# COMPUTER NETWORKS

# (EC-330)

PROJECT REPORT

<u>SUBMITTED BY</u>:

| | |
|---|---|
| AMMAR AMJAD | 123471 |
| SYED MOHAMMAD ALI | 147434 |
| QAZI TALHA HAMID | 126076 |

CE-37-A

11th January 2018

## Contents

PRIMARY AIMS: ........ 1

OVERVIEW: ........ 1

STARTING THE SERVER: ........ 2

CONNECTING THE CLIENT: ........ 5

GROUP MESSAGING: ........ 9

REMOVING MEMBERS FROM CHAT: ........ 10

WORKING OF ANDROID CLIENT: ........ 12

EXTRAS: ........ 17

OFFLINE MESSAGING: ........ 17

SOCIAL NETWORK: ........ 17

SOFTWARE TOOLS USED: ........ 25

## Primary Aims:

Design a group chat application (preferably for android).

- The application must allow addition of new members to the chat room
- The application must  allow removing of members from chat room
- The messages sent to the chat group must be delivered to all group members

## Overview:

To achieve these primary aims, we have made a Client-Server model using TCP Sockets. For the Client, we have developed an Android App that runs on any Android Device and the Server is developed using Java and can run on any Device that supports Java.

1. When the Server is started, it opens a Java Server Socket on port 6000 and Listens for incoming connection requests by the Android Clients. The Clients will try to connect to the Server at port 6000 and join the group chat. All previously connected Clients will be notified of the joining of new Members.

2. The Server maintains a password that can be changed before starting the Server. The admin of the group chat is whoever knows that password. The admin can then remove any member from the group chat, by messaging a command. For example the password is “password” and Client 0 is the admin and wants to

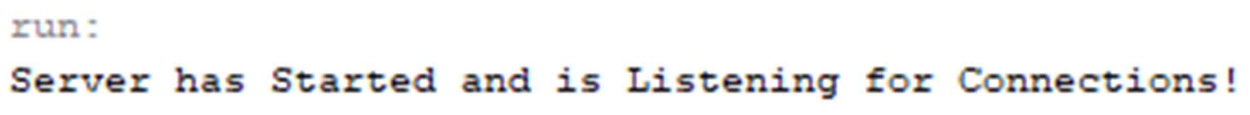
remove Client 1, the command he will message will be "@K1Ppassword" without the quotation marks. The Server will recognize this is a command to it and will not show it in other Clients' chat. 'K' in the command means kick out the client number specified after it, and 'P' in the command means with password specified after it. If the command is given in correct syntax with correct password, the Server will close the Socket of Client 1 and everyone in the group chat will be notified that Client 1 has been removed by Client 0.

3. Any member of the group chat can send a message and it will be sent to the Server, which will then re-transmit it to all connected Clients except the one from whom it was received. The Server can also take part in the group chat, however it has been given the flexibility to choose who it wants to send the message. It can send a message to any particular client or all the clients.

Now let's look in detail how we are accomplishing these three primary aims.

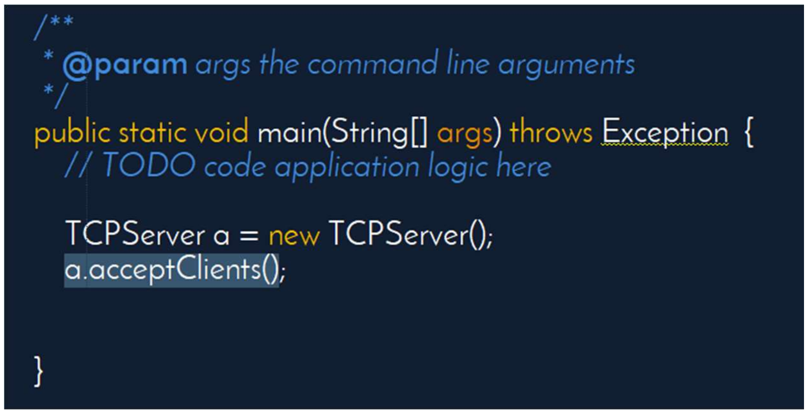
Starting the Server:

Once the Server is started, it will notify



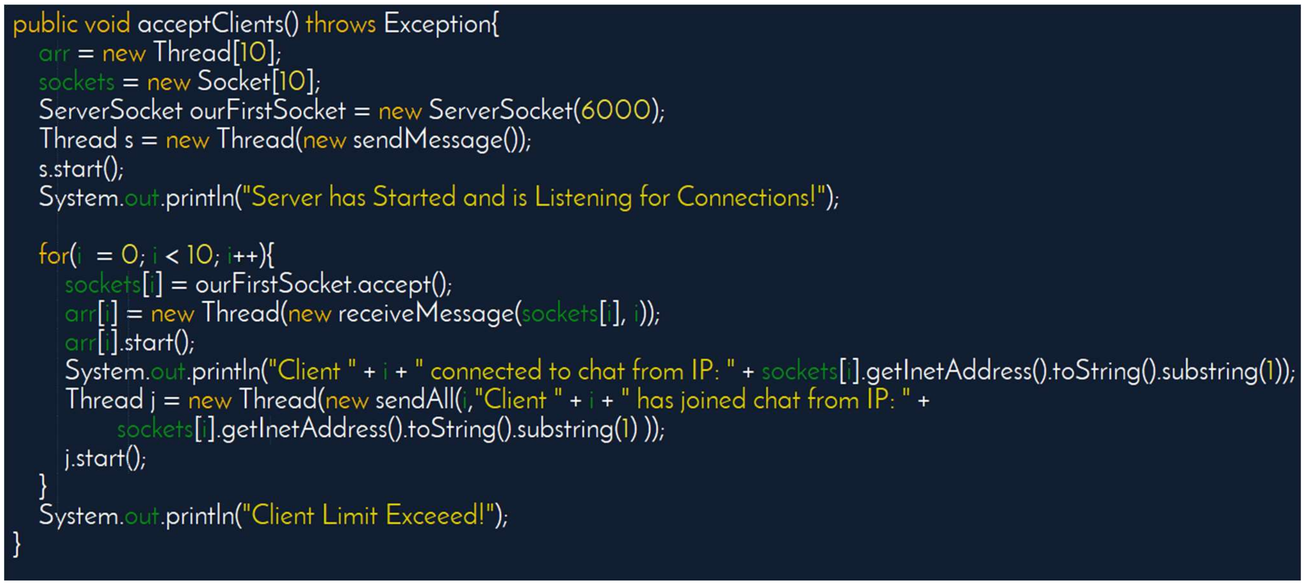
```
run:
Server has Started and is Listening for Connections!
```

On startup of our Server named TCPServer.java, an Instance of TCPServer class is made and its method acceptClients is called.



```
/**
 * @param args the command line arguments
 */
public static void main(String[] args) throws Exception {
    // TODO code application logic here

    TCPServer a = new TCPServer();
    a.acceptClients();


}
```

This is the acceptClients method.

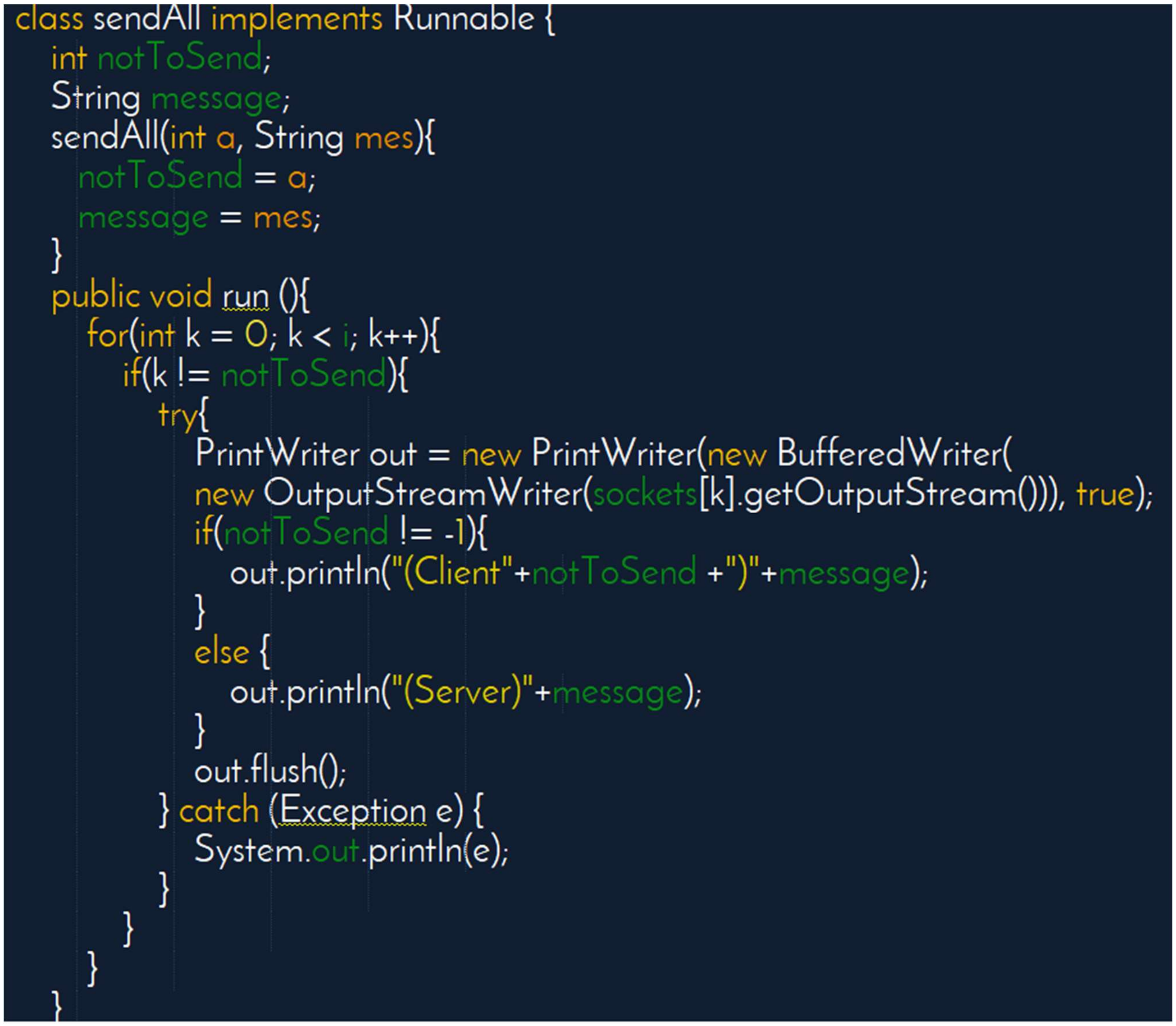


```
public void acceptClients() throws Exception{
    arr = new Thread[10];
    sockets = new Socket[10];
    ServerSocket ourFirstSocket = new ServerSocket(6000);
    Thread s = new Thread(new sendMessage());
    s.start();
    System.out.println("Server has Started and is Listening for Connections!");

    for(i  = 0; i < 10; i++){
        sockets[i] = ourFirstSocket.accept();
        arr[i] = new Thread(new receiveMessage(sockets[i], i));
        arr[i].start();
        System.out.println("Client " + i + " connected to chat from IP: " + sockets[i].getInetAddress().toString().substring(1));
        Thread j = new Thread(new sendAll(i,"Client " + i + " has joined chat from IP: " +
                sockets[i].getInetAddress().toString().substring(1) ));
        j.start();
    }
    System.out.println("Client Limit Exceeed!");
}
```

Arrays of Thread and Socket are maintained globally. The Server Socket listens for incoming connections in a loop with a length similar to that of arrays and when a client connects, it will be called Client 'i' where 'i' is the iteration number of the loop. Its socket will be stored on location 'i' of Socket array and a receiveMessage Method will be started for that client in a Thread stored at location 'i' of Thread array.

This way everything stays organized. An independent thread will also run a sendAll method to tell previously connected clients that a new client has connected.

