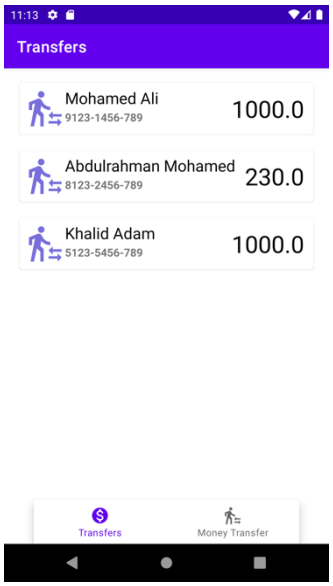
CMPS 312 Mobile Application Development
Lab 8 – Interacting with Web API using Coroutines and Retrofit

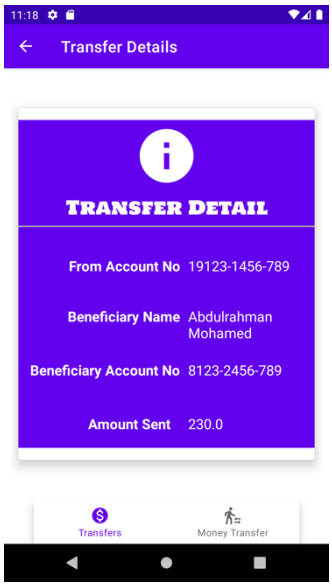
---

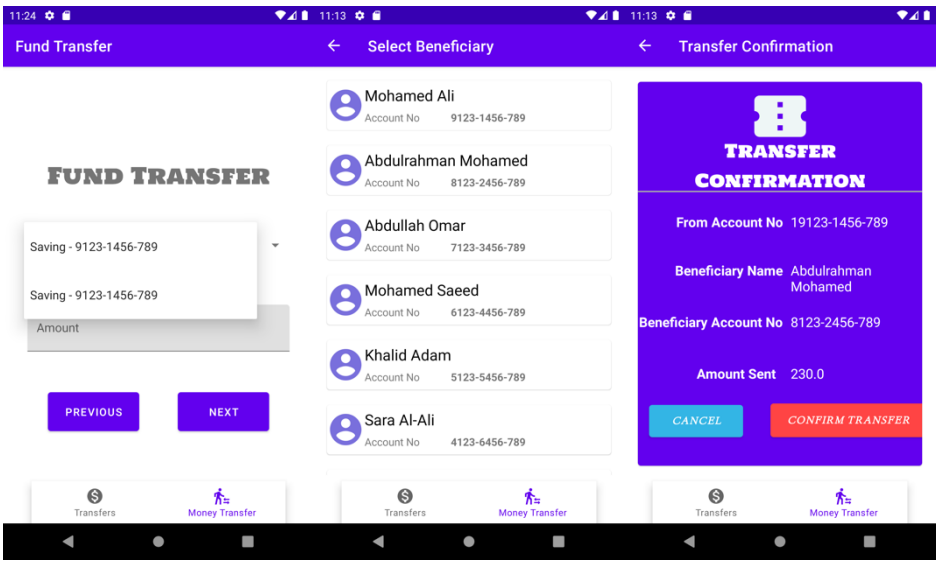
Objective

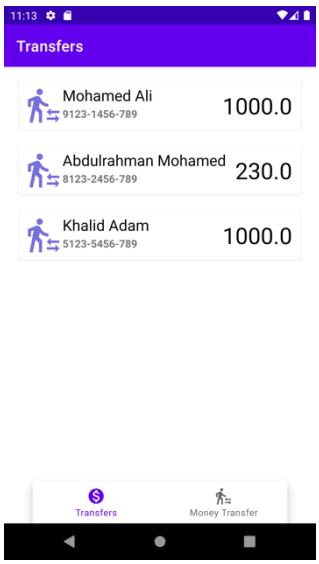
In this Lab, you will **continue building the Banking App** and make the application communicate with Web API. You will be using retrofit library in conjunction with coroutines to get, add, update, and delete transfers and beneficiaries.

