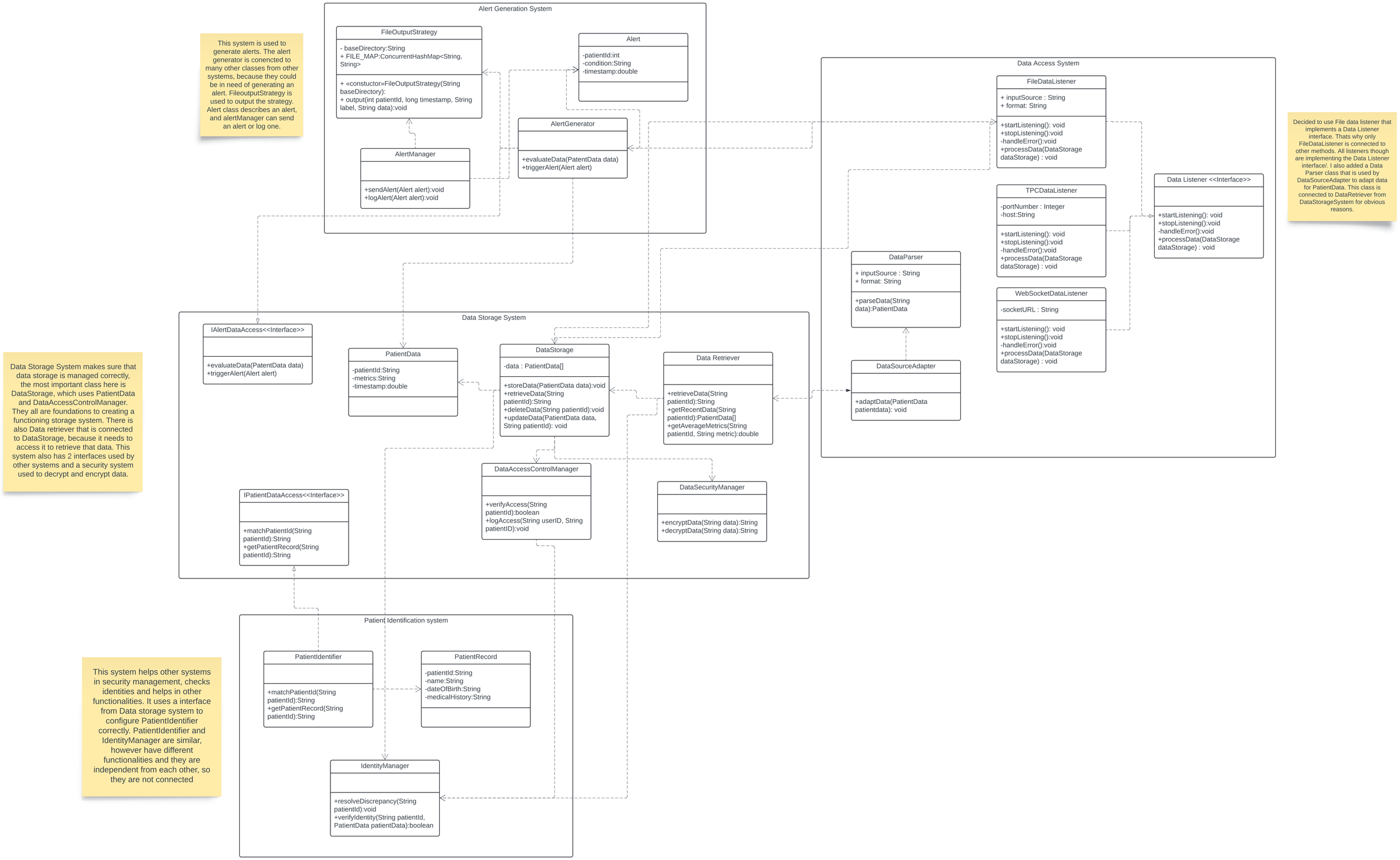
Alert Generation System
FileOutputStrategy
- baseDirectory:String
+ FILE_MAP:ConcurrentHashMap<String, String>
+ «constuctor»FileOutputStrategy(String baseDirectory):
+ output(int patientId, long timestamp, String label, String data):void
Alert
-patientId:int
-condition:String
-timestamp:double
AlertGenerator
+evaluateData(PatentData data)
+triggerAlert(Alert alert)
AlertManager
+sendAlert(Alert alert):void
+logAlert(Alert alert):void
This system is used to generate alerts. The alert generator is conencted to many other classes from other systems, because they could be in need of generating an alert. FileoutputStrategy is used to output the strategy. Alert class describes an alert, and alertManager can send an alert or log one.
Data Access System
FileDataListener
+ inputSource : String
+ format: String
+startListening(): void
+stopListening():void
-handleError():void
+processData(DataStorage dataStorage) : void
TPCDataListener
-portNumber : Integer
-host:String
+startListening(): void
+stopListening():void
-handleError():void
+processData(DataStorage dataStorage) : void
WebSocketDataListener
-socketURL : String
+startListening(): void
+stopListening():void
-handleError():void
+processData(DataStorage dataStorage) : void
Data Listener <<Interface>>
+startListening(): void
+stopListening():void
-handleError():void
+processData(DataStorage dataStorage) : void
Decided to use File data listener that implements a Data Listener interface. Thats why only FileDataListener is connected to other methods. All listeners though are implementing the Data Listener interface/. I also added a Data Parser class that is used by DataSourceAdapter to adapt data for PatientData. This class is connected to DataRetriever from DataStorageSystem for obvious reasons.
DataParser
+ inputSource : String
+ format: String
+parseData(String data):PatientData
DataSourceAdapter
+adaptData(PatientData patientdata): void
Data Storage System makes sure that data storage is managed correctly, the most important class here is DataStorage, which uses PatientData and DataAccessControlManager. They all are foundations to creating a functioning storage system. There is also Data retriever that is connected to DataStorage, because it needs to access it to retrieve that data. This system also has 2 interfaces used by other systems and a security system used to decrypt and encrypt data.
Data Storage System
IAlertDataAccess<<Interface>>
+evaluateData(PatentData data)
+triggerAlert(Alert alert)
PatientData
-patientId:String
-metrics:String
-timestamp:double
DataStorage
-data : PatientData[]
+storeData(PatientData data):void
+retrieveData(String patientId):String
+deleteData(String patientId):void
+updateData(PatientData data, String patientId): void
Data Retriever
+retrieveData(String patientId):String
+getRecentData(String patientId):PatientData[]
+getAverageMetrics(String patientId, String metric):double
DataAccessControlManager
+verifyAccess(String patientId):boolean
+logAccess(String userID, String patientID):void
DataSecurityManager
+encryptData(String data):String
+decryptData(String data):String
IPatientDataAccess<<Interface>>
+matchPatientId(String patientId):String
+getPatientRecord(String patientId):String
This system helps other systems in security management, checks identities and helps in other functionalities. It uses a interface from Data storage system to configure PatientIdentifier correctly. PatientIdentifier and IdentityManager are similar, however have different functionalities and they are independent from each other, so they are not connected
Patient Identification system
PatientIdentifier
+matchPatientId(String patientId):String
+getPatientRecord(String patientId):String
PatientRecord
-patientId:String
-name:String
-dateOfBirth:String
-medicalHistory:String
IdentityManager
+resolveDiscrepancy(String patientId):void
+verifyIdentity(String patientId, PatientData patientData):boolean