ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

DA HL7 A SERVIZI WEB REST PER LA FRUIZIONE DEI DATI DI APPARATI MEDICI ATTIVI - UN CASO DI STUDIO

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
LUCA SAMBUCHI

Corelatore
Dott. Ing. ANGELO CROATTI

Anno Accademico 2018 – 2019

dedicato alla mia famiglia, alla mia ragazza e ai miei amici

Indice

Introduzione

In ambito sanitario, i moderni apparati medici attivi – categoria alla quale appartengono apparecchiature per l'esecuzione di esami diagnostici e/o analisi di laboratorio – sono oggigiorno sempre più in grado di comunicare via rete, per ricevere comandi e trasmettere i risultati degli esami effettuati. Ovvero, si sta assistendo ad una permeazione sempre maggiore dell'Internet of Things (IoT) in ambito ospedaliero e non solo.

In questo contesto, una delle problematiche che emergono è relativa all'accesso pervasivo ai dati e alla loro fruizione da parte di applicazioni terze a supporto dell'operatività di medici ed infermieri. In effetti, sebbene quasi tutti questi apparati siano in grado ormai di essere accessibili via rete, generalmente comunicano esclusivamente con gateway/server ad hoc gestiti dalla azienda produttrice dell'apparato e, solo a volte, tali gateway consentono un accesso interoperabile alle informazioni che memorizzano.

Qualora questo sia consentito, in genere il protocollo utilizzato per la comunicazione è HL7. Si tratta di un protocollo pensato specificatamente per la comunicazione tra apparati biomedicali, che tuttavia oggigiorno non risponde alle esigenze dei moderni sistemi software.

Obiettivo di questa tesi è studiare e progettare un sistema software in grado di fungere da mediatore tra gli apparati medici attivi e i moderni sistemi software, in modo che questi ultimi possano accedere agevolmente ai dati d'interesse indipendentemente dalla loro natura e provenienza. In particolare, sebbene abbia potenzialmente una valenza più ampia, il progetto di tesi si inserisce nell'ambito del progetto TraumaTracker, un sistema a supporto della gestione dei traumi gravi attivo presso il Trauma Center dell'Ospedale "M.Bufalini" di Cesena.

Il caso di studio considera nello specifico il dispositivo EGA (emogasanalizzatore) che effettua particolari esami (emogasanalisi) del sangue di un paziente. L'obiettivo è sviluppare un servizio web con interfaccia REST che consenta a TraumaTracker di accedere ai dati prodotti dall'EGA per i propri pazienti di interesse. Sarà compito del sistema software interfacciarsi con il gateway del dispositivo, interpretarne i messaggi HL7 e memorizzarne il contenuto in modo opportuno.

La tesi è articolata in cinque capitoli: nel primo capitolo sono introdotti ed approfonditi i concetti di HL7 come organizzazione e standard sanitario, esaminandone le differenti versioni, la struttura dei messaggi ed il loro scambio. Oltretutto, è evidenziata in modo particolare l'importanza dell'interoperabilità. Nel secondo capitolo è introdotto il concetto di emogasanalisi ed il funzionamento del macchinario che la esegue, base del progetto di tesi. A questo punto viene affrontato approfonditamente lo scopo che l'intero progetto si pone, nonché i vantaggi che apporterà all'ospedale "M.Bufalini" di Cesena. Ci si sofferma inoltre non solo sulle specifiche del macchinario EGA, bensì anche sui requisiti del gateway coinvolto. Nel terzo capitolo è analizzato l'intero progetto, la sua architettura e la gestione del web-service. Inoltre sono descritte le caratteristiche e la struttura delle API REST. Il penultimo capitolo tratta l'implementazione prototipale, spiegando sia lo sviluppo del web-service che del client di prova, descrivendo da ultimo i messaggi scambiati tra i due sistemi. Infine all'interno del capitolo cinque sono affrontati i test relativi alle funzionalità del sistema e alle casistiche di errore approfondendo, come ultimo punto trattato, i possibili sviluppi futuri dell'intero progetto.

L'applicazione web sviluppata potrebbe essere perfezionata nel momento in cui verrà sperimentata in ospedale, così da essere certi che sia pronta ad operare in un contesto reale.

Capitolo 1

Il protocollo HL7

Nel primo capitolo sono introdotti ed approfonditi i concetti di HL7 come organizzazione e standard sanitario, descrivendone le varie caratteristiche, per poi sottolineare il significato di *Level Seven*.

Ci si sofferma poi sull'interoperabilità, caratteristica che riguarda la possibilità di interazione tra sistemi che utilizzano tecnologie differenti, di cui è evidenziato il notevole valore.

In seguito sono descritte le differenti versioni di HL7, spiegando quale sia stata scelta per la realizzazione del progetto e quali vantaggi presenta rispetto alle altre.

L'ultimo argomento trattato in questa sezione riguarda invece i messaggi HL7, di cui è analizzata la struttura, rafforzando quanto descritto tramite l'esempio di un messaggio concreto. Da qui si approfondisce infine il processo di scambio di messaggi e le tipologie di comunicazione possibili tra sistemi.

1.1 Health Level Seven International (HL7)

La HL7 è un'organizzazione no profit la quale fornisce un quadro completo e standard correlati per lo scambio, l'integrazione, la condivisione e il recupero di informazioni sanitarie elettroniche che supportano pratica clinica e valutazione dei servizi sanitari. Essa è fondata nel 1987 negli USA ed è composta da membri che risiedono in oltre 50 paesi. HL7 è un'associazione *Standards Developing Organizations* (SDO) accreditata presso l'ANSI, nel 1994, che opera nel settore della sanità. Le SDO hanno come obiettivo la produzione di standard per particolari domini, infatti HL7 comprende i dati clinici e amministrativi della sanità. Il 20 Marzo 2003 è stata costituita la Sezione Italiana di HL7 la quale intende adattare lo standard alle necessità nazionali, all'interno delle regole generali stabilite da HL7 e in collaborazione con l'UNI, Ente Italiano di Normazione.

Si inizia a parlare per la prima volta di standard sanitari negli anni '60 per lo scambio di messaggi, salvataggio di dati medici e sicurezza dei sistemi sanitari. Attualmente gli standard più utilizzati sono:

- *Health Level 7* - HL7
- *Digital Imaging and Communication in Medicine* - DICOM
- *Open*EHR

Alla base del progetto della tesi vi è appunto lo standard HL7, per questo motivo è l'unico tra questi che verrà trattato in maniera approfondita.

1.2 Il significato di Level Seven

Level Seven si riferisce al livello dell'applicazione, ovvero il settimo livello del modello di comunicazione a sette strati dell'Organizzazione Internazionale per la Standardizzazione (ISO) per Open Systems Interconnection (OSI). Esso è a diretto contatto con le applicazioni, come ad esempio programmi di posta elettronica o browser, ciò significa che non fa riferimento agli aspetti implementativi. Nel *Level Seven* non solo avvengono l'input e l'output dei dati, ma viene inoltre stabilita la connessione con i livelli inferiori del modello.

1.3 Il valore dell'interoperabilità

Oltre alle caratteristiche precedentemente descritte, l'HL7 ha come obbiettivo quello di garantire l'interoperabilità tra i vari sistemi informativi utilizzati dai diversi reparti. Infatti, inizialmente, uno dei problemi principali a cui porre rimedio era trovare una soluzione al quesito: come avviene lo scambio di informazioni tra reparti ospedalieri? I sistemi separati in un ospedale devono essere in grado di interagire tra loro nel modo più efficiente possibile. Una delle risposte, con conseguente risoluzione del problema, venne con la nascita di HL7, il quale è appunto lo standard per la comunicazione di messaggi più diffuso al mondo nel settore dell'ICT sanità.

Per interoperabilità si intende la capacità di due o più sistemi differenti di scambiare informazioni e di essere poi in grado di utilizzarle e si di divide in:

- Funzionale: la capacità di scambiare informazioni in modo affidabile senza errori
- Semantica: la capacità di interpretare e, quindi, fare un uso efficace delle informazioni così scambiate

Si tratta di un aspetto fondamentale per far si che l'impiego delle tecnologie informatiche, in ambito medico, sia possibile su larga scala. Oltre a ciò, gli ospedali devono essere connessi tra loro per supportare il passaggio di informazioni. L'interoperabilità permette di creare sistemi con maggiore robustezza, minori costi di manutenibilità e migliore scalabilità. Inoltre, occorre considerare che il modello elettronico permette una riduzione dei rischi ed una miglior comunicazione, superando l'inefficienza del modello cartaceo. Infine, essa comporta un miglioramento del fattore economico, il quale è invece in peggioramento a causa dell'aumento dell'età media e dell'incremento della cultura sanitaria. Tutto ciò, infatti, porta la domanda di servizi ad aumentare costantemente, il che si riflette in un aumento della spesa pubblica.

Per tutte queste ragioni è quindi importante lavorare sugli standard per lo scambio e la condivisione dei dati per permettere la comunicazione fra applicazioni, sistemi e organizzazioni diverse.

1.4 Le versioni HL7

Attualmente la versione HL7 più utilizzata è la 2.x, la quale si occupa specificatamente della standardizzazione dei messaggi. Il primo standard versione 2 è stato creato nel 1989 e da allora è stato costantemente rivisto e aggiornato. Le prime versioni 2.x erano piuttosto imprecise e prive di specifiche formali ma successivamente, con la crescita dell'adozione dello standard, vennero prodotte versioni sempre più specifiche ed in grado di coprire un numero sempre maggiore di aree, mantenendo però la compatibilità con le versioni precedenti. La versione più recente è la terza, nata per creare un modello dei dati più stretto e preciso. I suoi obbiettivi principali sono:

- Migliorare la chiarezza e la precisione delle specifiche
- Migliorare l'adattabilità dello standard ai cambiamenti
- Sfruttare le tecnologie emergenti come XML
- Progettare pensando all'interoperabilità

HL7 3 introduce il *Clinical Document Architecture* (CDA), il quale rappresenta un modello di scambio di documenti in ambito clinico, con vari livelli di complessità. Il CDA è un documento scritto in XML che può contenere testi, immagini, suoni ed altri elementi multimediali. Un esempio di messaggio HL7 versione 3 è mostrato in Figura 1.2. Inoltre questa versione facilita il parsing ed il controllo sintattico sui dati scambiati grazie all'uso del XML.

