ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

PROGETTAZIONE E SVILUPPO DI UN SISTEMA A SUPPORTO DEL PERSONALE MEDICO PER LA GESTIONE DEI REFERTI DELL'EMOGASANALISI

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Corelatore
Prof. ANGELO CROATTI

Presentata da
IVAN MAZZANTI

Anno Accademico 2019 – 2020

A tutte le persone care della mia vita

Indice

Introduzione

Al giorno d'oggi, l'introduzione e l'estensione delle tecnologie e dei sistemi informatici in diversi ambiti ha portato grandi vantaggi in termini di velocità, efficienza ed efficacia dell'attività richiesta. In particolare, nell'ambito attualmente in grande sviluppo dell'health information technology[3], cioè di tutti i sistemi informatici e standard sviluppati nel contesto dell'assistenza sanitaria, il supporto informatico ricopre un'importanza ancora maggiore perché consente non solo di aiutare il personale medico nel suo importante lavoro quotidiano ma anche di potenziare considerevolmente la cura e il trattamento dei pazienti. In questo contesto, uno dei problemi più significativi che il personale medico deve affrontare giornalmente è il problema del tracciamento, ovvero la raccolta di informazioni dettagliate sulle condizioni del paziente, sui risultati delle analisi effettuate, sui farmaci e altri trattamenti somministrati unita all'identificazione e alla localizzazione delle risorse sanitarie e del personale medico. L'acquisizione di queste informazioni è fondamentale in quanto consente di gestire al meglio la cura del paziente, velocizzare le operazioni, ridurre le probabilità di errore ed effettuare una revisione delle operazioni svolte per analizzarne i punti di forza ed eventuali margini di miglioramento. L'introduzione dei sistemi e delle tecniche informatiche a supporto del tracciamento porta indubbiamente un fondamentale incremento dell'efficienza dell'assistenza sanitaria, conferendo la possibilità di velocizzare e automatizzare le operazioni richieste. L'importanza del tracciamento aumenta ulteriormente se si considerano tutte le patologie che richiedono un intervento rapido, ben organizzato e in cui sono richieste funzionalità aggiuntive come la raccolta in tempo reale di ogni dato ed evento rilevante alla gestione del trauma e alla cura del paziente.

Per far fronte a queste esigenze, il Trauma Center dell'ospedale "M. Bufalini" di Cesena ha intrapreso un progetto di collaborazione con il dipartimento di Informatica - Scienza e ingegneria dell'università che ha portato alla progettazione e allo sviluppo di TraumaTracker, un sistema volto al completo supporto del tracciamento, in grado non solo di acquisire tutti i dati necessari alla gestione del trauma ma anche di monitorare l'intera missione di soccorso supportando il Trauma team sia nella fase pre-ospedaliera sia nella fase intra-ospedaliera.

Nel contesto della gestione del trauma, un esame molto importante e che richiede un'accurata ed efficiente gestione dei relativi referti è l'emogasanalisi. L'emogasanalisi permette di misurare le pressioni dei gas arteriosi e ottenere un insieme di parametri fondamentali per comprendere lo stato del trauma e organizzare i trattamenti necessari. Tuttavia, in TraumaTracker non è ancora presente un'integrazione che permetta di gestire, consultare e registrare automaticamente i referti prodotti dagli emogasanalizzatori dell'ospedale. Per supportare la gestione automatica dei referti, gli emogasanalizzatori inviano i dati relativi a un server di proprietà di Siemens Healthcare, la società di produzione degli emogasanalizatori. Questo server è programmabile per inoltrare tutti i dati memorizzati a qualsiasi sistema predisposto alla ricezione. I dati inviati sono strutturati secondo il formato HL7, lo standard di riferimento a livello globale per lo scambio di informazioni e dati clinici. Nell'ambito della collaborazione che ha portato alla nascita di TraumaTracker, è stata implementata una prima versione prototipale di un sistema che fosse in grado di ricevere questi messaggi e consentisse la facile consultazione e gestione dei referti. Tuttavia, questo prototipo offre funzionalità molto limitate, non ricava informazioni sufficienti dal messaggio HL7 e non permette una facile consultazione e gestione dei referti. Queste limitazioni rendono il prototipo completamente inutilizzabile dal personale medico, che si vede tuttora costretto a registrare, consultare e gestire manualmente i referti dell'emogasanalisi.

L'obiettivo della testi è quindi una sostanziale e importante estensione e integrazione del prototipo, che consenta di rispondere adeguatamente a tutte le esigenze introdotte. In aggiunta, si vuole non solo consentire l'integrazione con TraumaTracker ma anche rendere le funzionalità del sistema facilmente usufruibili da tutti i dispositivi autorizzati del sistema ospedaliero. In questo modo, il personale medico potrà consultare e gestire facilmente, velocemente e agevolmente i referti dell'emogasanalisi.

La trattazione della tesi è sviluppata in quattro capitoli. Nel primo capitolo viene presentato il contesto di riferimento, viene spiegata l'importanza del tracciamento, sono introdotti TraumaTracker e l'emogasanalisi. Viene poi approfondito lo standard HL7 con particolare enfasi sui dati più rilevanti per ottemperare agli obiettivi prefissati. Vengono infine illustrati nel dettaglio gli obiettivi della tesi. Il capitolo due si apre con la descrizione dettagliata del prototipo presentando le funzionalità implementate e analizzando le numerose mancanze e inadeguatezze. Segue poi un elenco dei requisiti tecnici che occorre introdurre e le tecnologie che saranno utilizzate in fase di implementazione. Dopo un'importante considerazione sulla privacy e sicurezza dei dati, il capitolo si conclude presentando il modello del dominio, lo standard FHIR e il diagramma dei casi d'uso. Il terzo capitolo illustra la progettazione del sistema, descrivendone l'architettura e le API predisposte alla gestione dei re-

ferti. Segue poi la trattazione dell'implementazione accompagnata da esempi di codice delle funzionalità più significative. L'ultimo capitolo introduce la virtualizzazione e la conteinerizzazione per poi esaminare il deployment con Docker. La tesi si conclude con delle valutazioni su quanto prodotto e delle considerazioni sugli sviluppi futuri.

Capitolo 1

Il tracciamento del trauma e l'emogasanalisi

Nel primo capitolo si introduce il problema del tracciamento nell'ambito dell'assistenza sanitaria, una sfida molto importante che, nell'elevato numero di scenari ospedalieri e trattamento dei pazienti, il personale medico deve affrontare quotidianamente. In questo contesto viene presentato TraumaTracker, un sistema in grado di rivoluzionare la gestione del tracciamento e che consente di velocizzare, migliorare e potenziare il lavoro del personale medico e la sicurezza dei pazienti. Si introduce poi l'emogasanalisi e gli emogasanalizzatori presenti all'ospedale "M. Bufalini" di Cesena, sottolineando quanto sia importante provvedere una gestione automatizzata dei relativi referti clinici e l'integrazione di questi in TraumaTracker. Viene successivamente analizzato lo standard HL7, il formato di riferimento per l'invio e la ricezione di informazioni cliniche, descrivendone le sue componenti più significative. Il capitolo si conclude con l'introduzione del prototipo preesistente e la definizione degli obiettivi che si vuole raggiungere.

1.1 Il contesto

Nell'ambito dell'assistenza sanitaria, uno dei problemi più significativi e importanti è il problema del tracciamento, ovvero l'identificazione e la localizzazione delle risorse sanitarie e del personale medico unita alla raccolta di informazioni dettagliate sulle condizioni del paziente, sulle analisi effettuate e su eventuali farmaci somministrati. L'acquisizione di queste informazioni apre la possibilità a un notevole miglioramento della gestione, dell'efficienza e dell'efficacia dell'assistenza sanitaria, nonché a una considerevole riduzione degli errori, dei tempi di lavoro e dei costi complessivi, portando dunque un importante contributo al lavoro del personale medico e, di conseguenza, un prezioso

aumento della sicurezza nella cura dei pazienti. Avere questi dati a disposizione infatti, significa non solo ottenere la possibilità di localizzare velocemente tutte le risorse necessarie alla cura del paziente, ma anche poter verificare a posteriori tutte le operazioni svolte analizzandone l'efficacia, i punti di forza, gli eventuali errori, poter capire se e dove si è perso tempo e cosa può essere migliorato.

Tra l'elevato numero di scenari ospedalieri e cura delle malattie, vi sono una serie di patologie che si dicono "tempo-dipendenti", ovvero, il risultato è fortemente migliorato se il paziente viene assistito prontamente (si pensi ad esempio a gravi condizioni come ictus, soffocamento, infarto ecc.). Per la categoria appena descritta, un efficiente sistema di tracciamento deve prevedere delle funzionalità aggiuntive, come la registrazione in tempo reale di ogni evento e dato rilevante ai fini del trattamento del trauma. Gli eventi possono riferirsi ad azioni effettuate sul paziente come la somministrazione di medicinali o altri trattamenti, ma anche, più in generale, a qualsiasi tipo di evento clinico. I dati invece, includono i parametri vitali, il risultato di un'analisi e ogni altra informazione fornita dai dispositivi medici. La gestione di queste specifiche patologie richiede una forte sinergia e coordinazione della fase pre-ospedaliera e della fase intra-ospedaliera, che vengono spesso gestite da unità differenti.

Per ottemperare a queste necessità, il Trauma Center dell'ospedale "M. Bufalini" di Cesena ha intrapreso un progetto di collaborazione con il dipartimento di Informatica - Scienza e ingegneria dell'università che ha portato all'analisi e alla progettazione di TraumaTracker[2], un sistema designato al supporto del tracciamento, della completa documentazione e dell'accurata gestione delle patologie tempo-dipendenti sin dalla fase pre-ospedaliera. Il sistema è infatti predisposto al tracciamento automatico di ogni evento rilevante che si verifica durante la gestione del trauma nonché alla registrazione e al monitoraggio dello stato e dei parametri vitali del paziente. Si può ben intuire che un sistema di tracciamento come TraumaTracker sia tanto più efficace quanto più alto è il numero di informazioni che riesce ad acquisire, sia direttamente dal trauma leader (il medico che sta eseguendo le operazioni), sia quelle raccolte automaticamente. In dettaglio, le principali funzionalità volte a supporto della fase pre-ospedaliera includono:

- il continuo monitoraggio della missione di soccorso, effettuando il tracciamento durante tutto il tragitto dall'ospedale al luogo di emergenza e viceversa.

- tracciare le informazioni necessarie del paziente come l'età, il sesso, l'anamnesi e tutti i trattamenti effettuati dal medico.

- il tracciamento dei parametri vitali del paziente dall'arrivo dei soccorritori e durante tutta la fase di trasporto.

Per quanto riguarda la fase intra-ospedaliera, TraumaTracker consente di:

- supportare il capo del Trauma Team nell'annotare i medicinali somministrati, i trattamenti effettuati, registrare la diagnostica e ogni variazione significativa delle condizioni del paziente.

- tracciare in modo continuo e autonomo gli spostamenti del paziente tra le diverse stanze del pronto soccorso (sala rossa, stanza della tomografia, ecc.) e memorizzarne il tempo di permanenza.

