Paging Library in Compose

By Moro


Tails, my pal, you are the only one that can help me to implement paging in compose.


Let's do it!

- **gradle (app module)**

```
implementation("androidx.paging:paging-compose:1.0.0-alpha20")
```

- **MoviesDataSource (repository layer)**

```
class ItemsDataSource(api: Api) : PagingSource<Int, ItemResponse>() {

    override fun getRefreshKey(state: PagingState<Int, ItemResponse>): Int? = null

    override suspend fun load(params: LoadParams<Int>): LoadResult<Int, ItemResponse>
    {
        val nextPageNumber = params.key ?: 1
        return try {
            val response = api.getItems(pageIndex = nextPageNumber)
            LoadResult.Page(
                data = response,
                prevKey = if(nextPageNumber > 1) nextPageNumber - 1 else null,
                nextKey = nextPageNumber.plus(1)
            )
        } catch(exception: Exception) {
            LoadResult.Error(exception)
        }
    }
}
```

Second, we need to build our data source

- **MoviesDataSource (repository layer)**

```
class ItemsDataSource(api: Api) : PagingSource<Int, ItemResponse>() {

    override fun getRefreshKey(state: PagingState<Int, ItemResponse>): Int? = null

    override suspend fun load(params: LoadParams<Int>): LoadResult<Int, ItemResponse>
    {
        val nextPageNumber = params.key ?: 1
        return try {
            val response = api.getItems(pageIndex = nextPageNumber)
            LoadResult.Page(
                data = response,
                prevKey = if(nextPageNumber > 1) nextPageNumber - 1 else null,
                nextKey = nextPageNumber.plus(1)
            )
        } catch(exception: Exception) {
            LoadResult.Error(exception)
        }
    }
}
```

This **load** method will be called each time we request a page.
The ***nextPageNumber*** represents the current page we will ask to the server…

- **MoviesDataSource (repository layer)**

```
class ItemsDataSource(api: Api) : PagingSource<Int, ItemResponse>() {

    override fun getRefreshKey(state: PagingState<Int, ItemResponse>): Int? = null

    override suspend fun load(params: LoadParams<Int>): LoadResult<Int, ItemResponse>
    {
        val nextPageNumber = params.key ?: 1
        return try {
            val response = api.getItems(pageIndex = nextPageNumber)
            LoadResult.Page(
                data = response,
                prevKey = if(nextPageNumber > 1) nextPageNumber - 1 else null,
                nextKey = nextPageNumber.plus(1)
            )
        } catch(exception: Exception) {
            LoadResult.Error(exception)
        }
    }
}
```

We need to help the paging library defining the previous key, and the next key to be used.

- **MoviesDataSource (repository layer)**

```
class ItemsDataSource(api: Api) : PagingSource<Int, ItemResponse>() {

    override fun getRefreshKey(state: PagingState<Int, ItemResponse>): Int? = null

    override suspend fun load(params: LoadParams<Int>): LoadResult<Int, ItemResponse>
    {
        val nextPageNumber = params.key ?: 1
        return try {
            val response = api.getItems(pageIndex = nextPageNumber)
            LoadResult.Page(
                data = response,
                prevKey = if(nextPageNumber > 1) nextPageNumber - 1 else null,
                nextKey = nextPageNumber.plus(1)
            )
        } catch(exception: Exception) {
            LoadResult.Error(exception)
        }
    }
}
```

If something goes wrong, we need our try-catch block!


Tails, it is not working…


We have more work to do Sonic!

- **Repository (repository layer)**

```
class Repository(api: Api) {

    fun data(): Flow<PagingData<ItemResponse>> {
        return Pager(
            pagingSourceFactory = {
                ItemsDataSource(api)
            },
            initialKey = 1,
            config = PagingConfig(
                pageSize = 20,
                maxSize = 2000
            )
        ).flow
    }
}
```

In our repository, we need to create our Pager, specifying the previous data source we created…

- **Repository (repository layer)**

```
class Repository(api: Api) {

    fun data(): Flow<PagingData<ItemResponse>> {
        return Pager(
            pagingSourceFactory = {
                ItemsDataSource(api)
            },
            initialKey = 1,
            config = PagingConfig(
                pageSize = 20,
                maxSize = 2000
            )
        ).flow
    }
}
```

The first page index depends on the API, in our case, the first page is 1


We need to send this to our Composable **LazyColumn**/ **LazyRow**, right?


Let’s do it!

We need to create our ViewModel…

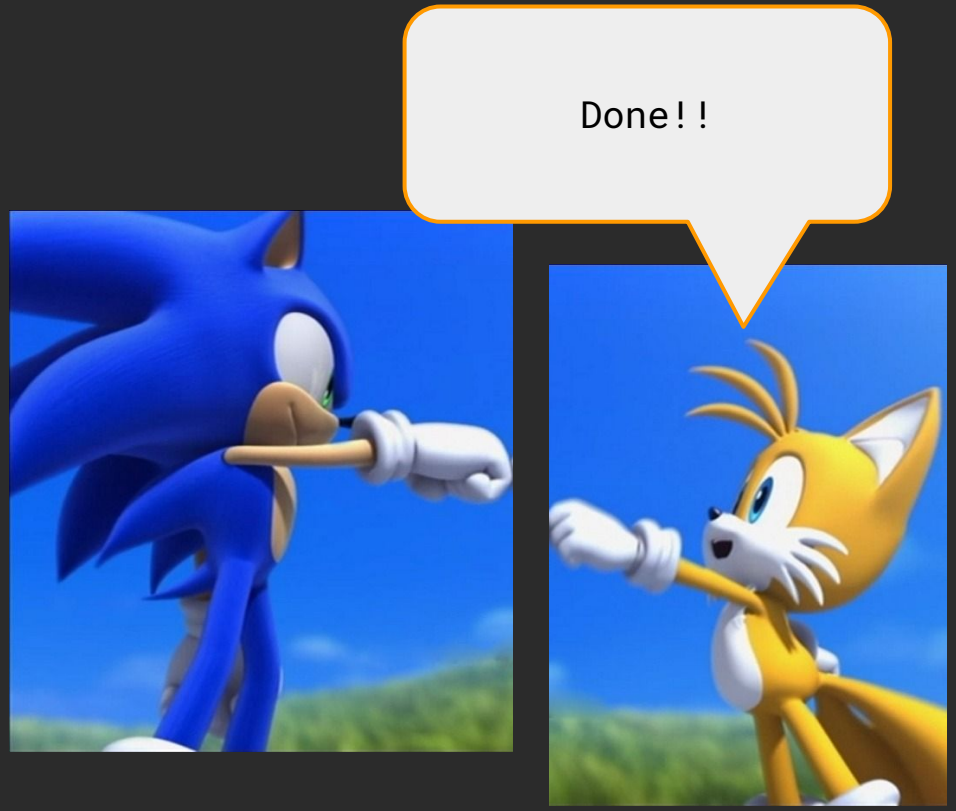
- **ViewModel (presentation layer)**

```
class MyBeautifulViewModel(repository: Repository) {

    fun pages() = repository.data()
}
```

Let’s call it in our LazyColumn…

- **Composable (presentation layer)**

```
val flowData = remember { viewModel.pages() }
val pageItems = flowData.collectAsLazyPagingItems()

LazyColumn(
    itemFactory = {
        items(
            count = pageItems.itemCount,
            key = pageItems.itemKey(),
            contentType = pageItems.itemContentType
        )
    }
) {
    pageItems[index]?.let {
        ItemCard(it)
    }
}
```


Done!!