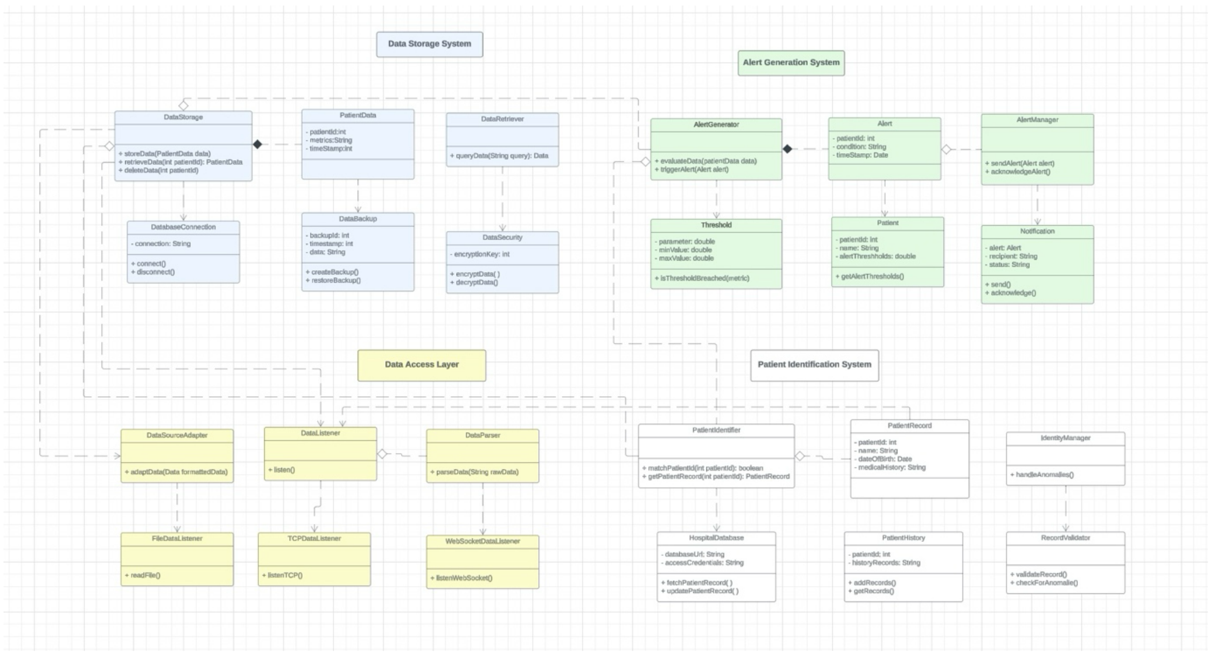
Tutorial week 2: UML Diagrams



1. Alert Generation System

The Alert Generation System is designed to keep an eye on patient data and send alerts when something is wrong. The main class here is **AlertGenerator**. This class is constantly checking the data coming in to see if any of it is abnormal. It does this by using the **Threshold** class, which has information on what counts as normal and what counts as abnormal for different metrics (like heart rate or blood pressure). Each patient has their own set of thresholds stored in the **Patient** class. If the data shows something abnormal, the **AlertGenerator** creates an **Alert** object. This object has important details like the patient's ID, what condition triggered the alert, and when it happened. The **AlertManager** then takes this alert and makes sure it gets to the right medical staff by creating and sending **Notification** objects. These notifications might show up on the medical staff's screens or mobile devices. This system ensures that alerts are specific to each patient and are sent quickly so that the medical team can respond right away. By having a system that constantly monitors and sends out alerts, the hospital staff can stay on top of any changes in a patient's condition and act quickly to provide the necessary care.

