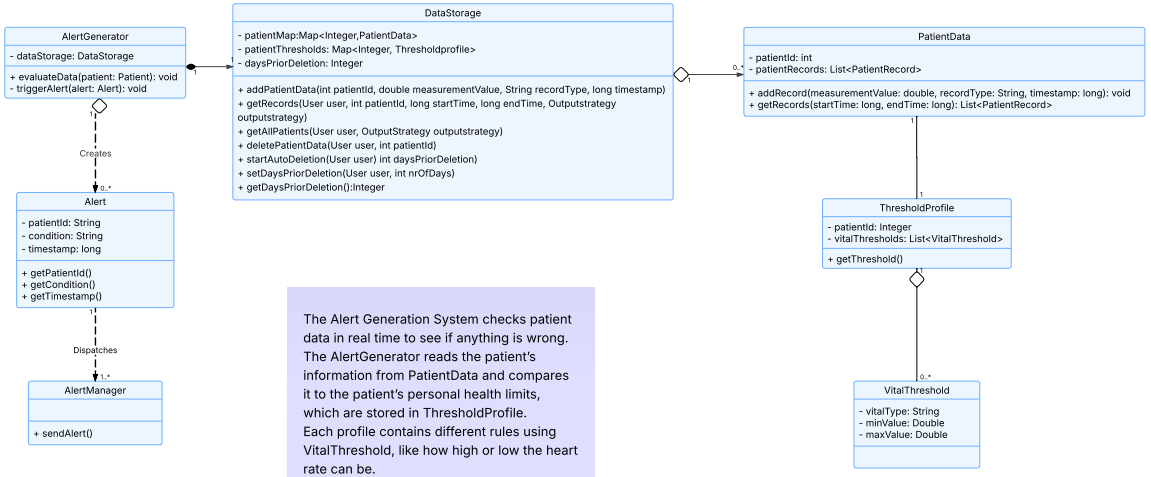
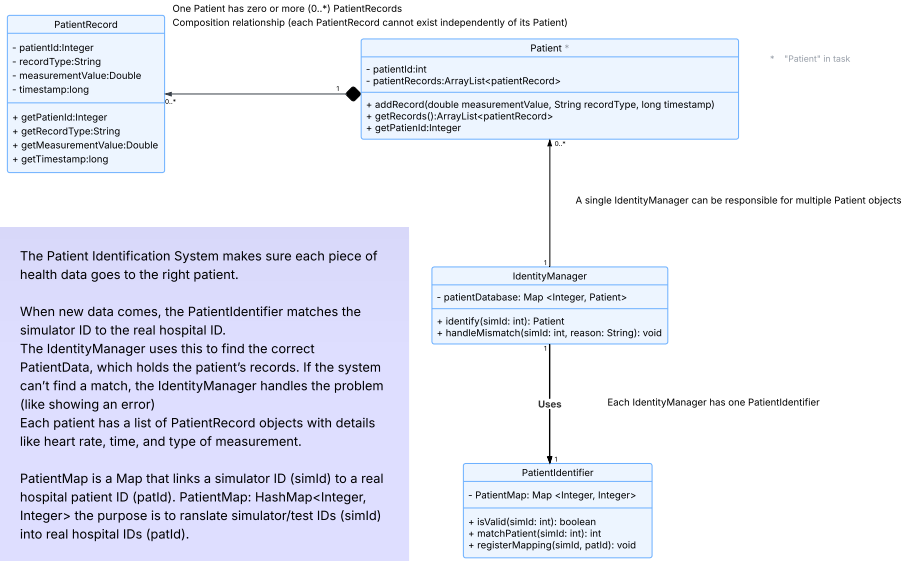
ALERT GENERATION SYSTEM



The Alert Generation System checks patient data in real time to see if anything is wrong. The AlertGenerator reads the patient's information from PatientData and compares it to the patient's personal health limits, which are stored in ThresholdProfile. Each profile contains different rules using VitalThreshold, like how high or low the heart rate can be.
If something is not normal, the AlertGenerator creates an Alert with the patient's ID, what went wrong, and the time it happened. Then the alert is sent to doctors or nurses using the AlertManager.

PATIENT IDENTIFICATION SYSTEM



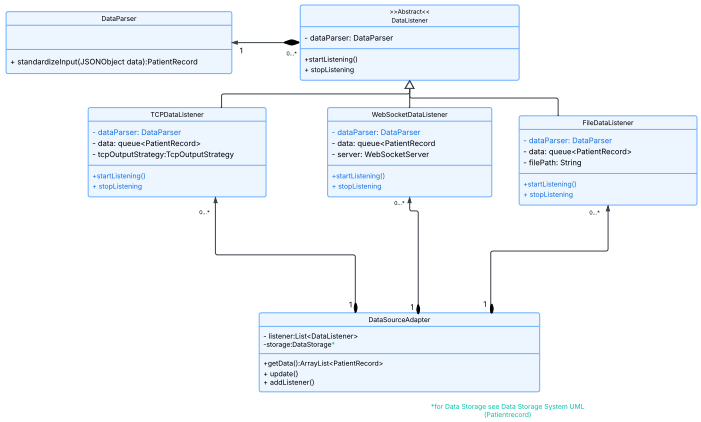
The Patient Identification System makes sure each piece of health data goes to the right patient.

