**3077 Assignment 2: Quarantine Coders Design Rationale**

**SDK and background information:**

We decided to create our cholesterol application using python 3, so that we were able to create modules that could easily interact with one another, and so that we could design a practical graphical user interface using the in-built python library, TKInter.

**Design Rationale:**

When designing our application, it was imperative that our system had minimal dependencies, so that our high-level modules could be easily reused and unaffected by changes in low level modules, which provide utility features. To do this we implemented forms of abstraction that separated these modules. It was also of high importance that these abstraction modules did not rely in the high- or low-level modules in any way. So, to maintain this, we used the Dependency Inversion Principle (DIP) *(FIT3077 Software Engineering: Architecture and Design, Lecture 4)*. This is depicted in our model with the FHIRclient class, and how it creates a patientList object. The Data is obtained from the CholesterolDataClient class, which creates an object of PatientData, where each patient within the PatientList obtains the cholesterol information corresponding to their name. In doing this, it ensures that the PatientList does not have associations with the CholesterolDataClient class itself, thus inverting the dependency while being able to use the class for its data. Also, this allows the CholesterolDataClient class to orchestrate the PatientList class without having to make many changes to it, if the system were to develop later, which saves us time in the future.

Our design also uses the Liskov Substitution Principle (LSP), which inevitably means that child/sub classes should be able to extend their parent/base class without having to change their behavior, thus can be used rather than the parent/base class *(FIT3077 Software Engineering: Architecture and Design, Lecture 4)*. This can be seen with the FHIRClient class, which is the base class, and the CholesterolDataClient class which is the child/sub class. By doing this, the classes become less dependent on each other and encouraged us to reuse our methods in the base class, rather than creating a duplicate method. Within our model, it illustrates that there would not be any issues if the FHIRClient class was replaced with an instance of CholesterolDataClient. On top of this, we were able to inherit methods from the FHIRClient class, meaning we did not have to rewrite the same method in the CholesterolDataClient class. The main reason for this is so the getPatientData() method would act as an abstract method so that other low-level modules wouldn’t depend heavily on the FHIRClient class. The creation of the CholesterolDataClient class which was strictly used for returning the patient data, which allowed for less interactions with the FHIRclient class.

Another principle we decided to adopt for our design was the Open-Closed principle (OCP). This was because we wanted to add new functionality to our application without changing the existing code *(FIT3077 Software Engineering: Architecture and Design, Lecture 4)*. This prevented situations where we would have to change all other classes that depended on the existing code that was to change. This can be seen with our abstract PatientData class which contains the data specific for each patient linked to a certain health practitioner. This class could be extended if the specifications of the tasks were to change, and we would not have to change other classes a large amount that depend on it, such as the Patient class. All we would need to do is call the extension on the PatientData class in the dependant class so that it can use the new methods/extensions. This implementation saves us time and effort, instead of having to change every module. This principle can also be seen associated with the Person class, which acts as an abstract class for the Patient and HealthPractitioner class. As these two subclasses differ in many ways, if they were to have added functionality later on, the Person class can be extended, and the subclasses can obtain the added extensions if need be.