Documentation du Serveur pour l'Application de Messagerie

Table des matières

Introduction

Ce document offre une vue d'ensemble détaillée du serveur de messagerie conçu spécifiquement pour une application de chat. Ce serveur joue un rôle essentiel dans l'assurance d'une communication fluide et sécurisée entre les utilisateurs. Il agit comme le pilier central pour la gestion des interactions, garantissant ainsi une transmission de messages efficace et fiable.

Le serveur, avec son architecture robuste et modulable, est au cœur de l'application de messagerie. Il s'occupe de la gestion des connexions, de l'authentification des utilisateurs, de la transmission des messages, de la gestion des salons de discussion, et du suivi des statuts des utilisateurs. Il intègre également des fonctionnalités de sécurité avancées, comme le bannissement ou l'expulsion d'utilisateurs, pour assurer un environnement de communication sûr et contrôlé.

Développé avec un accent sur la modularité et l'évolutivité, le serveur permet l'intégration aisée de nouvelles fonctionnalités et facilite la maintenance. L'utilisation d'une base de données SQL via SQLAlchemy offre une gestion ordonnée et efficace des données utilisateur et des messages.

Cette documentation est conçue pour offrir une compréhension complète de la structure interne du serveur, ses méthodes de fonctionnement, et ses interactions avec les clients et la base de données. Elle est destinée à guider les développeurs, les administrateurs système et toutes autres parties prenantes dans la compréhension, la maintenance, et l'expansion potentielle des capacités de ce serveur de messagerie.

Configuration et Démarrage du Serveur

- Initialisation du serveur (**__init__** dans la classe **Server**).
- Démarrage et écoute des connexions (**start** dans **Server**).
  Initialisation et Écoute des Connexions :
- La méthode **start** est appelée après l'initialisation du serveur (**__init__**). Elle configure le serveur pour qu'il commence à écouter les connexions entrantes sur l'adresse IP et le port spécifiés lors de la création de l'instance de **Server**.
- Elle définit un timeout pour le socket du serveur en utilisant **self.server_socket.settimeout(1)**. Cela permet au serveur de ne pas rester bloqué indéfiniment en attendant des connexions entrantes, offrant ainsi la possibilité de vérifier régulièrement d'autres conditions ou commandes.
- **Gestion des Connexions Client :**
- La méthode entre dans une boucle **while** qui se poursuit tant que le serveur est en mode **running**.

- À l'intérieur de cette boucle, le serveur tente d'accepter les connexions entrantes en utilisant **client_socket, client_address = self.server_socket.accept()**.
- Pour chaque client qui se connecte, le serveur imprime l'adresse du client et lance un nouveau thread (**client_handler**) en appelant **self.handle_client**, qui gère la communication avec ce client spécifique.
- **Écoute des Commandes d'Arrêt :**
- Parallèlement, la méthode **start** initialise et lance un autre thread (**shutdown_thread**) qui exécute **self.listen_for_shutdown_command**. Ce thread est responsable de l'écoute des commandes d'arrêt ou d'autres commandes administratives via la console du serveur.
- Cette dualité de threads permet au serveur de gérer les clients tout en restant réactif aux commandes administratives.
- **Gestion des Exceptions et Continuité :**
- La méthode gère les exceptions telles que les timeouts. En cas de timeout, le serveur ignore simplement cette exception et continue sa boucle.
- Pour les autres exceptions inattendues, le serveur les imprime sur la console pour le diagnostic, mais continue à fonctionner. Cela garantit que le serveur reste robuste et ne s'arrête pas inopinément.
- **Arrêt Propre du Serveur :**
- Lorsqu'une commande d'arrêt (**kill**) est reçue, **self.running** est défini sur **False**, ce qui met fin à la boucle **while**.
- Enfin, la méthode appelle **self.shutdown_server**, qui ferme proprement toutes les connexions clients et le socket du serveur, garantissant ainsi un arrêt ordonné et sans perte de données.

Gestion des Connexions Client

Acceptation et Gestion des Connexions Clients :

- Lorsqu'un nouveau client se connecte, la méthode **start** crée un thread séparé pour ce client et exécute la méthode **handle_client**.
- **handle_client** reçoit deux paramètres importants : **client_socket** (le socket connecté au client) et **client_address** (l'adresse du client). Ces éléments sont essentiels pour communiquer avec le client et identifier sa source.

