Documentation pour le Client de l'Application de Messagerie

Table des matières

Introduction

L'application de messagerie est conçue pour permettre aux utilisateurs de communiquer efficacement en temps réel. Elle offre des fonctionnalités telles que la messagerie privée, les discussions de groupe dans différents salons, et la possibilité de gérer le statut de connexion.

Les classes : Structuration du Code

Le code de l'application de messagerie est divisé en plusieurs classes, chacune ayant un rôle spécifique. Cette approche modulaire facilite la gestion, la maintenance et l'évolution du code. Voici les raisons principales de cette structuration :

***1.** Séparation des Responsabilités :*

Chaque classe est conçue pour gérer une fonctionnalité distincte. Cela permet d'isoler les différentes parties de l'application, rendant le code plus lisible, plus facile à déboguer et à tester.

***2.** Réutilisabilité et Maintenance :*

Les classes peuvent être réutilisées dans différents contextes de l'application. Par exemple, une classe gérant la communication réseau peut être utilisée à la fois pour les messages privés et les discussions de groupe. Cela évite la duplication de code et simplifie les mises à jour.

***3.** Facilité d'Extension :*

Avec des classes bien définies, il est plus facile d'ajouter de nouvelles fonctionnalités. Par exemple, introduire un nouveau type de message ou un nouveau salon nécessite souvent seulement l'ajout ou la modification d'une classe spécifique.

4. Encapsulation et Sécurité :

Les classes permettent d'encapsuler des données, c'est-à-dire de cacher certaines parties du code à d'autres parties. Cela aide à protéger les données et à prévenir leur modification inappropriée.

5. Clarté et Organisation :

Des classes bien conçues rendent le code plus clair et mieux organisé. Les développeurs peuvent rapidement comprendre la structure et les fonctionnalités du code, facilitant ainsi la collaboration et la prise en main rapide par de nouveaux développeurs.

Exemples de Classes dans l'Application :

- **LoginPage :** Gère l'interface de connexion.
- **CreateAccountPage :** Traite la création de nouveaux comptes.
- **MessagingPage :** S'occupe de l'affichage et de l'envoi des messages.
- **StatusPage :** Affiche les statuts des utilisateurs.
- **DemandesAccesTab :** Gestion des Demandes d'Accès aux Salons

La segmentation en classes contribue à l'efficacité globale de l'application en assurant que chaque partie du code a une responsabilité claire et bien définie. Cela se traduit par une meilleure performance, une meilleure fiabilité et une expérience utilisateur améliorée.

Création de Compte :

1. Saisie des Informations Requises :
    - L'utilisateur remplit les champs nécessaires sur la page de création de compte, y compris le prénom, le nom, l'alias (nom d'utilisateur) et le mot de passe.
2. Envoi de la Demande de Création de Compte :
    - En cliquant sur le bouton « Créer un compte », la méthode **create_account** est activée.
    - Cette méthode envoie une requête au serveur sous la forme **CREATE/{alias}/{nom}/{prenom}/{password}**. Elle utilise le socket client pour transmettre ces informations.
3. Traitement de la Réponse du Serveur :
    - Le client attend ensuite la réponse du serveur pour savoir si la création du compte a été réussie ou non.

- Si les conditions de création définies par le serveur sont remplies (par exemple, l'alias n'est pas déjà pris et le mot de passe répond aux critères de sécurité), le serveur répond avec **CREATE_SUCCESS**. Cela signifie que le compte a été créé avec succès.

2. Gestion des Réponses et Affichage des Erreurs :
    - Si la création du compte échoue pour une raison quelconque (par exemple, si l'alias choisi existe déjà), le serveur envoie un message d'erreur approprié.
    - Ces messages d'erreur sont affichés dans l'interface graphique en utilisant la méthode **show_error_message**. Cela informe l'utilisateur de la nature du problème et lui permet de corriger ses informations.
3. Retour à la Page de Connexion :
    - Après la création réussie du compte, l'utilisateur est redirigé vers la page de connexion pour qu'il puisse se connecter avec ses nouvelles informations d'identification.
4. Sécurité et Validation des Données :
    - Le processus de création de compte comprend des mesures pour s'assurer que les données saisies par l'utilisateur sont valides et sécurisées.
    - Le serveur effectue une validation pour éviter les doublons dans la base de données et pour s'assurer que les mots de passe sont suffisamment forts.

