



**UNIVERSITÉ
DE LORRAINE**



**Master EEA - Parcours Intelligence - Mesures
Énergétiques pour les Énergies Nouvelles**

Mini Projet Réseau

Par

MADANI Mohamed & JAMOUMI NOURA

Sous le thème

**Développement d'un mini
chat client-serveur en TCP**

Enseignant : Yann Morère

Année : 2023/2024

Sommaire

- 1 Table des matières
- 1. Introduction 3
- 2 Recherche Technique et Bibliographique 4
 - 2.1 Sélection des Outils 4
 - 2.2 Langages de Programmation 5
 - 2.3 Bibliothèques Graphiques 7
 - 2.4 Bibliothèques Réseau 8
- 3 Développement..... 9
 - 3.1 Architecture 9
 - 3.2 Implémentation..... 11
 - 3.3 Défis et Solutions 12
- 4 Conclusion..... 13
- ANNEXE 14

1. Introduction

Ce projet se concentre sur la conception et la réalisation d'un système de communication innovant basé sur le modèle client-serveur. Ce modèle, fondamental dans l'univers de la connectivité numérique, permet à plusieurs utilisateurs de dialoguer en temps réel au travers d'échanges textuels synchronisés.

Le concept repose sur un serveur centralisé qui orchestre les connexions et assure la transmission fluide des messages entre les clients. Ces derniers se connectent au serveur pour interagir, créant ainsi un espace de communication collaboratif et dynamique.

Les objectifs de ce projet sont multiples et ambitieux. Ils incluent la création d'un système robuste et sécurisé, capable de gérer de manière optimale les interactions multiples, la distribution efficace des messages et offrant une expérience utilisateur intuitive et enrichissante. Nous aspirons également à intégrer des fonctionnalités avancées telles que la gestion des salons de discussion, le support multimédia et éventuellement des mesures de sécurité renforcées telles que le chiffrement pour garantir la confidentialité des échanges.

Sur le plan technologique, nous prévoyons d'utiliser des langages de programmation comme le C pour le développement du serveur, le Python pour des scripts complémentaires et des bibliothèques comme libuv pour la gestion asynchrone des opérations réseau. Nous envisageons également d'exploiter des technologies web pour créer une interface utilisateur moderne et adaptable à différents appareils.

En synthèse, ce projet ambitionne de mettre en place un système de communication client-serveur sophistiqué, offrant une plateforme d'échange interactive et évolutive, répondant ainsi aux attentes et besoins des utilisateurs dans le contexte actuel de la communication numérique.

2 Recherche Technique et Bibliographique

2.1 Sélection des Outils

- **Système d'Exploitation :**

Raisons de l'utilisation de Linux :

Linux est un système d'exploitation open source largement utilisé dans le développement logiciel pour plusieurs raisons :

Open Source et Gratuit : Linux est distribué sous une licence open source, ce qui signifie qu'il peut être utilisé gratuitement. Cela réduit les coûts pour les développeurs individuels et les entreprises.

Stabilité et Fiabilité : Linux est réputé pour sa stabilité et sa fiabilité. Il est utilisé dans de nombreux environnements de production critiques, notamment les serveurs web et les centres de données.

Personnalisable : Linux offre une grande souplesse et peut être entièrement personnalisé pour répondre aux besoins spécifiques des développeurs. Il existe de nombreuses distributions Linux adaptées à différents cas d'utilisation.

Performance : Linux est conçu pour être performant, même sur du matériel moins puissant. Cela en fait un excellent choix pour le développement sur une gamme de machines, y compris les ordinateurs portables et les serveurs.

Soutien de la Communauté : La communauté open source qui entoure Linux offre un vaste ensemble de ressources, de forums de discussion et de documentation. Cela facilite le dépannage des problèmes et l'apprentissage pour les développeurs.

Environnement de Développement (IDE) :

Comparaison des IDE populaires :

Les environnements de développement intégrés (IDE) offrent une suite d'outils pour faciliter le processus de développement logiciel. Voici une comparaison de quelques-uns des IDE les plus populaires :

Visual Studio Code (VS Code) :

Développé par Microsoft, VS Code est un IDE léger et extensible.

