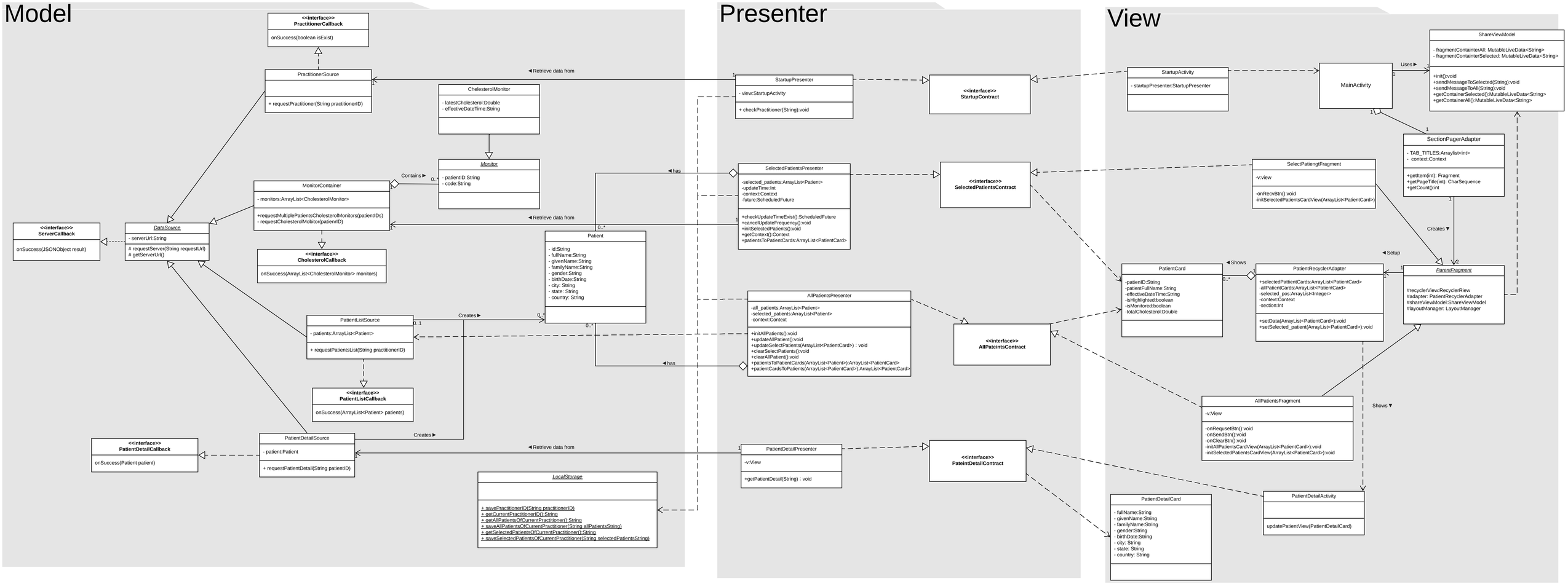
Model
Presenter
View
<<interface>>
PractitionerCallback
onSuccess(boolean isExist)
PractitionerSource
+ requestPractitioner(String practitionerID)
◄Retrieve data from
ChelesterolMonitor
- latestCholesterol:Double
- effectiveDateTime:String
Monitor
- patientID:String
- code:String
Contains►
0..*
1
MonitorContainer
- monitors:ArrayList<CholesterolMonitor>
+requestMultiplePatientsCholesterolMonitors(patientIDs)
- requestCholesterolMobitor(patientID)
◄Retrieve data from
<<interface>>
CholesterolCallback
onSuccess(ArrayList<CholesterolMonitor> monitors)
<<interface>>
ServerCallback
onSuccess(JSONObject result)
DataSource
- serverUrl:String
requestServer(String requestUrl)
getServerUrl()
Patient
- id:String
- fullName:String
- givenName:String
- familyName:String
- gender:String
- birthDate:String
- city: String
- state: String
- country: String
PatientListSource
- patients:ArrayList<Patient>
+ requestPatientsList(String practitionerID)
Creates►
0..1
0..*
<<interface>>
PatientListCallback
onSuccess(ArrayList<Patient> patients)
PatientDetailSource
- patient:Patient
+ requestPatientDetail(String patientID)
Creates►
<<interface>>
PatientDetailCallback
onSuccess(Patient patient)
◄Retrieve data from
LocalStorage
+ savePractitionerID(String practitionerID)
+ getCurrentPractitionerID():String
+ getAllPatientsOfCurrentPractitioner():String
+ saveAllPatientsOfCurrentPractitioner(String allPatientsString)
+ getSelectedPatientsOfCurrentPractitioner():String
+ saveSelectedPatientsOfCurrentPractitioner(String selectedPatientsString)
StartupPresenter
- view:StartupActivity
+ checkPractitioner(String):void
<<interface>>
StartupContract
SelectedPatientsPresenter
-selected_patients:ArrayList<Patient>
-updateTime:Int
-context:Context
-future:ScheduledFuture
+checkUpdateTimeExist():ScheduledFuture
+cancelUpdateFrequency():void
+initSelectedPatients():void
+getContext():Context
+patientsToPatientCards:ArrayList<PatientCard>
◄has
<<interface>>
SelectedPatientsContract
AllPatientsPresenter
-all_patients:ArrayList<Patient>
-selected_patients:ArrayList<Patient>
-context:Context
+initAllPatients():void
+updateAllPatient():void
+updateSelectPatients(ArrayList<PatientCard>) : void
+clearSelectPatients():void
+clearAllPatient():void
+patientsToPatientCards(ArrayList<Patient>):ArrayList<PatientCard>
+patientCardsToPatients(ArrayList<PatientCard>):ArrayList<PatientCard>
◄has
<<interface>>
AllPateintsContract
PatientDetailPresenter
-v:View
+getPatientDetail(String) : void
<<interface>>
PateintDetailContract
StartupActivity
- startupPresenter:StartupPresenter
MainActivity
Uses►
ShareViewModel
- fragmentContainerAll: MutableLiveData<String>
- fragmentContainerSelected: MutableLiveData<String>
+init():void
+sendMessageToSelected(String):void
+sendMessageToAll(String):void
+getContainerSelected():MutableLiveData<String>
+getContainerAll():MutableLiveData<String>
SectionPagerAdapter
- TAB_TITLES:Arraylist<int>
- context:Context
+getItem(int): Fragment
+getPageTitle(int): CharSequence
+getCount():int
SelectPatientgtFragment
-v:view
-onRecvBtn():void
-initSelectedPatientsCardView(ArrayList<PatientCard>)
Creates▼
◄Setup
ParentFragment
#recyclerView:RecyclerRiew
#adapter: PatientRecyclerAdapter
#shareViewModel:ShareViewModel
#layoutManager: LayoutManager
◄Shows
PatientCard
-patienID:String
-patientFullName:String
-effectiveDateTime:String
-isHighlighted:boolean
-isMonitored:boolean
-totalCholesterol:Double
PatientRecyclerAdapter
+selectedPattientCards:ArrayList<PatientCard>
-allPatientCards:ArrayList<PatientCard>
-selected_pos:ArrayList<Integer>
-context:Context
-section:Int
+setData(ArrayList<PatientCard>):void
+setSelected_patient(ArrayList<PatientCard>):void
AllPatientsFragment
-v:View
-onRequsetBtn():void
-onSendBtn():void
-onClearBtn():void
-initAllPatientsCardView(ArrayList<PatientCard>):void
-initSelectedPatientsCardView(ArrayList<PatientCard>):void
Shows▼
PatientDetailCard
- fullName:String
- givenName:String
- familyName:String
- gender:String
- birthDate:String
- city: String
- state: String
- country: String
PatientDetailActivity
updatePatientView(PatientDetailCard)

