CS 522—Summer 2024
Mobile Systems and Applications
Assignment Four—Application Architecture I

In this assignment, you will redo Assignment Three, where you saved messages and chat peers in a database. In this assignment, you will use the Room ORM to interface with the SQLite database, and you will use `LiveData` to make your application reactive to updates in the database. This includes querying the database asynchronously, and updating the user interface using the observer pattern when the query is completed. Since content providers use a low-level SQLite API, and the whole point of Room is to lift the level of the API up to the level of entity objects in the domain model, we will use an in-app database without a content provider for this and all future assignments.

For now, we will **not** use a `ViewModel` to persist the UI state between rotation events. Also, we will for simplicity use a `ListView`, and a simplified version of the `ArrayAdapter`, for a list of messages or peers. When we query the database, e.g. for the list of messages, we get back a LiveData object that wraps the query result as a list of message objects. When the query completes in the background, the observer is called to update the UI with a new list of domain objects. However, the array adapter does not provide an operation for updating its backing store, only for adding items to the store. We will therefore work with our own custom array adapter, subclassing the same `BaseAdapter` base class as `ArrayAdapter`, that provides this operation. Another issue with `ArrayAdapter` is that it assumes that there is a single `TextView` in each row of the `ListView` where data is to be shown, and calls the `toString()` operation on the corresponding data object. We will make the UI for the chat object slightly more sophisticated by showing each row in the list of messages as two lines: The sender name on the first line and the message below it. We will define a subclass of our custom array adapter for doing this.

You should follow these guidelines for your implementation.

First, annotate the `Message` and `Peer` entity classes that you defined in the previous assignment, to specify that these are to be stored in a Room-managed database. You should specify that the `sender` field in a `Message` object is a foreign-key reference to the `Peer` record for the sender. Since this foreign key does not reference the primary key of the peer object (We still have a long integer primary key because Android assumes it is there), you are required to define a unique index on the `name` field of the `Peer` object. You are also strongly advised, and required for the purposes of this assignment, to declare a (non-unique) index on the `sender` foreign key in the `Message` object, otherwise checking integrity constraints will generate linear searches of the entire table of messages.

Second, define a subpackage called `databases`. In this class, define the database adapter class `ChatDatabase`, as well as the DAO classes `MessageDao` and `PeerDao`. These classes are largely complete, you just have to add annotations for Room. The `PeerDao` class requires you to implement an "upsert" operation in Room: The operation first

queries the database for a peer, and updates that record if found, otherwise it inserts a new record. These operations must be made part of the same database transaction to avoid race conditions between the query and the subsequent actioni. **You should not change the signatures of the DAO operations.** The bulk query operations always return `LiveData` observable results, but we will allow the insert and update operations to be performed on the main thread **for this assignment only**.

Third, in your main activity you will again display the messages received in a `ListView`, with an array adapter holding the list of messages. However, the messages will be stored in the database, with a query returning a `LiveData` list of messages, and with an observer that reacts to an update in the list of messages by updating the adapter on the listview.

As in the previous assignment, provide another UI (accessible using the options menu from the action bar) for displaying a list of the peers. This UI just lists peers by name in a list view. You should in this assignment load the list of peers from the database, instantiating the `Peer` DAO in the activity for viewing chat peers.

If the user selects one of these peers, then provide in a third UI information about that peer (their currently known GPS coordinates, and the last time that a message was received from that user). Also display a list of messages just from this peer. You should pass the peer entity to this subactivity, as before.

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 apps (**both** client and server).

In addition, record short mpeg videos of your apps working. Make sure that your name appears at the beginning of the videos. 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*. The videos should demonstrate running the app from Android Studio, where the code including the database definition with annotations is clearly visible, exiting the app, then rerunning the app and demonstrating that the saved state from the previous run (messages for the chat app) are still present when the app is restarted. Make sure that you send messages from several peers to the chat server.

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 two Android projects, for the apps you have built. You should also provide videos demonstrating the working of your assignments, the schema generated as a Json file by Room[1], as well as a completed rubric.

---

[1] You can find the generated schema in the `app/schemas` subfolder of the Android Studio project.