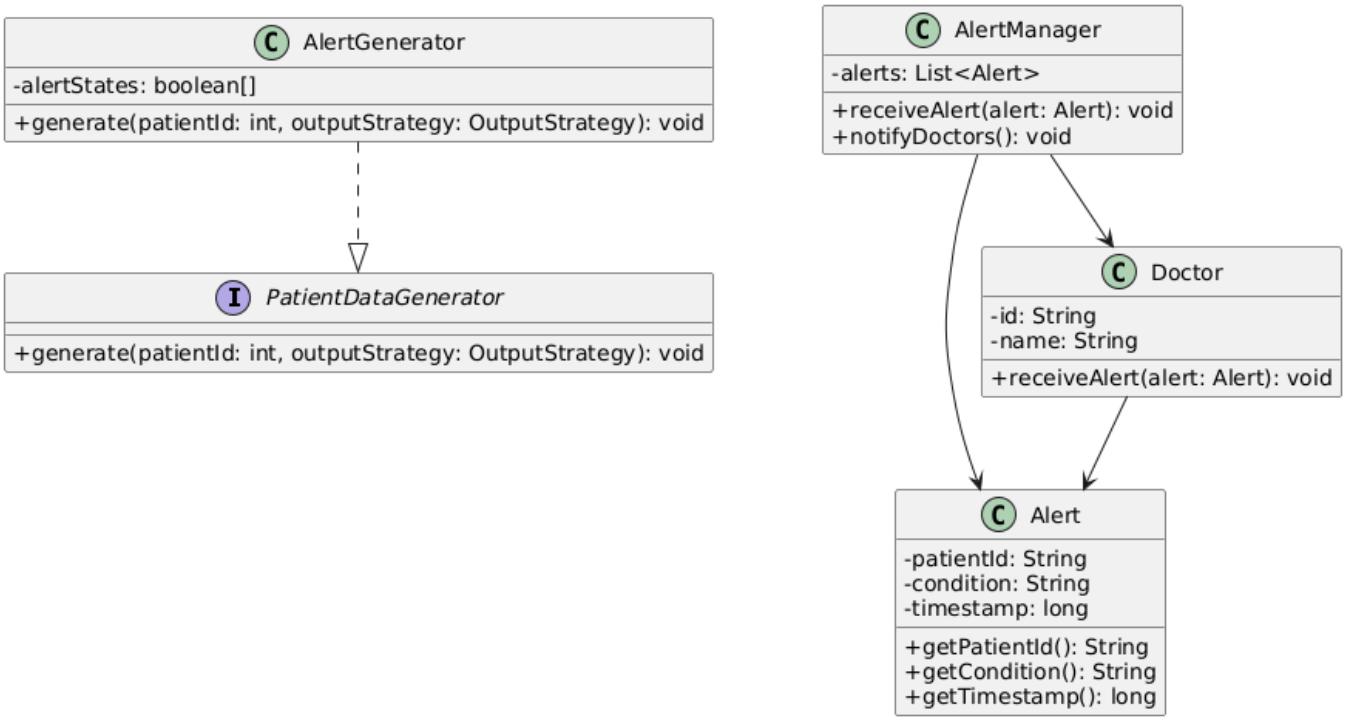
UML DIAGRAMS

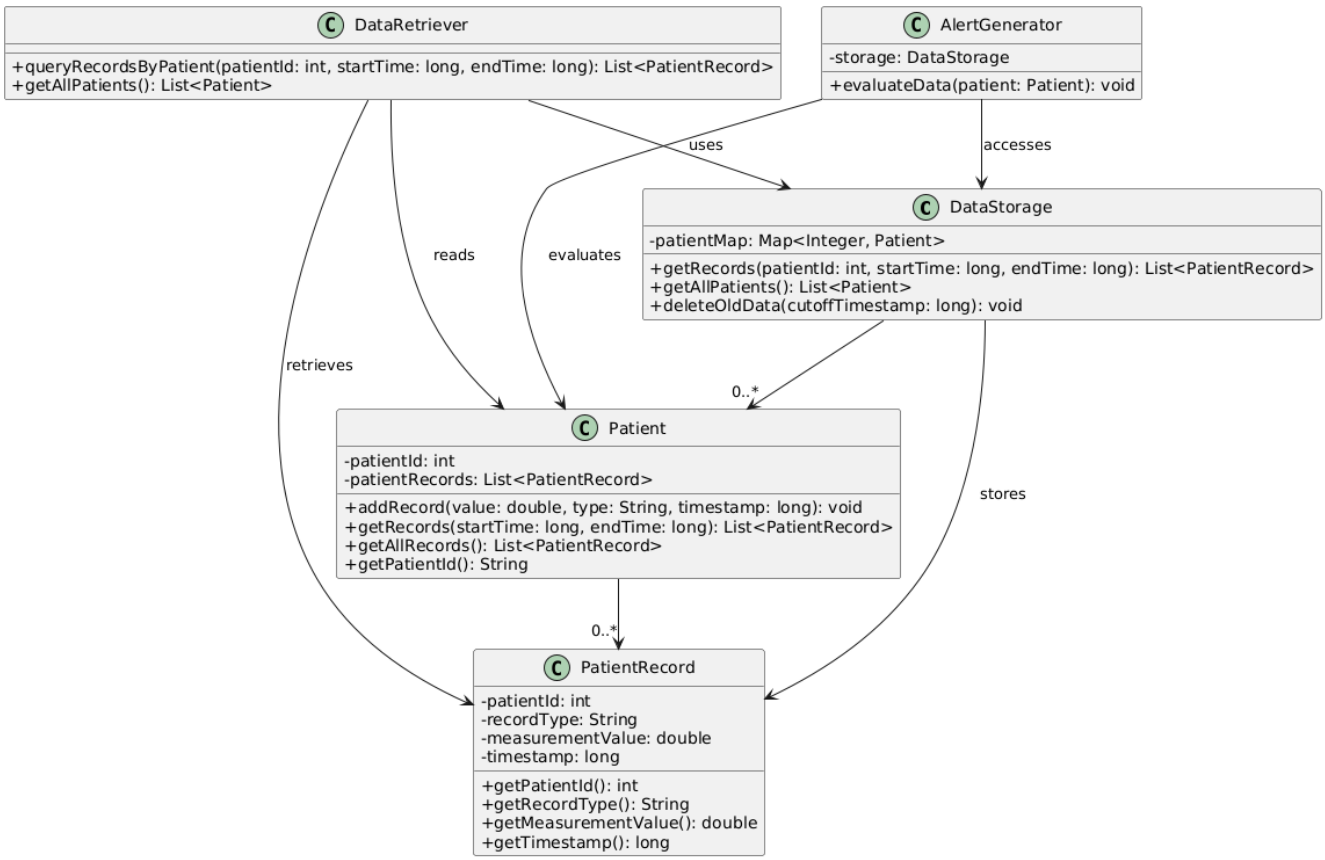
1. Alert Generation System



The ***Alert Generation System*** monitors patient data and triggers alerts based on specific health conditions. It consists of four key components: the Alert Generator, the Alert Manager, the Doctor, and the Alert. ***AlertGenerator***: This class generates alerts for patients based on their health data. If a patient already has an active alert, there's a 90% chance it will be resolved; if not, a small chance exists for a new alert to be triggered. ***Alert***: The Alert object contains the patient's ID, the condition causing the alert (e.g., "Critical Blood Pressure"), and the timestamp when the alert was triggered. ***AlertManager:*** Responsible for managing and dispatching alerts, the AlertManager forwards each alert to the Doctor. It ensures all alerts are handled efficiently. ***Doctor***: The Doctor receives alerts and takes necessary actions based on the patient's condition. They are responsible for responding to alerts and ensuring patient care. This system simulates real-time patient monitoring, where the AlertGenerator triggers alerts, the AlertManager oversees them, and the doctor responds, ensuring timely medical intervention for patients in need.

