Sockets + JavaFX

# Server side:

Class Server : une classe qui permet de créer une instance de Socket server :

Code :

package com.example;

import java.net.ServerSocket;

import java.util.ArrayList;

public class Server extends Thread {

    private ServerSocket serverSocket;

    private int port;

    static ArrayList<ClientHandler> clients = new ArrayList<ClientHandler>();

    private String id;

    public Server(int port) {

        this.port = port;

        this.start();

    }

    public void run() {

        try {

            serverSocket = new ServerSocket(port);

            System.out.println("Server started on port " + port);

            while (true) {

                System.out.println("Waiting for a client ...");

                ClientHandler client = new ClientHandler(serverSocket.accept(), id);

                clients.add(client);

                System.out.println("Client accepted");

                for (ClientHandler c : clients) {

                    System.out.println("new client " + c.id);

                }

            }

        } catch (Exception e) {

            System.err.println("Server exception: " + e.toString());

            e.printStackTrace();

        }

    }

    public static ClientHandler getClient(String idd) {

        for (ClientHandler c : clients) {

            if (c.id.equals(idd)) {

                return c;

            }

        }

        return null;

    }

    public static void SendListOfClients() {

        String list = "";

        for (ClientHandler c : clients) {

            list += c.id + ",";

        }

        for (ClientHandler c : clients) {

            System.out.println("sending list to " + c.id);

            c.sendMessage("server" + " " + list + " " + c.id);

        }

    }

}

Class Client Handler : permet de lancer un thread pour chaque client :

Code :

package com.example;

import java.io.DataInputStream;

import java.io.DataOutputStream;

import java.io.IOException;

import java.net.Socket;

public class ClientHandler extends Thread {

    Socket serverSocket;

    private DataInputStream in = null;

    public String id;

    private static DataOutputStream out = null;

    public ClientHandler(Socket serverSocket, String id) {

        try {

            System.out.println("ClientHandler created");

            this.serverSocket = serverSocket;

            this.id = id;

            in = new DataInputStream(serverSocket.getInputStream());

            out = new DataOutputStream(serverSocket.getOutputStream());

            this.start();

        } catch (IOException i) {

            System.out.println(i);

        }

    }

    public void run() {

        System.out.println("new ClientHandler started for client " + id + " ...");

        getUserId();

        String line = "";

        while (!line.equals("Over")) {

            try {

                line = in.readUTF();

                // split the line

                String c[] = line.split(" ");

                System.out.println(line);

                if (c[1] != "exit") {

                    String idReceiver = c[2];

                    System.out.println(" receiver id " + idReceiver);

                    Server.getClient(idReceiver).sendMessage(line);

                } else {

                    Server.getClient(id).sendMessage("exit");

                    break;

                }

            } catch (IOException i) {

                System.out.println(i);

                break;

            }

        }

        System.out.println("Closing connection");

        CloseConnection();

    }

    public void getUserId() {

        try {

            out.writeUTF(" Server : Enter your id ");

            setId(in.readUTF());

            System.out.println("ClientHandler id set to " + id);

            Server.SendListOfClients();

        } catch (IOException i) {

            System.out.println(i);

        }

    }

    public void sendMessage(String message) {

        try {

            out.writeUTF(message);

            System.out.println("Message sent: " + message);

        } catch (IOException i) {

            System.out.println(i);

        }

    }

    public void CloseConnection() {

        try {

            serverSocket.close();

            in.close();

            out.close();

        } catch (IOException i) {

            System.out.println(i);

        }

    }

    public void setId(String id) {

        this.id = id;

        System.out.println("ClientHandler id set to " + id);

    }

}

# Client SIDE :

Class Hello application :

public class HelloApplication extends Application {

    @Override

    public void start(Stage stage) throws IOException {

        FXMLLoader fxmlLoader = new FXMLLoader(HelloApplication.class.getResource("hello-view.fxml"));

        Scene scene = new Scene(fxmlLoader.load(), 320, 240);

        stage.setTitle("Hello!");

        stage.setScene(scene);

        stage.show();

    }

    public static void main(String[] args) {

        launch();

    }

}

Class App Controller :

public class HelloController {

    @FXML

    private Label welcomeText;

    @FXML

    private TextField nameField;

    @FXML

    protected void onHelloButtonClick(ActionEvent event) throws Exception {

        // get the root from the event

        Node node = (Node) event.getSource();

        // get the stage from the root

        Stage stage = (Stage) node.getScene().getWindow();

        // create a new FXMLLoader

        FXMLLoader loader = new FXMLLoader(getClass().getResource("chat.fxml"));

        // load the FXML file

        Parent root = loader.load();

        // get the controller

        chatController controller = loader.getController();

        // create an instance of the client

        Client client = new Client("localhost", "5000", nameField.getText(), controller);

        // set the scene to the stage

        stage.setScene(new Scene(root));

        // show the stage

        stage.show();

    }

}

Class Client : permet de se connecter avec le server avec un objet de type socket :

public class Client extends Thread {

    private static Socket socket = null;

    private static DataInputStream input = null;

    private static DataOutputStream out = null;

    private static String port = null;

    private static String host = null;

    private static String identity = null;

    private chatController controller;

    public Client(String host, String port, String identity, chatController controller) {

        if (socket != null) {

            return;

        }

        this.host = host;

        this.port = port;

        this.identity = identity;

        this.controller = controller;

        System.out.println("controller " + controller);

        this.start();