Invece HL7 2.x è costituito da dei messaggi di testo formattati grazie all'uso di determinati caratteri. Nonostante tutto, la terza versione è meno utilizzata rispetto alle versioni 2.x per i seguenti motivi:

- Più complessa
- Non è retrocompatibile con la versione 2.x
- Meno opzioni rispetto alla versione 2.x
- Maggior costo poiché la versione 3 non è compatibile con le versioni precedenti e, essendo che la maggior parte dei sistemi utilizza ancora la versione 2.x, occorre di volta in volta convertire un'interfaccia versione 2.x esistente in versione 3, cosa che oltre a richiedere tempo, può essere difficile

Per rimpiazzare la versione 3 e, allo stesso tempo, porre rimedio alle lacune di HL7 2.x, ad esempio eccessiva flessibilità e modello dei dati non esplicito, è ultimamente in via di sviluppo lo standard FHIR (Fast Healthcare Interoperability Resources). Un'ulteriore caratteristica di questo nuovo standard consiste nell'utilizzo di tecnologie e architetture leggere, in modo tale da agevolare le applicazioni mobile, così da permettere ai fornitori di usufruirne senza eccessiva difficoltà. Nonostante ciò, abbiamo dovuto optare per la versione 2.x poiché è quella utilizzata dal macchinario di interesse della tesi, presente all'interno dell'ospedale "M.Bufalini" di Cesena.

1.5 Messaggi HL7

Un messaggio HL7 ha lo scopo di passare informazioni di un certo evento da un sistema all'altro. Esistono varie tipologie di messaggio identificate da un codice di tre lettere; l'evento che scatena l'inizio di una comunicazione è denominato evento "trigger".

Il messaggio HL7 è un contenuto testuale, composto da un *header* e da una sequenza ordinata di segmenti. Ogni segmento ha un codice che ne definisce il contenuto (ad esempio PID sta per Patient Identification) ed è composto da campi, i quali hanno posizioni ben definite; sempre considerando il caso del PID, troveremo informazioni come il cognome, il nome, il codice fiscale e la data di nascita del paziente. Queste sono delimitate, all'interno della stessa riga, dal carattere "|" e, al loro interno, lo stesso campo può anche essere suddiviso in più sotto campi, a loro volta delimitati dal carattere "^". Per esempio, continuando a considerare il segmento PID, l'indirizzo è contenuto tra due "|" e si trovano i sotto campi, via, numero civico, cap, città, provincia.

```
MSH|^~\&|Rapidcomm|Hospital|HIS|Hospital|201506051656||ORU^R32|
0C0634M000Z000010404|P|2.3|||AL|AL|
PID|1||MRN^^^||LNAME^FNAME^MNAME^^^||DOB|SEX|||ADDR^^CITY^STATE^PCODE^
^^||PHONE||||ACCOUNT^^^|SSN|||||||
ORC|NW|ACCESSION|^|^|IP||001^^^201506061014^201506061014^S^^^^||201506
061014|||OPHYID^OPHYNAME ^^^^^^|||||||
OBR|1|PORDER|FORDER|Urinalysis||||||S^^^^^^^|O|||||
OPHYID^OPHYNAME^^^^^^||||||||||001^^^201506061014^201506061014^
S^^^^| ||||||||201506061014|
OBX|1|ST|ALB||10|mgL|||||F|||20150601112058||ROID^ROLNAME^ROFNAME||^^4
001^CLINITEK Status|20150601112058|
OBX|2|ST|BIL||Small|||||F|
OBX|3|ST|GLU||100|mg/dL|||||F|
```

Figura 1.1: Esempio di messaggio HL7 versione 2.x

```
<author>
                <authorPerson>
                        <time value="201105192000"/>
                        <id extension="100" root="2.16.840.1.113883.2.1.3.2.4.18.24"/>
                        <code code="OOH02"/>
                        <name>
                                <given>Mary</given>
                                <family>Jones</family>
                        </name>
                        <orgId extension="5L399" root="2.16.840.1.113883.2.1.3.2.4.19.1"/>
                        <orgName>Medway NHS Foundation Trust</orgName>
                </authorPerson>
</author>
```

Figura 1.2: Esempio di messaggio HL7 versione 3

Tutte queste caratteristiche del messaggio HL7 è possibile notarle nella Figura 1.1. Di seguito si illustrano alcuni dei tipi di messaggi disponibili.

- ADT - *Admission, Discharge and Transfer:* trasmissione di dati che riguardano ricoveri (Admission), dimissioni (Discharge) e trasferimenti(Transfer)
- ORM - *Order Message:* trasmissione di informazioni su un ordine
- ORU - *Observation Result:* messaggi riguardanti risultati di osservazioni, di solito è in risposta a un ordine
- ACK - *Acknowledge:* utilizzato per segnalare la corretta ricezione del messaggio

Ogni tipo di messaggio potrebbe essere associato a diversi trigger event: le operazioni di trasferire e registrare un paziente sono, ad esempio, due eventi trigger differenti nel messaggio ADT. Di quest'ultimo è mostrato un esempio in Figura 1.3.

Un messaggio HL7 ha una lista di tre tipi di segmenti: obbligatori, ovvero sempre presenti all'interno di uno stesso messaggio, opzionali, che quindi si possono omettere, e ripetibili, cioè possono essere presenti più volte in un messaggio. Inoltre ogni segmento è composto da campi obbligatori e facoltativi. I primi devono essere presenti nel messaggio per il corretto funzionamento, mentre i campi facoltativi possono essere utilizzati se lo si desidera. Ogni segmento HL7 termina con un ritorno a capo che, essendo un codice ASCII 13 (carattere non stampabile), non verrà mostrato quando si visualizza il messaggio. Di seguito si illustrano alcuni segmenti.

- MSH - *Message Header:* ogni messaggio inizia sempre con questo segmento e definisce l'intento, l'origine, la destinazione di un messaggio e contiene il *Message Control ID* (MCI) che viene utilizzato per confermare la ricezione di un messaggio HL7
- PID - *Patient Identification:* contiene le informazioni specifiche del paziente
- PV1 - *Patient Visit Segment:* utilizzato per comunicare informazioni specifiche per la visita del paziente. I messaggi possono contenere più segmenti PV1 per comunicare informazioni su più visite dei pazienti.
- OBX :utilizzato per trasportare le informazioni chiave di un osservazione clinica. Questo segmento può essere utilizzato più volte nello stesso messaggio.

Essendo che HL7 è il formato utilizzato per lo scambio di messaggi all'interno del progetto di tesi, è opportuno mostrarne la struttura tramite un esempio dell'ADT. Questo messaggio ha diversi segmenti obbligatori che sono: MSH, EVN, PID, PV1 opportunamente inseriti nella Figura 1.3; in questo caso, non sono presenti segmenti opzionali. L'evento trigger di questo messaggio ADT è A03, come mostrato nel campo MSH 9.2, cioè segmento MSH, campo 9, sotto campo 2. L'evento A03 di un messaggio ADT segnala la fine della permanenza di un paziente in una struttura sanitaria.

```
MSH|^~\&|HIS|Hospital|Rapidcomm|Hospital|200506051656||ADT^A03|
17-1-35:CC1JGE|P|2.3|||AL|NE|
EVN|A03|||||200508300930|
PID|1||MRN^^^||LNAME^FNAME^MNAME^^^||DOB|M|||ADDR^^CITY^STATE^PCODE^^^
||PHONE|||||ACCOUNT^^^|SSN|||||||
PV1|1|I|31^3107^1^^|E|||APHYID^APHYNAME^^^^^^|RPHYID^RPHYNAME^^^^^^|||
|||||||
```

Figura 1.3: Esempio di messaggio ADT

1.6 Lo scambio dei messaggi

I messaggi HL7 vengono in genere inviati tramite una connessione tra due sistemi. Ogni sistema ha un ruolo nella comunicazione: uno funge da client e l'altro da server. Tipicamente quello che invia i dati è il client, ma non è sempre vero. Sono due le modalità in cui vengono scambiati i messaggi:

- *Unsolicited:* definisce un'interfaccia che invierà i risultati HL7 senza che sia richiesto, necessitando solo che venga stabilita una connessione TCP con il sistema di ricezione. Il sistema ricevente risponde con un messaggio di tipo ACK.
- *Solicited:* definisce un'interfaccia che invierà un QRL HL7, messaggio di query, che richiede dati a un sistema sorgente, prima di tutto stabilendo una connessione TCP con esso.

Inoltre è possibile identificare due tipologie di comunicazione che sono:

- *Unidirezionale:* quando la comunicazione avviene in una sola direzione. Si considera unidirezionale anche se un ACK viene restituito da una destinazione, quando viene riconosciuto un messaggio HL7
- *Bidirezionale:* quando i ruoli dell'emittente e del ricevente sono continuamente scambianti, instaurando un flusso di comunicazione in più direzioni.

Il client apre una socket TCP/IP con il server e, questa connessione, viene utilizzata esclusivamente per la comunicazione tra questi due sistemi. Una gran parte di messaggistica HL7 viene trasportata dal *Minimal Lower Layer Protocol* (MLLP), noto anche come *Lower Layer Protocol* (LLP). Per la trasmissione, caratteri di intestazione e di coda vengono aggiunti al messaggio per identificarne l'inizio e la fine, perché TCP/IP è un flusso continuo di byte. Una volta stabilita la connessione, il sistema di invio può consegnare un messaggio HL7.

Capitolo 2

Il caso di studio

Il capitolo secondo si apre con un approfondimento in merito all'importanza delle tecnologie informatiche in ambito sanitario. Ne vengono sottolineati i valori positivi, ovvero una riduzione dei costi per effettuare le operazioni all'interno degli ospedali ed una più efficacie gestione dei dati relativi ai pazienti. Segue l'introduzione del concetto di *Internet of things* (IOT), in relazione al quale è presentato il macchinario alla base del progetto di tesi: l'emogasanalizzatore.

Viene poi affrontato approfonditamente lo scopo di questa tesi, mettendo in risalto i vantaggi che apporterà all'ospedale "M.Bufalini" di Cesena, nonché i requisiti secondo i quali è stato studiato e sviluppato il progetto stesso.

Infine ci si sofferma sulle particolarità del gateway, senza tralasciare la descrizione del modo in cui il nostro progetto si inserisce all'interno del sistema ospedaliero.