Connexion au Compte :

1. Saisie des Informations d'Identification :
    - L'utilisateur saisit son nom d'utilisateur (alias) et son mot de passe dans les champs correspondants de l'interface de connexion.
2. Envoi de la Demande de Connexion :
    - En cliquant sur le bouton « Se connecter », l'utilisateur active la méthode **login**.
    - Cette méthode envoie une requête au serveur sous la forme **LOGIN/{alias}/{password}**. Elle utilise le socket client pour transmettre les informations d'identification.
3. Réception et Traitement de la Réponse du Serveur :
    - Après l'envoi de la demande de connexion, le client attend la réponse du serveur.
    - Si les informations d'identification sont correctes et que l'utilisateur n'est pas banni ou temporairement expulsé, le serveur répond avec le message **AUTH_SUCCESS**.
4. Changement d'Interface en Cas de Succès :
    - En cas de réussite de l'authentification (**AUTH_SUCCESS**), l'utilisateur est redirigé vers la page principale de messagerie (**MessagingPage**).

- Cette page permet à l'utilisateur d'envoyer des messages privés, de participer à des discussions de salon, de gérer les demandes d'accès, et de voir les statuts des autres utilisateurs.

5. Gestion des Échecs de Connexion :
    - Si l'authentification échoue, le serveur renvoie un message d'erreur approprié, par exemple en cas de mauvais mot de passe ou d'utilisateur banni.
    - Le client affiche ces messages d'erreur dans une boîte de dialogue, informant l'utilisateur du problème (def show_error_message).
6. Sécurité et Confidentialité :
    - La méthode **login** assure que seuls les utilisateurs authentifiés peuvent accéder aux fonctionnalités de l'application.
    - Elle contribue à la sécurité de l'application en empêchant les accès non autorisés.

Messagerie

La messagerie est une fonctionnalité clé de l'application client, permettant aux utilisateurs de communiquer efficacement. Elle se divise en deux aspects principaux : l'envoi de messages privés et l'envoi de messages dans des salonset aussi l'envoie des demandes pour rejoindre un salon. Voici une explication détaillée des deux processus et de leur implémentation dans le code :

Envoi de Messages Privés

1. **Fonctionnement** :
    - Les utilisateurs peuvent envoyer des messages privés à d'autres utilisateurs en spécifiant l'alias du destinataire.
    - Ces messages sont confidentiels et ne sont visibles que par l'expéditeur et le destinataire.
2. **Code** :
    - La méthode **select_destinataire** de la classe **MessagingPage** est utilisée pour cet envoi.
    - L'utilisateur saisit l'alias du destinataire et le message dans des champs de saisie dédiés.
    - Après avoir cliqué sur le bouton d'envoi, le message est formaté sous la forme **"MP/{alias_destinataire}/{alias_source}/>>{message_priv}"** et envoyé au serveur via **client_socket**.
3. **Traitement Serveur** :
    - Le serveur reçoit la requête, extrait les informations et vérifie si le destinataire est disponible.

- Si disponible, le serveur transmet le message au destinataire. Sinon, il envoie une notification d'erreur ou stocke le message pour une livraison ultérieure.

Envoi de Messages dans un Salon

1. **Fonctionnement** :
    - Les salons sont des espaces de discussion de groupe où les utilisateurs peuvent envoyer des messages visibles par tous les membres du salon.
    - Les utilisateurs choisissent un salon spécifique et saisissent leur message.
2. **Code** :
    - La méthode **select_salon** de **MessagingPage** gère l'envoi de messages dans les salons.
    - L'utilisateur sélectionne un salon via **salon_box** et saisit son message.
    - Après avoir cliqué sur le bouton d'envoi, le message est formaté sous la forme **"SALON/{selected_salon}/{alias_source}/>>{message_salon}"** et envoyé au serveur.
3. **Traitement Serveur** :
    - Le serveur reçoit la requête, détermine si l'utilisateur a les droits nécessaires pour poster dans le salon choisi.
    - Si l'utilisateur a le droit, le serveur diffuse le message à tous les membres du salon. Si non, une notification d'erreur est envoyée à l'utilisateur.