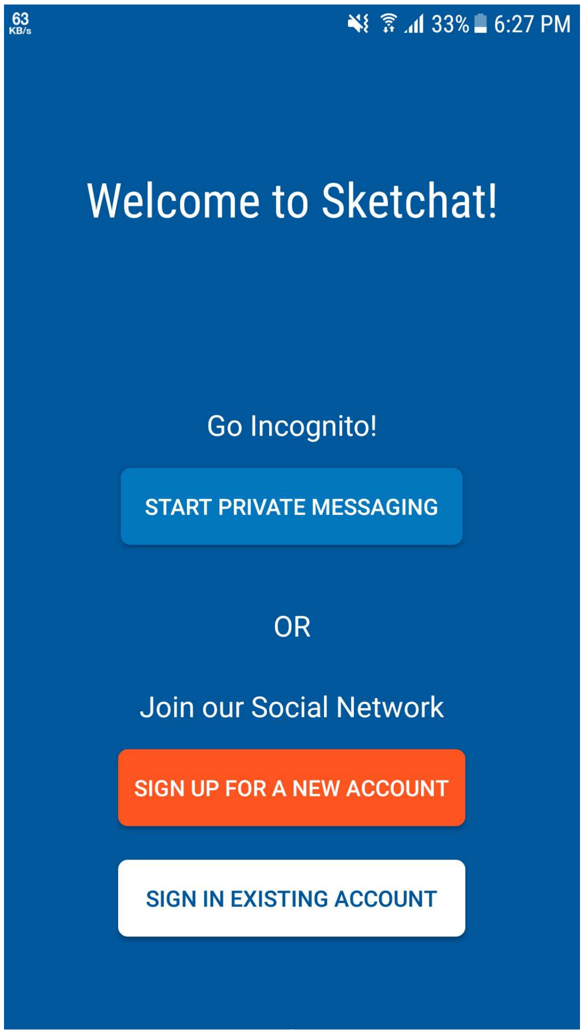
This is the sendAll method.

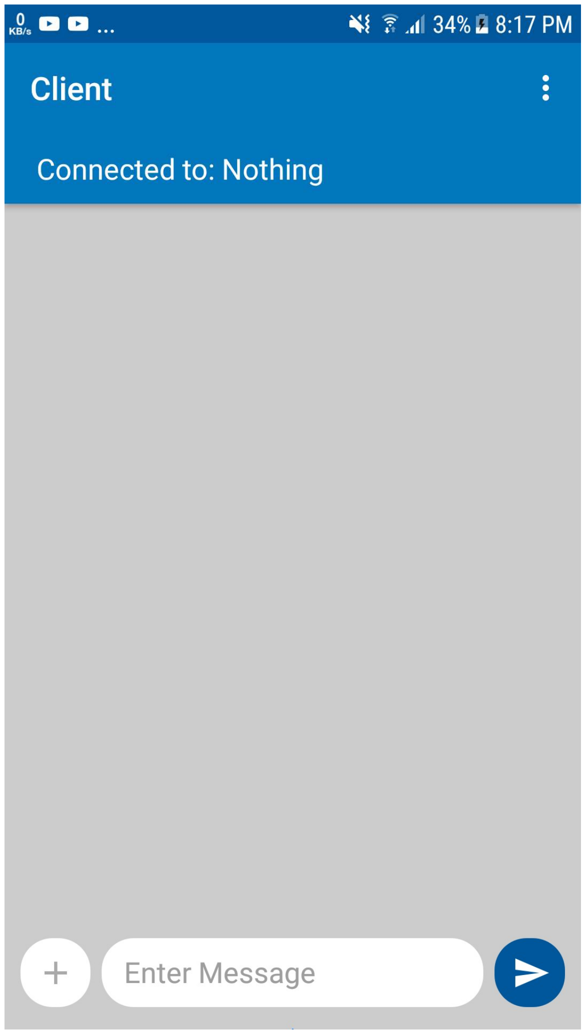


```
class sendAll implements Runnable {
   int notToSend;
   String message;
   sendAll(int a, String mes){
      notToSend = a;
      message = mes;
   }
   public void run (){
      for(int k = 0; k < i; k++){
         if(k != notToSend){
            try{
               PrintWriter out = new PrintWriter(new BufferedWriter(
               new OutputStreamWriter(sockets[k].getOutputStream())), true);
               if(notToSend != -1){
                  out.println("(Client"+notToSend +")"+message);
               }
               else {
                  out.println("(Server)"+message);
               }
               out.flush();
            } catch (Exception e) {
               System.out.println(e);
            }
         }
      }
   }
```

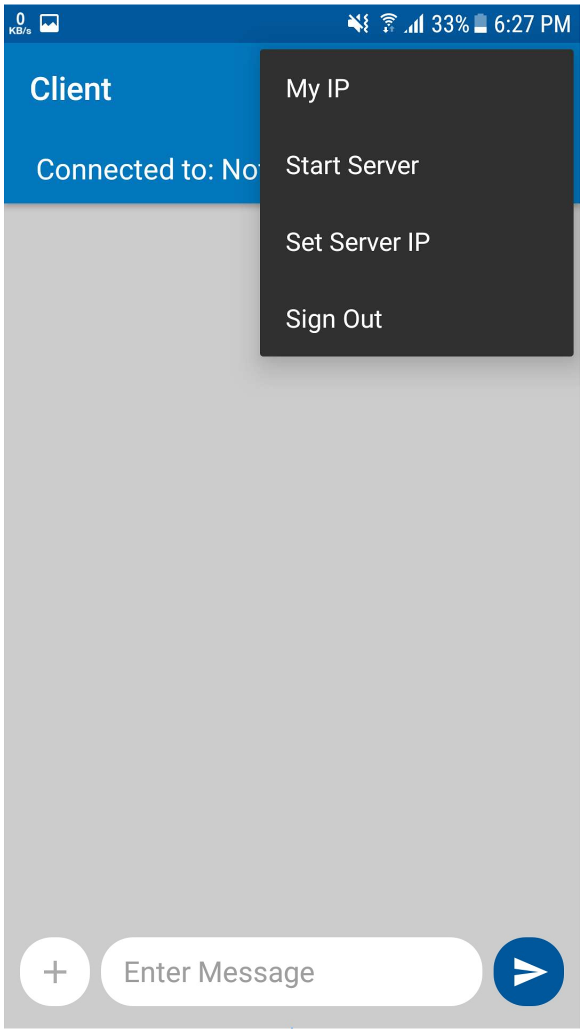
Now let's look at the client's side how it makes the connection.

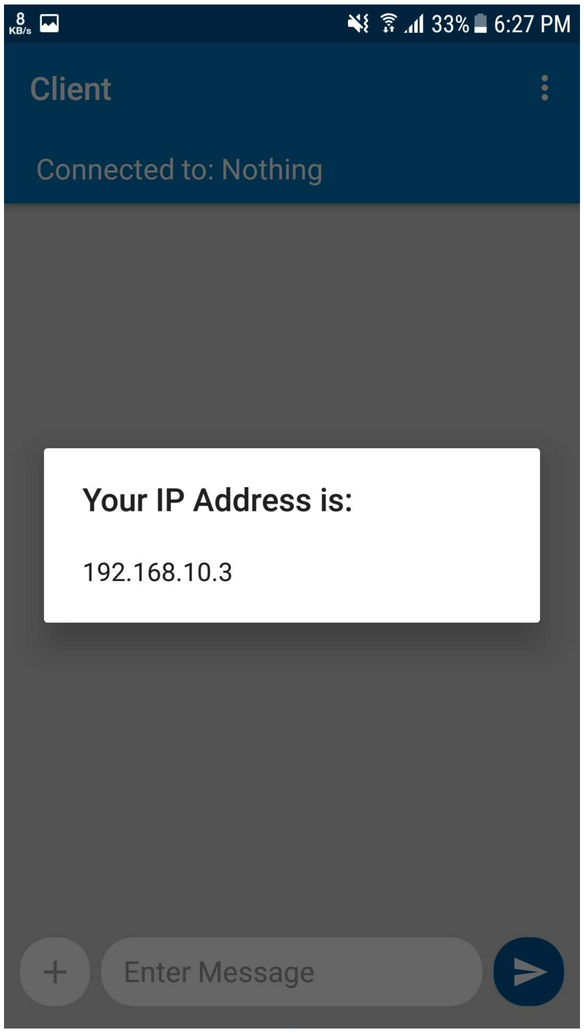
Connecting the Client:





The left picture shows the start activity that user will see when opening the app. Pressing the "Start Private Messaging" button will open the client activity shown on the right. It will inform the user at the top about the connectivity status to the server, which on start will be "Connected to: Nothing". The user can then open the menu which will allow him to check his IP address and Configure the Server.



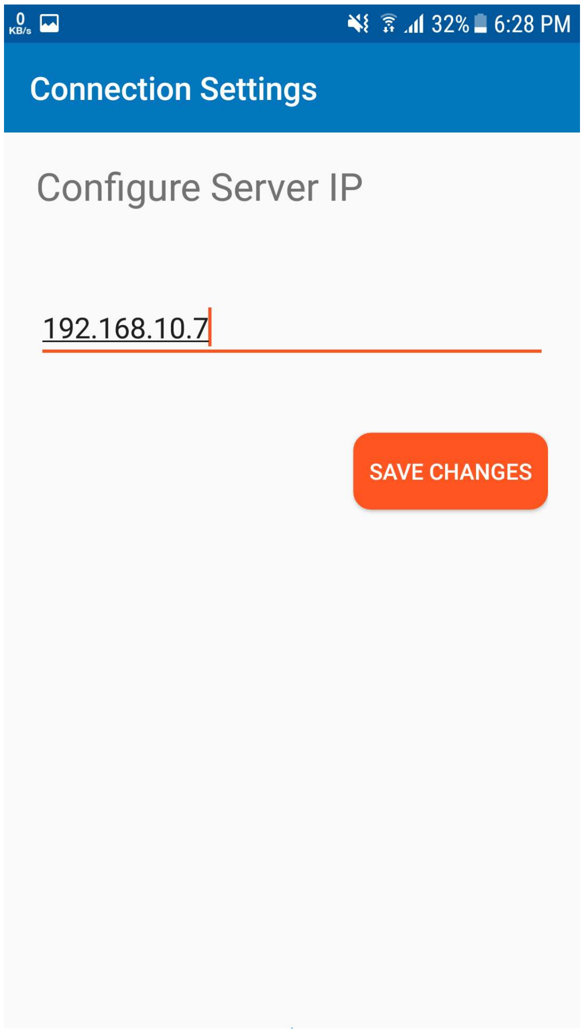


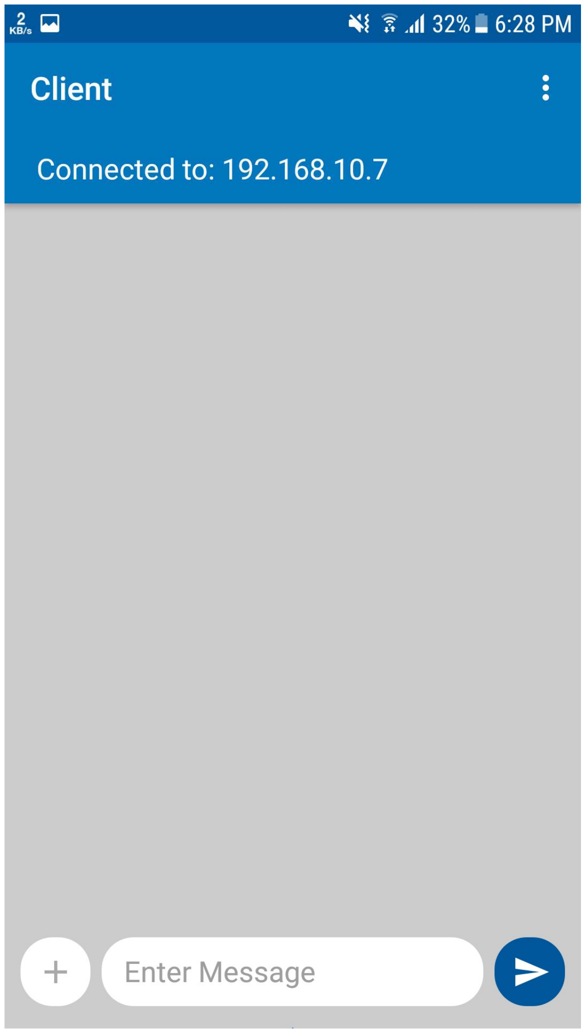
Your IP Address is:

192.168.10.3

The menu that user can access is shown in the picture on the left. On selecting "My IP", the IP Address of client will be displayed like in the picture on the right.

