CMPS 312 Mobile Application Development
Lab 8 – Model-View-ViewModel (MVVM) Architecture

Objective

In this Lab, you will **continue building the Banking App** following MVVM Architecture. In particular, you will practice using ViewModel and MutableState to manage the App state.

Part A: Store Data in ViewModel Code Lab

Complete the following code lab , which helps you understand the Android app architecture and how to use View Models.

Part B: Implement the Transfer List and Transfer Details

In part B, your task is to implement the Transfer List and Transfer Details use cases as shown in Figure 1. When the user selects from the ***Transfers*** menu option, the app should navigate to the *Transfers Screen*. Then, when the user clicks a transfer from the list then the app should navigate to the *Transfer Details Screen* as shown in Figure 1.

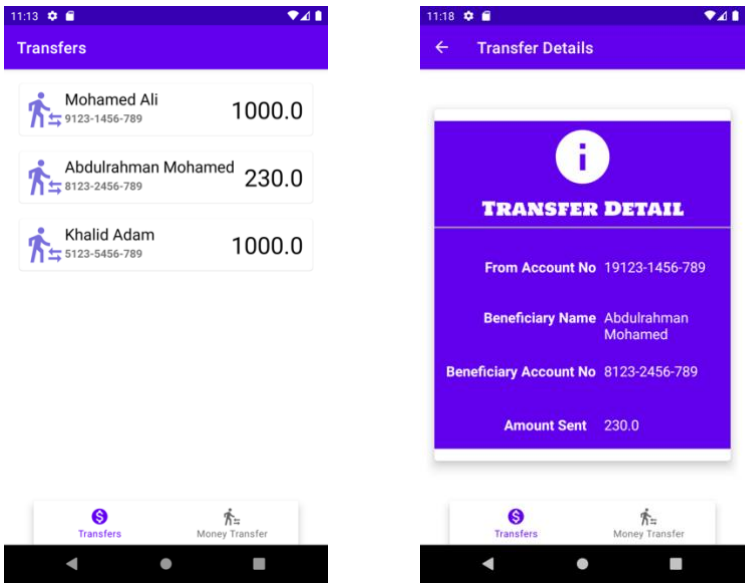

Figure 1. Transfers and Transfer Details use cases

1. Sync the Lab GitHub repo and copy the **Lab8-MVVM** folder into your repository.
2. Open the project ***Banking App*** in Android Studio. The project has the following folders and files:
    - **view**: contains app screens and navigation.
    - **model:** has Account, Transfer and Beneficiary entities.
    - **repository:** implements reading data from accounts.json, transfers.json and beneficiaries.json.
    - **drawable:** images needed for this app.
    - **assets:** contains the json files

3. Create a new kotlin file named `BankingViewModel` inside the `viewmodel` package.

```
class BankingViewModel(appContext: Application) :
AndroidViewModel(appContext) { ... }
```

The `BankingViewModel` should extend `AndroidViewModel` to get access to the `appContext` to be able to read json files from the assets folder.

4. Implement BankingViewModel class to hold the list of transfers.

Inside the class declare a `transfers` mutable state list, that will hold all the transfers. Any change that happens to this list will trigger UI recomposition

```
val transfers = mutableStateListOf<Transfer>( ... )
```

5. Create the `TransferList` Composable and display the list of transfers in a Lazy Column.

The `TransferList` should get an instance of the BankingViewModel to access the `transfers` list.

6. Implement the navigation of the `TransferList` Screen.

Test your implementation, the app should display the screen shown in Figure 2.

