**### Server Technical Document (English Version)**

#### Overview

This document outlines the technical specifications, architecture, and functionalities of the server-side component of the CHAT ROOM SAE302 application. The server is responsible for managing client connections, authenticating users, handling messages, and maintaining the overall state of the chat room.

#### Functionalities Implemented

- \*\*User Authentication\*\*: Handles user login requests and validates credentials against the database.

- \*\*Message Handling\*\*: Manages the reception, processing, and broadcasting of messages to clients.

- \*\*User Status Management\*\*: Keeps track of online and offline users and updates the list dynamically.

- \*\*Private Messaging\*\*: Facilitates private communication channels between clients upon request.

- \*\*Multithreaded Connections\*\*: Utilizes threading to manage multiple client connections simultaneously for scalability and performance.

#### System Architecture

- \*\*Server Application\*\*: A Python-based server application that uses socket programming for network communication.

- \*\*Database Connection\*\*: Integrates with a MySQL database for storing user credentials and other persistent data.

- \*\*Concurrency Management\*\*: Employs threading with thread-safe operations to handle concurrent client connections effectively.

#### Challenges and Solutions

- \*\*User List Management\*\*: Initially, updating the user list in real-time posed a challenge. A separate thread is dedicated to monitoring user status and updating the list accordingly.

- \*\*Database Synchronization\*\*: Ensures that database operations do not cause conflicts or inconsistencies, especially during concurrent access by multiple threads.

- \*\*Resource Management\*\*: The server is designed to handle a significant number of clients without resource exhaustion, using efficient thread management and optimized data structures.

#### Usage and Maintenance

- \*\*Starting the Server\*\*: The server is initiated with a specific IP and port, and it begins listening for incoming client connections.

- \*\*Monitoring\*\*: Includes logging capabilities for monitoring activities and debugging purposes.

- \*\*Maintenance\*\*: Regular updates and patches are applied to address security vulnerabilities and performance improvements.

#### Contact Information

For technical support or queries, please contact yulei.zhu@uha.fr.

---

**### Document Technique du Serveur (Version Française)**

#### Vue d'ensemble

Ce document décrit les spécifications techniques, l'architecture et les fonctionnalités du composant serveur de l'application CHAT ROOM SAE302. Le serveur est chargé de gérer les connexions clients, d'authentifier les utilisateurs, de traiter les messages et de maintenir l'état global de la salle de chat.

#### Fonctionnalités Implémentées

- \*\*Authentification des Utilisateurs\*\* : Gère les demandes de connexion des utilisateurs et valide les informations d'identification par rapport à la base de données.

- \*\*Gestion des Messages\*\* : Gère la réception, le traitement et la diffusion des messages aux clients.

- \*\*Gestion du Statut des Utilisateurs\*\* : Suit les utilisateurs en ligne et hors ligne et met à jour la liste de manière dynamique.

- \*\*Messagerie Privée\*\* : Facilite les canaux de communication privés entre les clients sur demande.

- \*\*Connexions Multithread\*\* : Utilise le multithreading pour gérer simultanément de multiples connexions clients pour l'évolutivité et la performance.

#### Architecture du Système

- \*\*Application Serveur\*\* : Une application serveur basée sur Python qui utilise la programmation par sockets pour la communication réseau.

- \*\*Connexion à la Base de Données\*\* : Intègre une base de données MySQL pour stocker les informations d'identification des utilisateurs et d'autres données persistantes.

- \*\*Gestion de la Concurrence\*\* : Emploie le multithreading avec des opérations sécurisées par des threads pour gérer efficacement les connexions clients concurrentes.

#### Défis et Solutions

- \*\*Gestion de la Liste des Utilisateurs\*\* : Au départ, la mise à jour en temps réel de la liste des utilisateurs représentait un défi. Un thread séparé est dédié à la surveillance du statut des utilisateurs et à la mise à jour de la liste en conséquence.

- \*\*Synchronisation de la Base de Données\*\* : Assure que les opérations de la base de données ne provoquent pas de conflits ou d'incohérences, en particulier lors de l'accès simultané par de multiples threads.

- \*\*Gestion des Ressources\*\* : Le serveur est conçu pour gérer un nombre significatif de clients sans épuisement des ressources, en utilisant une gestion efficace des threads et des structures de données optimisées.

#### Utilisation et Maintenance

- \*\*Démarrage du Serveur\*\* : Le serveur est initié avec une IP et un port spécifiques, et il commence à écouter les connexions clients entrantes.

- \*\*Surveillance\*\* : Comprend des capacités de journalisation pour la surveillance des activités et les bes

