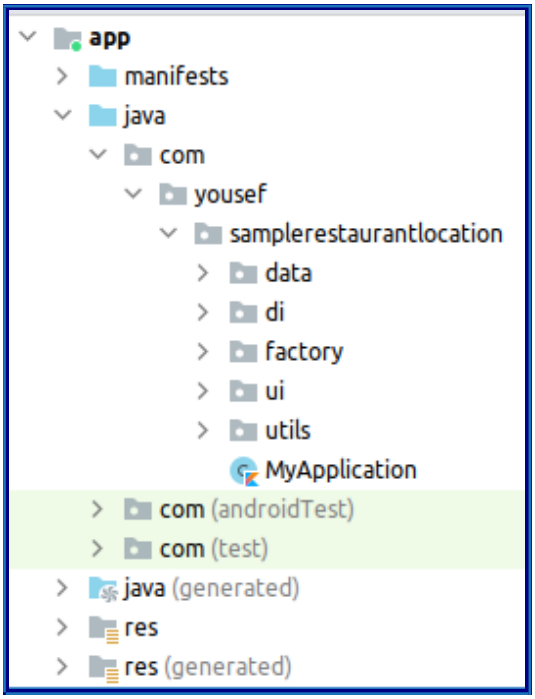
Document

In this project I've separated the responsibilities and used technologies to increase de-coupling and also make the project testable and scalable.



The app has following packages:

data: It contains all the data accessing and manipulating components.

di: Dependency providing classes using Dagger2.

factory: Contains ViewModelProviderFactory that is responsible to instantiate ViewModels.

ui: View classes along with their corresponding ViewModel.

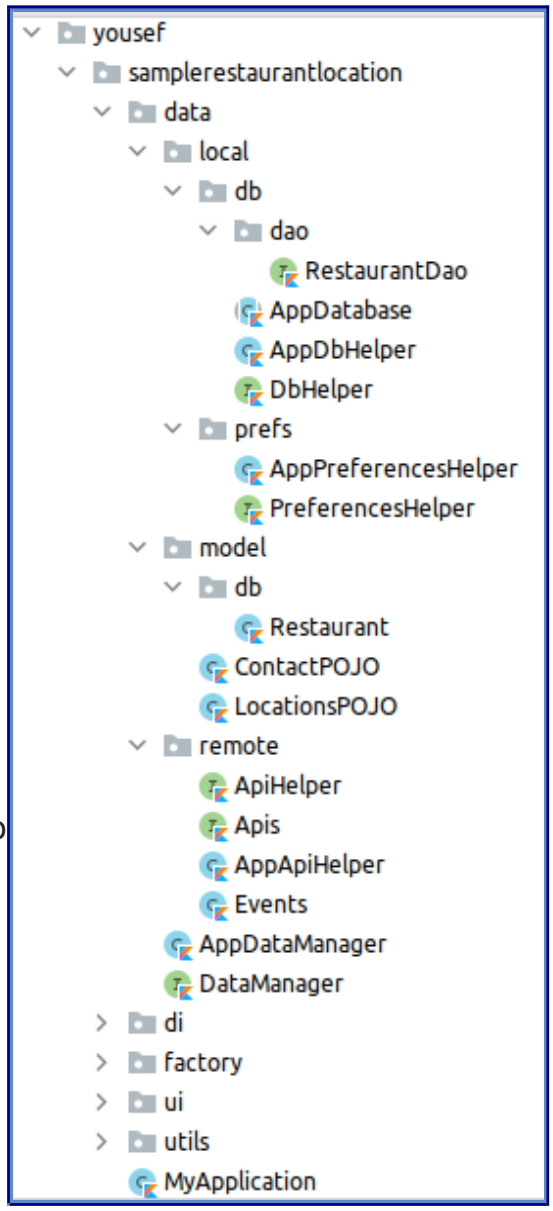
utils: Utility classes.

Classes have been designed in such a way that it could be inherited and maximize the code reuse.

I've used MVVM architecture to make it easy to maintain the code and develop it and although it was a sample project, its structure really ready to be extended.

List of Libraries: Coroutines, Dagger, Retrofit, mockito, calligraphy, Room.
By using of Navigation component to implement single activity app.