2.1 L'emogasanalizzatore (EGA)

Ai giorni nostri, per merito della continua crescita delle tecnologie, è impensabile non utilizzarle per migliorare la qualità della cura dei pazienti all'interno delle strutture sanitarie. Un esempio lampante è quello delle farmacie, nelle quali l'informatica permette di gestire in maniera precisa le scorte dei medicinali oppure di segnalare consumi crescenti e così via. Uno dei fattori chiave di utilizzo è quello economico, che mira a ridurre il più possibile i costi per effettuare le operazioni in ambito sanitario.

Le tecnologie informatiche permettono il miglioramento della gestione amministrativa dell'ospedale e sono essenziali anche in campo clinico, per migliorare le diagnosi e rendere più efficaci le cure grazie alla creazione di sistemi informatici specifici. Inoltre permettono l'intera gestione dei pazienti, archiviandone i dati oppure registrando le terapie prescritte e gli esiti ottenuti. Questa rivoluzione tecnologica tende ad aumentare la produttività effettuando

operazioni in maniera molto più rapida, ad esempio permette di indirizzare il paziente immediatamente verso la struttura libera più idonea, dopo che il medico ha prescritto la terapia. Vero è che ciò comporta una riduzione delle ore lavorative del personale ospedaliero ma, anzitutto, si ottiene una riduzione dei costi delle ore, e in aggiunta vengono a crearsi nuovi posti di lavoro nei settori in espansione che ne beneficiano.

Tutte queste innovazioni in atto nelle strutture sanitarie sono influenzate dalla quarta rivoluzione industriale, che sta sconvolgendo radicalmente quasi tutti i settori aziendali. Questa rivoluzione sfrutta l'ambito *Internet of things* (IOT) con il grande vantaggio di raccogliere dati da dispositivi sempre connessi, con la possibilità di valutare e gestire meglio le informazioni apprese.

Un esempio di questi dispositivi è il macchinario EGA, che è alla base del progetto di tesi. Gli emogasanalizzatori da cui riceviamo i dati all'interno dell'ospedale "M.Bufalini" di Cesena, corrispondenti al modello RAPIDPoint 500 Systems, sono tre: uno situato nel pronto soccorso e due nel reparto di terapia intensiva. Inoltre è presente un ulteriore macchinario utilizzato solamente nei casi di urgenza, nel momento in cui quello all'interno del pronto soccorso abbia delle problematiche. I medici ricorrono all'emogasanalisi per capire qual è l'efficienza, nell'individuo esaminato, dello scambio di gas che coinvolge il sangue e l'aria inspirata, all'interno degli alveoli polmonari. Tra le principali condizioni mediche per le quali è utile il ricorso all'emogasanalisi, rientrano: le malattie polmonari e respiratorie, l'insufficienza renale, l'insufficienza cardiaca e il diabete. In Figura 2.1 è mostrato un esempio di macchinario EGA.

2.2 Lo scopo del progetto

Nell'ambito della gestione dei traumi gravi, produrre un'adeguata documentazione del trauma è uno dei compiti del trauma team – ovvero la squadra di medici ed infermieri che si occupa della cura del paziente traumatizzato in fase di emergenza. Il progetto TraumaTracker – sviluppato in collaborazione con il Trauma Center dell'ospedale "M.Bufalini" di Cesena – consiste in un sistema informatico a supporto del trauma team per la produzione automatica di un'efficace documentazione del processo trauma. In particolare, consente al trauma team di registrare tutte le procedure effettuate, i farmaci somministrati e di tracciare in modo automatico e in real-time i parametri vitali del paziente e di correlarli alle azioni svolte dal team.

Tra gli esami effettuati dal team in fase di gestione del trauma, di particolare importanza per via delle informazioni che offre è l'emogasanalisi (EGA). Presso il pronto soccorso dell'ospedale "M.Bufalini" di Cesena sono presenti degli emogasanalizzatori che, data in ingresso una siringa con il sangue del

Figura 2.1: Emogasanalizzatore modello RAPIDPoint 500 Systems

paziente, effettuano l'analisi e producono un referto con un insieme di valori relativi a parametri di particolare interesse nella gestione del trauma (pH, saturazione dell'ossigeno, emoglobina, etc.).

Tale referto è fornito dalle apparecchiature ai medici in formato cartaceo e trasmesso via rete dati ad un gateway/server dedicato installato sulla rete informatica dell'ospedale. Ciò comporta però il sollevarsi di due problematiche: in primo luogo la gestione dei dati tramite formato cartaceo è ormai inefficiente, per quanto l'ospedale attualmente stia procedendo in questo modo. La seconda problematica consiste invece nel fatto che, una volta che il macchinario EGA ha inviato i risultati al gateway/server, l'ospedale non riesce più ad accedervi.

Lo scopo di questa tesi è porre rimedio a queste due condizioni, analizzando, progettando e prototipando un web-service studiato per offrire all'ospedale la possibilità di recuperare ed interagire con i dati inviati al gateway, attraverso un'opportuna interfaccia, in modo che sistemi come TraumaTracker possano recuperali in automatico. Il web-service permetterà di passare quindi ad una gestione digitale dei dati relativi ai pazienti, abbandonando il formato cartaceo, e rendendo ogni interazione e ricerca più facile ed immediata.

Nella realtà dell'ospedale "M.Bufalini", il gateway trasmette i dati d'interesse codificati secondo il protocollo HL7. Il sistema informatico oggetto di questa tesi dovrà quindi ricevere e decodificare i referti EGA trasmessi dal gateway, memorizzarli secondo un opportuno modello del dominio in una base di dati locale e, infine, offrire un'opportuna web API mediante la quale i servizi esterni possano accedere alle informazioni in modo interoperabile.

2.3 I requisiti del gateway e la rete AUSL

Il gateway/server che riceve e salva i referti inviati dagli emogasanalizzatori dell'ospedale di Cesena, è situato a Forlì. Tutti i parametri sono inviati al gateway in formato HL7 di tipologia ORU, poiché esso è il messaggio che permette l'invio dei risultati delle osservazioni. Una volta ricevuti, oltre ad immagazzinarli, il gateway li reindirizza al nostro web-service, che li salva e li rende disponibili tramite opportune richieste HTTP.

Per quanto il nostro applicativo web abbia stabilito una connessione con il gateway, per questioni di sicurezza, non ci è possibile inoltrargli richieste relative ai dati degli esami. Infatti, il requisito più interessante che caratterizza il gateway, riguarda il fatto che non sia possibile interagirvi direttamente, ma occorra attendere che sia esso stesso ad inviarci i dati.

Il nostro web-service si inserisce quindi all'interno del sistema ospedaliero esistente, mantenendolo il più possibile inalterato. È da sottolineare che, al di là delle precauzioni messe in atto dall'intera struttura sanitaria, essa inoltre

usufruisca di una rete sicura e protetta, l'AUSL. Ed è all'interno di questa rete che avvengono tutte le comunicazioni tra il nostro web-service, il gateway ed i macchinari EGA.

Capitolo 3

Progettazione del sistema

Il terzo capitolo si apre con l'illustrazione dell'architettura del sistema, iniziando con la descrizione del tipo di comunicazione instaurata tra l'applicativo web ed i macchinari EGA, spiegando inoltre le funzionalità dell'applicativo stesso. A questo proposito si approfondisce il concetto di API REST esponendone caratteristiche e utilizzo. In seguito si passa alla descrizione della gestione dell'applicativo web, creato tramite Vert.x, un toolkit di cui si prende in analisi, in particolare, il modo in cui esso implementa le API REST. L'ultimo concetto toccato in questo capitolo, riguarda la descrizione delle tecnologie delle componenti che strutturano il web-service, approfondendo il motivo per cui sono state utilizzate.

3.1 Architettura del sistema

Come già specificato nel capitolo precedente, i messaggi HL7 che otteniamo non provengono direttamente dai macchinari EGA; questi ultimi infatti li inviano al gateway/server situato a Forlì dal quale avviene la comunicazione, essendo che ci è permesso di interfacciarci con esso. Quindi abbiamo a disposizione una comunicazione non diretta con i macchinari EGA all'interno dell'ospedale e diretta con il gateway. La connessione fra l'applicativo web e il gateway/server è TCP. Esso riceve i vari messaggi HL7 e deve essere in grado di analizzarli e filtrare solo quelli di interesse. Per fare ciò è messo in atto il processo di parsing, il quale analizza il messaggio e successivamente salva solo i valori di interesse, non in HL7, ma in un formato più semplice. Così facendo, nel momento in cui viene richiesta questa risorsa, il mittente non riceve l'intero messaggio ma solamente i dati desiderati disposti nella maniera più chiara possibile. Inoltre tutti questi messaggi devono essere sempre raggiungibili, per questo vengono salvati in un database e si è pensato di implementare il web-service con un back end che fornisca delle API REST.

I componenti principali del progetto sono il macchinario EGA che invia i risultati dei prelievi, il gateway/server che riceve i risultati di tutti i macchinari e li spedisce all'applicativo web che li immagazzina e li rende disponibili ai client. In Figura 3.1 vengono mostrate le iterazioni fra i vari componenti.