Communication avec le Client :

- À l'intérieur de **handle_client**, une boucle **while** est utilisée pour écouter en continu les messages provenant du client via **client_socket.recv(1024)**.
- Cette boucle reste active tant que des données sont reçues du client. Si le client se déconnecte proprement ou si le flux de données est interrompu, la boucle se termine.

Traitement des Messages Reçus :

- Les messages reçus du client sont décodés et traités en fonction de leur contenu. Par exemple, si un message commence par "LOGIN", "CREATE", "MP", etc., la méthode appropriée est appelée pour gérer ces demandes spécifiques.

Gestion des Déconnexions et Erreurs :

- Si le client se déconnecte ou si une erreur se produit (par exemple, une déconnexion inattendue), le serveur intercepte ces événements dans un bloc **try-except**.
- En cas de déconnexion inattendue, **handle_client** met à jour le statut de l'utilisateur dans la base de données pour refléter sa déconnexion, en appelant **self.update_user_status**.
- Après la gestion de la déconnexion ou de l'erreur, les ressources associées au client (comme son socket) sont fermées proprement pour libérer la mémoire et les descripteurs de fichier.

Envoyer les Statuts des Utilisateurs :

- **handle_client** peut également envoyer les mises à jour de statut des utilisateurs aux autres clients connectés. Cela garantit que tous les utilisateurs sont informés des changements de statut (par exemple, lorsqu'un utilisateur se connecte ou se déconnecte).

Authentification et Gestion des Utilisateurs

Création de Nouveaux Utilisateurs (create_new_user) :

- **Fonctionnement :** Cette méthode est appelée lorsque le serveur reçoit une demande de création de compte d'un client. Elle extrait les informations nécessaires (comme l'alias, le nom, le prénom et le mot de passe) du message reçu et crée un nouvel utilisateur dans la base de données.
- **Interaction avec la Base de Données : create_new_user** vérifie d'abord si l'alias choisi par le nouvel utilisateur est déjà pris. Si l'alias est disponible, elle enregistre le nouvel utilisateur dans la table **Authentification** et configure les droits par défaut dans la table **Droit_Salon**, ainsi que le statut initial dans la table **Status**.
- **Gestion des Erreurs :** En cas de données manquantes ou d'informations insuffisantes (comme un alias ou un mot de passe trop court), la méthode informe le client de l'échec de la création du compte.

Processus de Connexion (login) :

- **Vérification des Informations :** Lorsqu'un utilisateur tente de se connecter, **login** reçoit son alias et son mot de passe, puis vérifie ces informations dans la table **Authentification**.
- **Authentification de l'Utilisateur :** Si les informations sont correctes, l'utilisateur est authentifié avec succès. La méthode vérifie également si l'utilisateur n'est pas banni ou temporairement expulsé (kick).
- **Mise à Jour des Statuts :** Une fois authentifié, le statut de l'utilisateur est mis à jour dans la table **Status** pour indiquer qu'il est connecté.
- **Gestion des Réponses :** En fonction de l'issue de la tentative de connexion, **login** envoie une réponse appropriée au client, soit confirmant la connexion, soit signalant une erreur (comme un échec d'authentification ou un bannissement).

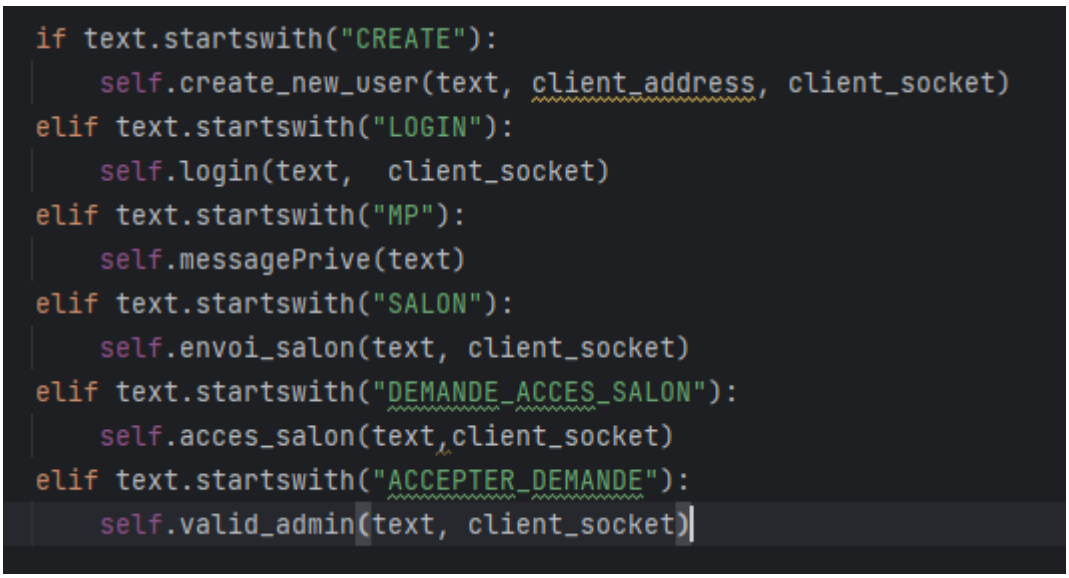
Modèles de Données pour les Utilisateurs :