- memorizzare il tempo che intercorre tra la richiesta di un esame e la ricezione del referto.

- tracciare in modo continuo e autonomo i parametri vitali del paziente in specifici istanti e in base alle procedure eseguite e/o ai farmaci somministrati. Questo è possibile grazie all'utilizzo di sensori integrati nell'ambiente clinico in modo pervasivo ma non invasivo, che memorizzano ad esempio la frequenza cardiaca, la temperatura corporea, la pressione arteriosa ecc.

- produrre e memorizzare il report finale di ogni trauma e fare in modo che questo sia memorizzato nel sistema ospedaliero e facilmente reperibile per un'analisi retrospettiva.

Di conseguenza, di fondamentale importanza è anche il recupero e la gestione dei referti degli esami clinici effettuati i quali vengono memorizzati nei sistemi informativi degli ospedali.

Nel contesto della gestione del trauma, un esame di particolare importanza e che necessita quindi di una robusta ed efficiente gestione dei relativi referti, è l'emogasanalisi. L'emogasanalisi consente di ottenere un insieme di parametri fondamentali per definire lo stato del trauma, per capire quali farmaci somministrare e che trattamenti effettuare sul paziente. Tuttavia, in TraumaTracker non è ancora presente un'integrazione con i referti prodotti dagli emogasanalizzatori dell'ospedale "M. Bufalini".

1.2 TraumaTracker e l'emogasanalisi

L'emogasanalisi (EGA o blood gas analysis in inglese) è un esame clinico che, dato un campione di sangue prelevato da un paziente, permette di misurarne la pressione dei gas arteriosi, il ph e altri parametri di interesse. Una

volta effettuato l'esame, gli emogasanalizzatori producono un referto cartaceo del risultato. Il personale medico dovrà poi registrare manualmente ogni singolo referto, un'operazione onerosa che richiede una quantità tempo di lavoro aggiuntivo non indifferente. Inoltre, allo stato attuale ci sono molti problemi di interoperabilità nei sistemi informatici ospedalieri. Spesso infatti, i dati di interesse sono contenuti in singoli file che vengono modificati da personale diverso, in momenti diversi e secondo esigenze diverse. Queste operazioni ovviamente, oltre a essere particolarmente inefficienti, sono alquanto soggette a possibili errori. Vi è quindi una forte necessità di automatizzare la gestione di questi processi e di integrare l'emogasanalisi al sistema TraumaTracker in modo da garantire l'interoperabilità e la facile reperibilità dei risultati, migliorando così notevolmente il lavoro del personale medico e la cura dei pazienti.

Figura 1.1: RAPIDPoint 500, l'emogasanalizzatore di Siemens utilizzato dall'ospedale "M. Bufalini" di Cesena

1.2.1 Lo stato attuale

In aggiunta alla produzione dei referti cartacei, gli emogasanalizzatori dell'ospedale trasmettono i risultati delle analisi a un server di proprietà di Siemens installato presso la rete di AUSL Romagna. Questo server è a sua volta

programmabile per inoltrare tutti questi dati verso qualsiasi sistema che sia in grado di riceverli. Tuttavia, questi dati non sono al momento usufruibili dal sistema ospedaliero in quanto non vi è una struttura che ne consenta il recupero e la facile interpretazione. Inoltre, anche nel caso in cui ci fosse un sistema predisposto alla ricezione di questi messaggi, il server di Siemens invierà sempre tutti i dati a sua disposizione, senza dare la possibilità di ricevere esclusivamente i soli dati di interesse per TraumaTracker. I dati scambiati sono formattati secondo lo standard HL7, uno degli standard di riferimento per lo scambio di informazioni nell'ambito dei servizi sanitari.

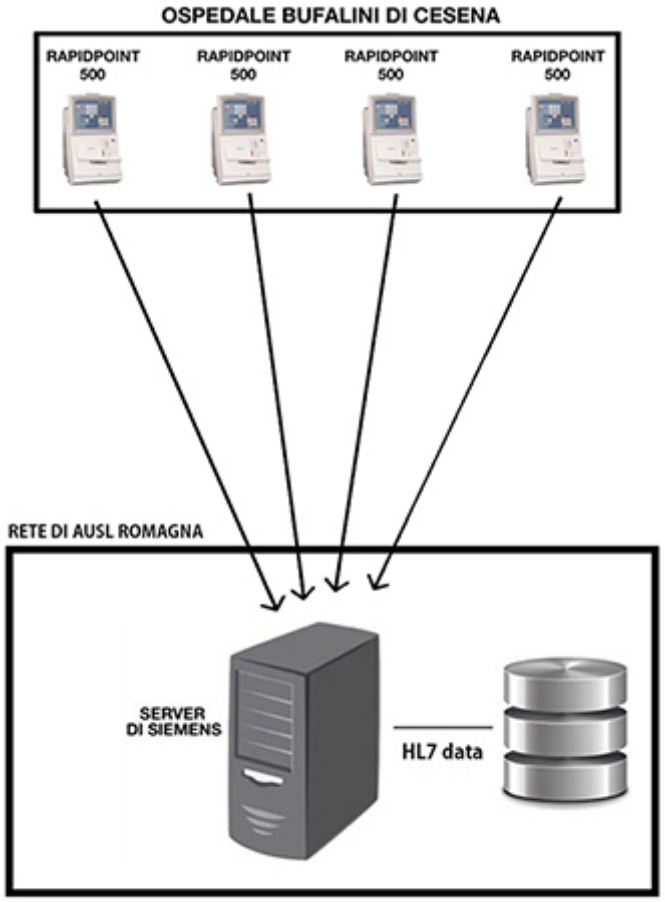


Figura 1.2: Gli emogasanalizzatori dell'ospedale inviano i risultati delle analisi al server di Siemens dove vengono salvati in formato HL7

1.2.2 HL7

Health Level Seven International (HL7) è un'associazione non profit che si occupa dello sviluppo di standard per lo scambio, l'integrazione e il recupero di informazioni sanitarie digitali e che supporta la gestione e la fornitura dei servizi sanitari. Lo standard HL7 è approvato anche da organizzazioni di standardizzazione come American National Standards Institute (ANSI) e International Organization for Standardization (ISO). La versione 2 dello standard di messaggistica HL7, probabilmente il più diffuso e implementato per i servizi sanitari nel mondo, agevola notevolmente lo scambio di informazioni e dati clinici tra sistemi digitali[1] [2].

Ogni messaggio HL7 ha uno specifico scopo e una specifica struttura. Le informazioni inviate usando questo standard sono strutturate in singoli messaggi e organizzate in segmenti. Ogni segmento termina con un carattere di ritorno a capo <CR>, ogni messaggio termina con un carattere di ritorno a capo <CR><LF>. Ogni segmento contiene diversi campi separati da limitatori (il carattere | generalmente), ogni campo occupa una specifica posizione del segmento. Lo standard HL7 è stato pensato e scritto partendo dal presupposto che un evento nel mondo reale dell'assistenza sanitaria crei la necessità di uno scambio di dati tra sistemi. Quando il trasferimento di informazioni viene avviato dal sistema applicativo che si occupa dell'evento scatenante, lo scambio viene definito unsolicited update.

Saranno ora considerati i messaggi prodotti dal modello RAPIDPoint 500 di Siemens Healthcare Diagnostics, in quanto è il modello in utilizzo all'ospedale "M. Bufalini" di Cesena. RAPIDPoint 500 utilizza la versione 2.4 dello standard HL7 e, di conseguenza, i seguenti segmenti per inviare e ricevere dati: MSH (Message Header), PID (Patient identification), PV1 (Patient Visit), ORC (Common Order), OBR (Observation Request), OBX (Observation/Result) e NTE (Notes and Comments). Eventuali dati in altri segmenti vengono ignorati[1].

La trattazione si concentra sui messaggi di tipo patient result (i messaggi che si riferiscono al risultato di un analisi effettuata su un paziente) relativi all'emogasanalisi[3]. In questa descrizione verranno usati i termini ordine e servizio. Un servizio è una specifica operazione, spesso periodica, a intervalli di tempo regolari, che deve essere effettuata su un paziente (ad esempio prelievo di sangue, trattamenti medici, trattamenti chirurgici ecc.). Un ordine è l'insieme delle istruzioni o delle direttive di un operatore sanitario in merito al trattamento di un paziente.

[1] `https://www.hl7.org/`

[2] `https://en.wikipedia.org/wiki/Health_Level_7`

[3] `https://hl7-definition.caristix.com/v2/HL7v2.4/Segments`

Campo HL7	Nome	Descrizione	Richiesto/Opzionale
MSH 3	Sending Application	Questo campo identifica univocamente l'applicazione mittente. Nel nostro caso si tratta sempre di RAPIDComm	O
MSH 7	Message date and time	Questo campo contiene la data e ora in cui il sistema mittente ha creato il messaggio. Se viene specificato un fuso orario, questo verrà utilizzato in tutto il messaggio come fuso orario predefinito	R
MSH 9.1	Message Type	Specifica a quale tipologia appartiene il messaggio. Nel nostro caso siamo sempre in presenza di messaggi di tipo **ORU** (unsolicited result message) che identificano quindi l'invio non sollecitato al sistema del risultato dell'analisi	R
MSH 9.2	Event Type	Descrive la tipologia alla quale appartiene l'evento scatenante (**trigger event**). Nel nostro caso sono tutti **R01 (ORU/ACK)** ovvero "Unsolicited transmission of an observation message"	R
MSH 9.3	Message Structure	Descrive la struttura del messaggio	O
MSH 10	Message Control ID	Questo campo contiene un numero o un altro identificatore (nel nostro caso è una stringa) che identifica in modo univoco il messaggio	R

Figura 1.3: Descrizione dei campi di interesse del segmento MSH

Il primo segmento è sempre MSH (figura 1.3), ovvero il segmento che definisce il mittente, il destinatario e lo scopo del messaggio, più qualche specifica relativa alla sintassi. Terminato MSH, segue il segmento PID. Questo segmento contiene le informazioni necessarie per identificare il paziente, come dati anagrafici e/o demografici e/o altre informazioni personali.

Campo HL7	Nome	Descrizione	Richiesto/Opzionale
PID 3	Medical record number	Questo campo contiene gli identificatori utilizzati dalla struttura sanitaria per identificare in modo univoco il paziente, ad esempio il numero di cartella clinica, il numero di fatturazione, il registro delle nascite o un identificativo individuale univoco nazionale	O
PID 5.1	Patient last name	Contiene il cognome del paziente	R
PID 5.2	Patient first name	Contiene il nome del paziente	R
PID 5.3	Patient middle name	Contiene il secondo nome del paziente	R
PID 7	Date of birth	Contiene la data di nascita del paziente	O
PID 11	Address	Contiene l'indirizzo di residenza del paziente	O
PID 13	Phone number	Contiene il numero di telefono del paziente	O
PID 20	Numero patente	Contiene il numero identificativo della patente del paziente	O

Figura 1.4: Descrizione dei campi di interesse del segmento PID

Viene ora descritto il segmento PV1. Questo segmento è utilizzato dalle applicazioni di registrazione/amministrazione del paziente per comunicare informazioni in base al resoconto della visita.