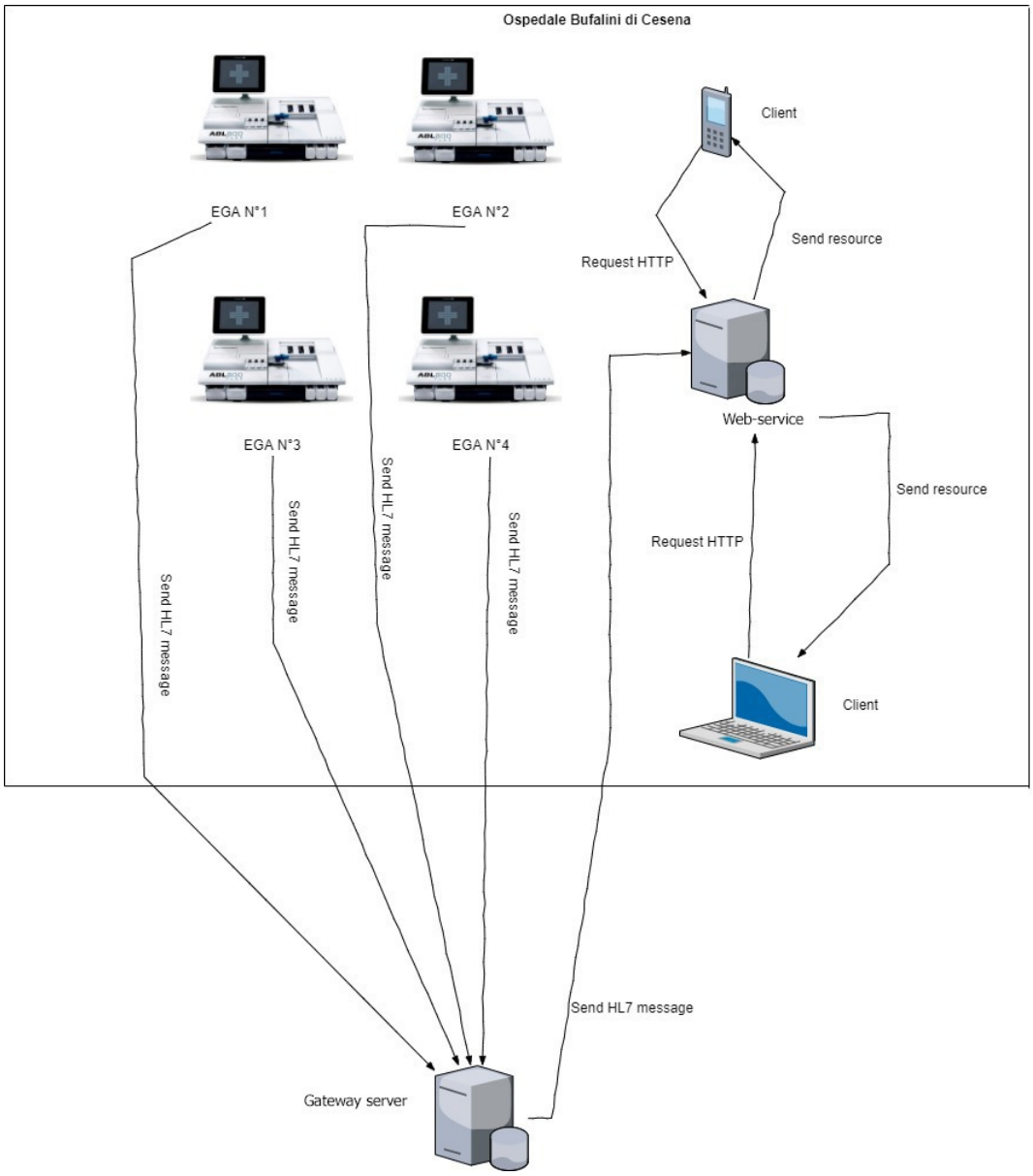

Figura 3.1: Architettura del sistema

3.2 Le API REST

Il web-service creato possiede quindi delle web API che ci permettono di raggiungere i messaggi HL7 desiderati. Siccome si è pensato di utilizzare il protocollo HTTP per effettuare le richieste dei dati, un'architettura software che rispetta le specifiche è il REST(Representational State Transfer). Infatti il REST è un sistema di trasmissione dei dati che utilizza l'HTTP e che fa un grande uso delle sue funzioni, ricorrendo al comando GET per il recupero di informazioni, POST, PUT, PATCH, DELETE per la modifica e OPTIONS per altri scopi. Questo tipo di architettura permette di nascondere le complessità lato server, fornendo solamente delle API che ai client risultano molto semplici da utilizzare. Per ora l'unico metodo HTTP che ci interessa è il GET che permette di richiedere delle risorse, cosa che noi dobbiamo fare in quanto ci occorre ottenere degli esami salvati in un database all'interno del web-service. Un concetto importante in REST è l'esistenza di risorse, nel nostro caso sono le principali informazioni di un esame EGA, a cui si può accedere tramite un identificatore globale (un URI). Un'altra caratterista del REST consiste nel fatto che sia stateless (privo di stato), cioè non è necessario implementare sull'applicativo web funzionalità per mantenere le sessioni con i client. Infatti, ciascuna richiesta da parte del client contiene tutte le informazioni di cui si ha bisogno.

Prima di passare all'implementazione, è importante decidere la struttura che deve avere il messaggio di risposta ricevuto dalle applicazioni, dopo aver effettuato un'opportuna richiesta HTTP. La modellazione delle API è stata effettuata tramite Swagger IO, il quale permette di rappresentare in maniera chiara i vari dettagli di cui sono composte. Lo scopo è di definire uno standard per la creazione e la documentazione di API REST, indipendentemente dal linguaggio di programmazione usato. Swagger IO è composto da un insieme di tool, nel nostro caso è stato utilizzato Swagger Editor. Quest'ultimo permette di definire l'interfaccia delle API tramite uno specifico linguaggio, in questa circostanza è stato adoperato YAML, che deriva dal JSON. Si potrebbero impiegare entrambi i linguaggi in realtà, ma la scelta è ricaduta proprio su YAML perché semplifica in maniera efficace la sintassi. Il JSON è invece utilizzato come formato per il messaggio di risposta del web-service verso il client.

Alla luce di ciò, è stata progettata un'opportuna API che offrisse al client la possibilità di ricevere le informazioni relative ad un esame. Per la costruzione del messaggio JSON bisogna considerare che gli esami EGA, ricevuti dal gateway sotto forma di messaggio HL7, possono essere composti da più segmenti OBX, ognuno dei quali contiene le informazioni relative a un parametro vitale. Il formato JSON di risposta è composto da due campi:

- ID: è il campo identificativo del messaggio e rappresenta appunto l'ID dell'esame richiesto
- Observation: questo campo è composto da un array di JSON, ognuno contenente i valori di uno specifico parametro vitale. L'array è composto da tanti JSON quanti sono i segmenti OBX all'interno dell'esame EGA

In Figura 3.2 è mostrata la modellazione dell'API tramite Swagger Editor.

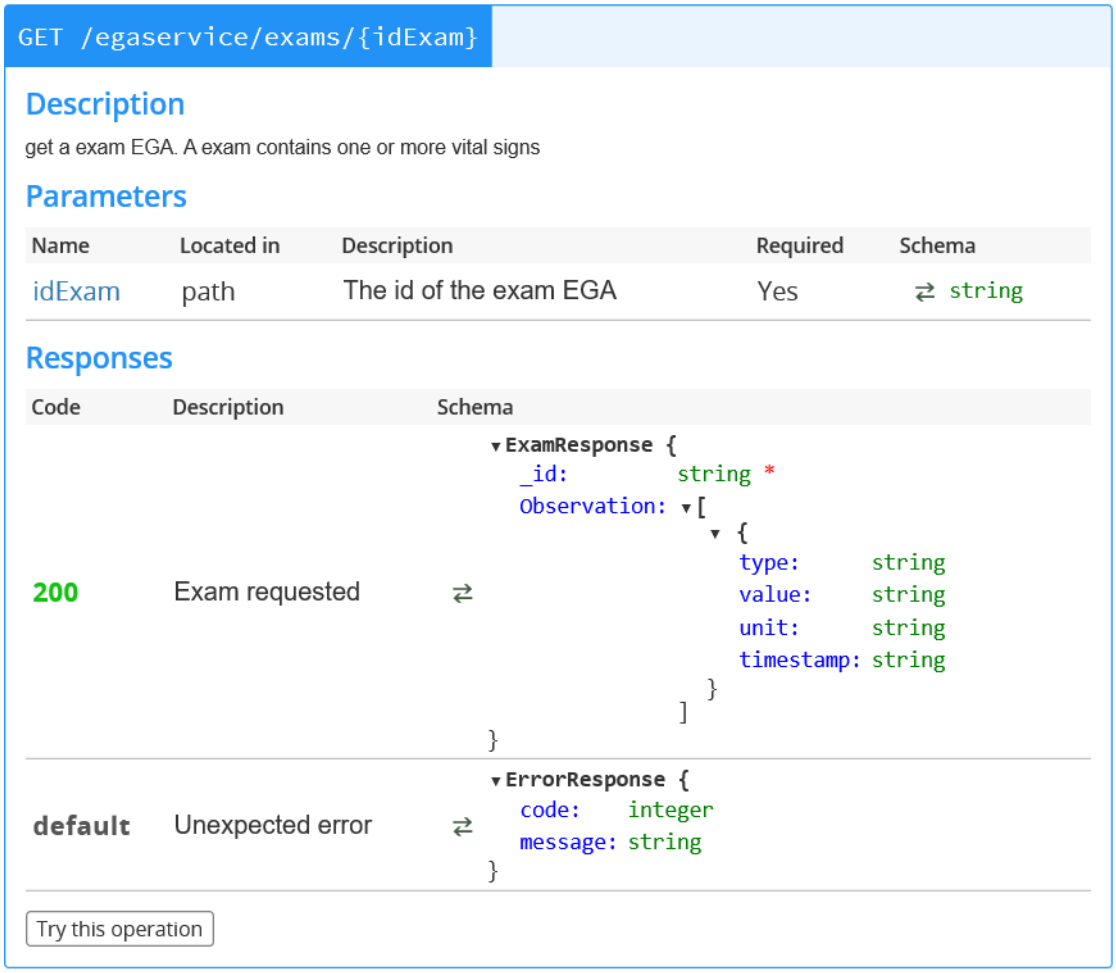

Figura 3.2: API modellata tramite Swagger Editor

3.3 Il funzionamento dell'applicativo web

L'applicativo web, programmato interamente tramite Java, deve svolgere contemporaneamente due funzionalità: gestire i messaggi HL7 ricevuti dal gateway e mettere a disposizione le risorse tramite API REST. Per questo motivo, è stato diviso in due componenti attive che si occupano separatamente dei due compiti.

Si è deciso di costruire l'applicativo web tramite Vert.x perché mette a disposizione delle funzionalità molto importanti per il progetto, come gestire l'accesso ai dati utilizzando database relazionali e non e lo sviluppo web con la possibilità di creare client e server. Vert.x è un toolkit che sfrutta il modello ad event loop e permette inoltre di non gestire manualmente aspetti critici di multithreading. Un'applicazione Vert.x è composta da uno o più Verticle i quali, se necessario, possono comunicare fra loro grazie all'event bus. Un'altra caratteristica che presenta è la possibilità di utilizzare l'architettura REST tramite il concetto di routing, cioè è possibile definire dei path i quali sono abbinati a dei handler specifici. Essi contengono le azioni da compiere per processare le richieste. Il path viene definito quando il client effettua una richiesta HTTP, all'interno dell'URI. Quest'ultimo è composto solitamente da 4 elementi che sono il protocollo e l'host, obbligatori, e la porta e il path che sono invece facoltativi. Un esempio di URI è *http://www.esempio.it:8070/risorsa* il quale è composto dal protocollo HTTP, dall'host *www.esempio.it*, dalla porta *8070* e dal path */risorsa*. Vert.x infine permette di creare all'interno del web-service le due componenti attive, le quali rappresentano due Verticle, descritti nei paragrafi successivi.

