1. ClientGUISSL.java — Client Java Swing avec SSL/TLS et gestion utilisateurs

java

**import** javax.net.ssl.\*;

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** java.io.\*;

**import** java.security.KeyStore;

**import** java.util.HashMap;

**import** java.util.Map;

*/\*\**

*\* Client messagerie SSL/TLS avec interface graphique Swing.*

*\**

*\* Fonctionnalités :*

*\* - Saisie IP serveur et pseudo utilisateur,*

*\* - Connexion sécurisée via SSLSocket,*

*\* - Liste dynamique des utilisateurs connectés,*

*\* - Gestion multi-fenêtres de chat sans doublons,*

*\* - Envoi et réception des messages texte,*

*\* - Bouton déconnexion propre.*

*\*/*

**public** **class** ClientGUISSL **extends** JFrame {

*// Composants GUI*

**private** JTextField txtIP;

**private** JTextField txtPseudo;

**private** JButton btnConnecter;

**private** JButton btnDeconnecter;

**private** DefaultListModel<String> modelContacts;

**private** JList<String> listContacts;

**private** JTextArea areaConsole;

*// Réseau*

**private** SSLSocket socket;

**private** PrintWriter out;

**private** BufferedReader in;

**private** String nomUtilisateur;

**private** **final** **int** PORT = 12345;

*// Fenêtres de discussion ouvertes : clé = pseudo du contact*

**private** **final** Map<String, FenetreDiscussion> fenetresChats = **new** HashMap<>();

**public** ClientGUISSL() {

**super**("Client Messagerie SSL");

*// --- Panneau de connexion ---*

JPanel panelConn = **new** JPanel(**new** FlowLayout());

panelConn.add(**new** JLabel("Adresse IP serveur :"));

txtIP = **new** JTextField("127.0.0.1", 12);

panelConn.add(txtIP);

panelConn.add(**new** JLabel("Pseudo :"));

txtPseudo = **new** JTextField(10);

panelConn.add(txtPseudo);

btnConnecter = **new** JButton("Connecter");

panelConn.add(btnConnecter);

btnDeconnecter = **new** JButton("Déconnecter");

btnDeconnecter.setEnabled(false);

panelConn.add(btnDeconnecter);

*// --- Liste des contacts ---*

modelContacts = **new** DefaultListModel<>();

listContacts = **new** JList<>(modelContacts);

listContacts.setSelectionMode(ListSelectionModel.SINGLE\_SELECTION);

JScrollPane scrollContacts = **new** JScrollPane(listContacts);

scrollContacts.setPreferredSize(**new** Dimension(200, 300));

JPanel panelContacts = **new** JPanel(**new** BorderLayout());

panelContacts.setBorder(BorderFactory.createTitledBorder("Contacts connectés"));

panelContacts.add(scrollContacts, BorderLayout.CENTER);

*// --- Console pour affichage général ---*

areaConsole = **new** JTextArea(10, 40);

areaConsole.setEditable(false);

JScrollPane scrollConsole = **new** JScrollPane(areaConsole);

*// --- Layout de la fenêtre principale ---*

setLayout(**new** BorderLayout());

add(panelConn, BorderLayout.NORTH);

add(panelContacts, BorderLayout.WEST);

add(scrollConsole, BorderLayout.CENTER);

*// Configuration fenêtre*

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

pack();

setLocationRelativeTo(**null**);

setVisible(true);

*// Actions boutons*

btnConnecter.addActionListener(e -> connecterServeur());

btnDeconnecter.addActionListener(e -> deconnecterServeur());

*// Double-clic sur un utilisateur ouvre fenêtre chat dédiée*

listContacts.addMouseListener(**new** MouseAdapter() {

@Override

**public** **void** mouseClicked(MouseEvent e) {

**if** (e.getClickCount() == 2) {

String pseudoContact = listContacts.getSelectedValue();

**if** (pseudoContact != **null** && !pseudoContact.equals(nomUtilisateur)) {

ouvrirFenetreChat(pseudoContact);

}

}

}

});

}

*/\*\**

*\* Tente de se connecter au serveur avec l'adresse IP et le pseudo saisis.*

*\* Initie SSL et lance le thread d'écoute.*

*\*/*

**private** **void** connecterServeur() {

String ip = txtIP.getText().trim();

String pseudo = txtPseudo.getText().trim();

**if** (ip.isEmpty() || pseudo.isEmpty()) {

JOptionPane.showMessageDialog(**this**, "Veuillez saisir l'adresse IP et un pseudo.", "Erreur", JOptionPane.ERROR\_MESSAGE);

**return**;

}

**try** {

*// Chargement du truststore client (contenant certificat du serveur)*

KeyStore trustStore = KeyStore.getInstance("JKS");