- **Authentification :** Cette classe représente la table **Authentification** dans la base de données. Elle stocke les informations d'authentification des utilisateurs, incluant les alias, mots de passe et autres détails personnels.
- **Status :** La classe **Status** correspond à la table **Status** dans la base de données. Elle est utilisée pour suivre le statut de connexion des utilisateurs (connecté, déconnecté) et enregistrer la dernière mise à jour de leur statut.

Communication et Gestion des Messages

Réception et Traitement des Messages (handle_message) :

- **Fonctionnement :** Cette méthode est appelée chaque fois qu'un message est reçu d'un client. Elle détermine le type de message (création d'un compte, connexion, message privé, message de salon, etc.) et délègue son traitement à la méthode appropriée.
- **Polyvalence : handle_message** est capable de gérer divers types de requêtes, en les redirigeant vers les méthodes spécifiques comme **create_new_user**, **login**, **messagePrive**, **envoi_salon**, etc. Cela rend le serveur modulaire et facile à étendre.



```
if text.startswith("CREATE"):
    self.create_new_user(text, client_address, client_socket)
elif text.startswith("LOGIN"):
    self.login(text,  client_socket)
elif text.startswith("MP"):
    self.messagePrive(text)
elif text.startswith("SALON"):
    self.envoi_salon(text, client_socket)
elif text.startswith("DEMANDE_ACCES_SALON"):
    self.acces_salon(text,client_socket)
elif text.startswith("ACCEPTER_DEMANDE"):
    self.valid_admin(text, client_socket)
```

Envoi de Messages Privés (messagePrive) :

- **Traitement des Messages Privés :** Lorsqu'un message privé est reçu, cette méthode s'occupe de le transmettre au destinataire. Elle extrait l'alias de l'expéditeur, du destinataire et le contenu du message du texte reçu.
- **Enregistrement dans la Base de Données :** Le message est enregistré dans la table **Discussion** pour garder une trace des conversations privées entre les utilisateurs.
- **Gestion des Utilisateurs Non Connectés :** Si le destinataire n'est pas connecté, le message est enregistré mais n'est pas envoyé immédiatement. Le destinataire recevra le message lors de sa prochaine connexion.

Gestion des Messages dans les Salons (envoi_salon) :

- **Diffusion dans les Salons :** Cette méthode permet d'envoyer des messages dans un salon spécifique. Elle vérifie les droits de l'utilisateur pour s'assurer qu'il a le droit de poster dans le salon choisi.
- **Stockage des Messages :** Les messages envoyés dans les salons sont stockés dans la table **Salon**, permettant ainsi un historique des conversations de salon.

- Envoi aux Utilisateurs Autorisés : Le message est envoyé à tous les utilisateurs ayant accès à ce salon, basé sur leurs droits configurés dans Droit_Salon.

Gestion des Salons de Discussion

Gestion des Salons (Salon) :

- **Structure des Salons :** La classe **Salon** correspond à la table 'Salon' dans la base de données. Elle est utilisée pour stocker les informations relatives aux différents salons, y compris leur nom et les messages échangés.

