Project 2

Moods - Due 2/16 @ 11:59 pm

For Easy Reading, Go to View -> Disable Print Layout

PLEASE MAKE SURE THAT YOUR GITHUB REPOS ARE PRIVATE

Demo and Design Details

A demo video and design specifications (assets, fonts, etc.) are provided in Canvas. (Designs)

Background

The popular saying of “health is wealth” can be dated back to many years ago and has acted as a cornerstone in advocacy for a more mindful society. This notion of health remains true especially today in the midst of today's pandemic and various unprecedented conditions. To help with coping and addressing these events in our lives, we will be developing the “Moods” app as a mental health tracker for our second project. Being able to track thoughts gives people a good starting place to vent and open up. This app is designed to help people jot down their thoughts, feelings, and moods.

Technical Description

For this project we will be leveraging Fragments and the Jetpack Navigation library to implement a Single Activity application structure. We will also use a Room Database to store data locally in the application. Through both of these technologies we will be able to understand and perform navigation and storage persistence models used in modern android application development.

Note

You might read or hear us use ‘Mood Note” and ‘Note’ interchangeably, but we mean the same thing.

As android is an evolving language, at times you may get alerts from AndroidStudio saying certain portions are deprecated, however please don't try to update such areas since the project may not work as intended if you do.

Key Concepts

- Fragments (Lecture)
- Jetpack Navigation and Safe Args (Demo)
- Room Databases (Demo)

Requirements

Screen 1: List of Notes

- This is the "Home Page" and has a list of notes taken by the user
- Each item in the list contains:
    - Mood
    - Title
    - Notes
- Interactions
    - Clicking a note in the list should send you to Screen 3, where you can view the full contents of the note
    - Create a new note ('+' icon) which will send user to Screen 2, where they can add new notes

Screen 2: Create Note

- Display Note inputs
    - Mood selection
    - Title (editable text)
    - Notes (editable text)
- Interactions
    - Cancel Note Creation ('x' icon)
    - Add Note to Database (check icon) and send user to back to Screen 1

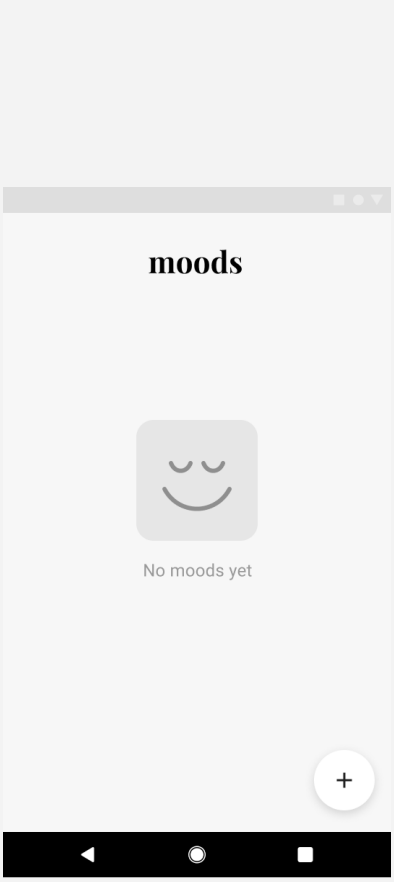
Screen 3: View Note

- Display Note
    - Mood
    - Title
    - Notes
- Interactions
    - Delete Note (trash can icon) and send user to screen 1
    - Edit Note (pencil icon) by sending user to Screen 4