Il prend en charge de nombreux langages de programmation et offre une vaste gamme d'extensions pour personnaliser l'expérience de développement.

VS Code est particulièrement apprécié pour son intégration avec Git, son terminal intégré et son système de débogage.

PyCharm :

PyCharm est spécifiquement conçu pour le développement en Python, bien qu'il prenne également en charge d'autres langages.

Il offre des fonctionnalités avancées telles que l'analyse statique, la refactorisation du code et le débogage.

PyCharm est apprécié pour sa facilité d'utilisation et sa compatibilité avec les principaux frameworks Python tels que Django et Flask.

IntelliJ IDEA :

IntelliJ IDEA est un IDE polyvalent pour le développement Java et Kotlin, mais il prend également en charge d'autres langages grâce à des plugins.

Il offre des fonctionnalités avancées telles que l'achèvement du code, l'analyse de la qualité du code et les outils de refactorisation.

IntelliJ IDEA est souvent utilisé dans les projets Java de grande envergure en raison de ses capacités de gestion de projet avancées.

Eclipse :

Eclipse est un IDE open source populaire, principalement utilisé pour le développement Java, bien qu'il prenne également en charge d'autres langages via des plugins.

Il est apprécié pour sa grande flexibilité et son écosystème de plugins étendu.

Eclipse est souvent utilisé dans les environnements de développement d'applications d'entreprise.

2.2 Langages de Programmation

Choix des Langages de Programmation :

Langages possibles :

C/C++ :

C et C++ sont des langages de programmation populaires, largement utilisés dans le développement système, le développement d'applications embarquées, les jeux vidéo, etc.

Ces langages offrent un contrôle précis sur le matériel et sont privilégiés pour les applications nécessitant des performances maximales et une faible consommation de ressources.

Python :

Python est un langage de programmation interprété, polyvalent et facile à apprendre.

Il est utilisé dans de nombreux domaines, notamment le développement web, le traitement des données, l'intelligence artificielle, l'automatisation, etc.

Python est réputé pour sa lisibilité et sa syntaxe claire, ce qui en fait un excellent choix pour les débutants en programmation.

Java :

Java est un langage de programmation orienté objet largement utilisé pour le développement d'applications d'entreprise, les applications Android, les applications web, etc.

Il est apprécié pour sa portabilité, sa sécurité et sa robustesse, grâce à la machine virtuelle Java (JVM).

Java bénéficie également d'une vaste bibliothèque standard et d'un écosystème de frameworks et d'outils bien établis.

Critères de Sélection :

1. Facilité d'utilisation :

C/C++ : Ces langages peuvent être plus complexes à apprendre en raison de leur syntaxe et de leur gestion manuelle de la mémoire.

Python : Python est réputé pour sa simplicité et sa lisibilité, ce qui en fait un choix idéal pour les débutants.

Java : Java offre un équilibre entre puissance et simplicité, bien qu'il puisse être plus verbeux que Python.

2. Support de la Programmation Réseau :

C/C++ : Des bibliothèques comme sockets peuvent être utilisées pour la programmation réseau, mais cela nécessite une gestion manuelle des sockets.

Python : Python offre une API de haut niveau pour la programmation réseau, avec des modules comme socket et asyncio.

Java : Java propose des API robustes pour la programmation réseau, notamment avec les classes Socket et ServerSocket.

3. Gestion des Threads :

C/C++ : La gestion des threads peut être réalisée à l'aide des fonctions système ou de bibliothèques tierces comme pthreads.

Python : Python offre un support natif pour les threads avec le module threading, ainsi que pour les tâches asynchrones avec asyncio.

Java : Java intègre la gestion des threads dans son API standard avec le package java.lang.Thread.

4. Popularité :

C/C++ : Toujours largement utilisés dans les domaines nécessitant des performances maximales.

Python : En pleine expansion en raison de sa polyvalence et de sa facilité d'utilisation, notamment dans les domaines de l'IA, du web et du traitement des données.

Java : Reste populaire pour le développement d'applications d'entreprise, d'applications Android, etc.

