Mobile Systems Task 4 Report
Messaging with Android Smartphones

Wenchang Liu
ID: 10406141
2019/03/26

Content

1. Overview

This lab is to develop an Android app that allow multiple users to communicate with each other through a common server. I learned how to use sockets to connect server and communicate with each other with android mobile devices, how to design UI layouts and how to do simple android programming and debugging using java in this exercise.

2. Implementation

2.1 Introduction

The development has 3 stages, the first 2 stages are the transitive stages, so we are not going to do the thorough tests (I will just describe what tests I have done by texts) on those stages, the following contents including the idea of how I implemented the design and functions, but to see the actual codes, please see the code and documentation part.

Development environment:
android studio 3.3.2.0 + ubuntu 18.04 + API 23(emulator) / API 26(real phone)
(You will have to change SERVERIP to local ip of the server if you want to use a real phone)

2.2 Stage 1

2.2.1 Development

This stage I just extend the provided template with:

1. Another textView to show online users' status.
   In order to achieve this:
   - Create another textView in activity_main.xml
   - Also, in NetworkConnectionAndReceiver class, I will check the message that received from the server, if the message starts with "WHO", I will display the message to the new textView, otherwise, the message will be displayed in the original textView.
2. A linear layout that allow the user to register his username into the server.
   In order to achieve this:
   - Create a linear layout which includes a textView showing "Username:", and an editText allowing users to input the username, and a register button.

- Also, in onCreate(), we will need to find our editText and button using findViewById(), then in button's setOnClickListener method, we get the string from editText using getText().toString(), then we can use transmitter to transmit the message to server.

3. Add refresh, kill and disconnect buttons and implement their functions.
   The implementation is almost the same as "register" button:
   - We just do not need to get text from editText, we just need to transmit "disconnect"("who" for refresh button) to the server.
   - For "kill" button, we also need to add "System.exit(0)".
4. In order to do more testing stuff, I implemented a linear layout that allow user to send command like what we can do in telnet. The implementation is the same as "register" function.
5. Make users to see who is online after register automatically.
   The idea is when we pressed the "register" button, we not only transmit "register" to the server, but also transmit "who" to the server. There were 2 problems:
   - One is that the UI thread ran after the main thread, so the 2 messages will be print together afterwards and the first message will be overlapped by the second message because we all use the same "message" memory.
   - The second is that we want "who" arrive at the server later than "register", but this is not necessarily the case in reality.
   - So for the first problem, we initiate a new string inside the loop, then the 2 transmissions will have separate memory to avoid overlapping; for the second problem, we can add Thread.sleep(200) after sending "register" to ensure that it can arrive first.

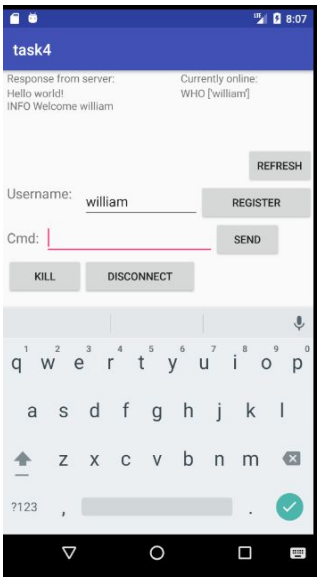

Figure 2-1 Stage 1 layout screen shot

2.2.2 Testing

The testing part of the initial version is simple, we just try the required functions:

1. Pressing “kill”, the app will disconnect the server and then terminate.
2. Pressing “disconnect”, the app will disconnect the server and the server will send “INFO Goodbye” to client.
3. Typing “william” in the “username” editText box, and press register button, the server will send “INFO Welcome william”, and Currently online textView will show “WHO [‘william’]”.
4. Let BarryBot be online, then press refresh button, then the Currently online textView will show “WHO [‘william’, ‘BarryBot’]”.

2.2.3 Evaluation

Good for an initial version: This is just a starting point of the app, we can see the responses and online users from the textViews, we can register ourselves without typing commands, however, other functions are not implemented as GUI, so this version is simple but inconvenient to use, but good enough for a initial version of the app.

2.3 Stage 2

2.3.1 Development

This stage is to extend the functions that allows us to set up connections and send messages to communicate with other users, new features include:

1. Invite other users to set up connections with you
2. Accept or decline other users’ invitation
3. Send one-line messages to a specific user

The implementation is simple, just as how we implement “register” function.

Figure 2-2 Part 4.4 layout screen shot

2.3.2 Testing

This is not the final stage, so we just simply test the new features:

1. Refresh, register, kill, disconnect functions are the same as part 4.3.
2. After registering, input “BarryBot” in the “invite others” editText box and press “invite” button, and we will see the server respond that “ACCEPT BarryBot”.
3. After setting up a connection with “BarryBot”, input “BarryBot” in the “Msg to” editText box and input an one-line message in the “Msg:” editText box and then press “send msg” button, then we will receive BarryBot’s response through the server.
4. Register another user using telnet and invite android client, then we will see the invitation through the server, then we can input the username of telnet client in the “invitations” editText and press “accept” or “decline” button, then we will see info messages from the server saying that we have set up connections with other users or we have declined the invitation.

2.3.3 Evaluation

Fully-functional but not user-friendly: For this version of app, we have introduced the GUI that allows users to set up connections and communicate with other users, however, the whole GUI layout looks like a form for user to fill and we have to type other users’ username in different editText boxes many times which can be a little bit annoying, it is fully-functional but not user-friendly enough.

2.4 Stage 3

2.4.1 Development

This is the final stage of the development, in this stage, we want to make the layout more simple and beautiful and easier to user, so we improve the GUI as follows:

1. Improve app appearances by changing background and text colors.
2. Inform the user if the app cannot connect the server.
3. Forbid user to change the register editText box after registered.
    - Implement using setCursorVisible(false), setFocusable(false), setFocusableInTouchMode(false) to lock up the editText after send “register” message to the server.
4. Parse the “WHO” response to show the actual users, not the whole string.
    - Implement using split(“:”) to split the response string and store them into String[].
5. Combine editText boxes of invite/accept/decline/msgTo into one “select user” dropdown list(will be updated with online users’ textView), in this way, users can simply select the

other user instead of inputting the whole username, which will be less laborious and can avoid typos.

- Implement drop down list using spinner, and change the contents after user registering or pressing “refresh” button.
- Add a get method in NetworkConnectionAndReceiver class, then we can use get method in main activity to access the updated online users’ list and update the spinner’s content.
- We can get the text in the spinner by using spinner.getSelectedItem.toString(), then anything else is the same as a editText box.

6. Add “end” function to end the connection with other users.
    - Implementation is similar to “invite” button.
7. Add auto scroll function to the “Response from server” textView to guarantee we can always see the latest response from the server. Also increase the ratio of this textView for better user experience.
    - Implement by calculating offset, which is the number of lines multiply the height of each line, then if the offset is larger than the height of the textView, we do scroll to the (offset-height) position using scrollTo() method.
8. Different kind of messages now have different colours to be easier distinguished.
    - Implemented using SpannableStringBuilder to set different colours to the strings we append to the textView according to the messages’ starting part. (e.g. “MSG”, “ERROR”, etc.)
9. Change all the layout to “match_parent” in order to adapt all kinds of android machines.
10. Also remove “Cmd” from the users because this is for development use only.

