Mode d'emploi : Documentation du développement et de la création du code client

Table des matières

I. Introduction 4
1. Objectif de l'Application de Chat 4
2. Importance de l'Interface Utilisateur Conviviale 4
3. Communication Facilitée 4
4. Accessibilité Universelle 4
5. Engagement Utilisateur 4
II. Installation 5
1. Étape 1 : Vérification de l'installation de Python 5
2. Étape 2 : Installation de PyQt5 5
III. Page d'Inscription 5
1. Labels 6
➢ Nom d'utilisateur 6
➢ Alias 6
➢ Mot de passe 6
2. Champs de Saisie 7
➢ Nom d'utilisateur 7
➢ Alias 7
➢ Mot de passe 7
3. Boutons 7
➢ S'inscrire 7
4. Conseils pour un Mot de Passe Fort 7
IV. Adhésion au salon 8
1. Processus d'Adhésion 9
➢ 1.1. Boîte de Dialogue d'Adhésion 9
➢ Méthode adherer_au_salon 9
➢ Méthode send_automatic_adhesion_request 9
2. Importance de l'Adhésion à un Salon 9
➢ Personnalisation de l'Expérience 9
➢ Communication Ciblée 9
3. Exemples d'Utilisation 9
➢ Adhésion Manuelle 9
➢ Adhésion Automatique 9
V. Thread Client 10
1. Rôle du Thread Client 11
➢ Connexion au Serveur 11
➢ Authentification 11
➢ Réception des Données Initiales 11

- Écoute des Messages 11
2. Gestion des Erreurs 11
- Erreur d'Authentification 11
- Erreurs de Communication 12
- Détection des Messages JSON 12
- Mécanismes de Reconnexion 12

VI. Interface graphique du client 12
1. Description des Éléments de l'Interface 14
- Nom d'Utilisateur et Mot de Passe 14
- Connexion et Inscription 14
- Affichage des Messages et Saisie de Message 14
- Sélection de Salon et Déconnexion 14
2. Choix Esthétiques 14
- Couleurs 14
- Polices 14
- Adaptabilité à Différentes Résolutions d'Écran 14

VII. Connexion au serveur 15
1. Étapes de Connexion 15
2. Gestion des Erreurs de Connexion 16
3. Gestion de la Connexion en Arrière-Plan 16

VIII. Envoi de messages et gestion des salons 16
1. Envoi de Messages 18
2. Affichage des Messages dans la Zone de Chat 19
3. Gestion des Salons 19
4. Scénarios d'Utilisation 20

IX. Déconnexion 20
1. Étapes de Déconnexion 20
2. Gestion des Ressources après la Déconnexion 21

I. Introduction

Bienvenue dans la documentation complète de l'application de chat, un outil innovant conçu pour simplifier et améliorer vos expériences de communication en ligne. Cette application a été développée dans le but de fournir une plateforme conviviale et efficace pour les utilisateurs qui souhaitent échanger des messages en temps réel dans divers salons.

1. Objectif de l'Application de Chat

L'objectif principal de cette application de chat est de faciliter la communication instantanée entre les utilisateurs au sein de salons virtuels. Que vous soyez un utilisateur régulier cherchant à discuter avec des amis ou un professionnel collaborant avec des collègues, cette application offre une plateforme dynamique pour répondre à vos besoins de communication.

2. Importance de l'Interface Utilisateur Conviviale

Une interface utilisateur conviviale est cruciale pour garantir une expérience utilisateur positive. Dans le contexte d'une application de chat, cela revêt une importance particulière, car la facilité d'utilisation et la clarté de l'interface sont essentielles pour une communication fluide.

3. Communication Facilitée

Une interface utilisateur bien conçue simplifie le processus de communication, permettant aux utilisateurs de se concentrer sur l'échange d'idées plutôt que sur la compréhension de l'outil. Des fonctionnalités intuitives, des boutons bien placés et une navigation sans heurts sont autant d'éléments qui contribuent à une communication facilitée.

4. Accessibilité Universelle

Une interface conviviale rend l'application accessible à un large éventail d'utilisateurs, qu'ils soient novices en technologie ou experts. Les icônes explicites, les menus clairs et les indications visuelles contribuent à une expérience utilisateur positive pour tous les utilisateurs, quel que soit leur niveau de compétence.

5. Engagement Utilisateur

Une interface utilisateur bien pensée favorise l'engagement des utilisateurs. Des éléments visuels attrayants, des animations subtiles et une disposition logique des éléments incitent les utilisateurs à explorer davantage l'application, à participer activement aux discussions et à en faire un outil central dans leur quotidien.

