FIT3077 Assignment 2

Quarantine Coders Design Rationale

SDK and background information:

Our team made the decision to implement the cholesterol monitoring application using Python 3. This is because Python comes with the requests package, which allows us to make API calls easily. In addition, Python also has an inbuilt library for designing a graphical user interface, TkInter, making it a very suitable choice for this project.

Design Rationale:

In designing our application, our main focus was to minimise dependencies and maximise extensibility and maintainability. To achieve this, we applied a number of different design principles for our system.

The first principle we will discuss is the Liskov Substitution Principle (LSP), which states that child/sub classes should be able to extend their parent/base class without having to change their behavior, and thus can be used in place of the parent/base class. This can be seen with the **FHIRClient** class, the base class, which is extended by **CholesterolDataClient** subclass. Within our model, it illustrates that there would not be any issues if the **FHIRClient** class was replaced with an instance of **CholesterolDataClient**. Common code for retrieving practitioner and patients' information are placed in the base class. The **get_patient_data** method is abstract to provide extensibility for future requirements of the system, as there may be other types of patient data other than cholesterol. In such cases, we can create another subclass of the **FHIRClient**, which provides its own separate concrete implementation of the **get_patient_data** method. This creates a hinge point in our design where new functionalities may be added to the app. The type of client used to retrieve patient data may also be changed during runtime (through the Application class) which provides greater flexibility in the system.

To maintain extensibility in the system, we applied the Dependency Inversion Principle (DIP). This is depicted in our model with the **FHIRClient** class, which creates a **PatientList** object. The PatientList class is a collection of patients, each having their own **PatientData** object. The data is obtained from the **CholesterolDataClient** class, and it maintains the information of each patient inside of the **PatientList**. Our design ensures that the **PatientList** does not have any direct association with the **CholesterolDataClient** class itself. This also allows the **CholesterolDataClient** as well as any new types of clients to be added to indirectly orchestrate the **PatientList** class, making the system more maintainable if it was to be changed in the future.

Another principle we decided to adopt for our design was the Open-Closed principle (OCP). This was because we aim to be able add new functionality to our application without modifying the existing system. This prevented situations where we would have to change all other classes that depended on the existing code that was to change. An example of this is the abstract **PatientData** class, which contains the data specific for each patient. This class could be extended if the specifications of the system were to change, for example, if a new type of data needed to be monitored (blood pressure/glucose levels). This can be achieved easily by extending the **PatientData** class and can provide its own concrete implementation to the abstract methods. Another instance where the OCP principle is applied is the **Person** class, which acts as an abstract class for the **Patient** and

HealthPractitioner classes. Having an abstract Person class allows us to extend the base functionalities for any new users of the system in the future.

Our design also utilizes the Observer design pattern, where the subject is the **PatientList**. Any time a change occurs to the patient data within our **PatientList**, its observers will be notified and updated. For the purpose of this assignment, there is currently only one type of concrete Observers, the **MonitorTreeview**, which is just a table which displays the list of monitored patient and their data. However, the Observer design pattern adds another hinge point to our design, where extension can easily occur in the future (addition of more observers for the data).

Class Diagram