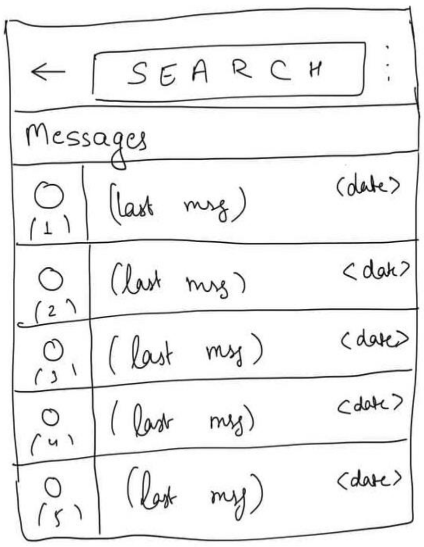
What Is Messaging?

Messaging is a method of communication between software components or applications. A messaging system is a peer-to-peer facility: A messaging peer can send messages to, and receive messages from, any other peer. Here we engage facilities for creating, sending, receiving, and reading messages.

A component sends a message to a destination, and the recipient can retrieve the message from the destination. However, the sender and the receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver, nor does the receiver need to know anything about the sender. The sender and the receiver need to know only which message format and which destination to use. The B4 Message Service is a Java API that allows to create, send, receive, and read messages using React JS as a user interface.

Pseudo Code for Messaging Module

Frontend implementation of messaging module:



- Click on “Message” in the main window.
- **To Send/Compose the message:**

[Create a new session using session.getdefaultinstance( )].

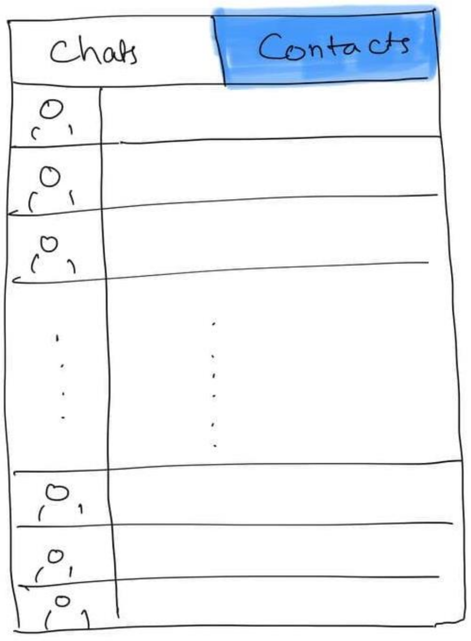
- Set SMS window true.
  - Click on “**BACK**” - to return to main window.
  - Click on “**SEARCH**” - get search from glue code
  - Click on the “**:**” - pop up for new chat, new group, and settings. Settings will have another pop up that has option of “mute the incoming message/No notification” etc.

a. Click on **“TO” – to select the receiver of the message**

[Initialising http request; causing the Document Object Model (DOM) elements to change.]

b. **Get the address book** from Glue Code (using Address Book upload).

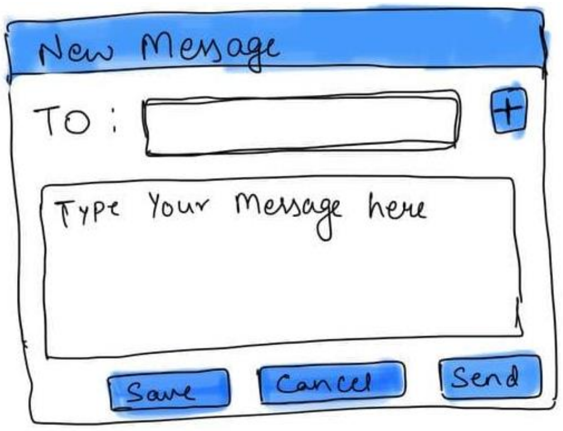
c. Select the receiver(s) from the data base; can be one or more.
(To select **multiple recipients**, string array is created.
string [ ] recipients = newString [Size])



- **Generate the hash ID** of the selected receivers.
[using SHA algorithm; SHA1.getSHA1()]

- **SET TEXT** : Click on "**Type your Message**" to craft the message
[sms.setText(" ")]

- Click on
    a. "**Save"** as Draft [sms.saveChanges( )] to save the crafted message;
    b. "**Cancel**" to abort the session.
    c. "**Send**" to send the message to (all) the selected recipients
    (If there is **no recipient** selected in the "To" list, then give a pop-up and ask the user to select at least one recipient, and when the user closes the pop-up, the user sees the previous screen only which was already open all the while, and the message is not send in that case.) Here message will get automatically saved in the respective recipient(s) conversation.
- **Encrypt the entire message**.
(Encrypting involves adding information of Name and IP Address of the node to which it will go, and the information that the packet has to be finally handed over to the Messaging Module of that node.)
- Put the encrypted message in the output buffer of the Messaging System, which is the **input buffer of the Glue Code**.