Dans les sections suivantes de cette documentation, nous explorerons en détail les différentes fonctionnalités de l'application, en mettant l'accent sur l'installation, l'utilisation des salons, l'interaction avec l'interface graphique, etc…

II. Installation

L'installation de PyQt5 est une étape cruciale pour assurer le bon fonctionnement de l'application de chat. Si PyQt5 n'est pas déjà installé sur votre système, suivez ces étapes détaillées pour garantir une installation et pouvoir commencer à développer l'application

1. Étape 1 : Vérification de l'installation de Python

Avant d'installer PyQt5, assurez-vous d'avoir Python installé sur votre machine. Si ce n'est pas le cas, il faut le télécharger et l'installer ici : [Welcome to Python.org]

2. Étape 2 : Installation de PyQt5

Utilisez la commande suivante pour installer PyQt5 à l'aide de pip (le gestionnaire de paquets Python) :

```
pip install PyQt5
```

III. Page d'Inscription

```
class InscriptionDialog(QDialog):
    def __init__(self, server_address):
        """
        Initialise la boîte de dialogue d'inscription.
        Args:
            server_address (tuple): L'adresse du serveur auquel l'utilisateur souhaite
s'inscrire.
        super().__init__()
        self.setWindowTitle("Page d'inscription")
        self.setMinimumWidth(300)

        self.username_label = QLabel("Nom d'utilisateur:")
        self.username_input = QLineEdit()

        self.alias_label = QLabel("Alias:")
        self.alias_input = QLineEdit()

        self.password_label = QLabel("Mot de passe:")
        self.password_input = QLineEdit()
        self.password_input.setEchoMode(QLineEdit.Password)

        self.inscription_button = QPushButton("S'inscrire")
        self.inscription_button.clicked.connect(self.inscrire)

        self.server_address = server_address

        layout = QVBoxLayout()
        layout.addWidget(self.username_label)
        layout.addWidget(self.username_input)
        layout.addWidget(self.alias_label)
```

```
        layout.addWidget(self.alias_input)
        layout.addWidget(self.password_label)
        layout.addWidget(self.password_input)
        layout.addWidget(self.inscription_button)
        self.setLayout(layout)

    def inscrire(self):
        """
        Méthode appelée lors du clic sur le bouton d'inscription.
        """
        username = self.username_input.text().strip()
        alias = self.alias_input.text().strip()
        password = self.password_input.text().strip()

        inscription_request = f"INSCRIPTION:{username}:{alias}:{password}"
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
            try:
                client_socket.connect(self.server_address)
                client_socket.send(inscription_request.encode('utf-8'))

                inscription_response = client_socket.recv(1024).decode('utf-8')
                print(inscription_response)
                QMessageBox.information(self, "Résultat de l'inscription", inscription_response)

                if inscription_response == "Inscription réussie!":
                    adhesion_request = f'JOIN:Général'
                    client_socket.send(adhesion_request.encode('utf-8'))
                    self.accept()
            except Exception as e:
                print(f"Erreur lors de l'envoi de la demande d'inscription : {e}")
```

1. Labels

- Nom d'utilisateur

Rôle : Indiquez à l'utilisateur qu'il doit saisir un nom d'utilisateur.

Conseil : Choisissez un nom d'utilisateur unique qui vous identifie sans révéler d'informations sensibles.

- Alias

Rôle : Demandez à l'utilisateur de fournir un alias qui sera visible par les autres utilisateurs.

Conseil : Utilisez un alias qui reflète votre personnalité sans divulguer trop d'informations personnelles.

- Mot de passe

Rôle : Incitez l'utilisateur à créer un mot de passe sécurisé.

Conseil : Un bon mot de passe doit contenir une combinaison de lettres majuscules et minuscules, de chiffres et de caractères spéciaux. Évitez d'utiliser des informations personnelles évidentes.

2. Champs de Saisie

➢ Nom d'utilisateur

Type : Zone de texte (QLineEdit).

Utilisation : L'utilisateur saisit son nom d'utilisateur.

➢ Alias

Type : Zone de texte (QLineEdit).

Utilisation : L'utilisateur saisit l'alias qu'il souhaite utiliser.

➢ Mot de passe

Type : Zone de texte masquée (QLineEdit avec EchoMode.Password).

Utilisation : L'utilisateur entre un mot de passe sécurisé.

3. Boutons

➢ S'inscrire

Rôle : Bouton pour déclencher le processus d'inscription.

