**Articles to look at here**

[**https://github.com/weeeBox/mobile-system-design**](https://github.com/weeeBox/mobile-system-design)

**Cursor vs offset pagination**

**How to implement?**

The client provides the following parameters in the request to the server:

page\_size: 50  
page: 1

The server sends back the first 50 results.

On the next request, the client might provide the following

page\_size: 50  
page: 5

**Con:**

1. Not very easy to scale here because the scanning happens from start to the end of the database

**Using the cursor pagination here**

{  
 items: [  
 // 50 rows of data  
 ],  
 cursor: "some opaque string"  
}

When the client requests the next set of 50 rows, it sends the server the cursor.

page\_size: 50  
cursor: "some opaque string"

Cursor is like a special pointer that shows where the next set of results can be found. The server uses the cursor to determine where the data that the client needs is located, this is very essential

**Pros and cons of the cursor based pagination**

Pros of the Cursor-based pagination

* Scales well. It is very efficient for large datasets because it can quickly seek to the cursor and return results without scanning through previous rows.
* Good for cases where the data is frequently changing, for example, if we are dealing with live-feed data for a social media platform
* Cursor-based pagination offers more consistent results than offset-based pagination. Even if the dataset changes, because the cursor always points to a specific location in the dataset.

Cons of the Cursor-based pagination

* Implementation is typically more complex
* Setting up logic to navigate between pages and jump to a specific page number is more complex.

**Next project here**

1.

**What happens when an app regains its connectivity?**

1. It eneds to sync local data with the network data here, that’s called synchoronization. There r 2 types: pull-based and push-based here

**What’s Pull-based here sync here?**

1. App reaches out to the network here, and then that’s part 1 here/

**Push-based sync?**

**It proactively fetches an appropriate amount of data on first start-up to set a baseline, after which it relies on notifications from the server to alert it when that data is stale.**

**What does the sync function look like?**

class UserDataRepository(...) {

suspend fun synchronize() {

val userData = networkDataSource.fetchUserData()

localDataSource.saveUserData(userData)

}

}

**Can we also have hybrid-based sync?**

Some apps use a hybrid approach that is pull or push based depending on the data. For example, a social media app may use pull-based synchronization to fetch the user's following feed on demand due to the high frequency of feed updates. The same app may opt to use push-based synchronization for data about the signed-in user including their username, profile picture and so on.

Ultimately, offline-first synchronization choice depends on product requirements and available technical infrastructure.

**How does conflict resolution get resolved?**

Device attaches timestamp to the data they write, so the network knows which data is the newest and then write to that as soon as possible when receiving them

So this way the newest data write will win here.

**How does workManager help in this scenario?**

1. We need both a read and a write queue here

When network connected again:

2. We drain from the read q

3. Also drain from the write q

The below is not really necessary here for knowing just for general, when the app starts up we have

 On start-up, the app performs the following actions:

1. Enqueue read synchronization work to make sure there is parity between the local datasource and the network datasource.
2. Drain the read synchronization queue and start synchronizing when the app is online.
3. Perform a read from the network datasource using exponential backoff.
4. Persist the results of the read into the local datasource resolving any conflicts that may occur.
5. Expose the data from the local datasource for other layers of the app to consume.

What are the components of a workManager?

1. SyncInitializer

- to sync any work with function enqueueUniqueWork()

2. SyncWorker()

- override fun doWork -> defines the work that needs to be done in here this is important

**The following section falls on the data layer?**

1. A diagram of data source

Description automatically generated

What does the naming look like?

class ExampleRepository(

private val exampleRemoteDataSource: ExampleRemoteDataSource, // network

private val exampleLocalDataSource: ExampleLocalDataSource // database

) { /\* ... \*/ }

Example: PaymentRepository contains everything related to payment

LoginRepository: has everything that the login credential needs here

And then here we have the file here

1.