Now lets see how to set the server IP and connect to it.





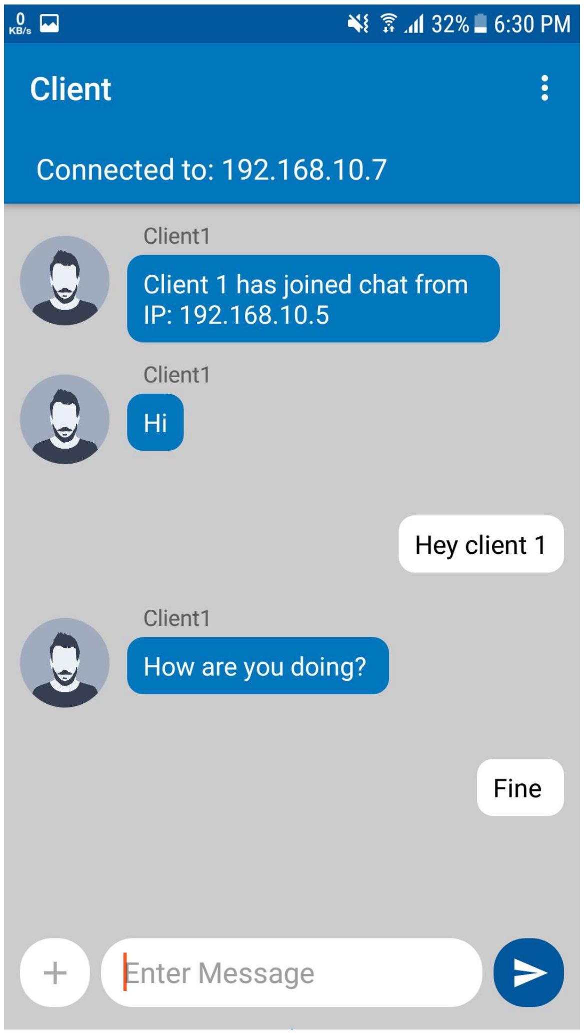
By selecting "Set Server IP" from the menu, the Set Server Activity will start, shown in the picture on the left. Here the client will have to enter the IP of the machine on which server is running. Using commands like "ipconfig" for Windows or "ifconfig" for Linux, that can be found out if the server is on same local network. If however, the server is located on another network, then its public IP will have to be entered here.

After pressing "Save Changes" button, client will return back to chat screen, this time showing on top the connection status and server IP.

Similarly, as many clients can connect to the server as the size of arrays in it. After each connection, the Server will notify on its side:

```
Client 0 connected to chat from IP: 192.168.10.3
Client 1 connected to chat from IP: 192.168.10.5
```

And notify previously connected clients of new members.



Connected to: 192.168.10.7

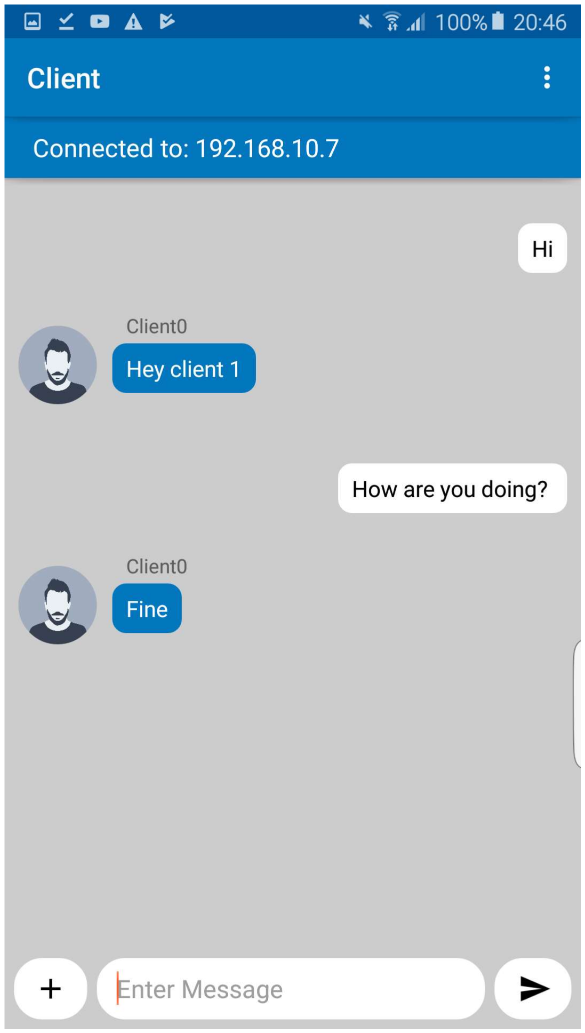
Client1
Client 1 has joined chat from IP: 192.168.10.5

Client1
Hi

Hey client 1

Client1
How are you doing?

Fine



Connected to: 192.168.10.7

Hi

Client0
Hey client 1

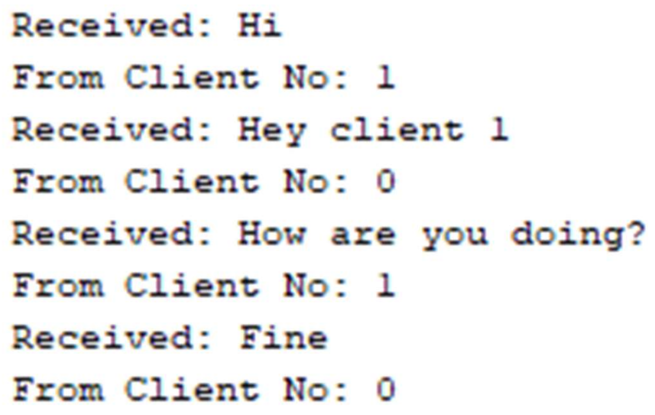
How are you doing?

Client0
Fine

The client 0 on left joined first and got a an automatic message from server to notify about the connection of client 1 later.

## Group Messaging:

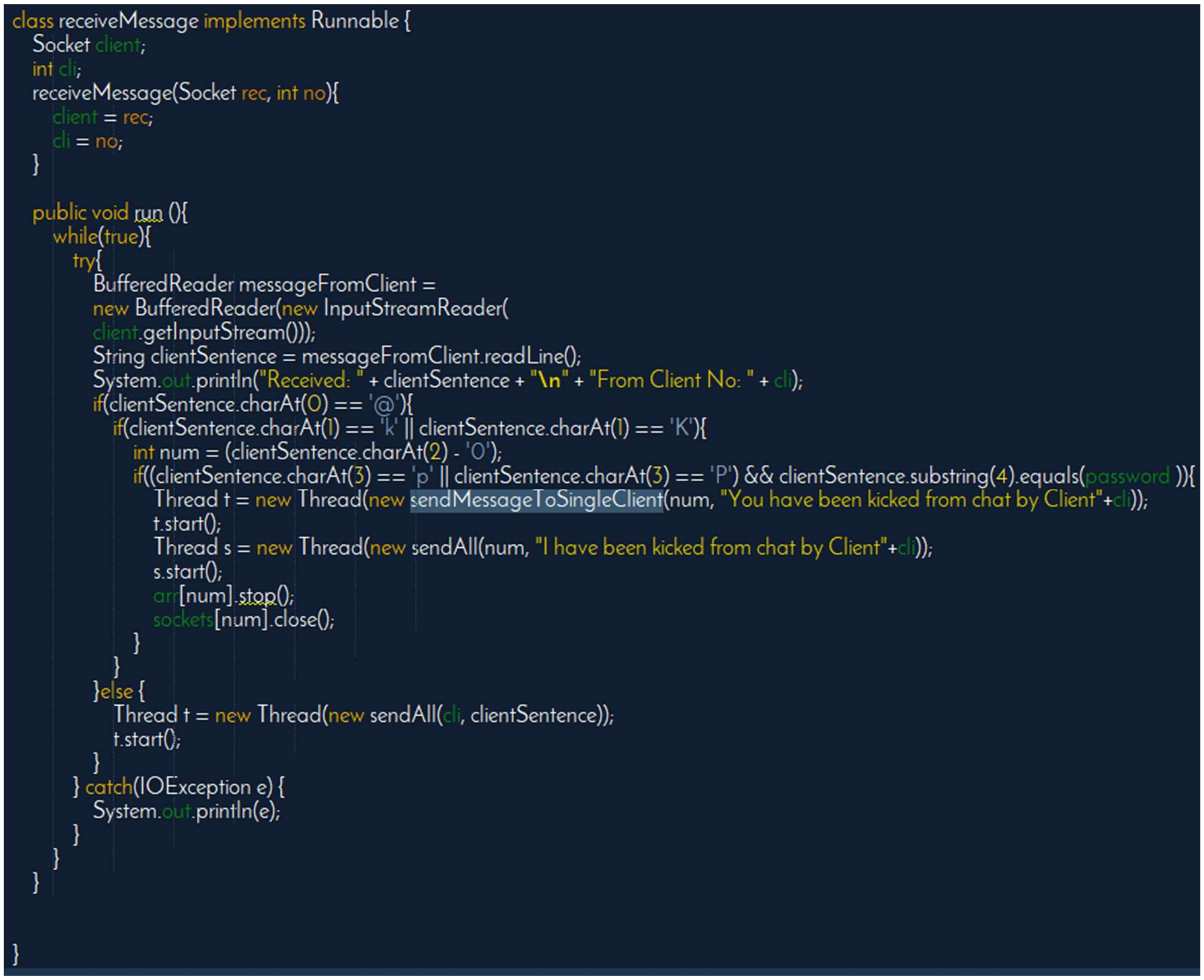
As seen in the last two images, the clients can send messages in the group and they will be delivered to all members of the group. The server also keeps log of the messages:



```
Received: Hi
From Client No: 1
Received: Hey client 1
From Client No: 0
Received: How are you doing?
From Client No: 1
Received: Fine
From Client No: 0
```

The messages are received in server in the individual threads running receiveMessage Method for each client when clients connect as we saw before in ‘Starting the Server’ section. The receiveMessage Method is shown in the picture on the next page.

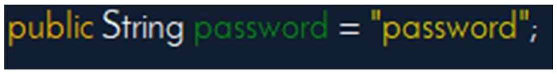
The BufferedReader is used to read the message. The readLine() method is the blocking call that will wait till the client sends a message. After receiving, it will be logged and check if it contains a command by checking if first character is ‘@’, which we will discuss later. If however, it is not a command, it will run the sendAll method in a new thread to send this message to all the other clients. The sendAll method has been shown before in ‘Starting the Server’ section.



