CS 522—Spring 2015
Mobile Systems and Applications
Assignment Eight—Fragments

In this assignment, you will augment the UI of the Web-based chat app from the previous assignment with a multi-pane interface. You will also get some practice with dialogs. All of your projects for this submission should target Lollipop (Android 5.0.1).

Part 1: Multi-pane Navigation of Chat Rooms

Up until now, we have been assuming a single chat room that all chat participants share. In this assignment, we will generalize this to allow multiple chat rooms. The Web chat server from the previous assignment already supports multiple chat rooms, although you only had to support communication through a "default" chat room. In this assignment, we will extend this with the ability for the chat app to navigate between chat rooms using a multi-pane UI. Ideally this would be used for a tablet interface, however it should also work for a large cell phone or small tablet (e.g. the Nexus 7).

1. In *landscape orientation*, show a user interface that has a navigation pane for chat rooms on the left (say 1/3 to ¼ of the screen), listing the chat rooms that are available. Selecting one of these chat rooms from the list should open up, in the main pane, a list of messages posted to that chat room (along with the identity of the poster, and the time and date of the message. Specify the fragment for the navigation pane in the layout for the activity (using the `fragment` element), but use the `framelayout` element for the details pane, and install a fragment for each chat room dynamically using the fragment manager[1]. The chat room view should initially be empty. If the user presses the BACK button while viewing the contents of a chat room, the chat room view should resume an empty view, but the navigation view should remain.
2. In *portrait orientation*, show (just) a list of the chat rooms, as in earlier assignment. Selecting one of these chat rooms should launch another activity to display the messages in that chat room. In other words, the behavior of your app in portrait orientation is the same as it was in the previous assignment. Your app should be able to dynamically switch between these two forms of user interfaces as the orientation of the device changes. Android will perform this switching for you, as long as you don't prevent adaptation to orientation changes.

In a similar way, adapt the user interface for viewing other users with whom you are exchanging messages. In landscape mode, show a list of the users in a navigation pane on the left. Selecting a user should display a list of chat messages from that particular user, in the details pane. In portrait mode, show these two lists (of peers and peer-specific messages) on separate screens, as in previous assignments.

---

[1] This is not necessary if we are always just displaying chat messages, but this will be useful in future assignments when there is more than one type of inquiry that we will want to make of a chat room.

The main change in the data model for your app is that you will need to add a table for chat rooms to the messages content provider, with a one-to-many relationship from chat rooms to messages:

```
CREATE TABLE Chatrooms (
  _id INTEGER PRIMARY KEY,
  name TEXT NOT NULL,
  ...
);
CREATE TABLE Messages (
  _id INTEGER PRIMARY KEY,
  ...
  sender_fk INTEGER NOT NULL,
  chatroom_fk INTEGER NOT NULL,
  FOREIGN KEY sender_fk REFERENCES Peers(_id) ON DELETE CASCADE,
  FOREIGN KEY chatroom_fk REFERENCES Chatrooms(_id) ON DELETE CASCADE
);
CREATE INDEX MessagesSenderIndex ON Messages(sender_fk);
CREATE INDEX MessagesChatroomIndex ON Messages(chatroom_fk);
```

When displaying chat messages for a particular peer, you should show for each message the chatroom in which that message was posted. You should use a join on the `Messages` and `Chatrooms` tables to combine the text of each message and the chatroom where it was posted. For example:

```
SELECT Chatrooms.name, Messages.messageText, Messages.whenPosted
FROM Chatrooms JOIN Messages
ON Chatrooms._id = Messages.chatroom_fk
WHERE Messages.sender_fk = ...
```

As before, the names of tables and columns should be defined as string literals, stitched into String-valued expressions that produce prepared SQL code. Input arguments (e.g. chat room names or peer names) should be factored into separate selection argument arrays, to prevent SQL injection attacks.

Part 2: Dialogs for dialogues

You should define several dialog user interfaces for your app.

The first dialog supports the creation of a chat room. The creation of a chat room should be provided as a menu item from the action bar. Choosing this option should launch a dialog that prompts the user with a message and allows them to enter the name of the new chat room. Provide a button for creating the chat room, and another button for canceling the dialog. If the user confirms the creation of the chat room, then it should be added to the navigation pane. If you have set things up right, this latter part should happen automatically as a result of a refresh of the cursor for the list of chat rooms, by the loader manager and cursor loader. If the chat room already exists, then display another dialog

that informs the user of this fact, and give them a button for dismissing this warning dialog. Do not use toasts for this notification.

Another dialog should be used to post a message. Again, the command to post a message should be a menu item available from the action bar. When selected, it should display a dialog message with that prompts the user, and accepts the name of the chat room (defaulted to the current chat room) and the text of the message. The user confirms the message to be sent by pressing a SEND button on the dialog. As before, you should also provide a CANCEL button on the dialog. Note that the user can always cancel a dialog by pressing the BACK button, but also providing a CANCEL button can be good human factors.

As demonstrated in class, define a dialog class for each one of these dialogs. Follow these guidelines for each dialog class:

1. The class should inherit from `DialogFragment`.
2. The class should define a static method, called `launch`, that encapsulates the logic for launching the dialog. This dialog should create the dialog fragment, pass it any arguments from the client, and then use the `show()` method to launch the dialog.
3. Define an explicit interface that defines any functionality that the dialog fragment requires from the parent activity. The fragment should bind to the activity with this interface in the `onAttach` callback.
4. In general you can define the dialog user interface either in `onCreateView` or in `onCreateDialog`. To get you used to the more flexible and general way of doing things, you should always create your dialog UI in `onCreateView`. You may still want to define some customization logic in `onCreateDialog`, e.g., ensuring that there is no title bar for the dialog, but the UI should be defined in `onCreateView`.
5. The business logic for updating any databases or invoking background services should be defined in the parent activity, invoked from the dialog through the parent interface, or invoked from dialog callbacks that are defined in the parent activity.

Submitting Your Assignment

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

1. Export your Android Studio project to the file system.
2. 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.
3. In that directory you should provide the Android Studio project for your Android app.
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 flash, mpeg, avi 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*. You can upload this video to the folder you are provided with in Google Drive. The video must be uploaded on time for the assignment

as a whole to be considered as on time. The time of submission is based on the latest of the time you submitted your code and report to Moodle, and the time you uploaded your videos to Google Drive.

Your solution should be uploaded via the Moodle classroom. Your solution should consist of a zip archive with one folder, identified by your name. Within that folder, you should have a single Eclipse project, for the app you have built. You should also provide a report in the root folder, called README.pdf, that contains a report on your solution, as well as videos demonstrating the working of your assignment (unless the videos were uploaded to Google Drive instead).