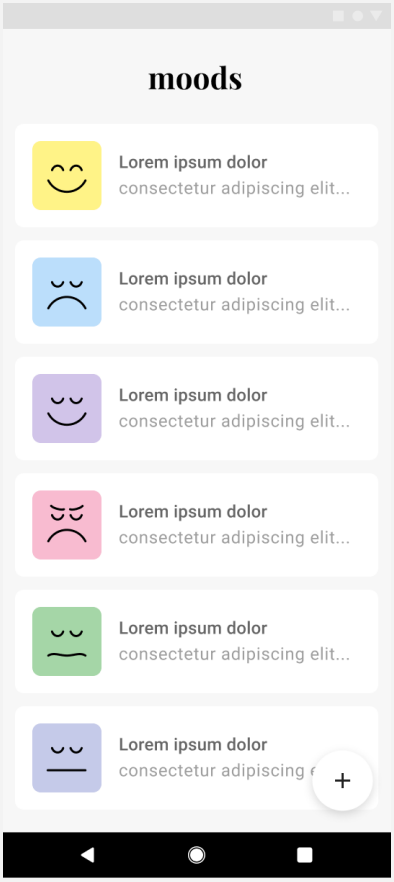
Screen 4: Update Note

- Display Note inputs
    - Mood selection
    - Title (editable text)
    - Notes (editable text)
- Interactions
    - Cancel Note Creation (‘x’ icon) by discarding any changes and sending the user back to Screen 1
    - Add Note to Database (check icon) and send user back to Screen 1

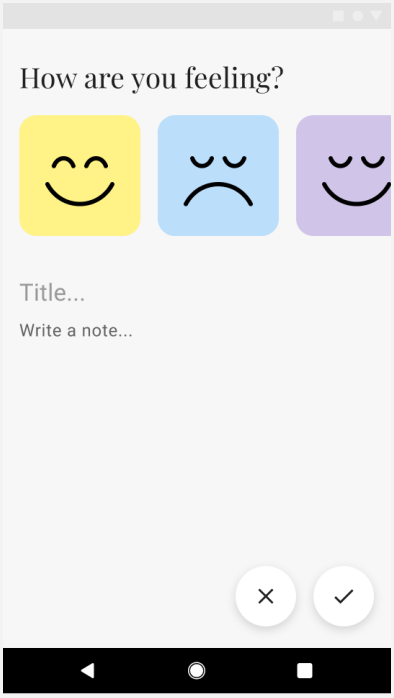
Screens



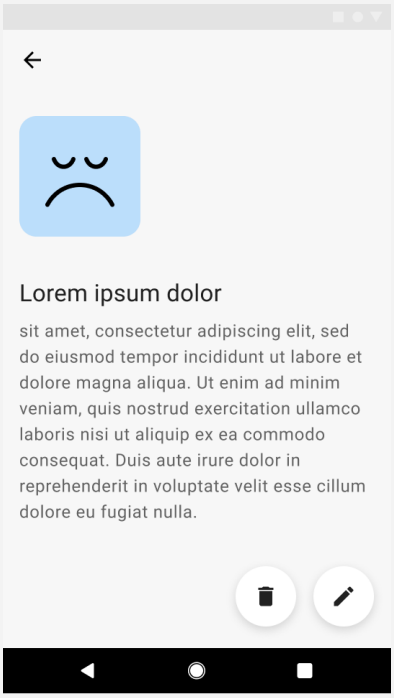
NoteFeedFragment
(Screen 1)
No Notes



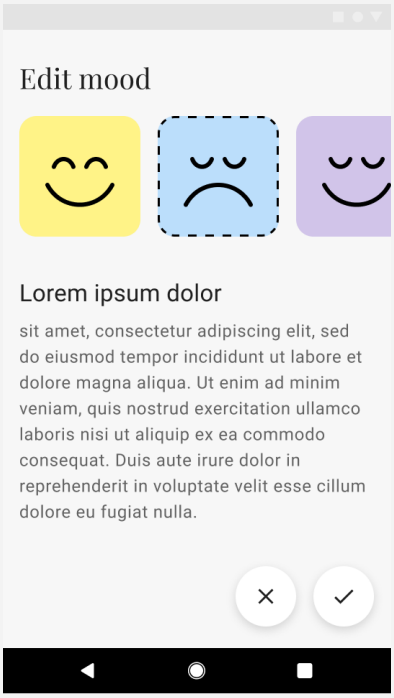
NoteFeedFragment
(Screen 1)
> 0 Notes



CreateNoteFragment
(Screen 2)



ViewNoteFragment
(Screen 3)



