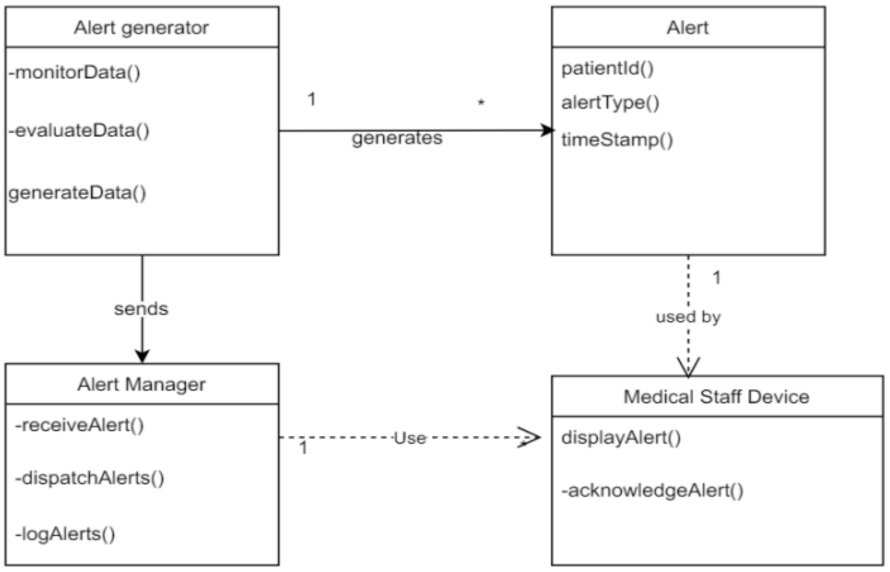
UML Diagrams

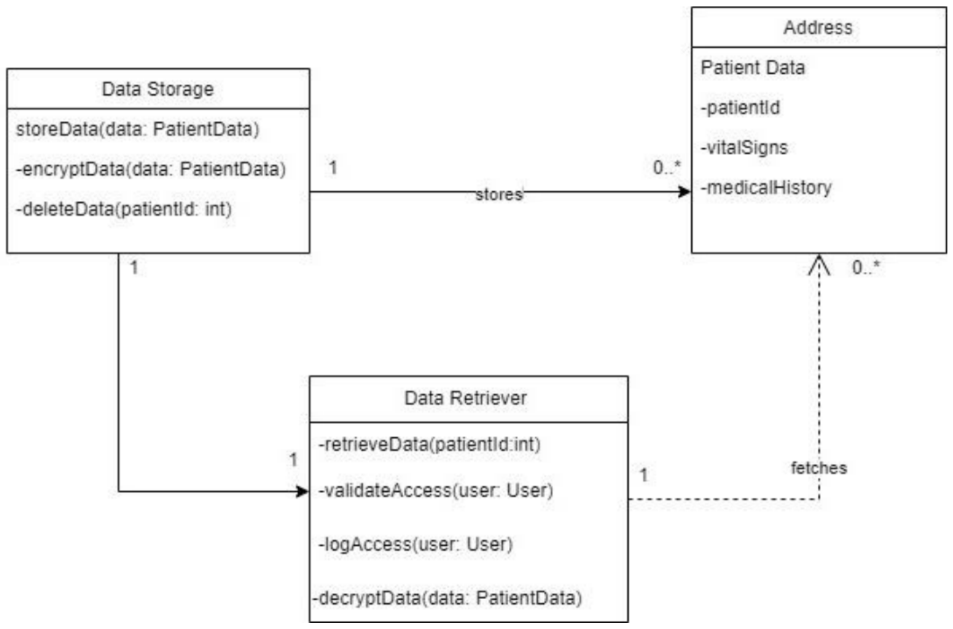
Alert Generation System:



Description:

In the Alert Generation System's UML diagram, the arrows represent crucial relationships and interactions between classes, enhancing the system's functionality and coherence. At its core, the AlertGenerator includes three methods. Their functionality is to receive real-time data streams from monitoring devices, analyzing them against personalized alert thresholds stored in the PatientProfile class. When a threshold is breached, an instance of the Alert class is instantiated, capturing essential details such as patient ID, alert type, and timestamp. This composition relationship between AlertGenerator and Alert signifies the AlertGenerator's responsibility for generating alerts based on analyzed data. Subsequently, the AlertManager takes charge of managing the lifecycle of alerts, organizing their dissemination to medical staff devices and workstations. Additionally, the association between Alert and AlertManager highlights how alerts are managed and distributed efficiently for timely interventions. Furthermore, the associations between Alert and PatientProfile, as well as PatientProfile and Threshold, illustrate how alerts are personalized based on individual patient profiles and their respective thresholds. Overall, these relationships contribute to a cohesive and efficient alert generation system within the cardiovascular ward, ensuring prompt responses to critical health events

