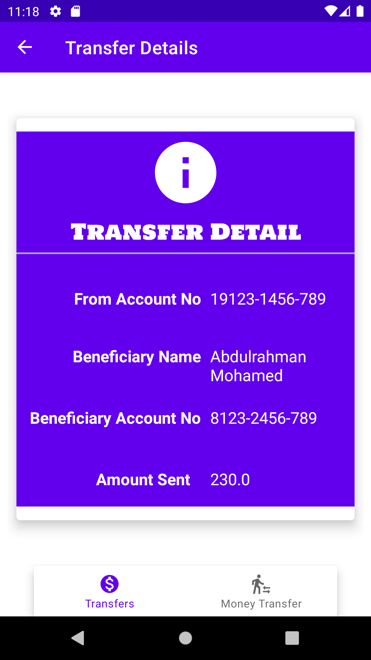
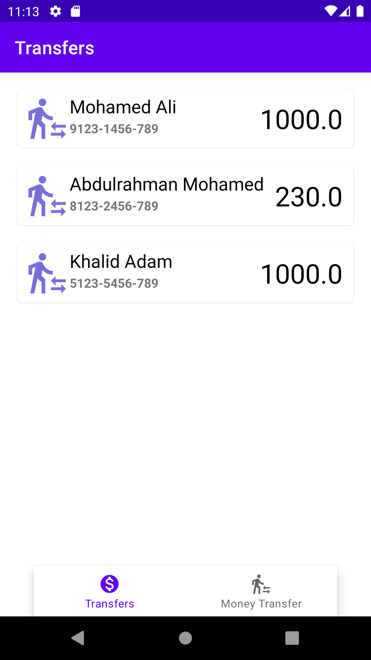
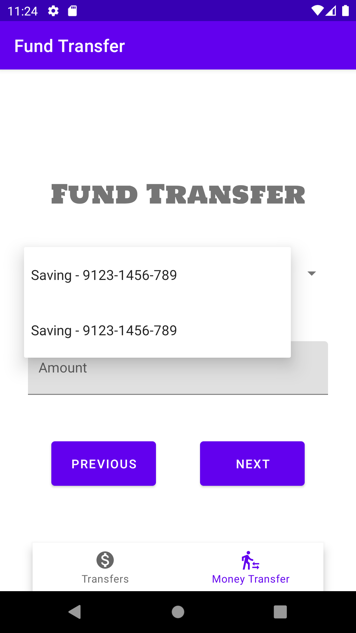
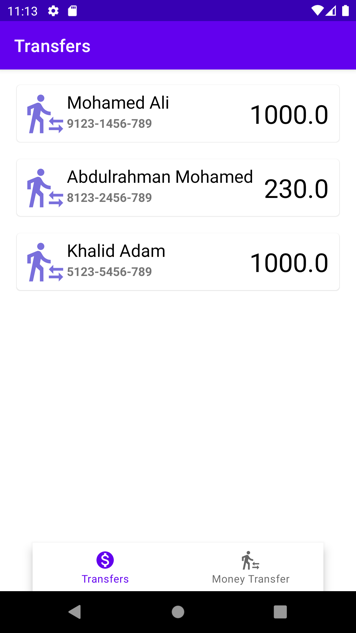
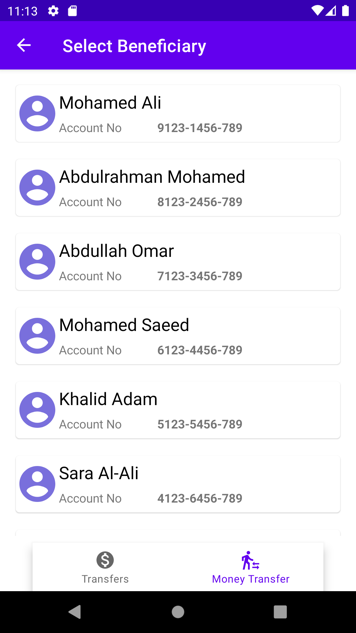
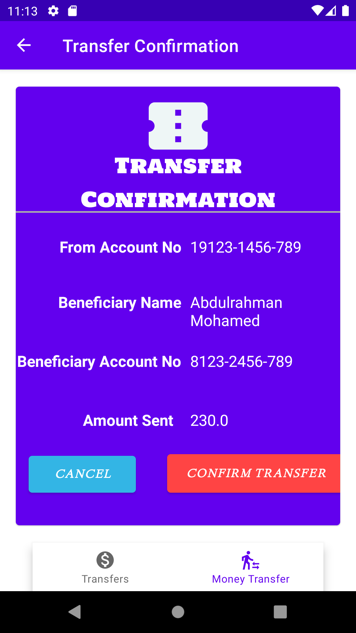
**­­CMPS 312 Mobile App Development**