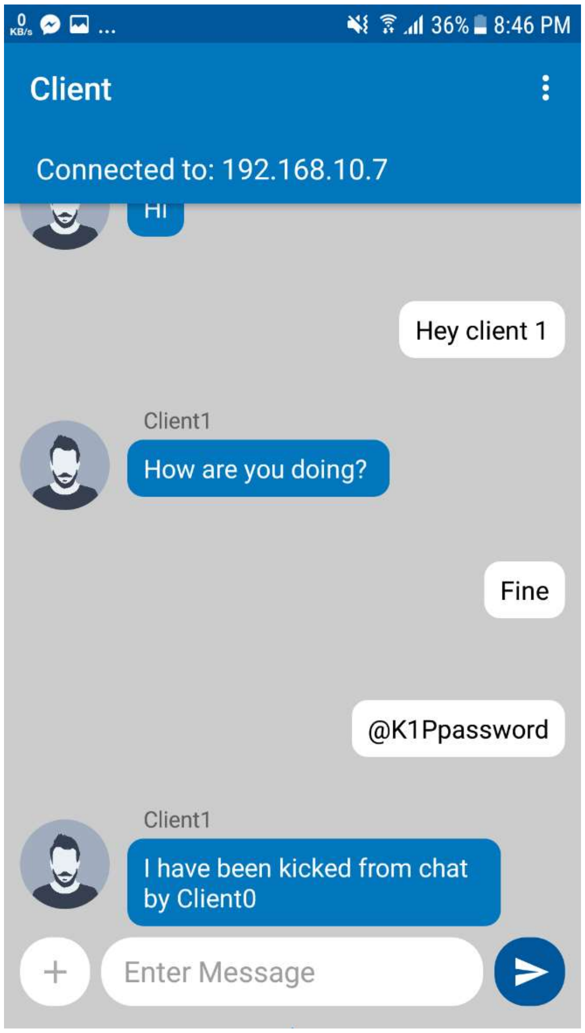
```
class receiveMessage implements Runnable {
   Socket client;
   int cli;
   receiveMessage(Socket rec, int no){
      client = rec;
      cli = no;
   }

   public void run (){
      while(true){
         try{
            BufferedReader messageFromClient =
            new BufferedReader(new InputStreamReader(
            client.getInputStream()));
            String clientSentence = messageFromClient.readLine();
            System.out.println("Received: " + clientSentence + "\n" + "From Client No: " + cli);
            if(clientSentence.charAt(0) == '@'){
               if(clientSentence.charAt(1) == 'k' || clientSentence.charAt(1) == 'K'){
                  int num = (clientSentence.charAt(2) - '0');
                  if((clientSentence.charAt(3) == 'p' || clientSentence.charAt(3) == 'P') && clientSentence.substring(4).equals(password )){
                     Thread t = new Thread(new sendMessageToSingleClient(num, "You have been kicked from chat by Client"+cli));
                     t.start();
                     Thread s = new Thread(new sendAll(num, "I have been kicked from chat by Client"+cli));
                     s.start();
                     arr[num].stop();
                     sockets[num].close();
                  }
               }
            }else {
               Thread t = new Thread(new sendAll(cli, clientSentence));
               t.start();
            }
         } catch(IOException e) {
            System.out.println(e);
         }
      }
   }


}
```

## Removing Members from Chat:

As discussed in Overview, the server maintains a password that can be set before starting the server.



```
public String password = "password";
```



Connected to: 192.168.10.7

Hi

Hey client 1

Client1

How are you doing?

Fine

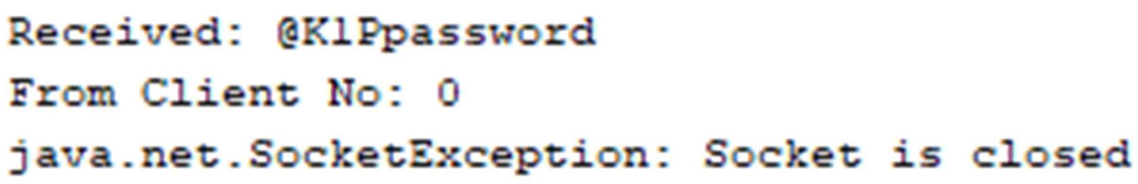
@K1Ppassword

Client1

I have been kicked from chat by Client0

The Client 0 will send a command to server as shown on left like "@K1Ppassword". From the code of receiveMessage method on previous page we can see that it will not show up in other clients' chat. Instead it will be checked using multiple conditional statements and if the syntax is correct and password matches, the server will tell Client 1 that it has been removed from chat by Client 0 and tell the rest of the Group same as well.

The last message on the left shows an automatic message generated by server to tell the rest of the group that Client 0 has removed Client 1 from the group chat. The server will then close the socket for Client 1 and it will be removed from chat. No more messages will be exchanged from it. The server will log the disconnection as below:

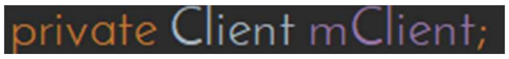


```
Received: @K1Ppassword
From Client No: 0
java.net.SocketException: Socket is closed
```

Now we can take a brief look at how the Android code of client is working.

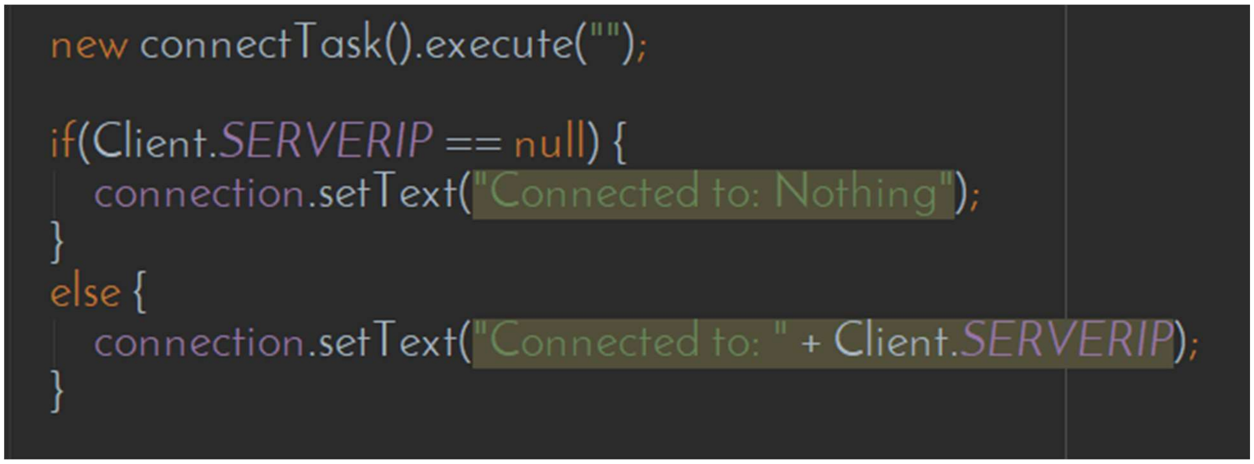
## Working of Android Client:

The Client Activity maintains an instance of Client class that will be shown later.



```java
private Client mClient;
```

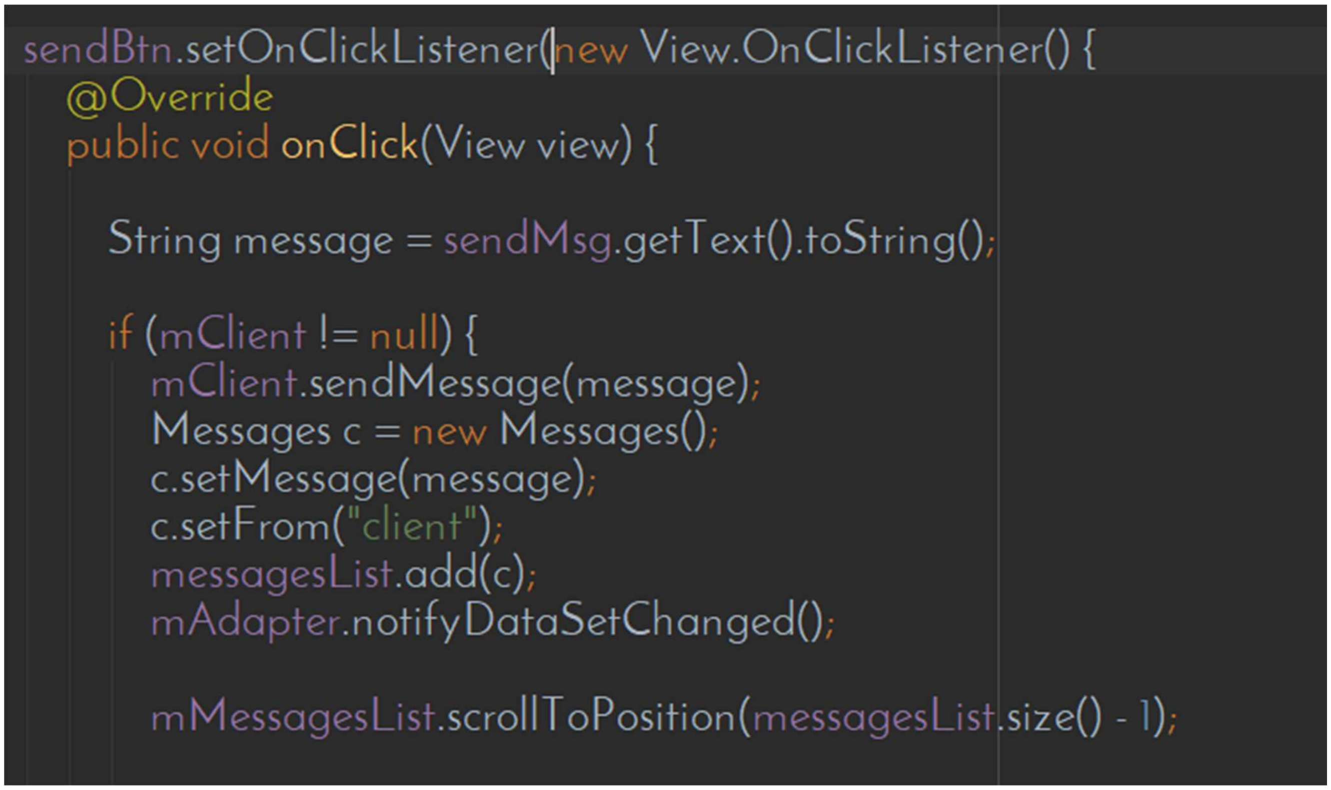
The conditional statements check the "SERVERIP" string of the Client instance to update the connection status shown to the user.



```java
new connectTask().execute("");

if(Client.SERVERIP == null) {
    connection.setText("Connected to: Nothing");
}
else {
    connection.setText("Connected to: " + Client.SERVERIP);
}
```

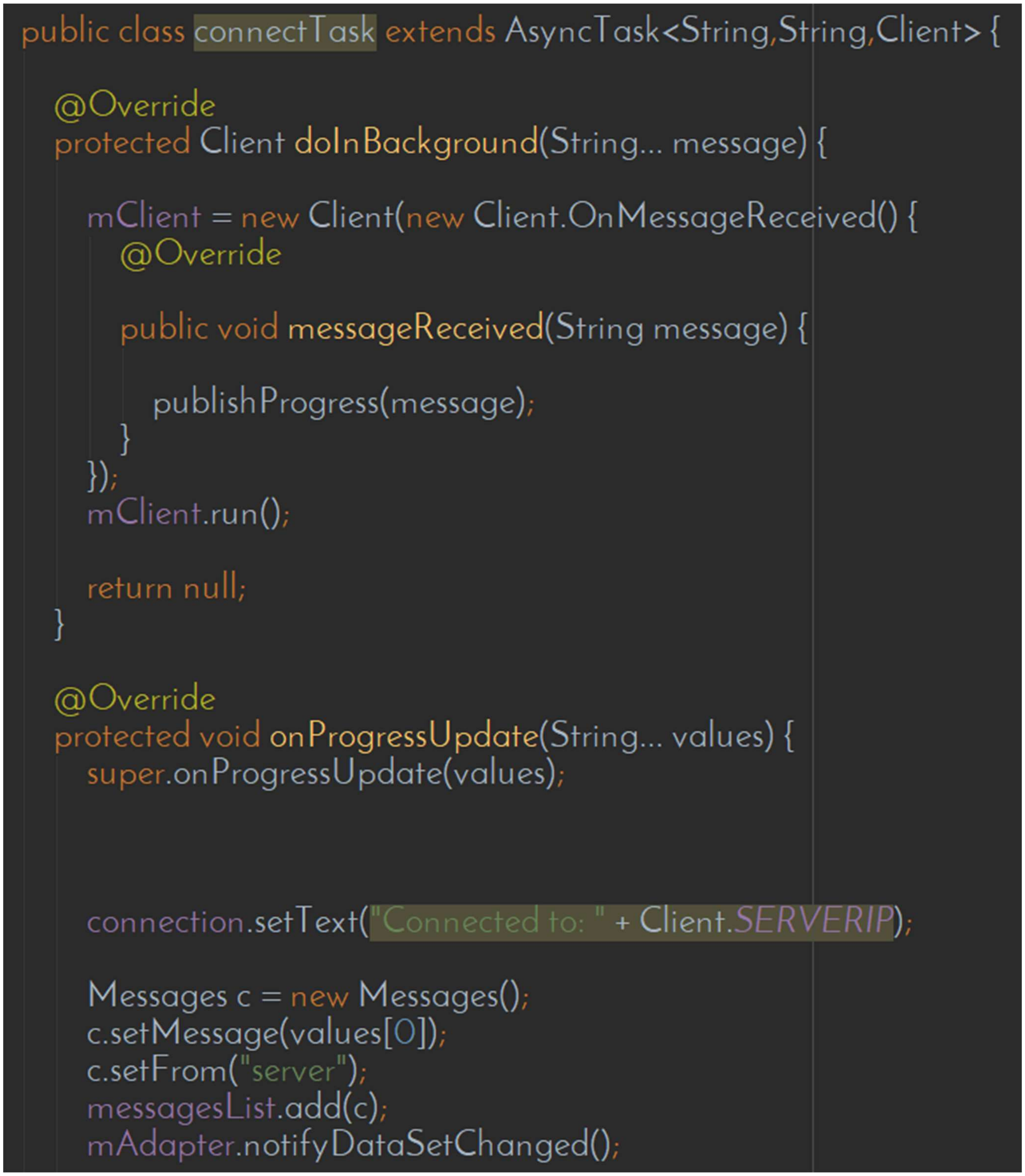
An instance of a connectTask is made that is a derived class of Android library AsyncTask and will run the Client code in a thread.