Campo HL7	Nome	Descrizione	Richiesto/Opzionale
PV1 2	Patient class	Questo campo viene utilizzato dai sistemi per classificare i pazienti in base al luogo in cui si trovano. Alcuni possibili valori sono I (ricoverato), O (ambulatorio), E (emergenza)	O
PV1 3.1	Point of care	Contiene il piano (ed eventuali altri informazioni associate) della struttura in cui si trova il paziente	O
PV1 3.2	Assigned Room	Contiene il numero della stanza in cui si trova il paziente	O
PV1 3.3	Assigned Bed	Contiene il numero del letto in cui si trova il paziente	O
PV1 4	Admission type	Questo campo indica le circostanze in cui il paziente è stato o sarà ricoverato. Dei possibili valori sono E (emergenza), A (incidente), U (urgenza)	O
PV1 7.1	Attending physician ID	Contiene l'identificativo del medico curante	O
PV1 7.2	Attending physician Name	Contiene il nome del medico curante	O
PV1 44	Admit date and time	Contiene la data e l'ora in cui il paziente è stato ricoverato	O
PV1 45	Discharge date and time	Contiene la data e l'ora a cui in cui il paziente è stato dimesso. Se non è presente, il paziente non è ancora stato dimesso	O

Figura 1.5: Descrizione dei campi di interesse del segmento PV1

Segue la descrizione del segmento ORC, un segmento facoltativo che viene utilizzato per trasmettere dati comuni a tutti i tipi di servizi richiesti (ad esempio, informazioni su un ordine verso un paziente).

Campo HL7	Nome	Descrizione	Richiesto/Opzionale
ORC 5	Order status	Questo campo specifica lo stato dell'ordine. Alcuni possibili valori sono CM (completato), CA (cancellato), ER (errore, non trovato)	O
ORC 7.1	Quantity	Questo campo indica in che quantità effettuare il trattamento descritto nel servizio	O
ORC 7.4	Start date and time	Indica la data e l'ora alla quale il servizio deve iniziare	O
ORC 7.5	End date and time	Questo campo deve contenere la data e l'ora entro quando il servizio deve essere effettuato	O
ORC 7.6	Priority	Questo campo indica il livello di urgenza del servizio. Un possibile valore è S (livello di priorità massimo)	O

Figura 1.6: Descrizione dei campi di interesse del segmento ORC

Il segmento successivo è OBR. Questo segmento è utilizzato per trasmettere informazioni specifiche di un esame, un'osservazione clinica, uno studio diagnostico. Il placer è la persona o il servizio che richiede o effettua l'ordine per un'osservazione clinica. Esempi di placer includono il medico, lo studio, la clinica o il servizio di reparto che ordina un test di laboratorio. Il produttore è la persona, o il servizio (ad esempio laboratorio clinico o servizio infermieristico), che effettua le osservazioni cliniche (esegue l'ordine) richieste.

Campo HL7	Nome	Descrizione	Richiesto/Opzionale
OBR 1	OBR sequence number	Questo campo contiene il numero identificativo dell'ordine, deve essere incrementato di 1 ad ogni ordine trasmesso	O
OBR 4.1	Universal service identifier	Questo campo contiene il codice identificativo per il test, l'analisi clinica richiesta	R
OBR 25	Result Status	Questo campo contiene lo stato dei risultati dell'ordine. Un valore che può assumere è F (risultato finale)	O
OBR 27.1	Quantity	Questo campo indica la quantità del trattamento che si deve effettuare ad ogni intervallo di tempo in cui il servizio viene effettuato	O
OBR 27.4	Start date and time	Questo campo indica la data e l'ora alla quale il servizio deve iniziare	O
OBR 27.5	End date and time	Questo campo deve contenere la data e l'ora entro quando il servizio deve essere effettuato	O
OBR 27.6	Priority	Questo campo indica il livello di urgenza della richiesta. Un possibile valore è S (livello di priorità massimo) oppure R (routine, è il valore di default)	O
OBR 36	Scheduled date and time	Questo campo contiene la data e l'ora prevista per l'osservazione clinica	O

Figura 1.7: Descrizione dei campi di interesse del segmento OBR

Viene infine descritto il segmento OBX. Questo segmento viene utilizzato per trasmettere il risultato di un'analisi clinica e altre informazioni di resoconto correlate.

Campo HL7	Nome	Descrizione	Richiesto/Opzionale
OBX 1	Set ID	Questo campo contiene il numero di sequenza dei segmenti OBX in questo messaggio. Il segmento OBX 1 include campi sia per il primo risultato del test che per gli identificatori del campione e dell'operatore. I segmenti OBX successivi includono solo campi demografici o risultati di test	R
OBX 2	Value type	Questo campo contiene il formato dell'analisi clinica. Ad esempio, si può usare ST per indicare il formato stringa	O
OBX 3	Observation identifier	Questo campo contiene l'identificatore univoco dell'analisi clinica. Nel nostro caso, il nome del test (ad esempio pH)	O
OBX 5	Observation value	Questo campo contiene il valore rilevato dal test	O
OBX 6	Units	Questo campo contiene l'eventuale unità di misura	O
OBX 7	Reference range	Questo campo contiene il limite inferiore e/o superiore in cui dovrebbe trovarsi il valore rilevato dal test per essere nella norma	O
OBX 8	Abnormal flags	Questo campo indica se il risultato del test è nella norma o no e in che misura. Dei possibili valori sono A (anormale), AA (molto anormale), N (normale), L (sotto la norma), H (sopra la norma)	O
OBX 14	Observation date and time	Questo campo contiene la data e l'ora dell'analisi clinica e può essere utile quando quelle specificate nel segmento OBR hanno date e/o orari differenti	O
OBX 18.3	Equipment ID	Questo campo contiene il numero seriale del dispositivo che ha analizzato il campione	O
OBX 18.4	Model ID	Questo campo contiene il nome del modello del dispositivo che ha analizzato il campione	O
OBX 19	Date and time of the analysis	Questo campo contiene la data e l'ora in cui è stata effettuata l'analisi	O

Figura 1.8: Descrizione dei campi di interesse del segmento OBX

1.2.3 Obiettivo della tesi

Allo stato attuale quindi, gli emogasanalizzatori dell'ospedale "M. Bufalini", dopo aver effettuato un'analisi, producono un referto cartaceo e inviano il risultato al server di Siemens installato nella rete di AUSL Romagna dove vengono memorizzati in formato HL7. Il server di Siemens può solamente inoltrare tutti i referti a sua disposizione e senza offrire la possibilità di ricevere esclusivamente i dati di interesse del personale medico, cosa che ovviamente ne complicherebbe notevolmente la gestione, la registrazione e la consultazione, motivo per cui, queste operazioni vengono ancora svolte manualmente.

Nell'ambito della collaborazione che ha portato allo sviluppo di TraumaTracker è stata sviluppata una prima versione prototipale di un sistema predisposto alla ricezione dei messaggi dal server di Siemens e che consentisse il recupero di alcuni dati presenti nei referti relativi all'emogasanalisi. Questo prototipo, la cui trattazione sarà comunque approfondita nel capitolo successivo, soffre però di numerose mancanze ed è molto esiguo nelle funzionalità che

offre, il che lo rende completamente inadatto a soddisfare le necessità appena descritte. L'obiettivo di questa tesi è quindi provvedere a una sostanziale, fondamentale integrazione ed estensione del prototipo, dotandolo di tutte le funzionalità necessarie a ottemperare le esigenze emerse nella trattazione. L'obiettivo prefissato è però ancor più ampio poiché si desidera garantire sia la possibilità di integrazione sia l'astrazione da TraumaTracker in quanto si vuole che i dati relativi ai referti siano disponibili, consultabili e gestibili in modo facile e indipendente per tutto il sistema ospedaliero. In questo modo il personale medico non sarà più costretto a registrare manualmente tutti i risultati delle analisi ma avrà invece la possibilità di recuperare, consultare, gestire agevolmente e velocemente ogni singolo referto, trovando quindi soluzione anche al problema dell'interoperabilità precedentemente descritto.

Capitolo 2

Analisi del sistema

In questo capitolo viene presentato il prototipo di partenza da cui si è costruito l'intero progetto analizzandone le funzionalità già esistenti e, al contempo, sottolineandone l'inadeguatezza e le numerose mancanze. Si prosegue quindi con un'analisi tecnica di ciò che occorre introdurre per adempire alle necessità trattate nel capitolo precedente. Segue una descrizione delle tecnologie che verranno utilizzate in fase di implementazione e le nuove funzionalità con cui integrare il sistema. Dopo un importante cenno alla questione sicurezza e privacy dei dati, il capitolo si conclude con un'analisi del modello del dominio, l'introduzione di FHIR, la specifica dei requisiti e il diagramma dei casi d'uso.

2.1 Il prototipo esistente

Come introdotto a conclusione del capitolo precedente, nell'ambito della collaborazione che ha portato allo sviluppo di TraumaTracker, è stata sviluppata una prima versione prototipale di un sistema volto alla ricezione dei messaggi HL7 provenienti dal server di Siemens e che fosse in grado di mettere a disposizione i dati presenti nei referti relativi all'emogasanalisi. Vengono ora trattati i componenti principali che costituiscono il prototipo e tutte le funzionalità che sono state implementate. Questo darà l'occasione non solo di descrivere quanto già esistente ma anche di effettuare considerazioni sull'adeguatezza e sulle mancanze del prototipo in funzione degli obiettivi prefissati nel capitolo precedente.

In virtù delle considerazioni effettuate, il primo componente di cui è stato dotato il prototipo è un server predisposto al continuo ascolto dei messaggi HL7 in arrivo dal server di Siemens. Tuttavia, lo standard HL7, benché sia un punto di riferimento nell'ambito dello scambio di dati in ambito sanitario, è un formato non semplice da interpretare e, come si può evincere dalla trattazione

nel capitolo precedente, è spesso sovrabbondante nella quantità di informazioni che fornisce.

```
MSH|^~\&|Rapidcomm|Hospital|HIS|Hospital|201506051656||ORU^R30|
0C08YQM3014000010404|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|RE
OBR|1|||ABG^BG-BLOOD GAS ANALYSIS^^^^||||||S^^^^^^^
|O|||||OPHYID^OPHYNAME^^^^^||||||||||001^^^201506061014^201506061014^
S^^^^|||||||||201506061014|
OBX|1|ST|pH||7.350||7.350-7.450||||F|||20150601112058||ROID^ROLNAME^
ROFNAME||^^40001^RAPIDPoint400|20150601112058|
OBX|2|ST|pCO2||33.0|mmHg|35.0-45.0|L|||F|
OBX|3|ST|pO2||88.0|mmHg|75.0-100.0||||F|
OBX|4|ST|Na+||152|mmol/L|135.0-148.0|H|||F|
OBX|5|ST|K+||6.8|mmol/L|3.50-5.30|H|||F|
OBX|6|ST|Ca++||.8|mmol/L|1.13-1.32|LL|||F|
OBX|7|ST|Cl-||48|mmol/L|98-106|L|||F|
OBX|8|ST|Smp. Type||BLDA||||||F|
OBX|9|ST|Site||User list||||||F|
```

Figura 2.1: Esempio di un messaggio HL7

Data questa complessità, unita al fatto che il server di Siemens invia sempre tutti i referti a sua disposizione senza offrire la possibilità di ricevere esclusivamente i dati di interesse della specifica applicazione, il prototipo è stato fornito di apposite librerie in grado non solo di ricevere messaggi in formato HL7 ma anche di leggere, interpretare, elaborare e gestire questo standard, convertire i dati in un formato facilmente leggibile e agevolmente adoperabile conservando esclusivamente le informazioni necessarie al trattamento del paziente e di memorizzare stabilmente in locale il risultato dell'elaborazione. Al netto di queste esigenze aggiuntive, è stata dunque una scelta ragionevole introdurre un'entità intermedia predisposta all'adempimento di questi doveri senza delegare tutto questo carico di lavoro aggiuntivo a TraumaTracker e/o a qualsiasi altra applicazione interessata a ricevere i referti.

