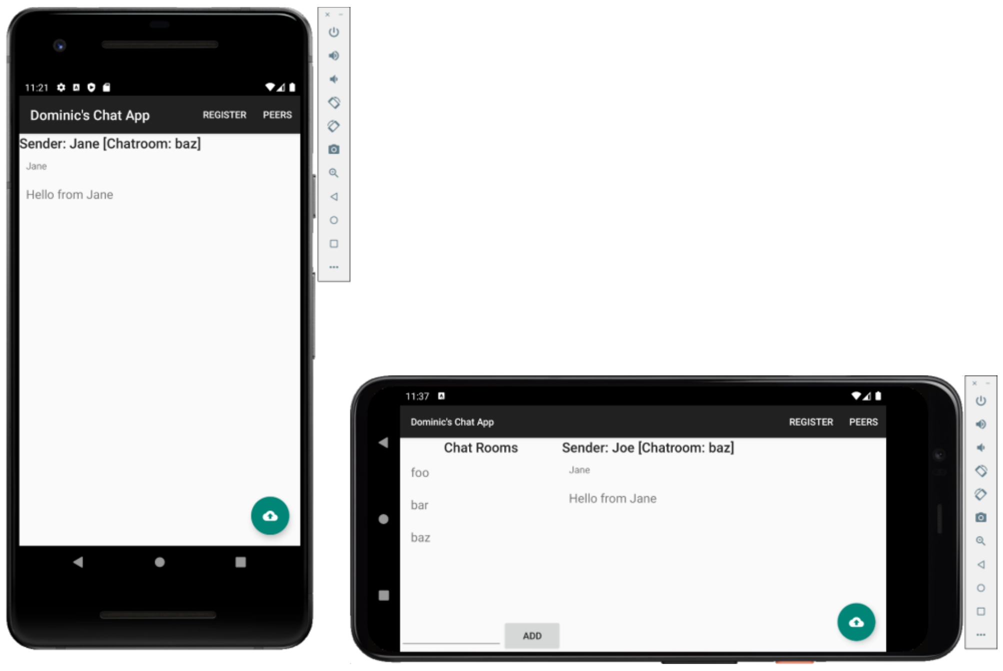
CS 522—Fall 2022
Mobile Systems and Applications
Assignment Eight—Chat App with Service

In this assignment, you will combine the client and server apps from the previous Chat app assignments into a single Chat app, that will allow bidirectional communication between different Android devices.



Sender: Jane [Chatroom: baz]
Jane
Hello from Jane

Sender: Joe [Chatroom: baz]
Jane
Hello from Jane

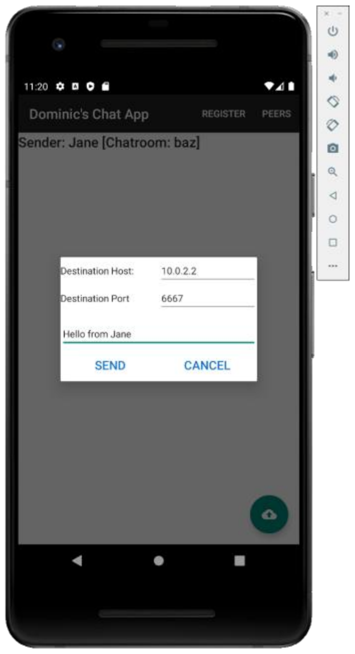
One of the problems with the previous server app is that it does a blocking message receive on the main UI thread when you press the "Next" button. This means that the whole app freezes up until a client sends a message. This is against best practice for Android programming, and we will fix this in the current assignment. The trick is to define the logic for waiting for incoming messages on a separate background thread. The Android component responsible for managing this thread is called a Service. It is like an Activity, but without a UI. In this assignment, you work with a single application. This has a foreground activity, `ChatActivity`, that displays messages received and also allows messages to be sent (so the app is now like a two-way radio). A background service, `ChatService`, handles the sending and receipt of messages in the background. This service should define a background thread, that waits for incoming messages without blocking on the main UI thread. It also defines a background thread for sending messages (use a handler thread for this). Define the service as one that the main chat activity binds to when it starts up.

The background service needs to bind to a UDP port for sending and receiving messages. The chat service will provide functionality to the UI for sending messages. It provides a binder for allowing the UI to call into a service operation for sending a message. The interface provided to the UI client is called `IChatService`, and just provides a single send operation. For this assignment, assume that the UI and services are always in the same process, so you can return the binder as a callback object that provides a reference to the sending service API for the main activity.