Utilisation : Lorsque l'utilisateur a rempli les champs, il clique sur ce bouton pour soumettre ses informations.

4. Conseils pour un Mot de Passe Fort

- Utilisez une combinaison de lettres majuscules et minuscules.
- Intégrez des chiffres et des caractères spéciaux.
- Évitez les mots de passe évidents, tels que des informations personnelles.
- Assurez-vous qu'il a une longueur suffisante (au moins 8 caractères).
- N'utilisez pas le même mot de passe sur plusieurs sites.

IV. Adhésion au salon

L'adhésion à un salon est une fonctionnalité cruciale qui permet aux utilisateurs de rejoindre des espaces de discussion spécifiques en fonction de leurs intérêts. Cette section expliquera en détail le processus d'adhésion, son importance et fournira des exemples pratiques d'utilisation.

```python
class AdhesionDialog(QDialog):
    def __init__(self, salon_name, client):
        """
        Initialise la boîte de dialogue d'adhésion.
        Args:
            salon_name (str): Le nom du salon auquel l'utilisateur souhaite adhérer.
            client (Client): L'instance du client associée à cette boîte de dialogue.
        """
        super().__init__()
        self.setWindowTitle("Adhésion au Salon")
        self.salon_name = salon_name
        self.server_response = None
        self.client = client
        label = QLabel(f"Voulez-vous adhérer au salon {salon_name}?")
        join_button = QPushButton("Oui")
        join_button.clicked.connect(self.adherer_au_salon)
        cancel_button = QPushButton("Non")
        cancel_button.clicked.connect(self.reject)
        layout = QVBoxLayout()
        layout.addWidget(label)
        layout.addWidget(join_button)
        layout.addWidget(cancel_button)
        self.setLayout(layout)

    def adherer_au_salon(self):
        """
        Méthode appelée lors du clic sur le bouton "Oui" pour adhérer au salon.
        """
        formatted_request = f'JOIN:{self.salon_name}'
        self.server_response = formatted_request
        self.accept()

    def send_automatic_adhesion_request(self):
        """
        Méthode pour envoyer une demande d'adhésion automatique à Blabla (ou un autre salon).
        """
        blabla_request = f'JOIN:Blabla'
        self.client.socket.send(blabla_request.encode('utf-8'))
```

1. Processus d'Adhésion

➢ 1.1. Boîte de Dialogue d'Adhésion

La boîte de dialogue d'adhésion est une interface utilisateur simple qui informe l'utilisateur du salon auquel il est sur le point d'adhérer. Elle comporte deux boutons, "Oui" et "Non".

➢ Méthode adherer_au_salon

Lorsque l'utilisateur clique sur le bouton "Oui", la méthode adherer_au_salon est déclenchée. Cette méthode formate la demande d'adhésion et la transmet au serveur.

➢ Méthode send_automatic_adhesion_request

En plus de l'adhésion manuelle, il peut y avoir des scénarios où l'utilisateur souhaite s'abonner automatiquement à certains salons lors de son inscription. La méthode send_automatic_adhesion_request illustre cette fonctionnalité en envoyant automatiquement une demande d'adhésion à un salon spécifique (par exemple, "Général").

2. Importance de l'Adhésion à un Salon

➢ Personnalisation de l'Expérience

En adhérant à des salons spécifiques, les utilisateurs peuvent personnaliser leur expérience en se connectant uniquement aux espaces de discussion qui les intéressent.

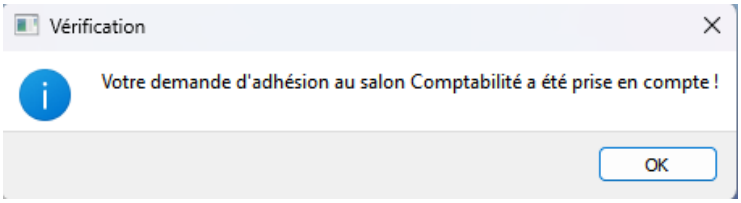
➢ Communication Ciblée

L'adhésion à un salon permet une communication plus ciblée. Les utilisateurs peuvent participer à des discussions pertinentes à leurs centres d'intérêt plutôt que d'être submergés par un flux constant de messages.

3. Exemples d'Utilisation

➢ Adhésion Manuelle

Supposons qu'un utilisateur découvre un salon appelé "Marketing" et souhaite rejoindre ce salon. Après l'avoir sélectionné dans la liste, la boîte de dialogue d'adhésion s'affiche. En cliquant sur "Oui", l'utilisateur demande à rejoindre le salon et une requête est envoyée au server qui va décider si oui ou non il accepte l'utilisateur dans le salon.