Considerando l'esigenza di memorizzare stabilmente i referti, dell'avere la possibilità di gestirli, aggiornarli, cancellarli ed eventualmente modificarli, è stato introdotto un database non relazionale a supporto del server. È ora possibile analizzare le peculiarità dei dati che si vuole memorizzare ed esaminare le operazioni che saranno maggiormente richieste alla base di dati per capire che tipo di database e quale Database Management System sia più opportuno

utilizzare e, di conseguenza, stabilire se la scelta effettuata per il prototipo risulti appropriata. Data la natura di utilizzo del sistema, si dovrà di certo prediligere una base di dati che sia molto efficiente nelle operazioni in lettura, questo perché saranno ovviamente molto più frequenti le richieste di visualizzazione dei referti rispetto a quelle di modifica. Inoltre, il numero dei referti da memorizzare potrebbe anche diventare rapidamente molto grande e non si vuole che questo comporti un problema nel garantirne l'accesso simultaneo da parte di più utenti contemporaneamente. Per quanto concerne il modo in cui i messaggi HL7 dovranno essere memorizzati, la scelta più efficiente e utile per le applicazioni che richiederanno i dati è salvare direttamente le informazioni ricavate dai campi HL7 appena descritti, si predilige dunque un tipo di rappresentazione dell'informazione che consenta elevate prestazioni, una buona scalabilità e che sia semplice e intuitiva. Alla luce di tutte queste considerazioni, è opportuno scegliere un DBMS non relazionale e orientato ai documenti. Il prototipo fa uso di MongoDB, un DBMS che rispecchia esattamente queste caratteristiche, la scelta effettuata risulta quindi adeguata per le esigenze del sistema.

Infine, il prototipo mette in campo un servizio web disposto all'esposizione di API RESTful[1] per consentire l'invio di tutti i referti o di un unico referto dato il suo ID univoco. La figura 2.2 mostra quindi le entità coinvolte nella modellazione del prototipo. In particolare, EgaService è la classe designata all'esposizione delle API RESTful, HL7Receiver dispiega il server in grado di ricevere i messaggi HL7 provenienti dal server di Siemens i quali verranno poi interpretati e memorizzati dalla classe OruEndPoint che si avvale di metodi progettati specificamente per la gestione dei messaggi in formato HL7 e per la memorizzazione nel database.

Tuttavia, è importante notare come il prototipo soffra di numerose mancanze e sia molto esiguo nelle funzionalità che offre. In particolare, è molto limitato nell'interpretazione dei messaggi HL7 in quanto ne ricava una quantità di dati insufficienti per quelle che sono le esigenze descritte. Per quanto riguarda la presentazione dei dati, il sistema espone delle RESTful API che, oltre a fornire un numero molto ristretto di informazioni, sono prive di qualsiasi forma di controllo nei confronti di chi stia cercando di accedervi, il che rende i referti sanitari potenzialmente esposti anche a utenti non autorizzati. In aggiunta, non è presente nessun applicativo che consenta agli utenti autorizzati di accedere, visualizzare e gestire facilmente, comodamente, velocemente le informazioni messe a disposizione, rendendo di fatto il prototipo completamente inutilizzabile soprattutto dal personale medico.

[1] `https://en.wikipedia.org/wiki/Representational_state_transfer`

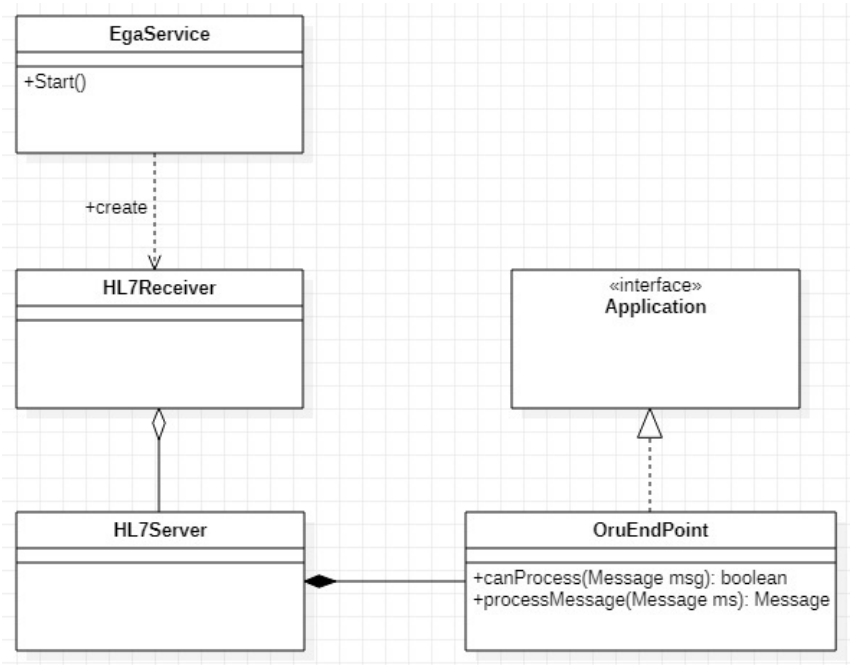


Figura 2.2: Diagramma delle classi del prototipo

2.2 Requisiti tecnici del sistema

Per realizzare un sistema che sia in grado di adempiere alle necessità indicate nell'esposizione degli obiettivi prefissati, sopperire alle carenze del prototipo e completarne lo sviluppo, occorre sia introdurre nuove funzionalità sia migliorare ed estendere quelle già esistenti. In particolare, è necessario:

- ampliare l'interpretazione dei messaggi HL7 estrapolando tutte le informazioni necessarie alla gestione del trauma e quelle comunque utili nel contesto ospedaliero di riferimento. Sarà poi ovviamente fondamentale salvare anche le nuove informazioni nel database in un formato facilmente leggibile come descritto in precedenza.

- migliorare le API RESTful esposte dal servizio web aggiungendone di nuove che consentano il recupero dei referti in base a criteri supplementari e che permettano una più articolata gestione dei referti, aggiungendo ad esempio la possibilità di eliminare quelli meno recenti.

- introdurre un'applicazione web che consenta a tutti gli utenti autorizzati di sfruttare al meglio le API RESTful presentate al punto precedente. Gli utenti saranno quindi in grado di visualizzare facilmente, velocemente e agevolmente l'elenco di tutti i referti, cancellare quelli più obsoleti, visua-

lizzare solo quelli riguardanti un determinato paziente e poter esaminare nel dettaglio i risultati dell'analisi contenuti in ogni singolo referto.

Per quanto concerne la verbosità dello standard HL7, è necessario individuare e memorizzare esclusivamente i dati utili e necessari in ambito ospedaliero. In particolare, i seguenti campi del messaggio HL7 forniscono delle informazioni fondamentali per il contesto di riferimento:

- MSH 10: contiene l'identificatore univoco del messaggio ed è quindi essenziale per la corretta memorizzazione e gestione dei referti.
- MSH 7: indica la data e l'ora in cui il messaggio è stato creato.
- PID 3: contiene l'identificatore univoco assegnato dal sistema ospedaliero per identificare il paziente.
- PID 5: racchiude il nome e cognome del paziente.
- PID 7: contiene la data di nascita del paziente.
- OBX 3: specifica il tipo di analisi effettuato.
- OBX 5: contiene il valore rilevato dall'analisi.
- OBX 6: specifica l'eventuale unità di misura del risultato dell'analisi.
- OBX 18: contiene il numero seriale del dispositivo che ha effettuato l'analisi.
- OBX 19: indica la data e l'ora in cui è stata effettuata l'analisi.

L'immagine 2.3 mostra il quadro completo dell'infrastruttura del sistema.

2.3 Descrizione delle tecnologie

Riassumendo, da un punto di vista puramente architetturale il sistema sarà suddiviso in due componenti principali: il server e un'applicazione web che comunicheranno attraverso la tipica interazione client-server. Il server (denominato per comodità EGA Server), oltre a poter ricevere i messaggi in formato HL7, esporrà un servizio web che consentirà la visualizzazione e la gestione dei referti. L'applicazione web dovrà prevedere tutte le funzionalità necessarie alla corretta comunicazione con il server e alla facile visualizzazione e gestione dei referti da parte del personale medico.

L'applicazione web sarà implementata utilizzando HTML[2] come linguaggio di markup per la creazione delle pagine web, JavaScript[3] come linguaggio di programmazione per implementare le funzionalità necessarie e Bootstrap[4], uno dei framework più famosi e più utilizzati, per curare lo stile grafico e garantire un design responsivo dell'applicazione.

Per quanto riguarda il server, questo sarà implementato in Java[5] e si avvarrà delle seguenti tecnologie, già utilizzate nell'implementazione del prototipo:

- Vert.x: Vert.x[6] è un toolkit basato su un'architettura a eventi e programmazione asincrona ideato per semplificare notevolmente lo sviluppo di innumerevoli applicazioni. Vert.x è agilmente integrabile in qualsiasi progetto, la sua struttura modulare consente di utilizzare esclusivamente i componenti di cui si ha bisogno. Scambio di messaggi, API web, database, Vert.x include tutto il necessario per agevolare e velocizzare in modo significativo lo sviluppo di un sistema, fornendo al contempo garanzie riguardo la correttezza implementativa, le performance e la scalabilità in base alle proprie esigenze. In particolare, Vert.x-Web consente di creare facilmente un server web, fornisce supporto per lo scambio di messaggi HTTP, per la gestione della sessione, dei cookie ed è perfetto per esporre un servizio HTTP RESTFul. Questo componente, coadiuvato da Vert.x MongoDB Client e Vert.x Json Schema, permetteranno la creazione del servizio web, l'esposizione delle API RESTful, la comunicazione con MongoDB per la gestione dei referti e l'interazione con l'applicazione web.

- HAPI: HL7 application programming interface (HAPI)[7] è un insieme di procedure open source e orientate agli oggetti per Java che consentono di interpretare e gestire agevolmente i messaggi della versione 2.x di HL7. Sono delle API molto pratiche e funzionali, permettono la facile creazione, interpretazione e validazione dei messaggi HL7, di ricavarne semplicemente e in modo efficace le informazioni contenute in ogni campo di ogni segmento e la creazione di un server in grado di ricevere, ed eventualmente inviare, messaggi HL7, esattamente quello di cui il sistema ha bisogno per ricevere e utilizzare i dati provenienti dal server di Siemens.