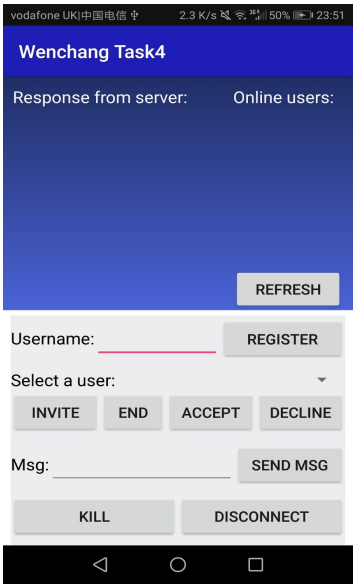

Figure 2-3 Part 4.5 layout

2.4.2 Testing

This is the final stage of the app, so I will make sure it works well:

1. Start the app without starting the server.
   We will see that there is a error message telling us “socket failed”.

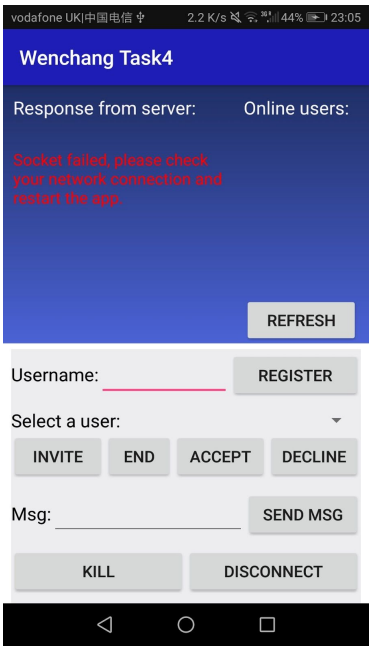

Figure 2-4 socket failed error handling

2. Then press “kill” button to kill the app.
   The app will terminate.
3. Connect to the server and try “refresh”, “register”, “invite”, “end”, “accept”, “decline”, “send msg” without register.
   We will see for refresh it need to register first, and to register without a name, it need provide a username, and the automatically get online users also failed, for invite/end/accpet/decline/send msg, because we make transmitText to be empty string when the dropdown list is empty, so it tells us it is not a command.

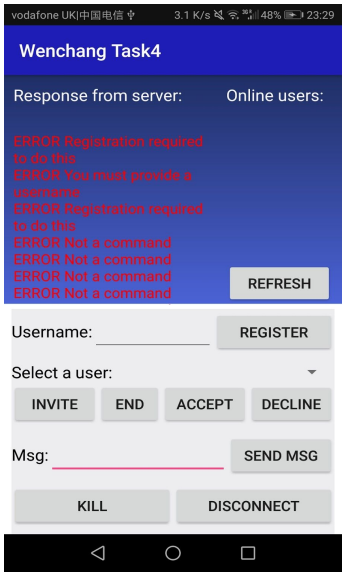

Figure 2-5 try different buttons without register

4. Try register by input username and press “register” button. After registering, try to edit the username and register again.
   We will see the INFO welcome message and see who is currently online easily, when we try to register again, we received an error message.

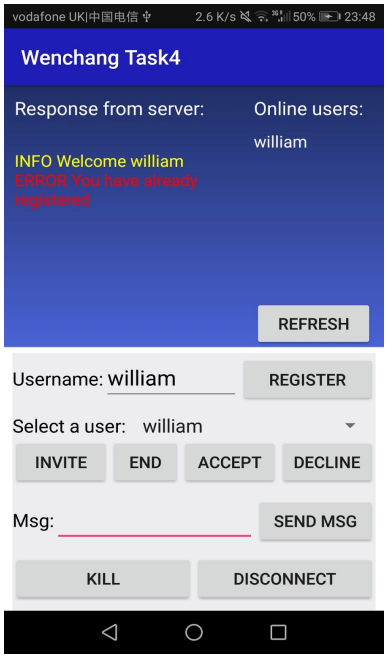

Figure 2-6 register and then try to register again

5. Let BarryBot to be online, then press “refresh” to check online users.
   We will see that BarryBot appears in the online user list, we also can select BarryBot from the dropdown list now.

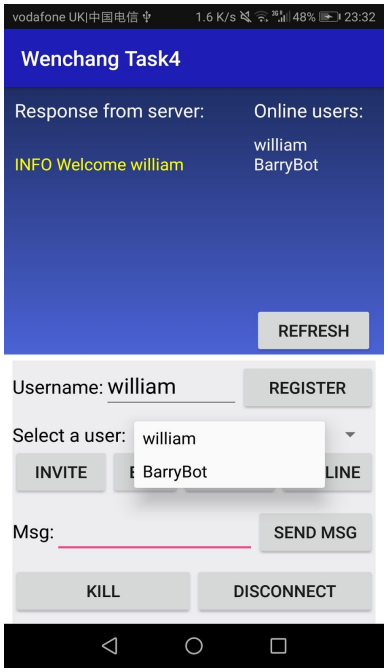

Figure 2-7 refresh after BarryBot is online

6. Select BarryBot in the dropdown list, then press “invite” to set up connection.
   We will see that BarryBot has accepted our invitation

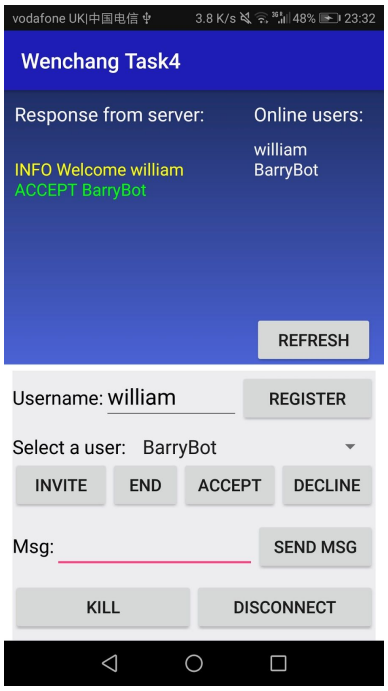

Figure 2-8 invite BarryBot

7. Press “end” to end connection with BarryBot, then try to send a message to BarryBot to check if the connection actually ended.
   We will receive no response for “end”, but when we are not allowed to chat with BarryBot after ending the connection with him.

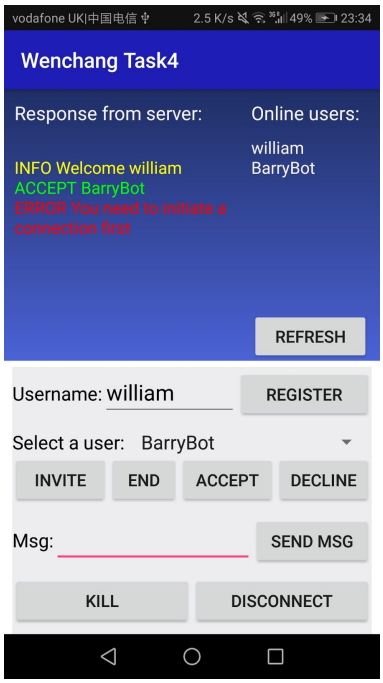

Figure 2-9 end connection with BarryBot

8. Invite BarryBot again, and then send a message to him and check the reply.
   We will see that BarryBot has accepted our invitation again, and after we send a message to BarryBot, we received his reply.

