

**UNIVERSITÉ NATIONALE DU VIETNAM À HANOÏ**  
**INSTITUT DE LA FRANCOPHONIE POUR L'INFORMATIQUE**

---

**Option : Systèmes Intelligents et Multimédia (SIM 27)**

**Messagerie Instantanée Interactive sur un réseau Local**

**Groupe 3**

**Constitué de :**

- **EBWALA EBWALETTE Priscille**
- **TAKOUCHOUANG Fraisse Sacré**
- **LUTALA LUSHULI David**
- **NZAZI NGABILA Boaz**

**Supervisé par : M Nguyen Hong Quang**

**Année Universitaire : 2024**

## Table de Matiere

1. Introduction.....	3
2. Cahier des Charge.....	3
2.1.Contexte et Définition du Projet.....	3
2.2. Objectif du Projet.....	3
2.3. Périmètre du Projet.....	4
2.4. Description Fonctionnelle des Besoins.....	4
2.5. Enveloppe Budgétaire.....	5
2.6. Délais.....	5
2.7. Ressources Humaines et Techniques.....	6
2.8. Critères de Performance.....	6
2.9. Critères de Sécurité.....	6
2.10. Maintenance et Évolution.....	7
3. Présentation du Protocole.....	7
3.1 aperçu du protocole conçus.....	7
3.1.1. Structure d'un paquet de communication.....	7
3.1.2.Services.....	8
3.1.3. Codes d'état.....	9
3.1.4. États de session.....	9
3.1.5. Opérations principales.....	10
3.2. Description.....	11
3.2.1. Format des Messages.....	11
3.2.2. Commandes et Opérations.....	11
3.2.3. Gestion des Connexions et Diffusion.....	11
3.2.4. Robustesse et Flexibilité.....	12
4. Implémentation du logiciel.....	12
4.1 Algorithmes.....	12
4.2 Structure du Programme.....	12
5. Tests et Évaluation des Performances.....	13
5.1 Scénarios de Test.....	13
5.2 Résultats des Tests.....	17
5.3 Évaluation des Performances.....	17
6. Perspectives d'Amélioration.....	18
7. Conclusion.....	18
8. Documentation de l'Application de Chat.....	19
8.1 Vue d'ensemble.....	19
8.2. Architecture du Système.....	19
8.3. Protocoles de Communication.....	20
8.4. Instructions de Configuration et de Lancement du Projet.....	21
8.5. Code Source.....	22
8.6. Références.....	23

# 1. Introduction

Dans un monde où la communication instantanée est devenue indispensable, la messagerie interactive joue un rôle clé dans la facilitation des échanges entre utilisateurs. Ce projet de messagerie instantanée interactive sur réseau local a pour objectif principal de concevoir un protocole et de l'implémenter sous forme d'une application permettant aux utilisateurs de communiquer sur un réseau IP local avec un serveur décentralisé. Ce projet repose sur l'application de connaissances en programmation réseau et en conception de protocoles, en utilisant Java comme langage principal de développement.

## 2. Cahier des Charge

### 2.1.Contexte et Définition du Projet

Aujourd'hui, la messagerie instantanée est devenue un outil de communication incontournable, offrant une réponse rapide aux besoins des utilisateurs dans divers contextes, qu'il s'agisse de travail, d'études ou de loisirs. Ce projet a pour objectif de concevoir une application de messagerie instantanée fonctionnant sur un réseau local (LAN), sans recourir à un serveur central. Cette architecture décentralisée permettra aux utilisateurs d'interagir entre eux directement, rendant le système plus robuste face aux déconnexions et aux fluctuations des adresses IP.

### 2.2. Objectif du Projet

L'objectif principal de ce projet est de développer un système de communication interactif et décentralisé, permettant à des utilisateurs de s'échanger des messages texte sur un réseau local. Contrairement aux systèmes traditionnels qui reposent sur un serveur centralisé, cette messagerie doit être entièrement distribuée, garantissant ainsi une gestion autonome des connexions et des déconnexions des utilisateurs.

Hormis l'objectif principal, ce projet possède également des objectifs spécifiques dont nous pouvons lister :

- les postes des utilisateurs se trouvent sur un réseau local (donc sur le même réseau IP) ;
- les utilisateurs sont nomades, ils ne conservent pas leurs adresses IP et peuvent se connecter à partir de postes différents ;
- La gestion de système est décentralisée, donc pas de serveur central pour gérer les utilisateurs et les sessions d'échange. Elle doit être réalisée d'une façon totalement distribuée sur les postes clients en s'appuyant sur le protocole dont nous allons concevoir ;
- La communication sera faite par des messages asynchrones sous forme de chaînes de caractères de longueurs différentes ;
- on a un seul groupe de discussion sur le réseau ;

- chaque utilisateur doit s'enregistrer sur le réseau lors de sa connexion en fournissant un identifiant (pseudonyme);
- chaque poste doit maintenir une liste des utilisateurs connectés au système de messagerie (gestion dynamique des connexions/déconnexions) ;
- la gestion des états de l'utilisateur (connecté, absent, occupé, etc.) ;

### **2.3. Périmètre du Projet**

Le projet vise à couvrir les éléments suivants :

- Conception d'un protocole de communication dévante : Un protocole sur mesure sera développé pour garantir une transmission efficace et sécurisée des messages entre utilisateurs. Ce protocole gèrera les connexions, les déconnexions, ainsi que la mise à jour dynamique de la liste des utilisateurs.
- Développement d'un logiciel de messagerie instantanée : Ce logiciel fonctionnera sur un réseau local et utilisera une architecture distribuée, sans serveur central pour gérer les échanges de messages.
- Gestion dynamique des connexions et déconnexions : Le système devra être capable de détecter automatiquement l'entrée ou la sortie d'un utilisateur et mettre à jour la liste des participants en temps réel.
- Envoi de messages texte asynchrones : Les utilisateurs pourront envoyer des messages de longueur variable à tout moment, sans que cela n'interfère avec d'autres opérations. Cela inclut la gestion de file d'attente pour s'assurer que tous les messages sont bien diffusés.
- Affichage des états des utilisateurs : Chaque utilisateur aura un statut visible (connecté, absent, occupé) que les autres participants pourront consulter pour savoir s'ils sont disponibles.

Le projet ne couvrira pas les aspects suivants pour la version initiale :

- Gestion de plusieurs groupes de discussion.
- Sécurisation avancée des messages via un chiffrement.
- Interface graphique sophistiquée.

Ces fonctionnalités pourraient être ajoutées dans des versions futures après la réalisation du projet de base.