UpdateNoteFragment
(Screen 4)

Architecture Breakdown

Given Core Pieces

- Gradle
- Libraries
- Fragments
- General graphic assets
- Base Files

Students need to develop

- Screen Navigation using Jetpack Navigation
- Database operations using Room Database

Given/Expected Files

The project starter code is on GitHub

Project File Structure

- **Bold Black files -** are ones you need to complete the codebase with logic
- **Bold Blue files** - are ones you need to create and implement yourselves
- Any non-formatted files should be left alone

*The files below are listed in no particular order

- ❏ app
  - ❏ java
    - ❏ Com.android.example.moods
      - ❏ data
        - ❏ Note
        - ❏ **NoteDao**
        - ❏ **NoteDatabase**
        - ❏ **NoteRepository**
        - ❏ **NoteViewModel**
      - ❏ **CreateNoteFragment.kt**
      - ❏ MainActivity.kt
      - ❏ **NoteFeedFragment.kt**
      - ❏ **NoteListAdapter**
      - ❏ **UpdateNoteFragment.kt**
      - ❏ **ViewNoteFragment.kt**
  - ❏ res
    - ❏ drawable
      - ❏ -> contains all the vector assets you should need, feel free to add more to your liking. We have provided 2 vector assets for each of the 6 provided Moods: anxious, content, happy, mad, neutral, sad
        - ❏ ic_<mood>_outline_false.xml - this vector asset does not have a border and can be used to signify that a mood is not selected
        - ❏ ic_<mood>_outline_true.xml - this vector asset has a border and can be used to signify that a mood is selected

- ❏ layout
  - ❏ **activity_main.xml**
  - ❏ **fragment_create_note.xml**
  - ❏ **fragment_note_feed.xml**
  - ❏ **fragment_update_note.xml**
  - ❏ **fragment_view_note.xml**
  - ❏ **Notelist_item.xml**
- ❏ navigation
  - ❏ **nav_graph.xml**

Gradle Scripts

- Do not touch anything in here

Phases

Phase 0: Understanding the Codebase

There is code already provided to you, make sure that you can understand and coordinate with the given code. This is done to make your jobs easier so that you can make apps with more functionality. The code base is sprinkled with all the phases.

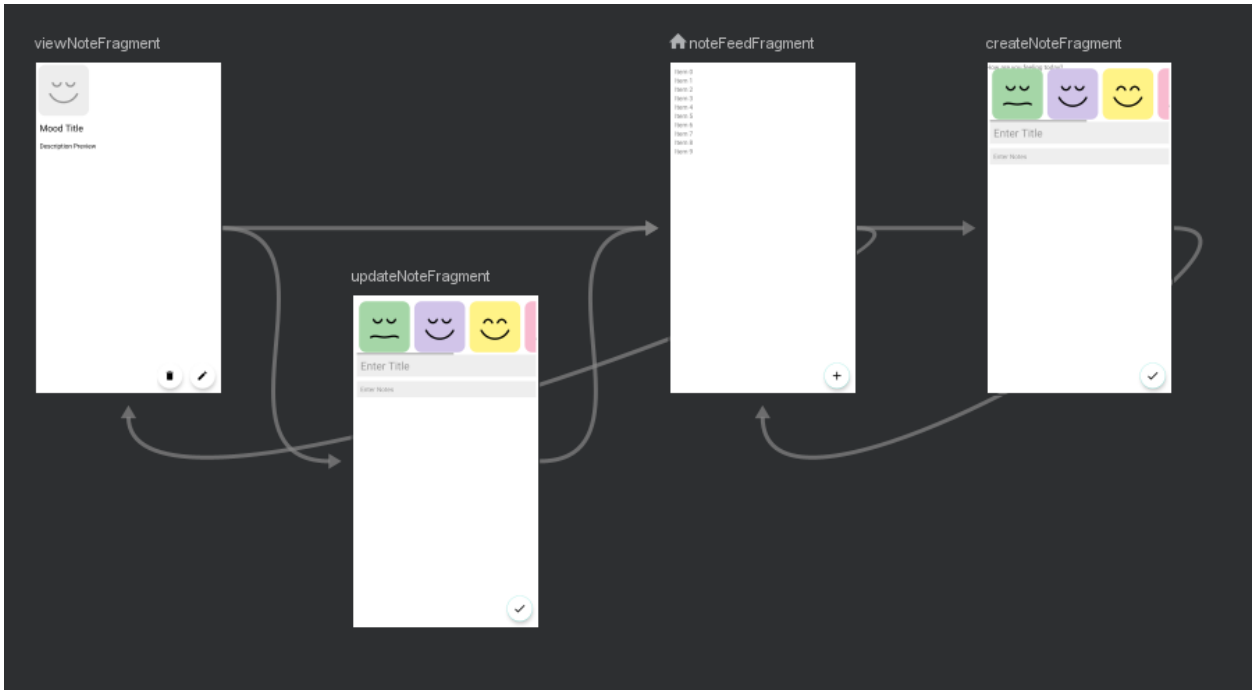
There are some key files to look through, but look through as many as you see fit.