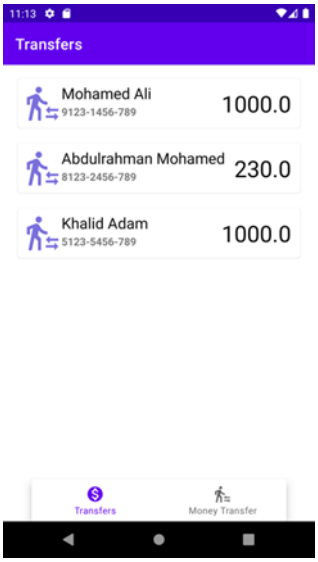

Figure 2. Transfer List screen

7. Add a click listener to `TransferList`. Whenever, the user clicks on a transfer, the app should navigate them to the Transfer Details Screen and pass the selected `transferId` as a parameter.
8. Implement the Transfer Details Screen:

- Get an instance of the BankingViewModel to getTransfer by `transferId`.
- Display the transfer details.

Part C : Implement Add Transfer

In part C your task is to implement the *money transfer* use case. As shown in Figure 3, first the user selects the From Account and specifies the Transfer Amount. Then the user selects a beneficiary. Upon confirmation the transfer is added to the transfers list.

1. In the FundTransfer screen get add the list of accounts, display them in the from account dropdown. When next is clicked (a) assign the From Account and the Amount to bankingViewModel.newTransfer (b) navigate to the Beneficiary screen.
2. In the Beneficiary screen, display the list of beneficiaries to allow the user to select one of them. When next is clicked (a) assign the selected beneficiary details to bankingViewModel.newTransfer (b) navigate to the Confirmation screen.
3. In the file Confirmation screen display the transfer details and allow the user to confirm the transfer.
4. Finally, when confirm is clicked, add the new transfer object to the list of transfers then navigate to the Transfers List (tip: use popUpTo navigation options to clear the backstack up to and including the Transfers List).

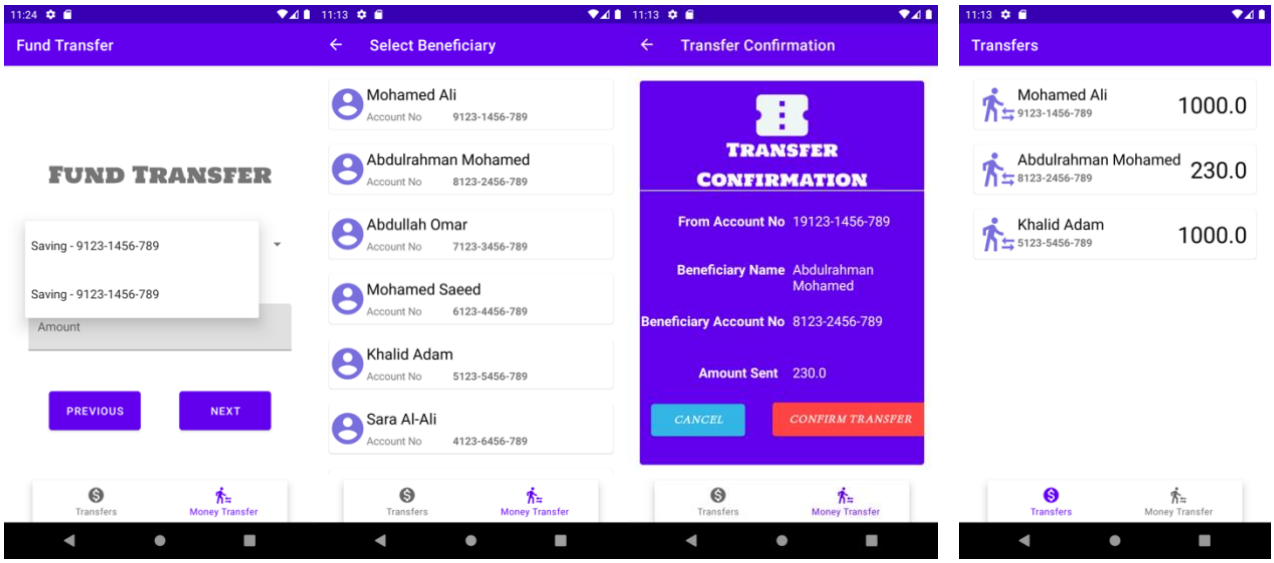

Figure 3. Money Transfer use case