CMPS 312 Mobile App Development
Lab 9 – Web API with Coroutines and Ktor

Objective

In this Lab, you will **extend the Banking App** to communicate with the Bank Web API. You will be using the **Ktor** library, asynchronous suspend functions, and coroutines to get, add, update, and delete transfers and beneficiaries.

PART A: Practice Coroutines

In part A, you will create a Stock Market App that displays different stocks and their current prices. You are given the application design, and you need to implement the missing methods. This App will help you understand and practice coroutines.

1. Sync the Lab GitHub repo and copy the **Lab9-Coroutines** folder into your repository.
2. Open the Stock Market App in Android studio and add the following dependencies for ktor and coroutines in your build.gradle app module.

```
//For Ktor Client
val ktor_version: String by project
dependencies {
    implementation("io.ktor:ktor-client-core:$ktor_version")
    implementation("io.ktor:ktor-client-cio:$ktor_version")
     implementation("org.jetbrains.kotlinx:kotlinx-coroutines-
android:1.3.9")
}
```

3. Under the webapi package, create a class called **SimulatedStockQuoteService** that will help us simulate network calls

- Create an attribute named "companies" that has the following values

```
private val companies = mapOf(
    "Apple" to "AAPL",
    "Amazon" to "AMZN",
    "Alibaba" to "BABA",
    "Salesforce" to "CRM",
    "Facebook" to "FB",
    "Google" to "GOOGL",
    "IBM" to "IBM",
    "Johnson & Johnson" to "JNJ",
    "Microsoft" to "MSFT",
    "Tesla" to "TSLA"
)
```

- Create the following four methods. In each of the methods, before returning the values, make sure you delay them between [5000 millis to 1000 millis] to simulate waiting for a response from a remote Web API.

```
/*return the company symbol from the companies map/
suspend fun getStockSymbol(name: String): String

/*return a random number between 50 and 500 as the price
/*val price = (50..500).random()
suspend fun getPrice(symbol: String): Int

/*should utilize the above methods and return a StockQuote object
suspend fun getStockQuote(name: String): StockQuote

/*return the companies list using companies.keys.toList()
suspend fun getCompanies(): List<String>
```

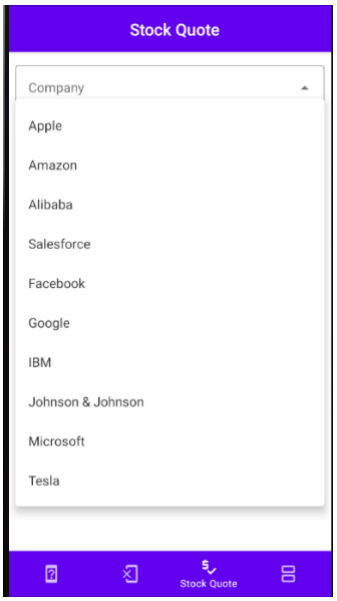
4. First, complete the implementation of the **Get Stock Quote screen**. Open the StockQuoteViewModel and implement the following methods:

a) `suspend fun getCcompanies(): List<String>`

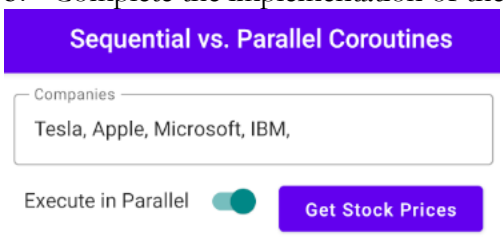
This method should call the stock service's `getCompanies` to initialize the companies list used to populate the company's dropdown.

b) `suspend fun getStockQuote(name: String)`

This method should call the stock service's `getStockQuote` method and pass the selected stock. This method is called from the Get Stock Quote screen when the user selects a company, as shown in the image.



5. Complete the implementation of the Get Stock Quotes screen.



This screen allows the user to enter a comma-delimited list of companies to get their stock prices. The user can select to run getting the stock quotes either sequentially or in parallel.

