Distributed Chat System

Luis Fernando Moreno **2ºHV student**

Introduction:

In this project, we have developed a chat system where multiple clients can connect to a server, create and join chat rooms, and send messages to other clients connected to the same chat room.

The system uses Java sockets for communication between the server and the clients, and includes features such as password authentication, room creation, and encryption of messages using RSA encryption algorithms.

The project is divided into two main parts: the server and the client. The server is responsible for handling client connections, creating chat rooms, and managing the communication between clients. The client, on the other hand, allows users to connect to the server, create or join chat rooms, and send and receive messages.

Overall, the chat system provides a secure and efficient way for users to communicate with each other in real-time.

Classes used:

Conexion:

Provides the functionality to establish a connection between a client and a server using sockets. It allows the creation of either a server or a client socket. The server socket is created on the specified port, while the client socket is created using the IP address and port number of the server. The Conexion constructor takes a parameter specifying whether the socket to be created is for a server or a client.

MainServidor:

Class in charge of initializing the server

MainCliente:

Class in charge of initializing the client

Servidor:

Functions as the server in the chat system, establishing a socket and waiting for client connections, launching a thread for each connected client. It also keeps track of the chat rooms and the connected clients, and notifies the server console when a new client is connected. The "iniciar()" method is responsible for waiting for client connections and launching threads for each connected client.

ThreadServidor:

This class manages the communication between each client and the server. It receives commands from the client and calls methods accordingly. It creates input and output channels to communicate with the client and establishes public and private keys for secure communication. It listens for messages from the client and evaluates each message, calling the appropriate method in each case. It can join a client to a room, create a new room, list all available rooms, and end a session.

Cliente:

This class represents each client that wants to connect to the server. It manages communication with the server and the options that are shown to each client as a user interface.

The **startClient()** method is called when the client is initialized. It creates the client's public and private keys, creates input and output channels to communicate with the server, and receives and prints the connection confirmation message from the server. It prompts the user to set a nickname using the **selectNickname()** method, and then enters a while loop that displays a menu of options to the user and prompts them to choose an option. It calls the appropriate method depending on the user's selection, until the user chooses to close the session.

Other methods include **seleccionarSala()** prompts the user to enter the name of an existing room, checks that it doesn't contain spaces, and sends the name to the **unirseASala** method. There are also methods for creating a new room, listing existing rooms, changing the user's nickname, and closing the session.

HiloClienteEscucha:

This class is a thread that listens to incoming messages from the server and prints them to the client. It takes a **DataInputStream** object, a **Cliente** object, and a **nombreSala** string as parameters in its constructor. It enters an infinite loop that continuously reads incoming messages from the server via the **DataInputStream**. If the message received is "LEAVE_ROOM", the loop is broken and the thread is terminated. Otherwise, the message is printed to the client's console after decrypting it with the **cliente's desencriptar()** method.

HiloClienteEscribe:

Java thread responsible for sending messages written by the client to the server. It receives input from the client through a Scanner object and writes messages to the server using a DataOutputStream object.

An infinite loop is opened waiting for the client to write a message. Each message written by the client is sent to the server using the out.writeUTF() method, after being encrypted with the Client’s encrypt method. If the message exceeds the maximum length of 140 characters, an error message is displayed. If the client writes the message "LEAVE_ROOM", the loop is broken and the thread is finished.

Sala:

The class Sala represents a chat room in the server. It has fields for the room ID, privacy status, password (if it is private), the list of client names connected to the room, and a list of server threads (one per client). It has methods for verifying the password of a private room, generating a random ID for the room, and sending a message to all clients in the room except the sender. It also has getters and setters for its fields, and an overridden toString() method for displaying information about the room.

Conclusion:

In conclusion, the system presented is a simple chat server and client application that allows multiple clients to join different chat rooms and communicate with each other. The system is implemented in Java using sockets and threads, and it includes features such as password-protected rooms and message encryption. While the system is not designed for large-scale deployment, it demonstrates fundamental concepts of network programming and provides a solid foundation for building more complex chat applications.

Bibliography:

Some of the sources that I have consulted for the development of this practice are:

https://www.baeldung.com/java-random-string

https://stackoverflow.com/questions/7733270/java-public-key-different-after-sent-over-socket

https://stackoverflow.com/questions/52384809/public-key-to-string-and-then-back-to-public-key-java

https://es.stackoverflow.com/questions/142933/error-al-desencriptar-rsa-java

https://stackoverflow.com/questions/10007147/getting-a-illegalblocksizeexception-data-must-not-be-longer-than-256-bytes-when