- Display "Message Status":
    a. if "Sent" - message sent successfully
    b. if "Cancel" - message is discarded
- Close the session and return to main window.

- **To Receive the message:**

- **Notify** whenever there is message in Input buffer of messaging module. (generates a popup “Message received”)
  1. Fetch details of the sender, and **send delivery report** to glue code stating that the message is successfully delivered.

  2. **Decrypt** the entire message and save it in the conversation of the respective username.

  3. **Display** the message.

  4. If Reply, go to “SET TEXT” in sender thread.

  5. Else close the session and return to main window.

- **To Delete a message:**
  - Select the message. (press and hold)
  - Display Delete option.
  - Click on delete and confirm
  - Display: Message deleted. Message selected will be flushed and can’t be recovered back.

Backend implementation of messaging Module:

Basic concept involved in this module is: one user/node sends a message to the other user/node, and the other user/node reads the message and displays it. Sending of the message is treated as Request whereas the message received is the response.

Concepts of OOPS, Multithreading and layering are used here. Threads in java programming are used in order to give smooth and fast processing, it allows multiple things to operate efficiently at the same time, in messaging module of B4 two threads; sender and receiver are made. Since we aspire simultaneous sending and receiving.

The concept of Layering is used to store conversation from multiple peers and to save their respective messages into a layer.

Here we need to use services of authentication manager to get the Date and Time value in a generic format, i.e. common for all modules of B4, services of routing manager to get IP address of source and destination, services of search module to perform search operation wherever required, services of web GUI to access user input data. All these services are accessed with the help of Glue code, the main module of the B4 system.

Code snippet:

```
package com.company;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.nio.charset.StandardCharsets;;
//import com.ehelpy.brihaspati4.gc;

public class sms{

    public static void main(String msg, String emailID){
        final String user1;
        final String user2;
        final BufferedReader in;
        final BufferedReader out;
        final int chatID;
        final String sc;

        try{
            //user1 = System.getProperty("user.name");
            //user2 =
            //string [ ] recipients = newString [Size])
            out = new BufferedReader(user2.getOutputStream());
            in = new BufferedReader(new
InputStreamReader(user1.getInputStream()));
            boolean seen = false;

        Thread sender = new Thread(new Runnable() {

            byte[] sendData;
            public InetAddress sendPacket;
            String msg;

            @Override
             public void run() {
                  while(true){
                  msg = sc.nextLine();
                  if(!seen){
                      sendData = msg.getBytes();
                  //call routing for sender and receiver IP address fetch
                            InetAddress IPAddress = sendPacket.getAddress();
                            DatagramPacket Packet = new
DatagramPacket(sendData, sendData.length, chatID, IPAddress);
                  //serverSocket.send(Packet);
                             System.out.println(msg);
                      //call datetimecheck from authenticate manager via
glue code

 // public static void main(String[]) throws IOException {
//     DateTimeFormatter dtf = DateTimeFormatter.ofPattern("DD/MM/YYYY
HH:mm:ss");
//    LocalDateTime now = LocalDateTime.now();
//    System.out.println(dtf.format(now));
//}
                        }
                  else{
                      // send delivery report
                      return;
                  }
                  out.close();
                  }
            }
        });
        sender.start();
        //sleep when sending window not activated

        Thread receive = new Thread(new Runnable() {
            String msg;
            public InetAddress receivePacket;

            @Override
            public void run() {
                try{
                    //call routing for IP address fetch
                    InetAddress IPAddress = receivePacket.getAddress();
                    seen = true;
                        //call sender thread
                        //send delivery report to sender when flag is true

                    byte[] receiveData;
                    DatagramPacket Packet = new DatagramPacket(receiveData,
receiveData.length, chatID, IPAddress);
                    msg = new String(receiveData, StandardCharsets.UTF_8);

                    while(msg!=null){
                        //System.out.println("Client : "+msg);
                        msg =in.readLine();
                        //display msg
                    }
                    //System.out.println("Client offline");

                    out.close();
                    //window close();
                }catch (IOException e){
                    e.printStackTrace();
                }
            }
        });
        receive.start();
        //receive thread to run in background

        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

Backend implementation of messaging Module (06.01.22)

Basic concept involved in this module is: one user/node sends a message to the other user/node, and the other user/node reads the message and displays it. Sending of the message is treated as Request whereas the message received is the response

~~Here I have used concepts of **Socket programming** in Java, which is used to establish a communication between the applications running at two nodes and~~ **Java threads**, that allows multiple things to efficiently operate at the same time, in this case I have used thread for sending and receiving messages. Since we aspire simultaneous sending and receiving.

~~**Two types of sockets** are used in this module, one that is engaged with the connection requests while the other that help in sending and receiving messages.~~

Two classes are made initially, a server class and a client class.

Server Class