3.3.1 La gestione dei messaggi HL7

La componente dedicata alla gestione dei messaggi HL7, è programmata per riceverli dal gateway e modellarli tramite la libreria HAPI. Le principali operazioni compiute da questa libreria sono: creare i messaggi HL7 e esaminarli per poterli modellare ed effettuare operazioni di rete, come inviare un messaggio HL7. Una volta analizzato e filtrato il messaggio, ricavando solo le informazioni di interesse, avviene il salvataggio in un database. Il formato utilizzato è il JSON perché è semplice da capire e facile da utilizzare, proprio per questo, è il formato più comune per lo scambio di messaggi. Inoltre si è deciso di utilizzare un database NoSQL, chiamato MongoDB, per ottenere ottime prestazioni. Siccome il modello dei dati in questione può avere dei cambiamenti in futuro, l'approccio NoSQL è migliore perché può gestire con facilità eventuali variazioni. Il diagramma di sequenza in Figura 3.3, mostra le varie iterazioni degli elementi di questa componente.

3.3.2 La gestione delle richieste HTTP

Per quanto riguarda la gestione delle richieste HTTP, questa componente si occupa di fornire delle API per ricevere i dettagli degli esami. Essa sta in ascolto in attesa che un client faccia una richiesta e, una volta ricevuta, effettua una ricerca nel database, tramite una query, restituendo un JSON che incapsula le informazioni di una risorsa. Inizialmente abbiamo pensato di dare al client la possibilità di ricercare un esame effettuato inserendo nell'URI proprio l'ID di quell'esame. Ovviamente le ricerche possibili si possono ampliare, dipende tutto dalle esigenze dell'ospedale. Il diagramma di sequenza, in Figura 3.4, mostra le varie iterazioni degli elementi di questa componente.

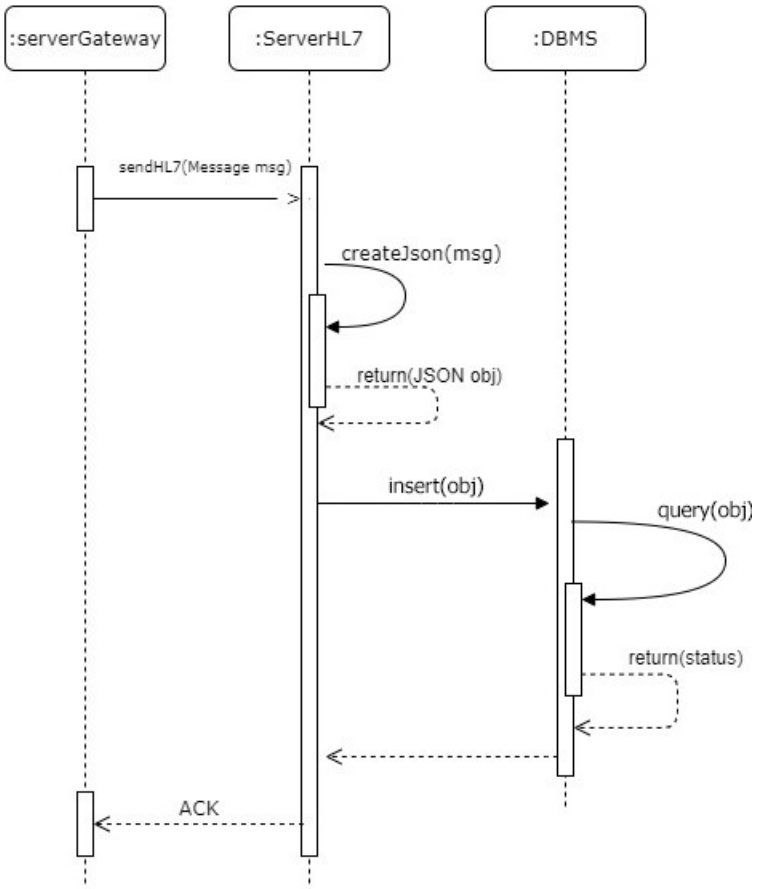

Figura 3.3: Diagramma di sequenza per la gestione dei messaggi HL7

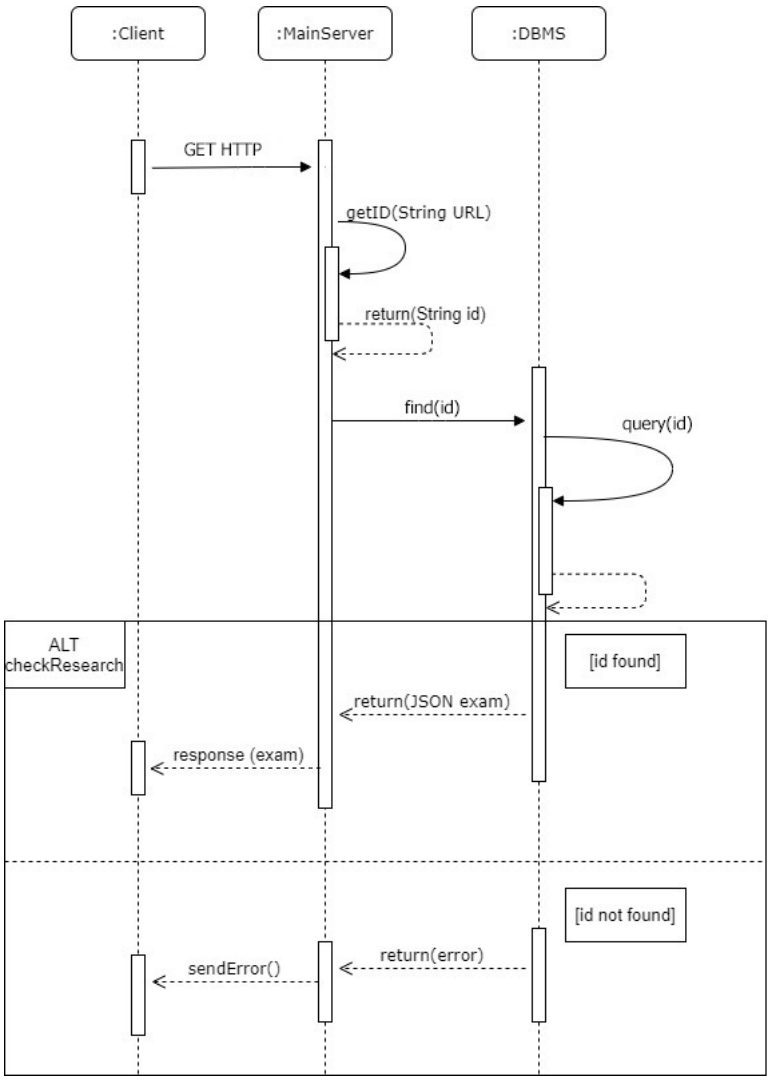

Figura 3.4: Diagramma di sequenza per la gestione delle richieste HTTP

Capitolo 4

L'implementazione prototipale

In questo capitolo viene trattata l'implementazione prototipale. Anzitutto è spiegata l'organizzazione del lavoro ed il modo in cui è stato svolto, scandendo i diversi passaggi compiuti, arrivando così alla fase di sviluppo dell'applicativo web, del quale sono riportate struttura e funzionalità. In seguito ci si focalizza sul client di prova e sulla sua comunicazione con il web-service, comprendendo approfondimenti sul messaggio ORU ed il messaggio ACK.

4.1 L'organizzazione del lavoro

È buona norma creare un sistema di simulazione per testare le funzionalità dell'applicativo web prima di inserirlo in un ambiente reale. Grazie alla simulazione è possibile capire e risolvere le problematiche in maniera più efficiente, perché tutto il sistema simulato, in questo caso, si trova all'interno di uno stesso dispositivo.

Come prima fase di lavoro non è stato creato il web-service, bensì un client di prova con la funzionalità di creare, ed in seguito inviare all'applicativo web, un messaggio HL7 identico a quelli spediti dal macchinario EGA.

Una volta accertato che il client lavorasse correttamente, si è passato alla creazione del web-service. Il primo obbiettivo stabilito, consisteva nel fatto che dovesse essere in grado di ricevere i messaggi HL7 spediti dal client di prova.

Ricevuto tale messaggio, il passo successivo è stato analizzarlo, ricavando da esso solamente i valori di interesse e salvando il tutto in formato JSON, inizialmente in una struttura dati chiamata MAP. Si è deciso di non inserire subito nel progetto il database perché è più complicato da implementare e, in un primo momento, volevamo avere in breve tempo un sistema funzionante in grado di salvare le informazioni apprese dal client e renderle disponibili tramite le richieste HTTP. Infatti, a questo punto, tramite Vert.x ho diviso

il web-service in due componenti attive: una incaricata di ricevere i messaggi HL7 e l'altra che espone delle API REST.

Una volta sviluppate le API REST, le ho provate tramite un software chiamato Postman, il quale permette di effettuare le richieste HTTP con un metodo specifico. Verificato che riuscissi a ricevere i messaggi JSON salvati nella MAP, l'ho sostituita con MongoDB, un database NoSQL.

4.2 L'applicativo web

Per quanto riguarda lo sviluppo del web-service, mi sono concentrato principalmente sugli aspetti legati ai requisiti e alle funzionalità del sistema. L'applicativo web, la cui struttura è mostrata in Figura 4.1, è composto da due componenti attive gestite dalle classi:

- MainServer: rappresenta la componente che gestisce le richieste HTTP
- ServerHL7: ovvero la componente che gestisce la ricezione dei messaggi HL7 ricevuti dal gateway
- OruEndPoint: utilizzato dalla classe ServerHL7 per processare i messaggi ORU

La classe MainServer è composta da una prima fase in cui si definiscono le API, nelle quali si assegnano i path ai corrispettivi handler, cioè un insieme di azioni per processare le richieste. È possibile vederne un esempio tramite questo codice(Listato 4.1):

```
router.get("/egaservice/exams/:examID").handler(this::handleGetExam);
```

Listato 4.1: Esempio di routing

L'URI corretto per poter eseguire l'handler chiamato handleGetExam, in questo caso deve:

- Utilizzare il protocollo HTTP con metodo GET
- Contenere l'indirizzo IP e la porta del web-service per poterlo raggiungere
- Utilizzare un path del tipo */egaservice/exams/:examID*, con *:examID* che dovrà essere sostituito con l'ID di un specifico esame, così che sarà possibile ricercarlo all'interno del database e spedirlo al client

Quindi una volta effettuata la richiesta HTTP corretta viene eseguito l'handler, il quale processa la richiesta e invia una risposta al client. L'handleGetExam(Listato 4.2) ha il compito di ricercare nel database l'esame tramite l'ID ricavato dall'URI della richiesta.