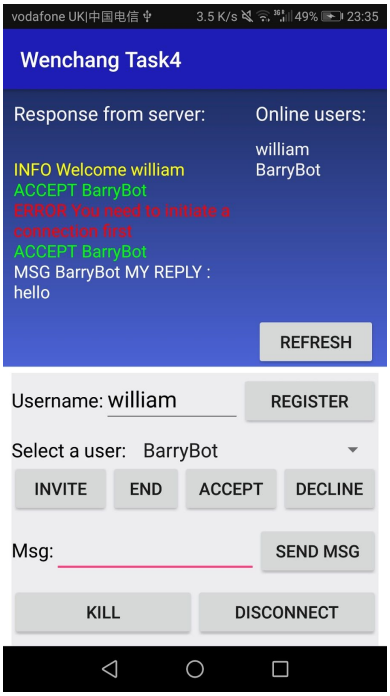

Figure 2-10 invite BarryBot again and send him a message

9. Try “accept” and “decline” without receiving a invitation.
   We will see that we are only allowed to accept or decline invitations.

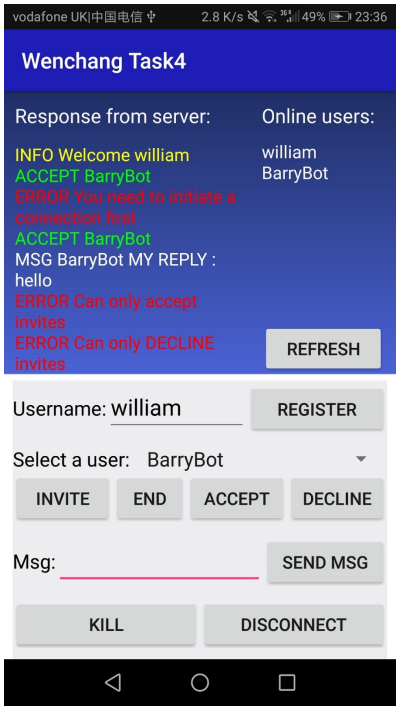

Figure 2-11 illegal accept or decline

10. Register another user using telnet, send a invitation to android device, then try press “decline” the invitation without selecting the right user, and then refresh the online list, select the right user and decline him.
    When we select BarryBot, we still cannot decline(or accept), when we refresh the online user list and select telnet, we can decline the invitation.

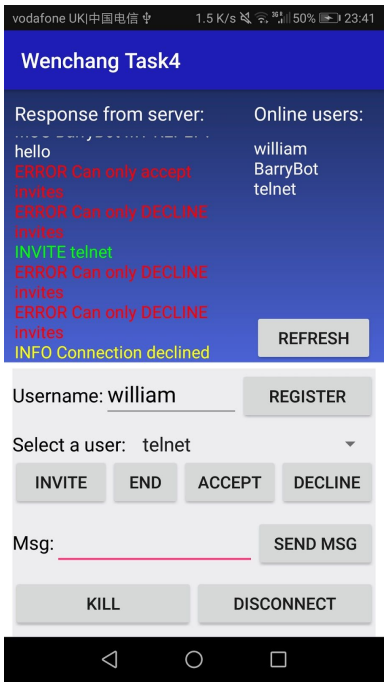

Figure 2-12 decline an invitation

11. Invite our android device again, and this time we accept the invitation by pressing “accept” button, then try to send some messages to each other.
    We will be informed that the connection has been set up, and we can easily talk to telnet with GUI, we can also see that telnet can receive our messages successfully.

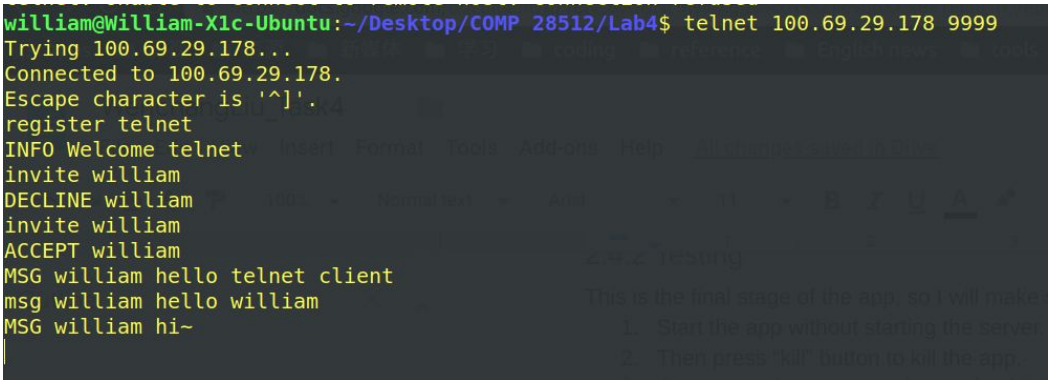

Figure 2-13 telnet client screenshot

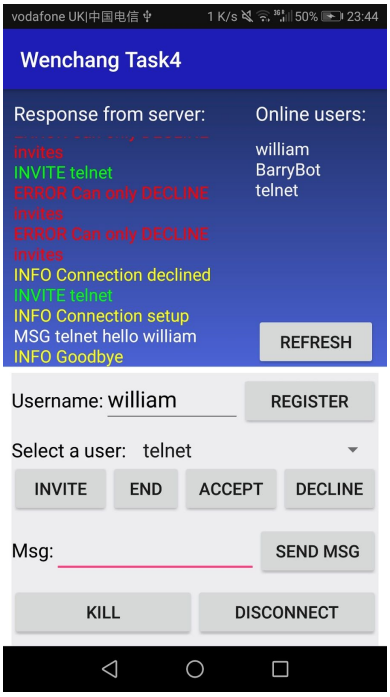

Figure 2-13 send message to telnet and disconnect

12. Finally, try “disconnect” button.
    We will see the server say goodbye to us.

2.4.3 Evaluation

Fully-functional and user-friendly app: This is the final stage of the app, we implemented several improvements on stage 2, it has better appearance and user experience compared with previous version.

3. Answer Questions

Part 4.1

1. Are you satisfied that the two Python programs ‘lab4-server.py’ and ‘BarryBot.py’ will allow you to test a simple messaging system developed in Android?
    - Yes. The server could be a real big server and both BarryBot and telnet we run can be regarded as Android devices clients, the clients can message each other via the server we provided.
    - There are some problems (mentioned in Part 4.5), but it is OK for a small server-client communication.
2. If BarryBot were a real user, how would he know when the TELNET client has terminated your connection to him? How could the protocol be improved in this respect?

- It depends on how the client terminated the connection, if it uses “end” command, then the other user can see the connection has ended. However, if the client has been “killed”(e.g. In telnet pressed ^] and quit) or “disconnect” from the server, then the other user will not receive any message, and, If BarryBot sends another message to telnet client after telnet has terminated, then there will be a critical error “ERROR Critical error. Please see trace on server”.
- The protocol can be improved by when a user disconnect the server, the server will automatically end the connections related to the disconnected user and send a message to tell those clients who has a connection with the disconnected user.