### **2.4. Description Fonctionnelle des Besoins**

Voici les principales fonctionnalités que devra inclure l'application :

- Enregistrement d'un pseudonyme unique : Lors de sa connexion au réseau, chaque utilisateur devra choisir un pseudonyme unique pour l'identifier. Le système devra vérifier que ce pseudonyme n'est pas déjà utilisé pour éviter des conflits d'identification.
- Mise à jour dynamique de la liste des utilisateurs : La liste des utilisateurs connectés devra être actualisée en temps réel sur tous les postes clients dès

qu'un nouvel utilisateur se connecte ou qu'un utilisateur existant se déconnecte. Cette liste sera visible pour tous les utilisateurs.

- Envoi et réception de messages asynchrones : Les messages texte pourront être envoyés et reçus sans bloquer les autres opérations. Le système gèrera une diffusion efficace des messages, garantissant qu'ils sont envoyés à tous les utilisateurs connectés sans latence notable.
- Affichage de l'état des utilisateurs : Chaque utilisateur pourra définir son état (connecté, absent, occupé), qui sera visible pour les autres utilisateurs afin de favoriser une meilleure gestion des interactions.
- Interface utilisateur minimale : L'application sera principalement textuelle, fonctionnant en mode terminal, et offrira une interface simple pour envoyer et recevoir des messages ainsi que pour consulter la liste des utilisateurs connectés et leur état.

## **2.5. Enveloppe Budgétaire**

Le projet de projet académique, l'enveloppe budgétaire est limitée. Les principaux coûts seront relatifs aux ressources humaines :

- Développeurs : Ils seront chargés de la conception et du développement de l'application.
- Testeurs : Ils valident le bon fonctionnement du système, testeront la robustesse du protocole de communication et vérifieront l'ergonomie de l'interface utilisateur.

Le projet utilisera des outils open-source pour limiter les dépenses, tels que :

- Eclipse ou Visual Code comme environnements de développement.
- Java comme langage de programmation, idéal pour la gestion des réseaux et la communication via sockets.

Le matériel requis comprend des ordinateurs connectés à un réseau local pour les tests.

## **2.6. Délais**

Le projet devra être réalisé dans le délai d'un de trois semaines :

- Semaine 1 : Analyse des besoins et conception du protocole de messagerie. Les spécifications techniques et fonctionnelles seront finalisées durant cette période.
- Semaine 2 : Développement du logiciel et implémentation des fonctionnalités de base, incluant la communication asynchrone et la gestion des utilisateurs.

- Semaine 3 : Intégration, tests des fonctionnalités avancées et optimisation du protocole pour assurer la robustesse et la fluidité du système.

## **2.7. Ressources Humaines et Techniques**

Ressources Humaines :

- Chef de projet : Responsable de la coordination des différentes phases du projet, de la répartition des tâches et de la gestion des délais.
- Développeurs : Experts en Java et en programmation réseau, ils conçoivent l'architecture décentralisée et implémentent le protocole de communication.
- Épreneurs : Chargés de valider la robustesse du système, en testant la gestion des connexions et la fluidité des échanges.

Ressources Techniques :

- Un réseau local (LAN) pour tester les interactions entre les utilisateurs.
- Des ordinateurs équipés de systèmes d'exploitation compatibles avec Java (Linux, macOS, ou Windows).
- Des environnements de développement intégrés (IDE) comme Eclipse ou Visual Studio.

## **2.8. Critères de Performance**

Le système du système du système Respect les critères de performance suivants :

- Latence : Le temps de réponse pour l'envoi et la réception des messages devra être inférieur à 1 seconde sur le réseau local, garantissant ainsi une communication en temps réel.
- Capacité : Le système devra supporter au moins 10 simultanément à l'utilisateur sans dégradation notable de la performance (latence, perte de message).
- Robustesse : Le système devra gérer les connexions et déconnexions fréquentes sans interruption des échanges entre les utilisateurs restants.

## **2.9. Critères de Sécurité**

Bien que la version initiale du projet n'inclut pas un chiffrement avancé des messages, certaines mesures de sécurité de base devront être mises en place :

- Unicité des pseudonymes : Le système devra s'assurer que chaque utilisateur possède un pseudonyme unique pour éviter toute confusion.
- Vérification des messages : Les messages échangés devront être vérifiés afin de prévenir les attaques de type injection de code.
- Protection des États-utilisateurs : Les états des utilisateurs devront être protégés contre les modifications non autorisées.

## 2.10. Maintenance et Évolution

Après la livraison du projet, un plan de maintenance a été mis en place pour corriger les cas de bugs et optimiser les performances. À long terme, des rallonges possibles incluront :

- Gestion de plusieurs groupes de discussion : Permettre aux utilisateurs de créer et de rejoindre différents salons de discussion.
- Interface graphique plus conviviale : Remplacer l'interface textuelle par une interface graphique (GUI) plus intuitive.
- Chiffre d'emploi des messages : Intégrer un système de chiffrement pour garantir la sécurité des échanges de messages.

## 3. Présentation du Protocole

Le protocole de communication de cette application est basé sur un modèle décentralisé et fonctionne sans serveur centralisé complexe, offrant ainsi une grande flexibilité pour les utilisateurs d'un réseau local. Voici les détails du protocole conçu :

### 3.1 aperçu du protocole conçus

#### 3.1.1. Structure d'un paquet de communication

Chaque paquet envoyé entre le Client et le Serveur suit une structure fixe avec différents champs :

```
+-----+
| PROTO (4B) | version (4B) | pkt_len (2B) |
+-----+
| service (2B) | statut (4B) | session_id (4B) |
+-----+
| DONNÉES : Clé/valeur (0 - 65535 octets) |
+-----+
```

Détails des champs :

- PROTO (4 octets) : Identifiant du protocole (toujours "PROTO"). Pour indiquer le protocole de communication utilisé.
- Version (4 octets) : Numéro de version du protocole. Pour la version 1, c'est 0x01 0x00 0x00 0x00.

- Pkt\_len (2 octets) : Taille des données envoyées dans le paquet, en octets (limite théorique à 65535 octets, mais généralement autour de 1000 octets).
- Service (2 octets) : Code définissant le type de service ou d'opération en cours.
- Statut (4 octets) : Indique l'état de la requête ou de la réponse (par exemple, succès, échec, etc.).
- Session ID (4 octets) : Identifiant unique de session, utilisé pour lier la session en cours entre client et serveur.
- Données : Une série de paires clé/valeur, représentant les informations spécifiques à chaque service. Les paires sont terminées par 0xc0 0x80.

### 3.1.2.Services

Le protocole utilise différents services, qui correspondent aux opérations effectuées entre le client et le serveur. Voici une liste des services avec leurs codes hexadécimaux:

Services	code(Hex )	Description
SERVICE_CONNEXION	0x01	Connexion d'un utilisateur
SERVICE_DECONNEXION	0x02	Déconnexion d'un utilisateur
SERVICE_MESSAGERIE	0x03	Envoi d'un Message Texte
SERVICE_STATUT	0x04	Affichage de l'état de l'utilisateur(Online, Busy , Offline)
SERVICE_REPONSE_CONNEXION	0x05	Client est connecté
SERVICE_MESSAGE_GROUPE	0x06	communication du groupe.