La seconda componente gestita dal web-service, invece, utilizza due classi che sono ServerHL7 e OruEndPoint. Per merito della libreria HAPI è stato possibile creare, avviare il server HL7, situato all'interno del web-service, e analizzare il messaggio HL7. Inoltre, siccome si possono aggiungere degli EndPoint con dei messaggi HL7 specifici, ci ha permesso di filtrare i messaggi solo della tipologia desiderata, cioè ORU. Di seguito, ovvero nel Listato 4.3, viene mostrato un frammento di codice nel quale avviene il filtraggio dei messaggi ORU.

```
Application handlerORUR01 = new OruEndPoint(vertx);
 //filters ORU messages with any trigger event
server.registerApplication("ORU", "*", handlerORUR01);
```

Listato 4.3: Registrazione ORU nel web-service

Così facendo, nel momento in cui il l'applicativo web riceve quel particolare messaggio, in automatico viene eseguita la funzione specifica processMessagge(Message msg) all'interno della classe OruEndPoint. Siccome questa funzione riceve come parametro proprio il messaggio HL7, non resta che trasformarlo in formato JSON, inserendo al suo interno solo i campi di interesse del messaggio HL7 originario.

Ogni JSON è identificato dall'ID dell'esame EGA, che si trova all'interno del messaggio HL7. Successivamente si salva il JSON all'interno del database MongoDB e da questo punto in poi è possibile, tramite opportuna richiesta HTTP, riprendersi queste informazioni.

Tutti i messaggi JSON salvati nel database vengono chiamati documenti e sono tutti situati all'interno di una stessa collezione che ho chiamato *ExamsEga*.

La caratteristica di salvare i dati sotto forma di documento è tipica all'interno dei database NoSQL e permette di semplificare la ricerca e la memorizzazione di dati, aumentando le prestazioni. Infine il web-service invia il messaggio di ACK al client per confermare la ricezione del messaggio.

```
private void handleGetExam(RoutingContext routingContext) {
     //ricavo dall'url l'ID del esame
       String examID = routingContext.request().getParam("examID");
       //utilizzo MongoDB per salvare i vari report
       MongoClient mongoClient = MongoClient.createShared(vertx,
           createConfig());
       HttpServerResponse response = routingContext.response();
       if (examID == null) {
        sendError(400, response);
       } else {
        //creo una query che ricerca l'esame del database
        JsonObject query = new JsonObject().put("_id", examID);
        //effettuo la ricerca
        mongoClient.find("ExamsEga", query, res -> {
          //Errore ricerca
            if (!res.succeeded()) {
               sendError(400, response);
             }
            //Esame trovato
            else if(!res.result().isEmpty()){
                 response.putHeader("content-type",
                     "application/json")
                         .end(res.result().get(0).toString());
            }//Caso in cui non ha trovato l'esame
            else {
               sendError(404, response);
            }
        });
       }
   }
```

Listato 4.2: Funzione handleGetExam

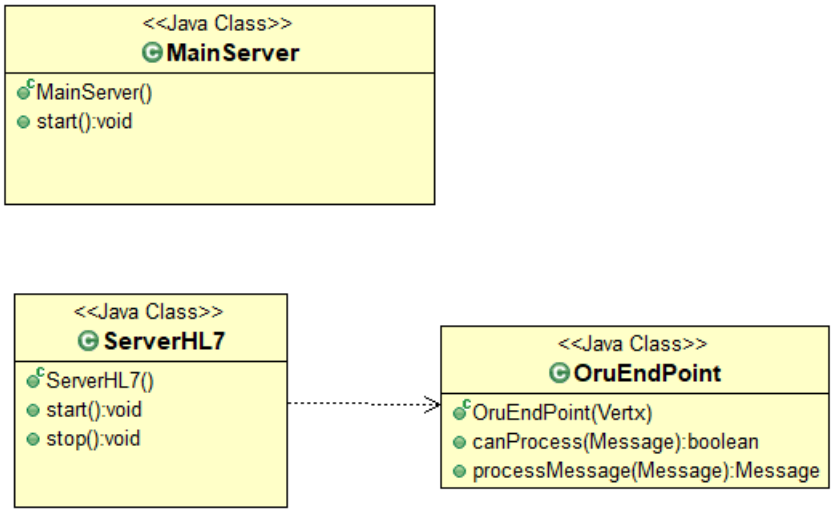

Figura 4.1: Diagramma delle classi UML del web-service

4.3 Il client di prova

Il client è stato creato per testare il funzionamento dell'applicativo web. Come prima fase si è creato il messaggio HL7 di tipologia ORU, tramite il linguaggio Java, utilizzando la libreria HAPI. Questa libreria supporta tutte le versioni 2.x dello standard HL7, infatti la versione del messaggio è la 2.4 proprio perché i messaggi reali inviati dai macchinari EGA utilizzano tale versione. La libreria HAPI permette di creare un messaggio HL7 in differenti modalità:

- Scrivere il messaggio sotto forma di stringa e successivamente trasformarlo in messaggio HL7
- Creare il messaggio mediante la programmazione ad oggetti; HAPI offre di fatti la possibilità di utilizzare metodi specifici per settare ogni singolo campo di un messaggio HL7

Una volta che è stato creato in maniera corretta, si passa alla comunicazione con il web-service. Per far ciò bisogna inserire nel client l'IP che, siccome si lavora in locale sarà *localhost*, e la porta dell'applicativo web. Inviato il messaggio, il client attende l'arrivo dell'ACK, segnale di controllo trasmesso

dal web-service per confermare la ricezione del messaggio stesso. Nel caso l'ACK non dovesse ritornare, significherebbe probabilmente che il messaggio inviato non ha raggiunto il web-service. Ciò accadrebbe, ad esempio, nel caso in cui il client inviasse le informazioni ad un IP o porta differenti da quelle che realmente possiede l'applicativo web, oppure se quest'ultimo non fosse attivo. Invece, ogni volta che il client riceve l'ACK, esso viene processato perché offre la possibilità di capire se:

- Il messaggio ORU è stato accettato
- C'è stato un errore nel processare la richiesta
- Il messaggio ORU è stato rifiutato

Infine, per far sì che il client sia robusto, è necessario gestire le varie casistiche di errore controllando, per esempio, che la connessione sia valida, oppure accertandosi che il messaggio HL7 inviato dal client sia corretto. Di questi casi di errore è possibile accorgersi direttamente lato client, il che li rende problematiche facilmente comprensibili.

4.3.1 Il Messaggio ORU

Il messaggio ORU inviato dal client è di tipo *unsolicited* e unidirezionale; in questo caso è il client ad iniziare la connessione ed una volta stabilita, invia il messaggio al web-service. Il sistema ricevente deve rispondere con un messaggio di tipo ACK.

In Figura 4.2 è mostrato un esempio di messaggio HL7 di tipologia ORU. Qui di seguito è riportata la descrizione delle componenti principali di questo messaggio. Tra parentesi sono indicati i valori corrispondenti all'interno della Figura 4.2.

- MSH Segment - Message Header:
  - Sending application (HIS, hospital information systems): nome dell'applicativo che invia il messaggio.
  - Receiving application (RapidComm): nome dell'applicativo a cui è destinato il messaggio.
  - Message date and time (201506051656): timestamp del messaggio nella forma YYYYMMDDHHmm. Rappresenta la data in cui il messaggio viene inviato.
  - Message type (ORUˆR32): indica la tipologia di messaggio.

- Message Control ID (0C0634M000Z00001040): numero univoco identificativo del messaggio.
- Version ID (2.3): indica la versione HL7 usata.

```
MSH|^~\&|Rapidcomm|Hospital|HIS|Hospital|201506051656||ORU^R32|0C0634M0
00Z000010404|P|2.3|||AL|AL|
PID|1||MRN^^^||LNAME^FNAME^MNAME^^^||DOB|M|||ADDR^^CITY^STATE^PCODE^^^|
|PHONE|||||ACCOUNT^^^|SSN|||||||
PV1|1|I|FLOOR^ROOM^BED^^|E|||APHYID^APHYNAME^^^^^^|RPHYID^RPHYNAME^^^^^
^||||||||||I|001|||||||||||||||||||||||201506051656||||||
ORC|NW|ACCESSION|^|^|IP||001^^^201506061014^201506061014^S^^^^||2015060
61014|||OPHYID^OPHYNAME ^^^^^^|||||||
OBR|1|PORDER|FORDER|ABG^BG-BLOOD GAS ANALYSIS^^^^||||||S^^^^^^^
|O|||||OPHYID^OPHYNAME^^^^^^||||||||||001^^^201506061014^201506061014^
S^^^^| ||||||||201506061014|
NTE|1|L|Patient semicomatose|
OBX|1|ST|pH||7.350||7.350-7.450||||F|||20150601112058||
ROID^ROLNAME^ROFNAME||^^1001^RAPIDLab 1265|20150601112058|
NTE|1|L|MIC|
OBX|2|ST|pCO2||33.0|mmHg|35.0-45.0|L|||F|
NTE|1|L|MIC|
OBX|3|ST|pO2||88.0|mmHg|75.0-100.0||||F|
NTE|1|L|MIC|
OBX|4|ST|Na+||140.0|mmol/L|135.0-148.0||||F|
NTE|1|L|MIC|
OBX|5|ST|K+||3.50|mmol/L|3.50-5.30||||F|
NTE|1|L|MIC|
OBX|6|ST|Ca++||1.02|mmol/L|1.13-1.32|L|||F|
```