```
//required packages
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class Server {
    public static void main(String[] args){
        final ServerSocket serverSocket;     //final keyword used so that the
corresponding variable is made constant.
        final Socket clientSocket ;
        final BufferedReader in;    //to read input buffer
        final PrintWriter out;    //to write msg
        final Scanner sc = new Scanner(System.in);  //to read input from
user's keyboard

        try{
            //serverSocket = new ServerSocket(port:3000);  //using a port
to build a connection and listen to the connection requests
            clientSocket = serverSocket.accept();    // to accept the
connection requests
            out = new PrintWriter(clientSocket.getOutputStream());
//send msg to client
            in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));    //read msg from client,
input msg is fetched , then converted into bytes(InputStreamReader) and then into
```

characters(BufferedReader)

//Sender Thread will get input msg and send it to the client.

```
        Thread sender = new Thread(new Runnable() {
            String msg;
            @Override
            public void run() {
                while(true){ //send to?
                    msg = sc.nextLine();
                    out.println(msg);
                    out.flush();
                }
            }
        });
        sender.start();
```

//Receive Thread will display msg received from the client.

```
        Thread receive = new Thread(new Runnable() {
            String msg;
            @Override
            public void run() {
                try{
                    msg = in.readLine();

                    while(msg!=null){
                        System.out.println("Client : "+msg);
                        msg =in.readLine();
                    }
                    System.out.println("Client offline");

                    out.close();
                    clientSocket.close();
                    serverSocket.close();
                }catch (IOException e){
                    e.printStackTrace();
                }
            }
        });
        receive.start();

        }catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Client Class:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

```java
import java.io.PrintWriter;
import java.net.Socket;
import java.util.Scanner;

public class Client {
    public static void main(String[] args){
        final Socket clientSocket ;
        final BufferedReader in;
        final PrintWriter out;
        final Scanner sc = new Scanner(System.in);
        try{
            clientSocket = new Socket( host:"202.3.77.204",port: 3000);
            out = new PrintWriter(clientSocket.getOutputStream());
            in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            Thread sender = new Thread(new Runnable() {
                String msg;
                @Override
                public void run() {
                    while(true){
                        msg = sc.nextLine();
                        out.println(msg);
                        out.flush();
                    }
                }
            });
            sender.start();

            Thread receive = new Thread(new Runnable() {
                String msg;
                @Override
                public void run() {
                    try{
                        msg = in.readLine();
                        while(msg!=null){
                            System.out.println("Server : "+msg);
                            msg =in.readLine();
                        }
                        System.out.println("Server unavailable");
                        out.close();
                        clientSocket.close();
                    }catch (IOException e){
                        e.printStackTrace();
                    }
                }
            });
            receive.start();
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Messaging module (30.12.21)

Entire message code is divided into two sub codes, one being the java code and other the React JS code. React JS code is written for the user interface or the frontend design whereas java code is for the backend processing of the selected commands.

Click on the message from the main window and send the response to open the messaging tab. This is handled by the Web UI and displayed on the browser hence, done using react js code.

To fetch the address book or contacts, the query from the react js code is sent to glue code which redirects to the sms java code (backend), which will further send the query to the glue code that address book is required by sms, the glue code then fetches the contacts by calling address book and returns to the sms java code and display it using web UI.

Messaging Module pseudo code(23.12.21)

reactJS- threading(start,stop) and socket programming , OOPS concept