➢ Adhésion Automatique

Au moment de l'inscription, l'utilisateur peut choisir de s'abonner automatiquement à des salons spécifiques sans vérification. Par exemple, si l'utilisateur sélectionne le salon "Blabla", ça déclenche automatiquement une demande d'adhésion (si bouton oui pressé) à ce salon et place l'utilisateur dans la base de données dans la table salons_adheres.

V. Thread Client

Le thread client joue un rôle essentiel dans l'application en assurant la communication entre le client et le serveur de chat. Cette section explorera en détail le fonctionnement du thread client, sa gestion des erreurs, et fournira des conseils de débogage pour les développeurs.

```
class Client(QThread):
    message_received = pyqtSignal(str)
    subscribed_salons_received = pyqtSignal(list)
    def __init__(self, username, host, port, password):
        """
        Initialise le client.
        Args:
            username (str): Le nom d'utilisateur du client.
            host (str): L'adresse IP du serveur.
            port (int): Le numéro de port du serveur.
            password (str): Le mot de passe du client.
        """
        super().__init__()
        self.username = username
        self.host = host
        self.port = port
        self.password = password
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    def run(self):
        """
        Méthode principale pour exécuter le thread client.
        """
        try:
            self.socket.connect((self.host, self.port))
            credentials = f"{self.username}:{self.password}"
            self.socket.send(credentials.encode('utf-8'))
            initial_response = self.socket.recv(1024).decode('utf-8')
            if initial_response == "CONNECTED":
                print(f"Connexion établie avec {self.username}.")

                salons_data = self.socket.recv(1024).decode('utf-8')
                self.message_received.emit(salons_data)

                subscribed_salons_data = self.socket.recv(1024).decode('utf-8')
                subscribed_salons = json.loads(subscribed_salons_data)
                self.subscribed_salons_received.emit(subscribed_salons)

                while not self.isInterruptionRequested():
                    data = self.socket.recv(1024).decode('utf-8')

                    if not data:
                        break
```

```
                    if data == "AUTH_FAILED":
                        print("L'authentification a échoué. Veuillez vérifier votre nom
d'utilisateur et mot de passe.")
                        break

                    if data.startswith('{') and data.endswith('}'):
                        try:
                            json_data = json.loads(data)
                            self.message_received.emit(json_data)
                        except json.decoder.JSONDecodeError as e:
                            print(f"Erreur lors du décodage JSON : {e}")
                    else:
                        self.message_received.emit(data)

        except Exception as e:
            print(f"Erreur dans le client : {e}")
        finally:
            self.socket.close()
```

1. Rôle du Thread Client

Le thread client est responsable de plusieurs tâches cruciales pour le bon fonctionnement de l'application :

➢ Connexion au Serveur

Lorsque le thread client démarre, il établit une connexion avec le serveur en utilisant l'adresse IP et le numéro de port fournis.

➢ Authentification

Une fois connecté, le client envoie ses informations d'identification (nom d'utilisateur et mot de passe) au serveur pour l'authentification.

➢ Réception des Données Initiales

Après une authentification réussie, le client reçoit les données initiales, telles que la liste des salons disponibles et les salons auxquels l'utilisateur est déjà abonné.

➢ Écoute des Messages

Le thread client reste à l'écoute des messages du serveur pendant toute la durée de son exécution.

2. Gestion des Erreurs

La gestion des erreurs est essentielle pour assurer la robustesse de l'application. Voici comment le thread client gère les erreurs :

➢ Erreur d'Authentification

Si l'authentification échoue, le client affiche un message indiquant que l'authentification a échoué, s'assurant ainsi que l'utilisateur est informé du problème.

➢ Erreurs de Communication

Toute erreur survenue pendant la communication avec le serveur est capturée, affichée dans la console, et le thread client se termine. Cela garantit que les problèmes de communication sont traités de manière adéquate.

➢ Détection des Messages JSON

Le client tente de décoder les messages reçus du serveur en tant que JSON. En cas d'échec du décodage, une exception est gérée et un message d'erreur est affiché.

➢ Mécanismes de Reconnexion

Le thread client est conçu pour fonctionner de manière continue et se reconnecter en cas de déconnexion inattendue. La boucle principale gère la reconnexion de manière transparente en rétablissant la connexion avec le serveur.