Ces services déterminent la nature de chaque requête ou réponse échangée entre le client et le serveur.

Exemples de services:

- SERVICE\_CONNEXION : Utilisé pour établir une session initiale entre le client et le serveur.



- SERVICE\_MESSAGERIE : Permet l'envoi de messages instantanés.

### 3.1.3. Codes d'état

Les codes d'état sont utilisés pour définir la disponibilité de l'utilisateur ou l'état d'une opération. Ils peuvent être utilisés par le serveur pour informer le client sur la réponse ou l'état du service demandé.

Etats	code(HEX)	Description
Statut_Busy	0x00	L'utilisateur est occupé
Statut_Online	0x01	L'utilisateur est connecté(Online)
Statut_Offline	0x02	L'utilisateur est déconnecté

- Busy (0): L'utilisateur est occupé.
- Online (1) : Indique que l'utilisateur est connecté.
- Offline (2) : Indique que l'utilisateur est hors ligne mais toujours connecté.

Ces statuts peuvent être utilisés pour informer le client ou le serveur du résultat d'une requête ou d'un problème rencontré.

### 3.1.4. États de session

Les sessions de communication passent par deux états principaux : Connexion et Messagerie.

#### 3.1.4.1. Connexion

- **Étape 1** : Le client envoie une demande de connexion avec son identifiant utilisateur via un paquet de type SERVICE\_CONNEXION.
- **Étape 2** : Le serveur renvoie une chaîne de défis, à laquelle le client doit répondre.
- **Étape 3** : Le client envoie sa réponse via un paquet de type SERVICE\_REPONSE\_CONNEXION. Si la réponse est correcte, la session passe à l'état suivant, autrement une erreur est renvoyée.

#### 3.1.4.2. Messagerie

Une fois la connexion réussie, le client peut interagir avec le serveur pour :

- Recevoir la liste des utilisateurs en ligne et leurs statuts.

- Envoyer et recevoir des messages.
- Modifier son statut.
- La session prend fin lorsqu'une déconnexion est initiée via un paquet SERVICE\_DECONNEXION.

### **3.1.5. Opérations principales**

Voici les principales opérations supportées par le protocole, leur structure et leur usage.

#### **3.1.5.1. Connexion**

- Service: SERVICE\_CONNEXION (0x01)
- Champs:
  - ❖ 1: Adresse IP du Serveur
  - ❖ 2: Nom de L'utilisateur

Le serveur répond avec un défi, auquel le client répond via un paquet SERVICE\_REPONSE\_CONNEXION contenant :

- ❖ 0: Nom de L'utilisateur
- ❖ 1 : Liste des utilisateurs et leurs statuts

#### **3.1.5.2. Envoi d'un message**

- Service: SERVICE\_MESSAGERIE (0x03)
- Champs:
  - 0: Nom de L'utilisateur
  - 1: Status
  - 14: contenu du message

#### **3.1.5.3. Définir le statut**

Pour indiquer un changement de statut (Online, busy, Offline), le service SERVICE\_STATUT est utilisé. Détails:

- Service : SERVICE\_STATUT (0x04)
- Champs:
  - 10: Choix Status (online, offline, busy)
  - 19 : Statut à afficher Status

#### **3.1.5.4. Déconnexion**

Lorsqu'un utilisateur souhaite se déconnecter, il envoie un paquet de type SERVICE\_DECONNEXION. Ce paquet ne contient pas de clé/valeur particulière.

- Service: SERVICE\_DECONNEXION
- Aucune clé/valeur requise.

## **3.2. Description**

### **3.2.1. Format des Messages**

Le protocole repose sur des messages texte asynchrones envoyés entre les utilisateurs. Chaque message est structuré de manière simple et efficace, selon le format suivant :

- En-tête : Indique le type de message (connexion, message texte, déconnexion, changement d'état).
- Corps : Contient les données spécifiques au type de message. Par exemple, pour une connexion, le corps inclut le pseudonyme de l'utilisateur ; pour un message, il contient le texte du message.

### **3.2.2. Commandes et Opérations**

Le protocole gère plusieurs types de commandes essentielles :

- CONNECT [pseudonyme] : Enregistrer un utilisateur sur le réseau, permettant d'ajouter son pseudonyme à la liste des utilisateurs connectés et d'informer tous les autres utilisateurs.
- MESSAGE [texte] : Envoie un message texte à tous les utilisateurs connectés via le réseau.
- STATUS [état] : Change l'état de l'utilisateur (disponible, absent, occupé), ce qui est immédiatement transmis aux autres utilisateurs.
- DISCONNECT : Indique la déconnexion de l'utilisateur du système, supprimant son pseudonyme de la liste des utilisateurs connectés et diffusant cette information aux autres clients.

### **3.2.3. Gestion des Connexions et Diffusion**

L'une des particularités de ce protocole est sa capacité à gérer des connexions dynamiques et distribuées. Voici comment cela fonctionne :

- Connexion d'un utilisateur : Lorsqu'un nouvel utilisateur se connecte, son pseudonyme est ajouté à la liste des utilisateurs connectés. Cette liste est maintenue à jour sur tous les postes clients grâce à la diffusion continue des changements d'état.
- Diffusion des messages : Chaque message envoyé par un utilisateur est instantanément diffusé à tous les autres utilisateurs connectés, garantissant ainsi une communication en groupe en temps réel.

- Synchronisation des utilisateurs : À chaque connexion ou déconnexion, la liste des utilisateurs connectés est mise à jour et diffusée à tous les clients pour assurer une synchronisation parfaite.

#### **3.2.4. Robustesse et Flexibilité**

Le protocole est conçu pour être robuste et tolérant aux erreurs :

- En cas de déconnexion inattendue d'un utilisateur, le système met automatiquement à jour la liste des utilisateurs actifs.
- Les connexions simultanées de plusieurs utilisateurs sont prises en charge sans interruption, grâce à une gestion multithread des clients via la classe ClientHandler.

Ce protocole de messagerie interactive garantit une communication asynchrone efficace et dynamique sur un réseau local, tout en assurant la simplicité de l'interface et la gestion décentralisée des utilisateurs.

### **4. Implémentation du logiciel**

L'implémentation a été réalisée en Java, en utilisant des sockets pour la communication réseau. Chaque poste exécute une instance du programme, ce qui permet de gérer à la fois l'envoi et la réception des messages en utilisant un modèle d'écoute active.

#### **4.1 Algorithmes**

L'algorithme principal se divise en deux parties :

1. Écoute des connexions entrantes : Le serveur écoute les messages envoyés par d'autres utilisateurs sur le réseau.
2. Envoi des messages : Lorsqu'un utilisateur envoie un message, celui-ci est diffusé à l'ensemble des autres utilisateurs connectés via le réseau par le serveur .

