## UML class diagrams

### Alert generator system

ln designing the UML class diagram for the Data Storage System, several key choices and connections were made to accurately represent the system's requirements and functionality. Firstly, the central component of the system, the DataStorage class, was identified as responsible for managing historical patient data. To reflect this, the class was named appropriately and equipped with methods to support the fundamental operations of storing, retrieving, and deleting data. These methods—storeData, retrieveData, and deleteData—were chosen to align with the specified requirements and provide clear and concise functionality for interacting with the data storage system.

The PatientData class was identified as crucial for representing individual instances of patient data. It was designed to encapsulate attributes such as patientId, metrics, and timestamp, as outlined in the requirements. These attributes were chosen to accurately capture the essential information associated with each data point, facilitating efficient storage and retrieval within the system.

Regarding connections, an association was established between the DataStorage class and the PatientData class to represent the relationship between the data storage system and the patient data it manages. This association enables the DataStorage class to store instances of PatientData and retrieve them when needed.

The directionality of the association arrow indicates that the DataStorage class holds a reference to instances of PatientData, indicating a unidirectional relationship where DataStorage is aware of PatientData instances but not vice versa.

### Data Storage System

In designing the UML class diagram for the Data Storage system, several classes were created to represent key components and their interactions within the system architecture. The DataStorage class serves as the central component responsible for storing and managing patient data. It contains methods such as storeData(), retrieveData(), and deleteData() to handle data storage operations. The association between DataStorage and PatientData represents the relationship where the DataStorage class stores instances of PatientData.

PatientData represents the data for a patient at a specific point in time, including attributes such as patientId, metrics, and timestamp. This class encapsulates the individual data points associated with each patient.

The AlertGenerationSystem and PatientIdentificationSystem classes represent external systems that interact with the Data Storage system. These classes are associated with the DataStorage class, indicating their dependency on it for accessing patient data.

Additionally, the DataRetriever class is introduced to handle the retrieval of data from external sources or systems. It has a method fetchData() to fetch data and processData() to process it before storing it in the DataStorage. This class has an association with DataStorage, as it interacts with it to store the retrieved data.

Similarly, the DataProcessor class is designed to process data stored in the DataStorage system. It has a method process() to perform various data processing tasks. This class also

has an association with DataStorage, indicating its dependency on it for accessing patient data.

Each class and their relationships were chosen to reflect the modular structure of the Data Storage system and to facilitate efficient data management and processing. The associations between classes illustrate the flow of data and the dependencies between different components within the system, ensuring seamless interaction and functionality.

## Patient Identification System

In the Patient Identifier UML class diagram, key classes collaborate to ensure the accurate matching of incoming patient data with existing records. At the center of this system is the PatientIdentifier class, pivotal for aligning patient IDs from incoming data with stored records. It heavily relies on the DataStorage class to execute essential operations like matching, retrieval, addition, and updating of patient records. The Dependency link between PatientIdentifier and DataStorage underscores the former's dependence on the latter for efficient data management.

Complementing the patient identification process is the IdentityManager class, responsible for overseeing identity resolution and logging discrepancies. Like PatientIdentifier, IdentityManager relies on DataStorage for accessing patient records, forming another Dependency relationship. These dependencies highlight the crucial role of DataStorage in facilitating data management functions essential for patient identification.

To bolster the system's functionality, supplementary classes come into play. DataManager handles the processing of incoming patient data, while AlertGenerator monitors real-time data and triggers alerts for specific conditions. Both classes rely on DataStorage for data access, emphasizing their interdependence with the core data management component.

These Dependency relationships underline the integrated nature of the system, where various components collaborate to ensure accurate patient identification and data management.

## Data Access Layer

At the core of the system lies the DataListener class, which serves as an abstract base class for specific data listener implementations. The DataListener class defines essential methods for listening to incoming data, parsing it, and processing it within the system.

Derived from the DataListener class are the TCPDataListener, WebSocketDataListener, and FileDataListener classes, each tailored to handle data from specific sources. The choice to subclass DataListener allows for a modular and extensible design, enabling the system to accommodate different communication protocols seamlessly.

The TCPDataListener class manages communication over TCP/IP connections, utilizing attributes such as ipAddress and port to establish and maintain connections. Similarly, WebSocketDataListener handles data from WebSocket connections, utilizing attributes like url and protocol to manage connection details. FileDataListener, on the other hand, reads data from local files, with filePath and fileType attributes defining the file's location and type.

The DataParser class, responsible for standardizing raw data into a uniform format, collaborates closely with the DataListener classes. Once data is received and parsed, it is

passed to the DataSourceAdapter for final processing and storage. This separation of concerns ensures modularity and allows for independent development and testing of data processing logic.
Overall, the design choices made in the UML class diagram prioritize flexibility, modularity, and scalability, enabling the Cardiovascular Health Monitoring System to adapt to diverse data sources and communication protocols while maintaining robustness and efficiency.

## UML state diagram

The UML State Diagram for the Alert Generation System encapsulates the lifecycle of alerts, outlining the series of states and transitions that occur from the moment an alert is generated to its resolution. Each state represents a distinct stage in the alert process, while transitions depict the actions or events that trigger the progression from one state to another.
The choice of states—Generated, Sent, Acknowledged, and Resolved—was made to reflect the essential phases of an alert's journey within the system. The Generated state signifies the initial creation of an alert by the system based on predefined criteria, while the Sent state represents the transmission of the alert to medical staff for notification. Upon receiving the alert, medical staff either manually or automatically acknowledge it, transitioning it to the Acknowledged state. Finally, the Resolved state indicates the conclusion of the alert, signifying that the underlying condition triggering the alert has been addressed.
Each transition in the diagram corresponds to a specific action or event that triggers the movement between states. For example, the transition from Generated to Sent occurs when the system sends the alert to medical staff, while the transition from Acknowledged to Resolved happens when the alert is resolved, either manually by medical staff or automatically by the system.

## UML sequence diagram

The UML sequence diagram illustrates the alert generation process within the Alert Generation System. The sequence begins with the Patient actor detecting a vital sign exceeding a predefined threshold. Upon detection, the AlertGenerator component receives the vital sign data and interacts with the DataStorage component to retrieve historical data relevant to the patient's condition.
The involvement of the DataStorage component ensures the accuracy of the alert generation process by validating the current vital sign reading against past trends. Following the retrieval of historical data, the AlertGenerator analyzes the current and historical data to determine if the condition threshold has been met.
If the threshold is exceeded, the AlertGenerator proceeds to create an Alert, which is then transmitted to the Nurse actor for immediate attention. The medical staff acknowledges the alert and takes appropriate actions to address the patient's condition. After resolving the situation, the alarm shuts off.