Alert generator system

The “HeartRateMonitor” and “BloodPressureCuff” are specific types of the “Device” class. This relationship is indicated by the dashed arrow. They inherit the “streamData” method from “Device” but both provide their own implementation (“streamBloodPressure()” and “streamHeartRate()” to handle specific types of data).

The “Device” class is directly associated with the “AlertGenerator” class. This means that the devices send data streams to the “AlertGenerator” class but do not receive anything back. The “AlertGenerator” receives this data and evaluates it against the patient’s threshold by using the “PatientPortal”(dependency relationship).

When the “AlertGenerator” class detects a value above/below the threshold, it creates an instance of the “Alert” class. Therefore the “Alert” and “AlertGenerator” class have a composition relationship and “Alert” instances do not exist independently of the “AlertGenerator”.

The “Alert” class is associated with the “PatientProfile”, by containg a patientID field, which links it to a specific “PatientProfile”. After an “Alert” object is created, it is passed to the “AlertManager”, thus the “Alert” objects are aggregated by “AlertManager”.

“AlertManager” has methods to manage and further send “Alert” objects to “MobileDevice” and “WorkStation”, but it does not own the “Alert” objects. The solid lines pointing to “MobileDevice” and “WorkStation” from the “AlertManager” imply directed association, meaning that the “AlertManager” sends alerts to these devices but does not receive anything back.

Data storage system

“DataStorage” is the core class responsible for handling the data operations. It includes methods like “storeData()” to add new patient data, “deleteData()” to remove old data, and “retrieveData()” to fetch existing data. These data operations concern the “PatientData” class, thus these classes are directly associated.

“PatientData” encapsultes a patients’s information, including a personal identifier, health metrics in a key-value map and a time stamp, representing the data records managed by the system.

The “DataRetriever” class acts like the mediator between data storage and the end-user. Therefore it is directly associate with both the “User” and “DataStorage” classes.It includes methods to “fetchData()” for a specific patient and “verifyAccess()” to ensure that data retrieval is secure and restricted to authorized personnel.

The “User” class represents people, typically medical staff, who interact with the patient data management system. The attributes “id”, “roles”)and method “hasAccessTo()” of the class are designed to identify the user within the system and

to determine their access rights to the patient data. This class is important for implementing security.

The “Policy” class features a method to determine whether patient data can be deleted, ensuring that all policies are followed. It is associated with “DataStorage” class.

Patient Identification System

“DataStream” provides incoming data and sends it to “PatientIdentifier”. When “PatientIdentifier” gets data from the “DataStream”, it is responsible for matching patient IDs with this data. Next the “IdentifyManager” ensures that the data is correct, handles any anomalies and provides integrity of patient records. “PatientRecords” represents any information about a patient and they are stored in the “HospitalDataBase”. “PatientRecords” does not exist without the “HospitalDataBase”, that is why it is connected with a special kind of an arrow (composition).

Data Access Layer

An “ExtertnalDataSource” sends data to the “DataListener” abstract class, that defines simple operation of listening provider data. “TCPDataListener”, “WebSocketDataListener”, and “FileDataListener” are concrete implementations of data listeners for specific data sources (“TCPDataListener” for IP addresses, “WebSocketDataListener” for endpoints, “FileDataListener” for file paths). “DataParser” is an abstract class, that defines the common operation for parsing data, which allow the standardisation. “DataSourceAdapter” takes care of processing the data from the parsers and is responsible for storing them. This data can be then send to some “InternalSystem” that retrieve and process it.