Data package has the following packages:



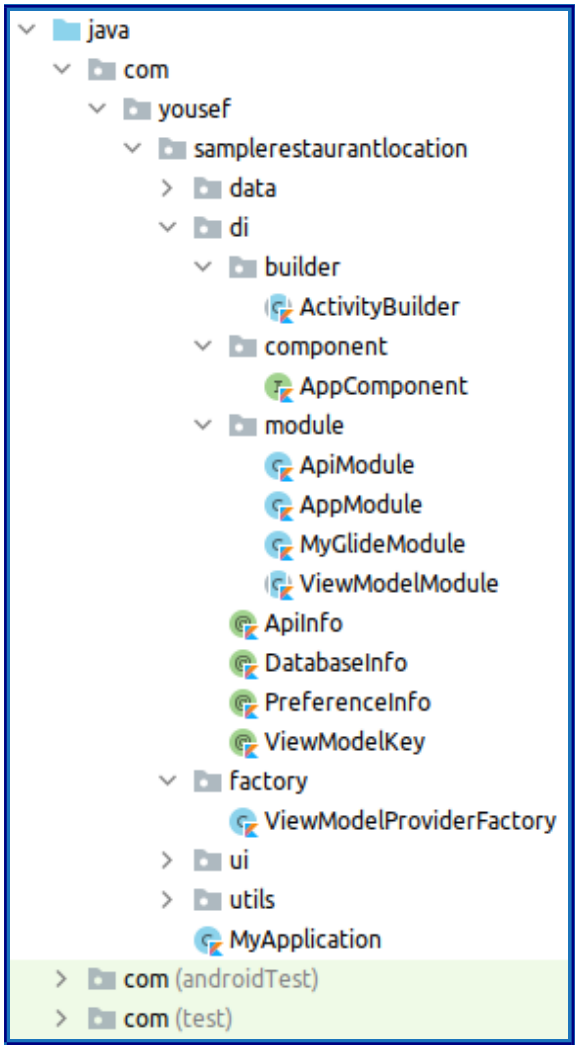
loca**l**: It contains db and prefs packages for saving restaurant structured data in a local database using Room, and some key-value pairs like last LatLng and the selected restaurant to be displayed in details respectively.

model: Model files are responsible for mapping the received Json data put each data in its place.

remote: to handle all API related operations by using Retrofit features.

and AppDataManager and DataManager: to express the dependencies of Application Context, DbHelper, ApiHelper and PreferencesHelper in the contructor. This part of project provides all the apis to access the data of the application.

Di package has the following packages:



builder: It contains the file named ActivityBuilder which is a module class. This subcomponent takes modules to be installed in the generated and will

be added in AppComponent as a module.

component: I created a Component which links the MyApplication and modules.

AppComponent is an interface that is implemented by Dagger. In this interface using @Component we specify the class to be a Component.

When the dependencies are provided through field injection on the member variables, we have to tell the Dagger to scan this class through the implementation of this interface.

module: Contains some modules to provide dependency and are responsible for providing objects which can be injected.

and ViewModelKey and Info Files: Using ViewModelKey we create an annotation class that Dagger will use to create the key of the map. Each info file is a qualifier which helps the dagger to distinguish between objects of the same type but with different instances in the dependency graph. For example DatabaseInfo is used to provide the database name in the class dependency. Since a String class is being provided as a dependency, it always a good idea to qualify it so that the Dagger can explicitly resolve it.

Factory Package contains ViewModelProviderFactory to instantiate ViewModels in a lifecycle-aware way.

Ui package has the following packages:

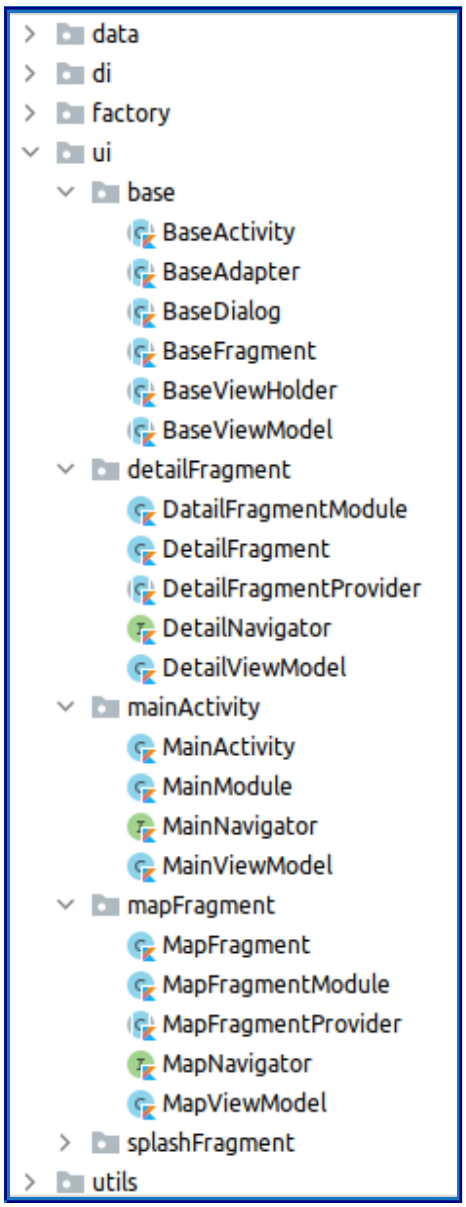
base: It contains some base classes that have been designed in such a way that it could be inherited and maximize the code reuse.