2. Data Storage System

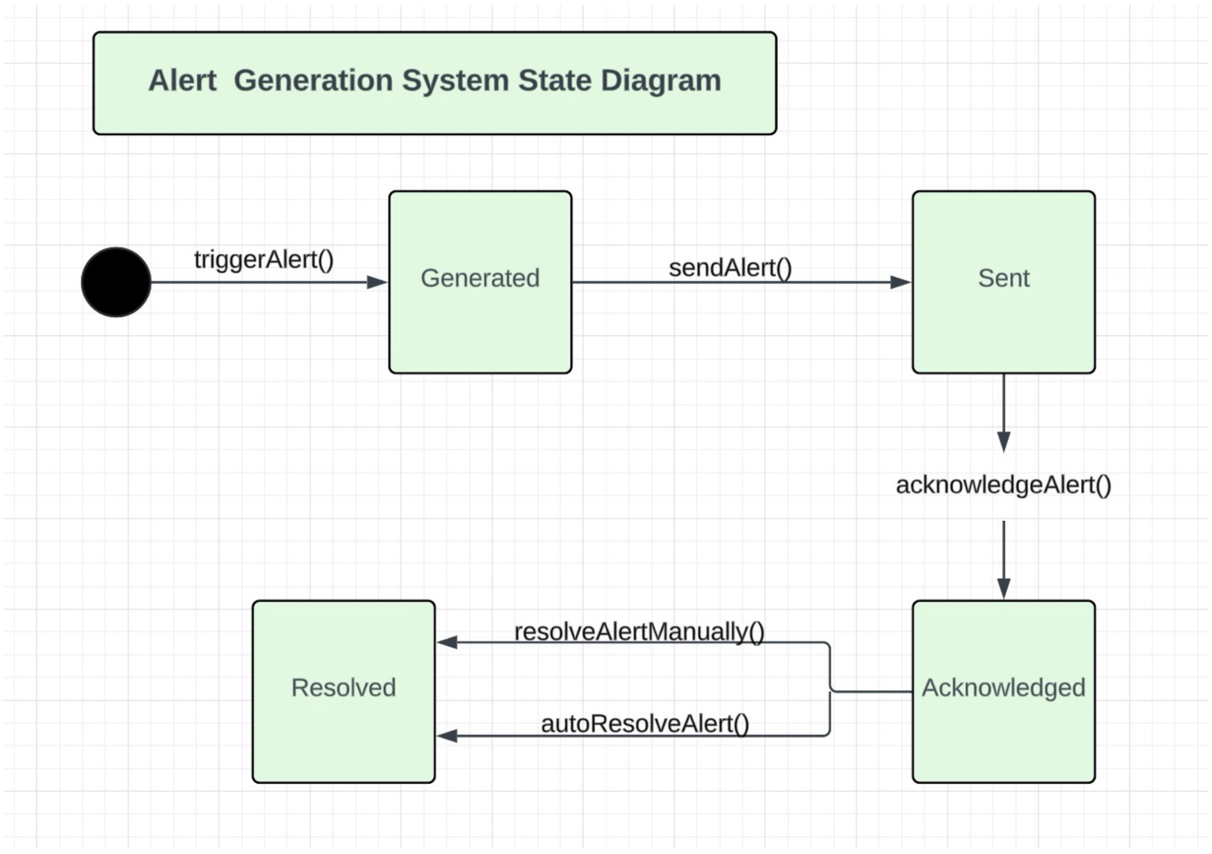
The Data Storage System is crucial for keeping all the patient data organized and secure. The **DataStorage** class is responsible for storing, retrieving, and deleting patient data. **PatientData** objects are used to store information about each patient, including their metrics (like heart rate and blood pressure) and the time these metrics were recorded. This makes it easy to keep track of how a patient's condition changes over time. When medical staff need to look up a patient's information, they use the **DataRetriever** class, which fetches the necessary data from the storage. Security is a big deal in this system, which is why the **DataSecurity** class is used to encrypt and decrypt the data, ensuring that only authorized people can access it. The **DatabaseConnection** class manages the connection to the database, making sure that data can be accessed quickly and efficiently. **DataBackup** is another important class, as it creates backups of the data and can restore it if anything goes wrong. This system ensures that all patient data is stored safely, can be retrieved easily, and is protected from unauthorized access. It supports both real-time monitoring and long-term analysis of patient data, which is essential for providing continuous and effective care.

3. Patient Identification System

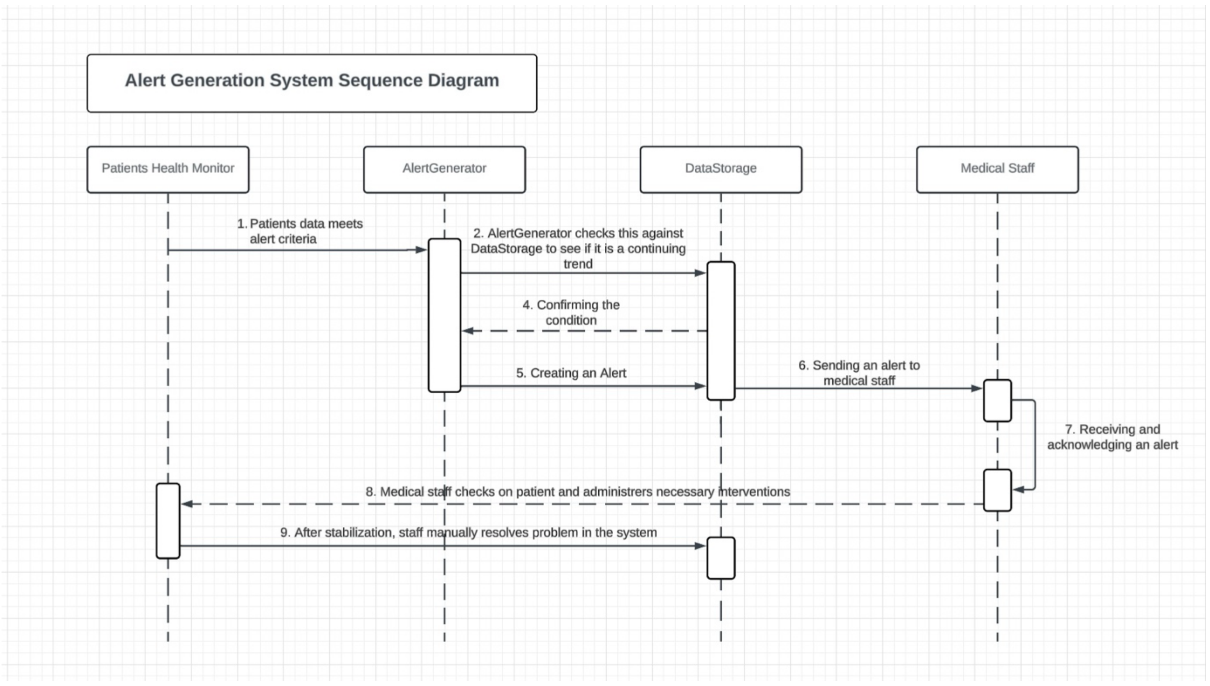
The Patient Identification System makes sure that the data coming in from the various sources is correctly matched to the right patient records. The **PatientIdentifier** class plays an important role in this process. It matches patient IDs from the incoming data with the IDs in the hospital's database. **PatientRecord** objects hold detailed information about each patient, such as their personal details and medical history. The **IdentityManager** oversees the whole process, handling any issues that come up during the identification. The **HospitalDatabase** class is used to fetch and update patient records from the main hospital database. It connects to the database and makes sure the system has the most up to date information. **PatientHistory** keeps a log of all medical events for each patient, making it easy to see a complete picture of their health over time. **RecordValidator** checks the records for any errors or inconsistencies, ensuring that the data is accurate and reliable. This system prevents mistakes in patient identification, which is crucial for making sure that the right treatments are given to the right patients. By accurately matching data to patient records, the system helps maintain the integrity of patient information and supports better decision-making by the medical staff.