Contrôle d'Accès aux Salons (acces_salon, valid_admin) :

- **Gestion des Droits d'Accès (acces_salon) :**
  - **Vérification des Droits :** Cette méthode vérifie si un utilisateur a les droits nécessaires pour accéder à un salon spécifié. La vérification se base sur la table **Droit_Salon** de la base de données, où les droits d'accès sont représentés sous forme de chaînes, telles que '1,1,1,1,1' (1 signifie qu'il a l'accès au salon et 0 signifie qu'il n'a pas l'accès au salon), chaque chiffre correspondant à un droit d'accès pour un salon particulier.
  - **Réponse au Client :** Selon les droits de l'utilisateur, la méthode répond de manière appropriée, confirmant l'accès ou informant de la nécessité d'une validation par un administrateur.
- **Validation par l'Administrateur (valid_admin) :**
  - **Gestion des Demandes d'Administration :** Permet de traiter les demandes de droits d'administration sur un salon spécifique, en vérifiant et en mettant à jour les droits des utilisateurs.
  - **Interaction avec la Base de Données :** Les changements de droits sont consignés dans la base de données pour une gestion cohérente et sécurisée des accès.

Suivi et Mise à Jour des Statuts

Mise à Jour des Statuts des Utilisateurs (update_user_status) :

- **Mécanisme de Mise à Jour :** Cette méthode met à jour le statut de connexion d'un utilisateur dans la base de données. Elle recherche l'utilisateur par son alias et modifie son statut (par exemple, de 'connecté' à 'déconnecté') ainsi que le timestamp de la dernière mise à jour.
- **Interaction avec la Base de Données :** Les informations de statut sont stockées dans la table **Status**, permettant un suivi en temps réel de l'état de connexion des utilisateurs.

Diffusion des Statuts aux Clients (send_user_statuses) :

- **Transmission des Informations de Statut :** Cette méthode collecte les statuts de tous les utilisateurs et les compile en un message unique, qui est ensuite envoyé à un client spécifique. Cela permet aux clients de recevoir des mises à jour en temps réel sur l'état de connexion des autres utilisateurs.

- **Format des Messages de Statut :** Les informations de statut sont formatées et envoyées sous forme d'une chaîne, comme "STATUS_UPDATE/alias1:status1,alias2:status2", pour une interprétation facile par le client.

Fonctionnalités de Sécurité

Bannir des Utilisateurs (ban_user) :

- **Action de Bannissement :** Cette méthode permet de bannir un utilisateur, empêchant ainsi toute tentative ultérieure de connexion au serveur. Elle recherche l'utilisateur spécifié dans la base de données et modifie son état de bannissement en 'True'.
- **Application du Bannissement :** Le bannissement affecte l'accès de l'utilisateur à toutes les fonctionnalités du serveur, y compris l'envoi et la réception de messages.

Débannir des Utilisateurs (unban_user) :

- **Révocation du Bannissement :** Cette méthode inverse l'action de bannissement, permettant à l'utilisateur de se reconnecter et d'utiliser le serveur normalement. Elle trouve l'utilisateur dans la base de données et réinitialise son statut de bannissement en 'False'.

Expulser Temporairement des Utilisateurs (kick_user) :

- **Expulsion Temporaire :** La méthode **kick_user** sert à expulser un utilisateur du serveur pour une durée déterminée. L'utilisateur expulsé ne peut pas se reconnecter avant l'expiration de cette période.
- **Gestion de la Durée d'Expulsion :** L'expulsion est gérée en définissant une date d'expiration dans la base de données, après laquelle l'utilisateur pourra de nouveau accéder au serveur.

Réintégrer des Utilisateurs (unkick_user) :

- **Annulation de l'Expulsion :** Cette fonctionnalité permet d'annuler une expulsion avant la fin de la durée prévue. La méthode **unkick_user** réinitialise la date d'expiration de l'expulsion dans la base de données, permettant ainsi un retour immédiat de l'utilisateur.

Administration et Commandes Serveur

Gestion des Commandes Administratives (listen_for_shutdown_command) :

- **Surveillance des Commandes :** Cette méthode écoute en permanence les commandes administratives entrantes via la console. Elle permet aux administrateurs de gérer le serveur en temps réel.
- **Types de Commandes :** Les commandes incluent l'arrêt du serveur (**kill**), le bannissement (**ban**), l'expulsion (**kick**), ainsi que les fonctions inverses (**unban**, **unkick**). Ces commandes permettent une gestion dynamique et réactive des utilisateurs et du serveur lui-même.
- **Traitement des Commandes :** Chaque commande est traitée individuellement, avec des actions spécifiques déclenchées en fonction de son type, telles que l'arrêt du serveur ou la modification des statuts des utilisateurs.