Open the StockQuotesViewModel and implement the following methods:

a) `fun getStockQuotesSequential(companies: List<String>)`

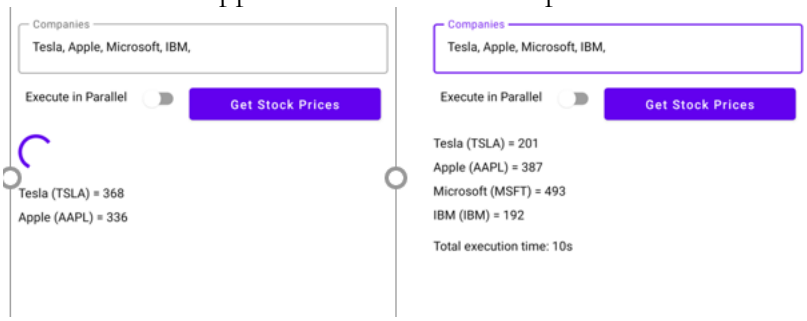
This method should get the stock quote for each company and add the received quote to companyStockQuotes list.

b) `fun getStockQuotesInParallel(companies: List<String>)`

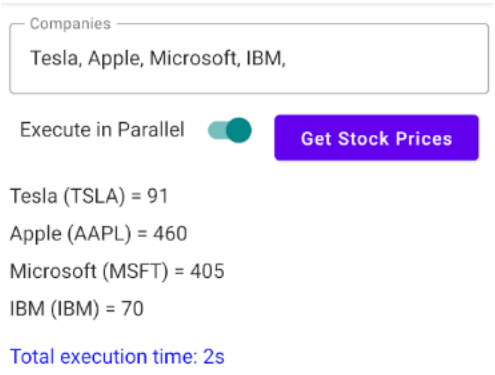
This method should get in parallel the stock quote for each company using:

```
withContext(Dispatchers.IO) {
   companies.map { async { stockQuoteService.getStockQuote(it) } }
            .map { it.await() }
```

6. Run the application and turn of the parallel execution



7. Turn on the Parallel execution and re-run the App. You should see a big improvement with the time it takes to execute.



PART B: Implement the Bank Web API using Coroutines

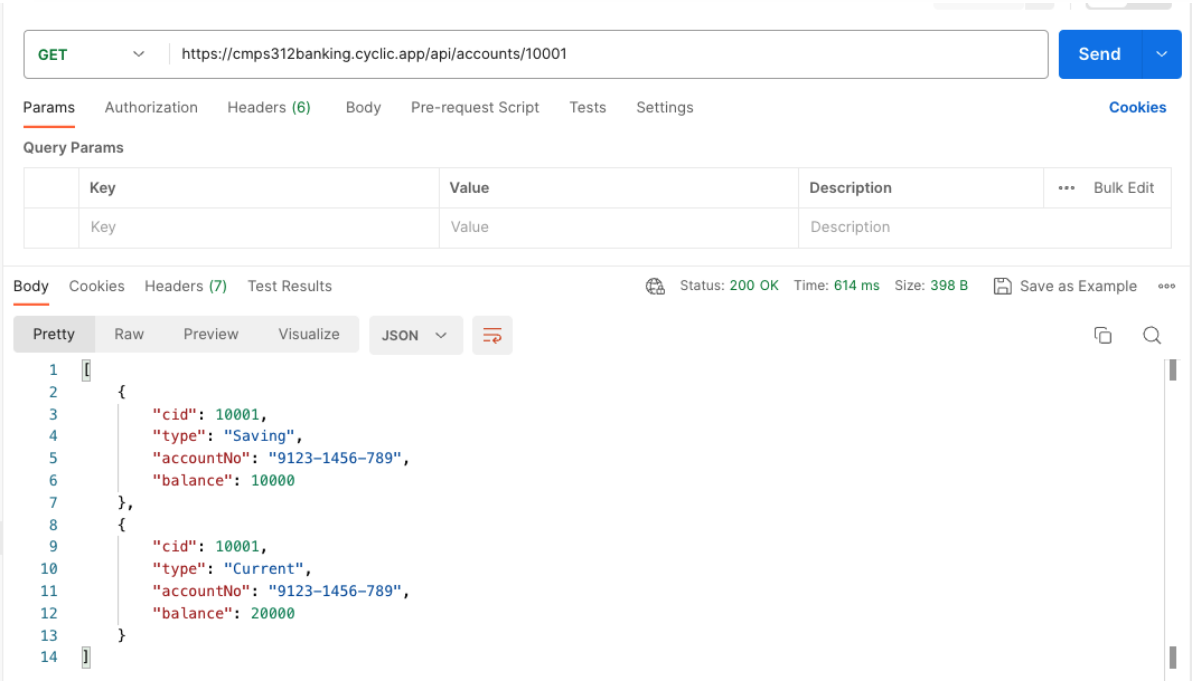
1. Open on Android Studio the ***Banking App*** project from **Lab9-Coroutines** folder in your repository. This project has the complete implementation of **Lab7/8-BankingApp** with some minor modifications such as delete button and new properties added to the Account class such as cid (i.e., Customer id).
2. Your task is to implement calling the Bank Web API using coroutines to read/write data from/to the remote bank service. You will be using Ktor with coroutines to achieve this.

