

# 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 need to complete the provided base solution and 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 Quote App in Android studio and add the following [ktor](#) dependencies to build.gradle of the app module.

```
val ktorVersion = "2.3.5"
implementation("io.ktor:ktor-client-core:$ktorVersion")
implementation("io.ktor:ktor-client-okhttp:$ktorVersion")
implementation("io.ktor:ktor-client-content-negotiation:$ktorVersion")
implementation("io.ktor:ktor-serialization-kotlinx-json:$ktorVersion")
```

3. Under the webapi package, create **SimulatedStockQuoteService** class to simulate network calls.
- Create an attribute named **companies** and initialize it as shown below:

```
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 4 methods. In each of the methods, before returning the values, make sure you delay them between 5000 and 1000 milliseconds 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
```

```

(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>

```

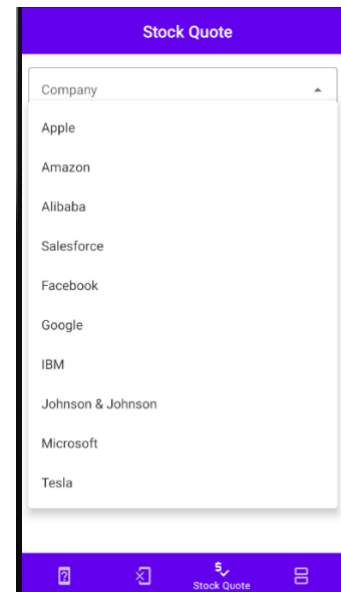
4. First, complete the implementation of the **Stock Quote screen**. Open the StockQuoteViewModel and implement the following methods:

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

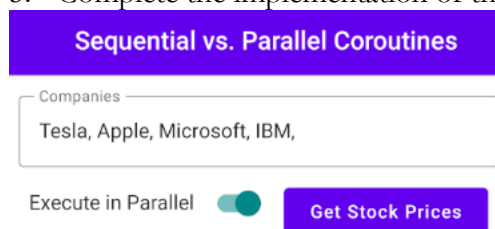
This method should call the stock service's `getCompanies` to initialize the companies list used to populate the company's dropdown.

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

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



5. Complete the implementation of the 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>)`

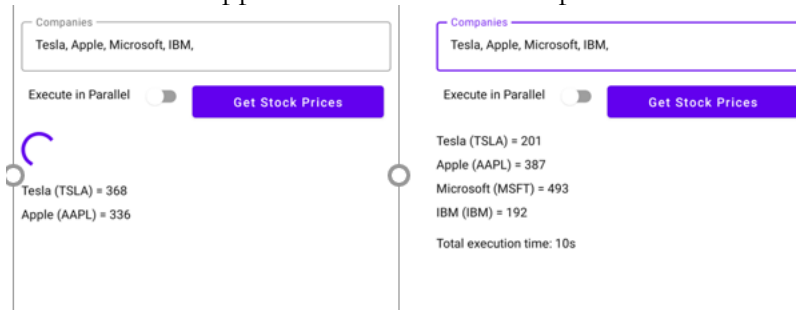
This method should get the stock quote for each company and add the received quote to stockQuotes list.

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

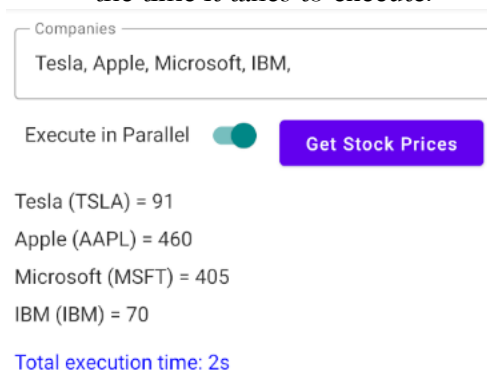
This method should get in parallel the stock quote for each company using:

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

6. Run the application and turn off 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	<a href="https://cmps312banking.cyclic.app/api/accounts/cid">https://cmps312banking.cyclic.app/api/accounts/cid</a>	GET
GET Transfers	<a href="https://cmps312banking.cyclic.app/api/transfers/cid">https://cmps312banking.cyclic.app/api/transfers/cid</a>	GET
ADD Transfers	<a href="https://cmps312banking.cyclic.app/api/transfers/cid">https://cmps312banking.cyclic.app/api/transfers/cid</a>	POST
DELETE Transfers	<a href="https://cmps312banking.cyclic.app/api/transfers/cid/transferId">https://cmps312banking.cyclic.app/api/transfers/cid/transferId</a>	DELETE
GET Beneficiaries	<a href="https://cmps312banking.cyclic.app/api/beneficiaries/cid">https://cmps312banking.cyclic.app/api/beneficiaries/cid</a>	GET
ADD Beneficiary	<a href="https://cmps312banking.cyclic.app/api/beneficiaries/cid">https://cmps312banking.cyclic.app/api/beneficiaries/cid</a>	POST [Required cid in the URL]
UPDATE Beneficiary	<a href="https://cmps312banking.cyclic.app/api/beneficiaries/cid">https://cmps312banking.cyclic.app/api/beneficiaries/cid</a>	POST [Requires cid in the URL]
DELETE Beneficiary	<a href="https://cmps312banking.cyclic.app/api/beneficiaries/cid/accountNo">https://cmps312banking.cyclic.app/api/beneficiaries/cid/accountNo</a>	DELETE [Requires cid and accountNo in the URL]
Local Banks	<a href="https://cmps312banking.cyclic.app/api/banks">https://cmps312banking.cyclic.app/api/banks</a>	GET