Data Storage System:


Data Storage
storeData(data: PatientData)
-encryptData(data: PatientData)
-deleteData(patientId: int)
1
stores
0..*
Address
Patient Data
-patientId
-vitalSigns
-medicalHistory
1
0..*
Data Retriever
-retrieveData(patientId:int)
-validateAccess(user: User)
-logAccess(user: User)
-decryptData(data: PatientData)
1
1
fetches

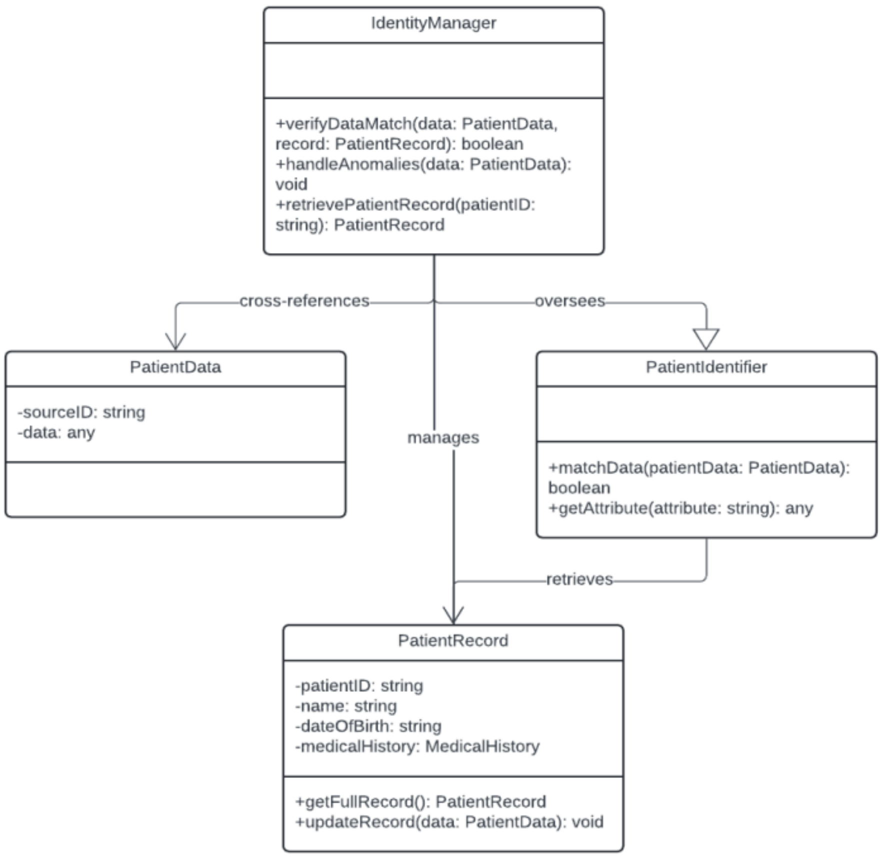
The UML diagram shows relationships and interactions between three main classes: DataStorage DataRetriever and Address within a patient data management system. The DataStorage class is central to the system. It is responsible for storing, encrypting and deleting patient data. It contains the key methods. For example storeData(data: PatientData) handles the storage of patient data. EncryptData(data: PatientData) ensures data is securely encrypted before storage. The method deleteData(patientId: int) allows for removal of patient data based on a unique patient ID. This class has a one or more relationship with the Address class.

The Address class encapsulates patient information with attributes such as patientId for uniquely identifying a patient. VitalSigns capture the most important health metrics. MedicalHistory maintains a comprehensive record of patient's health background. This class serves as data repository. The DataStorage class interacts with the Address class to manage patient data.

The DataRetriever class is very important in the secure retrieval and access control of patient data. It includes methods like retrieveData(patientId: int). This method fetches patient data by their unique ID. Another method validateAccess(user: User), checks if user has necessary permissions to access the data. Additionally the logAccess(user: User) method logs access attempts for control purposes and decryptData(data: PatientData) decrypts the data for use. The DataRetriever class interacts closely with the DataStorage class. It provides and decrypts the stored patient data, ensuring that only authorized access is allowed.