3. (a) Why are literal strings discouraged in Android?
    - Because suppose you want to the same text in more than one places in your app. It will be difficult to edit strings if we put the text every time hard coded, especially if the text is somewhat large. Also, this will be a waste of storage.

   (b) What is the preferred alternative?
    - Using @string/your_string syntax and put all the strings into string.xml.
4. What is the purpose of the following files and directories created by Gradle:

   (a) src/main/java
    - The directory contains all source codes and test files. These are the main logic of the whole program, we describe how the program will perform.

   (b) res/
    - The directory contains the resources files, which determine how the app will look like, such as layouts, colors, styles (xml files) and so on.

   (c) AndroidManifest.xml
    - This file is used to store the information for the whole project. Resides within '...\main' and defines some characteristics of your application including its name, permissions needed, SDK version (version of Android studio), etc.
5. How does the naming of a package affect the file structure of an Android project?
    - Package is like namespace(cpp) for java program, declarations like “protected”, or default will take effects on different packages.
    - Several classes with an exactly the same name can be used in a single project as long as they belong to different packages which distinguish one class from another.
    - The package is the directory structure of the source codes of the project. We can put the related classes into the same packages to make the structure clearer and tidier.

Part 4.2

1. Where did you store your fixed text message?
    - In the transmitterText string defined in MainAcitivity class, this the the string that will be sent to the server.
2. How did you check that the ‘btnSendCmd’ and ‘btnKill’ buttons had the desired effect?

- By using the simulator in Android Studio, we can simply click these buttons with our mouse to check whether they can perform as expected.
- Also using logcat to track the message that send to the server and received from the server.

3. Explain how you used the logcat debugging facility and show an example of the output obtained (in your demo and your report).
    - For example, we can use logcat where we perform a action(e.g. click a button) to check whether the button was actually clicked or not if the action is not performed.
    - Here is an example that I used logcat to track the click action on “send”, so if there are anything wrong (e.g. nothing was transmitted, text is wrong, etc.) then we can check our logcat to see what happened when we clicked the send button (e.g. Is the text correct? Is the button being actually clicked?)

```
// Button click handler
sendButton.setOnClickListener((v) → {
        // OnClick actions here
        // Instantiate the transmitter passing the output stream and text to it
        if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that output stream has be setup
            Log.i(LOGTAG, msg: "transmission started, text:"+transmitterText);
            Transmitter transmitter = new Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
            transmitter.start();        // Run on its own thread
        } else {
            Log.e(LOGTAG, msg: "output stream not set");
        }
});
```

Figure 3-1 a screenshot of an example code segment of using logcat

```
03-17 17:19:08.987 3546-3590/com.example.mbassjsp.task4 I/Transmitter: Thread now closing
03-17 17:21:17.205 3719-3719/com.example.mbassjsp.task4 I/Main UI: send button clicked, prepare to transmit text:register william
03-17 17:21:17.213 3719-3756/com.example.mbassjsp.task4 I/Transmitter: Running in new thread
03-17 17:21:17.213 3719-3756/com.example.mbassjsp.task4 I/Transmitter: Sending command: register william
03-17 17:21:17.216 3719-3756/com.example.mbassjsp.task4 I/Transmitter: Thread now closing
```

Figure 3-2 a screenshot of an example logcat output

4. How does the ‘runOnUiThread’ method deal with inter-thread communications in Android?
    - A non-UI thread communicates the desire to request work be run on the UI thread. This is accomplished under the covers by publishing the requested action to the event queue of the UI thread. When it can, the UI thread picks up the action message in the event queue and performs the UI change.
5. Why is ‘runOnUiThread’ needed for the NetworkConnectionsAndReceiver thread but not for the Transmitter thread?
    - Because we need to display the message we received from server in the UI, but we do not need to display what we send to the server(even if we want to show what we send to the server, we can directly access in the main thread).

Part 4.3

1. How is your screen layout (however simple) made suitable for the application as developed so far?

- This is a very simple layout to satisfy all the requirements from part 4.3 including the name of the app “task4”, a textView to show all responses from the server on the top left corner, another textView to show current online users and a button to refresh it as required, a linear layout for register function, two buttons at the end for users to stop the app or disconnect from server and a linear layout to send any command like the what we can do in terminal(convenient to debug).

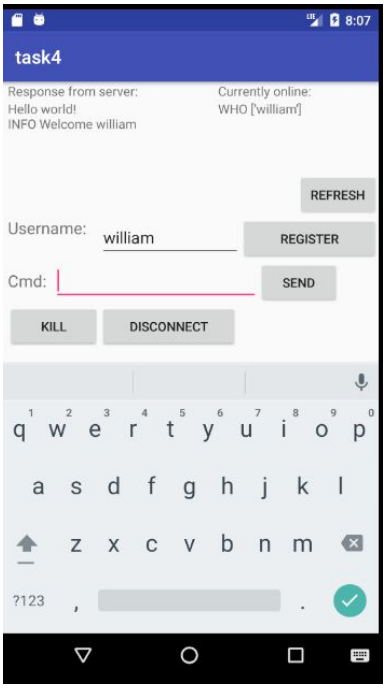

Figure 3-3 Part 4.3 layout screen shot

2. What happens if the proxy-server is not on-line?
    - Because all we have developed client, so if the server is not on-line, we cannot receive(or send) any message to(or from) others or the server. Also, logcat will tell us “socket failed”.
3. What happens if you try to register with a name that is already in use?
    - We will receive an error message “ERROR Username already taken” from server.
4. Could anything else go wrong with this preliminary version of the app?
    - Don’t think anything could go wrong, all the error will be displayed in the textView, however, it could be inconvenient and not user friendly, e.g. most of the functions can only been used by sending “cmd”.

Part 4.4

1. What are the main features of your new layout and how do they work?
    - The new layout extends the layout in part 4.3, this part I add all the functions into the new layout, we can send messages with other online users using buttons

easily, new features including invite other users to chat, accept or decline other users' invitations, and send one line simple message to those who has set up connections with you.

- Online users will be shown on the top right corner and you can press "refresh" button to get the latest online users information whenever you want, all other responses from server will be shown in the top left textView.
  If you want to register, you simply input your username in the editText box and press register, then you will receive info message from the server.
  If you want to invite other people, you just put the user's username in the editText box and press invite, when other people accept or decline, the server will send message to let you know.
  Also, if you receive invitations from other user, you can put his/her name in the editText box and choose accept to set up connection with him/her or decline the invitation.
  And after set up connections with others, you can send message to others by first typing the username you want to chat with in the editText box, and then just typing the content and press "send msg" button to send your message.
  All the things you can do with buttons and editText box you can also do it by using "send cmd" as 4.3, however, since we can do all the things we need from buttons, the "cmd" can just needed for us to debug.

Figure 3-4 Part 4.4 layout

2. What are the possible disadvantages of using the proxy-server's MSG command to convey the communications to and from clients, especially if you are thinking about introducing spoken messages and multimedia?

- It is not convenient to convey audio or other media messages using this type of communication, because all we transmit or receive is just strings.
- Group chats are not allowed.

