JetCost technical interview

This document will contain the analysis of important technical and structural decision taken to develop the test application.

# Single Activity Architecture

I choose to use single activity architecture because it’s always best practice to use fragments since you can re-use them. Also in this case the application flow is really simple and switching fragments inside the activity is lighter than switching activity every time.

Another upside is that we can easily share viewmodel between fragment, if needed.

# Navigation component

Navigation wise, I selected the native Android Jetpack navigation component with bottom navigation view. I consider it the best option since it’s native from Google and it handles the back stack perfectly and without any additional code. Last but not least the Android Studio navigation editor to organize and link all the fragments is very productive and convenient.

# Koin

I decided to use Koin as a dependency injection tool because it’s the simplest among all and doesn’t bring the complexity of Dagger in a project that doesn't really need it. Koin provides every needed functionality and is built on Kotlin.

# Google Search API

For the image search part the app uses the Google Custom Search API + Programmable Search Engine. I preferred this API for its value, since Google Image is the largest and most important search engine for images on the internet and it’s API key was one of the most accessible, without needing to submit a request for approval, like Flickr or Pinterest. Also it’s response contains all the data that we needed for the application.

# Paging3

To display the data I used a RecyclerView paired with Android Jetpack’s library, Paging3. Given the fact that we are displaying a list of images loaded from the internet, with an API that has a quota that you don't want to exceed and we don't want to waste user’s data, Paging3 was the best solution to load data on the run, only if users need it. It’s the perfect solution because it’s not over complex, native and really functional.

# MVVM

As a design pattern, MVVM was the best choice. To make it perfectly work, I paired it with a repository pattern. It’s the perfect solution to divide the application in different “duty-areas” to better organize, maintain and develop the code.

# Room

To save bookmarks offline and persistently, the app uses Room, the Android Jetpack native Google’s library. It’s based on SQLite and it suits perfectly what I needed, being lightweight, fast and perfectly integrated.