#### **4.2 Structure du Programme**

Le programme est divisé en plusieurs modules :

- Serveur : Responsable de l'écoute et de la réception des messages.
- Client : Gère l'envoi des messages et la gestion de l'interface utilisateur.
- Gestion des utilisateurs : Stocke et met à jour la liste des utilisateurs actifs ainsi que leurs statuts.

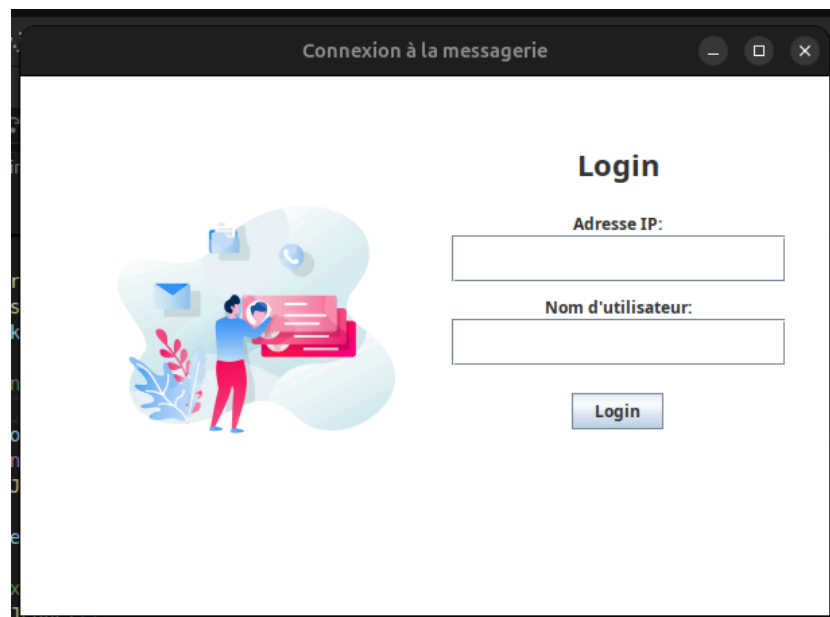
## 5. Tests et Évaluation des Performances

### 5.1 Scénarios de Test

Plusieurs scénarios de test ont été réalisés pour valider le fonctionnement du système :

Test de connexion/déconnexion : Des utilisateurs se connectent et se déconnectent simultanément du réseau.

#### A. Interface de connexion client

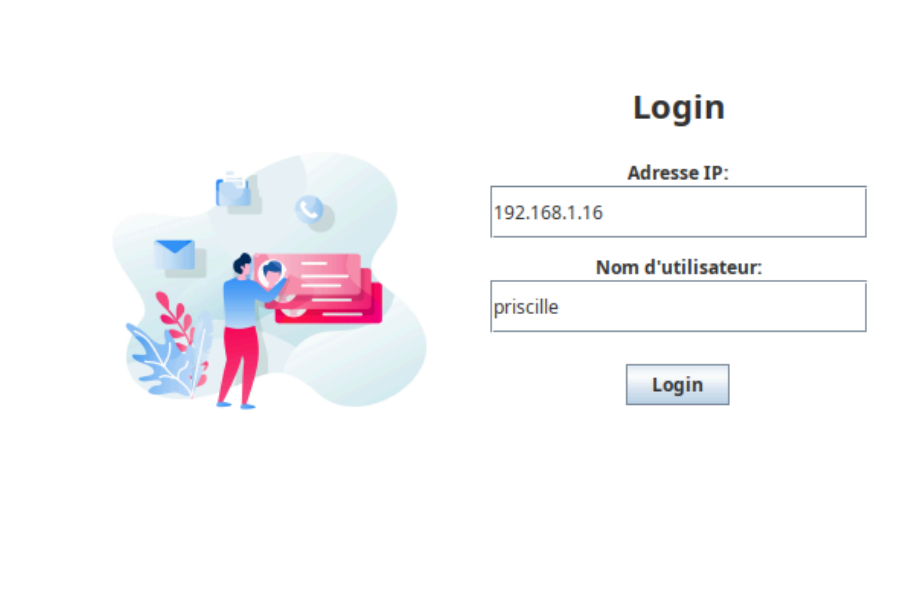


The screenshot shows a web browser window titled "Connexion à la messagerie". Inside, there is a "Login" section. On the left, there is a colorful illustration of a person at a desk with a computer, surrounded by icons of a mail envelope, a calendar, and a clock. On the right, there are two input fields: "Adresse IP:" and "Nom d'utilisateur:". Below these fields is a "Login" button.

Figure 1 : présentation de la page de connexion

Cette interface représente l'interface de connexion sur l'application, cela représente le point d'accès à l'application (serveur) pour les utilisateurs.

#### B. Remplissage de l'étape de connexion



This screenshot shows the same login form as Figure 1, but with test data entered. The "Adresse IP:" field contains "192.168.1.16" and the "Nom d'utilisateur:" field contains "priscille". The "Login" button remains below the fields.

Figure 2 : Remplissage de la page d'authentification

Cette figure représente le remplissage des informations de connexion, cela permet d'ouvrir la page de chat pour les utilisateurs.

### C. Interface de chat



Figure 3 : Interface chat

Cette figure représente l'interface de chat utilisateur. A travers cette interface l'utilisateur peut envoyer des messages et voir tous ceux qui sont connectés. Il peut lancer la discussion qui sont correctement reçus par tous les autres utilisateurs connectés au groupe de chat.