[2] https://www.w3schools.com/html/html_intro.asp
[3] https://en.wikipedia.org/wiki/JavaScript
[4] https://getbootstrap.com/
[5] https://www.java.com/it/
[6] https://vertx.io/
[7] https://hapifhir.github.io/hapi-hl7v2/

- MongoDB: MongoDB[8] è un Database Management System open source, classificato come sistema NoSQL e orientato ai documenti. Questo significa che, a differenza di molte altre basi di dati, MongoDB non segue il modello relazionale con la tipica struttura basata su tabelle ma utilizza una rappresentazione dell'informazione orientata ai documenti in formato JSON con schema dinamico, ovvero non vincolato ad una specifica struttura e che garantisce elevata flessibilità e semplicità d'uso. Questa specifica architettura rende MongoDB un DBMS ottimizzato per gli accessi in lettura, particolarmente adatto per memorizzare grandi quantità di dati in modo efficiente, consente un'elevata scalabilità e garantisce un'ottima efficienza anche in presenza di un'ingente mole di richieste dei dati, esattamente come richiesto dalle specifiche del sistema. MongoDB è utilizzato da molti dei servizi più importanti e famosi del mondo.

- JSON: JavaScript Object Notation (JSON)[9] è un formato di interscambio dati leggero, testuale, particolarmente adatto allo scambio e alla memorizzazione di informazioni tra sistemi informatici. JSON è basato sulla sintassi di JavaScript ma, nonostante il nome, è completamente indipendente dal linguaggio di programmazione utilizzato. È notevolmente facile da comprendere e pratico nell'utilizzo, la sua semplicità ne ha decretato un vasto impiego e una rapida diffusione. Nel formato JSON i dati sono strutturati tramite coppie attributo-valore in cui al nome di ogni attributo viene associato il corrispettivo valore assunto. Questa sintassi lo rende particolarmente adatto alla rappresentazione e alla successiva interpretazione dei referti clinici richiesta dal sistema. In conclusione, una valida alternativa a JSON sarebbe stata l'altrettanto diffuso XML[10], tuttavia, la scelta è ricaduta su JSON in quanto è un formato più semplice da leggere, da scrivere, da interpretare, è nativamente supportato dalla programmazione JavaScript ed è più leggero, il che velocizza l'invio e la ricezione dei dati.

2.4 Privacy e sicurezza dei dati

Naturalmente, dato che il sistema memorizza ed espone risultati di analisi cliniche, è molto importante trattare l'aspetto della sicurezza dei dati e del rispetto della privacy in ottemperanza al regolamento europeo sulla protezione dei dati personali (General Data Protection Regulation) che pone severe restrizioni all'invio in rete di dati sensibili. Per quanto concerne la sicurezza, questa

[8] `https://www.mongodb.com/it`
[9] `https://www.json.org/json-en.html`
[10] `https://en.wikipedia.org/wiki/XML`

è in parte affidata alla rete del sistema ospedaliero, la quale garantisce che lo scambio di dati con il server di Siemens avvenga in modo sicuro. Si ha inoltre la certezza che i dati contenuti nei referti non vengano immessi su rete esterna, perché sono propagati esclusivamente in una sottorete privata, gestita tramite VPN, di AUSL Romagna. Inoltre, anche TraumaTracker è notevolmente organizzato per affrontare la gestione della sicurezza. Considerando unicamente il sistema che si vuole implementare, sarebbe invece necessaria una transizione al protocollo HTTPs, il quale fornisce tutti i meccanismi necessari per implementare una comunicazione sicura e garantisce una protezione adeguata alle esigenze del sistema.

Per quanto riguarda il rispetto della privacy, la legislazione europea distingue differenti tipologie di informazioni e ne prescrive diverse metodologie di conservazione e di gestione. In particolare, sono definiti dati "sensibili" tutte le informazioni riguardanti aspetti più riservati di un individuo come, ad esempio, i dati sanitari. Sono invece definiti dati "personali" informazioni più generiche come i dati anagrafici, il numero di telefono, gli indirizzi e l'email. Non tutte queste informazioni sono attualmente di interesse per il sistema, tuttavia, lo standard HL7 prevede comunque la possibilità di memorizzarle. La legislazione europea pone dei limiti molto stretti e prevede leggi molto severe che vincolano la possibilità di ottenere e utilizzare queste informazioni. La sola implementazione di tutte le tecniche di sicurezza non consente l'utilizzo né di dati sensibili né di dati personali. Il Regolamento Europeo 679/2016 sulla protezione dei dati personali definisce responsabilità e procedure operative da attuare per l'utilizzo e la memorizzazione di questo tipo di informazioni. In particolare, sarebbe necessario, prima di effettuare il dispiegamento effettivo del sistema, implementare una procedura di anonimizzazione e cifratura dei dati che permetta quindi che questi non siano direttamente riconducibili ai pazienti da parte dei soggetti non autorizzati mantenendo al contempo intatte le funzionalità e la semplicità d'uso del sistema

2.5 Analisi dei requisiti

Prima di procedere con l'implementazione, data la semplicità, l'inadeguatezza e le molteplici mancanze del prototipo già analizzate, è necessario affrontare lo sviluppo del software con un approccio ingegneristico, iniziando quindi con l'analisi del dominio e la produzione di una specifica dei requisiti. Questo consentirà non solo di avere una descrizione esauriente, coerente e non ambigua di quello che occorre realizzare ma anche di estendere e completare lo sviluppo del prototipo rendendolo conforme alle specifiche risultanti dalla fase di analisi e predisposto a ottemperare gli obiettivi che sono stati prefissati.

2.5.1 Modello del dominio

Viene anzitutto presentato il modello del dominio volto alla presentazione delle entità più significative coinvolte nel contesto ospedaliero di riferimento e alla descrizione di come queste siano relazionate tra loro.

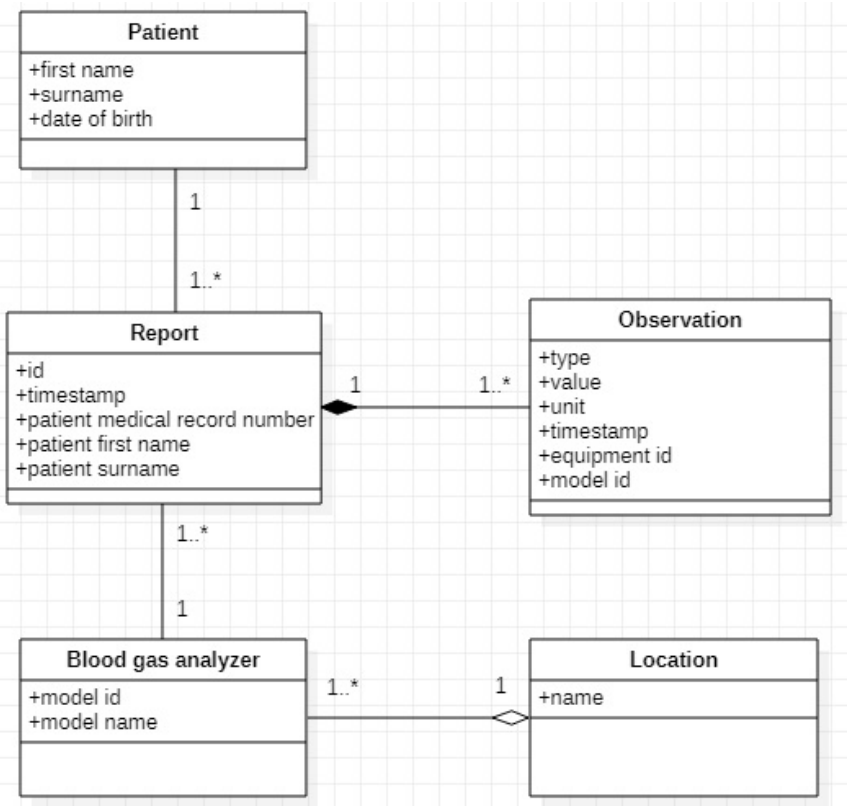


Figura 2.4: Modello del dominio di riferimento

Segue ora un glossario delle entità presentate in figura 2.4:

- Location: è il luogo, il reparto dell'ospedale in cui è stata eseguita l'emogasanalisi (ad esempio sala rossa, terapia intensiva ecc.). Come si evince dalla figura 2.4, è di interesse modellare solo i reparti in cui è presente almeno un emogasanalizzatore.

- Blood gas analyzer: rappresenta l'emogasanalizzatore vero e proprio dotato di nome del modello e numero seriale. Gli emogasanalizzatoi in uso all'ospedale "M. Bufalini" di Cesena sono tutti del modello RAPIDPoint 500.

- Patient: rappresenta il paziente sottoposto all'emogasanalisi.

- Report: è il referto clinico prodotto dall'emogasanalizzatore dopo aver eseguito l'emogasanalisi sul paziente.

- Observation: è il risultato dell'esame clinico a cui è stato sottoposto il paziente. Al suo interno viene specificato il tipo di analisi effettuata, il risultato e la relativa unità di misura. In un referto sono presenti i risultati di tutte le analisi previste dall'emogasanalisi.

2.5.2 FHIR

Fast Healthcare Interoperability Resources[11] è uno standard, pubblicato da HL7, ideato per semplificare notevolmente lo scambio di informazioni in ambito sanitario e garantire una forte interoperabilità tra sistemi e organizzazioni differenti. FHIR si basa sul concetto di "Risorsa". Le risorse sono il fulcro dello scambio di dati e possono essere integrate facilmente in qualsiasi sistema sanitario che abbia la necessità di affrontare diverse realtà cliniche e relativi problemi amministrativi. Ogni risorsa rappresenta un'entità appartenente al sistema sanitario, per esempio, gli elementi di base dell'assistenza sanitaria come pazienti, ricoveri, referti, farmaci ecc. possono essere facilmente modellati come risorse FHIR. Tutte le risorse hanno una rappresentazione univoca: sono identificate da un URL, contengono tutte le informazioni significative dell'entità che rappresentano e sono appositamente strutturate per adeguarsi a variazioni o integrazioni future. FHIR utilizza l'approccio REST per concretizzare lo scambio di informazioni, ogni risorsa può essere agevolmente recuperata tramite il suo URL e può essere rappresentata in formato JSON. Questa architettura rende FHIR utilizzabile a livello globale e in una vasta varietà di scenari clinici, la sua efficienza e l'elevata interoperabilità ne consentono il perfetto funzionamento e la facile integrazione anche tra applicazioni e dispositivi differenti. Alla luce di queste considerazioni, si vuole prevedere l'introduzione nel sistema di un server FHIR da affiancare a EGA Server. In questo modo potranno essere implementate tutte le funzionalità sinora discusse mantenendo al contempo un sistema modulare, dotato di elevata manutenibilità e facilmente integrabile.

Le risorse FHIR da introdurre nel sistema sono state efficacemente individuate analizzando il modello del dominio del contesto di riferimento.