5. Documentation :

C/C++ : De nombreuses ressources disponibles en ligne, y compris la documentation officielle et des tutoriels.

Python : Documentation exhaustive disponible sur le site officiel de Python, avec de nombreux exemples et guides de référence.

Java : Documentation complète fournie par Oracle, ainsi que de nombreux tutoriels et guides communautaires.

2.3 Bibliothèques Graphiques

Options de Bibliothèques Graphiques :

1. Qt :

Avantages :

Puissante et polyvalente, adaptée au développement d'applications de bureau et mobiles.

Offre une interface graphique riche avec des widgets personnalisables et une esthétique moderne.

Dispose d'une communauté active, d'une documentation complète et d'outils de développement avancés.

Inconvénients :

Peut-être plus complexe à apprendre pour les débutants en raison de sa taille et de sa complexité.

Les versions commerciales de Qt peuvent nécessiter des frais de licence pour une utilisation commerciale.

2. GTK (GIMP Toolkit) :

Avantage :

Utilisé dans de nombreux environnements de bureau Linux, notamment GNOME.

Dispose d'une conception modulaire avec des bibliothèques bien définies et des widgets prêts à l'emploi.

Licence libre et open source avec une grande communauté de développeurs.

Inconvénients :

Moins populaire que Qt dans certains contextes, ce qui peut limiter les ressources disponibles.

La documentation peut être moins exhaustive que celle de certaines autres bibliothèques.

3. Tkinter :

Avantage :

Fourni en standard avec Python, ce qui le rend facilement accessible et intégré à l'écosystème Python.

Interface simple et intuitive pour la création d'interfaces graphiques de base.

Convient particulièrement aux petits projets et aux débutants en programmation.

Inconvénients :

Moins puissant que Qt ou GTK en termes de capacités graphiques et de personnalisation.

Les applications Tkinter peuvent sembler moins modernes et esthétiques que celles développées avec d'autres bibliothèques.

Analyse Comparative :

Avantages Généraux :

Qt : Puissance, polyvalence, esthétique moderne, documentation complète.

GTK : Utilisé dans les environnements de bureau Linux, conception modulaire, licence libre.

Tkinter : Accessibilité avec Python, simplicité pour les petits projets, convivialité pour les débutants.

Inconvénients Généraux :

Qt : Complexité pour les débutants, frais de licence pour les versions commerciales.

GTK : Moins populaire dans certains contextes, documentation moins exhaustive.

Tkinter : Moins puissant que d'autres bibliothèques, aspect moins moderne.

Intégration avec le Langage Choisi :

Qt : Intégration solide avec C++ (Qt est écrit en C++ mais offre des liaisons pour d'autres langages comme Python).

GTK : Principalement utilisé avec C, mais dispose également de liaisons pour d'autres langages, y compris Python.

Tkinter : Intégré directement avec Python, car il est livré en standard avec la bibliothèque standard.

Ressources Disponibles :

Qt : Documentation complète, nombreux tutoriels, grande communauté de développeurs.

GTK : Documentation disponible, communauté active mais peut être moins importante que celle de Qt.

Tkinter : Documentation disponible, ressources d'apprentissage abondantes en raison de sa popularité avec Python.

Capacités Graphiques et Performance :

Qt : Puissant et performant, avec des capacités graphiques avancées pour les applications exigeantes.

GTK : Bonnes capacités graphiques adaptées à de nombreux types d'applications, performances acceptables.

Tkinter : Capacités graphiques de base, performances adéquates pour les applications simples, peut être moins performant pour les projets plus complexes.

2.4 Bibliothèques Réseau

1. Boost.Asio (pour C++) :

Avantages :

Bibliothèque réseau asynchrone et hautement performante, adaptée aux applications réseau exigeantes.

Prend en charge les opérations asynchrones, les timers, les sockets, les signaux et d'autres fonctionnalités réseau.

Offre une API riche et flexible pour la programmation réseau, avec une gestion efficace des connexions et des transferts de données.

Intégration avec la Bibliothèque Graphique :

Boost.Asio peut être intégré à des bibliothèques graphiques telles que Qt ou GTK en utilisant des threads ou des mécanismes de traitement asynchrone pour gérer les opérations réseau en arrière-plan tout en maintenant la réactivité de l'interface utilisateur.