VI. Interface graphique du client

```
class ClientGUI(QWidget):
    def __init__(self):
        """
        Initialise l'interface graphique du client.
        """
        super().__init__()
        self.setWindowTitle("Server de Chat")
        self.setMinimumWidth(500)
        self.client = None

        self.username_label = QLabel("Nom d'utilisateur:")
        self.username_input = QLineEdit()

        self.password_label = QLabel("Mot de passe:")
        self.password_input = QLineEdit()
        self.password_input.setEchoMode(QLineEdit.Password)

        self.connect_button = QPushButton("Se connecter")
        self.connect_button.clicked.connect(self.connect_to_server)

        self.inscription_button = QPushButton("S'inscrire")
        self.inscription_button.clicked.connect(self.afficher_page_inscription)

        self.chat_display = QTextEdit()
        self.message_input = QLineEdit()

        self.salon_label = QLabel("Sélectionnez un salon:")
        self.salon_combobox = QComboBox()

        self.send_button = QPushButton("Envoyer")
        self.send_button.clicked.connect(self.send_message)

        self.disconnect_button = QPushButton("Se déconnecter")
```

```
        self.disconnect_button.clicked.connect(self.disconnect_from_server)

        layout = QGridLayout()
        layout.addWidget(self.username_label, 0, 0)
        layout.addWidget(self.password_label, 0, 1)
        layout.addWidget(self.salon_label, 0, 3)
        layout.addWidget(self.username_input, 1, 0)
        layout.addWidget(self.password_input, 1, 1)
        layout.addWidget(self.salon_combobox, 1, 3)
        layout.addWidget(self.connect_button, 2, 0, 1, 2)
        layout.addWidget(self.inscription_button, 2, 2)
        layout.addWidget(self.chat_display, 3, 0, 1, 3)
        layout.addWidget(self.message_input, 4, 0, 1, 2)
        layout.addWidget(self.send_button, 4, 2)
        layout.addWidget(self.disconnect_button, 4, 3)
        self.setLayout(layout)

        self.client = None
        self.salon_combobox.currentIndexChanged.connect(self.update_salon_display)
        self.salons_adheres = set()
        self.set_gradient_background()
        self.subscribed_salons = []

    def set_gradient_background(self):
        """
        Applique un dégradé en tant que fond pour l'interface graphique.
        """
        palette = self.palette()
        gradient_color1 = QColor(173, 216, 230)
        gradient_color2 = QColor(0, 0, 128)

        gradient = QLinearGradient(0, 0, 0, self.height())
        gradient.setColorAt(0, gradient_color1)
        gradient.setColorAt(1, gradient_color2)

        brush = QBrush(gradient)
        palette.setBrush(QPalette.Window, brush)
        self.setPalette(palette)

        self.setStyleSheet("color: black;")

        font = QFont()
        font.setPointSize(15)
        self.setFont(font)

        self.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)
```

L'interface graphique du client est une composante cruciale de l'application, permettant à l'utilisateur d'interagir de manière conviviale avec le serveur de chat. Cette section détaille chaque élément de l'interface, explique les choix esthétiques et fonctionnels, et aborde l'adaptabilité de l'interface à différentes résolutions d'écran.

1. Description des Éléments de l'Interface

➢ Nom d'Utilisateur et Mot de Passe

username_label et username_input : Ces éléments permettent à l'utilisateur d'entrer son nom d'utilisateur. Ils sont cruciaux pour l'identification.

password_label et password_input : Ces éléments sont destinés à la saisie du mot de passe, offrant une couche de sécurité.

➢ Connexion et Inscription

connect_button : Ce bouton lance le processus de connexion au serveur.

inscription_button : Permet à l'utilisateur de passer à la page d'inscription pour créer un nouveau compte.

➢ Affichage des Messages et Saisie de Message

chat_display : La zone de texte où les messages du salon sélectionné sont affichés.

message_input et send_button : Ils permettent à l'utilisateur de saisir et d'envoyer des messages respectivement.

➢ Sélection de Salon et Déconnexion

salon_label et salon_combobox : Ces éléments permettent de choisir le salon auquel l'utilisateur souhaite participer.

disconnect_button : Permet à l'utilisateur de se déconnecter du serveur.

2. Choix Esthétiques

➢ Couleurs

Le dégradé du fond utilise des nuances de bleu, favorisant une esthétique apaisante et liée à la communication.

Le texte en noir assure une lisibilité optimale sur le fond coloré.

➢ Polices

Une police de taille 15 est choisie pour assurer une lecture confortable sans paraître trop imposante.

