Android Application - Image Locator

by Sanika Patil and Vijeta Agrawal

Aim:

The main focus of this report is to present the development life cycle of the Android application, **Image Locator**. It contains the Requirements, Design, Code implementation, Application preview and Bibliography.

Requirements:

- **Functional Requirements:**

1. User should be able to browse through preview of photos stored in the phone's photo library (gallery).
2. The application should allow the user to view location of photos on a map using the latitude and longitude stored in the metadata.
3. If the user selects a photo, another activity with its full image and an option to view its metadata should be displayed.
4. The user must be able to edit and persist the metadata of each photo available in the photo gallery.
5. The application should also enable the user to take pictures using the phone's camera and store them in the phone's photo library with the associated metadata.
6. It should be able to save pictures and metadata to a local database and give the option to search them by their title and description.
7. The application should be able to locate user's location using sensors and the phone's location services.

- **Fundamental Requirements:**

1. It must work on multiple devices with different screen size, processing power and screen resolution.
2. It must work at least for Android>6.0 and/or iOS>9.0
3. It must be efficient, able to cope with a library of thousands of photos
4. The interface must work both in portrait and landscape mode.

Design:

- Overview of the architecture :

Here we have used MVVM architecture because it has some advantages over other architectures. MVVM provides clear separation of the UI and application logic. As well as Unit testing becomes easier as there is no dependency on the view. From developers point of view the code maintainability is also one of the crucial factors in the long run. Thus, because MVVM provides clean separation of concerns, it becomes easier to make changes without distorting the entire code.

If we were to compare MVVM architecture with MVP architecture, MVVM solves one of the biggest issues that developers face while working with MVP, which is of tight coupling. Because there is one to one relationship between the presenter and the activity, making it difficult to maintain this relationship when the number of view increases.

