PGR-208 Android development report

We have created a nutrition application to help choose the right diet for the user. It has a search function on the main page, where the user can search for recipes with their favorite ingredients, and show the search history if they can't remember what recipes they were searching for earlier. In the settings page, it's possible to filter the recipes on desired daily calorie intake and set the maximum number of search history items.

In our app, we have 3 activities. MainActivity, SettingsActivity, and HistoryActivity.

Main Activity

In the Android manifest it is marked with an intent filter to define that main activity will start on launch where we have one main function that retrieves the data which is called downloadAssetList and it will return the result as a form of list.

Since the structure of the API is an object with plenty of fields we have to access them with the correct methods. The focus is the main entry point which is "hits" in the API as a form of array. We have to cast it as an array and loop through the entire array. We also have a data class called AssetData that holds the fields we are interested in from the API, which is essential for mapping the JSON string from API back into our app as objects. With the JSON object we mapped, we then loop through for each item inside the array we extract the item and cast it as a type of JSON object. It is crucial to understand how the API is structured and what fields we are interested in retrieving from the endpoint result. For instance "recipe" inside "hits" is a form of object, while "url" is only a string, thus we have to cast them explicitly of this type.

Finally we have to map them correctly to our created data-transfer-object which we created (asset data) and for each object we loop through we have to get the name matching the one from API.

The "yield" field from the API is equivalent to serving size for that recipe and for each recipe the servings are fixed. What we tried to do is divide the total calories with yield and pass the value as a single serving to display in a recyclerview, in order to display

the green when setting total calorie is less than recipes and red if recipes calorie is greater than settings.

But we have commented it out as the calorie will only show for single serving only, thus it will not match the purpose of settings where the filters are meant for total daily calories. Most of the recipes are over 2000 calories.

The downloadAssetList runs on a coroutine which is in simple terms an asynchronous function where we can dispatch the function when we want thus we have more control in case of bottlenecks on the same thread.

RecyclerView

In a recycler view which is an alternative way of list view we show the data from API. To use a recycler view we have to use an adapter and a view holder to bind the data into the rows of the recycler view. The purpose of the adapter is like a placeholder for XML components to bind to the view. Inside the adapter we can implement program logic such as passing the data as argument into the custom class adapter. With the help of a view we can now inflate the data that contains the objects from API in order to display.

Kotlin Coroutines

With kotlin coroutines we have a way of writing asynchronous, non blocking code in sequence. We can use it to write code for tasks like networking or file IO that performs long-running tasks without blocking the main thread and end up having an unresponsive UI. With coroutines you can call the coroutines functions just like regular functions by using the ‘suspend’ keyword. So when you call a coroutine function, the main thread can continue running thanks to the suspended execution of the function until the function running in the main thread is finished.

Database

In the MainActivity, we start off by setting up a room database, where we can store search history and a HistoryDAO interface for defining methods for inserting and querying search history data. Defining the database and its entities in the **AppDatabase** class. Representing the search history entry in the database we have the **HistoryData** class.

We populate the **RecyclerView** with the asset data using the **ItemAdapter**. After we define the suspend function **downloadAssetList** that downloads the list of assets from

the API and returns it as an ArrayList with AssetData objects. The downloadAssetList runs in the background using kotlin coroutines. By clicking the Search button, the function is called on the main thread.

We have tried to use SQLITE and Room libraries. Both were easy to insert the data when we are searching in the main activity but when we are displaying in another activity such as search it was challenging because we need to create a recycler view with a custom adapter in order to show the data correctly and we feel this is too much of code repetition which is anti-pattern.

ViewBinding

In the top of the MainActivity class on lines [39, 44, 45], we initiate ViewBinding/data-binding. We also had to activate it in the **build.gradle** file on lines [34, 35, 36]. The choice to use view binding was made because of these reasons. We don't have to call findViewById, the ability to directly access views by their ID, and last but not least reduce the risk of NullPointerExceptions since the fields in a binding object have a non-null guarantee. It's all done using the ActivityHistoryBinding class, which is provided by the data binding library which is based on the XML layout file for the activity.

Settings Activity

Overview The SettingsActivity is used to set several settings for what to display in the recipe search. It has the functionality to set the desired daily calorie intake, maximum search history items, and desired diet type maximum amount. By pressing the save button the user can save its settings and return to the MainActivity displaying a different result in the RecyclerView. Reset the settings or return to the MainActivity without saving.

It binds 3 TextViews that work as labels for the 3 EditTexts that collect user input. 3 buttons, and 2 spinners. They are all binded to the settings_activity.xml via the id.

SharedPreferences

With the use of the SharedPreferences file "MySettings" the settings are stored as integers. SharedPreferences has the ability to save small amounts of data as key-value pairs as you have in the web-browsers localStorage.

Spinners

The 2 spinners have drop-down options that are stored in lists and displayed using array adapters for selecting diet type or meal type.

We did not use pagination for displaying more than the 20 results from the API, but we had to let it go due to the amount of time available. The same goes for animations, which can give the application some more finesse.

It was a really steep learning curve. And it was a lot of new things and concepts to learn, and we felt that it went beyond what to learn in a 7.5 studypoint course.

References

https://developer.android.com/

https://youtube.com

https://stackoverflow.com/

https://www.geeksforgeeks.org/

PGR-208 course videos

PGR-208 Lecture code