In summary the diagram highlights a robust system. The DataStorage class manages secure storage and encryption of patient data. The Address class has detailed patient information and the DataRetriever class ensures secure access and retrieval of this data. The relationships and methods defined in these classes ensure that patient data is handled securely.

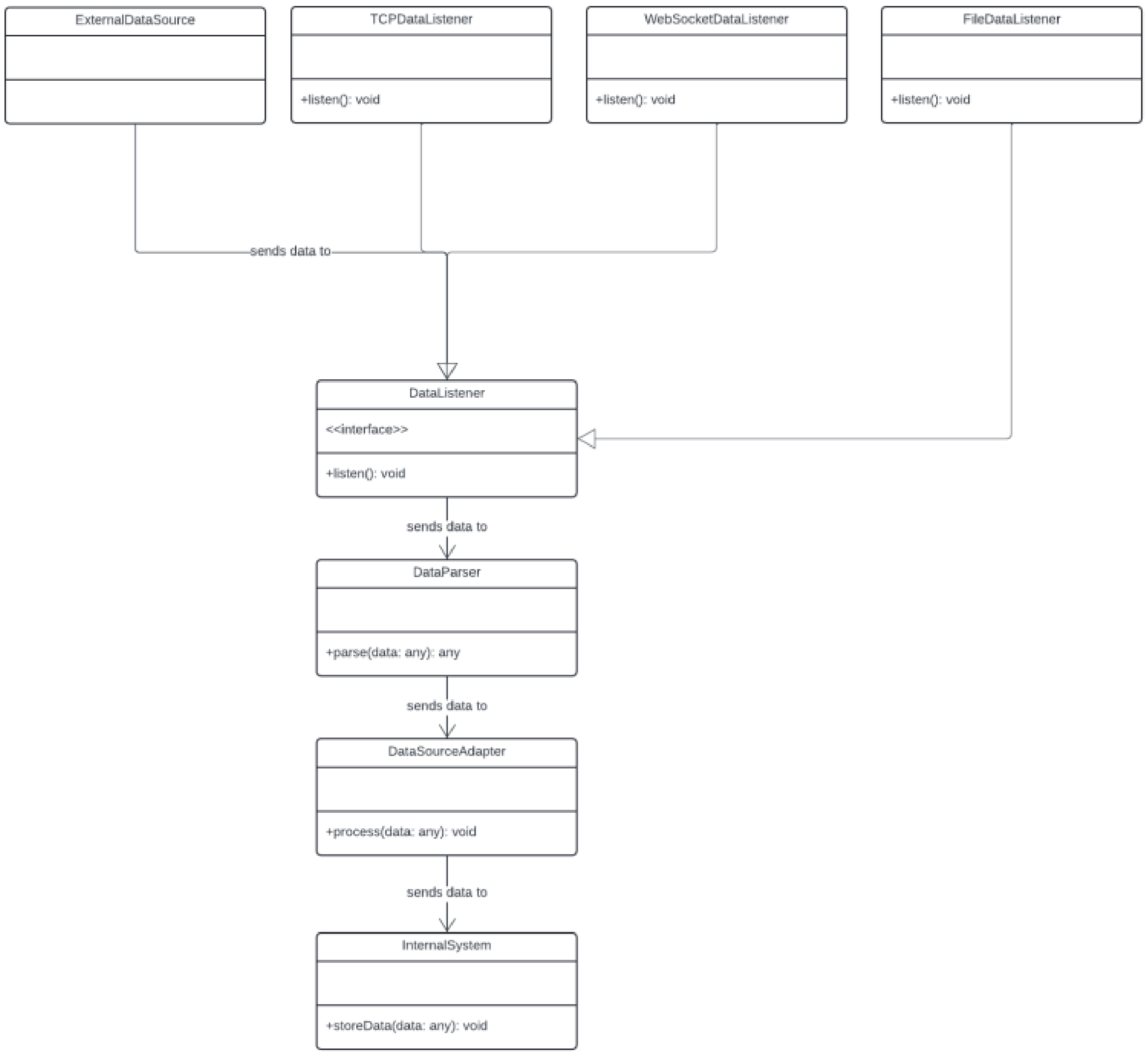
Patient Identification System:


IdentityManager
+verifyDataMatch(data: PatientData, record: PatientRecord): boolean
+handleAnomalies(data: PatientData): void
+retrievePatientRecord(patientID: string): PatientRecord
cross-references
oversees
manages
PatientData
-sourceID: string
-data: any
PatientIdentifier
+matchData(patientData: PatientData): boolean
+getAttribute(attribute: string): any
retrieves
PatientRecord
-patientID: string
-name: string
-dateOfBirth: string
-medicalHistory: MedicalHistory
+getFullRecord(): PatientRecord
+updateRecord(data: PatientData): void

The UML diagram illustrates a system for controlling patient data with four main classes: IdentityManager, PatientData, PatientRecord, and PatientIdentifier.