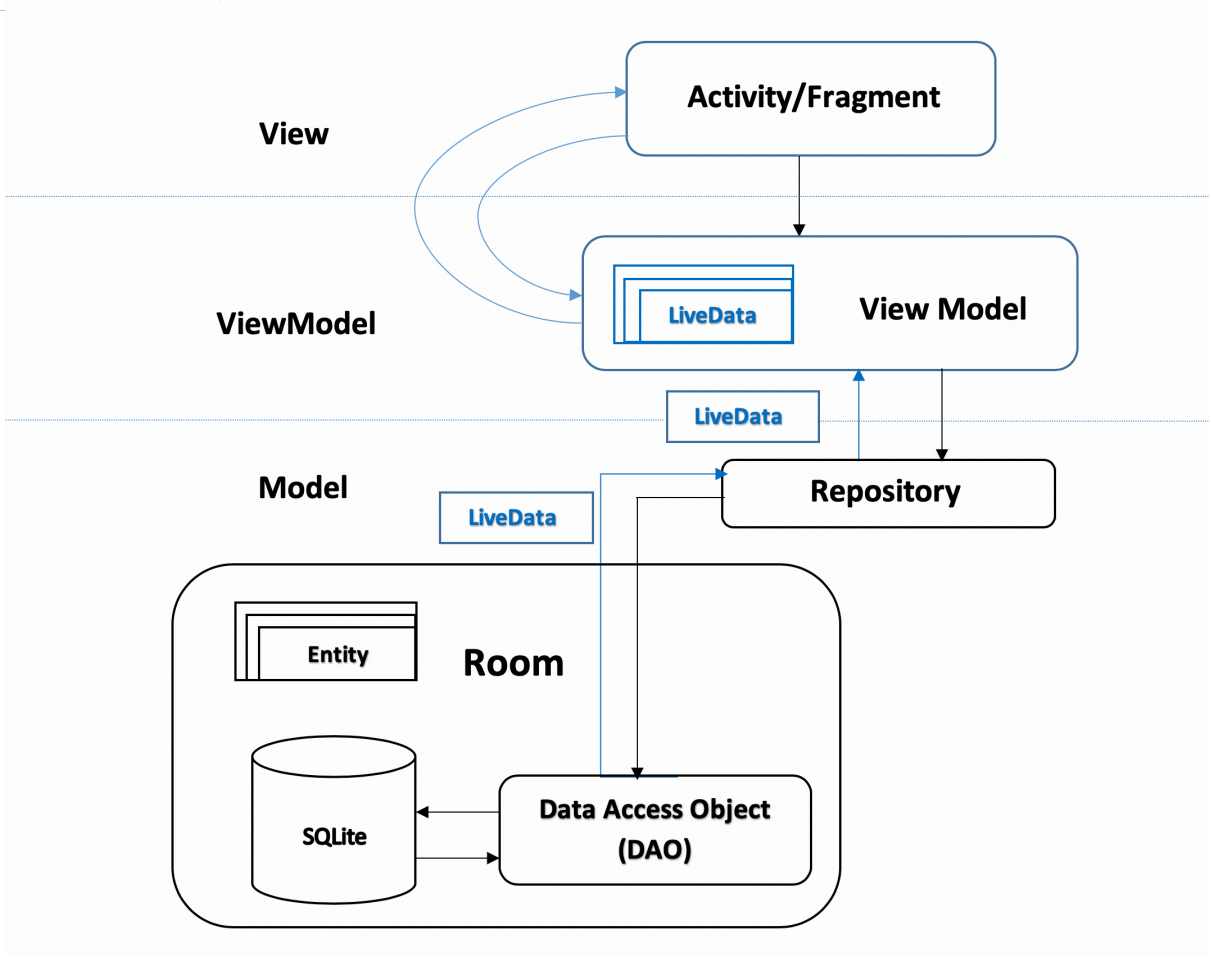


Figure 1

As shown in Figure 1, View and Model are connected via ViewModel. The View continuously observes the changes in ViewModel via LiveData. If any changes occurs in the ViewModel, the View is notified.

- Important design concepts used:

1. **Room:** It provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite. There are 3 major components in Room:
   a. Database: Contains the database holder and serves as the main access point for the underlying connection to your app's persisted, relational data.
   b. Entity: Represents a table within the database.
   c. DAO: Contains the methods used for accessing the database.

2. **AsyncTask:** It enables proper and easy use of the UI thread. This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers**.**

3. **RecyclerView.Adapter:** RecyclerView includes a new kind of adapter. It's a similar approach to the Base Adapter but with some peculiarities, such as a required ViewHolder. We will have to override two main methods: one to inflate the view and its view holder, and another one to bind data to the view.

Code Implementation:

Our code is divided into three major groups:

1. **Model:**
   - It contains a java class named **Image**. It contains all the required attributes defined for the images that we are dealing within our application. For example, title, description Primary key, the image location etc. Also we have implemented the getter/setter methods for these attributes.
   - **database:** It contains the interface ImageDao, in which the queries required to build the local database, images.db are given. This interface is implemented in ImageDatabase using Room which provides an abstraction layer over SQLite. It gets the images from the gallery one by one and put it in the database using *insertSingleImageOnDB* method.

2. **ViewModel and View:**
   - **ViewModel – BaseViewModel and View – BaseActivity:**
     BaseActivity class is created for data binding of the activities. It extends the *AppCompatActivity()* class. It requests for permission for storage access using *onRequestStorage()* method. Then it binds the data to the database. After that we override the *onRequestPermissionResult()* method for storage request and checks is the permission is granted through the package manager. We override another method *onActivityResult()* to get the result form the activity.
     BaseViewModel class extends ViewModel class in which an abstract subclass Factory extends the ViewModelProvider.NewInstanceFactory class. It is used to get the class instance.

   - **ViewModel – ImageViewModel and View – ImageActivity:**
     ImageViewModel extends the observable class and observes for any change in the ImageActivity view. Change in the image list overrides the observe class by extracting the new image list using *getImageList()* method.
     On clicking the Camera button it checks for camera permissions using is *isCameraPermissionGranted()* method. While, on clicking the Map button, it calls the ImageMapActivity view.
     Image Search functionality is implemented using *SearchDbAsync()* to search the image via imageDao. It calls the *getFilterImages()* method to give the filtered images according to the search applied.