Réception des Messages

Réception des Messages Privés

1. **Fonctionnement** :
    - Lorsqu'un message privé est envoyé par un autre utilisateur, il est reçu en temps réel par le destinataire si celui-ci est en ligne.
    - Les messages sont transmis de manière sécurisée pour assurer la confidentialité.
2. **Traitement dans le Client** :
    - La méthode **receive_message** dans **MessagingPage** est constamment en écoute pour de nouveaux messages du serveur.
    - Quand un message privé arrive (formaté comme **"alias_source >> message"**), il est détecté et traité.
    - Le message est affiché dans la zone de texte dédiée aux messages privés, indiquant clairement l'expéditeur.
3. **Expérience Utilisateur** :

- Les utilisateurs reçoivent des notifications instantanées des messages privés.
- Ils peuvent visualiser et répondre directement depuis l'interface de la messagerie.

Réception des Messages des Salons

1. **Fonctionnement** :
    - Les messages envoyés dans un salon sont reçus par tous les utilisateurs ayant accès à ce salon.
    - Cela permet une communication de groupe où plusieurs utilisateurs peuvent interagir en temps réel.
2. **Traitement dans le Client** :
    - La méthode **receive_message** gère également la réception des messages de salon.
    - Lorsqu'un message de salon est reçu (formaté comme **"messageDuSalon {selected_salon} de la part de {alias_source} >> {message_salon}"**), il est affiché dans la zone de texte du salon concerné.
    - Les utilisateurs peuvent voir qui a envoyé le message et dans quel salon.
3. **Expérience Utilisateur** :
    - Les messages de salon offrent une expérience interactive où les utilisateurs peuvent participer à des discussions de groupe.
    - Les utilisateurs peuvent facilement suivre les conversations dans différents salons et y contribuer.

Réception des Messages d'erreur

1. **Fonctionnement** :
    - Les messages d'erreur sont envoyés par le serveur en réponse à diverses actions de l'utilisateur, comme des tentatives de connexion échouées, des erreurs de création de compte, ou des tentatives d'accès à des salons sans les droits nécessaires.
    - Ces messages aident les utilisateurs à comprendre ce qui n'a pas fonctionné et comment ils peuvent corriger le problème.
2. **Traitement dans le Client** :
    - Lors de la réception d'un message d'erreur du serveur (typiquement formaté comme **"ERROR/...quelque message..."**), la méthode **receive_message** dans **MessagingPage** identifie et extrait le contenu du message d'erreur.
    - Un signal **error_signal** est émis avec le message d'erreur comme paramètre.
    - Ce signal est connecté à une méthode **show_error** qui affiche le message d'erreur dans une boîte de dialogue.
3. **Affichage des Erreurs** :
    - La méthode **show_error** crée une boîte de dialogue QMessageBox pour afficher l'erreur.

- Cette boîte de dialogue informe l'utilisateur de l'erreur de manière claire et concise, permettant à l'utilisateur de comprendre et de réagir en conséquence.

Expérience Utilisateur

- Les utilisateurs reçoivent des retours immédiats en cas d'erreur, leur permettant de comprendre rapidement la nature du problème.
- La présentation claire des messages d'erreur aide à éviter la confusion et améliore l'expérience utilisateur globale.
- Cette approche permet également de guider l'utilisateur pour corriger les erreurs, que ce soit en modifiant ses saisies ou en comprenant les restrictions de l'application.

Réception du Message de Shutdown

Fonctionnement du Shutdown

1. **Message de Shutdown** :
    - Le serveur envoie un message spécial, typiquement formaté comme **"ERROR/SERVER_SHUTDOWN"**, pour informer les clients qu'il est en train de s'arrêter.
    - Ce message sert à alerter les clients qu'ils doivent terminer leurs opérations en cours et se préparer à se déconnecter.
2. **Traitement dans le Client** :
    - La méthode **receive_message** de **MessagingPage** écoute en permanence les messages entrants du serveur.
    - Lorsqu'un message de shutdown est reçu, la méthode identifie ce message et déclenche une série d'actions pour gérer la fermeture du client de manière ordonnée.

Actions en Réponse au Shutdown

1. **Signal de Shutdown** :
    - Un signal **shutdown_signal** est émis dès la réception du message de shutdown.
    - Ce signal est connecté à la méthode **handle_shutdown** de **MessagingPage**.
2. **Fermeture de l'Interface Utilisateur** :
    - La méthode **handle_shutdown** s'occupe de fermer proprement le socket client et de quitter l'application.
    - Elle garantit que toutes les ressources sont libérées correctement, évitant ainsi tout problème de fuite de mémoire ou de connexion résiduelle.
3. **Expérience Utilisateur lors du Shutdown** :
    - Les utilisateurs sont informés du shutdown imminent par une notification ou un changement dans l'interface, selon la façon dont la méthode **handle_shutdown** est implémentée.

