420-616-AB Oracle Technologies
Fall 2019

November 11th
Worth 6%

Assignment IV

(Due on Check on Léa)

1 Overview

In this assignment, your Notes app will be modified to store data on the server instead of the local SQLite database. Specifically, the note list, the edit/create feature and the collaborator list will work with server data.

The UI is demonstrated in the video alongside these instructions. Please watch it before continuing on. Below are the requirements and additional resources. Note that not all app features from the previous assignments will be implemented with the server. For example, the undo create/edit note feature from last assignment will no longer work. I will do my best below to list what features might not be working.

2 JSON

The server's REST API sends and receives JSON serialized data. Using the `gson` library , update the model classes `Note` , `User` and `Collaborator` to format and parse the expected server JSON representations. It is unnecessary to implement all the variations of parse and format in each class. The use cases in the unit tests provided should be enough to complete the assignment.

Requirements. For each class, create an instance method `format()` and a class (static) methods `parse(String json)` and `parseArray(String json)`. Implement only those you must to pass all the tests in `NoteServerJsonTest` .

Hints. Since the format and parse methods work with slightly different JSON objects, you will likely need to use the `@Expose` annotation in one of the two operations. Exposed fields are only considered when `excludeFieldsWithoutExposeAnnotation()` is called on the Gson builder.

To serialize bitmap images as 64-bit encoded strings, add the following inner class:

```
private static class BitmapToBase64TypeAdapter implements JsonSerializer<Bitmap>, JsonDeserializer<Bi
    public Bitmap deserialize(JsonElement json, Type typeOfT, JsonDeserializationContext context) thr
        byte[] bytes = Base64.decode(json.getAsString(), Base64.NO_WRAP);
        return BitmapFactory.decodeByteArray(bytes, 0, bytes.length);
    }
    public JsonElement serialize(Bitmap src, Type typeOfSrc, JsonSerializationContext context) {
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        src.compress(Bitmap.CompressFormat.PNG, 100,  outputStream);
        return new JsonPrimitive(Base64.encodeToString(outputStream.toByteArray(), Base64.NO_WRAP));
    }
}
```

and use it when constructing your `GsonBuilder`:

```
GsonBuilder builder = new GsonBuilder();
builder.registerTypeHierarchyAdapter(Bitmap.class, new BitmapToBase64TypeAdapter())
```

3 NotesServer

Setup and run the `NoteServer` included with these instructions. Read and follow the installation instructions in the file `README.md` in the `NoteServer` directory.

4 Application Class

We will store data global to all the activities and fragments of and app in an "application" class.

Add a class `NotesApplication` to your project and configure it to be your app's application class. This class instantiated when your app launches and can be used to store information used across all activities and fragments. Perform the following:

1. Create class `NotesApplication` that extends the class `Application`.
2. Update the file `AndroidManifest.xml` to use your application class:

```
<application
    android:name=".ui.NotesApplication"
    ...
```

3. Override the `onCreate(..)` method of your application class.

Use this class to initialize your application and store information relating to server and current user.

Requirements. The application class must store the following information:

1. Store the UUID of a User entity from the server. Usually this would be obtained through a login process, but for this assignment we will hardcode this value.
2. Store the networking setup of the running `NoteServer`: protocol, host and port.

Hints. You can access your application class in your activities:

```
NotesApplication application = (NotesApplication) getApplication();
```

and in your fragments:

```
NotesApplication application = (NotesApplication) getActivity().getApplication();
```

To help with testing, here are the UUIDs of the users on the server:

```
String ian    = "7bdba0fe-fe95-4b1c-8247-f2479ee6e380";
String usef   = "3faa2495-f0f1-4408-ae24-d482f37caf1c";
String aref   = "6e840afc-5c0a-4679-bcfa-8a210e50ecfc";
String jim    = "97489bce-1c85-4ff2-b457-ba53589d12cc";
String sandy  = "2c77dafe-1545-432f-b5b1-3a0011cf7036";
String nobody = "13cea3c0-4b18-471f-9bee-e9060ac62213";
userUuid = aref;
```

5 Note List

When the user swipes down on the activity, retrieve from the server all the notes that the current user is collaborator on and display them in the `RecyclerView`.

