Image Chooser using MVVM and Coroutines

Rahul Yadav[1], Ruchi Rautela[2]

[1]*Student, Vivekanand Education Society's Institute of Technology, Mumbai*
[2]*Assistant Professor, Dept. of MCA, Vivekanand Education Society's Institute of Technology, Mumbai*

Abstract - *Image choosing is very common behaviour in android app development and there are various ways to do this but in very clean and performance efficient. To solve this problem, the model is developed for image choosing using android's own Model View ViewModel(MVVM) design pattern and coroutines for more efficient and better performance. This model will show all the images present in the user's android phone and show it in grid and give the image's physical address where it is stored in the android phone. This model is done with the help of coroutines and MVVM and written in Kotlin programming language.*

Key Words: Model View ViewModel(MVVM), Coroutines, Image Chooser, Kotlin.

1. INTRODUCTION

While developing android apps it is a very common use case where users need to upload images to the server with their android phone. And there are different-different ways by which you can tackle this problem but in most cases what happens is the user has to go out of the app to select the images or the app freezes for a while until the user chooses which is memory consuming. There is another way to do this where you can load data on a different thread and show it in the app only with help of MVVM and Coroutines.

So in this model with the help of MVVM we developed a very nice and understandable model and with help of coroutines which will help to boost the performance.

2. Technology Used

Kotlin is nowadays an official programming language for Android Development by Google. Before Kotlin, Java was the official language for android development but due its concise and ability to write more code in less lines it's now the official programming language for android development and now it also has the support of Coroutines which is very useful in asynchronous programming. I have used libraries like LiveData, ViewModel, Coroutines,Material Design and Glide.

Fetching all the images from the user's android phone is very essential for showing it in the grid layout to the user. So to fetch the data from the android phone I have used the MediaStore class of android SDK. And arranged it in descending order. And for better performance I have used Coroutines which will help us to fetch data on another thread rather than the main thread on which app is running. Because if I fetch data on the main UI thread then it will freeze the UI which is not a good scenario where users can not use the app.

LiveData is a library provided by Google which will observe the changes on the variable. It is very helpful here because now we don't have to look for the images, because of livedata we have to just observe the variable where we are storing all the images address and it will automatically get notified when anything changes in that variable. So by this way we will put all the images which we get using Coroutines on another thread and put it in an variable and we will observe that variable where we want to show the images.

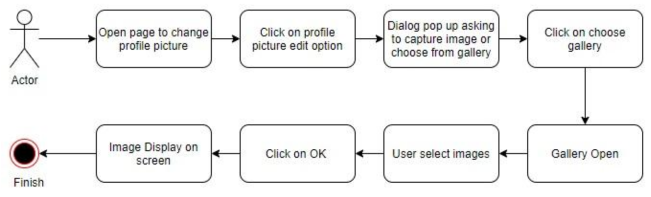
Now there is one more concern of area in the android development is configuration changes or screen orientation. When any configuration changes happen then Android recreates its activity. So to tackle this kind of situation we will use the android ViewModel library. What ViewModel does is it retains the configuration changes in the android app. So to leverage this functionality we will put aur LiveData variable in ViewModel. If we don't put LiveData in ViewModel then it will load data every time anyone rotates the screen of the app where every time it will fetch the image from the android phone.

For loading images on the UI screen we will use RecyclerView. Now to load images into the recyclerview from their image location we have used Glide library. Glide is a very fast and easy library to load the images.

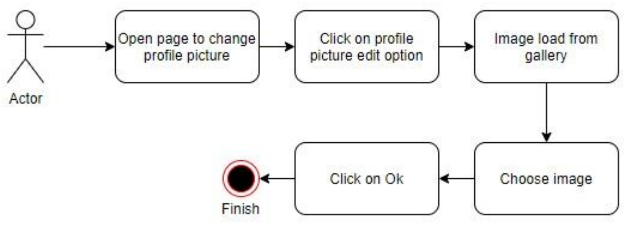
To complete this model we have used the Model View ViewModel(MVVM) architecture pattern. Where in Model we basically store the image address, in ViewModel we load the images and in View we show the images on the UI screen. And when someone selects any image then with the help of LiveData and ViewModel it returns the image address.

3. Proposed Solution

In other models what users generally do is in order to select the image they go out of the app in gallery or any other third party app to select the images by doing this user lose time over there and there will also be more click events which degrade the User Experience.


Actor
Open page to change profile picture
Click on profile picture edit option
Dialog pop up asking to capture image or choose from gallery
Click on choose gallery
Gallery Open
User select images
Click on OK
Image Display on screen
Finish

With the use of our model images can be selected quickly and with great experience. Users can also select multiple images. And it will reduce the click events and improve the User Experience.


Actor
Open page to change profile picture
Click on profile picture edit option
Image load from gallery
Choose image
Click on Ok
Finish

4. Implementation Elaboration

This model uses MediaStore class from the Android SDk to fetch all the images from the android phone. Then we used Coroutines to load images on the different thread which will not disturb the main UI thread and it also improves the performance of the app. And LiveData to load data on the UI when it arrives.

By using the ViewModel approach we make sure the app did not lose the data on screen rotation. And then we use RecyclerView to show the image in the grid format. And to load images into the UI we have used Glide.

4.1 Steps in Technique

Model goes through various phases like Coroutines, LiveData, ViewModel, RecyclerView, Glide.

Consider an Example:- Users want to upload 2-3 images from their android phone to the server.
Then the process goes like as below:-

4.1.1 Load Images

Here we will first ask for read storage permission to the user. Once a user allows that permission then with the help of MediaStore class load all the image addresses. Since the user's phone can have many images which will take more time to fetch all those images. And if we load all the images on the main thread then it will take more time and freeze the app so it is a better option to execute this operation on another thread rather than main thread by doing this we will maintain the app performance.

4.1.2 Design Model View ViewModel pattern

Now we have fetched all the image addresses now the next step is wrap all the data in LiveData. We will use live data because with the help of live data it will be very easy to get the image addresses. To do that we have to just load all the images into the live data variable and observe that variable into the UI. So when anything changes in the live data variable then it will notify us automatically.

Now to complete the MVVM architecture we have to create a ViewModel class and define that livedata variable in that class only so that if any configuration changes occur then the data won't get fetched every time.

4.1.3 Show Images

So we have now fetched all the image addresses in the livedata variable. Now it is the time to observe that live data in the activity. To show images in the grid view we will use RecyclerView in the layout. And to load images from image addresses we will use Glide.

4.1.4 Return Image Address

Now when the user clicks on the image then with the help of live data we will notify the user with the selected image address.

4.2 Evaluation

With the help of this model we reduce the click event by almost 50% and also reduce the time consumption by 40 percent.

For some cases where there are a lot of images in the android phone then the time consumption will be around 20 percent.

5. Conclusions

With the use of this model, we have produced a good result for Image chooser, We can use it in the gallery application also.

With the help of this model, the Image Chooser model system has been improved to best standards still there is a gap for improvement.

For time being we just have worked on the reducing click event, there can be future expansion.

REFERENCES

[1] https://kotlinlang.org/docs/reference/
[2] https://developer.android.com/guide