3. Download postman from https://www.postman.com/downloads/ and test the following Banking Service Web API available at https://cmps312banking.cyclic.app

Available API

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

Click here for course material page



1. Inside the **webapi** package, create an **interface** called **BankService**
   List all the interfaces methods that allow the App to communicate with Banking Web API available at https://cmps312banking.cyclic.app

```
interface BankService {
    suspend fun getTransfers(cid: Int): List<Transfer>
    suspend fun addTransfer(transfer: Transfer): Transfer?
    suspend fun deleteTransfer(cid: Int, transferId: String): String
    suspend fun getAccounts(cid: Int): List<Account>
    suspend fun getBeneficiaries(cid: Int): List<Beneficiary>
    suspend fun updateBeneficiary(cid: Int, beneficiary: Beneficiary): String?
    suspend fun deleteBeneficiary(cid: Int, accountNo: Int): String
}
```

2. Create a QuBankService class under the webapi package that implements all the methods that allow the App to communicate with the Bank Web API.
   - Declare the BASE_URL constant and set it to https://cmps312banking.cyclic.app/api
   - Create a HTTP client and add JsonFeature auto-parse from/to json when sending and receiving data from the Web API.

```
private val BASE_URL = "https://cmps312banking.herokuapp.com/api"
private val client = HttpClient { this: HttpClientConfig<*>
    //This will auto-parse from/to json when sending and receiving data from Web API
    install(JsonFeature) { serializer = KotlinxSerializer() }
}
```

   - Implement the **BankService** interface including getTransfers , addTransfer , deleteTransfer , getBeneficiaries , updateBeneficiary , deleteBeneficiary. Create BankServiceTest class and add a main method to test your implementation as you make progress.

   For example: the following getTransfers() method sends a get request to url https://cmps312banking.cyclic.app/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)
}
```

PART C: Change the App ViewModels to use QuBankService implemented in Part B

Your task is to change the app view models to use the **new QuBankService** implemented in Part B.

1. Modify the **TransferViewModel** to use QuBankService and implement the following methods

```
fun getTransfers() {}
fun getAccounts()
fun addTransfer(transfer: Transfer) {}
fun getBeneficiaries() {}
fun deleteTransfer(transferId: String) {}
```

2. Modify the **BeneficiaryViewModel** methods to use QuBankService. You should not change anything else in the App, and it should work as before. **This is the fruit of using MVVM!!!** 👍👌