The message sending is done by pressing the send button, so a listener is checking for when that button is pressed as shown on next page. It takes the String message and passes it to the sendMsg method of Client instance. It then updates the UI to show the message in list.



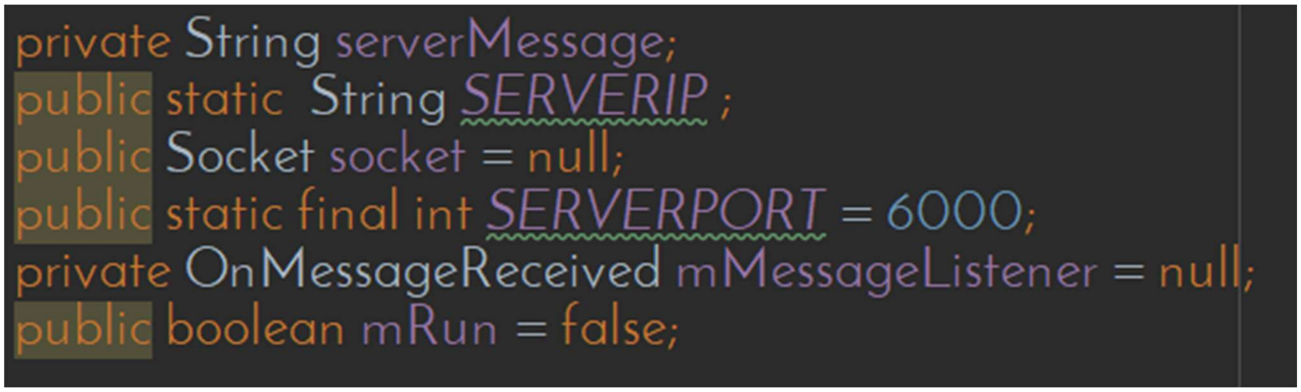
```
sendBtn.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View view) {

    String message = sendMsg.getText().toString();

    if (mClient != null) {
      mClient.sendMessage(message);
      Messages c = new Messages();
      c.setMessage(message);
      c.setFrom("client");
      messagesList.add(c);
      mAdapter.notifyDataSetChanged();

      mMessagesList.scrollToPosition(messagesList.size() - 1);
```

The connectTask is shown on the next page. It derives the AsyncTask and runs the client in background thread and when a message is received, the onProgressUpdate method will be called automatically to update the UI and show the message in the list.



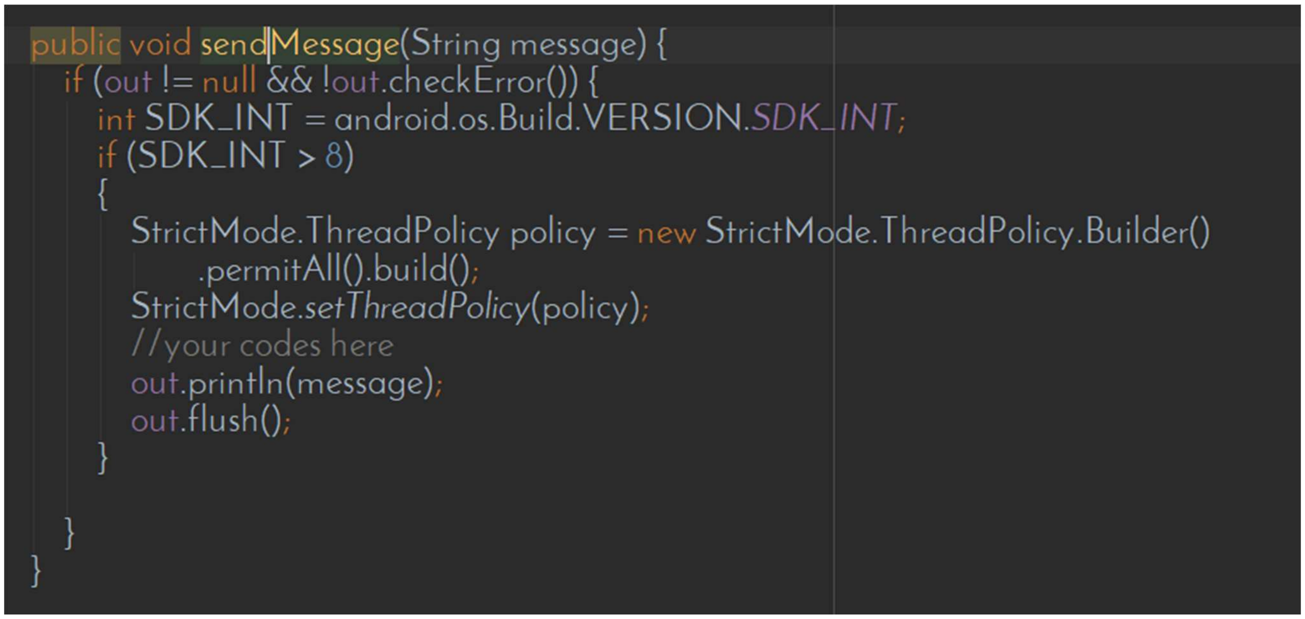
```
public class connectTask extends AsyncTask<String,String,Client> {

    @Override
    protected Client doInBackground(String... message) {

        mClient = new Client(new Client.OnMessageReceived() {
            @Override

            public void messageReceived(String message) {

                publishProgress(message);
            }
        });
        mClient.run();

        return null;
    }

    @Override
    protected void onProgressUpdate(String... values) {
        super.onProgressUpdate(values);



        connection.setText("Connected to: " + Client.SERVERIP);

        Messages c = new Messages();
        c.setMessage(values[0]);
        c.setFrom("server");
        messagesList.add(c);
        mAdapter.notifyDataSetChanged();
```

The Client maintains the following variables including the “SERVERIP” string which will store the IP of server when the user enters it in Set Server Activity. The default port is also stored as 6000 at which connection will be implemented.



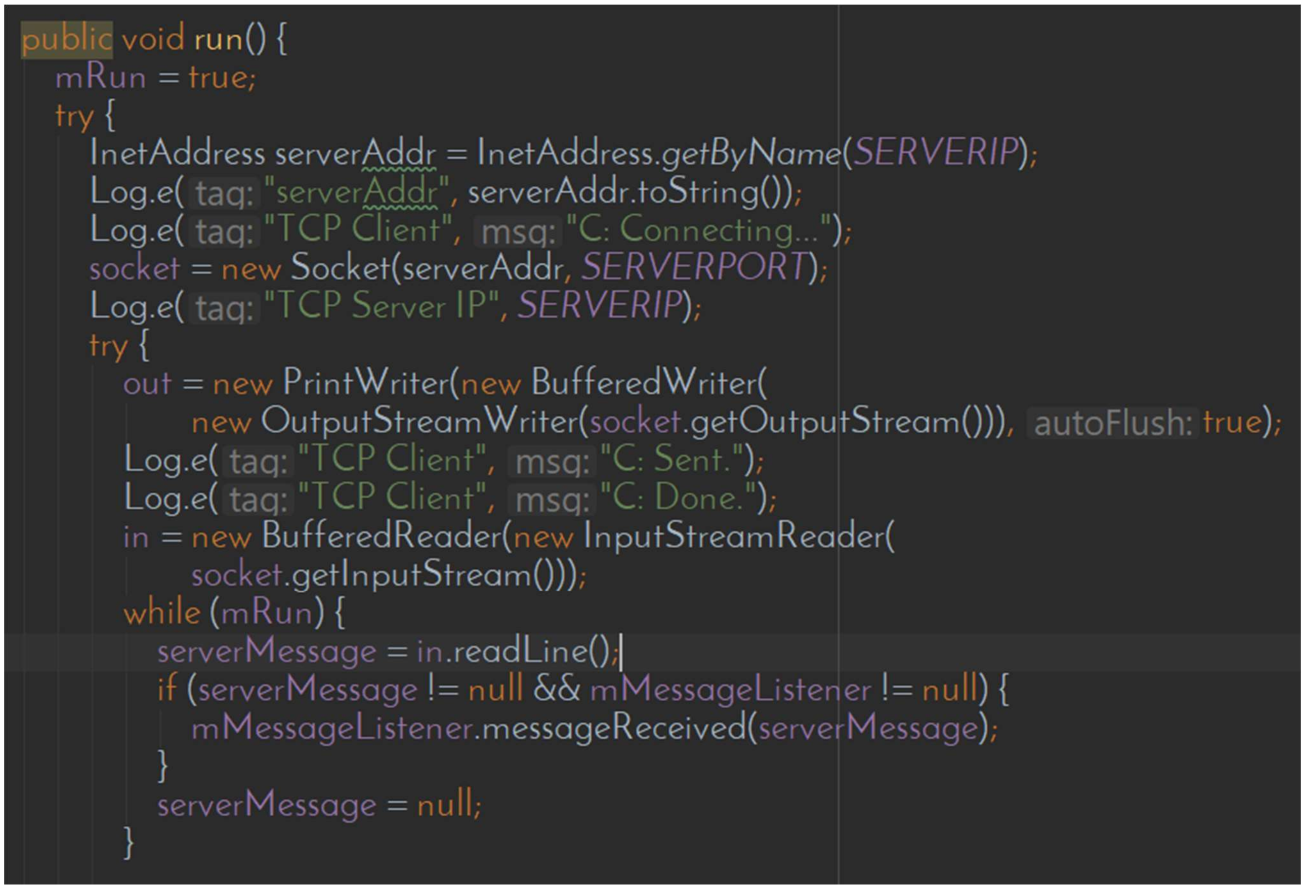
```java
private String serverMessage;
public static  String SERVERIP ;
public Socket socket = null;
public static final int SERVERPORT = 6000;
private OnMessageReceived mMessageListener = null;
public boolean mRun = false;
```

The sendMessage method of client is shown below which will be called from client Activity as we saw before and the message will be sent to server using a PrintWriter instance named “out”.



```java
public void sendMessage(String message) {
    if (out != null && !out.checkError()) {
        int SDK_INT = android.os.Build.VERSION.SDK_INT;
        if (SDK_INT > 8)
        {
            StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
                    .permitAll().build();
            StrictMode.setThreadPolicy(policy);
            //your codes here
            out.println(message);
            out.flush();
        }

    }
}
```

Shown below is the run() method of client that is the executed first when it runs in a thread in connectTask. Here, a TCP Socket is made and connection is attempted to the server using the IP Address specified by the user and the default port 6000. The BufferedReader instance "in" and PrintWriter instance "out" are created for receiving and sending of messages respectively.



```
public void run() {
    mRun = true;
    try {
        InetAddress serverAddr = InetAddress.getByName(SERVERIP);
        Log.e( tag: "serverAddr", serverAddr.toString());
        Log.e( tag: "TCP Client",  msg: "C: Connecting...");
        socket = new Socket(serverAddr, SERVERPORT);
        Log.e( tag: "TCP Server IP", SERVERIP);
        try {
            out = new PrintWriter(new BufferedWriter(
                    new OutputStreamWriter(socket.getOutputStream())),  autoFlush: true);
            Log.e( tag: "TCP Client",  msg: "C: Sent.");
            Log.e( tag: "TCP Client",  msg: "C: Done.");
            in = new BufferedReader(new InputStreamReader(
                    socket.getInputStream()));
            while (mRun) {
                serverMessage = in.readLine();
                if (serverMessage != null && mMessageListener != null) {
                    mMessageListener.messageReceived(serverMessage);
                }
                serverMessage = null;
            }
```