Phase 1: Jetpack Navigation

In this phase we want to have all the connections and screens (fragments) laid out prior to diving into making database operations. We can add our Buttons (or Floating Action Buttons (FABs)) to switch screens respectively.

Phase 1.0: Navigation Graph Setup

Here is an example of how the navigational graph should look. Note the Layout design is not the end result, it is just to give you an idea of which file is which and how connections may look like.



- Create the navigation graph
- Add fragments (use provided fragments to add into the navigation graph)
- Add NavHostFragment to activity_main with navigation graph attached
- Add all actions (arrows)
- Add arguments to destinations (updateNoteFragment and viewNoteFragment), you'll want to pass a Note object to these screens

Phase 1.1: Adding a RecyclerView

In order to successfully create a list for the moods, a RecyclerView must be implemented

- NoteFeedFragment & fragment_note_feed.xml
    - Establish a RecyclerView similar to Project 1, the main difference being that it is now developed inside a fragment instead of an activity.
    - Feel free to add dummy data initially into the RecyclerView list to make sure your code is working before using data from the Room Database
- NoteListAdapter & notelist_item.xml
    - Define views in ViewHolder
    - Update views in the layout with the Note data in onBindViewHolder, just as we did in Project 1
    - Inside the onBindViewHolder you will also want to use the navController to switch to the ViewNote Screen (which will be built out in Phase 1.3)

Phase 1.2: Creating a Note

This is where the basics must be set up for making a Mood Note

- fragment_create_note.xml
    - Add a horizontal scroll view for the mood image selection
    - Make two FABs for confirming and cancelling a Mood Note submission respectively
    - Make two EditText Views for users to provide the title and content respectively for the Mood Note
- CreateNoteFragment.kt
    - Add an OnClickListener to the FAB for navigating to the next fragment
    - Make sure that the FAB connects to insertNoteIntoDB() with the help of setOnClickListener()
    - An on click listener is needed for the selection of the last mood chosen
    - Don't forget to connect the Navigation controller to for switching to the NoteFeedFragment
    - Optional: Perform input validation within the insertNoteIntoDB() function
    - You might find it helpful to have a resetImages() function that sets all the images to be borderless prior to setting the selected image to have a border. It might also be helpful to have a global

Phase 1.3: View a Note

This is where the basics must be set up for viewing a note

- fragment_view_note.xml
    - Have two FAB buttons to edit or delete the Mood Note respectively
        - These FAB buttons should also have their respective icons for editing and deleting
    - Also have places for the image view, title and content
- ViewNoteFragment.kt
    - Make sure to have navigation controller ready for navigating to the NoteFeedFragment
    - Need to set data through grabbing the note from safe args
    - Each action should respond accordingly to the next call

Phase 1.4: Update a Note

This is where the basics must be set up for updating a note

- fragment_update_note.xml

    - Add the horizontal scroll view (should be same as all the others)
    - Make FABs for Mood Note submission and cancellation
    - Also make space for mood title and mood note content
- UpdateNoteFragment.kt
    - In this file you will need to set data on the screen via safe argos, like previously done
        - This includes texts, images views,and other updates data-dependent features
    - Add on click listeners to each of the respective areas

Phase 2: Note Database

When making the Note Database, it is important to understand the relationship between the data generated and how it is used. The interaction can be understood across 4 main areas: Query, Insert, Update, and Delete. Through these operations we will be able to generate, modify and traverse through the various data points. We are providing how a Mood Note should look in the database inside the Note file (this is to streamline the process, you are more than welcome to make any changes).