When new data comes, the PatientIdentifier matches the simulator ID to the real hospital ID.
The IdentityManager uses this to find the correct PatientData, which holds the patient's records. If the system can't find a match, the IdentityManager handles the problem (like showing an error)
Each patient has a list of PatientRecord objects with details like heart rate, time, and type of measurement.

PatientMap is a Map that links a simulator ID (simId) to a real hospital patient ID (patId). PatientMap: HashMap<Integer, Integer> the purpose is to ranslate simulator/test IDs (simId) into real hospital IDs (patId).

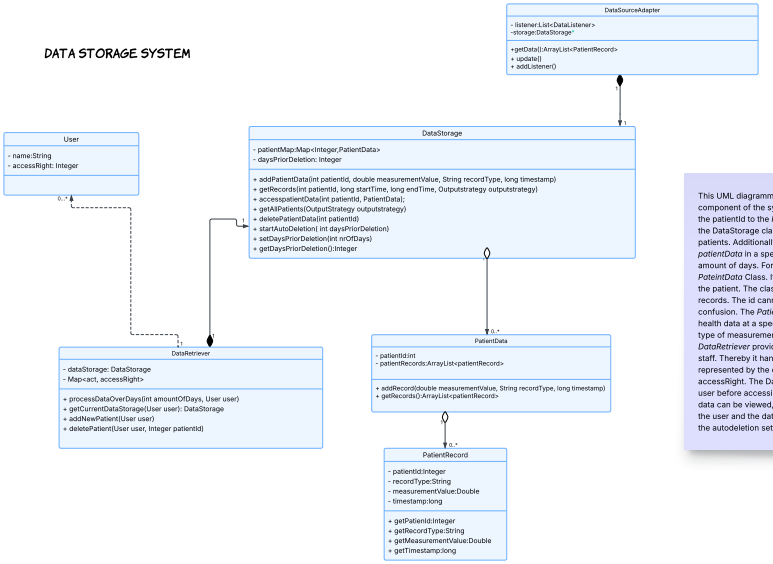
patientDatabase: Map<Integer, Patient> the purpose is to store the actual Patient records using the hospital ID (patId) as the key.

DATA ACCESS LAYER



The UML diagram represents the Data Access Layer of a software system. It is responsible for processing and managing incoming patient data from various sources. The DataListener classes are used to receive the data and temporarily save it as Patientrecord.
To enable the processing of different types of data, there is an abstract class which can be inherited by different types of data listeners. Currently, classes inheriting from the abstract class are TCPDataListener, WebSocketDataListener and File DataListener. Depending on the type of dataListener the source of the data differs. Hence, the subclasses of the DataListener class have different private fields depending on the type of data. They all have the methods startListening and stopListening to enable real time data processing. Inside the startListening method, the listeners use a DataParser to turn the data into patient records.
The parser is working with JSON objects. That way different the processing of the data is kept inside the class, where it is currently stored in. However, adaptations to the parsing (in case the structure of the PatientRecords change) can still easily be done via the dataParser. Finally, the DataSourceAdapter receives the patientrecords of the dataListeners and adds them to the storage. One DataSourceAdpater can have multiple DataListeners. They are stored in an ArrayList.

DATA STORAGE SYSTEM



This UML diagramm shows a medical storage system. The core component of the system is the *DataStorage* class. It connects the patientId to the *PatientData* using a map. Several methods in the DataStorage class enable access to add and delete data of patients. Additionally, some modified methods allow access to *patientData* in a specific time range and auto-deletion after a set amount of days. For each patient, there exists an object of the *PateintData* Class. It holds the patientId and the *patientrecords* of the patient. The class contains getter and setter methods for the records. The id cannot be changed once it was set to avoid confusion. The *PatientRecord* class is used to store a patients health data at a specific time. Each record holds the patientId, the type of measurement, the value and a timestmap. The class *DataRetriever* provides access to the dataStorage for the medical staff. Thereby it handles access rights: Each staff member is represented by the class *User* with the private fields name and accessRight. The DataRetriever checks the access rights of the user before accessing the DataStorage. Via the DataRetriever, data can be viewed, deleted and added. As it is the link between the user and the database, it also provides the option to change the autodeletion setting of the *DataStorage*.