Part 4.5

1. What are your improvements?
    - The improvements including:
        (1) Improve app appearances by changing background and text colors.
        (2) Inform the user if the app cannot connect the server.
        (3) Forbid user to change the register editText box after registered.
        (4) Parse the “WHO” response to show the actual users, not the whole string.
        (5) Combine editText boxes of invite/accept/decline/msgTo into one “select user” dropdown list(will be updated with online users’ textView), in this way, users can simply select the other user instead of inputting the whole username, which will be less laborious and can avoid typos.
        (6) Add end function to end the connection with other users.
        (7) Add auto scroll function to the “Response from server” textView to guarantee we can always see the latest response from the server. Also increase the ratio of this textView for better user experience.
        (8) Different kind of messages now have different colours to be easier distinguished.
        (9) Change all the layout to “match_parent” in order to adapt all kinds of android machines.

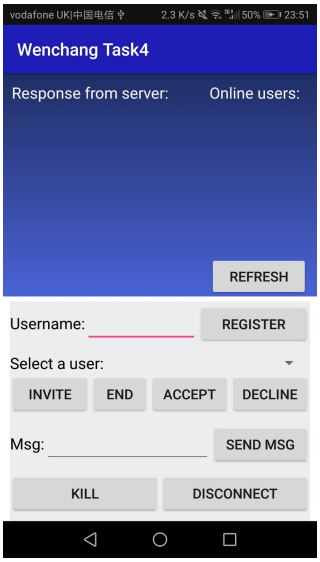

Figure 3-5 Part 4.5 layout

2. Are there any deficiencies in the protocol as currently implemented by the proxy-server?

- It is inconvenient to transmit audio or multimedia messages.
- It can only achieve 1 to 1 chat, group chats are not allowed.
- Server cannot detect offline behavior actively. If an user quit the program without disconnecting the server(e.g. using ^C), the server will still regard him as one of the online users.
- Critical error occurs when sending message to offline users because the server will still keep the connections even if a user disconnects from the server without end connections with others.
- Lack of responses, e.g. end the connections with others will not receive any messages from to server to show whether succeeded or not.
- It is inconvenient to implement auto update on online users' textView, if we want to implement auto update online users list, we will need to consistently send "who" to the server to update the list.

3. How could the proxy-server be improved?
    - Need some more complex protocol rather than using string to transmit audio or multimedia messages.
    - When a user create a group chat, we can simply regard the "group" as a user and make all users who join the group set up a connection with the "group", then when a user send msg + group name + message to the server, the server just make the "group" send the same message to all users who have a connection with it.
    - We can see that when a client terminate without disconnect from the server, the server will show an error, the server just need to remove the offline user who cause this error from the online list.
    - We should make the server to end all connections that the disconnected user has and inform all other users who have connection with the disconnected user.
    - Add more responses, e.g. tell the user who send "end" to the server if he/she has successfully end the connections with others.
    - It could be better if the server can tell the clients if someone disconnected or registered, in this way, if we want to implement automatically update the online user list, we can do it like "interrupt" style, which is only do updates when the change happens, otherwise, we have to constantly send "who" to the server to check online users like "polling" style, which will be a waste of resources.

4. Code and Documentation

This part will contain code that I developed with some documentation for those codes. Codes in the template or automatically generated by android studio are not included here.

(1) strings.xml

Documentation:

This part is just store all the literal strings that we are going to use in the UI, this is better for us to update the strings in the UI and use less storage to store them if some same strings appear more than once.

Code:

```
<resources>
    <string name="app_name">Wenchang Task4</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="send">Send</string>
    <string name="cmd">Type Command</string>
    <string name="register">Register</string>
    <string name="response_from_server">Response from server:</string>
    <string name="online_user">Online users:</string>
    <string name="refresh">Refresh</string>
    <string name="username">Username:</string>
    <string name="select_user">Select a user:</string>
    <string name="invite">invite</string>
    <string name="end">end</string>
    <string name="accept">accept</string>
    <string name="decline">decline</string>
    <string name="msg">Msg:</string>
    <string name="msg_send">send msg</string>
    <string name="kill">kill</string>
    <string name="disconnect">disconnect</string>
</resources>
```

(2) activity_main.xml

Documentation:

This part is where we put all our UI design, we basically used linear layout, textView, editText, Buttons, etc. We can define the properties of the “view” in this file, we can make the UI adapt to all size of screens by using “match_parent”, and we can adjust

“layout_weight” to adjust the ratio of each widget in each linear layout. The id is used for us to get the UI widget in activity, and we set text on the widgets and so on.

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="com.example.mbassjsp.task4.MainActivity">

   <LinearLayout
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_alignParentEnd="true"
      android:layout_alignParentRight="true"
      android:layout_marginLeft="0dp"
      android:layout_marginTop="0dp"
      android:layout_marginEnd="0dp"
      android:layout_marginRight="0dp"
      android:background="@drawable/background_alter"
      android:orientation="vertical"
      android:weightSum="1">

      <LinearLayout
         android:layout_width="match_parent"
         android:layout_height="match_parent"
         android:layout_marginLeft="10dp"
         android:layout_marginTop="10dp"
         android:layout_marginRight="10dp"
         android:layout_weight="0.5"
         android:orientation="horizontal">

         <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginRight="15dp"
            android:layout_weight="1"
            android:orientation="vertical">
```

```
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/response_from_server"
        android:textColor="@color/white"
        android:textSize="18dp" />

    <TextView
        android:id="@+id/txtServerResponse"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:layout_marginTop="10dp"
        android:gravity="top"
        android:maxLines="200"
        android:scrollbars="vertical"
        android:text=""
        android:textColor="@color/white" />

</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_weight="2"
    android:orientation="vertical">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/online_user"
        android:textColor="@color/white"
        android:textSize="18dp" />

    <TextView
        android:id="@+id/onlineList"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="10dp"
        android:layout_weight="1"
        android:gravity="top"
        android:maxLines="200"
```

```
            android:scrollbars="vertical"
            android:textColor="@color/white" />

        <Button
            android:id="@+id/btnRefresh"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="right"
            android:text="@string/refresh" />

    </LinearLayout>

</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="5dp"
    android:layout_marginRight="8dp"
    android:layout_weight="0.5"
    android:background="@color/inputArea"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <TextView
            android:id="@+id/textViewRegister"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/username"
            android:textColor="@android:color/background_dark"
            android:textSize="18dp" />

        <EditText
            android:id="@+id/userName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:inputType="text"
```

```
        android:textColor="@android:color/background_dark" />

    <Button
        android:id="@+id/btnRegister"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/register" />
</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="10dp"
    android:orientation="horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/select_user"
        android:textColor="@android:color/background_dark"
        android:textSize="18dp"></TextView>

    <Spinner
        android:id="@+id/spinnerWho"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="8dp"
        android:layout_weight="1"></Spinner>

</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">

    <Button
        android:id="@+id/btnInvite"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom"
        android:layout_weight="1"
```

```
        android:text="@string/invite" />

    <Button
        android:id="@+id/btnEnd"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1.1"
        android:text="@string/end" />

    <Button
        android:id="@+id/btnAccept"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/accept" />