Phase 2.0: Setting up Database

For setting up the database follow the lecture and demo that explains how it works and how to implement Room Databases

- NoteDao
    - Declare and Define all the Queries necessary to get, update, insert, and delete notes. For getting notes, getting all notes should be sufficient as there aren't any situations that require querying a single note from the database.
- NoteDatabase
    - Here we will instantiate a singleton instance of our database to interact with. To do this we need to define a companion object for the database.
- NoteRepository
    - The repository allows for a clean API for the rest of the app to use for interacting with the database. Here we will want to define all the wrapper methods around our DAO queries.
- NoteViewModel
    - The ViewModel provides data to the UI and survives configuration changes. It acts as a communication center between the Repository and the UI. The

ViewModel is part of the lifecycle library. Here we will want to call the repository functions in a viewModel Scope.

Phase 2.1: Add Database interactions to CreateNoteFragment

- Use the NoteViewModel in order to add a note into the database when the user hits the checkmark. Make sure to add some input validation so the user does not submit bad data. Specifically, ensure that the user selects a mood.

Phase 2.2: Add Database interaction to NoteFeedFragment

- Use the NoteViewModel in order to get all notes from the database. You will need to use an observer to notify that the data has changed to update the RecyclerView list of Mood Notes.

Phase 2.3: Add Database interaction to ViewNoteFragment

- On this screen we do not need the NoteViewModel for display data but rather deleting it. Displaying the Mood Note was done using Safe Args from Jetpack Navigation. When the user clicks delete (trash can icon) we will use NoteViewModel in order to delete the Mood Note from the database.

Phase 2.4: Add Database interaction to UpdateNoteFragment

- On this screen we will be creating the functionality for updating any notes we have created. Use the NoteViewModel when the user clicks the “check-mark” icon to update the entry of the current Mood Note in the database.

Phase 3: UI design

As with any proper app, the user interface and experience are helpful for users to better navigate and understand the app. To help you out, all the required UI assets are already found within the starter code. As a reminder the design details can be found here. Below is a list of the layout files and assets that are required for the app:

- Layout Files:
    - activity_main.xml
    - fragment_create_note.xml
    - fragement_note_feed.xml
    - fragment_update_note.xml
    - fragment_view_note.xml

    - noteslist_item.xml

- Types of Image Files (Drawable Folder):
    - Ic_anxious_outline_false.xml
        - One of the mood images without the blackoutline (denoted by “false”)
    - Ic_anxious_outline_true.xml
        - One of the mood images with the blackoutline (denoted by “true”)
    - Ic_baseline_add_24.xml
        - “+” for adding
    - Ic_baseline_check_24.xml
        - “✓” for confirming
    - Ic_baseline_delete_24.xml
        - “x” for canceling
    - Ic_baseline_edit_24.xml
        - “✎” for editing

