**Assignment 2:**

**Real- Time Client- Server Messaging Application**:  
Here is a client- server messaging application built using java oop. It allows a client and server to exchange messages over a network using sockets.

**Components:**  
1. Message:

**Attributes:**

messageContent: the message itself.

messageId: Generates a unique Id for each message  
status: Indicates if a message is seen or not  
timestamp: The time the message was sent.

Methods:  
toString(); Returns a formatted String with message details.

**2. MessagingApp:**

Manages a list of messages and provides methods to add, display, delete and shuffle messages etc using **Array list** in java.

**Methods**:

addMessage(); Adds the message sent to the list.

displayMessages(); Displays all messages.

searchMessage(); Searches for a message by its Id.

deleteMessage();Deletes a message by its Id.

displayUnseenMessages(); and displaySeenMessages();

shuffleMessages(); Randomly shuffles the messages.

Networking Components:  
**1.Client class:**  
The client can send and receive messages from the server.

**Features:**  
It connects to the server at a specific IP and Port.

Allows the user to send and receive messages from server in real time.

Includes options to display, search, delete, and shuffle messages stored locally in the MessagingApp instance.

**User- Options:**  
Send Message:

Sends a typed message to the server.

Exit:

Disconnects from the server and closes the client.

**2. Server class:**

Purpose:  
Represents the server that can receive and respond to client messages   
User- Options:  
Send Message:

Sends a typed message to the server.

Exit:

Disconnects from the client and closes the server.

**Application Workflow:**

**Server Class:**

1. Server Startup and Client connection:

The server is initialized by running the server class. It begins by waiting for a **client connection** using **ServerSocket** on **port 5555.**

A client successfully connects to the server and connection is established. The server accepts this with the **accept()** method, which blocks until a client connects.

1. Server Options (Send Message or exit):

The server presents a menu with two options to the server.

To send a message to the client or exit the application.

1. Receiving messages from client

The server receives three messages from the client. These messages are displayed on the server console and added to the MessagingApp instance as "seen" messages, as they were received by the server.

1. Server sending the messages:

The server user chooses option 1 to send a message back to client. The message is transmitted to the client using **DataOutputStream**. The server sends messages repeatedly.

1. Client exits:

The client initiates exit by choosing 2. The server also terminates its own connection and stopping the application.

1. Servers Message Management Options:

After exiting the connection, the server allows the user to manage the stored messages through a set of options that include displaying, searching, deleting, shuffling, and filtering messages.

1. Displaying Messages:

This displays all the messages which are sent and receive by the server.

**Here the important feature is that the messages which are sent by server are marked unseen while the messages received by server from client are marked seen.**

1. Message Search:

The server searches for a message by ID, showing its details upon a successful search. Message ID "2" is found with the content "How are you?" and is marked as "Seen" along with a timestamp.

1. Delete Messages:
2. Shuffling Messages:

The server user selects the option to shuffle the messages, which rearranges the messages in random order and displays them with IDs, content, status, and timestamps.

1. Displaying Seen and Unseen Messages:

1. 7 to Exit:

**Socket and Server Socket:**

The Server Socket is initialized on a specific port (5555), which allows the server to listen for incoming connections. Once a client connects, the server accepts it using the accept() method, creating a Socket for communication.

**Data Streams (DataInputStream and DataOutputStream)**

These streams handle data transfer between the client and server. DataInputStream reads messages sent from the client, while DataOutputStream writes messages from the server to the client.

**Threading (Server Runnable)**

The server uses a separate Runnable thread (Server Runnable) to listen for incoming messages from the client not occurring at the same time. This allows the server to process client messages in the background while also handling user inputs for the server's console commands.

**Client Class:**client class has the same things it implements the same menu and functions. The key difference in Client class is:  
**1. Socket Connection:**

The client uses a socket to connect to the servers IP for example:(192.168.223.71) and port (5555) enabling the communication channel.

**2. Data Streams**:  
DataInputStream and DataOutputStream manage incoming and outgoing data. DataInputStream reads messages from the user. DataOutputStream sends messages to the server.

**3. Threading (Client Runnable):**

The client has a separate ClientRunnable thread, which constantly listens for messages from the server. This thread runs in the background, allowing the client to both send messages and display server responses without blocking.

**User Interaction and Menu Options**

The client console provides two main options: sending messages to the server or exiting the application. Upon exiting, the client can also access a menu to manage messages locally (view, search, delete, shuffle, etc.), similar to the server.