    }

    public void run() {

        try {

            socket = new Socket(host, Integer.parseInt(port));

            input = new DataInputStream(System.in);

            out = new DataOutputStream(socket.getOutputStream());

            System.out.println("Connected");

            DataInputStream serverInput = new DataInputStream(socket.getInputStream());

            System.out.println(serverInput.readUTF());

            out.writeUTF(identity);

            System.out.println("Send identity");

            while (true) {

                System.out.println("Waiting for server response");

                String response = serverInput.readUTF();

                handleResponse(response);

            }

        } catch (IOException e) {

            System.out.println("Error occurred: " + e.getMessage());

        }

    }

    public static void sendMessage(String message) {

        try {

            System.out.println("Sending message from client");

            out.writeUTF(identity + " " + message);

        } catch (Exception e) {

            System.out.println(e);

        }

    }

    public void handleResponse(String response) {

        System.out.println("Handling response");

        String responseArray[] = response.split(" ");

        String sender = responseArray[0];

        String message = responseArray[1];

        String receiver = responseArray[2];

        controller.addMessage(sender + ": " + message);

    }

}

Class chat Controller permet la gestion de chat :

public class chatController {

    @FXML

    private VBox chatBox;

    @FXML

    private TextField messageBox;

    @FXML

    private TextField destinationBox;

    public void SendMessage() {

        String message = messageBox.getText() + " " + destinationBox.getText();

        Client.sendMessage(message);

        messageBox.clear();

        destinationBox.clear();

    }

    public void addMessage(String message) {

        Platform.runLater(() -> {

            chatBox.getChildren().add(new Label(message));

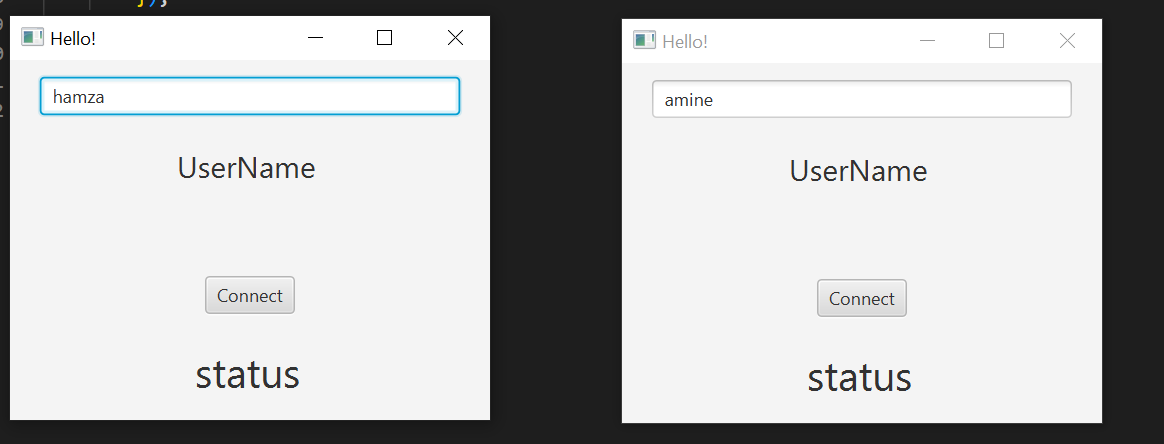
        });

    }

}

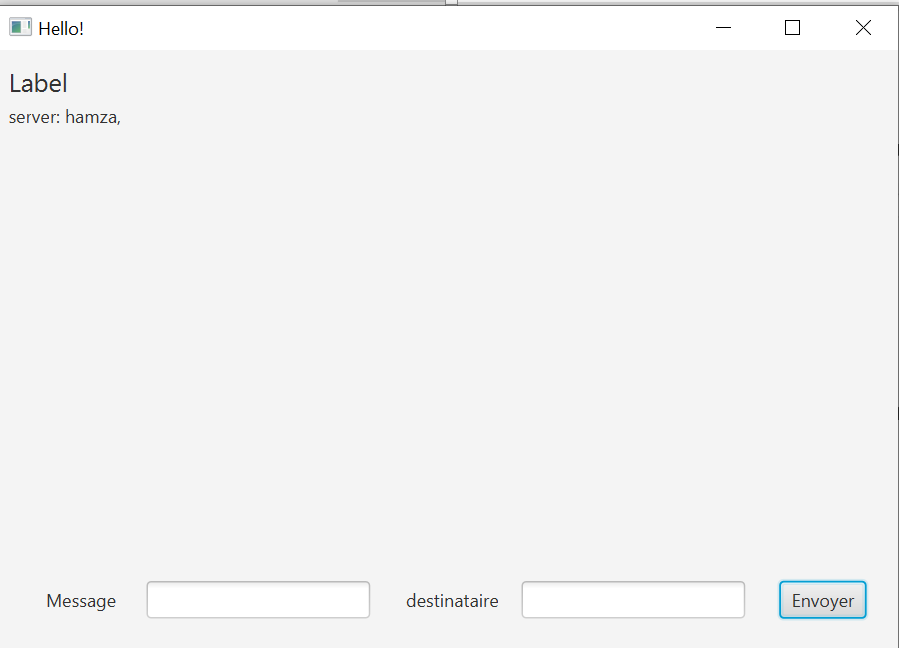
# Test d’exécution :

Etablissement de la connexion :



Interface principale :

Apres la connexion le server envoi une liste des clients connectées au client



Test envoi de message :