mainActivity: Includes all UI classes related to the app's single activity(MainActivity).

splahFragment, mapFragment and detailFragment: Includes all UI classes related to the fragments. Although it is a simple project, its structure ready to be extended and by using a module for each fragment and providing their dependencies to be injected.

Each package also contains navigator interface to use WeakReference to avoid a memory leak . A weak reference is a reference not strong enough to keep the object in memory. If we try to determine if the object is strongly referenced and it happened to be through WeakReferences, the object will be garbage-collected.

Each viewModel in packages is a bridge between the View(Activity or Fragment) and Model(business logic). The ViewModel should not be aware of the view who is interacting with. It interacts with the Model and exposes the observable that can be observed by the View.



Utils package has the following package and files:

r**x**: Contains schedulerProvider to make the code testable..

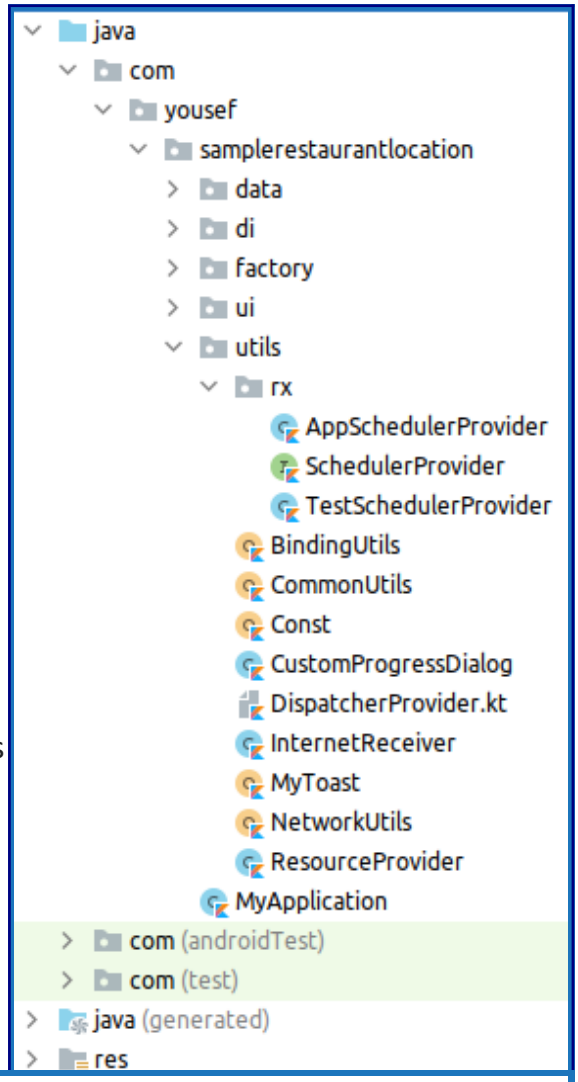
NetworkUtils: Handles the status of Internet connection.

BindingUtils: Handles the data biding and to make it to use Data Binding library to bind UI components in layouts to data sources in the app.

DispatcherProvider: Provides CoroutineDispatcher which performs both immediate and lazy execution of coroutines in tests and implements DelayController to control its virtual clock.

MyToast: My customized Toast.

Const: App constants.



I wrote Unit Test on **MapFragmentViewModel** and **DetailFragmentViewModel**.
I these tests I tested both valid and invalid restaurant Location Api calls and also getting details data related to specific restaurant.