   - **ViewModel – ImageDescriptionViewModel and View – DecsriptionActivity:**
     ImageDescriptionViewModel gets the Mutable Live data attributes of the image by implementing the interface idiscription. This includes attributes of the image like url, title and description. When the Save button is clicked on the DecsriptionActivity view, the value of the imageurl is set in the ImageDescriptionViewModel and thus it gets saved in the database.

   - **ViewModel – ImageDetailViewModel and View – ImageDetailActivity:**
     ImageDetailActivity extends AppCompatActivity and implements IDiscription, OnMapReadyCallback ,GoogleMap.OnMarkerClickListener. imageDetailActivityBinding is an instance used to bind the data of this activity to ImageDetailViewModel. It takes the meta data of the image using *getExtraFromIntent()* method and then sets the value for title, description and the image location on the map fragment.
     After getting all the required information about the image, it changes the layout using *changeLayout()* method so that the image information sidebar can be displayed along with the image.

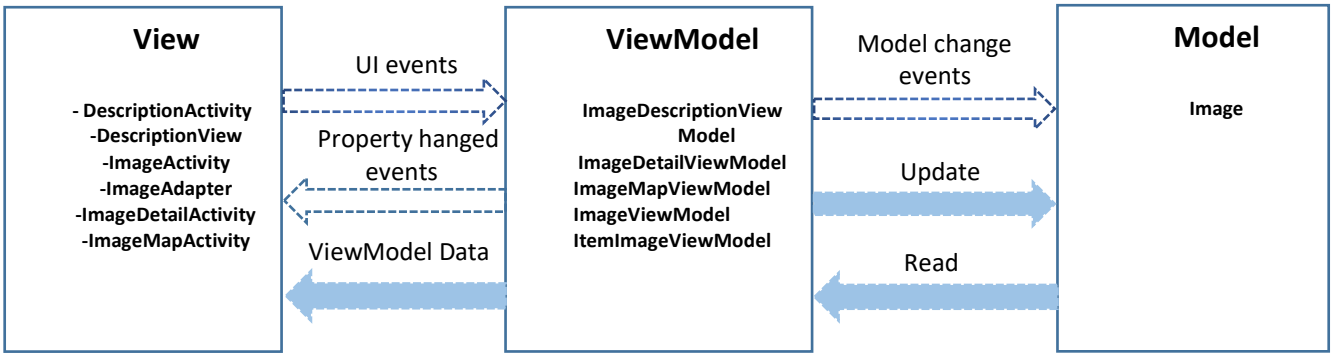
- **ViewModel – ImageMapViewModel and View – ImageMapActivity:**
  ImageMapActivity extends AppCompatActivity and implements IDiscription, OnMapReadyCallback ,GoogleMap.OnMarkerClickListener. imageMapActivityBinding is an instance used to bind the data of this activity to ImageMapViewModel. Using getSharedPreferences it gets the current location of the user from the map.
  It has also used the recycler view to show the list of images at the bottom of the screen using *ImageAdapter().* ImageMapViewModel observes the image list and if the image list is changed, it sets the peopleAdapter to the new image list by using *setImageList().*

  If the user wants to go to the google maps application to see the location of the image, then it ask for the permission which is implemented in *onMapReady().*

  ImageAdapter extends RecyclerView.Adapter and implements Filterable. It overrides *onCreateViewHolder()* method and returns *ImageAdapterViewHolder().* In onBindViewHolder() method, it binds the image list to the holder. ImageAdapterViewHolder() extends RecyclerView.ViewHolder and binds the image to the itemImageViewModel.