Les signaux et les slots de Qt peuvent être utilisés pour connecter les événements réseau aux mises à jour de l'interface graphique, tandis que GTK peut tirer parti de ses boucles d'événements pour une intégration similaire.

2. Sockets en Python :

Avantages :

Facilité d'utilisation grâce à une API simple et intuitive pour la programmation réseau.

Livré avec le langage Python en standard, ce qui le rend facilement accessible et utilisable sans installation supplémentaire.

Prend en charge TCP/IP, UDP et d'autres protocoles réseau, offrant ainsi une flexibilité pour divers besoins de communication réseau.

Intégration avec la Bibliothèque Graphique :

Les sockets en Python peuvent être intégrés à des bibliothèques graphiques telles que Tkinter en utilisant des threads ou des boucles d'événements pour gérer les communications réseau en parallèle avec l'interface utilisateur.

Les mises à jour de l'interface graphique peuvent être déclenchées en réponse à des événements réseau à l'aide de mécanismes de synchronisation tels que les queues de messages ou les signaux.

Analyse Comparative :

Boost.Asio :

Avantages : Hautes performances, fonctionnalités asynchrones, gestion efficace des connexions réseau.

Intégration : Utilisation de threads ou de mécanismes asynchrones avec les bibliothèques graphiques pour maintenir la réactivité de l'interface utilisateur.

Sockets en Python :

Avantages : Facilité d'utilisation, intégration transparente avec Python, support de divers protocoles réseau.

Intégration : Utilisation de threads ou de mécanismes de synchronisation pour gérer les communications réseau en parallèle avec l'interface utilisateur.

3 Développement

3.1 Architecture

Code Serveur

Le code serveur constitue le point central de la communication, permettant à plusieurs clients de se connecter, d'échanger des messages et de se déconnecter de manière coordonnée. Voici un aperçu de son fonctionnement :

- **Mise en Place du Serveur :**

Le serveur crée un socket TCP/IP et le lie à une adresse IP et un port spécifiques (généralement localhost et le port 12345). Il commence à écouter les connexions entrantes avec une file d'attente de 256 connexions.

- **Gestion des Connexions Utilisateur :**

Lorsqu'une connexion est acceptée, le serveur reçoit un message du client, indiquant l'action à effectuer (connexion, déconnexion, envoi de message, etc.).

En fonction de l'action spécifiée, le serveur exécute la fonction correspondante à l'aide du dictionnaire `supported_commands`.

- **Communication avec les Utilisateurs :**

Le serveur échange des messages avec les utilisateurs connectés en utilisant des sockets associés à chaque utilisateur. Il utilise le format JSON pour structurer les données échangées entre le serveur et les clients.

Code Client

Le code client permet à un utilisateur de se connecter au serveur, d'envoyer des messages et de recevoir des réponses. Voici son fonctionnement :

- **Connexion au Serveur :**

Le client crée un socket et se connecte au serveur en spécifiant l'adresse IP et le port.

Il envoie son nom d'utilisateur et la destination du message pour identification.

- **Gestion des Entrées Utilisateur :**

Le client détecte les touches pressées à l'aide de la bibliothèque `keyboard` et forme des messages à envoyer.

Il stocke les touches pressées dans un tampon (buffer) jusqu'à ce que la touche "Entrée" soit pressée.

- **Envoi et Réception de Messages :**

Une fois le message formé et la touche "Entrée" pressée, le client envoie le message au serveur au format JSON. Il reçoit et affiche les messages du serveur à l'utilisateur à l'aide de la méthode `recv()`.

Interaction Serveur-Client

Le serveur gère la logique de connexion, déconnexion et communication entre les utilisateurs, tandis que le client permet à l'utilisateur d'entrer des messages et de les envoyer au serveur.

Les deux codes communiquent via un protocole préétabli à l'aide de sockets TCP/IP, garantissant une communication synchronisée et coordonnée entre les différents acteurs.

