Assignment 9

Task 1

a) ID changed to “testingApp”

b)

The architecture of the app follow the MVVM (Model-View-ViewModel) pattern, which is a software architectural pattern that separates the development of the graphical user interface from the development of the business logic or back-end logic. Here's a breakdown of the layers:

UI Layer (View and ViewModel): The View layer is represented by the Composable functions in the StartScreen.kt, SamplingScreen.kt, and StatisticsScreen.kt files. These Composable functions define the UI and user interactions. They receive a ViewModel and a navigation function as parameters. The ViewModel layer is represented by the StartViewModel, SamplingViewModel, and StatisticsViewModel classes. These classes are responsible for preparing and managing the data for the View. They handle the business logic and expose the data for the UI. They also communicate with the data layer.

Data Layer (Repository and Data Source): The Repository layer is represented by the RepositoryImpl class in the RepositoryImpl.kt file. It is responsible for deciding whether to fetch data from a local database or a remote source, and it provides a clean API for the ViewModel to retrieve the data. The Data Source layer is represented by the CouchDB class in the CouchDB.kt file. This class is responsible for fetching and storing data from/to a CouchDB server. It provides get and put methods to retrieve and store JSON objects.

In summary, the View observes data from the ViewModel. The ViewModel retrieves data from the Repository, and the Repository fetches data from the Data Source. This separation of concerns makes the code more modular, easier to test, and easier to maintain.

c)

JSON documents are sent and retrieved asynchronously to keep the main UI thread responsive and handle I/O-bound tasks efficiently. Kotlin coroutines are used for this, allowing for non-blocking, concurrent operations in a readable manner.

Key Points:

1. Asynchronous Operations: Prevents UI freezes and improves user experience.
2. withContext Usage:
    - Context Switching: withContext(Dispatchers.IO) shifts execution to a background thread for blocking I/O tasks.

    - Resource Management: Ensures proper handling of resources using try-finally blocks.
3. Exception Handling: The try-finally block ensures HttpURLConnection is disconnected properly even if an exception occurs.

This approach maintains app responsiveness and ensures efficient handling of network **operations.**

Task 2

a) General Purpose of Map and Reduce Functions for Databases

1. Map Function:
    - Purpose: The map function processes each document in the database to generate key-value pairs.
    - Operation: It scans through documents and applies a user-defined function to each document, emitting zero or more key-value pairs based on the logic provided in the function.
    - Result: The emitted key-value pairs are indexed and can be queried later.
2. Reduce Function:
    - Purpose: The reduce function aggregates the results of the map function.
    - Operation: It takes the output from the map function and combines it in a meaningful way, such as summing values, calculating averages, or concatenating strings.
    - Result: The aggregated result is typically a smaller, summarized dataset that can provide insights or summaries over large datasets.

b) Explanation of the Map Function

The provided map function in the design document:

```
function(doc) {
   if (doc._id.search("test") == 0 && doc.c.length >= 1)
      emit(doc.b, doc.a)
}
```

Selection Criteria: The function selects documents that meet the following conditions:

- The document's _id starts with the string "test".

    - The document has a property c which is a string or array with a length of 1 or more.
- Key-Value Pairs Emitted:
    - Key: The value of the document's b property.
    - Value: The value of the document's a property.

Thus, for each document that meets the criteria, the map function emits a key-value pair where the key is the value of b and the value is the value of a.

c) Output for the Given View Invocation

The view is invoked using:

https://couchdb.myserver.de/mydb/_design/mapreduce1/_view/ctxt?key=%22x%22

This means we are querying the view with the key "x".

Given the example documents:

1. {"_id":"test1", "a":10, "b":"x", "c":"hello"}
    - _id starts with "test" and c has length >= 1.
    - Emits: {"key":"x", "value":10}
2. {"_id":"test3", "a":30, "b":"x"}
    - Does not meet the criteria as c is missing.
    - Emits: None
3. {"_id":"test4", "a":40, "b":"x", "c":[1,2,3]}
    - _id starts with "test" and c has length >= 1.
    - Emits: {"key":"x", "value":40}
4. {"_id":"test5", "a":50, "b":"x", "c":""}
    - _id starts with "test" but c has length < 1.
    - Emits: None

Therefore, the documents that match and emit key-value pairs are test1 and test4. Filtering by the key "x", we get the following results:

```
[
 {"key": "x", "value": 10},
 {"key": "x", "value": 40}
]
```

This is the JSON array result for the given view invocation.