    <Button
        android:id="@+id/btnDecline"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/decline" />

</LinearLayout>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:orientation="horizontal">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/msg"
        android:textColor="@android:color/background_dark"
        android:textSize="18dp" />

    <EditText
        android:id="@+id/msgContent"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="2"
```

```
        android:inputType="text"
        android:textColor="@android:color/background_dark" />

    <Button
        android:id="@+id/btnSendMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="3"
        android:text="@string/msg_send" />

</LinearLayout>

<!--<LinearLayout-->
<!--android:layout_width="match_parent"-->
<!--android:layout_height="wrap_content"-->
<!--android:orientation="horizontal">-->

<!--<TextView-->
<!--android:id="@+id/textViewCommand"-->
<!--android:layout_width="50dp"-->
<!--android:layout_height="30dp"-->
<!--android:textSize="18dp"-->
<!--android:text="Cmd:" />-->

<!--<EditText-->
<!--android:id="@+id/command"-->
<!--android:layout_width="match_parent"-->
<!--android:layout_height="wrap_content"-->
<!--android:layout_weight="2"-->
<!--android:inputType="text" />-->

<!--<Button-->
<!--android:id="@+id/btnSendCmd"-->
<!--android:layout_width="match_parent"-->
<!--android:layout_height="wrap_content"-->
<!--android:layout_gravity="right"-->
<!--android:layout_weight="3"-->
<!--android:text="@string/send" />-->
<!--</LinearLayout>-->

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
            android:layout_marginTop="10dp"
            android:orientation="horizontal">


            <Button
                android:id="@+id/btnKill"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="left"
                android:layout_weight="1"
                android:text="@string/kill" />

            <Button
                android:id="@+id/btnDisconnect"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="left"
                android:layout_weight="1"
                android:text="@string/disconnect" />

            <!--<Button-->
            <!--android:id="@+id/btnConnect"-->
            <!--android:layout_width="match_parent"-->
            <!--android:layout_height="wrap_content"-->
            <!--android:layout_weight="1"-->
            <!--android:text="Connect"/>-->

        </LinearLayout>
    </LinearLayout>
</LinearLayout>

</RelativeLayout>
```

(3) MainActivity.java

Documentation:

This is where we control our UI behavior, what we basically do here is to find the widget we defined in activity_main.xml using findViewById(), get the text that the user input in

editText or spinner using getText().toString() or getSelectedItem().toString(), set onClickListener public void onClick(View v) on buttons so we can transmit some messages according to the protocol when the user press the button, this is basically the client to server events oriented programming, which means that, if some important events happen in the client, we will tell the server by transmitting a message.

I implemented the spinner following a tutorial online, the spinner will dynamically changed with the list in the array adapter, we can acquire the online list and update this list each time we send “who” to the server. I also want to make sure that the message can be sent at a certain order so I used Thread.sleep() to do that.

Code:

```
package com.example.mbassjsp.task4;

// Created by A Leeming
// Modified JSP
// Date 17-1-2018
// See https://developer.android.com ,for android classes, methods, etc


// Import classes
	import android.support.v7.app.AppCompatActivity;
	import android.os.Bundle;
	import android.text.method.ScrollingMovementMethod;
	import android.util.Log;
	import android.view.Menu;
	import android.view.MenuItem;
	import android.view.View;
	import android.view.inputmethod.InputMethodManager;
	import android.widget.ArrayAdapter;
	import android.widget.Button;
	import android.widget.EditText;
	import android.widget.Spinner;
	import android.widget.TextView;

	import java.util.ArrayList;
	import java.util.Arrays;
	import java.util.List;

// Android apps must have a MainActivity class that extends Activity or
AppCompatActivity class
public class MainActivity extends AppCompatActivity {
```

```
    //Declare class variables
    private static final String LOGTAG = "Main UI"; //Logcat messages from UI are
identified
    private NetworkConnectionAndReceiver networkConnectionAndReceiver = null;
    private String transmitterText; //Transmitter data variable
    private String registerText; // register+username

    // Class methods
    @Override
    //Extend the onCreate method, called whenever an activity is started
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState); // Extend the onCreate method
        // Set up the view using xml description res>layout>activity_main.xml
        setContentView(R.layout.activity_main);

        Log.i(LOGTAG, "Starting task4 app"); // Report to Logcat

        // Instantiate the network connection and receiver object
        networkConnectionAndReceiver = new NetworkConnectionAndReceiver(this);
        networkConnectionAndReceiver.start();    // Start socket-receiver thread


        // Get the receiving text area as defined in the Res dir xml code
        TextView receiverTextArea = findViewById(R.id.txtServerResponse);
        // Make the receiving text area scrollable
        receiverTextArea.setMovementMethod(new ScrollingMovementMethod());

        // Get the kill button as defined in the Res dir xml code
        Button killButton = findViewById(R.id.btnKill);
        // Make the kill button receptive to being clicked
        // Button click handler
        killButton.setOnClickListener(new View.OnClickListener() {
            // onClick method implementation for the killButton object
            public void onClick(View v) {
                // OnClick actions here
                // Exit app
                if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
                    transmitterText = "disconnect";
                    Log.i(LOGTAG, "kill button clicked, prepare to transmit
text:"+transmitterText);
                    Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
```

```
                transmitter.start();        // Run on its own thread
            } else {
                Log.e(LOGTAG, "output stream not set");
            }
            Log.i(LOGTAG, "kill this app");
            System.exit(0);
        }
    });

    // register function
    final Spinner targetName = findViewById(R.id.spinnerWho);
    final List<String> list = new ArrayList<String>();

    final ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, list);

dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_it
em);
    targetName.setAdapter(dataAdapter);
    targetName.setPrompt("Select an online user:");

    final EditText userNameText;
    userNameText = findViewById(R.id.userName);
    Button registerButton = findViewById(R.id.btnRegister);
    registerButton.setOnClickListener(new View.OnClickListener() {
        // onClick method implementation for the sendButton object
        public void onClick(View v) {
            // OnClick actions here
            // Instantiate the transmitter passing the output stream and text to it
            if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
                registerText = "register " + userNameText.getText().toString();
                userNameText.setCursorVisible(false);
                userNameText.setFocusable(false);
                userNameText.setFocusableInTouchMode(false);
                Log.i(LOGTAG, "register button clicked, prepare to transmit
text:"+registerText);
                Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), registerText);
                transmitter.start();        // Run on its own thread

