Compte Rendu Projet Réseau

1. Classe AESenc

La classe AESenc fournit des méthodes permettant de chiffrer et déchiffrer des messages en utilisant l'algorithme AES. Elle offre également la possibilité de générer une clé secrète AES. Voici une analyse détaillée de cette classe :

1.1 Attributs

ALGO : Constante représentant l'algorithme de chiffrement utilisé (AES).
KEY_SIZE : Constante représentant la taille de la clé en bits (256 bits).

1.2 Méthodes

1.2.1 encrypt(String message, SecretKey key)

Cette méthode prend en entrée une chaîne de caractères (message) et une clé secrète (key). Elle utilise l'algorithme AES pour chiffrer le message et renvoie un tableau d'octets représentant les données chiffrées.

1.2.2 decrypt(byte[] encryptedData, SecretKey key)

Cette méthode prend en entrée des données chiffrées (encryptedData) et une clé secrète (key). Elle déchiffre les données à l'aide de l'algorithme AES et renvoie le message d'origine sous forme de chaîne de caractères.

1.2.3 generateKey()

Cette méthode génère une clé secrète AES et la retourne.

1.2.4 main(String[] args)

La méthode principale de la classe est utilisée pour tester le chiffrement et le déchiffrement en générant une clé secrète, chiffrant un message, puis le déchiffrant.

1.3 Utilisation

La classe est principalement utilisée pour sécuriser les messages échangés entre le serveur et les clients. Elle démontre également l'utilisation de l'AES pour le chiffrement des données.

2. Classe Client

La classe Client est la principale du côté client. Elle gère l'interface graphique, les interactions utilisateur, la communication avec le serveur et la gestion des messages chiffrés. Voici une analyse approfondie de cette classe :

2.1 Attributs

socket : Instance de la classe Socket pour la connexion au serveur.
inputStream : Flux d'entrée pour recevoir les données du serveur.

outputStream : Flux de sortie pour envoyer des données au serveur.
CurrentToWho : Destinataire actuel du message.
users_lists : Liste des utilisateurs en ligne.
window, cover, main : Composants de l'interface graphique.
host, port, clientName : Champs de saisie pour l'hôte, le port et le nom du client.
run : Bouton pour rejoindre le serveur.
clientColour : Couleur attribuée au client.
labels_cover[] : Tableau de composants d'étiquettes pour l'interface graphique.
server, input : Composants pour afficher le chat et saisir les messages.

2.2 Méthodes

2.2.1 runPanel()

Cette méthode initialise l'interface graphique du client, permettant à l'utilisateur de saisir les détails de connexion et de rejoindre le serveur.

2.2.2 mainInterface()

Méthode interne utilisée pour configurer l'interface principale du client une fois connecté au serveur.

2.2.3 serverReader

Cette classe interne héritée de Thread est utilisée pour lire les messages du serveur de manière asynchrone. Elle décrypte les messages reçus et les affiche dans l'interface graphique.

2.2.4 communicate()

Méthode pour envoyer un message au serveur. Elle chiffre le message à l'aide de la classe AESenc avant de l'envoyer.

2.2.5 joinUser()

Méthode appelée lors de la connexion initiale au serveur. Elle informe le serveur de la présence du nouveau client.

2.2.6 updateOnlineUsers(ArrayList<String> updated_users)

Méthode pour mettre à jour la liste des utilisateurs en ligne à partir des informations reçues du serveur.

2.2.7 closeConnections()

Méthode pour fermer les connexions avec le serveur.

2.3 Utilisation

La classe Client encapsule la logique client complète, gère les interactions utilisateur, et assure la sécurité des messages via le chiffrement AES

3. Classe Serveur

La classe Serveur représente le serveur central qui gère les connexions avec les clients. Elle utilise la classe Transaction pour gérer chaque interaction client-serveur. Voici une analyse approfondie de cette classe :

3.1 Attributs

serverSocket : Instance de la classe ServerSocket pour écouter les connexions des clients.

transactions : Objet de la classe Transaction pour gérer les transactions avec les clients.
outputs : Liste synchronisée des flux de sortie vers les clients.
port_number : Numéro de port sur lequel le serveur écoute.
online_users : Liste des utilisateurs en ligne.
window, cover : Composants de l'interface graphique du serveur.
labels_port, port, run : Composants pour la configuration du serveur.

3.2 Méthodes

3.2.1 runPanel()

Cette méthode initialise l'interface graphique du serveur, permettant à l'utilisateur de spécifier le numéro de port.

3.2.2 runServer()

Méthode pour lancer le serveur et gérer les connexions des clients. Elle crée une instance de la classe Transaction pour chaque client.

3.2.3 main(String[] args)

Méthode principale du programme serveur, elle crée une instance du serveur.

3.2.4 actionPerformed(ActionEvent e)

Gestionnaire d'événements pour les actions utilisateur, notamment le démarrage du serveur.