- Cela permet aux utilisateurs de comprendre pourquoi l'application se ferme et de sauvegarder ou terminer leurs actions en cours.

Gestion des Demandes d'Accès aux Salons

La classe **DemandesAccesTab** joue un rôle crucial dans l'interface utilisateur du client pour la gestion des demandes d'accès aux différents salons de discussion. Voici une explication détaillée de son fonctionnement et de ses méthodes principales :

Fonctionnalité de l'Onglet

1. **Objectif Principal** :
    - Cet onglet permet aux utilisateurs d'accepter ou de refuser les demandes d'accès aux salons provenant d'autres utilisateurs.
    - Il fournit une interface intuitive pour visualiser et gérer ces demandes en temps réel.
2. **Interface Utilisateur** :
    - L'onglet contient une liste (**QListWidget**) qui affiche les demandes d'accès en attente.
    - Deux boutons, 'Accepter' et 'Refuser', permettent de répondre à ces demandes.

Méthodes Clés

1. **update_demandes_list** :
    - Cette méthode ajoute une nouvelle demande d'accès à la liste des demandes.
    - Chaque demande est affichée sous forme d'élément dans la liste, avec les détails pertinents (comme l'alias du demandeur et le nom du salon).
2. **accepter_demande** :
    - **Uniquement l'administrateur** sélectionne une demande et clique sur 'Accepter', puisqu'il est le seul à recevoir le message.
    - Elle envoie un message au serveur indiquant que la demande d'accès au salon spécifié a été acceptée.
    - La demande est ensuite retirée de la liste.
3. **refuser_demande** :
    - Cette méthode est appelée lorsque l'utilisateur refuse une demande d'accès.
    - Elle envoie un message au serveur pour notifier le refus de la demande d'accès au salon.
    - La demande est également retirée de la liste.

Expérience Utilisateur

- **Gestion des Demandes** :
    - L'administrateur peux facilement surveiller les demandes d'accès et prendre des décisions rapides, ce qui facilite la gestion des accès aux salons et renforce la sécurité et la confidentialité des discussions.

- **Interaction Simple** :
    - La conception intuitive de l'interface utilisateur rend la gestion des demandes d'accès facile et directe, même pour les utilisateurs moins expérimentés.

Gestion des Statuts des Utilisateurs

La classe **StatusPage** est un composant essentiel de l'interface utilisateur du client, destiné à afficher les statuts des utilisateurs connectés au serveur. Voici une analyse détaillée de son rôle et de ses fonctions principales :

Fonctionnalité de l'Onglet

1. **Objectif Principal** :
    - Cette page fournit une vue d'ensemble en temps réel des statuts des utilisateurs, tels que 'connecté', 'déconnecté', ou d'autres statuts personnalisés.
    - Elle aide à comprendre qui est actuellement en ligne ou indisponible.
2. **Interface Utilisateur** :
    - L'onglet comprend une liste (**QListWidget**) qui affiche les alias des utilisateurs et leurs statuts respectifs.
    - La mise en page est simple et épurée pour une lecture facile.

Méthodes Clés

1. **update_status** :
    - Cette méthode est essentielle pour actualiser la liste des statuts des utilisateurs.
    - Elle prend en entrée les données de statut sous forme de chaîne, les divise et les affiche dans la liste.
    - Chaque entrée dans la liste représente un utilisateur et son statut actuel.

Expérience Utilisateur

- **Surveillance en Temps Réel** :
    - Les utilisateurs peuvent voir en temps réel qui est en ligne, ce qui facilite la planification des communications ou des sessions de chat.
- **Clarté et Simplicité** :
    - L'affichage clair et sans encombrement assure que les utilisateurs peuvent rapidement identifier les statuts sans confusion.

Conclusion

La documentation du client pour l'application de messagerie détaille les différentes composantes de l'application, leur fonctionnement et leur interaction avec le serveur. L'accent est mis sur la facilité d'utilisation, l'organisation logique du code et la communication efficace entre le client et le serveur, le tout pour fournir une expérience utilisateur optimale.