ensuite effectuer des commandes, cette fonctionnalité est incluse pour l’interface et/ou s’il on veut utiliser, dans le terminal.

Kick {alias} avec la raison : Exclut temporairement un utilisateur du chat.

Ban {alias} avec la raison : Bannit un utilisateur de façon permanente.

Kill : Arrête le serveur.

accepter 1/ refuser 1 : commande qui accepte/refuse un utilisateur de rejoindre un salon.

Explication détailler des commandes d’administration dans la section « guide pour utiliser des commandes administratives »

Avec l'ajout de l'interface graphique, ces commandes peuvent également être exécutées à travers des boutons et des formulaires, rendant la gestion plus visuelle et moins dépendante de la connaissance des commandes textuelles.

Transition vers une Interface Graphique

L'ajout d'une interface graphique pour le serveur marque une évolution importante du projet. Cette interface, est réalisée avec PyQt5 et non pas tkinter, offre une visualisation en temps réel de l'état du serveur, des connexions clients, et permet l'exécution des commandes administratives de manière intuitive.

### 3.Panneau de Contrôle du Serveur :

La classe ControlPanelWindow est une fenêtre de l'interface graphique du serveur qui permet de visualiser et d'interagir avec le serveur de manière graphique. Elle intègre des fonctionnalités telles que l'affichage des logs, la gestion des utilisateurs connectés, et l'exécution de commandes telles que kick et ban, kill, warning.

class ControlPanelWindow(QMainWindow):

    """

    Fenêtre du panneau de contrôle du serveur.

    Cette interface graphique permet à un administrateur de gérer le serveur,

     compris le bannissement ou l'exclusion d'utilisateurs, l'arrêt du serveur, etc.

    """

RequestManagementWindow est une fenêtre de l'interface graphique qui permet aux administrateurs de gérer les demandes d'adhésion aux salons

class RequestManagementWindow(QMainWindow):

    """

    Fenêtre pour gérer les demandes d'adhésion au salon.

    Permet aux administrateurs de voir et de répondre aux demandes de participation des utilisateurs.

    """

## Conclusion

En résumé, le serveur a commencé simple avec juste des commandes dans le terminal, mais maintenant il a aussi une interface graphique. Vous pouvez toujours utiliser le terminal si vous voulez, mais l'interface graphique rend les choses plus faciles, surtout pour des tâches administratives comme gérer les utilisateurs ou voir les messages. Tout cela aide à garder le chat sécurisé et agréable pour tout le monde.