[11] `http://hl7.org/fhir/`

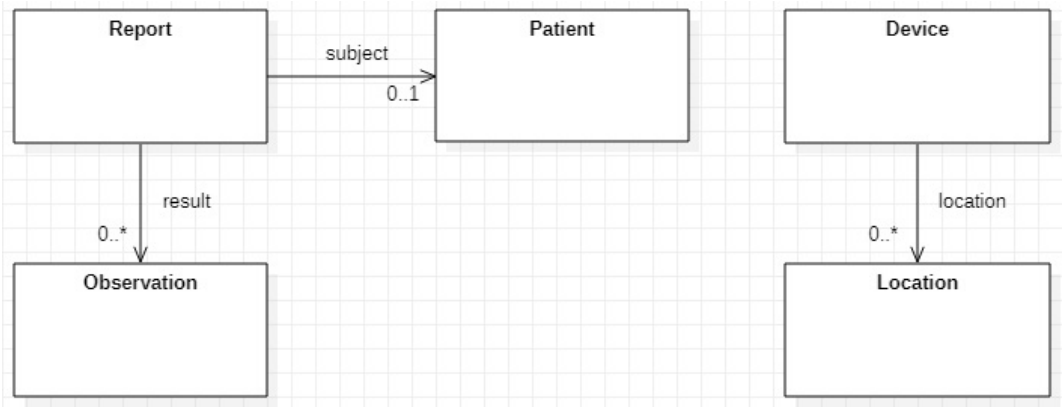


Figura 2.5: Modello delle risorse FHIR individuate nel contesto di riferimento

Come si evince dalla figura 2.5, sono due le risorse che fungono da "punto d'ingresso" per il collegamento con le altre risorse FHIR individuate: il referto e il dispositivo. I referti possono essere ricondotti a un paziente e ai relativi risultati delle analisi effettuate. Il dispositivo, ovvero l'emogasanalizzatore, può essere collegato al luogo in cui esso si trova.

2.5.3 Specifica dei requisiti e diagramma dei casi d'uso

Dopo un'attenta analisi del contesto ospedaliero di riferimento unita alla conoscenza delle più comuni e diffuse convenzioni in ambito informatico concernenti l'accesso, la consultazione e la gestione dei dati, è stato definito il seguente elenco di requisiti:

- solo gli utenti registrati nel database e correttamente autenticati potranno usufruire dei servizi del sistema.
- gli utenti registrati si divideranno in utenti ordinari e utenti amministratori.
- gli utenti potranno visualizzare l'elenco di tutti i referti disponibili.
- gli utenti potranno visualizzare uno specifico referto dato il suo codice identificativo.
- gli utenti potranno visualizzare i referti relativi a uno specifico paziente dato il suo nome.
- solo gli utenti amministratori avranno facoltà di cancellare i referti più datati.

- i referti potranno ovviamente essere recuperati, con le stesse possibilità degli utenti, anche da tutte le applicazioni di ambito ospedaliero che dovessero averne bisogno.

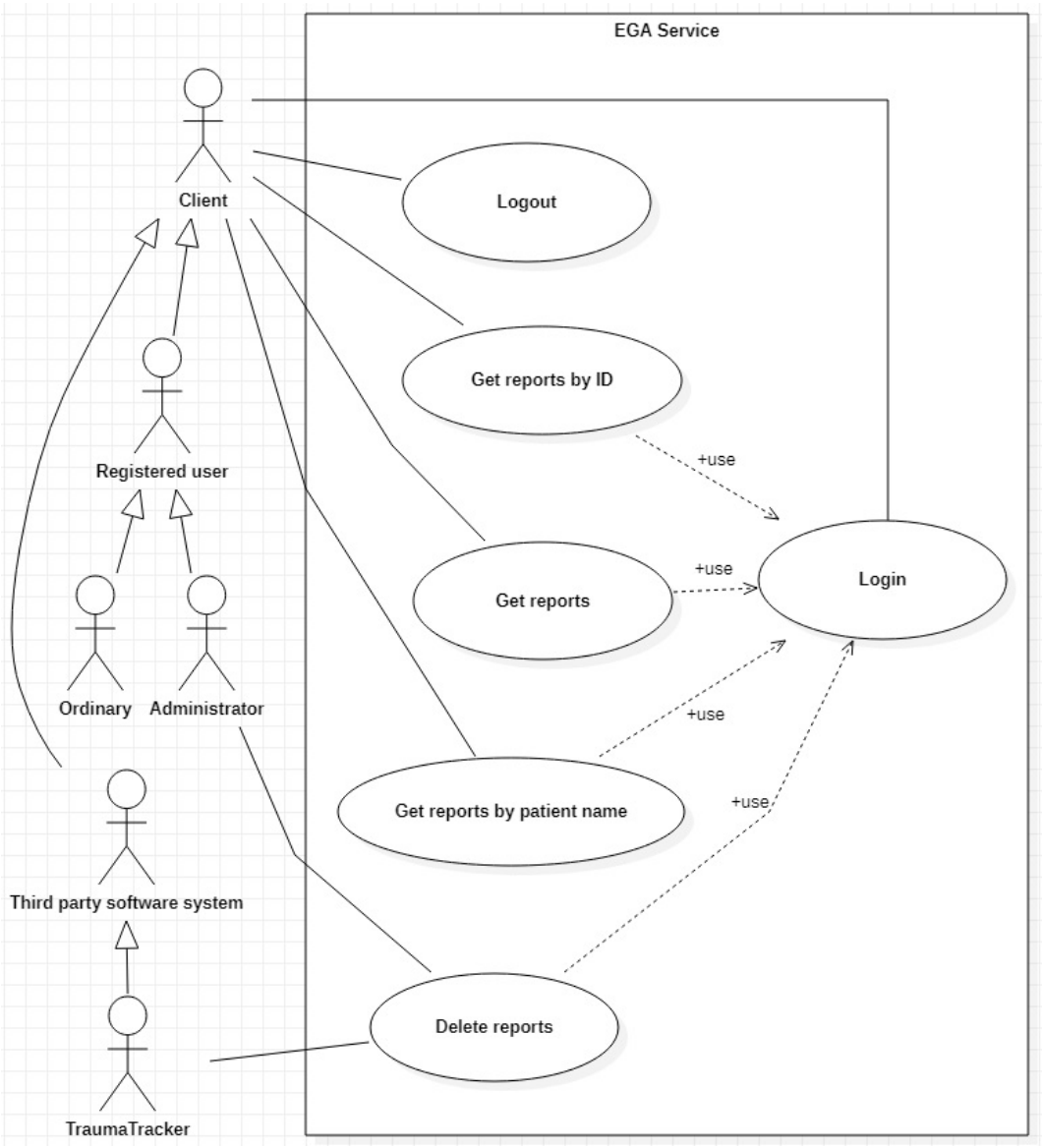


Figura 2.6: Diagramma dei casi d'uso conseguente alla specifica dei requisiti

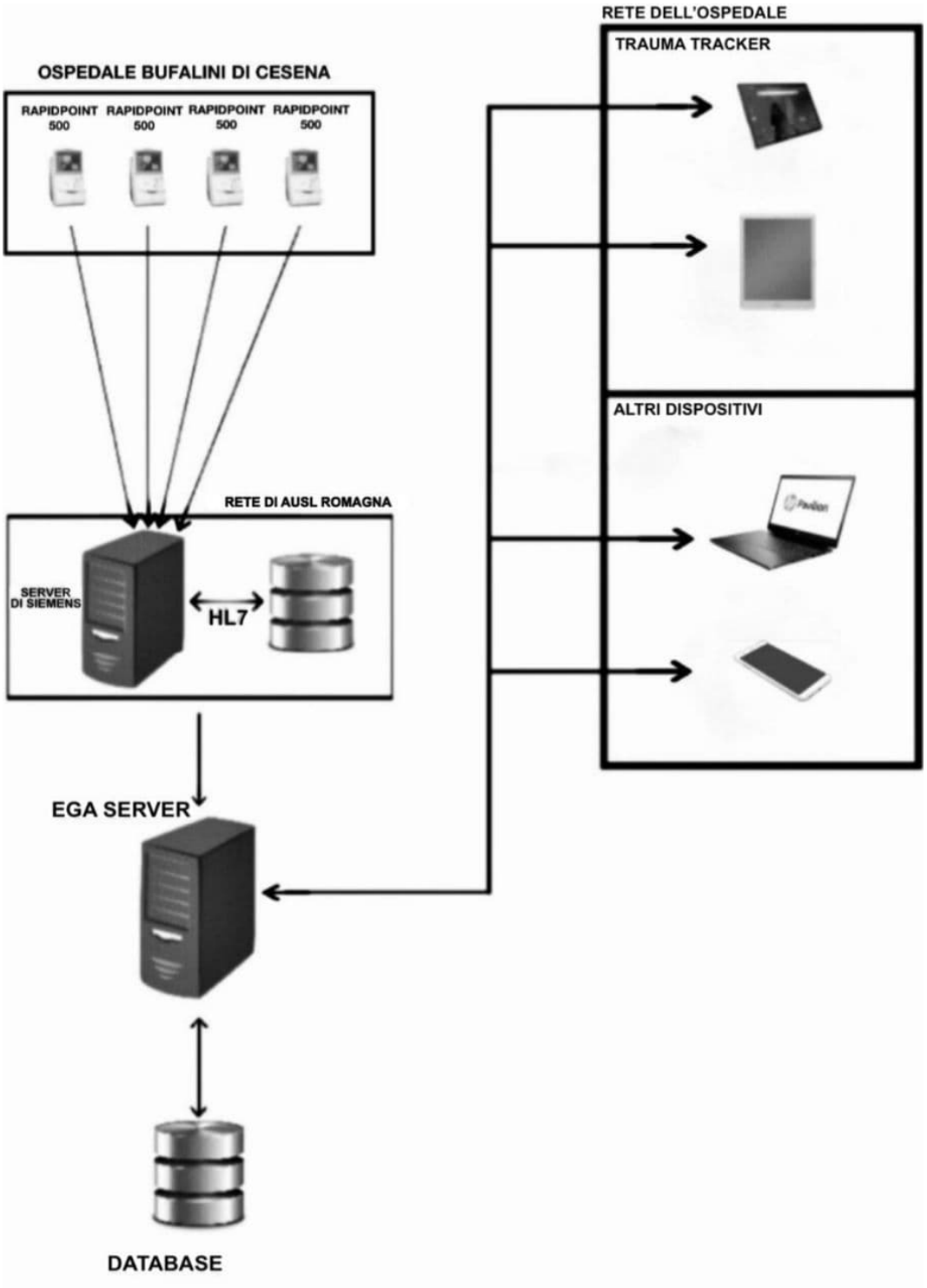


Figura 2.3: Quadro completo dell'infrastruttura del sistema

Capitolo 3

Progettazione e sviluppo

In questo capitolo si affronta lo sviluppo del progetto, partendo dalla descrizione delle API Restful, procedendo con un'accurata progettazione e analizzando lo sviluppo del server e dell'applicazione web.