3.2 Implémentation

Tests et Débogage

- **Serveur**

Test de Connexion au Serveur :

```
madani49@debian:~$ python3 mn.py 12345
Le serveur est en ligne et écoute les connexions.
Connexion acceptée de ('127.0.0.1', 34840)
Connexion acceptée de ('127.0.0.1', 46768)
|
```

Ces résultats indiquent que le serveur Python a été exécuté avec succès et est maintenant en ligne, prêt à accepter les connexions entrantes. Voici ce que chaque ligne signifie :

Le serveur est en ligne et écoute les connexions : Cela indique que le serveur s'est lancé avec succès et qu'il attend maintenant des connexions entrantes de clients.

Connexion acceptée de ('127.0.0.1', 34840) : Cela signifie qu'une connexion a été établie avec succès entre le serveur et un client, identifié par l'adresse IP '127.0.0.1' (qui est l'adresse locale ou 'localhost') et le numéro de port 34840.

Connexion acceptée de ('127.0.0.1', 46768) : De manière similaire, une deuxième connexion a été établie avec un autre client, identifié par l'adresse IP '127.0.0.1' et le numéro de port 46768.

En résumé, ces résultats montrent que le serveur fonctionne correctement et est capable d'accepter des connexions entrantes de clients sur le port spécifié (12345 dans ce cas)

- **Client**

Test de Connexion au Serveur :

```
madani49@debian:~$ python3 cl.py 12345
Entrez votre nom d'utilisateur : MADANI
Entrez le nom d'utilisateur de destination : NOURA
Connecté au serveur
```

Ce code est un script client en Python qui se connecte à un serveur distant sur un port spécifié. Voici une explication ligne par ligne :

python3 cl.py 12345 : Cela lance le script Python nommé cl.py en spécifiant le numéro de port 12345 en tant qu'argument de ligne de commande.

Entrez votre nom d'utilisateur : MADANI : Le script demande à l'utilisateur d'entrer son nom d'utilisateur. Dans cet exemple, l'utilisateur a entré "MADANI".

Entrez le nom d'utilisateur de destination : NOURA : Le script demande également à l'utilisateur d'entrer le nom d'utilisateur de la personne à laquelle il souhaite envoyer un message. Dans cet exemple, l'utilisateur a entré "NOURA".

Connecté au serveur : Une fois que l'utilisateur a entré son nom d'utilisateur et le nom d'utilisateur de destination, le script se connecte au serveur distant. Cette ligne indique que la connexion au serveur a réussi.

En résumé, ce script permet à l'utilisateur de se connecter à un serveur distant en spécifiant son nom d'utilisateur et le nom d'utilisateur de la personne à laquelle il souhaite envoyer un message. Une fois la connexion établie, l'utilisateur peut commencer à envoyer des messages au destinataire spécifié.

3.3 Défis et Solutions

Problèmes identifiés :

Erreur de connexion (BrokenPipeError) :

Lors de l'exécution du code du serveur, une erreur de type BrokenPipeError est survenue lors de l'envoi de données à partir du client vers le serveur. Cette erreur peut être due à une déconnexion inattendue ou à une fermeture de connexion entre le client et le serveur.

Affichage incorrect des utilisateurs connectés :

L'affichage des utilisateurs connectés peut présenter des incohérences, telles que des retards dans la mise à jour de la liste des utilisateurs ou des données incorrectes affichées. Cela peut rendre difficile le suivi précis des utilisateurs connectés au serveur.

Gestion des touches pressées :

La fonctionnalité de saisie des touches pressées dans le client peut être améliorée pour garantir une capture précise des touches et éviter la perte de données. Actuellement, le code utilise une liste buffer pour stocker les touches pressées, mais des problèmes peuvent survenir lors de la gestion de cette liste.

Solutions envisagées :

Gestion des erreurs :

Mettre en place des mécanismes de gestion des erreurs pour identifier et résoudre les problèmes de connexion, tels que la réouverture des sockets en cas de déconnexion inattendue ou la gestion appropriée des exceptions lors de l'envoi de données.

Optimisation de la communication :

Optimiser les mécanismes de transmission de données entre le client et le serveur en utilisant des protocoles de communication efficaces et en mettant en œuvre des stratégies de gestion de la latence et de la bande passante.