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







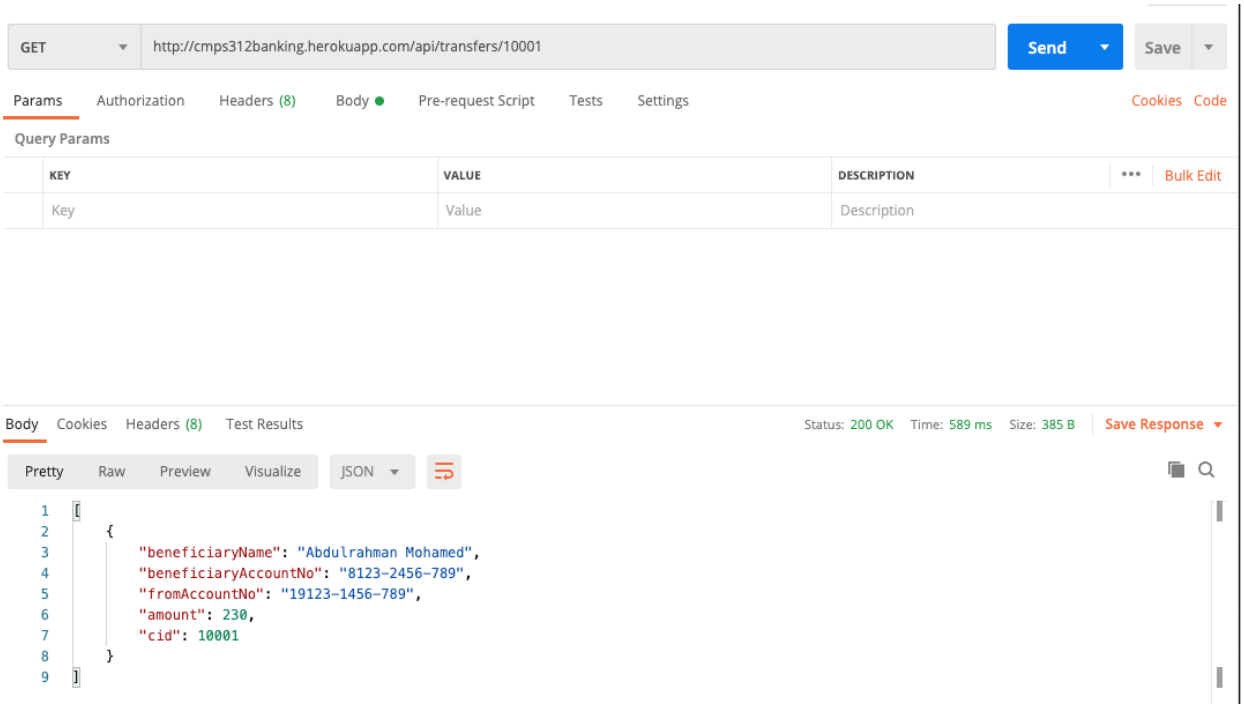


Preparation

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 two projects **Coroutines*ExampleApp and the Banking App*** in Android Studio. The ***Banking App*** 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