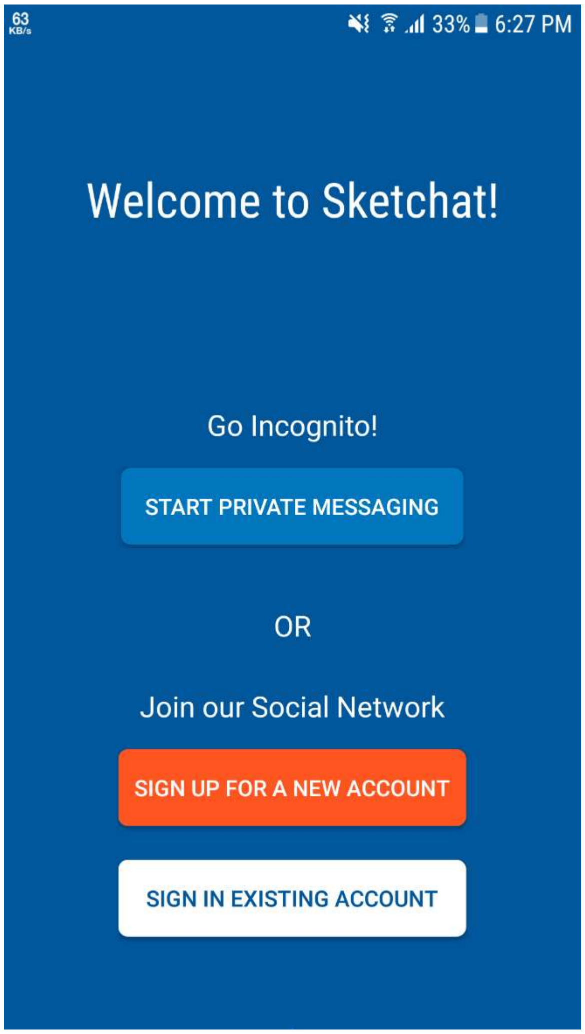
This covers all the primary aims of the project we have completed. Now let's look at the Extras included in the project.

## Extras:

Apart from the required objectives, we have also implemented the following extra features.

### Offline Messaging:

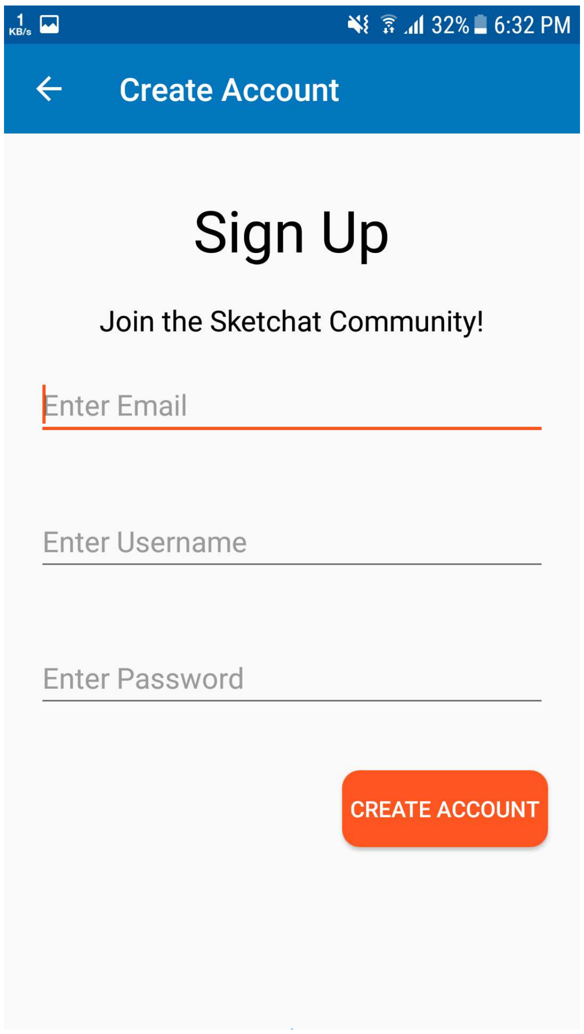
We only need an active internet connection if the server is running on a machine outside the local network. However, in a lot of cases, it might be used locally. In this scenario, no internet will be required and the group chat can work without connection to outside networks.

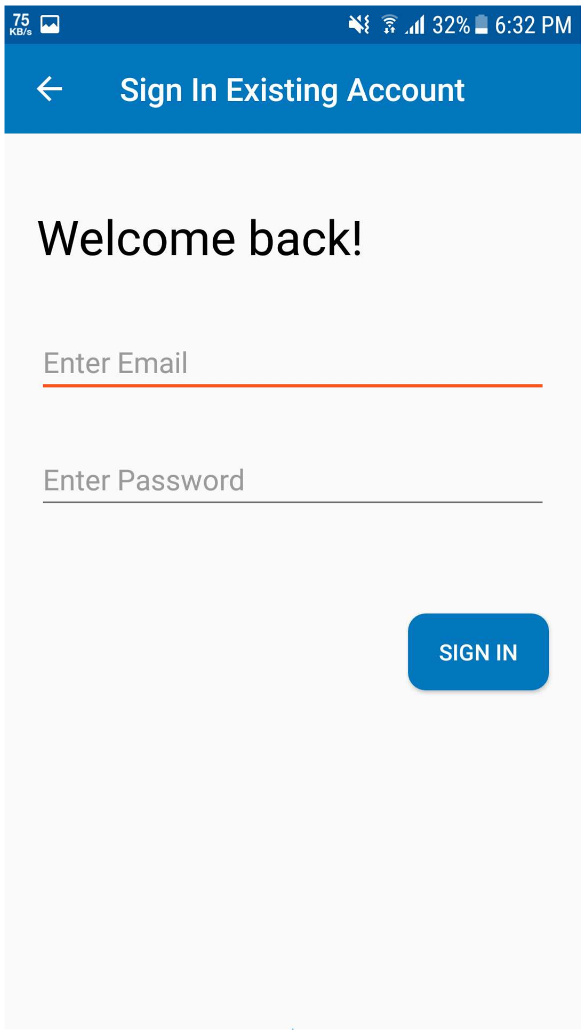


### Social Network:

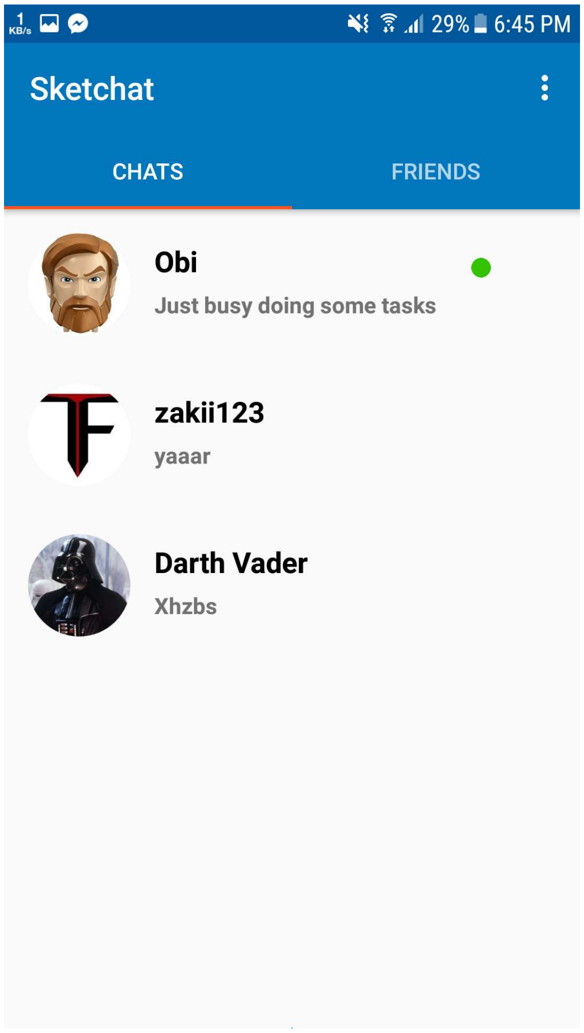
In the Start Activity, apart from the “Start Private Messaging” button there are also options to join our Social Network by Signing up for a new Account or Signing into an already created account.

To achieve this, we are using Firebase that is a cloud real-time database and server. Using it allows us to create a bigger environment for the user to interact with users globally. All chatting activity is stored in the Firebase database and is retrieved by the recipient.





Upon pressing the "Sign up for a new account" button, user will enter in the Sign Up activity as shown in the left picture above and similarly, pressing the "Sign in existing account" button will enter the user in Sign In activity as shown in the right picture above. User can go back to Start Activity, or upon successfully Sign Up/ Sign In, he will enter Main Activity that houses all the information about the Social Network.