```
//import the necessary packages
//public class sms extends JFrame{
    //session.getDegaultInstance()
    //JFrame for “back” and “settings”
    private JTextField searchText;      //search field
    private JTextArea chatList;         //all chats display area
    private JTextField userText;      //type your message field
    private JTextArea chatWindow;     //message thread area
    private ObjectOutputStream output;   //output stream
    private ObjectInputStream input;       //input stream
    public ComposeSMS(){
    //JFrame for display of “address Book”
    string[] recipients = newString{};
    //hash = SHA1.getSHA1();     //generate Hash id
    public setSMS(){
        userText = new JTextField();
        userText.setEditable(true);
        userText.addActionListener(
            new ActionListener(){
                public void actionPerformed(ActionEvent
            event){
```

```
                        sendSMS(event.getActionCommand());
                        cancelSMS(event.getActionCommand());
                        userText.setText(“”);
                        saveAsDraftSMS(event.getACtionComma
                    d());
                        }
                    }
                ):

        add(userText,    BorderLayout.NORTH);        //display    the
    message
        chatWindow = new JtTextArea();
        add(new JScrollPane(chatWindow));   //display the chat
        setSize(500,350);
        setVisible(true);

    }
}

public  receiveSMS extends JFrame{
string username = query.sender.name;
//send delivery report to username
string msg = query.sender.msg;
receivedText = decrypt(query , key);
```

```
display receivedText;
receivedText.addActionListener(
        new ActionListener(){
                public void actionPerformed(ActionEvent event){
                                if(replySMS(event.getActionCommand()))
                                        setSMS();
                                elseif(backSMS(event.getActionComman
                        d()));
                                        setVisible(false);
                                else
                  setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
                                //close the session;
                                }
                        }
                    ):
}
```

Messaging Service (SMS) Pseudo Code(16.12.21-updated)

- Click on “Message” in the main window.
- **To Send/Compose the message:**

[Create a new session using session.getdefaultinstance( )].

- Set SMS window true.
  - Click on “**BACK**” - to return to main window.
  - Click on “**SEARCH**” -
  - Click on the “**:**” - pop up for new chat, new group, and settings. Settings will have another pop up that has option of muting the messages, profile

    d. Click on **“TO” – to select the receiver of the message**

    [Initialising http request; causing the Document Object Model (DOM) elements to change.]

    e. **Get the address book** from Glue Code (using Address Book upload).

    f. Select the receiver(s) from the data base; can be one or more.
    (To select **multiple recipients**, string array is created.
    string [ ] recipients = newString [Size])

- **Generate the hash ID** of the selected receivers.
  [using SHA algorithm; SHA1.getSHA1()]

- **SET TEXT** : Click on “**Type your Message**” to craft the message
  [sms.setText(“ “)]

- Click on
  d. “**Save as Draft**” [sms.saveChanges( )] to save the crafted message;
  e. “**Cancel**” to abort the session.
  f. “**Send**” to send the message to all the selected recipients
  (If there is no recipient selected in the “To” list, then give a pop-up and ask the user to input at least one recipient, and when the user closes the pop-up, the user sees the previous screen only which was already open all these while, and the message is not send in that case.) Here

message will get automatically saved in the respective recipient(s) conversation.

- **Encrypt the entire message**.
  (Encrypting involves adding information of Name and IP Address of the node to which it will go, and the information that the packet has to be finally handed over to the Messaging Module of that node.)

- Put the encrypted message in the output buffer of the Messaging System, which is the **input buffer of the Glue Code**.

- Display "Message Status":
  c. if "Sent" - message sent successfully
  d. if "Cancel" - message is discarded

- Close the session and return to main window.

- **To Receive the message:**
- **Notify** whenever there is message in Input buffer of messaging module. (generates a popup "Message received")

6. Fetch details of the sender and send delivery report to glue code stating that the message is successfully delivered.//

7. Decrypt the entire message and save it in the conversation of the respective username.

8. Display the message.

9. To Reply , go to "SET TEXT"

10. Else close the session and return to main window.

- **To Delete a message:**
- Select the message. (press and hold)
- Display Delete option.
- Click on delete
- Display : Message deleted