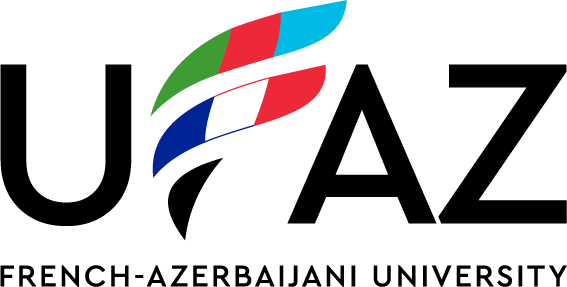
**Students:** Ibrahim Aliyev, Elnur Maharramov

**Subject:** Distributed computing

**Topic:**  Chat application project

**Description:** This report presents the development and implementation of a chat application using sockets, with the client built in Java and the server implemented in Node.js. We also made application which built only in Java (both server and client). The application enables real-time communication between multiple clients and facilitates instant messaging over a network connection. The report provides an overview of the architecture, design, and implementation details of the chat application, along with instructions for running the application and potential areas for future enhancements.



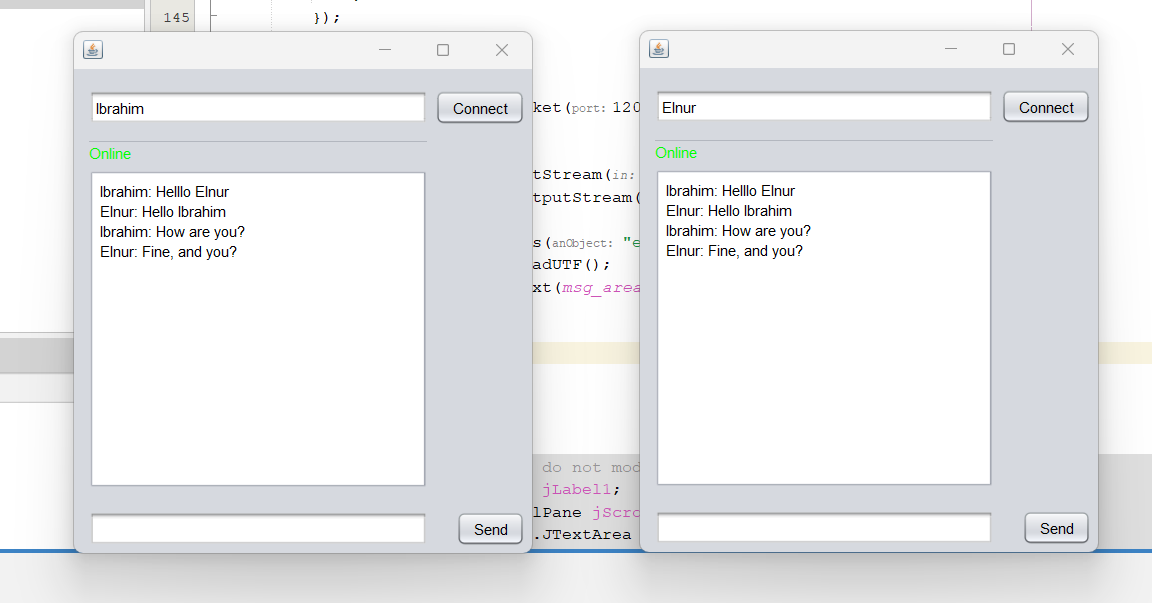
# Introduction

In today's interconnected world, real-time communication plays a vital role in various domains. Chat applications allow users to exchange messages instantly, enabling efficient collaboration and interaction. This project aims to develop a socket-based chat application that utilizes Java as the client-side programming language and Node.js as the server-side platform. In addition to that, we will talk on the development and implementation of a chat application using Java sockets for both the client and server components.

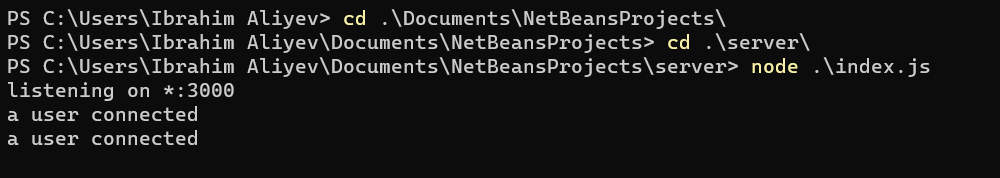
# Development and Implementation of a Socket-based Chat Application with Java Client and Node.js Server

**Architecture and Design:** The chat application follows a client-server model, where the Java-based client communicates with the Node.js server using sockets. The application employs TCP/IP sockets to establish a reliable and persistent connection between the client and the server. The overall architecture can be divided into two major components: the client-side application and the server-side application.

**Client-side Application:** The client-side application is developed using Java and provides the user interface for sending and receiving messages. It establishes a socket connection with the server, listens for incoming messages, and sends user-generated messages to the server for distribution to other connected clients.



**Server-side Application:** The server-side application, built using Node.js, manages the communication between multiple clients. It listens for incoming connections from clients and maintains a list of connected clients. When a client sends a message, the server broadcasts it to all connected clients, facilitating real-time communication.