**try** (InputStream ts = **new** FileInputStream("clienttruststore.jks")) {

trustStore.load(ts, "changeit".toCharArray()); *// Adapter mot de passe si besoin*

}

TrustManagerFactory tmf = TrustManagerFactory.getInstance("SunX509");

tmf.init(trustStore);

SSLContext sslContext = SSLContext.getInstance("TLS");

sslContext.init(**null**, tmf.getTrustManagers(), **null**);

SSLSocketFactory ssf = sslContext.getSocketFactory();

socket = (SSLSocket) ssf.createSocket(ip, PORT);

socket.setEnabledCipherSuites(socket.getSupportedCipherSuites());

socket.startHandshake();

out = **new** PrintWriter(**new** OutputStreamWriter(socket.getOutputStream(), "UTF-8"), true);

in = **new** BufferedReader(**new** InputStreamReader(socket.getInputStream(), "UTF-8"));

*// Thread d'écoute des messages serveur*

**new** Thread(**this**::ecouterServeur).start();

*// Gestion boucle authentification simple*

String reponse;

**do** {

reponse = in.readLine();

**if** (reponse == **null**) **throw** **new** IOException("Pas de réponse du serveur");

**if** (reponse.equals("SUBMITNAME")) {

out.println(pseudo);

} **else** **if** (reponse.equals("NAMEINUSE")) {

JOptionPane.showMessageDialog(**this**, "Pseudo déjà utilisé, choisissez-en un autre.", "Erreur", JOptionPane.ERROR\_MESSAGE);

socket.close();

**return**;

} **else** **if** (reponse.startsWith("NAMEACCEPTED")) {

nomUtilisateur = pseudo;

areaConsole.append("Connecté en tant que " + nomUtilisateur + "\n");

SwingUtilities.invokeLater(() -> {

btnConnecter.setEnabled(false);

btnDeconnecter.setEnabled(true);

txtIP.setEnabled(false);

txtPseudo.setEnabled(false);

});

}

} **while** (!reponse.startsWith("NAMEACCEPTED"));

} **catch** (Exception ex) {

ex.printStackTrace();

JOptionPane.showMessageDialog(**this**, "Erreur de connexion : " + ex.getMessage(), "Erreur", JOptionPane.ERROR\_MESSAGE);

}

}

*/\*\**

*\* Ecoute les messages en provenance du serveur et met à jour l'interface.*

*\*/*

**private** **void** ecouterServeur() {

**try** {

String ligne;

**while** ((ligne = in.readLine()) != **null**) {

**if** (ligne.startsWith("USER\_LIST ")) {

String[] users = ligne.substring(10).split(" ");

SwingUtilities.invokeLater(() -> {

modelContacts.clear();

**for** (String user : users) {

modelContacts.addElement(user);

}

});

} **else** **if** (ligne.startsWith("MSG ")) {

String reste = ligne.substring(4);

**int** sep = reste.indexOf(' ');

**if** (sep > 0) {

String expediteur = reste.substring(0, sep);

String msg = reste.substring(sep + 1);

SwingUtilities.invokeLater(() -> {

ouvrirFenetreChat(expediteur).afficherMessage(expediteur + " : " + msg);

});

}

}

}

} **catch** (IOException e) {

SwingUtilities.invokeLater(() -> {

areaConsole.append("Déconnecté du serveur.\n");

resetUI();

});

}

}

*/\*\**

*\* Ouvre ou remet au premier plan la fenêtre de discussion pour un contact donné.*

*\*/*

**private** FenetreDiscussion ouvrirFenetreChat(String pseudoContact) {

FenetreDiscussion fen = fenetresChats.get(pseudoContact);

**if** (fen == **null**) {

fen = **new** FenetreDiscussion(pseudoContact, **this**);

fenetresChats.put(pseudoContact, fen);

fen.addWindowListener(**new** WindowAdapter() {

@Override

**public** **void** windowClosed(WindowEvent e) {

fenetresChats.remove(pseudoContact);

}

});

fen.setVisible(true);

} **else** {

fen.toFront();

fen.requestFocus();

}

**return** fen;

}

*/\*\**

*\* Envoi un message au serveur.*

*\* Ici, pas de protocole avancé : juste envoi du texte.*

*\*/*

**public** **void** envoyerMessage(String destinataire, String texte) {

**if** (out != **null**) {

out.println(texte);

areaConsole.append("Vous à " + destinataire + " : " + texte + "\n");

}

}

*/\*\**

*\* Déconnexion propre du serveur.*

*\*/*

**private** **void** deconnecterServeur() {

**if** (out != **null**) {

out.println("LOGOUT");

}

**try** {

**if** (socket != **null**)

socket.close();