Figura 4.2: Esempio di messaggio ORU

- PID Segment - Patient Identification:
  - Medical record number: contiene il codice identificativo del paziente
  - Patient name: questo campo contiene principalmente il nome e cognome del paziente.
- PV1 Segment - Patient Visit:
  - Assigned Patient Location: definisce il punto di cura, la stanza e il letto del paziente.
  - Referring physician ID: rappresenta l'ID del medico di riferimento
  - Admit date and time (201506051656): indica la data e l'ora di ammissione
- OBR Segment - Observation Request:

  - Service identifier text (BG-BLOOD GAS ANALYSIS): questo testo permette di identificare un servizio
  - Start date and time (201506061014): timestamp in formato YYYYMMDDHHmm che identifica la data in cui l'osservazione è stata effettuata

- OBX Segment - Observation/Result:
  - Observation identifier: identifica un parametro vitale, come ad esempio il pH
  - Observation value: il valore del parametro
  - Units: rappresenta l'unità di misura con cui il parametro è riportato
  - Observation date and time: timestamp in formato YYYYMMDDHHmms che identifica la data in cui l'osservazione è stata effettuata. In un messaggio possono esserci più segmenti OBX, poiché da una stessa osservazione possono derivarne più di uno, ma il valore di questo campo resta per tutti lo stesso. Per questo motivo, può accadere che venga inserito solo nel primo segmento, per evitare ripetizioni

4.3.2 Il Messaggio ACK

In risposta al messaggio ORU inviato dal client si ottiene un messaggio ACK, mostrato in Figura 4.3. Esso permette al client di capire se il messaggio che ha inviato è arrivato correttamente o con degli errori.

```
MSH|^~\&|HIS|Hospital|Rapidcomm|Hospital|200506061014||ACK|
17-1-35:CC1JGE|P|2.3|||NE|NE|
MSA|AA|02-08T15:16:58D2I1S2|
```

Figura 4.3: Esempio di messaggio ACK

I segmenti che compongono questo messaggio e i loro campi principali sono:

- MSH Segment - Message Header
  - Sending application (Rapidcomm): nome dell'applicativo che invia il messaggio.
  - Receiving application (HIS, hospital information systems): nome dell'applicativo a cui è destinato il messaggio.

  - Message date and time (201506051656): timestamp del messaggio nella forma YYYYMMDDHHmm. Rappresenta la data in cui il messaggio viene inviato.
  - Message type (ORUˆR32): indica la tipologia di messaggio.
  - Message Control ID (0C0634M000Z00001040): numero univoco identificativo del messaggio.
  - Version ID (2.3): indica la versione HL7 usata.

- MSA Segment - Message acknowledgement
  - Acknowledgement code (AA): rappresenta lo stato del messaggio ORU inviato. Può assumere 3 valori che sono: AA (Application Accept), il che significa che la richiesta è stata accettata, AE (Application Error) ovvero c'è stato un errore nel processare la richiesta e AR (Application Reject) che indica che la richiesta è stata rifiutata.
  - Message control ID of original message (02-08T15:16:58D2I1S2): contiene lo stesso Message control ID del messaggio di richiesta, in questo caso del messaggio ORU.

Capitolo 5

Validazione

Gli ultimi aspetti presi in analisi all'interno della tesi sono i test del sistema. In seguito ad un'introduzione riguardante la modalità in cui web-service e client di prova siano stati fatti comunicare in rete, si passa alla descrizione dei test riguardanti le funzionalità del sistema, le API e, da ultimo, sulle casistiche di errore.

Sono poi affrontati i possibili sviluppi futuri che l'intero progetto potrebbe avere, comprese le indicazioni sui perfezionamenti da attuare nel momento in cui verrà effettivamente sperimentato all'interno dell'ospedale.

5.1 Test del sistema

Prima di effettuare i test è necessario che il web-service e il client di prova riescano a comunicare in rete. Siccome i test all'interno del sistema simulato si effettuano in locale, l'IP utilizzato è 127.0.0.1 chiamato *localhost*, che appunto è usato per far comunicare applicazioni nello stesso sistema, invece quello che permette di differenziare le varie richieste sono le porte. Infatti, per effettuare i test di prova, il web-service mette a disposizione la porta 8090 per ricevere i messaggi HL7 dal client, che quindi deve essere configurato per inviare ad essa i dati, e la porta 8080 per mettere a disposizione le API REST.

Inoltre l'applicativo web deve comunicare con il database per salvare e recuperare le informazioni e, per poter essere utilizzato, possiede anch'esso una porta che deve essere opportunamente inserita nell'applicativo web. Una volta finita la configurazione di rete è possibile iniziare il test.

Anzitutto si esegue il web-service che rimarrà in attesa di richieste, per farlo ho utilizzato Gradle, un potente tool di *build automation*. Successivamente si esegue il client che invia un messaggio HL7 al web-service. A questo punto è possibile notare, all'interno del database, se l'applicativo web ha lavorato correttamente salvando il messaggio del client al suo interno. In Figura 5.1

sono mostrati alcuni report salvati dai messaggi HL7 di prova inviati dal client. Come è possibile notare, non viene salvato l'intero messaggio ma solo i campi di interesse che sono:

- id: il quale rappresenta il numero identificativo dell'esame
- type: il cui ruolo è di identificare un parametro vitale
- value: ovvero il valore rilevato per questo specifico parametro vitale
- unit: l'unità di misura del valore rilevato
- timestamp: la data in cui è stata effettuata l'osservazione in formato YYYYMMDDHHmm.

Successivamente per poter testare le API e quindi ricevere questi report salvati nel database, ho utilizzato un programma chiamato Postman che permette appunto di inviare una richiesta HTTP con un metodo specifico, nel nostro caso una GET, e mostrare la risposta del web-service a questa richiesta. L'utilizzo di Postman è mostrato in Figura 5.2: in questo caso sono richieste le informazioni relative all'esame con ID 0C0634M000Z00001040.

È possibile notare che i dati dell'esame ricevuto dal web-service, con ID uguale a 0C0634M000Z00001040, sono gli stessi dell'esame con il medesimo ID mostrato in Figura 5.1, quindi questo significa che l'applicativo web ha lavorato correttamente, prendendo la risorsa giusta dal database. Oltre ai test relativi alle funzionalità, sono state testate anche le casistiche di errore, qui elencate:

- Connessione con web-service non riuscita: nel caso in cui il client provi a connettersi con l'applicativo web, ma esso non risulti raggiungibile,tale problema viene visualizzato lato client
- Creazione e invio messaggio HL7 sbagliato: nel caso in cui il client invii un messaggio HL7 con qualche errore, ad esempio inserendo in un campo il nome del paziente quando in realtà si aspettava una data, esso se ne accorge mentre processa l'ACK ricevuto dal web-service, mostrando di conseguenza la problematica
- Richiesta risorsa inesistente: questo errore si presenta nel momento in cui si richieda all'applicativo web l'ID di un esame che non è presente nel database; ho gestito questa problematica facendo sì che, se avvenisse, venga inviato in risposta al client un messaggio di errore senza bloccare il funzionamento del web-service