[Click here for course material page](#)

GET <https://cmps312banking.cyclic.app/api/accounts/10001> Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results Status: 200 OK Time: 614 ms Size: 398 B Save as Example ...

Pretty Raw Preview Visualize JSON

```

1  {
2    {
3      "cid": 10001,
4      "type": "Saving",
5      "accountNo": "9123-1456-789",
6      "balance": 10000
7    },
8    {
9      "cid": 10001,
10     "type": "Current",
11     "accountNo": "9123-1456-789",
12     "balance": 20000
13   }
14 }
```

- 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://cmpps312banking.cyclic.app/api>
- Create a HTTP client and add Json plugin to auto-parse from/to json when sending and receiving data from the Web API.

```
private val client = HttpClient(OkHttp) {
    install(ContentNegotiation) {
        json(
            json = Json {
                ignoreUnknownKeys = true
                prettyPrint = true
            }
        )
    }
}
```

- 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://cmpps312banking.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"
    val response = client.get(url)
    return response.body()
}
```

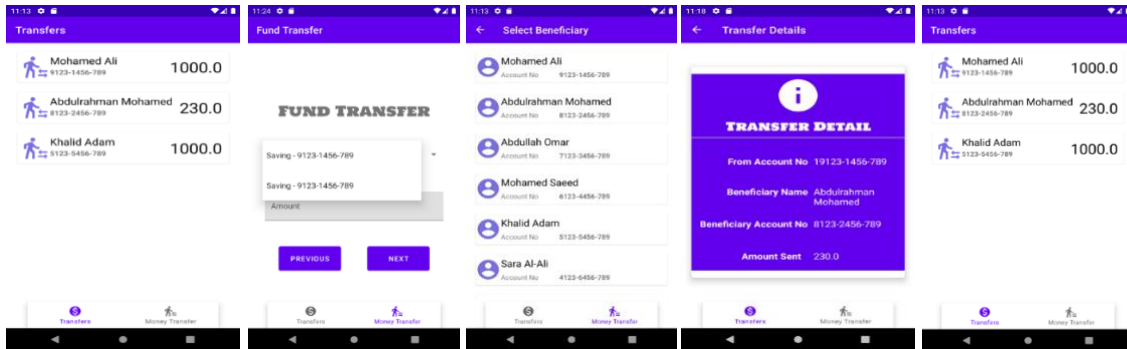
### **PART C: Change the App ViewModels to use QuBankService implemented in Part B**

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!!!** 👍👉



## PART D: Improve application by using Flows

Instead of manually retrieving transfer data from the server, we will modify our application to periodically request this data. To accomplish this, we will replace the existing **getTransfers** method inside the **BankService** and **QuBankService** classes with Flow-based implementations. Below is the code after making these changes: In the **BankService** interface, alter the return type of the **transfer** method to **Flow** as follows:

```
fun getTransfers(cid: Int) : Flow<List<Transfer>>
```

Within the **QuBankService** class, update the method's return type to **Flow** and implement a flow that periodically fetches data:

```
private val refreshIntervalMs: Long = 5000
override fun getTransfers(cid: Int) = flow<List<Transfer>> {
    val url = "$baseUrl/transfers/$cid"
    while (true) {
        val transfers = client.get(url).body<List<Transfer>>()
        emit(transfers)
        delay(refreshIntervalMs)
    }
}
```

Inside the **BankingViewModel**, replace the transfers **mutableStateList** with a Flow:

```
var transfers: StateFlow<List<Transfer>> =
    quBankService.getTransfers(10001).stateIn(
        scope = viewModelScope,
        started = SharingStarted.WhileSubscribed(5000),
        initialValue = emptyList()
    )
```

Additionally, comment out the **getTransfers** method inside the **init** function, as it is no longer needed. Update the code in the view/**TransfersList** composable to use the flow

```
val transfers = bankingViewModel.transfers.collectAsState().value
```

Run the application. Through PostMan try to add a new transfer, you should get this transfer immediately reflected in the application without you reloading the app.