3. **Utilities:** It contains the supporting classes which are used in various places during the coding of this application. It contains the following,
   a. **Constants:** It defines the various fixed constants and strings that are used in the application code.
   b. **CustomDialog:** Various methods related to dialogs that are used during the application are implemented here. The methods are *showDialog()* and *dismissDialog().*
   c. **FileUtilities:** File utilities describes the methods to get authority related to the various image storage locations in the phone.
   d. **LocationClass:** It implements LocationListener and checks for the GPS and Network services.
   e. **ToastUtilities:** It implements various toast messages that are to be displayed on the screen during success, failure. It also implements *toast()* method in which customized toast messages can be passed as an argument.

4. **Interfaces: It contains Idiscription and ILocation interfaces.**
   a. **Idiscription interface:** It contains undefined methods like *showMessage(). showMessageWithIcon()* and *OnCLickListener()* which can be implemented by any class that implements this interface.
   b. **ILocation interface:** It is the interface that contains an unimplemented method, (onLocationUpdate) for a Location update.



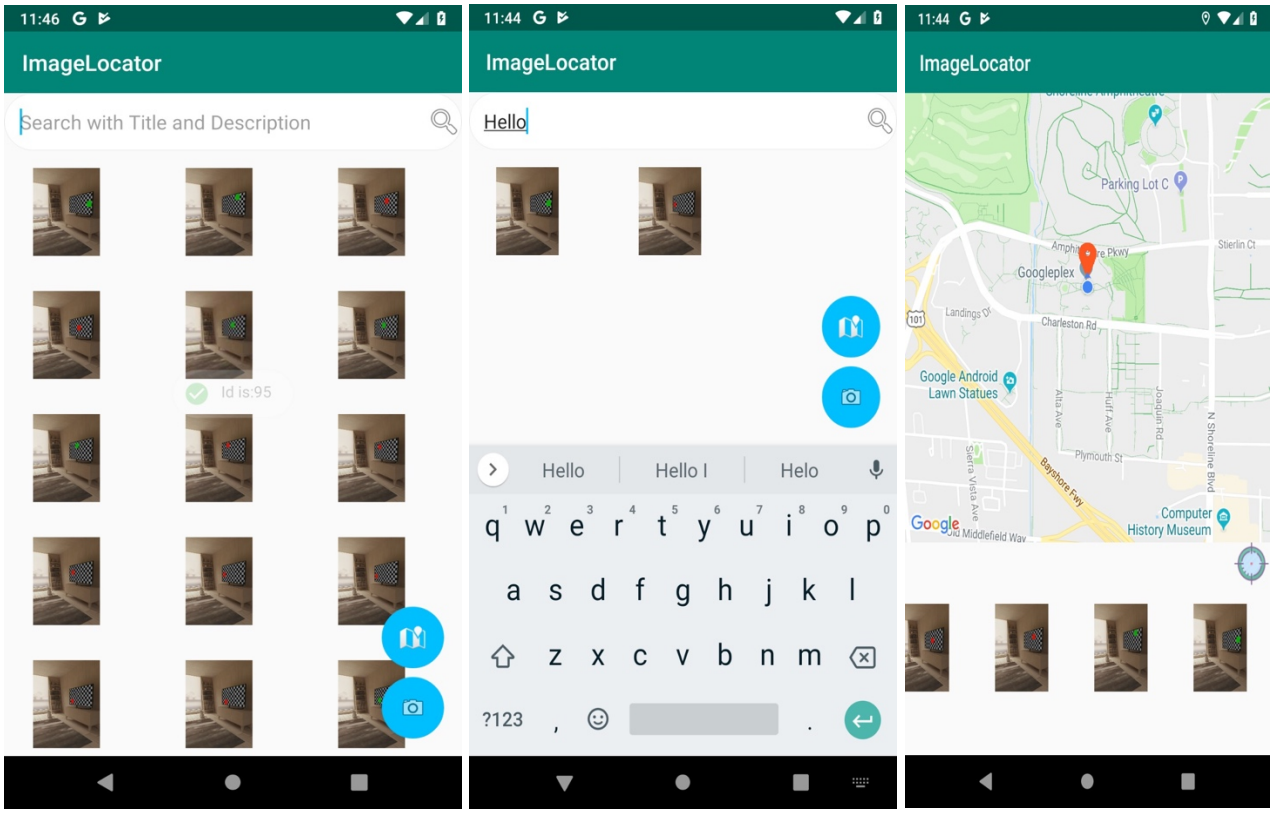
Division of the work:

