CS 522—Summer 2024
Mobile Systems and Applications
Assignment Seven—Fragments and Dialogs

In this assignment, you will augment the UI of the chat app from the previous assignments with a multi-pane interface. You will also get some practice with dialogs. Finally, we will incorporate some aspects of Material Design into the user interface.

Part 1: Multi-pane Navigation of Chat Rooms

Up until now, we have been assuming a single chat room for all chat messages. In this assignment, we will generalize this to allow multiple chat rooms. The chat server from the previous assignments already supports multiple chat rooms, since each chat message includes a chat room as a field in the JSON object, 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. We will assume a multi-pane layout for the main activity in the chat server app, as long as the device is in landscape orientation.

1. For a device in landscape orientation, show a user interface that has a navigation pane for chat rooms on the left (say 1/3 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). Specify the fragments for the navigation pane (list of chatrooms) and for the details pane (list of messages for a chatroom) in the layout for the activity (using the `fragment` element). The chat room view should initially be empty.
2. Otherwise, just show a list of the chat rooms. Selecting one of these chat rooms should replace the navigation pane with a details pane to display the messages in that chat room. In other words, the behavior of your app in portrait orientation, when viewing chat messages, is the same as it was in the previous assignment, for the "default" classroom. 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 enable adaptation to orientation changes in display settings, as shown in Assignment Five.

The main change in the data model for your app is that you will need to add an entity type for chat rooms to the database, referenced by the chat room name in message entities:

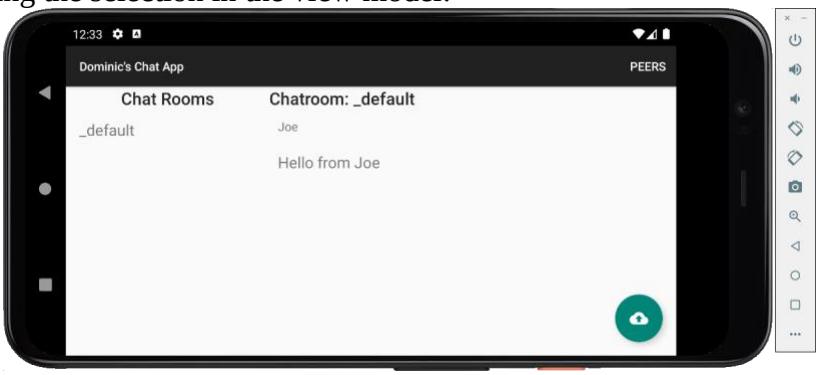
```
@Entity(indices = {@Index(value = {"name"}, unique = true)})
public class Chatroom implements Parcelable {
    @PrimaryKey(autoGenerate = true)
    public long id;

    public String name;
}
```

There is no separate interface for adding chat rooms. For simplicity, new chat room records are added to the server database as messages are added to the database, avoiding duplicates based on the chat room name.

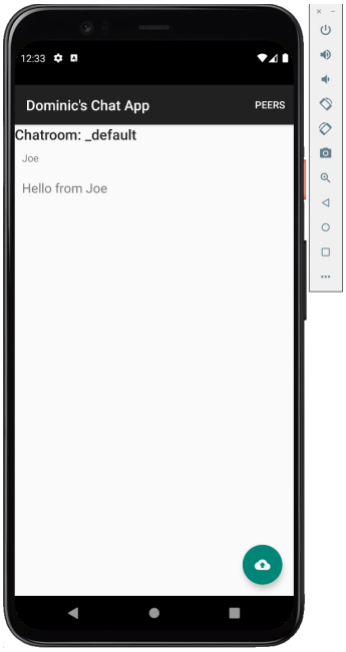
The main activity, `ChatServerActivity`, now has two fragments:

1. `ChatroomsFragment` for displaying a menu of chat rooms.
2. `MessagesFragment` for displaying a list of messages for the current chat room.

Each has their own view model, managed by the parent activity. In addition, the main activity and the messages fragment share a view model for the currently selected chat room. When the user selects a chat room, the chat rooms fragment calls into the activity to change the current chat room messages. If the device is in landscape mode, this just involves setting the selection in the view model:



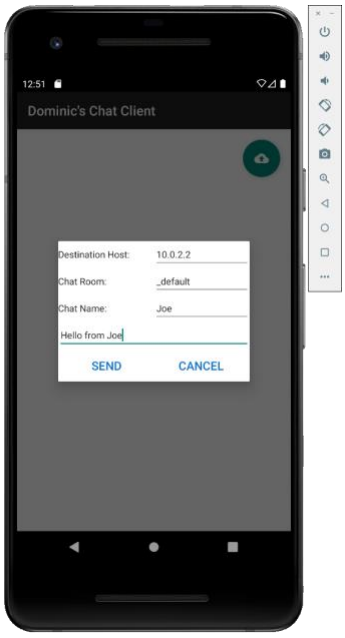
If the device is in portrait mode, the activity must in addition use a fragment transaction to replace the chat rooms fragment with the messages fragment (leaving the chat rooms fragment on the back stack in case the user returns to it with the "back" key):



The floating action button in this activity is used to receive the next message. When pressed, it blocks until a message is received. A toast message will tell the user in which chat room a message was received (It may not be the current chat room).

Part 2: Dialogs for dialogues

We will also incorporate frements in the chat client, in this case adding a dialog (using a `DialogFragment`) to post a message. We add a floating action button to an essentially empty user interface, to launch the dialog for sending a message[1]. When pressed, it should display a dialog message that prompts the user for the text of the message. The user confirms the message to be sent by pressing a SEND button on the dialog. 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.



Follow these guidelines for the 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.

[1] This will make more sense in the next assignment, when we combine these apps, with message receipt done in the background, and message sending initiated by pressing the floating action button.

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. In general, 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. In this case, the actual logic for sending a message is performed by the activity, implementing this interface:

```
public interface IMessageSender {
    public void send(String destinationhost,
                     String chatroom,
                     String chatname,
                     String text);
}
```

   The dialog just presents the interface for getting the details for the message to be 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 projects for your Android apps.
3. Also include the APK files for your projects in the root filder.
4. Include a completed rubric and the videos demonstrating your working assignment.

In addition, record short mpeg videos of a demonstration of your assignment working. Your videos should demonstrate starting the server app, showing a list of chatrooms, navigating to the messages in a chatroom, and using the client app (with a dialog) to add chatrooms and add messages to the chatrooms. You should demonstrate the server app working both the single-pane and dual-pane interfaces, using different orientations (portrait vs landscape. 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*.

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, as well as videos demonstrating your working app.