3.1 Le API RESTful

Il sistema, in linea con quanto già presente nel prototipo di partenza, è stato progettato seguendo la filosofia REST, uno stile architetturale basato su HTTP in cui i dati di interesse sono presentati sotto forma di risorse identificate accuratamente da uno specifico URL. In questa architettura il server espone quindi delle API RESTFul per concedere l'accesso alle risorse e ogni applicazione client autorizzata potrà richiederle, aggiornarle, modificarle e cancellarle facendo semplicemente ricorso ai tradizionali metodi HTTP. Aderire all'approccio REST conferisce, come discusso in precedenza, numerosi benefici al sistema garantendo un'ottima interoperabilità, velocità ed efficienza nello scambio dei dati.

La prima entità che è stata modellata come risorsa sono ovviamente i referti. Tramite apposite e ben strutturate chiamate HTTP GET è possibile ottenere la lista di tutti i referti oppure ottenere un unico referto specificando il suo codice identificativo.

GET /egaservice/reports

(getReports)

Gets every EGA report

Return type
array[Report]

Example data
Content-Type: application/json


```
[ {
  "messageDateAndTime" : "messageDateAndTime",
  "medicalRecordNumber" : "medicalRecordNumber",
  "patientName" : "patientName",
  "observations" : [ {
    "unit" : "unit",
    "modelID" : "modelID",
    "type" : "type",
    "value" : "value",
    "equipmentID" : "equipmentID",
    "timestamp" : "timestamp"
  }, {
    "unit" : "unit",
    "modelID" : "modelID",
    "type" : "type",
    "value" : "value",
    "equipmentID" : "equipmentID",
    "timestamp" : "timestamp"
  } ],
  "patientDateOfBirth" : "patientDateOfBirth",
  "_id" : 1604498636162,
  "_controlId" : "_controlId"
} ]
```


Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses
200
Successful operation
401
Unauthorized
500
Internal server error

Figura 3.1: Descrizione dell'API per ottenere l'elenco di tutti i referti

GET /egaservice/reports{controlId}

(getReportById)

Returns a single report given its ID

Path parameters

controlId (required)
Path Parameter — ID of report to return

Return type
inline_response_200

Example data
Content-Type: application/json


```
{
  "messageDateAndTime" : "messageDateAndTime",
  "medicalRecordNumber" : "medicalRecordNumber",
  "patientName" : "patientName",
  "observations" : [ {
    "unit" : "unit",
    "modelID" : "modelID",
    "type" : "type",
    "value" : "value",
    "equipmentID" : "equipmentID",
    "timestamp" : "timestamp"
  }, {
    "unit" : "unit",
    "modelID" : "modelID",
    "type" : "type",
    "value" : "value",
    "equipmentID" : "equipmentID",
    "timestamp" : "timestamp"
  } ],
  "patientDateOfBirth" : "patientDateOfBirth",
  "_id" : 1604498636162,
  "_controlId" : "_controlId"
}
```


Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

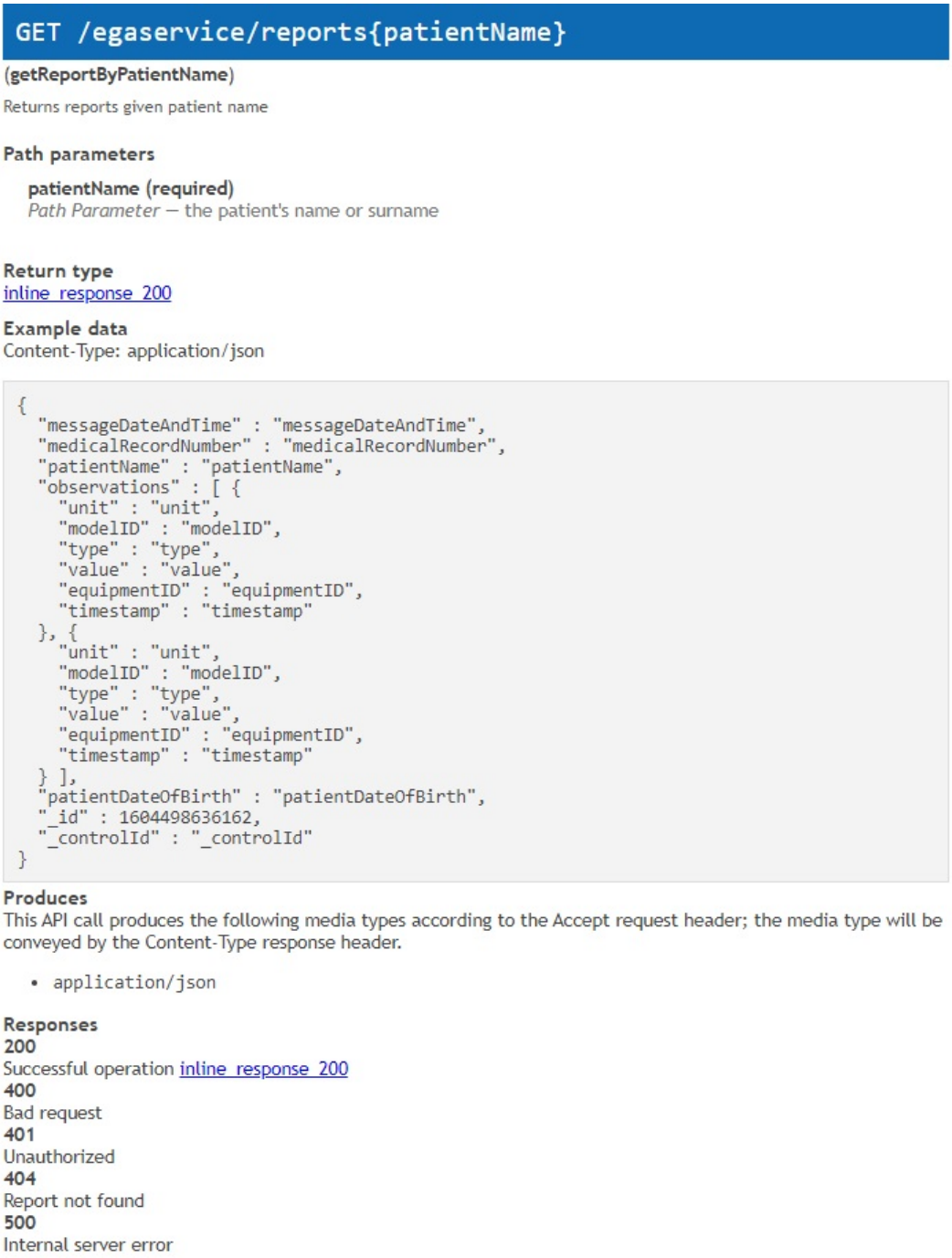
- application/json

Responses
200
Successful operation inline_response_200
400
Bad request
401
Unauthorized
404
Report not found
500
Internal server error

Figura 3.2: Descrizione dell'API per ottenere un singolo referto

È stata anche introdotta un'API che restituisce tutti i referti relativi al paziente il cui nome è passato come parametro. Se il parametro è vuoto verranno

semplicemente restituiti tutti i referti disponibili.



GET /egaservice/reports{patientName}

(getReportByPatientName)