**Lab 9 – Web API with Coroutines and Retrofit**

**Objective**

In this Lab, you will **continue building the Banking App** and make the app communicate with Web API. You will be using the Ktor library in conjunction with coroutines to get, add, update, and delete transfers and beneficiaries.

In this Lab, you will practice interacting with Web API using asynchronous suspend functions, coroutines and Ktor library.



**Preparation**

1. Sync the Lab GitHub repo and copy the **Lab 8-Web API with Coroutines** folder into your repository.
2. Download postman from <https://www.postman.com/downloads/> and test the following Banking Service Web API available at <https://cmps312banking.herokuapp.com>



A picture containing graphical user interface

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**PART A: Coroutines warm-up app**

Open **Coroutines*ExampleApp*** project Android Studio. The ***Banking App*** project has the complete implementation of **Lab7-BankingApp** with minor modifications such as swipe to delete and new properties added to the Account class such as cid (i.e., Customer id).

To warm up on basic coroutines, we will implement the following:

1. **Creating a coroutine**
2. **Cancelling a coroutine(cooperative cancellaton) after performing a task partially.**
3. **Cancelling a coroutine with timeout exception, that is the coroutine gets cancelled if it fails to get exceuted in a certain timebound.**
4. **Performing a sequential execution of suspend functions.**
5. **Performing a concurrent(parallel) execution of suspend functions.**

**PART B: Implementing the Service APIs**

Open the ***Banking App*** project on Android Studio. This project has the complete implementation of **Lab7-BankingApp** with some minor modifications such as swipe to delete and new properties added to the Account class such as cid (i.e., Customer id).

Your task is to interact with the remote Banking Service Web API to read/write data from/to the remote service. You will be using Ktor with coroutines to achieve this.

1. Add the following necessary dependencies for retrofit and coroutines in your graidle app module.

//*For Ktor Client*

*def ktor\_version = "1.6.4"*

*implementation "io.ktor:ktor-client-android:$ktor\_version"*

*implementation "io.ktor:ktor-client-serialization:$ktor\_version"*

1. Add the internet permission inside your **AndroidManifest.xml** file, or your app will not be allowed to access the network.

<**uses-permission android:name="android.permission.INTERNET"**/>

1. Inside the web**api** package, create an **interface** called **BankService**

List all the interfaces methods that allow the app to communicate with Banking Service API available at <https://cmps312banking.herokuapp.com>

Create a QUBankService class under the webapi package that implements all the methods that allow the app to communicate with the API.

1. Declare the BASE\_URL of the web API
2. Create a HTTP client which helps to parse (to and from) the json when sending and receiving data from the web API

For example: the following getTransfer() method allows sending a get request to Url <https://cmps312banking.herokuapp.com/api/accounts/100101> and returns the list of transfers for customer 10001.

override suspend fun getTransfers(cid: Int): List<Transfer> {  
 val url = "$BASE\_URL/transfers/$cid"  
 *println*(url)  
 return client.get(url)  
}

Now implement the remaining six methods.

**[getTransfers , addTransfer , deleteTransfer , getBeneficiaries**

**, updateBeneficiary , deleteBeneficiary**]

**PART C: Linking the App View Models with the Repository**

In PART C, your task is to replace the **old repository** with the **new repository** that uses the retrofit library.

1. Modify the **TransferViewModel’s** accounts, **\_transfers** to read the transfer from the Banking Service API. You should use the **Ktor instance** that you created inside the QUBankService data/repository/BankRepository object.
2. Do the same for the **BeneficiaryViewModel** and fetch the beneficiaries from the Web API instead of the assets folder. You should not change anything else in the app, and it should work as before. **This is the fruit of MVVM!!!** 👍👌

