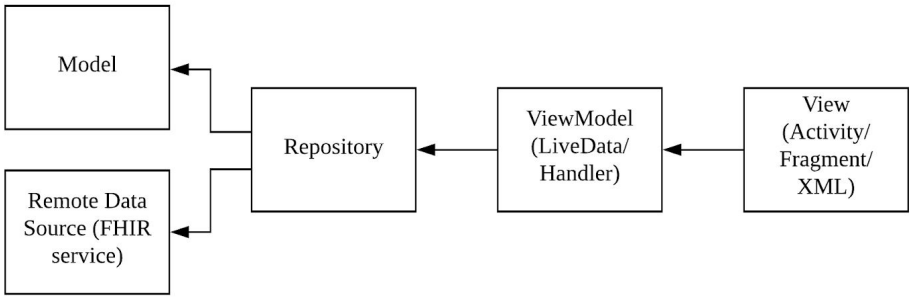
FIT3077 Assignment 2
Design rationale
Written by: Megan Ooi Jie Yi (30101670) & Hew Ye Zea (29035546)

MVVM (Model-View-ViewModel) architecture



We have chosen the MVVM layered software architecture for our system as it is the recommended architectural style for Android development[1]. Furthermore, since each component only relies on the component to its left (refer to diagram), this facilitates **separation of concerns**, introduces low coupling and makes our system modular so that the development of the frontend can be separated from the business logic/ backend. Each component has their own well defined responsibilities [1] to fulfill; View is responsible for UI logic, ViewModel for the view's display logic and Model to represent objects in the system. Similar reusable classes are grouped together into a package, obeying the **RREP principle** to increase reusability of classes.

One disadvantage of implementing MVVM is that it makes the system more complex and difficult for beginners to learn as opposed to a single system approach. However, MVVM provides a far more extensible solution because each component is clearly separated from each other and code becomes easier to maintain.

Shared ViewModel

Based on Android's documentation[2], it is recommended to use a SharedViewModel class to share data between fragments. Since the HomeFragment, SettingsFragment, and SelectPatientsFragment all share common data, the SharedViewModel class serves as a communicator between fragments, encapsulating the data and returning required information.

Since UI components such as fragments change very frequently, this is beneficial as it ensures **separation of concerns** so that fragments in our system do not know about each other and only communicate through a shared channel which is the SharedViewModel. Each fragment has its own lifecycle and is unaffected by other fragments[2]. One disadvantage of this approach is that the SharedViewModel class could get bloated very easily and violate the **Single Responsibility Principle**.

Repository pattern

The repository module encapsulates the logic required to get data from the server, maps the data to a model and provides a centralized common data access[3] for the ViewModel module. This pattern is important as it provides abstraction and allows more features to be added (such as getting a different type of data from the server) or changes in the repository module without affecting the data source or frontend layers.

Observer pattern

We have implemented the observer pattern to ensure that our UI is always displaying the latest information whenever an asynchronous call to the server is made. The advantage of this approach is that classes that are dependent on the 'Subject' are updated automatically and are allowed to provide their own implementation when they are notified of change.

The SharedViewModel class serves as the Subject by encapsulating PatientModel objects in Android's **LiveData** component. This class implements the polling method from the Poll class and uses a Handler to fetch data asynchronously in a specified time interval. It notifies its observers in UI every N seconds by using

LiveData's `post` method. LiveData is used as it is an observable data holder class that is lifecycle-aware allowing only active components to be notified when there is a change in data[4].

The Fragments that are responsible for displaying data from the SharedViewModel class acts as an observer and subscribes to the LiveData object. The Fragment updates its view accordingly to display the current observation readings whenever it is notified of changes.

Adapter pattern
In order to display data in our UI, we have implemented the adapter pattern for the HomeAdapter and SelectPatientsAdapter. These classes take the list of data that we would like to display and convert it to a list of views to be shown on the screen. The adapter class is responsible for reformatting and allows reuse of classes that previously would have been incompatible without the adapter class.

Open/Closed Principle
The ObservationModel class provides an interface for different observation types in the future. The class is open for extension[5] whereby different observations are required to provide their own implementation and functionality but the ObservationModel class is closed from modification. This is beneficial as it provides a layer of abstraction and ensures **loose coupling**. Likewise, similar methods have been abstracted into the BaseAdapter class so that its subclasses can extend its superclass and provide its own implementation for displaying the view in our UI.

References
[1] Android Open Source Project. (n.d.). Recommended app architecture. Retrieved from Guide to app architecture.
[2] Android Open Source Project. (n.d.). ViewModel overview. Retrieved from ViewModel Overview.
[3] Youssef, V., John, P., Maira. W. & Cedric, M. (2018). Design the Infrastructure Persistence Layer. Retrieved from Designing the infrastructure persistence layer.
[4] Android Open Source Project. (n.d.). LiveData Overview. Retrieved from LiveData Overview
[5] Meyer, B. (1988). *Object-Oriented Software Construction*. Prentice Hall.