} **catch** (IOException e) {

*// ignore*

}

areaConsole.append("Déconnecté.\n");

resetUI();

*// Ferme toutes les fenêtres de chat ouvertes*

fenetresChats.values().forEach(Window::dispose);

fenetresChats.clear();

}

*/\*\**

*\* Réinitialisation interface après déconnexion.*

*\*/*

**private** **void** resetUI() {

SwingUtilities.invokeLater(() -> {

btnConnecter.setEnabled(true);

btnDeconnecter.setEnabled(false);

txtIP.setEnabled(true);

txtPseudo.setEnabled(true);

modelContacts.clear();

});

}

*/\*\**

*\* Getter du pseudo connecté.*

*\*/*

**public** String getNomUtilisateur() {

**return** nomUtilisateur;

}

*// Point d’entrée*

**public** **static** **void** main(String[] args) {

SwingUtilities.invokeLater(ClientGUISSL::**new**);

}

}

2. FenetreDiscussion.java — Fenêtre de chat individuelle

java

**import** javax.swing.\*;

**import** java.awt.\*;

**import** java.awt.event.\*;

*/\*\**

*\* Fenêtre de discussion par utilisateur.*

*\* Permet d'envoyer et afficher les messages.*

*\*/*

**public** **class** FenetreDiscussion **extends** JFrame {

**private** **final** String contact;

**private** **final** ClientGUISSL client;

**private** JTextArea areaDiscussion;

**private** JTextField txtMessage;

**private** JButton btnEnvoyer;

**public** FenetreDiscussion(String contact, ClientGUISSL client) {

**super**("Chat avec " + contact);

**this**.contact = contact;

**this**.client = client;

areaDiscussion = **new** JTextArea(15, 30);

areaDiscussion.setEditable(false);

JScrollPane scrollPane = **new** JScrollPane(areaDiscussion);

txtMessage = **new** JTextField(25);

btnEnvoyer = **new** JButton("Envoyer");

JPanel panelBas = **new** JPanel(**new** FlowLayout());

panelBas.add(txtMessage);

panelBas.add(btnEnvoyer);

setLayout(**new** BorderLayout());

add(scrollPane, BorderLayout.CENTER);

add(panelBas, BorderLayout.SOUTH);

btnEnvoyer.addActionListener(e -> envoyerMessage());

txtMessage.addActionListener(e -> envoyerMessage());

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

pack();

setLocationRelativeTo(**null**);

}

*/\*\**

*\* Envoi message tapé au contact.*

*\*/*

**private** **void** envoyerMessage() {

String msg = txtMessage.getText().trim();

**if** (!msg.isEmpty()) {

*// Envoi direct le texte (le serveur diffuse)*

client.envoyerMessage(contact, msg);

afficherMessage("Vous : " + msg);

txtMessage.setText("");

}

}

*/\*\**

*\* Affiche un message dans la zone de chat.*

*\*/*

**public** **void** afficherMessage(String msg) {

areaDiscussion.append(msg + "\n");

}

}

3. Fichiers nécessaires à placer dans le dossier Windows client

* ClientGUISSL.java
* FenetreDiscussion.java
* clienttruststore.jks — Le truststore client contenant le certificat du serveur (à créer/importer avec keytool, correspond au certificat serveur généré côté serveur)

SECURITE SSL

*# Sur la machine cliente (Windows) :*

keytool -import -alias serveur -file serveur.cert -keystore clienttruststore.jks

**COMPILATION + CREATION EXECUTABLE .JAR**

2. Pour le **client** (exemple avec ClientGUISSL.java et FenetreDiscussion.java)

**Étapes :**

1. Ouvre une invite de commande ou PowerShell dans le dossier où se trouvent ClientGUISSL.java, FenetreDiscussion.java et le truststore clienttruststore.jks.
2. Compile les fichiers source Java :

text

javac ClientGUISSL.java FenetreDiscussion.java

1. Crée un fichier manifeste texte manifest.txt contenant (avec une ligne vide à la fin) :

text

Main-Class: ClientGUISSL

1. Crée un JAR exécutable incluant tous les fichiers .class du dossier :

text

jar cfm ClientGUISSL.jar manifest.txt \*.class

1. Pour lancer le client ensuite :

text

java -jar ClientGUISSL.jar

En résumé

| **Étape** | **Commande / Action** |
| --- | --- |
| Compiler | javac ServeurConnexionSSL.java |
| Créer manifeste | Fichier manifest.txt avec Main-Class |
| Créer jar | jar cfm ServeurConnexionSSL.jar manifest.txt \*.class |
| Lancer jar | java -jar ServeurConnexionSSL.jar |

Si tu veux, je peux te fournir un script ou un guide détaillé pour préparer un J