3.3 Utilisation

La classe Serveur est responsable de la gestion des connexions clients, de la coordination des transactions avec la classe Transaction, et de la gestion des utilisateurs en ligne.

4. Classe SocketConnection

L'interface SocketConnection propose un contrat standard pour les classes gérant des connexions via un socket dans une application réseau en Java. Elle expose deux méthodes essentielles :

communicate() : Gère la communication via le socket, définissant le comportement spécifique à chaque classe concrète.

closeConnections() : Ferme proprement les connexions du socket, libérant les ressources associées.

L'interface favorise une approche modulaire et flexible, séparant clairement les responsabilités liées à la communication et à la fermeture des connexions. Son utilisation encourage la cohérence, la maintenabilité et une conception robuste dans le contexte d'une application réseau.

En adoptant cette interface, les classes concrètes s'engagent à respecter ces normes, offrant ainsi un cadre cohérent pour la gestion des connexions réseau.

5. Classe SynchList

La classe SynchList représente une liste synchronisée d'ObjectOutputStream, permettant au serveur de gérer les connexions avec plusieurs clients simultanément. Voici une analyse approfondie de cette classe :

5.1 Attributs
list : Liste interne d'ObjectOutputStream.

5.2 Méthodes
5.2.1 get(int i)
Méthode pour obtenir l'ObjectOutputStream à un index spécifique dans la liste.

5.2.2 add(ObjectOutputStream o)
Méthode pour ajouter un ObjectOutputStream à la liste.

5.2.3 size()
Méthode pour obtenir la taille de la liste.

5.2.4 remove(ObjectOutputStream o)
Méthode pour supprimer un ObjectOutputStream de la liste.

5.3 Utilisation
La classe SynchList fournit une structure de données permettant au serveur de gérer efficacement les sorties vers plusieurs clients de manière synchronisée.

6. Classe Transaction

La classe Transaction représente une transaction entre le serveur et un client, gérant la communication et la coordination des messages. Voici une analyse approfondie de cette classe :

6.1 Attributs
inputStream : Flux de lecture d'objets depuis le client.
outputStream : Flux d'écriture d'objets vers le client.
socket : Socket de communication avec le client.
outputs : Liste synchronisée des flux de sortie vers les clients.
n : Numéro de transaction.
message : Objet de la classe Message.
online_users : Liste des utilisateurs en ligne.

6.2 Méthodes
6.2.1 run()
Méthode exécutée lorsque le thread est démarré. Elle appelle la méthode communicate pour gérer la communication avec le client.

6.2.2 communicate()

Méthode pour gérer la communication avec le client. Elle lit les messages du client, effectue des traitements spécifiques pour le message @join, puis envoie les messages à tous les clients connectés.

6.2.3 clientLeft(Message m)
Méthode appelée lorsqu'un client quitte la conversation. Elle met à jour la liste des utilisateurs en ligne, informe les clients restants et ferme les connexions avec le client sortant.

6.2.4 closeConnections()
Méthode pour fermer les connexions avec le client.

6.3 Utilisation
La classe Transaction gère la communication avec chaque client de manière indépendante, coordonne les messages entre les clients et prend des mesures spéciales lorsqu'un client quitte la conversation.

7. Classe Message :

La classe Message est une classe Java représentant un message à envoyer via le réseau. Elle est principalement utilisée dans le contexte d'une application de chat sécurisé, où les messages peuvent être chiffrés et destinés à des destinataires spécifiques. Voici une analyse détaillée de cette classe :

1. Attributs

name : Le nom de l'émetteur du message.

message : Le contenu du message, stocké sous forme de tableau d'octets (byte[]), permettant le chiffrement.

toWho : Le destinataire du message.

key : La clé secrète utilisée pour chiffrer le message.

onlineUsers : Une liste des utilisateurs en ligne, synchronisée avec le serveur.

3. Méthodes

3.1 Getters et Setters

getName() : Getter pour le nom de l'émetteur.

getKey() : Getter pour la clé secrète utilisée.

setName(String name) : Setter pour le nom de l'émetteur.

getMessage() : Getter pour le contenu du message.

setMessage(byte[] message) : Setter pour le contenu du message.

getToWho() : Getter pour le destinataire du message.

setToWho(String toWho) : Setter pour le destinataire du message.

addOnlineUser(String s) : Méthode pour ajouter un utilisateur en ligne à la liste.

getOnlineUsers() : Getter pour obtenir la liste des utilisateurs en ligne.

setOnlineUsers(ArrayList l) : Méthode pour définir la liste des utilisateurs en ligne.

3.2 Autres Méthodes

toString() : Méthode pour obtenir une représentation sous forme de chaîne du message. Cette méthode renvoie une concaténation du nom de l'émetteur et du contenu du message.

4. Utilisation

La classe Message est utilisée pour encapsuler les informations d'un message, fournissant des méthodes pour accéder et manipuler ces informations. Elle peut être utilisée dans un contexte de communication sécurisée où le chiffrement des messages est nécessaire.

Fait par : *Kakaev Ibrahim et Motassim Anwar*