**Implementation Details:** The client-side application is developed using Java's socket API, while the server-side application is implemented using the net module provided by Node.js. The Java client establishes a socket connection to the server using the server's IP address and port number. Once connected, it can send and receive messages through the established socket.

The Node.js server creates a TCP server instance that listens for incoming connections. Upon connection, it adds the client to a list of connected clients and assigns a unique identifier. Messages received from clients are then broadcasted to all other connected clients.

**Deployment and Usage:** To deploy and use the chat application, follow these steps:

Server Setup:

Install Node.js on the machine that will act as the server.

Obtain the source code of the server-side application.

Open a terminal and navigate to the server directory.

Run the command: npm install to install the required dependencies.

Start the server using: node index.js.

Client Setup:

Install the Java Development Kit (JDK) on the client machine.

Obtain the source code of the client-side application.

Open the project in an Integrated Development Environment (IDE) such as Eclipse or IntelliJ.

Compile and run the Java client application.

# Java Sockets Chat Application (Client and Server in Java)

**Client-side Application:** The client-side application is developed entirely in Java, leveraging the Java socket API for establishing a connection with the server and exchanging messages. It provides a user interface where users can input and receive messages in real-time.

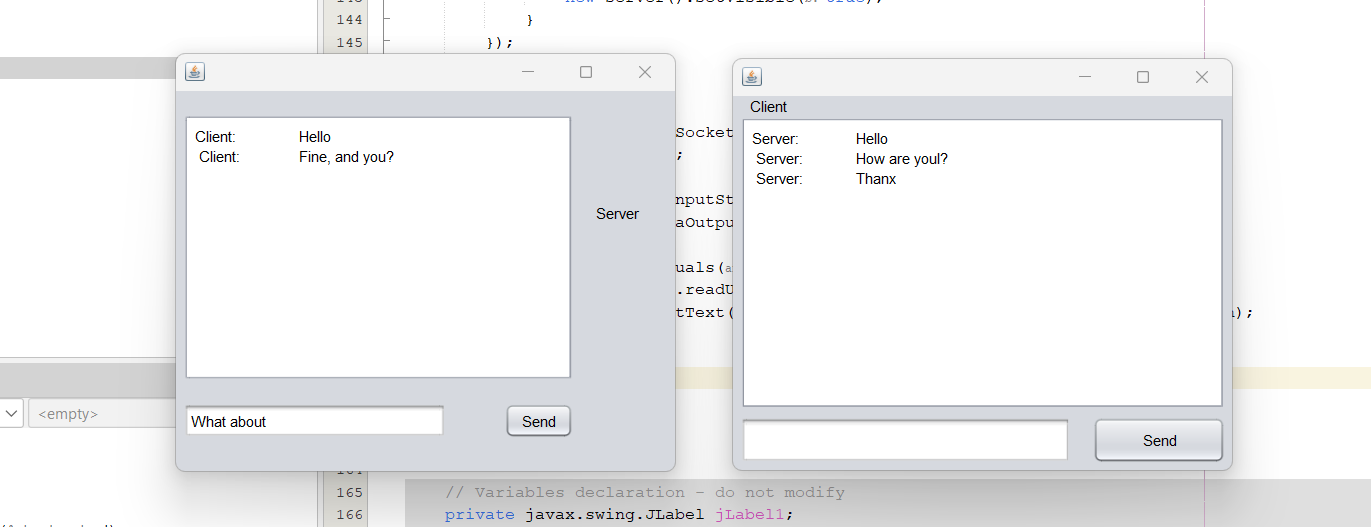
The client application consists of the following key functionalities:

**Socket Connection:** The client establishes a socket connection with the server by providing the server's IP address and port number. It uses the Socket class to create a socket object and initiate the connection.

**Message Sending:** Once the connection is established, the client can send messages to the server using the output stream obtained from the socket. User-generated messages are sent as byte arrays or strings, which are then transmitted to the server.

**Message Receiving:** The client continuously listens for incoming messages from the server using the input stream obtained from the socket. It receives the messages and displays them in the user interface.

**User Interface:** The client application provides a graphical user interface (GUI) using Java's Swing or JavaFX libraries. It includes text fields for message input and a chat window to display the conversation history.



**Server-side Application:** The server-side application is also developed using Java sockets, handling incoming connections from clients and managing message distribution.

The server application encompasses the following functionalities:

**Socket Binding:** The server creates a ServerSocket object and binds it to a specific IP address and port number. It listens for incoming connection requests from clients.

**Client Connections:** When a client attempts to connect, the server accepts the incoming connection by creating a new socket object using the accept() method of the ServerSocket class. It maintains a list of connected clients, each represented by a separate socket.

**Message Broadcasting:** When a client sends a message to the server, the server receives the message and broadcasts it to all connected clients. It iterates through the list of sockets representing connected clients and sends the message using the respective output streams.

# Conclusion

In conclusion, the development and implementation of socket-based chat applications using Java and Node.js provide reliable and efficient solutions for real-time communication. The client-server architecture and the use of sockets enable seamless message exchange between multiple clients. By following the provided deployment and usage instructions, users can run the chat applications and explore opportunities for further enhancements based on specific requirements.

The chat applications can be further enhanced to include features such as user authentication and authorization, private messaging, group chats, an improved user interface, file transfer capabilities, and message history with persistent storage. These enhancements would make the chat system more secure, versatile, and user-friendly.