                try {
                    Thread.sleep(200);
```

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

                transmitterText = "who";
                Log.i(LOGTAG, "after registered, who is online "+transmitterText);
                Transmitter transmitter2 = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
                transmitter2.start();        // Run on its own thread

                // update dropdown list
                try {
                    Thread.sleep(200);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                String[] onlineUsers = networkConnectionAndReceiver.getOnlineUsers();
                if(onlineUsers != null) {
                    list.clear();
                    for (int i = 0; i < onlineUsers.length; i++) {
                        if (!onlineUsers[i].equals('', '')) {
                            list.add(onlineUsers[i]);
                        }
                    }
                    targetName.setAdapter(dataAdapter);
                }
            } else {
                Log.e(LOGTAG, "output stream not set");
            }
        }
    });

    // set up connections
    Button inviteButton = findViewById(R.id.btnInvite);
    // Button click handler
    inviteButton.setOnClickListener(new View.OnClickListener() {
        // onClick method implementation for the sendButton object
        public void onClick(View v) {
            // OnClick actions here
            // Instantiate the transmitter passing the output stream and text to it
            if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
                if(targetName.getSelectedItem() == null) {
```

```
                Log.e(LOGTAG, "drop down list value is null");
                transmitterText = "";
            } else {
                transmitterText = "invite " + targetName.getSelectedItem().toString();
            }
            Log.i(LOGTAG, "invite button clicked, prepare to transmit
text:"+transmitterText);
            Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
            transmitter.start();        // Run on its own thread
        } else {
            Log.e(LOGTAG, "output stream not set");
        }
    }
});

// set up connections
Button endButton = findViewById(R.id.btnEnd);
// Button click handler
endButton.setOnClickListener(new View.OnClickListener() {
    // onClick method implementation for the sendButton object
    public void onClick(View v) {
        // OnClick actions here
        // Instantiate the transmitter passing the output stream and text to it
        if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
            if(targetName.getSelectedItem() == null) {
                Log.e(LOGTAG, "drop down list value is null");
                transmitterText = "";
            } else {
                transmitterText = "end " + targetName.getSelectedItem().toString();
            }
            Log.i(LOGTAG, "end button clicked, prepare to transmit
text:"+transmitterText);
            Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
            transmitter.start();        // Run on its own thread
        } else {
            Log.e(LOGTAG, "output stream not set");
        }
    }
});
```

```
        Button acceptButton = findViewById(R.id.btnAccept);
        Button declineButton = findViewById(R.id.btnDecline);
        // Button click handler
        acceptButton.setOnClickListener(new View.OnClickListener() {
            // onClick method implementation for the sendButton object
            public void onClick(View v) {
                // OnClick actions here
                // Instantiate the transmitter passing the output stream and text to it
                if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
                    if(targetName.getSelectedItem() == null) {
                        Log.e(LOGTAG, "drop down list value is null");
                        transmitterText = "";
                    } else {
                        transmitterText = "accept " + targetName.getSelectedItem().toString();
                    }
                    Log.i(LOGTAG, "accept button clicked, prepare to transmit
text:"+transmitterText);
                    Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
                    transmitter.start();        // Run on its own thread
                } else {
                    Log.e(LOGTAG, "output stream not set");
                }
            }
        });
        // Button click handler
        declineButton.setOnClickListener(new View.OnClickListener() {
            // onClick method implementation for the sendButton object
            public void onClick(View v) {
                // OnClick actions here
                // Instantiate the transmitter passing the output stream and text to it
                if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
                    if(targetName.getSelectedItem() == null) {
                        Log.e(LOGTAG, "drop down list value is null");
                        transmitterText = "";
                    } else {
                        transmitterText = "decline " + targetName.getSelectedItem().toString();
                    }
                    Log.i(LOGTAG, "decline button clicked, prepare to transmit
text:"+transmitterText);
```

```
                Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
                transmitter.start();        // Run on its own thread
            } else {
                Log.e(LOGTAG, "output stream not set");
            }
        }
    });

//      // Get the text area for commands to be transmitted as defined in the Res dir xml
code
//      final EditText command;
//      command = findViewById(R.id.command);
//      // Get the send button as defined in the Res dir xml code
//      Button sendButton =  findViewById(R.id.btnSendCmd);
//      // Make the kill button receptive to being clicked
//      // Button click handler
//      sendButton.setOnClickListener(new View.OnClickListener() {
//          // onClick method implementation for the sendButton object
//          public void onClick(View v) {
//              // OnClick actions here
//              // Instantiate the transmitter passing the output stream and text to it
//              if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
//                  transmitterText = command.getText().toString();
//                  command.setText("");
//                  Log.i(LOGTAG, "send button clicked, prepare to transmit
text:"+transmitterText);
//                  command.clearFocus();
//                  Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
//                  transmitter.start();        // Run on its own thread
//              } else {
//                  Log.e(LOGTAG, "output stream not set");
//              }
//          }
//      });

        // set disconnect button
        Button disconnectButton = findViewById(R.id.btnDisconnect);
        disconnectButton.setOnClickListener(new View.OnClickListener() {
            // onClick method implementation for the sendButton object
            public void onClick(View v) {
```

```
            // OnClick actions here
            // Instantiate the transmitter passing the output stream and text to it
            if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
                transmitterText = "disconnect";
                Log.i(LOGTAG, "disconnect button clicked, prepare to transmit
text:"+transmitterText);
                Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
                transmitter.start();        // Run on its own thread
            } else {
                Log.e(LOGTAG, "output stream not set");
            }
        }
    });

    // set refresh button
    Button refreshButton = findViewById(R.id.btnRefresh);
    refreshButton.setOnClickListener(new View.OnClickListener() {
        // onClick method implementation for the sendButton object
        public void onClick(View v) {
            // OnClick actions here
            // Instantiate the transmitter passing the output stream and text to it
            if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
                transmitterText = "who";
                Log.i(LOGTAG, "refresh button clicked, prepare to transmit
text:"+transmitterText);
                Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
                transmitter.start();        // Run on its own thread

                // update dropdown list
                try {
                    Thread.sleep(200);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
                String[] onlineUsers = networkConnectionAndReceiver.getOnlineUsers();
                if(onlineUsers != null) {
                    list.clear();
                    for (int i = 0; i < onlineUsers.length; i++) {
                        if (!onlineUsers[i].equals('', '')) {
```

```
                        list.add(onlineUsers[i]);
                    }
                }
                targetName.setAdapter(dataAdapter);
            }

        } else {
            Log.e(LOGTAG, "output stream not set");
        }
    }
});

// msg button
final EditText msgContent;
msgContent = findViewById(R.id.msgContent);
Button msgButton = findViewById(R.id.btnSendMsg);
msgButton.setOnClickListener(new View.OnClickListener() {
    // onClick method implementation for the sendButton object
    public void onClick(View v) {
        // OnClick actions here
        // Instantiate the transmitter passing the output stream and text to it
        if(networkConnectionAndReceiver.getStreamOut() != null) { // Check that
output stream has be setup
            if(targetName.getSelectedItem() == null) {
                Log.e(LOGTAG, "drop down list value is null");
                transmitterText = "";
            } else {
                transmitterText = "msg " + targetName.getSelectedItem().toString() + " " +
msgContent.getText().toString();
            }
            Log.i(LOGTAG, "msgSend button clicked, prepare to transmit
text:"+transmitterText);
            // msgTarget.setText("");
            msgContent.setText("");
            Transmitter transmitter = new
Transmitter(networkConnectionAndReceiver.getStreamOut(), transmitterText);
            transmitter.start();        // Run on its own thread
        } else {
            Log.e(LOGTAG, "output stream not set");
        }
    }
});
```

```
    } //End of app onCreate method

/*  // Following code used when using basic activity
    @Override
    //Create an options menu
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        // Uses res>menu>main.xml
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    } //End of app onCreateOptionsMenu

    @Override
    //Called when an item is selected from the options menu
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    } //End of app onOptionsItemSelected method
*/

}//End of app MainActivity class
```

(4) NetworkConnectionAndReceiver.java

Documentation:

This part is to set up the socket to communicate with the server, and we also receive the message from the server and display them in our textView.