UML DIAGRAMS

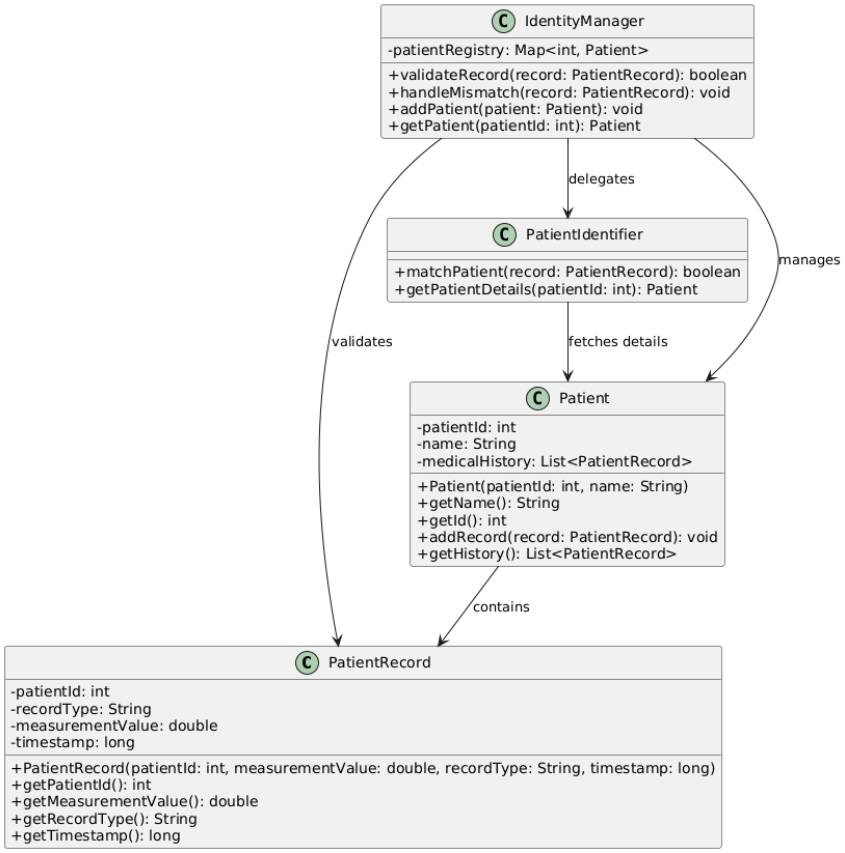
2. Data Storage System



The ***Data Storage System*** is responsible for managing all patient data, including important health information like heart rate and blood pressure, along with timestamps. This makes it easy for doctors to track patients in real-time and look at their past records to spot any changes or trends. The ***Alert Generator*** works with this data to check for any problems, like low oxygen levels or high blood pressure. If something is wrong, it sends out alerts so that medical staff can take action quickly. The system also keeps things organized by deleting old data after a certain period. Only authorized people can access or change the patient records, which ensures the data stays safe and private. Each ***Patient Record*** contains the patient's ID, the type of data, and the time it was taken, making it easy to track and manage all the information efficiently. This system helps doctors make informed decisions while keeping everything secure and up-to-date.

UML DIAGRAMS

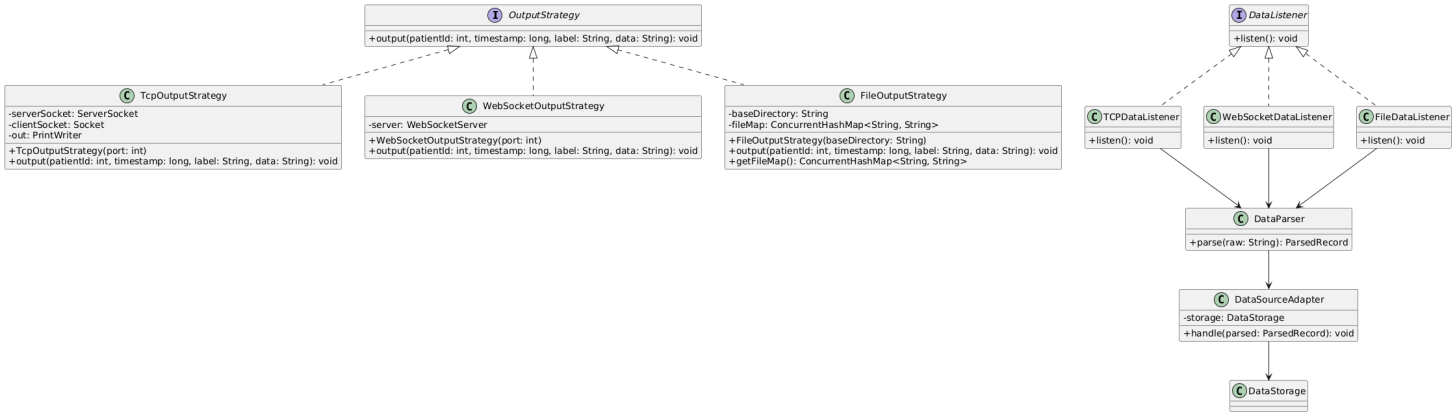
3. Patient Identification System



The ***Patient Identification System*** ensures that patient data is linked to the correct individual by matching IDs from the simulator with real hospital records. This process is essential for maintaining accurate patient histories and ensuring proper treatment. At the top of the hierarchy is the ***IdentityManager***, which oversees the entire process. It takes the ***PatientRegistry***, validates the patient's information, and retrieves the necessary details from the patient database. If a match is found between the incoming data and the hospital records, the relevant ***PatientRecord*** is retrieved, providing all of the patient's medical details. The ***PatientIdentifier*** plays a crucial role in matching incoming data with the correct patient. It cross-checks the ID from the signal generator against the hospital's database to ensure accuracy. If no match is found, the **IdentityManager** is responsible for handling the error, possibly by flagging the mismatch for review or requesting further information to resolve the discrepancy.

UML DIAGRAMS

4. Data Access Layer



The Data Access Layer connects the outside world to the internal healthcare system. It allows patient data to flow in from different sources—TCP, WebSocket, and File-based inputs. Each of these has its own listener class (***like TcpDataListener, WebSocketDataListener, and FileDataListener***), but they all follow a shared interface so the system stays flexible and organized. This layer listens for incoming data from a signal generator, processes it, and passes it along in a format the rest of the system can use. For example, a TCP server checks for client connections and sends patient data through sockets, while the WebSocket server can broadcast data live to multiple users. File-based input stores the data as logs for later analysis. Once the data is received, a DataParser converts it from raw formats (like CSV or JSON) into usable objects. The ***DataSourceAdapter*** then delivers this parsed data to the DataStorage system. ***The Data Access*** Layer keeps everything cleanly separated so that no matter how the data is sent, it can always be processed, stored, and accessed securely by doctors, patients, or other parts of the CHMS system.