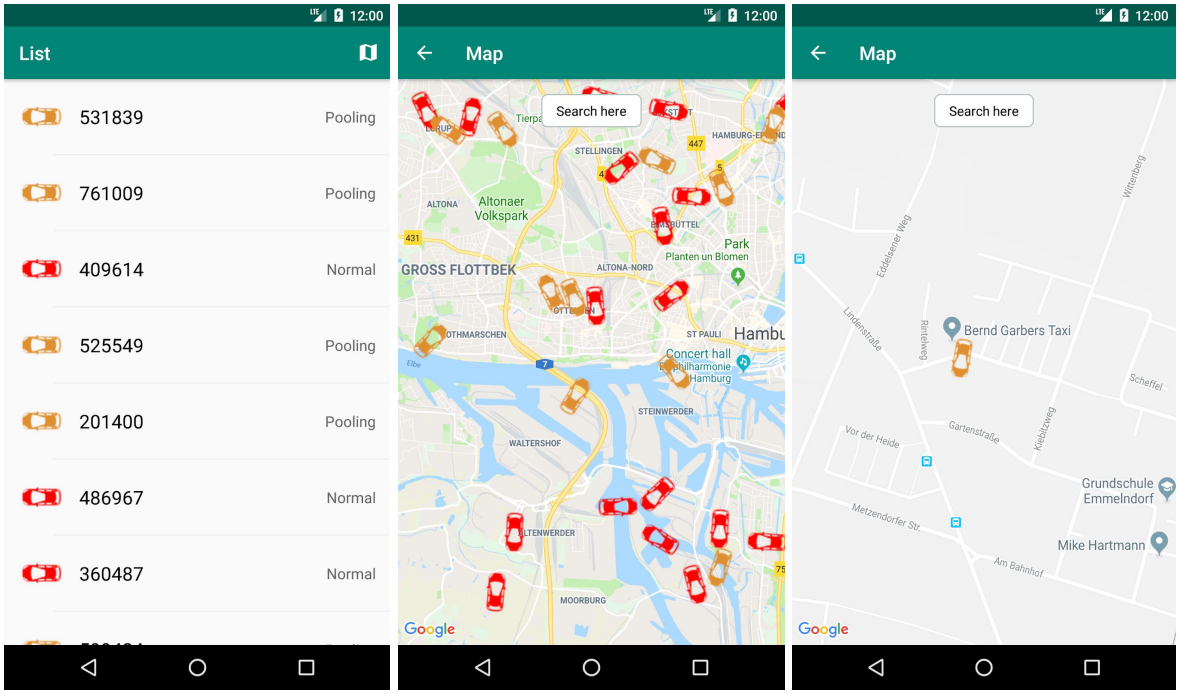
mytaxi code challenge

An app to display a list of taxis and also show them on a map.

Summary

A small app where the user sees a list of taxis that are within the bounds of the city of Hamburg, retrieved through a stub like API. The user can then switch to the map page to see those taxis laid out on a google map, search on a visible region on the map or select one of the items in the list and see the position of that taxi on the map.

Screenshots



Architecture

The architecture used here is a simple MVVM. It allows us to take full advantage of Android's architecture components, while keeping things as simple as possible. All classes in this project are made in a way that they can be created using dependency injection, `Dagger2` is the DI tools used here. The same principle makes them testable,

and there are tests found for all different layers. The tests present are rather a showcase than a complete test suite due to the nature of this project.

The data flow reactively flows from Model layer to the VM model using `rx` and from there using `LiveData`, the changes are observed on the view level and `Databinding` is then used to display the data on the actual layouts.

The view architecture is a little different than my usual setup where I use a parent activity that contains different pages as fragments and handles all the different interactions between different components. Here we have a main activity that contains the list fragment and then another activity that displays the map. This was done to prevent potential fragment issues (lifecycle, reloading of data) by starting a clean, new instance of the map activity each time it was needed. It is however completely feasible to have both pages as child fragments in that parent activity, in a real project where you are able to put more time and effort into it.

Different parts of the project (packages)

The app is separated by feature driven packages, meaning almost all classes related to a certain feature would end up in the same package. Apart from this, there are also function packages, that host classes that behave the same way but for various different classes across the entire app. This approach was picked because it makes it easier to jump between different classes and find what you are looking for, when everything is really close together. There is an argument for data classes to be included in the feature packages as they are closely related but since they are used in different features, it makes sense for them to have their own package here.

The different packages are

- core - app classes, base classes, general classes
- data - models, repository and rest service
- di - dependency injection classes
- providers - rx schedule providers
- taxiList - classes related to the view part of the taxi list feature
- taxiMap - classes related to the view part of the taxi map feature

Specs

Langage = Kotlin

compileSdkVersion = 28

targetSdkVersion = 28

minSdkVersion = 24

kotlin version = 1.3.21

gradle plugin version = 3.4.0

libraries used

- Kotlin libs
- AppCompat (Androidx)
- Material
- Arch components
- ConstraintLayout
- RecyclerView
- CoordinatorLayout
- Retrofit (and its converter, interceptor, mock web server)
- Google maps
- Dagger2
- RxAndroid and RxJava
- jUnit
- Mockito
- Mockito-Kotlin
- Espresso

Issues

There are a couple of known issues with the map, where a) The map just would not load b) The map loads but the markers would not be drawn. These issues happened very rarely and only appeared on the emulator. The second issue is resolved by simply

closing and opening the map page but the first issue required the app to be reinstalled (this has happened just once). The root cause of these was not found.

Improvements

As mentioned before, due to the nature of the project being a test with time constraints, the app is not fully fledged and there is a lot of room for improvement. Here are a few that spring to mind

- A `Toolbar` can be used in all pages, as it is the most up-to-date "version" of the `Actionbar` and allows better and cleaner usage
- Originally the map was planned to appear as a bottom sheet that would appear from below and go over the list, which would result in better UX
- A loading indicator for the map page (not needed currently as the stub API is very fast)
- Disk level persistence can be used for caching the data
- A more user friendly no-internet/timeout error

External sources

This section will cover a couple of classes where the code is not entirely written from scratch and at least some of it was taken from elsewhere.

- `CustomSerializerDeserializer` - the original code for this class was lifted from a few different stackoverflow questions, such as this one, but the version of it that has made it to the app is massively different as a lot of tinkering was needed to make it work for our specific needs.
- `CustomViewModelFactory` - In all honesty, I can't remember where this class comes from, I've been using it in all my projects as it just works :).

Github repository

The content of the project, along with its git history, can be found on the following repository

mytaxi-test - https://github.com/Namiii/mytaxi-test