In addition, when a message is received by the background service, it needs to update the UI with the new message received. In the case where the new message is added to the message database, this is catered for automatically by the observer that the UI registers with the live data when the database is initially queried. Note that since the message will be received in a background thread (managed by the service), you are allowed to use synchronous persist operations to add data to the database. *You should also add any messages sent by this device to the message database, so the list of messages will include messages both sent and received.*

Since every message has a foreign key reference to the sender record in the database, this record will need to be added to the database. Before the app can send any messages, the user must first "register" (available as an action from the task bar context menu). When a user registers, they choose a name, a peer record with this name is added to the database, and both the name and the primary key for this record are saved in settings. The app will not send any messages until the user has registered.

There is no longer a "Next" button in this app, for synchronously requesting the next message. However, we still have a floating action button, for sending a message, as with the chat client in the previous assignment:



Sender: Jane [Chatroom: baz]

Destination Host: 10.0.2.2
Destination Port 6667
Hello from Jane

The sending of the message will again be performed on a background thread. For simplicity, we will use a single (bound) service to manage the threads for sending and receiving messages.

To test this app, you will now run two or more instances of the same app on different devices. *You should demonstrate communication in both directions between at least two devices.* Create two or more devices as before and run the Chat app on each of the two devices. Let's assume that you are doing TCP forwarding[1], so you have this code in your service:

```
chatConnection = factory.getTcpConnection(chatPort);
```

In the code, the Chat app always binds to TCP port 6666 for receiving messages. The `DatagramConnection` library always sends messages to the local host (identified by address `10.0.2.2` on the Android emulator). The user of the chat app identifies the device they want to send a message to by a TCP port number on the host machine. You have to set up forwarding from that port on the host machine to TCP port 6666 on the recipient machine. You can set up this redirection as you have done for earlier assignments. For example, if you are using TCP for communication:

```
adb devices -l
adb –t transport-id-1 forward tcp:6666 tcp:6666
adb –t transport-id-2 forward tcp:6667 tcp:6666
```

This forwards messages on (TCP) port 6666 on the host machine to port 6666 on the device with transport id *transport-id-1*, and forwards messages on port 6667 on the host machine to port 6666 on the device with transport id *transport-id-2*. For example:

```
# adb devices -l
emulator-5554  ... transport_id:1
emulator-5556  ... transport_id:2
# adb -t 1 forward tcp:6666 tcp:6666
# adb -t 2 forward tcp:6667 tcp:6666
```

Both machines listen on local port 6666. With the definitions above, the first machine should send to port 6667 on the host machine, while the second machine should send to port 6666 on the host machine. The emulator forwards TCP packets sent to ports 6666 and 6667 to port 6666 on the appropriate device (`emulator-5554` and `emulator-5556`, respectively, in this example).

If you wish to use UDP for communication, then you will have to get a UDP connection in the service:

```
chatConnection = factory.getUdpConnection(chatPort);
```

You will have to telnet to the emulator for each device and set up UDP forwarding, as explained in the Assignment Two specification. Again, one device would send to port 6666 and the other to port 6667 on the host machine. You would have set up for example

[1] The `DatagramConnection` library also provides you with the options of UDP or SMS connection. UDP is the option that makes the most sense for this kind of application, if it works on the emulator.

the first emulator `emulator-5554` to forward UDP messages that are sent to (host machine) UDP port 6666 to its AVD port 6666, and you would have set up the second emulator `emulator-5556` to forward UDP messages that are sent to (host machine) UDP port 6667 to its AVD port 6666. The app in the first machine specifies port 6667 as its destination address, which means it gets delivered to the second device. The app in the second machine specifies port 6666 as its destination, which means it gets delivered to the first device., and the emulators then receive the UDP messages and deliver them to your app at port number 6666 on each AVD.

If you wish to use SMS for communication, then the devices can send SMS messages to each other at their emulator phone numbers (5554 and 5556), which the user specifies as the address in the UI for the app.

As in the previous assignment, your app should provide an activity for listing the peers with whom you have been in communication with, and another activity for listing details of a particular peer. You use the same database as in the previous assignment for this app, now used to save both messages received and messages sent.

Submitting Your Assignment
Once you have your code working, please follow these instructions for submitting your assignment:

1. Create a zip archive file, named after you, containing a directory with your name. E.g. if your name is Humphrey Bogart, then name the directory `Humphrey_Bogart`.
2. In that directory you should provide the Android Studio project for your Android app.
3. You should also provide an APK file for your compiled project.

In addition, record short mpeg videos of a demonstration of your assignment working. Make sure that your name appears at the beginning of the video. For example, put your name in the title of the app. *Do not provide private information such as your email or cwid in the video*. See the rubric for what the videos should demonstrate. You must also provide a completed rubric for the assignment.

Your solution should be uploaded via the Canvas classroom. Your solution should consist of a zip archive with one folder, identified by your name. Within that folder, you should have a single Android Studio project, for the app you have built. You should also provide a completed rubric in the root folder, as well as videos demonstrating the working of your assignment.