Rapport

Application Java – Chat RPC Sécurisé

---

1. Présentation Générale

Cette application est un système de **chat multi-utilisateur** en **Java**, basé sur une architecture **Client-Serveur** avec des **sockets TCP**. Elle implémente une interface graphique avec **Swing**, une sécurité via **SHA-256**, un **filtrage de contenu**, et une gestion **multithread**.

2. Technologies Utilisées

- **Langage** : Java SE
- **Communication** : Sockets TCP (type RPC)
- **Interface graphique** : Java Swing
- **Sécurité** : SHA-256
- **Concurrence** : Threads Java

3. Architecture Globale

Composants Principaux

lightgray **Classe / Fichier**	**Description**
`ChatClient.java`	Gère la connexion au serveur et l'envoi/réception des messages
`LoginFrame.java`	Interface de connexion utilisateur (Swing)
`MainFrame.java`	Interface principale du chat après authentification
`ServerWorker.java`	Thread qui gère les messages d'un client
`SecurityUtils.java`	Hachage SHA-256 des mots de passe
`MessageFilter.java`	Filtre les mots inappropriés
`history.log`	Contient l'historique local du chat

4. Fonctionnalités Détaillées

4.1 Authentification Sécurisée

Le mot de passe est haché en SHA-256 avant envoi, assurant une transmission sécurisée.

4.2 Interface Graphique

`LoginFrame` et `MainFrame` permettent à l'utilisateur de se connecter, lire et envoyer des messages dans une interface conviviale.

4.3 Communication Réseau

Chaque client utilise un `Socket` pour se connecter au serveur sur le port 12345. Les échanges se font via des flux texte, dans un protocole RPC simplifié.

4.4 Filtrage de contenu

La classe `MessageFilter` remplace les mots interdits (comme `puttain`) par `***` pour maintenir un environnement sain.

4.5 Historique de Chat

Les messages sont enregistrés dans `history.log` et rechargés dans l'interface au lancement de l'application.

4.6 Multithreading

Chaque client est géré dans un `Thread` indépendant via `ServerWorker`, assurant une communication fluide en simultané.

5. Détail des Classes

ChatClient.java

Connexion via `Socket`, écoute asynchrone des messages, envoi via `PrintWriter`.

LoginFrame.java

Saisie d'identifiants, appel à `SecurityUtils.hashPassword()`, redirection vers `MainFrame`.

MainFrame.java

Interface utilisateur avec `JTextArea`, `JScrollPane`, champ de saisie, bouton envoyer.

ServerWorker.java

Traite les messages entrants d'un client, filtre, puis rediffuse à tous les autres clients.

SecurityUtils.java

Utilise `MessageDigest` pour appliquer un hachage SHA-256 sur les mots de passe.

MessageFilter.java

Contient la méthode `filter()` qui nettoie les propos offensants.

6. Exemple de Log

```
user: hey
admin: hello
user1: puttain
```

Affichage filtré : `user1: ***`

7. Points Forts

- Authentification avec SHA-256
- Interface Swing fonctionnelle
- Communication en temps réel
- Historique local
- Filtrage de contenu

8. Améliorations Possibles

— Intégration SSL/TLS pour la couche socket
— Enregistrement dans une base de données (SQLite, MySQL)
— Gestion des statuts de présence (connecté/déconnecté)
— Groupes de discussion privés
— Version web (Java avec WebSockets ou REST API)

9. Conclusion

Ce projet de chat sécurisé avec RPC simplifié en Java illustre parfaitement l'usage des sockets, de la sécurité applicative et des interfaces graphiques. Il peut être enrichi facilement pour répondre à des besoins plus professionnels ou pédagogiques.