Before heading into the rationale, we made some assumptions

1. The practitioner enter his/her identifier to get all his/her patients
    a. So for the whole system, we only use practitioner identifier to query data instead of using id
2. The patient may have new cholesterol observation come out after the practitioner enter his/her identifier
    a. So we request server for all the selected patients to get their observation data every N seconds

Rational

As we developing the system in an Android App. In general, we implemented **MVP design pattern** which is a derivation of MVC.

In more specific for the Model package, the Abstract class *DataSource* provide **hinge point**, the method which responsible to request JSON object data from server. In order to separate different request queries(Observations,Practitioner, Encounter and Patient) we have four child classes that inherit DataSource and do their own data processing, each of them also implements their own interface(callback) to send back the processed request result to the presenter. This applies **the Interface Segregation Principle** that removes the coupling between different request queries. If we want both blood pressure and cholesterol than only cholesterol data, we can just do the modification in MonitorContainer without touching other queries classes, it increases the readability and maintainability of the code. On the other hand, when new requirement of the new query come in, we can just add a new class to inherit DataSource without touching other request queries, so this also applies the **Open Closed Principle**. However, the drawback is also obvious that we need to create a new interface whenever new query comes in, which makes the program bigger.

Regarding to **Adapter Design Pattern**, all the JSON data processing are within these classes(MonitorContainer, PatientDeatilSource, PatientListSource, PractitionerSource) before sending to the Presenter. These classes represent the role of Adapter that convert the JSON data into corresponding format needed for the presenter, it means that if there are some data format or structure changes from server, we can just modify codes in adapter, so this applies **Single Responsibility Principle** that separate data conversion from the business logic of the program but it also increase the overall complexity of the program as sometimes we can just simply put the data conversion code in presenter if it won't be used frequently.

Back on **MVP design pattern**, as it separate the View from Model which is different from MVC, the circular dependencies are introduced as View and Presenter both hold each other, which creates a high coupling. **Dependency injection** comes to help, with the addition of Contract(Interface), both of them now depend on the interface not the concrete class, it improves the testability and maintainability and the structure becomes more systematic with the use of the **Dependency Inversion Principle.** However, there is still some drawback by using MVP, as more requirements come in, most of time we need to create corresponding classes to comply with MVP, so the increase of classes will cost more compile resources.

Reference:

https://medium.com/cr8resume/make-you-hand-dirty-with-mvp-model-view-presenter-eab5b5c16e42
https://android.jlelse.eu/android-mvp-architecture-with-dependency-injection-dee43fe47af0