### D. Choix du statut

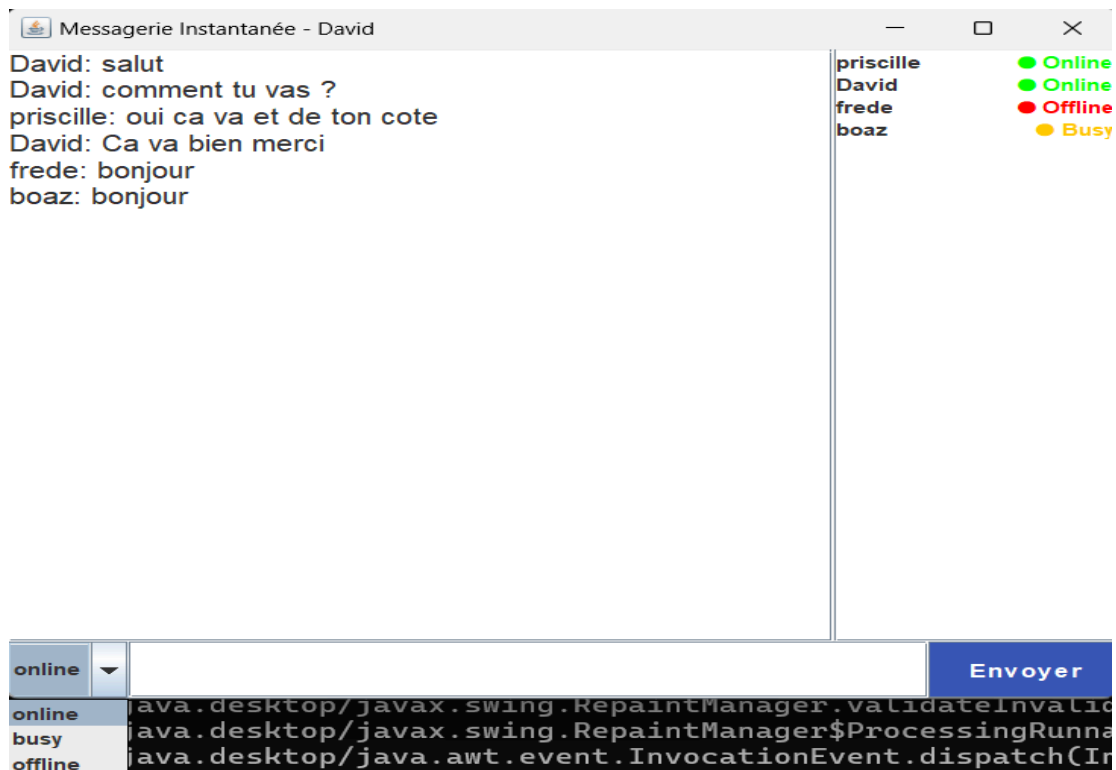


Figure 4 : Choix du statut

Cette figure représente un scénario du choix de statut utilisateur. Les utilisateurs peuvent choisir leur statut (disponible, absent, occupé).

### E. Changement de statut

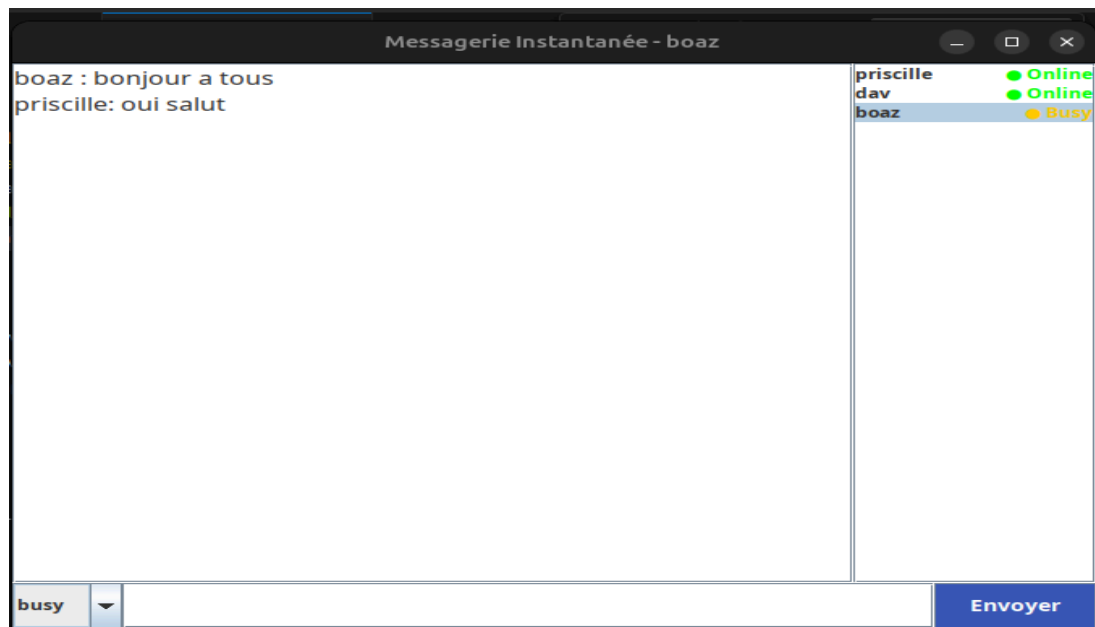


Figure 5 : Changement de statut

Cette figure représente un scénario de changement de statut d'un utilisateur. Lorsqu'un utilisateur change de statut, la liste des utilisateurs est mise à jour au serveur qui renvoie la nouvelle mise à jour à tous les utilisateurs du groupe chat. on

peut remarquer que l'utilisateur boaz a changé son statut à busy (occupé) cad qu'il est occupé malgré sa présence dans le chat, cela représente le cas d'un utilisateur qui était dans un chat en groupe mais qui a reçu un appel en groupe qui lui met dans une position où il ne peut plus être disponible pour le chat en groupe d'où il se met en mode busy pour signifier aux autres utilisateurs son absence de réponse instantanée.