Requirements.

1. When the fragment is first loaded, do not display any notes.
2. Add a `SwipeRefreshLayout` to your fragment.
3. The swipe refresh layout displays the progress only while the retrieval task is running.
4. The notes retrieved are only those from for the current user, whose UUID is stored in the application class (see above).
5. No longer read notes from the sample data or the local SQLite database.
6. Use the classes `HttpRequest`, `HttpResponse` and `HttpRequestTask` from class.

Hints.

- Recall that for development on the emulator, the IP address of the development machine is 10.0.2.2.
- Here is how to add a swipe refresh to your UI: `https://developer.android.com/training/swipe/`. The method `setRefresh(boolean)` is used to turn on/off the progress.
- Each user entity has a special URL for their notes:

  ```
  http://10.0.2.2:9999/user/2c77dafe-1545-432f-b5b1-3a0011cf7036/notes
  ```

  where you should replace server location or UUID as appropriate.

6 Note Create/Update

When a note is created or edited, send the change to the server.

Requirements.

1. An edited note is updated on the server using it's entity URL.
2. A new note is created on the server. An associated collaborator entity is created linking the user to the note. See the `collaborator` repo on the server.
3. No longer create and edit notes in sample data or the local SQLite database.
4. If you have the action-mode working "Reminder" and "Trash" will no longer work and will not be evaluated.
5. Use the classes `HttpRequest`, `HttpResponse` and `HttpRequestTask` from class.

Hints. Without adding a new collaborator entity on the server, a new note will not show up when retrieving a user's notes from the server (in the previous section).

When sending data to the collaborator repo, you need to send the entire URL of the user and note, not just their UUID. See the Unit test for the collaborator entity.

7 Display Collaborator

When the edit fragment loads, retrieve the list of collaborators and populate the `DisplayUsersFragment`
.

Requirements.

1. When the fragment is created, retrieve the list of collaborating users for the note.
2. For each user, retrieve their avatar image from the server on a separate task. This means running multiple tasks in parallel.
3. Update the `DisplayUsersFragment` when each image has been retrieved.

Hints.

- Each note entity has a special URL for their collaborators:

  ```
  http://localhost:9999/note/e187ecea-167c-4feb-a13a-eb34949bb400/collaborators
  ```

  where you should replace server location or UUID as appropriate.
- I have configured the server to omit the avatar image when a user appears in an "embedded" sub-object of the JSON representation. The avatar image only appears when directly retrieving the user with it's entity URL.

8 Server Error

TODO

9 Project Structure

Use the following directory structure for your project's source code, separating the app into model, ui, activities and utilities:

```
<package root>
    ├── model/*
    ├── networking
    │       ├── Either.java
    │       ├── HttpRequest.java
    │       ├── HttpRequestTask.java
    │       ├── HttpResponse.java
    │       ├── OnErrorListener.java
    │       └── OnResponseListener.java
    └── ui
        ├── NotesApplication.java
        ├── editor
        │     ├── NoteEditActivity.java
        │     └── NoteEditFragment.java
        ├── list
        │    ├── NoteListActivity.java
        │    └── NoteListFragment.java
        └── util/*
```

10 Requirements

- Your program should be clear and well commented. It must follow the “420-616 Style Guidelines” (on Léa).
- Create an Android project with minimum SDK 23 - Android 6.0 (Marshmallow) or later.
- The meets all the requirements above.
- Your code passes all unit tests in `NoteServerJsonTest`.
- Construct each URL using the values stored in `NoteApplication`.
- Use the classes `HttpRequest`, `HttpResponse` and `HttpRequestTask` from class.
- Your app must not perform network requests on the UI thread, i.e.: the lines

  ```
  StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
                                      .permitAll().build();
  StrictMode.setThreadPolicy(policy);
  ```

  must not be anywhere in your project.
- You can use the provided Assignment #4 starter.
- Submit using Git by following the instructions (on Léa).