Amélioration de l'affichage des utilisateurs connectés :

Mettre en œuvre des algorithmes de suivi des utilisateurs connectés plus robustes et efficaces pour garantir la cohérence et l'exactitude de la liste des utilisateurs affichés à tout moment.

Amélioration de la gestion des touches pressées :

Réviser la logique de capture et de traitement des touches pressées dans le client pour garantir une saisie précise des données, en évitant les pertes de données et en améliorant la réactivité de l'interface utilisateur.

4 Conclusion

Le projet de développement d'un mini chat client-serveur en TCP a été une expérience enrichissante qui a permis d'explorer de nombreux aspects de la programmation réseau et de la communication inter-processus. À travers ce projet, nous avons pu mettre en pratique les concepts fondamentaux des sockets TCP, de la sérialisation des données et de la gestion des connexions dans un environnement distribué.

L'implémentation d'un chat client-serveur a présenté plusieurs défis, notamment la gestion des connexions, la synchronisation des utilisateurs connectés et l'optimisation de l'expérience utilisateur. Tout au long du développement, nous avons été confrontés à des problèmes tels que les erreurs de connexion, les retards dans l'actualisation des utilisateurs et les limitations de la saisie des messages.

Cependant, grâce à une analyse approfondie des problèmes rencontrés et à une collaboration étroite avec l'équipe de développement, nous avons pu identifier des solutions efficaces pour surmonter ces obstacles. En adoptant une approche itérative et en mettant l'accent sur la résolution progressive des problèmes, nous avons réussi à améliorer progressivement la stabilité, la fiabilité et la convivialité du chat client-serveur.

En fin de compte, ce projet nous a permis de développer non seulement nos compétences techniques en programmation réseau et en développement logiciel, mais aussi notre capacité à résoudre des problèmes complexes et à collaborer efficacement en équipe. Nous sommes fiers du résultat final et convaincus que ce mini chat client-serveur en TCP constitue une base solide pour de futurs projets de communication en réseau.

ANNEXE

• code serveur

```
1 import socket
2 import sys
3 import json
4 import select
5
6 # Dictionnaire pour stocker les utilisateurs connectés
7 users = {}
8
9 # Fonction pour déconnecter un utilisateur
10 def disconnect(user):
11     print("Déconnexion de l'utilisateur:", user)
12     del users[user]
13
14 # Dictionnaire des commandes supportées avec des fonctions lambda
15 supported_commands = {
16     "connect": lambda user, sock: users.update({user: sock}), # Connecte un utilisateur
17     "disconnect": lambda user: disconnect(user), # Déconnecte un utilisateur
18     "send": lambda data: users[data["destination"]].send(data["message"].encode('utf-8')), # Envoie un message à un utilisateur
19     "info": lambda user: users[user].send(json.dumps({"users": list(users.keys())[1:]}).encode('utf-8')) # Envoie des informations sur les utilisateurs connectés
20 }
21
22 # Fonction pour démarrer le serveur
23 def start_server():
24     # Crée un socket pour le serveur
25     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
26     # Lie le socket à localhost sur un port spécifié (par défaut 12345)
27     port = int(sys.argv[1]) if len(sys.argv) > 1 else 12345
28     server_socket.bind(('localhost', port))
29     # Commence à écouter les connexions entrantes avec une file d'attente de 256 connexions
30     server_socket.listen(256)
31     print("Le serveur est en ligne et écoute les connexions sur le port", port)
32
33     while True:
34         # Ajoute le socket du serveur à la liste des sockets à surveiller
35         users["self"] = server_socket
36         # Sélectionne les sockets prêts à être lus
37         readable, _, _ = select.select(users.values(), [], [])
38         for sock in readable:
39             if sock is server_socket:
40                 # Accepte une nouvelle connexion entrante
41                 client_socket, client_address = server_socket.accept()
42                 print(f"Connexion acceptée de {client_address}")
43                 # Reçoit un message du client
44                 message = client_socket.recv(1024).decode('utf-8')
45                 try:
46                     # Essaie de décoder le message JSON
47                     jdata = json.loads(message)
48                 except json.JSONDecodeError:
49                     continue
50
51                 # Traite l'action basée sur le message reçu
52                 if jdata["action"] == "connect":
53                     supported_commands["connect"](jdata["user"], client_socket)
54                 elif jdata["action"] == "disconnect":
55                     supported_commands["disconnect"](jdata["user"])
56                 elif jdata["action"] == "send" and jdata["destination"] in users.keys():
57                     supported_commands["send"](jdata)
58                 elif jdata["action"] == "info":
59                     supported_commands["info"](jdata["user"])
60                 else:
61                     print("Commande non prise en charge")
62                 if message != "":
63                     print(f"Message reçu: {message}")
64                 print("Utilisateurs connectés:", list(users.keys())[1:])
65             else:
66                 # Reçoit un message d'un utilisateur déjà connecté
67                 message = sock.recv(1024).decode('utf-8')
68                 if message != "":
69                     print(f"Message reçu: {message}")
70                 try:
71                     # Essaie de décoder le message JSON
72                     jdata = json.loads(message)
73                 except json.JSONDecodeError:
74                     continue
75                 # Traite l'action basée sur le message reçu
76                 if jdata["action"] == "send" and jdata["destination"] in users.keys():
77                     supported_commands["send"](jdata)
78                 elif jdata["action"] == "info":
79                     supported_commands["info"](jdata["user"])
80
81 if __name__ == "__main__":
82     start_server()
```