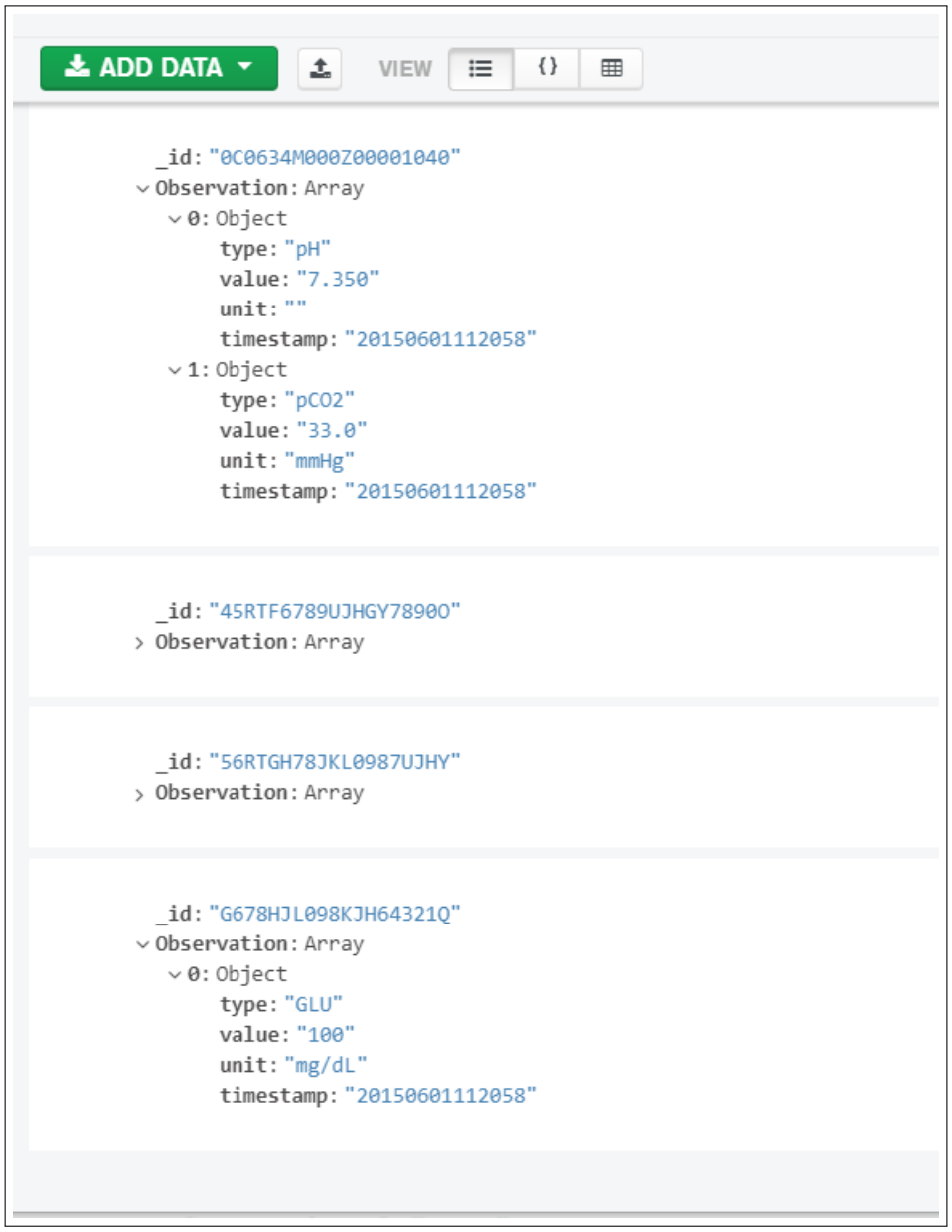

Figura 5.1: Visualizzazione di alcuni report all'interno di MongoDB

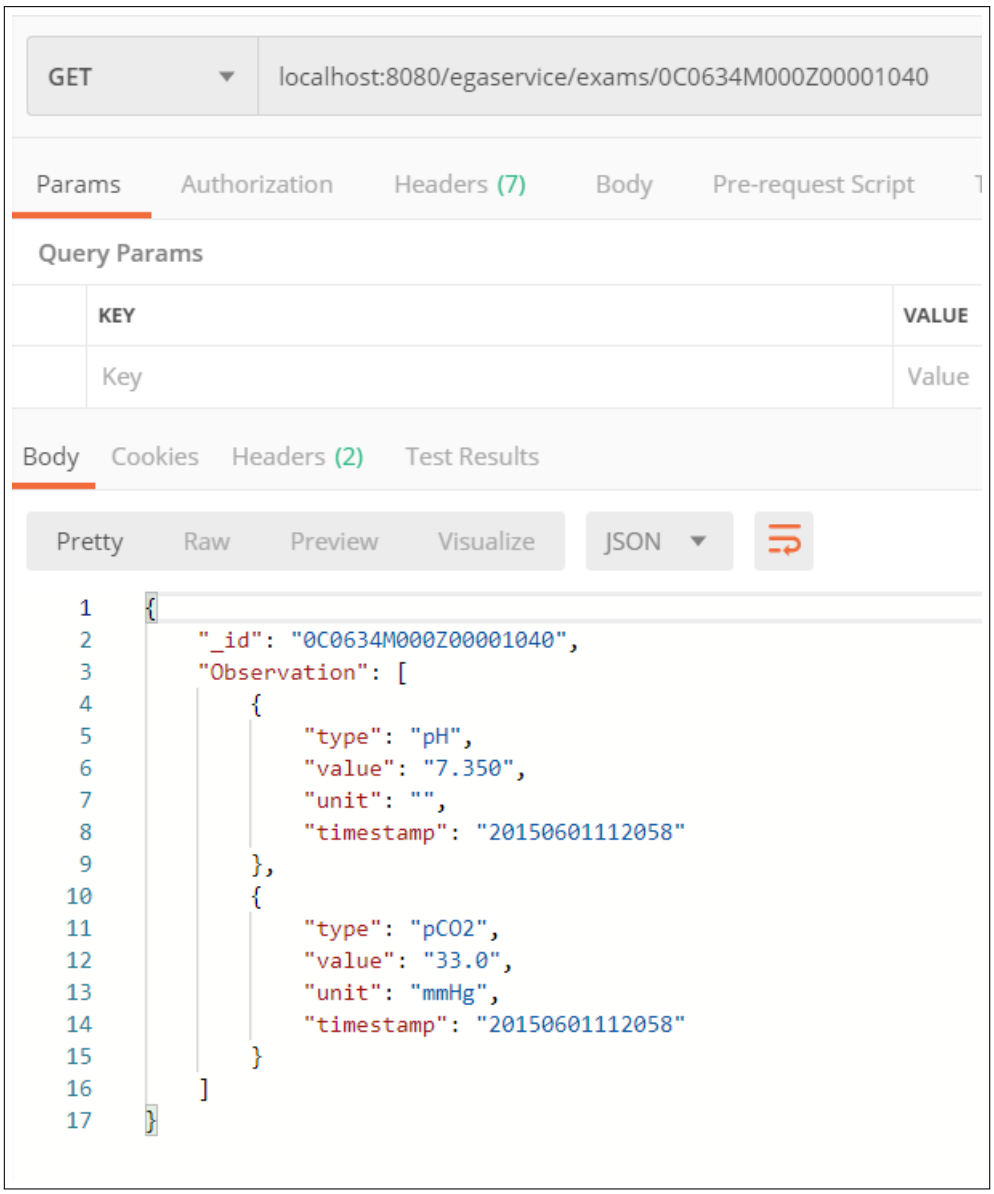

Figura 5.2: Parte del programma Postman dopo aver effettuato una richiesta HTTP

5.2 Sviluppi futuri

Ho cercato di rendere il progetto più modulare possibile al fine di facilitarne una modifica futura, che sarà sicuramente necessaria per affinare e migliorare il sistema. In particolare potrebbero emergere nuovi requisiti come:

- Nuove API REST: la creazione di nuove API dipenderà principalmente dalle esigenze dell'ospedale. Nel caso vorrà effettuare nuove ricerche all'interno del database, ad esempio trovare tutti gli esami effettuati in una singola giornata oppure tutti gli esami effettuati da un specifico macchinario, bisognerà creare nuove API REST per gestire tali problematiche.

- Estendere le applicazioni esistenti presso l'ospedale "M.Bufalini" di Cesena – in particolare TraumaTracker – in modo da consentire la comunicazione con le API del servizio realizzato, allo scopo di consentire loro di recuperare automaticamente gli esiti degli esami EGA prodotti dalle apparecchiature d'interesse e trasmessi al servizio sviluppato nell'ambito di questa tesi.

- Gestione di più macchinari: Oltre ai macchinari EGA, potrà essere implementato nell'applicativo web la gestione di altri macchinari, per permettere un'importante evoluzione all'interno dell'ospedale, che però non sarà mai definitiva visto il continuo progredire delle tecnologie.

C'è la possibilità che l'applicativo web debba essere affinato nel momento in cui il sistema verrà sperimentato in ospedale. I test effettuati in locale potrebbero non essere sufficienti, perché lo scenario all'interno dell'ospedale non è esattamente lo stesso.

Altri aspetti molto importanti che dovranno essere implementati riguardano la sicurezza informatica, nonostante il sistema ospedaliero sia già sicuro grazie all'utilizzo della rete protetta AUSL. Ad esempio l'applicativo web potrà essere fornito tramite il protocollo HTTPS, in modo tale che la comunicazione con i client o con il gateway sia ulteriormente sicura.

Conclusioni

Questo studio si confronta con il problema dell'accesso interoperabile ai dati sanitari prodotti dai dispositivi medici attivi, oggigiorno sempre più presenti in ambito sanitario. L'informatica permette una riduzione dei tempi per effettuare le operazioni interne e, di conseguenza, porta ad un aumento della produttività, il che è molto importante soprattutto in ambiti così delicati come le strutture sanitarie.

Nello specifico il progetto di tesi sviluppato, grazie alla collaborazione con l'ospedale "M.Bufalini" di Cesena, offre la possibilità di accedere in modo interoperabile e moderno, attraverso un web-service con interfaccia REST, ai dati prodotti dal dispositivo medico EGA, diversamente accessibili unicamente attraverso gateway proprietario e trasmessi via protocollo HL7. Attualmente infatti, l'ospedale usa salvare tali risultati in formato cartaceo, cosa che non sarà più necessaria grazie alla sua sostituzione con il nuovo sistema che è interamente digitale. Così facendo si avrà la possibilità di salvare ed effettuare ricerche relative ad un esame o ad un gruppo specifico di esami in maniera quasi immediata, agevolando quindi il lavoro svolto dagli operatori sanitari. Per queste ragioni si è creato un web-service che interagisce con i sistemi interni all'ospedale, senza alterare il loro funzionamento.

Il presente studio è strettamente collegato alla quarta rivoluzione industriale, recentemente in atto, la quale incarna una nuova era in cui l'innovazione tecnologica è la risposta per rimanere efficienti in ogni settore.

In conclusione, questo progetto ha confermato l'importanza dell'inserimento delle tecnologie informatiche e dell'Internet of Things (IoT) in un settore come quello medico. In particolare, per quanto il web-service potrebbe necessitare di alcuni accorgimenti una volta inserito nel contesto reale, sarà in grado di fungere da mediatore tra gli emogasanalizzatori dell'ospedale con cui abbiamo collaborato e qualsiasi tipo di moderno sistema software, tra cui smartphone, computer e TraumaTracker. Il progetto di tesi ha quindi i requisiti per migliorare notevolmente il processo di fruizione e gestione dei dati di apparati medici, modernizzando e rendendo più efficiente la struttura ospedaliera "M.Bufalini" di Cesena.

Ringraziamenti

Al termine di questi tre anni di studi, non posso non ringraziare coloro che sono sempre rimasti accanto.

In primis, un ringraziamento speciale al mio relatore e correlatore, i professori A.Ricci e A.Croatti, che mi hanno affiancato durate lo sviluppo del progetto e la stesura della tesi con disponibilità e grande competenza. Grazie alla mia famiglia, che mi ha sempre sostenuto nel mio percorso di studi e non solo. Grazie alla mia ragazza, Sara, che mi è stata accanto durante i lunghi mesi di lavoro. Grazie per tutto il tempo che mi hai dedicato. Grazie perché ci sei sempre stata. Grazie ai miei amici per aver sempre creduto in me e nelle mie capacità.

Bibliografia

[1] HL7 - http://www.hl7.org/.

[2] HAPI - The Open Source HL7 API for Java - https://hapifhir.github.io/hapi-hl7v2/.

[3] Vert.x - http://vertx.io/.

[4] Swagger IO - http://swagger.io/.

[5] MongoDB - https://www.mongodb.com/it.

[6] JSON - https://www.baeldung.com/java-org-json.

[7] YAML - https://www.tutorialspoint.com/yaml/index.htm.

[8] NoSQL - https://www.geeksforgeeks.org/introduction-to-nosql/.

[9] Postman - https://www.postman.com/.

[10] Jason H. Christensen. *Using RESTful web-services and cloud computing to create next generation mobile applications*. ACM, 2009.

[11] Eysenbach G. *What is e-health?* J Med Internet Res, 2001.

[12] D. Gubiani. *Introduzione ai Sistemi Informativi e alle Basi di Dati in Ambiente Medico*. 2011.

[13] Jim Webber Ian Robinson, Savas Parastatidis. *REST in Practice: Hypermedia and Systems Architecture*. O'Reilly Media, 2010.

[14] Health Information and Quality Authority. *Overview of Healthcare Interoperability Standards*. 2013.

[15] Jaehong Kim. *Understanding Vert.x Architecture - Part II*. 2013.

[16] Grant M. Wood. *HL7 Basic Overview*. HIMSS Las Vegas, 2012.