I made several improvements on the template including:

- A get method for the spinner UI
- Display an error message in the main activity when catch a exception
- Make “WHO” response to display in another view by using message.indexOf()

- Using a new string message2 to solve the overlap problem when we transmit 2 messages in a single time(we want to automatically get online users when the user register).
- Parse “WHO” response string using substring() and split()
- Use SpannableStringBuilder to color messages according to the prefix of the messages
- Implement auto scroll to the bottom on a textView by calculating the offset

Code:

```
package com.example.mbassjsp.task4;

// Created by A Leeming
// Modified JSP
// Date 17-1-2018
// See https://developer.android.com ,for android classes, methods, etc
// Code snippets from
http://examples.javacodegeeks.com/android/core/socket-core/android-socket-example

// import classes
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.text.Spannable;
import android.text.SpannableStringBuilder;
import android.text.method.ScrollingMovementMethod;
import android.text.style.ForegroundColorSpan;
import android.util.Log;
import android.widget.TextView;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;

public class NetworkConnectionAndReceiver extends Thread{
    //Declare class variables
    private Socket socket = null;
    private static final int SERVERPORT = 9999; // This is the port that we are connecting
to
    // Channel simulator is 9998
```

```
   private static final String SERVERIP = "10.0.2.2";  // This is the host's loopback
address
   // private static final String SERVERIP = "100.69.29.178";
   private static final String LOGTAG = "Network and receiver"; //  Identify logcat
messages

   private boolean terminated = false; // When FALSE keep thread alive and polling

   private PrintWriter streamOut = null; // Transmitter stream
   private BufferedReader streamIn = null; // Receiver stream
   private AppCompatActivity parentRef; // Reference to main user interface(UI)
   private TextView receiverDisplay; // Receiver display
   private TextView onlinelistDisplay;
   private String message = null; //Received message
   private String[] onlineUsers = null;

   //class constructor
   public NetworkConnectionAndReceiver(AppCompatActivity parentRef)
   {
      this.parentRef=parentRef; // Get reference to UI
   }
   // Start new thread method
   public void run()
   {
      Log.i(LOGTAG,"Running in new thread");

      //Create socket and input output streams
      try {   //Create socket
         InetAddress svrAddr = InetAddress.getByName(SERVERIP);
         // InetAddress svrAddr = InetAddress.getLocalHost();
         socket = new Socket(svrAddr, SERVERPORT);
         Log.i(LOGTAG, "socket: " + socket.toString());

         //Setup i/o streams
         streamOut = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()), true);
         streamIn = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
      }
      catch (UnknownHostException uhe) {
         Log.e(LOGTAG, "Unknownhost\n" + uhe.getStackTrace().toString());
         receiverDisplay = parentRef.findViewById(R.id.txtServerResponse);
         //Run code in run() method on UI thread
```

```
            parentRef.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    // Display message, and old text in the receiving text area
                    receiverDisplay.setTextColor(Color.RED);
                    receiverDisplay.append("\n" + "Unknownhost error, please check your
network connection and restart the app.\n");
                }
            });
            terminated = true;
        }
        catch (Exception e) {
            Log.e(LOGTAG, "Socket failed\n" + e.getMessage());
            e.printStackTrace();
            receiverDisplay = parentRef.findViewById(R.id.txtServerResponse);
            //Run code in run() method on UI thread
            parentRef.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    // Display message, and old text in the receiving text area
                    receiverDisplay.setTextColor(Color.RED);
                    receiverDisplay.append("\n" + "Socket failed, please check your network
connection and restart the app.\n");
                }
            });
            terminated = true;
        }

        //receiver
        while(!terminated) // Keep thread running
        {
            try {
                message = streamIn.readLine(); // Read a line of text from the input stream
                // If the message has text then display it
                if (message != null && message != "") {
                    Log.i(LOGTAG, "MSG recv : " + message);
                    if(message.indexOf("WHO") == 0) {
                        // if request to show online list
                        onlinelistDisplay = parentRef.findViewById(R.id.onlineList);
                        //Run code in run() method on UI thread
                        String message2 = message.substring(6, message.length()-2);
                        onlineUsers = message2.split(",");
                        parentRef.runOnUiThread(new Runnable() {
```

```
                    @Override
                    public void run() {
                        // Display message, and old text in the receiving text area
                        onlinelistDisplay.setMovementMethod(new
ScrollingMovementMethod());
                        onlinelistDisplay.setText("");
                        for(int i = 0; i < onlineUsers.length; i++) {
                            if(!onlineUsers[i].equals(", ")) {
                                onlinelistDisplay.append(onlineUsers[i] + "\n");
                            }
                        }
                    }
                });
            } else {
                //Get the receiving text area as defined in the Res dir xml code
                receiverDisplay = parentRef.findViewById(R.id.txtServerResponse);
                //Run code in run() method on UI thread
                final String message2 = new String(message);
                parentRef.runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        // Display message, and old text in the receiving text area
                        SpannableStringBuilder style = new SpannableStringBuilder("\n" +
message2);
                        receiverDisplay.setMovementMethod(new
ScrollingMovementMethod());
                        if(message2.indexOf("ERROR") == 0 || message2.indexOf("DUMP")
== 0) {
                            style.setSpan(new
ForegroundColorSpan(Color.RED),0,message2.length()+1,
Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
                        } else if(message2.indexOf("MSG") == 0) {
                            style.setSpan(new
ForegroundColorSpan(Color.WHITE),0,message2.length()+1,
Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
                        } else if(message2.indexOf("INFO") == 0) {
                            style.setSpan(new
ForegroundColorSpan(Color.YELLOW),0,message2.length()+1,
Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
                        } else {
                            style.setSpan(new
ForegroundColorSpan(Color.GREEN),0,message2.length()+1,
Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
```

```
                        }
                        receiverDisplay.append(style);
                        int offset =
receiverDisplay.getLineCount()*receiverDisplay.getLineHeight();
                        if (offset > receiverDisplay.getHeight()) {
                            receiverDisplay.scrollTo(0,offset-receiverDisplay.getHeight());
                        }
                    }
                });
            }
        }
        catch (Exception e) {
            Log.e(LOGTAG, "Receiver failed\n" + e.getMessage());
            e.printStackTrace();
        }

    }
    //Call disconnect method to close i/o streams and socket
    disconnect();
    Log.i(LOGTAG,"Thread now closing");
}


//  Method for closing socket and i/o streams
public void disconnect()
{
    Log.i(LOGTAG, "Closing socket and io streams");
    try {
        streamIn.close();
        streamOut.close();
    }
    catch(Exception e)
    {/*do nothing*/}

    try {
        socket.close();
    }
    catch(Exception e)
    {/*do nothing*/}
}
// Getter method for returning the output stream for the transmitter to use
public PrintWriter getStreamOut() {return this.streamOut;}
```

```
    // Setter method for terminating this thread
    // Set value to true to close thread
    public void closeThread(boolean value) {this.terminated = value;}

    public String[] getOnlineUsers() {
        return this.onlineUsers;
    }
}
```