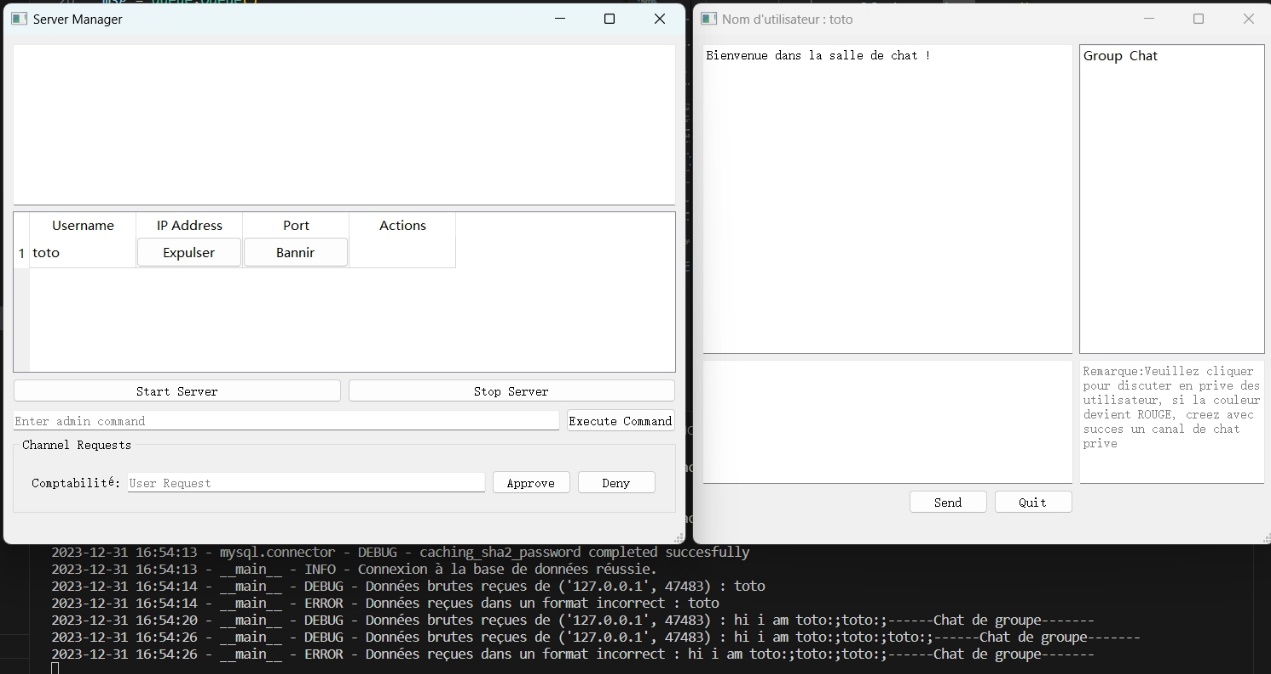
oins de débogage.

- \*\*Maintenance\*\* : Des mises à jour régulières et des correctifs sont appliqués pour traiter les vulnérabilités de sécurité et les améliorations de performance.

#### Informations de Contact

Pour le support technique ou les demandes de renseignements, veuillez contacter yulei.zhu@uha.fr.  
  
**Explications sur les fichiers V3/V4**

**V3** correspond à un serveur textuel avec un client graphique, et remplit toutes les exigences de SAE302.

**V4**, quant à lui, représente un serveur graphique et un client graphique. Toutefois, des problèmes sont survenus lors des interactions entre le client et le serveur, empêchant l'affichage correct sur l'interface utilisateur du client. Bien que le système permette à l'administrateur d'approuver les demandes d'accès aux canaux privés, les problèmes d'interaction ont causé une défaillance dans la réception des messages de réponse appropriés par le client, l'empêchant ainsi de rejoindre les canaux. [](https://github.com/Yulei-cn/SAE302_Yulei/blob/main/etage4.png)

## Analyse du Problème

Je pense que le problème réside dans les fonctions d'envoi et de réception, mais la complexité du code a augmenté en raison de la nécessité d'adapter le serveur graphique à l'interface utilisateur. Ce problème pourrait être résolu si nous avions plus de temps. L'utilisation de QT pour la création de l'interface utilisateur est très pratique, mais le débogage du code nécessite plus de temps.

**## +++Supplémentaire+++**

**### Partie 1: Ajout de Fonctionnalité Emoji**

\*\*Description\*\* :

Intégration d'un bouton emoji pour permettre aux utilisateurs de sélectionner et d'insérer des emojis dans leurs messages de chat.

\*\*Implémentation\*\* :

- \*\*UI Update\*\* : Ajouter un bouton emoji à côté de la zone de saisie du texte.

- \*\*Fonctionnalité\*\* : Ouvrir une liste d'emojis lors du clic sur le bouton pour permettre la sélection.

- \*\*Insertion d'Emoji\*\* : Insérer l'emoji sélectionné dans la zone de texte.

\*\*Exemple de Code\*\* :

```python

def insert\_emoji(self):

current\_text = self.plainTextEdit\_2.toPlainText()

emoji = "😊" # Exemple d'emoji

new\_text = current\_text + emoji

self.plainTextEdit\_2.setPlainText(new\_text)

```

\*Ce code peut être étendu pour inclure une gamme complète d'emojis.\*

\*\*Visualisation\*\* :

![image du mode chat privé](/etage5.png)

---

**### Partie 2: Fonction de Notification des Messages**

\*\*Description\*\* :

Méthode pour afficher des notifications lors de la réception de nouveaux messages dans l'application de chat.

\*\*Détails de l'Implémentation\*\* :

1. \*\*Signal PyQt\*\* : Utilisation de `QtCore.pyqtSignal(str)` pour transmettre le contenu du message.

2. \*\*Connexion du Signal\*\* : Connexion de `new\_message\_signal` à la méthode `show\_notification`.

3. \*\*Traitement des Messages Reçus\*\* : Analyse des données reçues et émission du signal avec le contenu du message.

4. \*\*Affichage des Notifications\*\* : La méthode `show\_notification` reçoit le contenu du message et l'affiche.

\*\*Code Concerné\*\* :

```python

# Définition du signal

new\_message\_signal = QtCore.pyqtSignal(str)

# Connexion du signal

self.new\_message\_signal.connect(self.show\_notification)

# Réception et traitement du message

def recv(self):

# ...

self.new\_message\_signal.emit('new\_message:' + data)

# Méthode de notification

def show\_notification(self, message):

QMessageBox.information(self, "新消息", message)

```

\*\*Visualisation\*\* :

![image du mode chat privé](/etage6.png)

Ces deux parties améliorent l'expérience utilisateur en fournissant une interface plus interactive et en facilitant la communication dans votre application de chat.