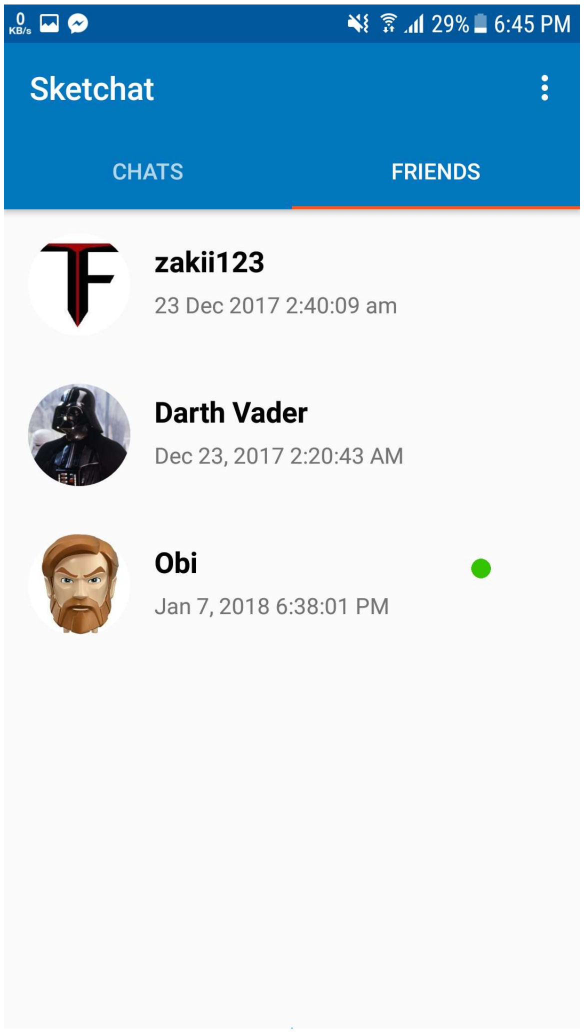

Sketchat

CHATS FRIENDS

Obi ●
Just busy doing some tasks

zakii123
yaaar

Darth Vader
Xhzbs


Sketchat

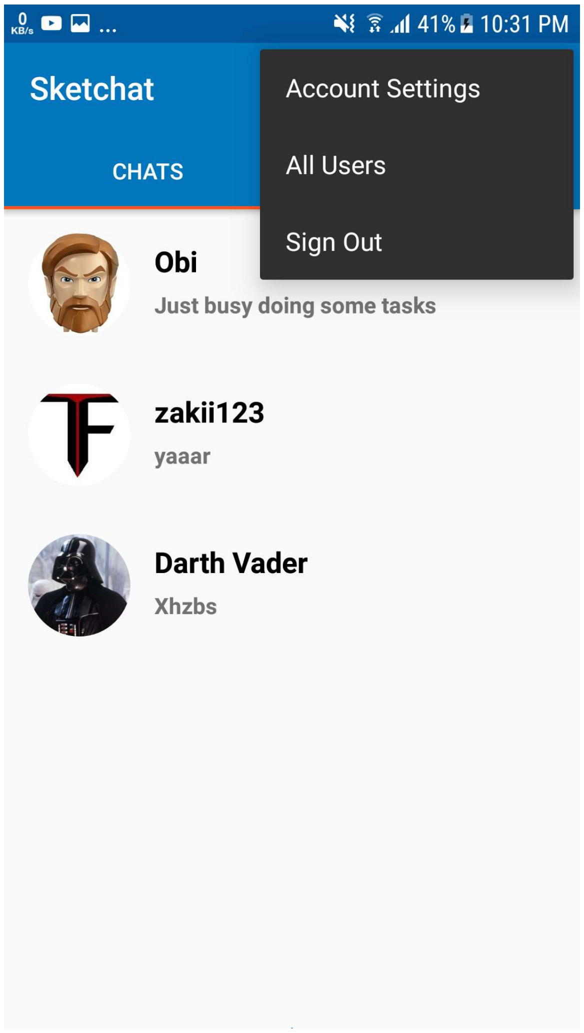
CHATS FRIENDS

zakii123
23 Dec 2017 2:40:09 am

Darth Vader
Dec 23, 2017 2:20:43 AM

Obi ●
Jan 7, 2018 6:38:01 PM

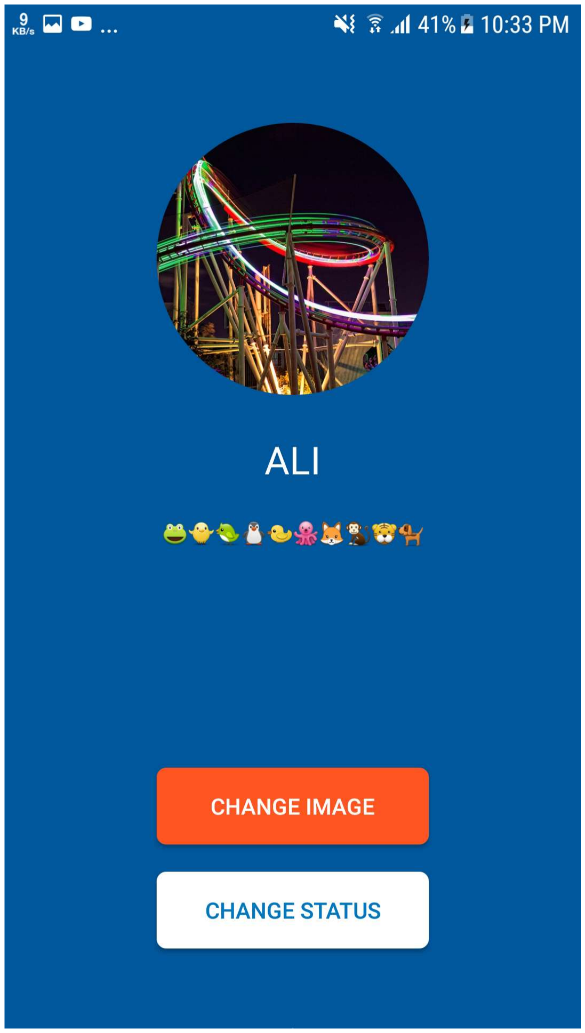
The main activity has two tabs, "Chats" and "Friends" as shown in pictures above left and right respectively. Both tabs will show the users that are online by a green circle. The Chat tab puts the most recent chats to the top while friends tab puts most recent accepted friends to the bottom.



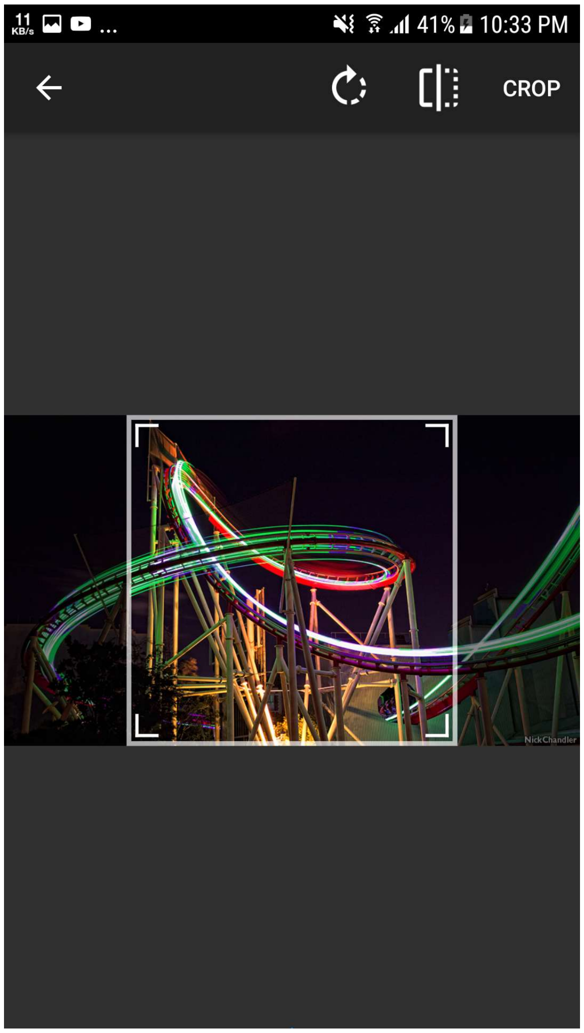
Obi
Just busy doing some tasks

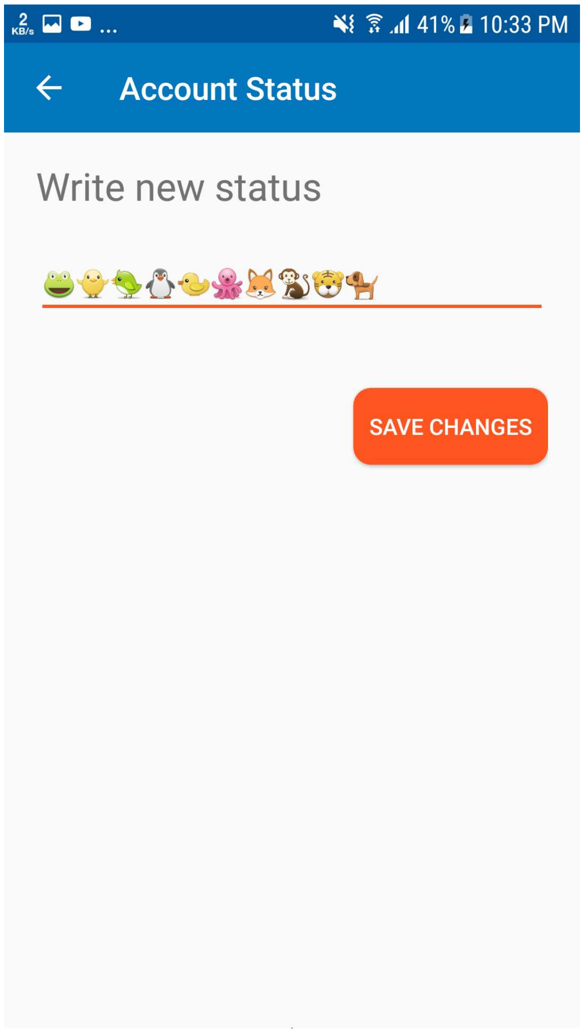
zakii123
yaaar

Darth Vader
Xhzbs

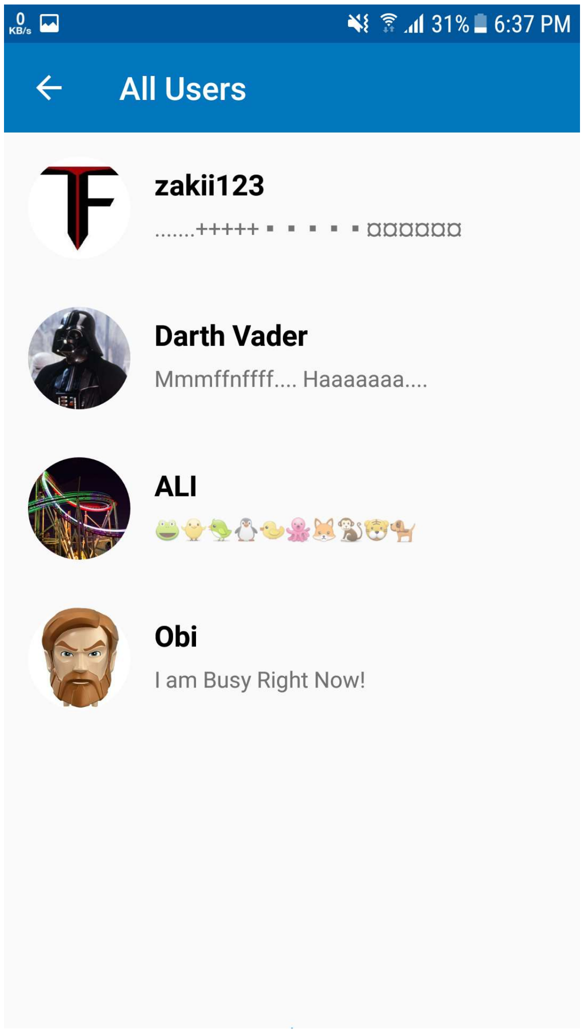


Here as well, the user can access a menu as shown in the picture on the left and go to account settings shown on right. He can also view all users in the Social Network by going into the All users activity or go back to start activity by pressing “Sign Out”.





In the Account Settings, the user can change his profile image and select a new one as shown in the picture above on the left. Similarly he can change his account status as shown in the picture above on the right. All these changes will be stored in the database other parts of the app can retrieve them from the database to populate the UI.

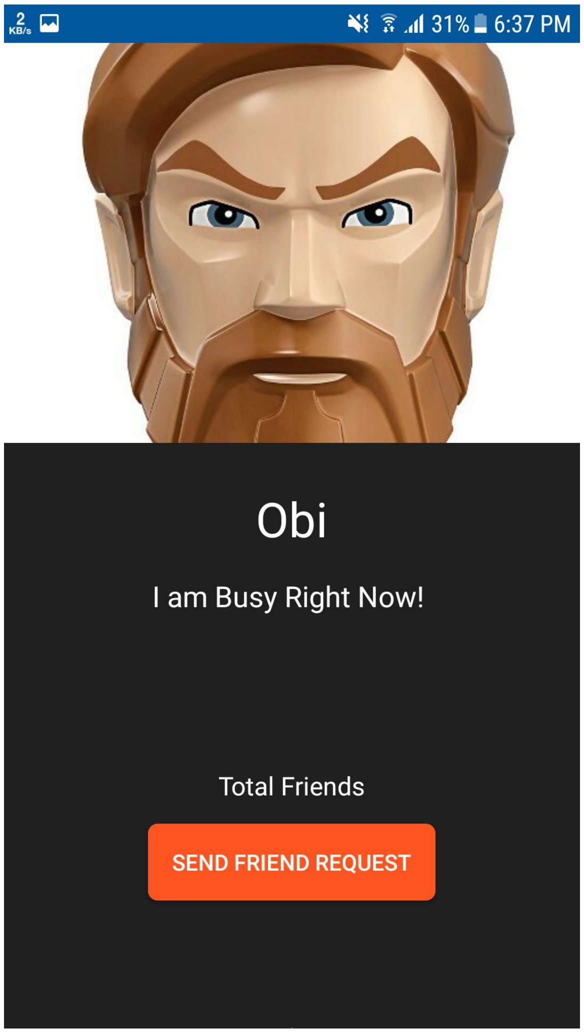


zakii123
.......+++++ ▪ ▪ ▪ ▪ ▪ ¤¤¤¤¤¤

Darth Vader
Mmmffnffff.... Haaaaaaa....

ALI
🐸🐥🦎🐧🐤🐙🦊🐒🐯🐕

Obi
I am Busy Right Now!