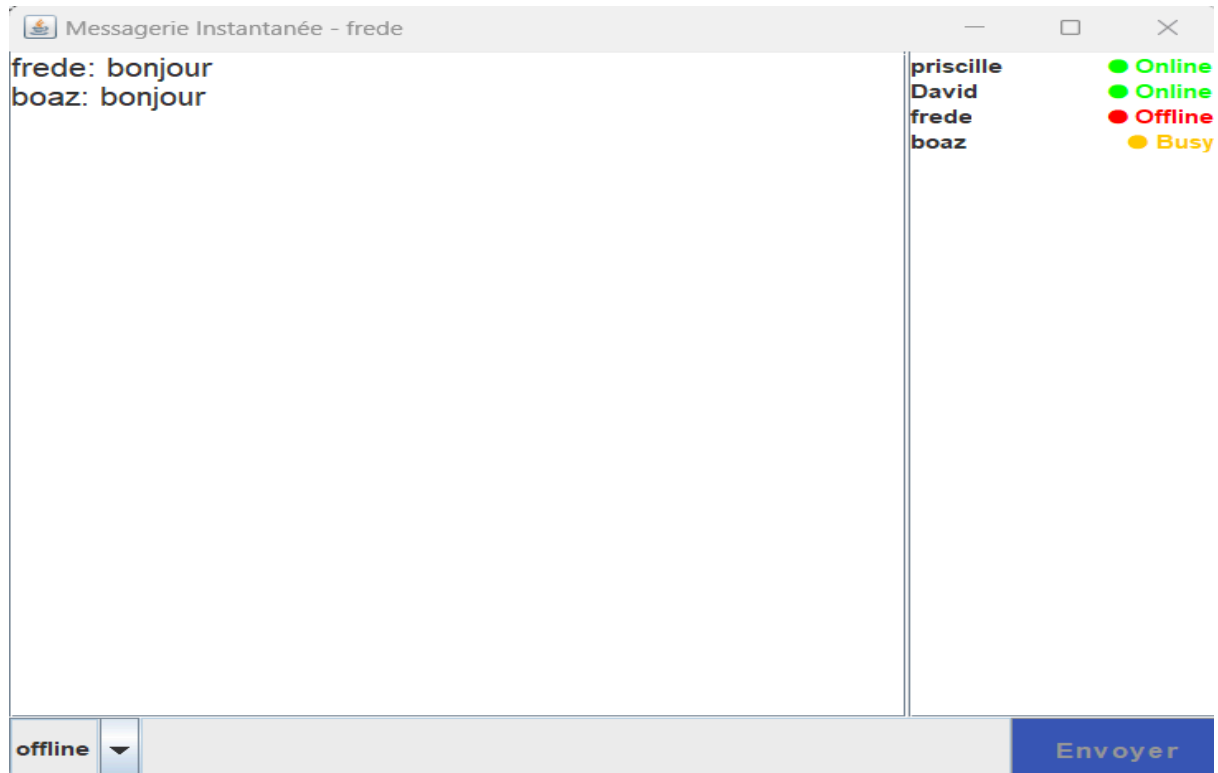


Figure 6 : Changement statut Offline

Dans l'illustration ci-dessus, on peut remarquer l'utilisateur frede qui est offline cad il est connecté mais ne pas en ligne dans le chat. On peut remarquer cet utilisateur ne peut plus écrire suite à son statut offline, qui lui enlève cette possibilité de participer à la discussion.