➢ Adaptabilité à Différentes Résolutions d'Écran

L'interface est conçue pour s'adapter à diverses résolutions d'écran grâce à plusieurs caractéristiques :

setSizePolicy : L'interface a une politique de redimensionnement extensible, permettant une adaptation harmonieuse.

Utilisation de QGridLayout : Les éléments sont disposés dans une grille, facilitant le réarrangement en fonction de l'espace disponible.

Palette de Dégradé : Le dégradé de couleur s'étend sur toute la hauteur de l'interface, assurant une apparence cohérente même avec des résolutions différentes.

VII. Connexion au serveur

La connexion au serveur est une étape cruciale dans l'utilisation de l'application de chat. La méthode connect_to_server de la classe ClientGUI gère ce processus. Cette section détaille les étapes de connexion, fournit des conseils sur la gestion des erreurs et explique comment l'application gère la connexion en arrière-plan tout en maintenant une interaction utilisateur fluide.

```
def connect_to_server(self):
    """
    Méthode pour se connecter au serveur.
    """
    if not self.client:
        username = self.username_input.text().strip()
        password = self.password_input.text().strip()
        if username and password:
            self.client = Client(username, 'localhost', 5000, password)
            self.client.message_received.connect(self.populate_salons)
            self.client.subscribed_salons_received.connect(self.update_subscribed_salons)
            self.client.start()
            self.setWindowTitle(f"Server de Chat - {username}")
        else:
            print("Veuillez entrer un nom d'utilisateur.")
```

1. Étapes de Connexion

La méthode connect_to_server s'exécute lorsque l'utilisateur clique sur le bouton "Se connecter". Voici les étapes détaillées :

- Vérification Client Existante : La méthode vérifie d'abord si une instance de client existe déjà. Si oui, la connexion n'est pas répétée.
- Récupération des Identifiants : Elle récupère le nom d'utilisateur et le mot de passe saisis par l'utilisateur depuis les champs correspondants.
- Création d'une Instance Client : Si des identifiants sont fournis, une nouvelle instance de la classe Client est créée avec ces informations, l'adresse du serveur ('localhost' avec le port 5000 dans cet exemple) et le mot de passe.
- Connexion des Signaux : Les signaux émis par le client, tels que message_received et subscribed_salons_received, sont connectés à des méthodes de mise à jour de l'interface utilisateur.
- Démarrage du Thread Client : Le thread client est démarré en appelant la méthode start().
- Mise à Jour du Titre de la Fenêtre : Si tout se passe bien, le titre de la fenêtre est mis à jour avec le nom d'utilisateur.

2. Gestion des Erreurs de Connexion

La méthode connect_to_server gère également les erreurs potentielles lors de la connexion :

- Validation des Identifiants : Avant de créer un client, la méthode vérifie que le nom d'utilisateur et le mot de passe ne sont pas vides.
- Affichage d'un Message d'Erreur : Si des erreurs sont détectées, un message est imprimé dans la console indiquant à l'utilisateur d'entrer un nom d'utilisateur.

3. Gestion de la Connexion en Arrière-Plan

L'utilisation d'un thread client (Client étendant QThread) permet à l'application de maintenir la connexion en arrière-plan tout en restant réactive à l'interaction de l'utilisateur. Ceci est réalisé grâce à la méthode run du thread, qui écoute en permanence les messages du serveur.

