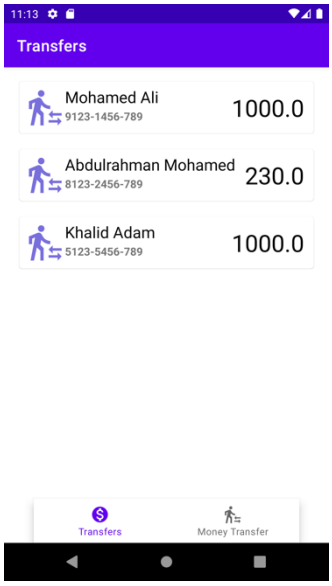
CMPS 312 Mobile Application Development
Lab 8 – Interacting with Web API using Coroutines and Retrofit

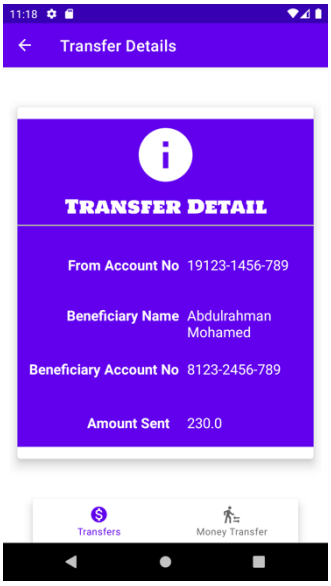
---

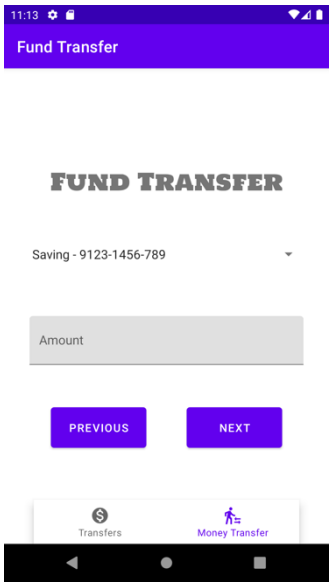
Objective

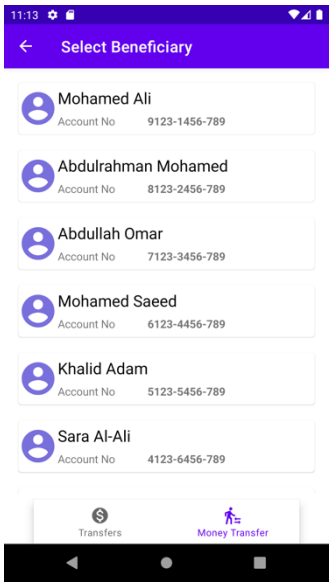
In this Lab, you will **continue building the Banking App** and make the application communicate with Web API. You will be using retrofit library in conjunction with coroutines to get, add, update, and delete transfers and beneficiaries.

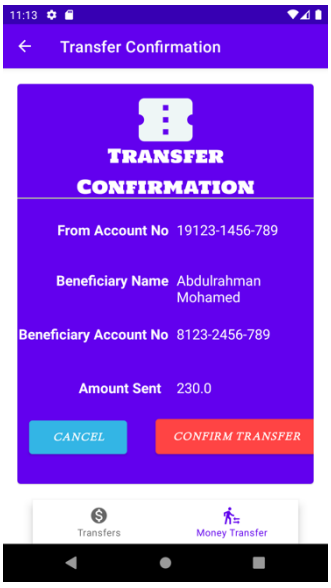
Upon completing this lab you should gain a better understanding of how to interact with Web API using asynchronous suspend functions and coroutines.