## F. Démarrage Serveur





Arrêter le Serveur

Figure 6 : Interface démarrage serveur

### G. Interface Fonctionnement Serveur

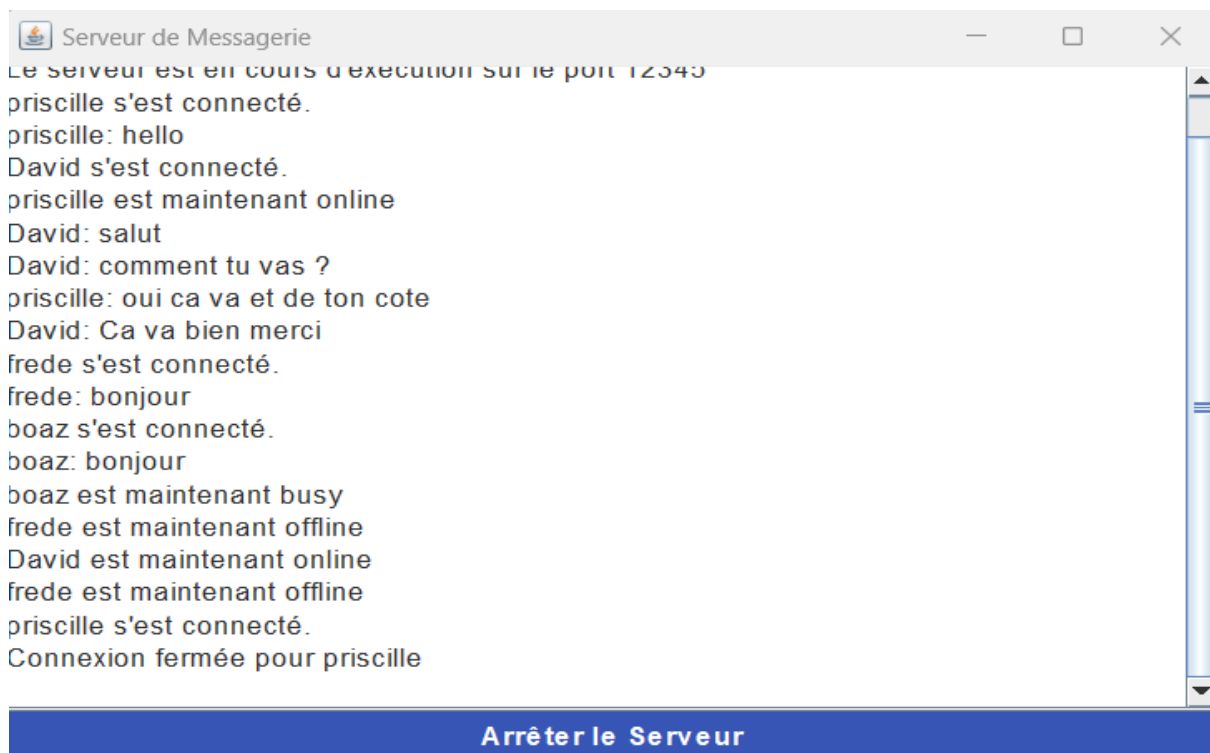


Figure 7 : Interface Fonctionnement Serveur

## H. Capture des segment avec Wireshark

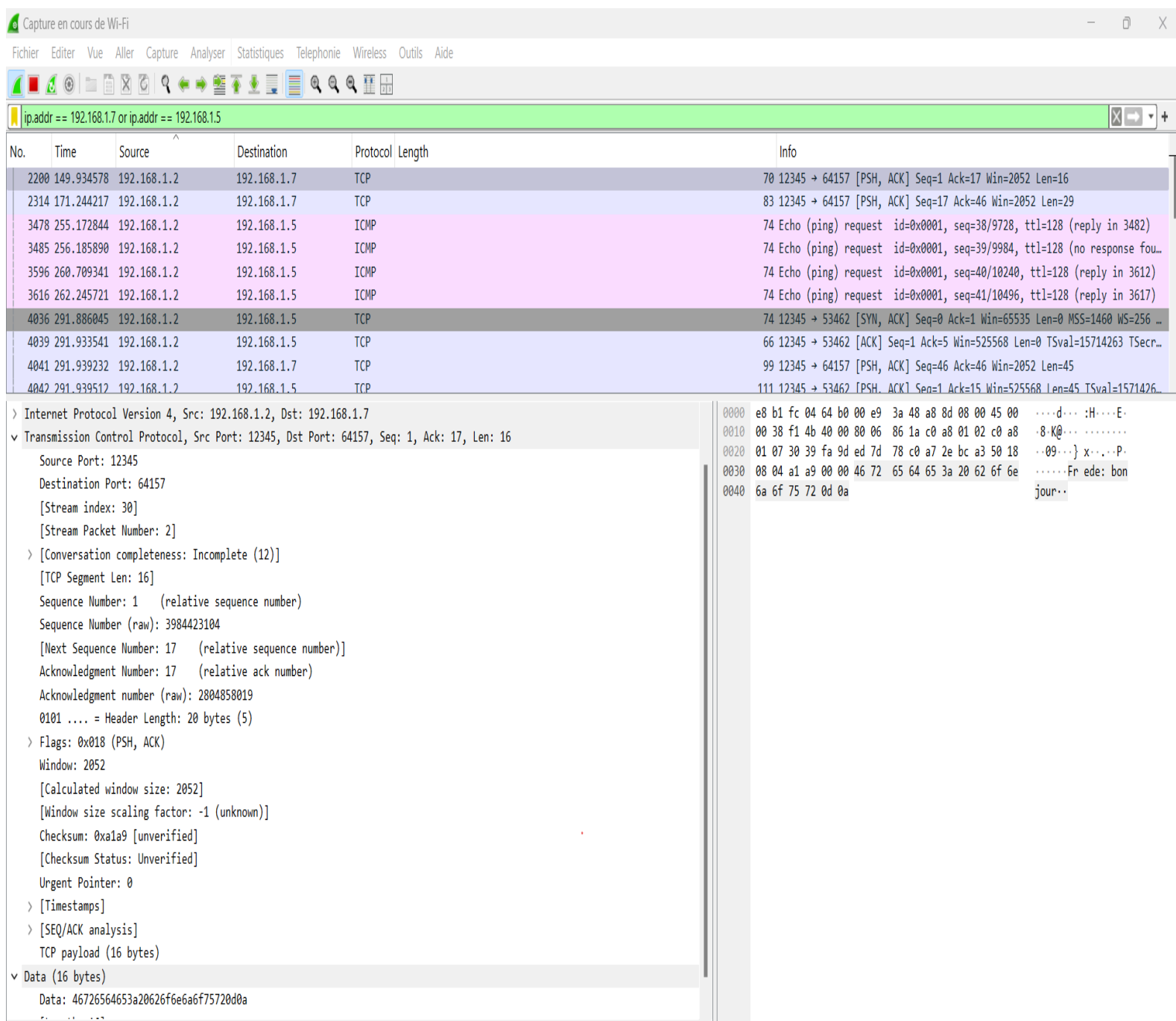


Figure 8 : Capture du segment TCP avec l'outil Wireshark

Cette figure représente une capture du segment TCP, du paquet IP et de la donnée Applicative avec l'outil Wireshark, nous pouvons constater qu'il s'agit d'une communication entre le serveur avec l'adresse IP 192.168.1.2 avec le client 192.168.1.7 par l'onglet à gauche décrivant le protocole IP dans sa version 4. et Nous pouvons voir sur le protocole TCP le port source du serveur 1234 et le port de destination client 64157. Nous pouvons voir le numéro de séquence 1 pour cette transaction, avec un numéro Ack 17. Avec le checksum 0xa1a9 avec plus de détails sur le protocole TCP. Et en observant la partie Data qui est décrite dans la partie droite, nous pouvons voir que l'utilisateur frede a envoyé un message "bonjour".

## 5.2 Résultats des Tests

Les tests ont montré que le système fonctionne correctement dans les conditions de réseau local. Tous les utilisateurs reçoivent les messages en temps réel, et les changements d'état sont correctement propagés à l'ensemble des postes connectés.

## 5.3 Évaluation des Performances

Le système est performant dans un environnement local avec un nombre limité d'utilisateurs (jusqu'à 10 utilisateurs). La latence des messages est négligeable, et le système supporte bien les connexions simultanées.

## 6. Perspectives d'Amélioration

Les perspectives d'évolution de ce projet sont vastes et touchent plusieurs axes d'amélioration comme :

- Groupes de discussion : La création de plusieurs groupes permettrait de structurer les conversations.
- Chat en groupe (conférence) : De plus, l'ajout de la gestion multi-salons (conférence) permettrait de diversifier les interactions entre utilisateurs.
- Partage de fichiers : Pour répondre à des besoins croissants, il serait également pertinent d'envisager d'ajouter une fonctionnalité de partage de fichiers entre utilisateurs permettant l'échange des documents entre users.

Ces améliorations pourraient rendre le système de messagerie plus riche en fonctionnalités et répondre à davantage de besoins.

Ainsi, ce projet pose les bases d'une solution de messagerie décentralisée, avec un potentiel d'évolution vers des systèmes plus complexes et sophistiqués, répondant aux exigences actuelles et futures de la communication instantanée.

## 7. Conclusion

Ce projet de messagerie instantanée interactive sur réseau local a réussi à répondre aux exigences initiales, tout en démontrant la faisabilité d'un système de communication décentralisé, performant et accessible. L'absence d'un serveur central a permis d'augmenter la robustesse du système face aux déconnexions et aux fluctuations des adresses IP, tout en offrant aux utilisateurs une expérience fluide et sans interruption. Grâce à une utilisation efficace des sockets Java pour la communication réseau, l'application gère non seulement l'échange asynchrone de messages, mais aussi la gestion dynamique des utilisateurs, permettant ainsi une mise à jour instantanée de la liste des participants.

Le protocole de communication développé s'est révélé être à la fois flexible et efficace, avec des services bien définis pour l'inscription des utilisateurs, la gestion des statuts (en ligne, occupé, absent, déconnecté) et la transmission de messages en temps réel. De plus, l'approche distribuée garantit que les utilisateurs peuvent entrer et sortir du réseau sans perturber les échanges en cours, ce qui renforce la résilience du système.

Les tests effectués dans un environnement local ont confirmé que le système peut supporter jusqu'à 10 utilisateurs simultanés sans impact notable sur la latence ou la performance globale. Le délai de réponse pour l'envoi et la réception des messages

est resté inférieur à une seconde, démontrant ainsi que l'application remplit son rôle de messagerie instantanée en temps réel.

Ce projet a également permis de consolider des compétences en programmation réseau, en conception de protocoles de communication et en gestion de connexions distribuées. Il a servi de preuve conceptuelle pour une solution de communication décentralisée pouvant être utilisée dans des environnements où la fiabilité et l'autonomie des utilisateurs sont primordiales, notamment dans des contextes d'urgence, des environnements professionnels locaux, ou des événements nécessitant une infrastructure réseau flexible.

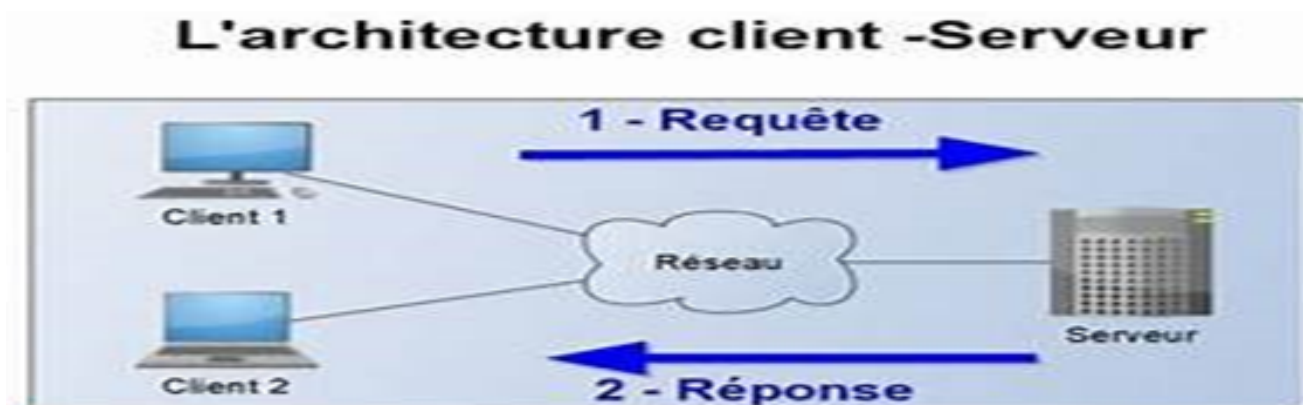
## 8. Documentation de l'Application de Chat

### 8.1 Vue d'ensemble

Ce projet est une application de chat simple basée sur l'architecture client-serveur, où les utilisateurs peuvent se connecter à un serveur central, échanger des messages et mettre à jour leurs statuts. Le serveur maintient la liste des utilisateurs connectés et relaie les messages entre les clients. L'application est implémentée en utilisant le TCP pour la communication et un protocole de message basé sur du texte. Vous trouverez cette documentation en version anglaise dans README.md du travail afin de faciliter l'utilisation. Nous avons souhaité le mettre en anglais car l'anglais est considéré comme la langue internationale de la technologie et de la science.