VIII. Envoi de messages et gestion des salons

```
def populate_salons(self, salons_data):
        """
        Méthode pour peupler la liste des salons.

        Args:
            salons_data (str): Les données sur les salons sous forme de chaîne JSON.
        """
        try:
            if salons_data:
                try:
                    salons = json.loads(salons_data)
                    if isinstance(salons, list) and salons:
                        if all(isinstance(salon, list) and len(salon) >= 2 for salon in salons):
                            self.salon_combobox.clear()
                            self.salon_combobox.addItems([str(salon[1]) for salon in salons])
                except json.decoder.JSONDecodeError as e:
                    self.display_message(salons_data)
        except Exception as ex:
            print(f"Error: {ex}")

    def update_subscribed_salons(self, subscribed_salons):
        """
        Méthode pour mettre à jour la liste des salons auxquels l'utilisateur est abonné.

        Args:
            subscribed_salons (list): La liste des salons auxquels l'utilisateur est abonné.
        """
        print("Salons adhérés:", subscribed_salons)
        self.subscribed_salons = [salon['salon_name'] for salon in subscribed_salons]

    def check_adhesion(self, selected_salon):
        """
        Vérifie si l'utilisateur est déjà abonné au salon sélectionné.
```

```
        Args:
            selected_salon (str): Le nom du salon sélectionné.
        """
        if selected_salon.lower() in (salon.lower() for salon in self.subscribed_salons):
            QMessageBox.information(self, "Déjà abonné", f"Vous êtes déjà abonné au salon
{selected_salon}.")
        else:
            adhesion_dialog = AdhesionDialog(selected_salon, self.client)
            result = adhesion_dialog.exec_()

            if result == QDialog.Accepted:
                formatted_request = adhesion_dialog.server_response
                print(f"Sending request to join salon: {formatted_request}")
                self.client.socket.send(formatted_request.encode('utf-8'))
                QMessageBox.information(self, "Vérification", f"Votre demande d'adhésion au salon
{selected_salon} a été prise en compte !")

    def update_salon_display(self):
        """
        Met à jour l'affichage en fonction du salon sélectionné.
        """
        selected_salon = self.salon_combobox.currentText()

        if selected_salon == "Général":
            return

        if selected_salon.lower() == "blabla":
            self.check_adhesion(selected_salon)
        elif selected_salon not in self.subscribed_salons:
            adhesion_dialog = AdhesionDialog(selected_salon, self.client)
            result = adhesion_dialog.exec_()

            if result == QDialog.Accepted:
                adhesion_dialog.send_automatic_adhesion_request()
                self.subscribed_salons.append(selected_salon)
                formatted_request = adhesion_dialog.server_response
                self.client.socket.send(formatted_request.encode('utf-8'))
                QMessageBox.information(self, "Vérification", f"Votre demande d'adhésion au salon
{selected_salon} a été prise en compte !")

        print(self.subscribed_salons)
        if selected_salon in self.subscribed_salons:
            self.message_input.setDisabled(False)
            self.send_button.setDisabled(False)
        else:
            QMessageBox.warning(self, "Accès refusé", f"Vous n'êtes pas abonné au salon
{selected_salon}. L'accès est refusé.")
            self.message_input.setDisabled(True)
```

```
            self.send_button.setDisabled(True)
    def send_message(self):
        """
        Envoie un message au salon sélectionné.
        """
        if self.client:
            username = self.username_input.text().strip()
            if username:
                message = self.message_input.text().strip()
                if message:
                    salon = self.salon_combobox.currentText()
                    formatted_message = f"MSG ({salon}): {message}"
                    self.client.socket.send(formatted_message.encode('utf-8'))
                    self.message_input.clear()
                else:
                    print("Veuillez entrer un message.")
            else:
                print("Veuillez vous connecter d'abord.")

    def display_message(self, message):
        """
        Affiche un message dans la zone de chat si l'utilisateur est abonné au salon.
        Args:
            message (str): Le message à afficher.
        """
        if self.is_user_subscribed_to_current_salon():
            self.chat_display.append(message)

    def is_user_subscribed_to_current_salon(self):
        """
        Vérifie si l'utilisateur est abonné au salon actuellement sélectionné.
        Returns:
            bool: True si l'utilisateur est abonné, False sinon.
        """
        selected_salon = self.salon_combobox.currentText()
        return any(salon.lower() == selected_salon.lower() for salon in self.subscribed_salons)
```

1. Envoi de Messages

La classe ClientGUI gère l'envoi de messages au serveur. Voici comment le processus fonctionne :

- Méthode send_message : Lorsque l'utilisateur clique sur le bouton d'envoi (send_button), la méthode send_message est appelée.
- Validation de l'Utilisateur : Elle vérifie d'abord si un nom d'utilisateur est saisi. Si non, elle imprime un message dans la console indiquant à l'utilisateur de se connecter d'abord.

- Formatage du Message : Le message est récupéré à partir du champ de saisie (message_input) et le nom du salon actuellement sélectionné est ajouté au message sous forme d'en-tête (MSG (nom_du_salon): message).
- Envoi au Serveur : Le message formaté est ensuite envoyé au serveur via la socket du client.
- Effacement du Champ de Saisie : Une fois le message envoyé, le champ de saisie est effacé pour permettre la saisie d'un nouveau message.
- Affichage Local : Le message n'est pas directement ajouté à la zone de chat (chat_display) ici. Il est seulement affiché localement si l'utilisateur est abonné au salon actuellement sélectionné.

2. Affichage des Messages dans la Zone de Chat

- La méthode display_message affiche un message dans la zone de chat si l'utilisateur est abonné au salon en cours. Elle est appelée lorsque de nouveaux messages sont reçus du serveur.
- Vérification de l'Abonnement : La méthode vérifie d'abord si l'utilisateur est abonné au salon actuellement sélectionné.
- Ajout du Message : Si l'utilisateur est abonné, le message est ajouté à la zone de chat.