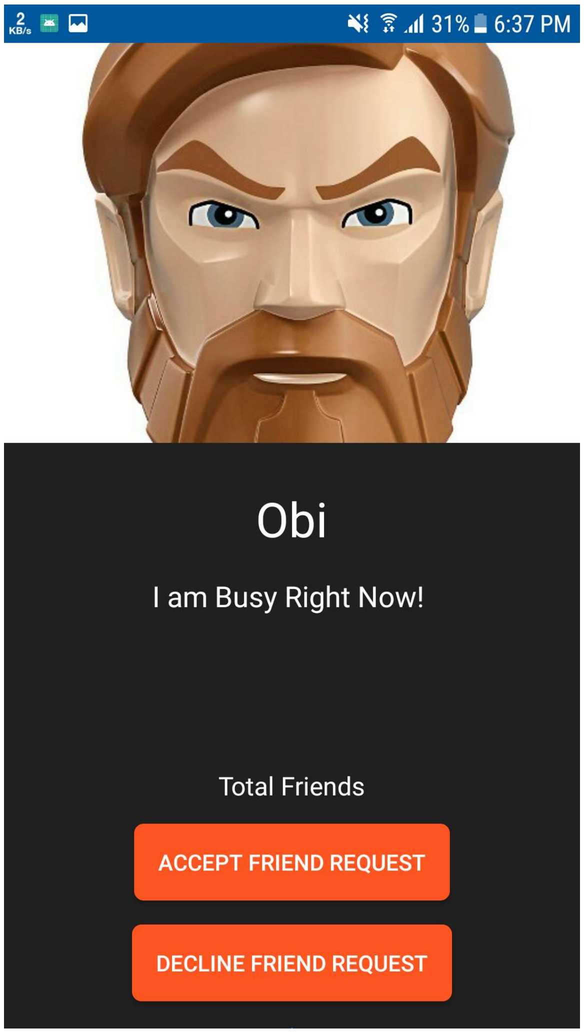


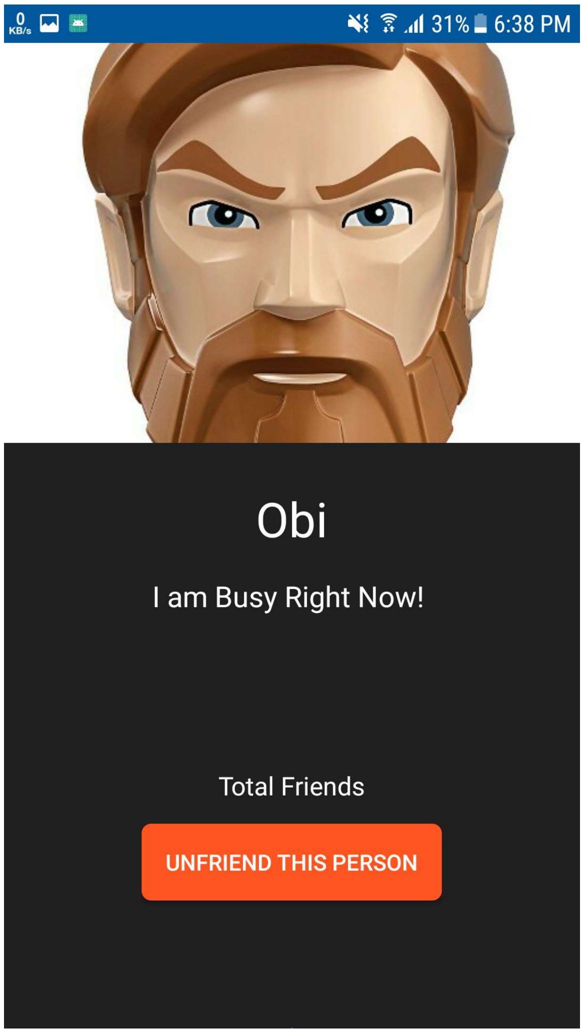
Obi

I am Busy Right Now!

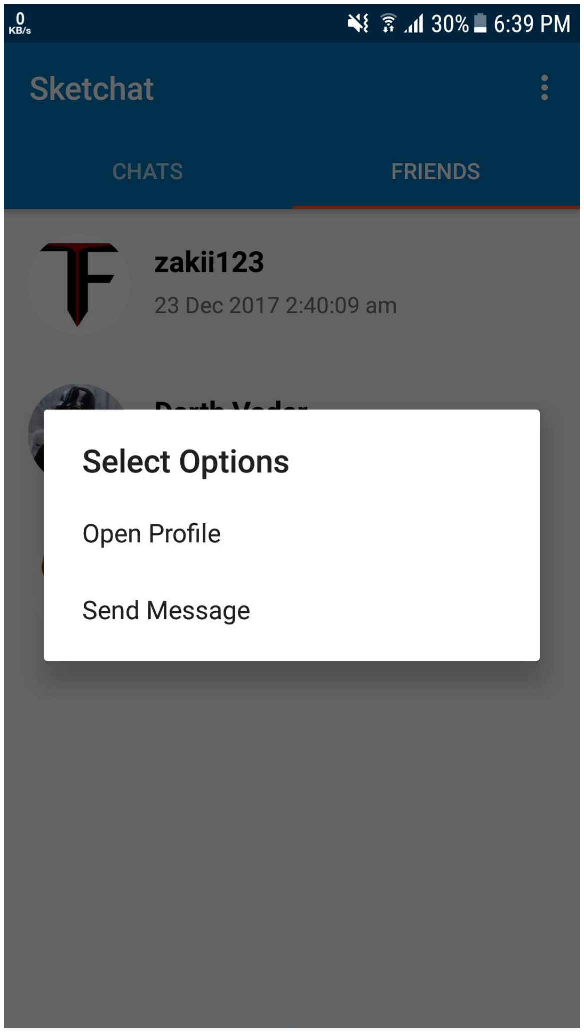
Total Friends

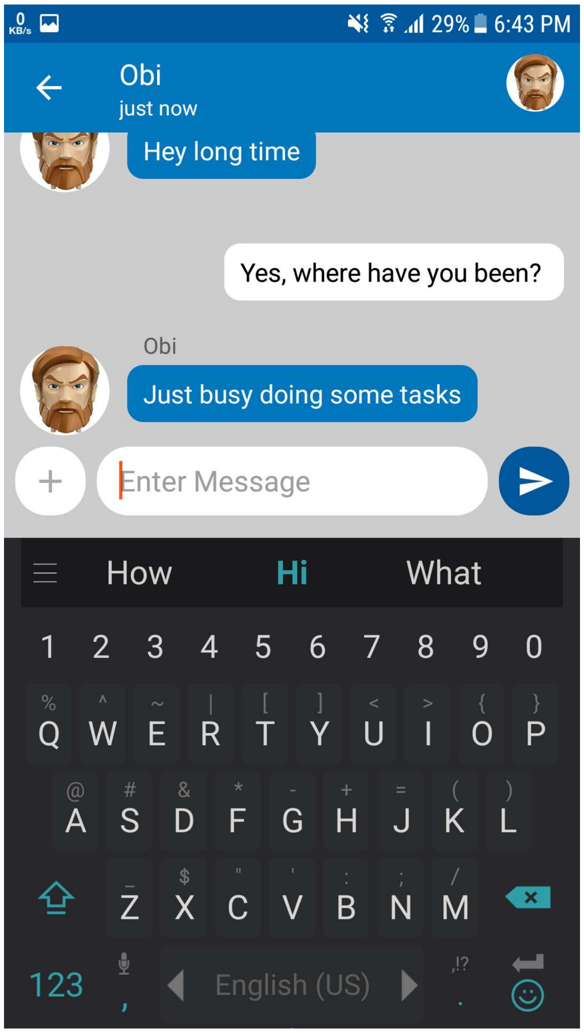
The all users activity shows all the users in the social network, stored in the firebase database, shown in the picture above on the left. The image on the right shows how a profile will look when a person is not made friend. There will be a button to send friend request.





However, if the other person sends us friend request, we can have options to accept it or decline it, as shown on the left image above. When accepted as friend, there will be an option to unfriend the person as shown in the image on the right.





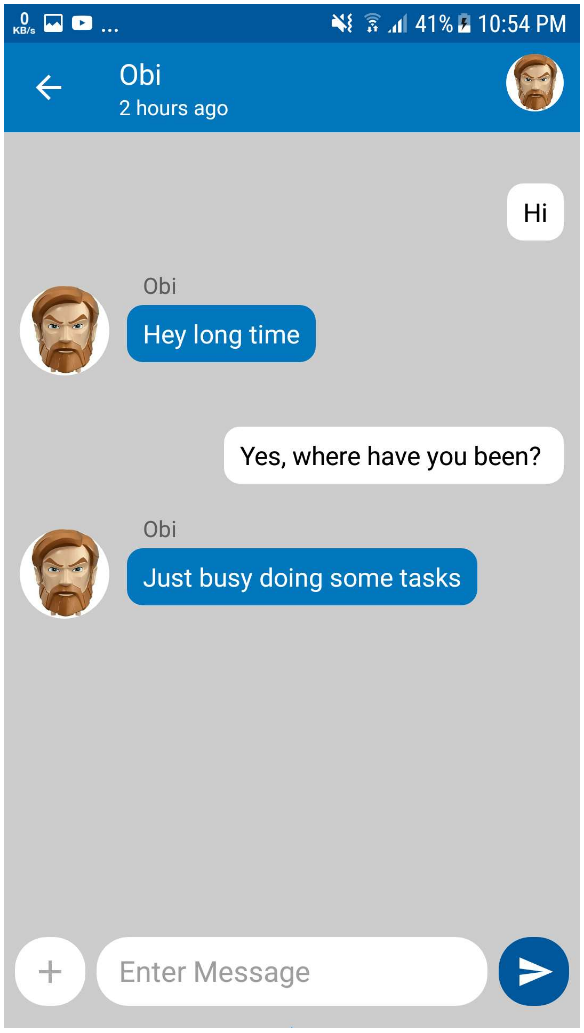
Hey long time

Yes, where have you been?

Obi

Just busy doing some tasks

After accepting friend request, the person will appear in the friends tab in the main activity as shown in the left image above and upon pressing his entry, there will be options to open profile or send message. Opening profile will take back to the activity on the previous page. Send Message will open the chat where messages can be sent. On the top, details about the last time the user was online will be shown.



Hi

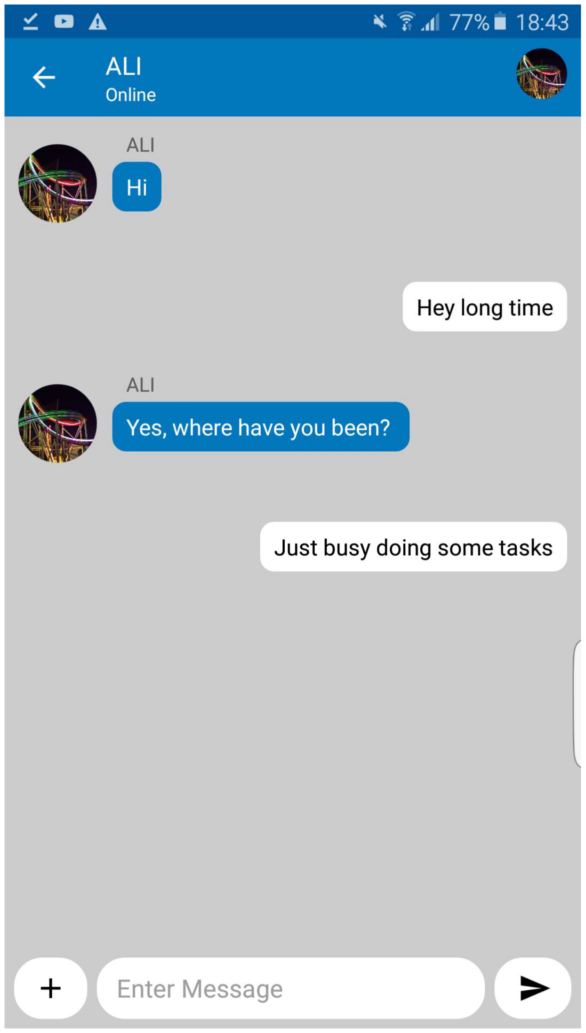
Obi

Hey long time

Yes, where have you been?

Obi

Just busy doing some tasks



ALI

Hi

Hey long time

ALI

Yes, where have you been?

Just busy doing some tasks

This covers everything we have implemented in this project.

## Software Tools Used:

- Android Studio (Client Development along with UI)
- Netbeans (Server Development in Java)