Resources

- Canvas Project 2 Page, Lectures, Demos & Links
- Piazza
- Discord
- Designs

Submission

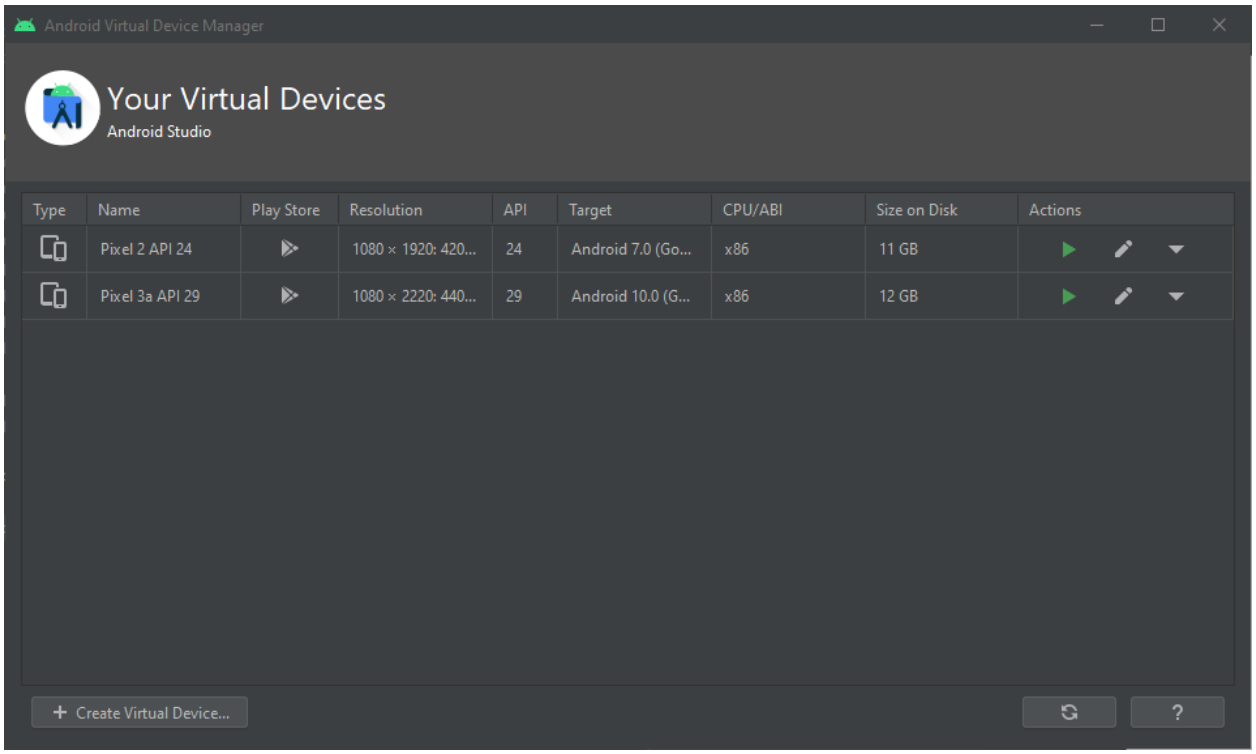
Projects will be submitted through gradescope via github. You can access gradescope through the tabs in Canvas. Please only have one partner submit the project and add the other partner to the gradescope submission so they can see it. Here are some details for submitting the project

- Include a README.md that:
    - Describes the objective of the project in your own words.
    - Includes a link to the video code tour/demo
    - A sample README.md has been added to the starter-code github
- Code Tour/Demo
    - No more than 7 minutes. If you can comprehensively cover the project in less than that, then there is no need to use the full 7 minutes. Concise and complete is better.

- Provide a video recording of your project and codebase describing how you accomplished the end result. Make sure to show the app running and all the possible interactions. Also outline and show comprehension of the sections/phases that are coded
- Suggested method for video submission: Make a Zoom cloud recording and add the link in the README.md. If you choose another method of recording make sure you can link a sharable video in the README.md

Additional Notes

- Make sure your emulator supports the google play store. You can verify this by checking if there is an icon under the play store columbine the Android Virtual Device Manager..



- **Mac Users with M1 Chips ONLY**
  - Mac M1 laptops currently do not have full support for the Android Studio emulator, but Android has since put out a preview emulator
    - https://github.com/741g/android-emulator-m1-preview/releases/tag/0.1
    - Download the android-emulator-m1-preview.dmg file

- The Android Emulator will be a piece of software which must be opened manually before clicking run in Android Studio
- Launching the Web Page may not fully function as expected. You may only see a white screen pop up, but you should still see the correct URL being displayed in the web search bar