- IdentityManager is the main class, managing and cross-referencing patient data. It includes methods to verify data matches, handle anomalies, and retrieve patient records.
- PatientData stores incoming patient data with attributes like sourceID and data. It provides raw data for verification and abnormal detection.
- PatientIdentifier is responsible for matching patient data with existing records. It has methods like matchData for checking data matches and getAttribute for retrieving specific attributes.
- PatientRecord contains detailed patient information, including patientID, name, dateOfBirth, and medicalHistory. It includes methods to get the full record and update records with anomalies.

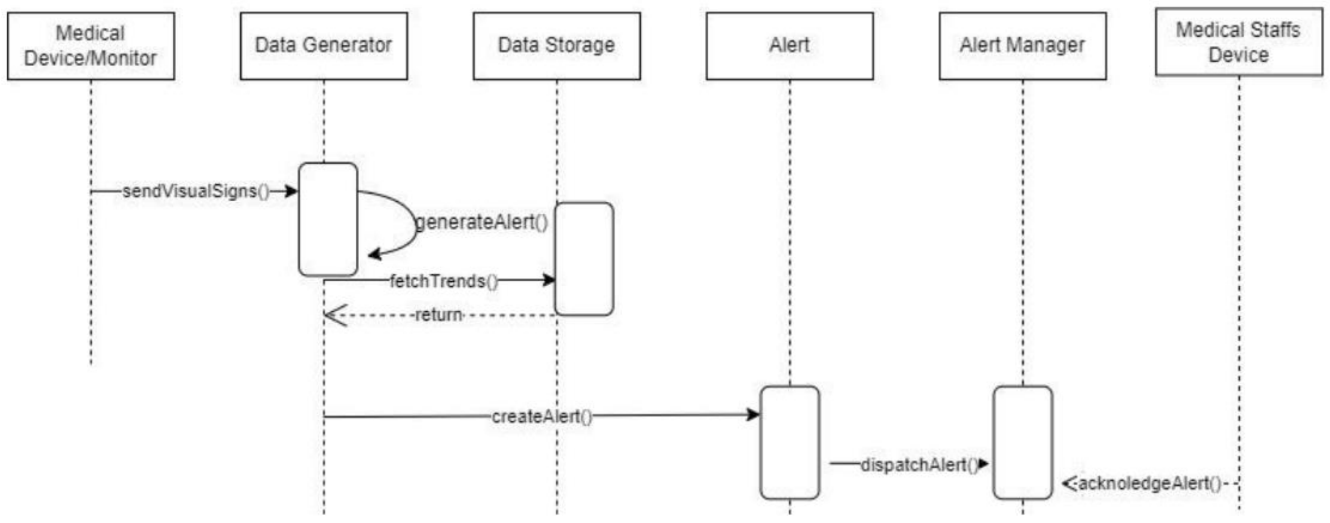
Data Access Layer:



Description:

The UML diagram shows a robust system is implemented to handle data from different sources, process it, and store it efficiently in an internal system. At the top layer, there are three data listeners (TCPDataListener, WebSocketDataListener, FileDataListener), each inherits from the DataListener interface. These listeners are receiving data from different sources such as TCP connections, WebSocket connections, and files. The DataListener interface enforces a standard listen () method, ensuring consistency across various listener types. Data received by these listeners is passed down to the DataParser, which processes the raw data into a structured format using the parse (data: any) method. The parsed data is then handled by the DataSourceAdapter, which further processes the data into a different format which is suitable for the inner system through the process (data: any) method. Finally, the InternalSystem stores the processed data using the storeData(data: any) method. This design promotes modularity and scalability, allowing the system to easily integrate new data sources and maintain consistent data handling procedures, ensuring efficient data processing and storage.

Sequence Diagram


Medical Device/Monitor
Data Generator
Data Storage
Alert
Alert Manager
Medical Staffs Device
sendVisualSigns()
generateAlert()
fetchTrends()
return
createAlert()
dispatchAlert()
acknoledgeAlert()