### 8.2. Architecture du Système

#### 8.2.1. Architecture Client-Serveur



Le système fonctionne selon une architecture client-serveur, où :

- Serveur : Fournit des ressources aux clients. Gère les connexions des clients, maintient la liste des utilisateurs connectés et diffuse des messages à tous les clients connectés.
- Clients : demande une ressource via une interface utilisateur (Interface chat) chargée de la présentation de cette ressource. Il se charge de contacter le serveur pour envoyer ou recevoir les messages.

### 8.2.2. Composants

- Serveur : Nœud principal qui gère toutes les communications entre les clients.
- Clients : Utilisateurs individuels qui se connectent au serveur pour envoyer/recevoir des messages et des mises à jour de statut.

### 8.2.3. Flux de Données

- Connexion Client :
  - Les clients envoient leurs noms d'utilisateur au serveur lors de la connexion.
  - Le serveur maintient une liste globale des utilisateurs connectés et diffuse cette liste à tous les clients chaque fois qu'un changement se produit.
- Diffusion des Messages :
  - Lorsqu'un client envoie un message, le serveur le diffuse à tous les clients connectés avec le nom d'utilisateur de l'expéditeur.
- Mises à Jour de Statut :
  - Les clients peuvent mettre à jour leur statut (par exemple, en ligne, absent, occupé), et le serveur propagera cette information à tous les autres clients.
- Déconnexion du Client :
  - Lorsqu'un client se déconnecte, le serveur met à jour la liste des utilisateurs connectés et diffuse la nouvelle liste à tous les clients.

## 8.3. Protocoles de Communication

### 1. Protocole de Transport : TCP

- Le TCP est utilisé pour assurer une communication fiable entre le serveur et les clients.
- Il garantit que les messages sont livrés dans le bon ordre et sans perte, ce qui est crucial pour les systèmes de chat.

### 2. Protocole de Message (Basé sur du Texte)

La communication entre le serveur et les clients suit un protocole simple basé sur du texte :

- Types de Messages :
  - Connexion Utilisateur :
    - Client → Serveur : Envoie le nom d'utilisateur lors de la connexion.
    - Serveur → Clients : Diffuse la liste mise à jour des utilisateurs connectés.
  - Message Utilisateur :
    - Client → Serveur : Envoie le message.
    - Serveur → Clients : Diffuse le message avec le nom d'utilisateur de l'expéditeur.
  - Mise à Jour de Statut Utilisateur :

- Client → Serveur : Envoie une mise à jour de statut (par exemple, en ligne, absent, occupé).
- Serveur → Clients : Diffusé le statut mis à jour à tous les clients.
- Déconnexion Utilisateur :
  - Client → Serveur : Informe le serveur lorsque l'utilisateur se déconnecte.
  - Serveur → Clients : Met à jour la liste des utilisateurs connectés et la diffuse.

## 8.4. Instructions de Configuration et de Lancement du Projet

Exigences :

- Java : Assurez-vous que Java est installé sur toutes les machines, la version à jour (serveur et clients). Si non tapez:

### ☐ mettre à jour les packages

```
(base) ebwala@ebwala-Latitude-E7450:~/Documents/Reseau/test1/chat_application$ sudo apt update
```

### ☐ installation jdk version 21 (Linux)

```
(base) ebwala@ebwala-Latitude-E7450:~/Documents/Reseau/test1/chat_application$ sudo apt install openjdk-21-jdk
```

### ☐ installation jdk version 21 (Windows)

Télécharger l'archive jdk-21 via le lien ci-dessous, puis installer l'archive en suivant les instructions d'installation.

[https://download.oracle.com/java/21/archive/jdk-21.0.3\\_windows-x64\\_bin.zip](https://download.oracle.com/java/21/archive/jdk-21.0.3_windows-x64_bin.zip) ( sha256 )

- positionnez vous dans le dossier du projet
- Toutes les machines doivent être sur le même réseau local pour les tests:

### 1. compiler

le projet en exécutant la commande suivante à partir de la racine du projet (là où se trouve le fichier build.xml si vous utilisez Ant) :

```
ant compile
```

## 2. Lancer le Serveur :

Trouvez l'adresse IP de votre machine (la machine exécutant le serveur) :

- Sur Linu

```
ifconfig
```

- Sur Windows :

```
ipconfig
```

Exécutez le serveur :

```
java -cp build com.example.ServerGUI
```

## 3. Lancer les Clients

Sur chaque machine cliente : Exécutez dans le CMD le client :

```
java -cp build com.example.LoginWindow
```

Lorsque vous êtes invité à entrer l'adresse IP du serveur, entrez l'adresse IP de la machine

Entrez un nom d'utilisateur lorsqu'on vous le demande.

## 4. Configuration du Pare-feu et du Réseau :

Assurez-vous que le port **12345** (ou le port utilisé) est ouvert pour la communication sur la machine serveur. Vous devrez peut-être configurer le pare-feu :

- Linux (en utilisant ufw) :

```
sudo ufw allow 12345/tcp
```

- Windows : Ouvrez le Pare-feu de Windows Defender et ajoutez une règle d'entrée pour le port TCP **12345**.

Assurez-vous que toutes les machines (serveurs et clients) sont connectées au même réseau local (par exemple, le même réseau Wi-Fi ou Ethernet).

## 5. Tester le Système de Chat :

Une fois tous les clients connectés au serveur :

- Envoyez des messages entre clients pour vérifier la diffusion des messages.
- Changez le statut des clients et observez les mises à jour sur tous les clients connectés.
- Déconnectez des clients et vérifiez que la liste des utilisateurs est mise à jour en conséquence.

### 8.5. Code Source

Le code source est fourni dans l'archive. Vous y trouverez des commentaires détaillés pour chaque partie du programme, ainsi que les fichiers nécessaires à la compilation.

### 8.6. Références :

1. Yahoo Messenger Protocol v9, <http://libyahoo2.sourceforge.net/ymsg-9.txt>
2. MQTT Protocol v3.1.1, <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>