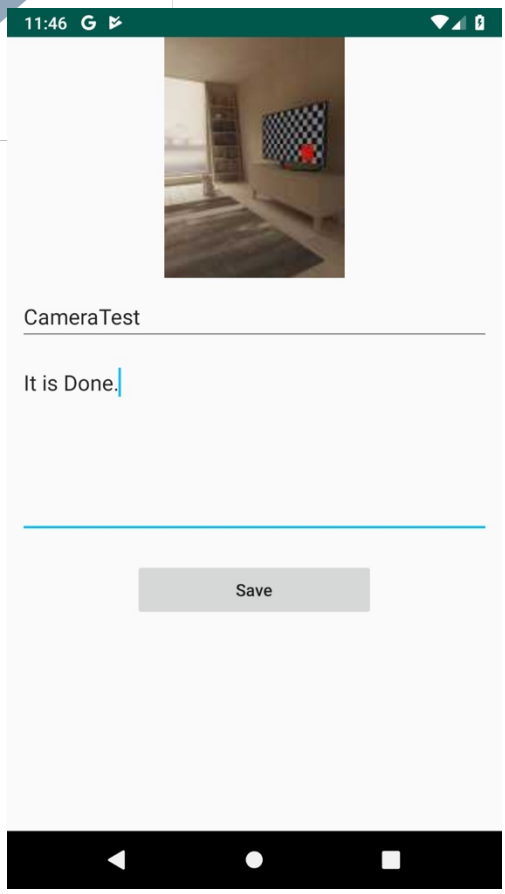
Fixing the layout and the basic foundation for the application is discussed and done by both.
Then the activities were divided as follows,
Map activity and database is implemented by Vijeta Agrawal.
Image Activity and search functionality is implemented by Sanika Patil.
Rest we have merged and worked on the code and report together.
We want the marks to be equally given for both of us.

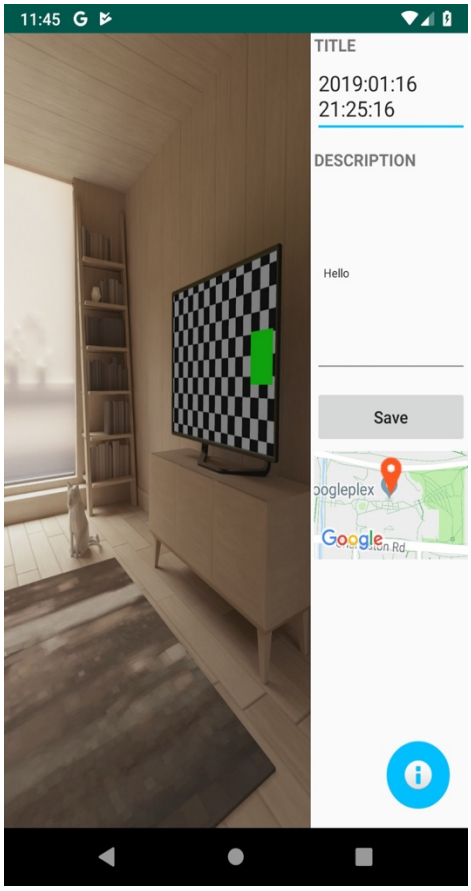
Details of libraries used:

a. retrofitDependencies.gson
b. 'android.arch.persistence.room:runtime:1.0.0'
c. ‘com.google.android.gms:play-services-maps:16.0.0'
d. ‘com.google.android.gms:play-services-location:16.0.0'

Screenshots of Application:







Bibliography:

https://developer.android.com/.