**UFOTracker App Development Notes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1. **Revision Summary** | | | | |
| **Rev** | **Description of changes** | **Changes by:** | **Approval by:** | **Date** |
| 1.0 | Initial Draft |  |  | 08/29/2021 |
|  |  |  |  |  |
|  |  |  |  |  |

**Purpose of Document:**

This document is intended as a development note for the development of the UFOTracker Android Application.

This document talks about the what I have done, choice of design for each component of the project, and level of efforts spent for development.

**What is UFOTracker App?**

UFO Tracker is an app that keeps track of U.F.O. (Unidentified Flying Object) sightings.

Features Implemented:

* The application displays a list of UFO sightings.
* Each sighting includes:

○ the date of the sighting

○ the type of the sighting

​ ○ the speed at which the UFO was travelling

* Each sighting belongs in one of two categories according to its type.
* Display each sighting under corresponding category tab.
* The list of sightings should be stored in memory only
* A “+” button will allow adding UFO sightings
* Ability to remove an entry
  + When an entry in the displayed list of items is selected, display a Remove button
  + **Tap on a list item repeatedly will toggle Remove button visibility**
  + When the remove button is clicked, the selected entry will be removed
* Layout implemented to match design spec.

**Code Repo:**

Github repo: <https://github.com/JohnYang524/UFO-Tracker> (Private)

**What I have done:**

* The app is built using MVVM architecture.
* Main UI components
  + *MainActivity –* Main Activity that holds an *AppBarLayout* and a *TabLayout* which is set up using a *ViewPager*.
  + *SightingListFragment -* A fragment displaying a list of Sighting items with a RecyclerView.
  + *TabPagerAdapter* - The *ViewPager* uses *TabPagerAdapter [FragmentPagerAdapter]* to create a fragment [*SightingListFragment*] corresponding to each tab in the TabLayout.
  + *SightingListAdapter* - RecyclerView adapter.
* Data Model
  + *Sighting* – A Sighting object is constructed with
    - *String date*
    - *SightingType type*
    - *String speed*
  + *SightingType* – Enum -> Blob, LAMPSHADE, MYSTERIOUS\_LIGHTS
    - SightingCategory category -> Category the type belongs to
    - int imageId -> Resource id of corresponding image resource
  + *SightingCategory* – Enum -> STRANGE\_FLYERS, MYSTERIOUS\_LIGHTS
    - int tabIndex -> Corresponding tab index in TabLayout
* ViewModel
  + *SightlingListViewModel* - ViewModel for keeping business logic and handling data updates in *SightingListFragment*.
    - List contains all Sighting objects:  
      MutableLiveData<List<Sighting>> mSightingsList;
    - The Fragment is observing a filtered data list: LiveData<List<Sighting>> filteredSightings
    - Whenever there is a data change in mSightingsList, we will also refresh the filtered list. And once Fragment has detected the change, it will initialize RecyclerView/Adapter or notify Adapter of data change.
    - filteredSightings is mapped with tab index for each tab.
* Adding a Sighting
  + Once user clicks the Add button in *MainActivity* layout, *MainActivity* will create test data and notify all attached listeners (*ListRefreshListener*) of this data change.
  + *SightingListFragment* will receive this callback and call ViewModel to refresh data list.
  + Once data list is updated, the LiveData observer will be notified to update the recyclerView data accordingly
* Removing a Sighting
  + *SightingListAdapter* expose a listener interface (*ListItemClickListener*) and requires a listener to be passed in when constructing the *SightingListAdapter* object.
  + The listener will be called back when ViewHolder item(s) is clicked on.
  + Clicking on a list item will toggle visibility of a Remove button
  + Clicking on Remove button will call back to *onItemRemoved*() function implemented by listener.
  + Fragment (listener) will then call remove method in ViewModel to remove corresponding data from the SightingList and trigger UI updates.

**Time Spent:**

* Effort Spent:
  + Github setup: 5min
  + Design: 15min
  + Create resources files and Implement UI components: 45 mins
  + Update UI to match design spec: 20min
  + Data update flow: 25min
  + Testing, debugging, and code cleaning up: 20min
  + Documentation: 15min