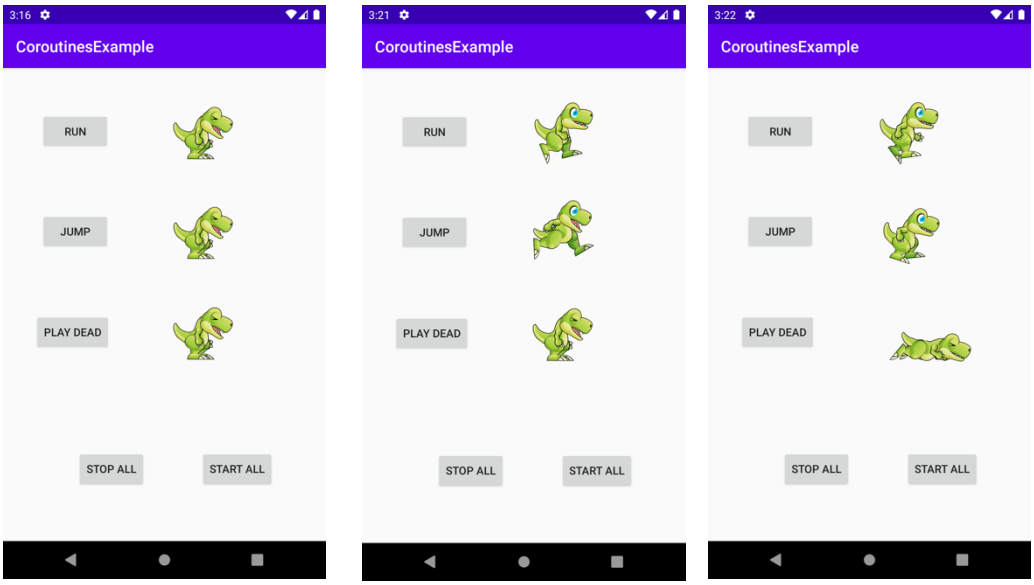
Available API

Description	Endpoint	Possible Methods
GET Accounts	https://cmps312banking.herokuapp.com/api/accounts/:cid	GET
GET Transfers	https://cmps312banking.herokuapp.com/api/transfers/:cid	GET
ADD Transfers	https://cmps312banking.herokuapp.com/api/transfers/:cid	POST
DELETE Transfers	https://cmps312banking.herokuapp.com/api/transfers/:cid/:transferId	DELETE
GET Beneficiaries	https://cmps312banking.herokuapp.com/api/beneficiaries/:cid	GET
ADD Beneficiary	https://cmps312banking.herokuapp.com/api/beneficiaries/:cid	POST [Required cid in the URL]
UPDATE Beneficiary	https://cmps312banking.herokuapp.com/api/beneficiaries/:cid	POST [Requires cid in the URL]
DELETE Beneficiary	https://cmps312banking.herokuapp.com/api/beneficiaries/:cid/:accounNo	DELETE [Requires cid and accountNo in the URL]
Local Banks	https://cmps312banking.herokuapp.com/api/banks	GET



PART A: Coroutines warm-up app

Before dividing deep into the web apis , we will have a small warmup exercises app to understand how coroutines work.



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

```
//todo 1 add the following two dependecies
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-core:1.3.5'
implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.3.5'
```

2. Write three functions named **run**, **jump** and **playDead** that load the sprits from the drawable folder. You can use the following code to load the images.

```
val imgeId = resources.getIdentifier(imageName, "drawable", packageName)
jumpImg.setImageResource(imgeId)
```

3. Then run a loop inside the three functions and animate the images . Below is the complete code that you need inside the jump function. Do the same for the other two functions [run and playDead]. Also, do not forget to add the log message.

```
repeat(1000) {
    val imgeId = resources.getIdentifier("jump${it % 12 + 1}", "drawable", packageName)
    jumpImg.setImageResource(imgeId)
    Log.d(TAG, "jump: ")
}
```

4. Call the three functions when the Start All button is clicked and see if the app will be able to animate the images.

   Question : How smooth is the animation ?

5. Let us try to improve the smoothness of the animation by using coroutines. The first coroutine scope we will see is the **GlobalScope** . So write the following code inside the onCreateMethod

```
        GlobalScope.launch {
            delay(1000) //pause it does not block ..this is a suspend function
//              delay(5000) //pause it does not block
            Log.d(TAG, "Coroutine Thread Name ${Thread.currentThread().name}")
        }
        Log.d(TAG, "Main Thread Name ${Thread.currentThread().name}")
```

Question : What do you understand from the names of the thread?

6. Try to execute your methods inside the coroutine . **How is the animation now?** Better but still not smooth right?
7. Now let us make it even better by making the three methods **suspend** method.

```
suspend fun run()
```

```
jump:
: Skipped 107 frames!  The application may be doing too much work on its main thread.
```

8. Make your suspend functions [**jump** , **run** and **playDead**] run in the IO context.

```
suspend fun run() = withContext(Dispatchers.IO)
suspend fun playDead() = withContext(Dispatchers.IO)
```

When you run your app you should see this error because you are trying to mofify the UI elements from a another thread which the coroutine is running at.

```
Process: cmps312.lab.coroutinesexample, PID: 31727
android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
    at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:8798)
    at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:1606)
```

9. You can fix this by witching the context when assigning the image. See the below code

```
        withContext(Dispatchers.Main) {
            runImg.setImageResource(imgeId)
        }
```

10. Try to make the three functions run in parallel, by launching three co-routiens
11. Try to stop all the coroutines when the user presses the stop all button
12. Compare the lifecycleScope , GlobalScope by adding this code to the pause

```
override fun onPause() {
    finish()
    super.onPause()
}
```

Question : What do you see in the log when you finish the main activity while using the GlobalScope vs lifeCycleScope?

PART B: Implementing the Service APIs

In Part B, 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.

13. 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"
```

14. 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"/>
```

15. Inside the **model** package create two sub package called **api** and **repository.**

16. 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]

17. Create an Kotlin file under **model/repository** package and name it **BankRepository.**
18. 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()
val jsonConverterFactory = Json { ignoreUnknownKeys = true }.asConverterFactory(contentType)
```

```
//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(jsonConverterFactory)  //json to kotlin object and vice versa
        .build()
        .create(BankService::class.java)
}
```

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 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.