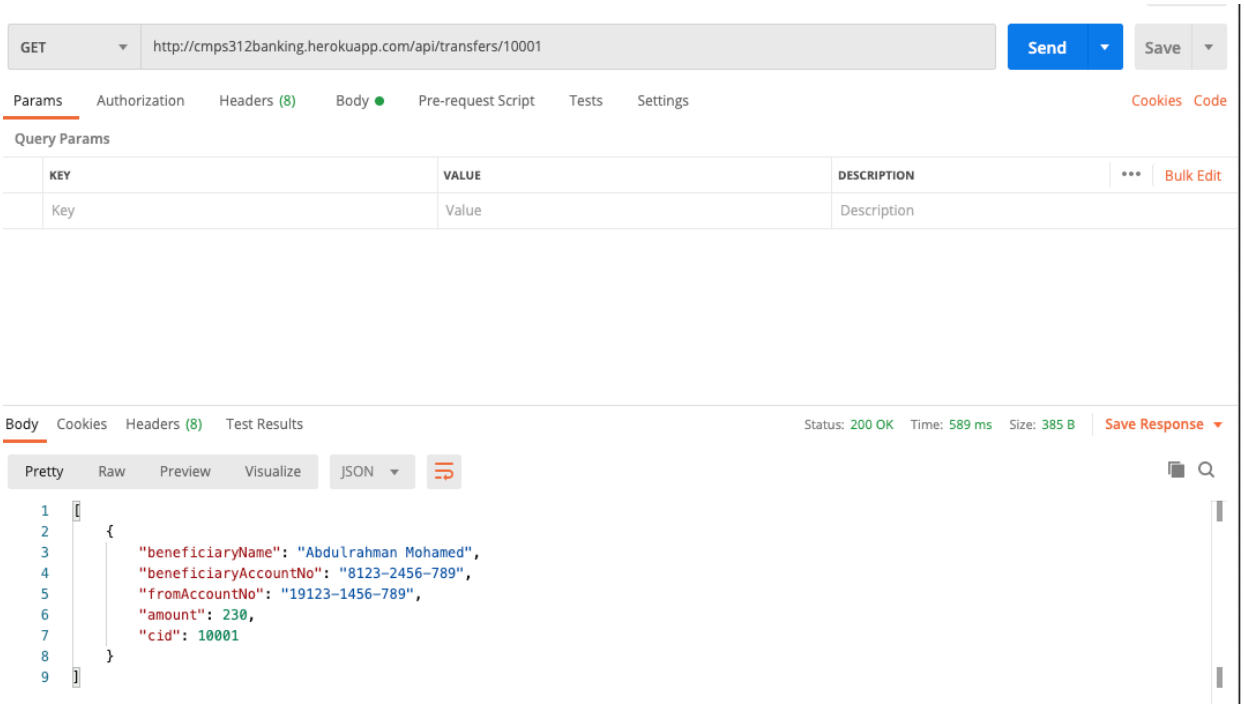


Preparation

1. Sync the Lab GitHub repo and copy the **Lab 8 - Interacting with Web API using Coroutines and Retrofit** folder into your repository.
2. Open the project ***Banking App*** in Android Studio. The project has the complete implementation of **Lab7-BankingApp** with very minor modifications such as swipe to delete and new properties added to the Account class such as cid (i.e., Customer id).
3. Download postman from https://www.postman.com/downloads/ and test the following Web APIs available at https://cmps312banking.herokuapp.com

Available API

Description	Endpoint	Possible Methods
GET Accounts	https://cmps312banking.herokuapp.com/api/accounts/:cid	GET
GET Beneficiaries	https://cmps312banking.herokuapp.com/api/beneficiaries/:cid	GET
ADD and UPDATE Beneficiary	https://cmps312banking.herokuapp.com/api/beneficiaries/:cid	Required cid and for POST and PUT you should provide a beneficiary object as a body
DELETE Beneficiary	https://cmps312banking.herokuapp.com/api/beneficiaries/:cid/:accounNo	DELETE [Requires cid and accountNo in the URL]
GET Transfers	https://cmps312banking.herokuapp.com/api/transfers/:cid	GET
DELETE Transfers	https://cmps312banking.herokuapp.com/api/transfers/:cid/:transferId	DELETE
Local Banks	https://cmps312banking.herokuapp.com/api/banks	GET



PART A: Implementing the Service APIs

In Part A, your task is to implement the services API's that allow the application to communicate with the remote services. You will be using retrofit with coroutines to achieve this.

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

```
//retrofit library
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
//this is to enable as to use the Kotlin Serlization
implementation("com.jakewharton.retrofit:retrofit2-kotlinx-serialization-converter:0.7.0")
// toMediaType() when using application/json
implementation 'com.squareup.okhttp3:okhttp:4.9.0'

// Coroutines
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.9'

def lifecycle_version = "2.2.0"
// ViewModel
implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:$lifecycle_version"
// LiveData
implementation "androidx.lifecycle:lifecycle-livedata-ktx:$lifecycle_version"
```

2. Add the internet permission inside your **AndroidManifest.xml** file or your application will not be allowed to communicate with the network.

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

3. Inside the **model** package create two sub package called **api** and **repository.**
4. Inside the **api** package create an **interface** called **BankService**

   Add all the interfaces methods that allows the application to communicate with the following end points that are listed under this link https://cmps312banking.herokuapp.com

   Example : the following getAccounts() method will be able to call the following end point https://cmps312banking.herokuapp.com/api/accounts/100101 and return a list of accounts for customer 10001.

```
@GET("accounts/{cid}")
suspend fun getAccounts(@Path("cid") cid : Int) : List<Account>
```

   Now add the remaining **eight** methods.

   [getTransfers , addTransfer , deleteTransfer , getBeneficiaries addBeneficiary , updateBeneficiary , deleteBeneficiary]

5. Create an Kotlin file under **model/repository** package and name it **BankRepository.**
6. In **BankRepository** object class declare the following three properties

```
//we are fixing for simplification of the lab. However, in real app this will come from   the logged in user.
 val customerID = 10001     private const val BASE_URL =
"https://cmps312banking.herokuapp.com/api/"
 private val contentType = "application/json".toMediaType()

//this will instantiate the retrofit instance that will allow us to perform the crud operation
 val bankService by lazy {
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(Json { ignoreUnknownKeys = true }.asConverterFactory(contentType))
            .build()
```

```
.create(BankService::class.java)
```

PART B: Linking the App View Models with the Repository

In PART B 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 use the server data. You should use the **retrofit instance** that you created inside the data/repository/BankRepository object.
2. Do the same for the **BeneficiaryViewModel** and make the data to be retrieved from the server instead of the assets folder. You should not change anything else from the application and it should work as before.