3. Gestion des Salons

La gestion des salons est un aspect important de l'application de chat. Voici comment elle est réalisée dans la classe ClientGUI :

- Peuplement Initial : La méthode populate_salons est responsable du peuplement initial de la liste des salons. Elle est appelée lors de la connexion initiale au serveur.
  - Les données des salons sont fournies sous forme de chaîne JSON et sont analysées.
  - La liste des salons est ensuite filtrée pour s'assurer qu'elle est correctement formatée.
  - Les noms des salons sont extraits et ajoutés à la liste déroulante (salon_combobox).
- Mise à Jour des Salons Abonnés : La méthode update_subscribed_salons est appelée pour mettre à jour la liste des salons auxquels l'utilisateur est abonné.
  - La liste mise à jour est imprimée dans la console.
- Vérification de l'Adhésion : La méthode check_adhesion vérifie si l'utilisateur est déjà abonné au salon sélectionné. Si oui, un message informatif est affiché. Sinon, une boîte de dialogue d'adhésion est ouverte.
- Mise à Jour de l'Affichage du Salon : La méthode update_salon_display est appelée lorsqu'un utilisateur change de salon.
  - Si l'utilisateur sélectionne le salon "Général", aucune action n'est entreprise.
  - Si le salon sélectionné est "Blabla", la méthode check_adhesion est appelée.
  - Sinon, si l'utilisateur n'est pas abonné, une boîte de dialogue d'adhésion est ouverte automatiquement.
- Vérification des Abonnements : La méthode is_user_subscribed_to_current_salon vérifie si l'utilisateur est abonné au salon actuellement sélectionné.

- Réactions en Fonction des Abonnements : En fonction des résultats de la vérification, l'interface utilisateur est mise à jour.
  - Si l'utilisateur est abonné, le champ de saisie et le bouton d'envoi sont activés.
  - Sinon, un avertissement est affiché, et le champ de saisie et le bouton d'envoi sont désactivés.

4. Scénarios d'Utilisation

Adhésion à un Nouveau Salon : Lorsqu'un utilisateur choisit un salon auquel il n'est pas encore abonné, il est invité à rejoindre automatiquement ce salon. Une demande d'adhésion est envoyée au serveur.

Affichage Sélectif des Messages : Les messages ne sont affichés localement que si l'utilisateur est abonné au salon actuel, ce qui permet une expérience de chat plus personnalisée.

Contrôle d'Accès : Si un utilisateur n'est pas abonné à un salon, l'accès au champ de saisie et au bouton d'envoi est désactivé, évitant ainsi les messages indésirables.

IX. Déconnexion

```
    def disconnect_from_server(self):
        """
        Déconnecte le client du serveur.
        """
        if self.client:
            self.client.socket.close()
            self.client = None
            self.chat_display.clear()
            self.salon_combobox.clear()
            self.salons_adheres.clear()

    def closeEvent(self, event):
        """
        Méthode appelée lors de la fermeture de l'application.

        Args:
            event (QCloseEvent): L'événement de fermeture.
        """
        if self.client:
            self.client.socket.close()
        event.accept()
```

1. Étapes de Déconnexion

La méthode disconnect_from_server gère la déconnexion du client du serveur. Voici les étapes de déconnexion :

Vérification du Client : La méthode vérifie d'abord si le client est actuellement connecté.

Fermeture de la Socket : Si le client est connecté, sa socket est fermée à l'aide de la méthode close. Cela interrompt la connexion avec le serveur.

Nettoyage des Données Locales : Après la fermeture de la socket, certaines données locales sont réinitialisées :

- La liste des salons (salon_combobox) est effacée.
- La zone de chat (chat_display) est effacée.
- La liste des salons auxquels l'utilisateur est abonné (salons_adheres) est également vidée.
- Définition du Client à None : La variable self.client est ensuite définie à None pour indiquer qu'aucune connexion n'est actuellement active.

2. Gestion des Ressources après la Déconnexion

La méthode closeEvent est appelée lors de la fermeture de l'application. Elle s'assure que la socket est correctement fermée même si l'utilisateur ferme l'application de manière inattendue. Voici comment cela fonctionne :

- Vérification du Client : La méthode vérifie si le client est actuellement connecté.
- Fermeture de la Socket : Si le client est connecté, sa socket est fermée.
- Acceptation de l'Événement de Fermeture : L'événement de fermeture est accepté, permettant à l'application de se fermer.