• Code Client

```
1 import socket
2 import sys
3 import json
4 import keyboard
5
6 # Liste pour stocker les touches pressées pour former un message
7 buffer = []
8
9 # Nom d'utilisateur et destination du message
10 user = ""
11 destination = ""
12
13 # Socket du client
14 client_socket = None
15
16 # Fonction appelée lorsqu'une touche est pressée
17 def on_key_event(e):
18     # Si Ctrl+M est pressé, on demande à l'utilisateur d'entrer le message.
19     if e.name == 'm' and e.event_type == 'down' and keyboard.is_pressed('ctrl'):
20         buffer.clear()
21         print("Enter your message: ")
22         keyboard.hook(capture_keys)
23
24 # Fonction pour capturer les touches pressées et les ajouter au buffer
25 def capture_keys(e):
26     # Si Enter est pressé, on arrête de capturer les touches et on affiche le message
27     if e.name == 'enter' and e.event_type == 'down':
28         keyboard.unhook(capture_keys)
29         message = ''.join(buffer[1:]) # Assemble les touches dans un message
30         data = json.dumps({"user": user, "destination": destination, "message": message}) # Formate les données en JSON
31         client_socket.send(data.encode('utf-8')) # Envoie les données au serveur
32     else:
33         # Sinon, on ajoute la touche au buffer
34         if e.name == 'backspace':
35             buffer.pop() # Supprime la dernière touche si c'est "backspace"
36         else:
37             buffer.append({"space": " ", "backspace": ""}.get(e.name, e.name)) # Ajoute la touche au buffer
38
39 # Fonction pour démarrer le client
40 def start_client():
41     global client_socket
42     global user
43     global destination
44     user = input("Enter your username: ") # Demande le nom d'utilisateur
45     destination = input("Enter the destination username: ") # Demande la destination du message
46     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) # Crée un socket
47     client_socket.connect(('localhost', int(sys.argv[1]) if len(sys.argv) > 1 else 12345)) # Connecte au serveur
48     client_socket.settimeout(1) # Définit une limite de temps pour la réception de données
49     print("Connected to server")
50     keyboard.add_hotkey('ctrl+q', lambda: sys.exit(0)) # Permet de quitter le programme avec Ctrl+Q
51     keyboard.on_press(on_key_event) # Déclenche la fonction on_key_event lorsqu'une touche est pressée
52     while True:
53         try:
54             res = client_socket.recv(1024) # Reçoit les données du serveur
55             if res:
56                 print(f"Received message: {res.decode('utf-8')}") # Affiche le message reçu
57         except socket.timeout:
58             continue
59
60 if __name__ == "__main__":
61     start_client() # Lance le client
62
```