4. Data Access Layer

The Data Access Layer is what connects the cardiovascular health monitoring system to various data sources. The **DataListener** class acts as a generic interface for receiving data from different sources. There are specialized listener classes like **TCPDataListener**, **WebSocketDataListener**, and **FileDataListener**, each designed to handle data from the specific types of sources. For example, **TCPDataListener** listens for data coming through TCP connections, **WebSocketDataListener** handles data from the WebSocket connections, and **FileDataListener** reads data from files. Once the data is received, it is sent to the **DataParser** class, which converts the raw data into a standard format that the system can use. This parsed data is then sent to the **DataSourceAdapter**, which is processing and storing it. This design makes the system very flexible and scalable because it can easily adapt to handle different types of data inputs. Whether the data is coming from real-time monitoring devices, log files, or other sources, the system can process it efficiently. This layer ensures that all the incoming data is properly formatted, processed, and stored, allowing the health monitoring system to function smoothly and effectively. By being able to handle data from multiple sources, the system can be integrated with real world patient monitoring devices.



The Alert Generator System State Diagram represents the states and transitions that happen in the system. The main goal of this system is to ensure timely detection of distress or critical medical conditions in patients by continuously evaluating incoming data and triggering alerts when it is necessary.
The first state in which the alert finds itself, is waiting for incoming patient data to evaluate. Once the data is evaluated and meets the criteria for generating an alert, which may include elevated heart rate or irregular blood pressure levels, the system transitions to the Generated state. In this state the Alert Generator class sends an alert containing critical information, including the patientsID, the condition that triggered the alert and a timestamp. The Alert Generator transitions to the Acknowledged state. When the medical staff acknowledge the receipt of an alert, either manually or automatically, the Alert Generator transitions to the Acknowledged state. Finally, alerts are resolved through either automatic detection or manual confirmation by medical staff after assessing and potentially adjusting treatment. The Alert Generator transitions into the Resolved state.
This visual representation offers a comprehensive understanding of how alerts are managed within the system, facilitating timely responses to critical medical conditions and ultimately enhancing patient care and safety, by ensuring effective communication and intervention.


Alert Generation System Sequence Diagram
Patients Health Monitor
AlertGenerator
DataStorage
Medical Staff
1. Patients data meets alert criteria
2. AlertGenerator checks this against DataStorage to see if it is a continuing trend
4. Confirming the condition
5. Creating an Alert
6. Sending an alert to medical staff
7. Receiving and acknowledging an alert
8. Medical staff checks on patient and administrers necessary interventions
9. After stabilization, staff manually resolves problem in the system

Sequence diagram of an Alert Generation System represents flow of interactions between the key components (Patient Health Monitor, Alert Generator, Data Storage and Medical Staff) in response to the detecting and handling of an alert triggered by Patient Health Monitor (for example too fast heart rate).
First, the Patient Health Monitor detects the stimulus triggering an alert and informs Alert Generator about it. Then the Alert Generator checks the trend of the detected data against the Data Storage to determine if it signifies a continuing condition. Upon confirmation the Alert Generator generates an alert and sends it to the Data Storage and next to Medical Staff. When they get it, they take necessary actions to solve the problem and when the condition is stabilized, they manually resolve the alert in the system.
The sequence diagram gives us an opportunity to show the chronological order of the actions and message exchanges. Each separate event contributes to the patient safety and medical intervention, reflecting the system's ability to support healthcare professionals. We chose a simple design in which every arrow represents the start/ finish of an occurrence and every block is an execution. Above very arrow there is a message indicating an action.