CS 522—Fall 2018
Mobile Systems and Applications
Assignment Five—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

Set your minimum SDK version to Lollipop (Android 5.1, API 22) for your submission for this assignment. Your app theme should extend `android:Theme.Material.Light.DarkActionBar`. We will be testing your app on a virtual Google Nexus 5X running API 22, so you should ensure that your app works on that platform, although you may develop on another device, e.g., your personal telephone.

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, `ChatApp`, 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 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. Define the service as one that the main chat activity binds to when it starts up. There is no longer a "Next" button in this app.

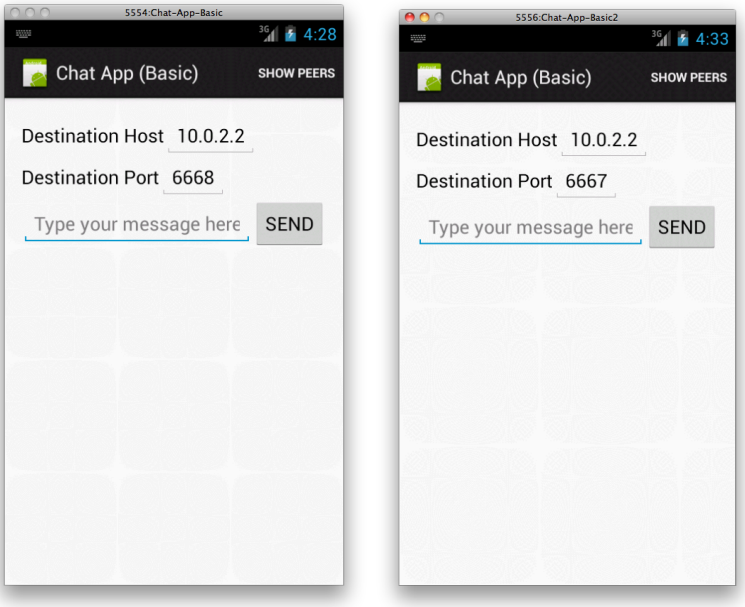
The background service needs to bind to a UDP port for sending and receiving messages. The chat service will provide functionality to the UI for sending messages. It provides a binder for allowing the UI to call into a service operation for sending a message. The interface provided to the UI client is 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 content provider for received messages, this is catered for automatically by the update notification that is sent to the cursor (and hence the cursor loader and loader manager) when the content is updated (in your implementation of insert).

To test this app, you will now run two instances of the same app on different devices. Create two devices as before, and define run configurations for running the Chat app on each of the two devices. We will assume that the Chat app always binds to UDP port 6666 for receiving messages. We extend the UI to specify both the destination host and

port for a message that is being sent. The latter is unnecessary on "real" devices. However for testing on virtual devices, we will specify the development host loopback interface 10.0.2.2 as the destination host, as before. The destination port is then a UDP port on the development host, that should be redirected to port 6666 on the target Android device. You can set up this redirection again using the AVD console, as you have done for earlier assignments.

When you run these two devices, say with the first device named Server and the second device named Client, specify 6668 as the target port for the former and 6667 as the target port for the latter.



Use a `PreferenceActivity` to set the user name in the app. This name is prepended to every message that the app sends. On a receiving app, the name is separated from each incoming request. As in the previous assignment, your should 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.

As with earlier assignments, you must ensure that a query on a content provider does not execute on the UI thread. Build on your solution from the previous assignment, where you used entity managers with asynchronous operations to encapsulate the details of data storage and retrieval, and asynchronous execution on background threads. Use services and background threads in this assignment to handle the network communication on background threads.

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 files for your compiled project.
4. Also include in the directory a report of your submission. This report should be in PDF format. Do not provide a Word document.

In addition, record short mpeg or Quicktime 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.

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.