Alert generation system

DataStorage is used by AlertGenerator to store each patient's unique data and thresholds to trigger an alert. AlertGenerator uses this storage to evaluate current incoming data such as the vitals, and if the threshold is crossed, it concludes that an alert is needed. This is all an automated and always running process, as the current vitals should always be measured. After AlertGenerator decides that an alert is needed, the alert itself is formed using Alert class, which stores and outputs the time of alert, condition, and patient's ID. This is then used by AlertRouter and Alert Manager to route the alert the necessary staff and send it, as well as store in logs, respectively.

AlertManager and AlertRouter are separated classes even though they do similar tasks, to be able to allow the alert to immediately reach the staff (the priority), then take care of logging without overwhelming it. Alert. While AlertManager is not the priority, it is still important to keep a track off. AlertGenerator itself does not actually generate alert, as storing the condition, timestamp and patientID alongside all the data handling would cause a slow programme and a confusing control flow, therefore it is also split.

Data Storage System

The different functions of the Data Storage System are broken down between classes, so that one is not responsible for too many things. PatientData records one medical record at a time, and then Patient links each individual patient to all of their medical records in a list. Database is the storage for all of the data, creating a map of all of the Patient objects. The Database class implements DataStorage, which is an interface that can be used by other classes also. There is a separate class DataDeletion, which takes care of deleting old records from the Database after a set time. DataRetriever is where queries by medical staff get handled. There are two methods - one that returns the records of a specific patient during a specific period of time, and one that retrieves all of the medical data available. This class is also linked to AccessControl, which verifies that whoever wants to access a patient's records is qualified to do that.

By giving each function to a different class, the following of the single responsibility principle is ensured. The inclusion of an interface also makes its functions more versatile.

Patient Identification system

Hospital staff such (those to whom it is relevant) can access the records through HospitalPatient and change them through PatientRecords. However they are 2 different classes, because accessing PatientRecords (changing them) needs higher authentication in management. PatientIdentifier can be used at eg. reception to match the records and find the correct patient usign the ID. IdentityManager is called automatically when a problem occurs in PatientIdentifier due to eg. no matches or more than 1 match. The match is returned through PatientIdentifier and accessed by the needed hostpital staff too.

The code functions so that PatientIdentifier uses HospitalPatient to match incoming data to the data in the hospital's database stored through DataStorage and PatientRecords, which can also add/delete records. IdentityManager handles any issues that may occur while matching identities, such as no matches or too many matches.

In this way, no class gets overloaded with tasks, but not too many classes perform the same task. DataStorage needs to be connected to PatientRecords to store the hospital's database. PatientRecords is implemented separately, as it has more details and performs higher level tasks than DataStorage. HospitalPatient creates an object of a patient including the records, so that other classes can use it such as IdentityManager.

Data Access Layer

The purpose of the Data Access Layer is to link external data sources, more specifically TCP, WebSocket and files, to the system. The different listeners for each data source type implement a common interface. This allows the code to be reused instead of rewritten. Also, if another data source is wanted to be added, it can be done so more easily. After the listeners collect raw data, they pass it onto the DataParser, which transforms it into a ParsedData object, which stores information like patient ID, measurement value, timestamp, etc. The DataSourceAdapter class serves as a bridge between the parsed data and the DataStorage class, where it is ultimately being stored.

One of the advantages of this design is its modularity, which ensures that changes will not affect parts of the code that do not need to be changed. Another advantage is the separation of concerns - each class has its own function, so there is no overlapping. The system is also testable, as the isolated components enable unit testing. Moreover, it has great reusability potential, as all the listeners use the same parser and the adapter works with any DataStorage implementation.