Arrêt Propre du Serveur (shutdown_server) :

- **Fermeture Sécurisée :** Cette méthode assure une fermeture ordonnée et sécurisée du serveur. Elle est primordiale pour prévenir la perte de données et maintenir l'intégrité du système.

- **Notification des Clients :** Avant la fermeture, le serveur envoie des notifications à tous les clients connectés, les informant de l'arrêt imminent. Cela permet une déconnexion propre et organisée.
- **Libération des Ressources :** Le serveur ferme toutes les connexions client et libère les ressources système avant de s'arrêter complètement. Cette étape garantit qu'aucune donnée n'est perdue ou corrompue durant la fermeture.

Intégration avec la Base de Données

Connexion et Interaction avec la Base de Données SQL :

- **Configuration de la Connexion :** Le serveur établit une connexion avec la base de données SQL via SQLAlchemy. Cette configuration permet une interaction fluide et efficace avec la base de données pour toutes les opérations de données.
- **Manipulation des Données :** SQLAlchemy offre une interface intuitive pour effectuer des requêtes, des mises à jour et des suppressions dans la base de données, simplifiant considérablement la gestion des données.

Modèles de Données pour la Gestion :

- **Discussion :** Ce modèle stocke l'historique des conversations privées, y compris l'expéditeur, le destinataire et le contenu des messages. Il est essentiel pour la préservation des communications privées entre les utilisateurs.
- **Droit_Salon :** Gère les droits d'accès des utilisateurs aux différents salons. Chaque enregistrement indique les salons auxquels un utilisateur peut accéder, facilitant ainsi la gestion des droits.
- **Status :** Suit le statut de connexion des utilisateurs (connecté/déconnecté) et enregistre la dernière modification de leur statut. Ce modèle est crucial pour le suivi en temps réel de la présence des utilisateurs sur le serveur.
- **Authentification :** Contient les informations d'authentification des utilisateurs, y compris les alias, mots de passe et autres informations personnelles. Ce modèle est fondamental pour la sécurité et l'intégrité des comptes utilisateurs.
- **Salon :** Correspond à la table 'Salon' dans la base de données, utilisée pour gérer les différents salons de discussion, leurs noms et les messages échangés.
- **Authentification**
    - Alias: **VARCHAR** (Primary Key)
    - Adresse_IP: **VARCHAR**
    - Nom: **VARCHAR**
    - Prenom: **VARCHAR**
    - Password: **VARCHAR**
    - Droit: **VARCHAR** (par défaut '0')

    - Is_Banned: **BOOLEAN** (par défaut FALSE)
    - Kick_Expiration: **DATETIME** (Null)
- **Status**
    - ID: **INT** (Primary Key, Auto-Increment)
    - Alias: **VARCHAR** (Foreign Key référençant Authentification.Alias)
    - Status_Connexion: **VARCHAR**
    - Timestamp: **TIMESTAMP** (par défaut CURRENT_TIMESTAMP)
- **Salon**
    - ID: **INT** (Primary Key, Auto-Increment)
    - Nom: **VARCHAR**
    - Message: **TEXT**
    - Timestamp: **TIMESTAMP** (par défaut CURRENT_TIMESTAMP)
- **Discussion**
    - ID: **INT** (Primary Key, Auto-Increment)
    - Alias: **VARCHAR** (Foreign Key référençant Authentification.Alias)
    - Destination: **VARCHAR**
    - Conversation: **TEXT**
    - Timestamp: **TIMESTAMP** (par défaut CURRENT_TIMESTAMP)
- **Droit_Salon**
    - ID: **INT** (Primary Key, Auto-Increment)
    - Alias: **VARCHAR** (Foreign Key référençant Authentification.Alias)
    - Droits: **VARCHAR** (par défaut '1,0,0,0,0' parce que le salon général est accessible à tout le monde)

Conclusion

En conclusion, ce serveur de messagerie représente une solution complète pour la gestion de la communication dans l'application de chat. Il est prêt à être déployé et utilisé dans un environnement de production, offrant une plateforme stable et évolutive pour la communication numérique. Cette documentation devrait servir de guide de référence pour les développeurs, les administrateurs système et toute autre partie prenante intéressée par la compréhension, la maintenance ou l'extension des fonctionnalités de ce serveur.