Returns reports given patient name

Path parameters

patientName (required)
Path Parameter — the patient's name or surname

Return type
inline_response_200

Example data
Content-Type: application/json

```
{
  "messageDateAndTime" : "messageDateAndTime",
  "medicalRecordNumber" : "medicalRecordNumber",
  "patientName" : "patientName",
  "observations" : [ {
    "unit" : "unit",
    "modelID" : "modelID",
    "type" : "type",
    "value" : "value",
    "equipmentID" : "equipmentID",
    "timestamp" : "timestamp"
  }, {
    "unit" : "unit",
    "modelID" : "modelID",
    "type" : "type",
    "value" : "value",
    "equipmentID" : "equipmentID",
    "timestamp" : "timestamp"
  } ],
  "patientDateOfBirth" : "patientDateOfBirth",
  "_id" : 1604498636162,
  "_controlId" : "_controlId"
}
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses
200
Successful operation inline_response_200
400
Bad request
401
Unauthorized
404
Report not found
500
Internal server error

Figura 3.3: Descrizione dell'API per ottenere tutti i referti relativi ad paziente dato il suo nome

A seguito dell'analisi dei requisiti è emerso che in ambito ospedaliero è fortemente utile provvedere alla cancellazione dei risultati delle analisi più obsolete. Naturalmente, non è immaginabile che il personale medico analizzi ogni singolo referto per poi provvedere alla sua cancellazione. Per questo motivo, si è pensato di automatizzare questa funzionalità e lasciare che sia Ega Server stesso a effettuare periodicamente il controllo e l'eventuale eliminazione. Gli utenti del sistema dovranno semplicemente impostare il criterio di cancellazione dei referti, ovvero scegliere il periodo di tempo entro cui un referto deve essere stato prodotto per non essere cancellato. Per implementare questa necessità si è scelto di introdurre una risorsa aggiuntiva riferita proprio a questo criterio di cancellazione.

Tramite una richiesta HTTP GET è possibile ottenere l'attuale criterio di cancellazione dei referti.

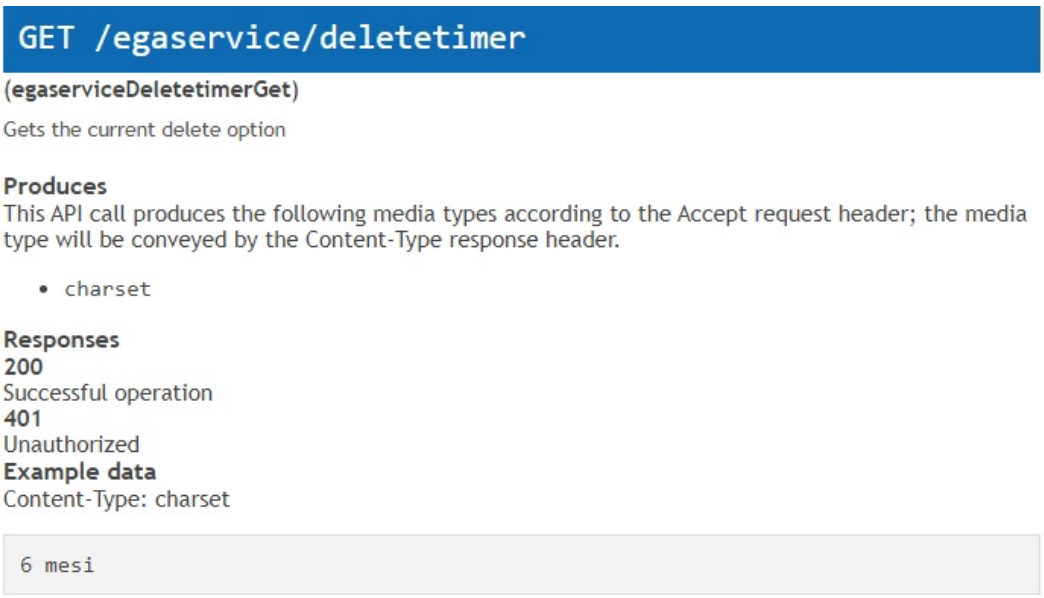


Figura 3.4: Descrizione dell'API per ottenere l'attuale impostazione di cancellazione dei referti

Tramite richiesta HTTP PUT, ogni utente autorizzato potrà cambiare il criterio di cancellazione dei referti.

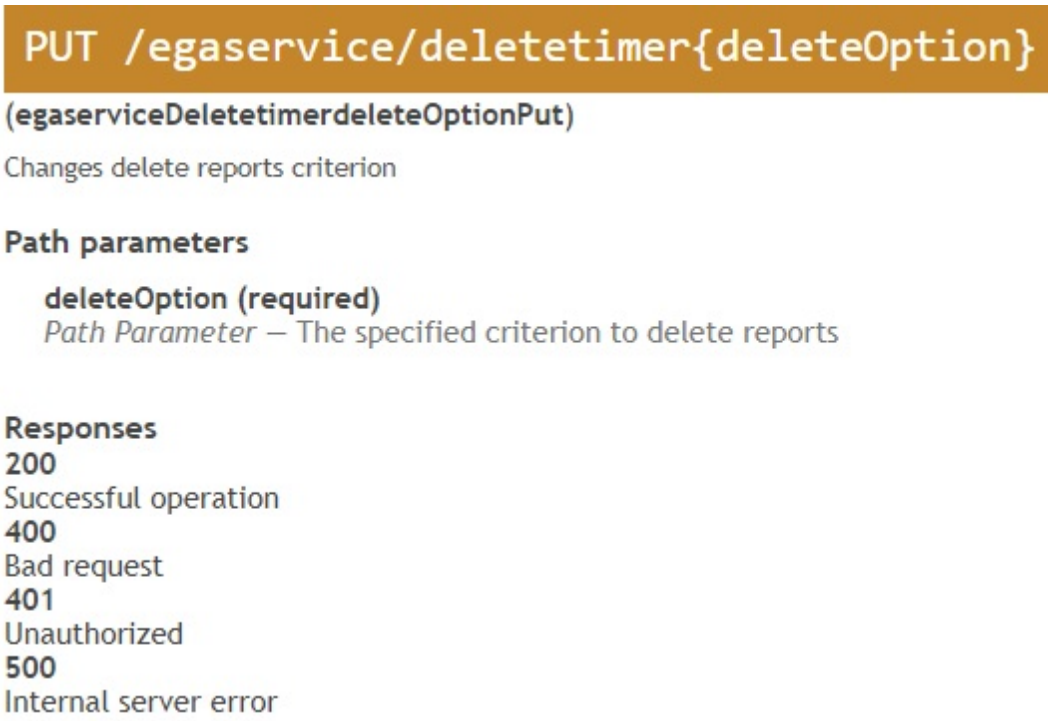


Figura 3.5: Descrizione dell'API per cambiare il criterio di cancellazione dei referti

In considerazione della specifica dei requisiti, nonché di tutti gli obiettivi prefissati nei capitoli precedenti, per garantire l'accesso al sistema agli utenti che usufruiranno dell'applicazione web è stata introdotta la risorsa utenti. Al momento, la gestione delle richieste verso questa risorsa riguardano esclusivamente i tentativi di login e disconnessione dal sistema ma potranno eventualmente essere integrate in caso sia necessario introdurre una gestione specifica degli utenti registrati. Per effettuare il login è quindi sufficiente effettuare una richiesta HTTP GET specificando le credenziali dell'utente e aggiungendo il parametro "isLogin".

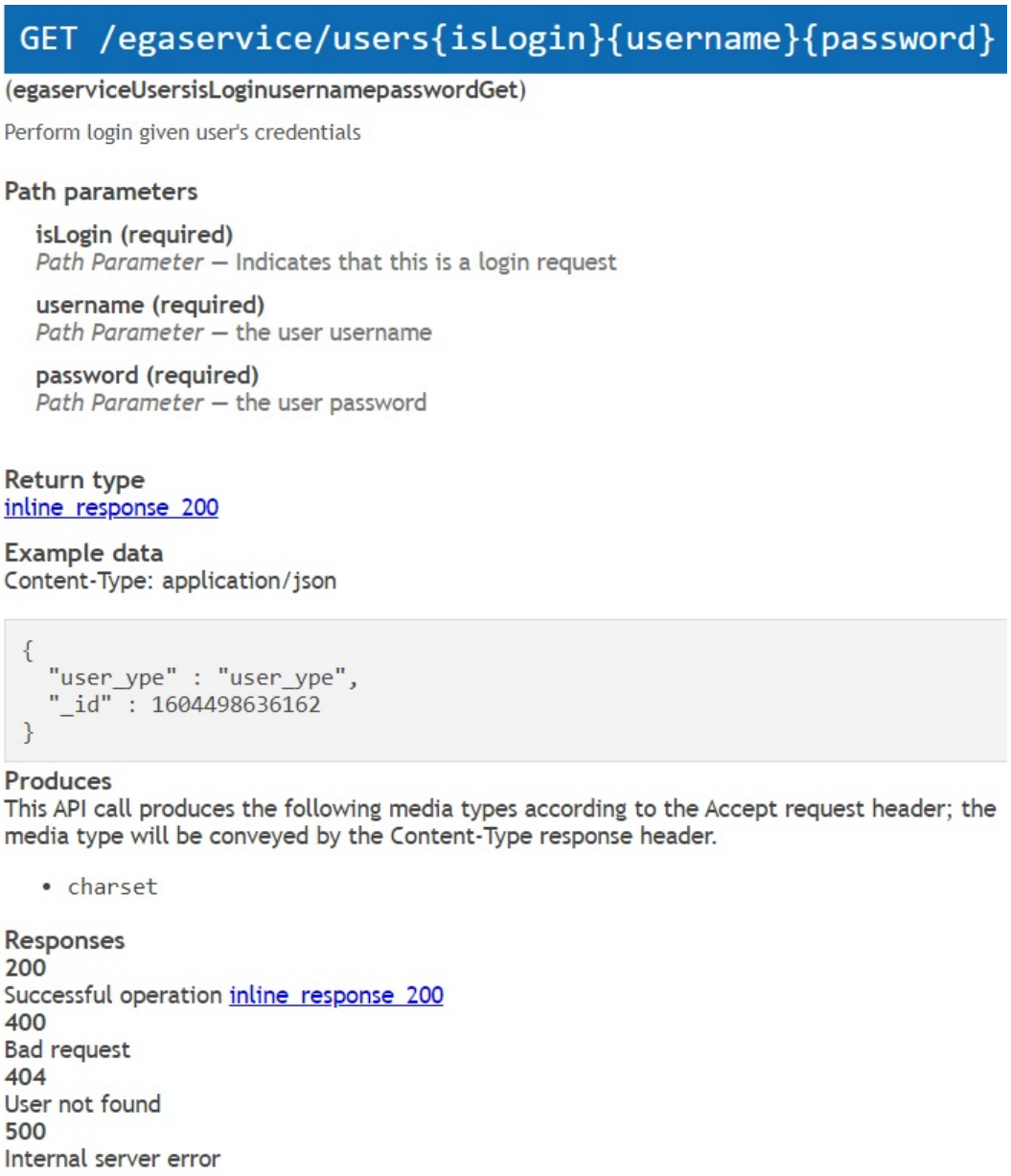


Figura 3.6: Descrizione dell'API per accedere al sistema tramite chiamata HTTP GET

È stato deciso di prevedere un'equivalente gestione di una richiesta HTTP POST per supportare il tradizionale login tramite form HTML.

Figura 3.7: Descrizione dell'API per accedere al sistema tramite chiamata HTTP POST

In entrambe le chiamate è possibile aggiungere il parametro opzionale "keepUserLogged" che, se contiene il valore "true", indica al server di creare i cookie che consentono all'utente di mantenere l'accesso al sistema e di non dover eseguire frequentemente la procedura di login. Infine, per consentire il logout dell'utente è stata predisposta un'API che, tramite una richiesta HTTP DELETE e un apposito parametro, consentisse la cancellazione dei cookie im-

postati in fase di login, permettendo poi all'applicazione web di completare correttamente la disconnessione dal sistema.

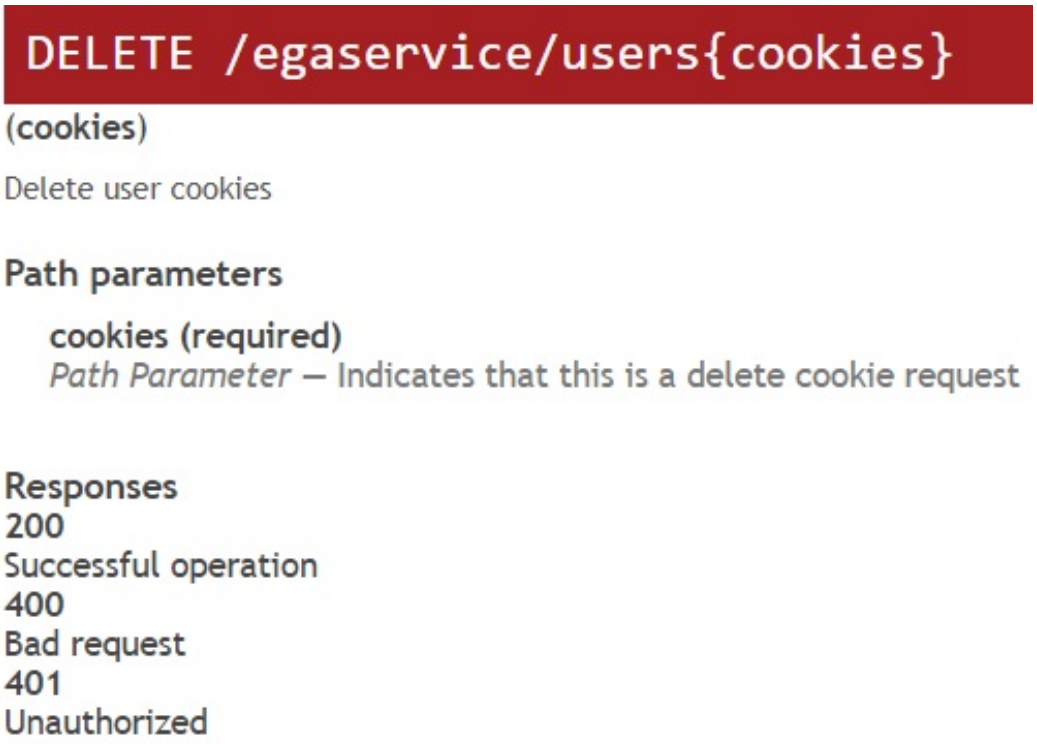


Figura 3.8: Descrizione dell'API per cancellare i cookie e permettere il logout dell'utente

3.2 Architettura logica del sistema

Alla luce delle risorse appena introdotte, si procede con la descrizione della struttura logica di EGA Server integrando e ampliando la modellazione del prototipo di partenza.

Il primo aspetto trattato è la gestione delle risorse che il server dovrà mettere a disposizione. La figura 3.9 mostra come ogni risorsa di cui il sistema necessità potrà essere modellata e gestita in modo appropriato. La classe EgaService si occupa quindi della creazione del server HTTP che rende possibile l'accesso alle risorse.

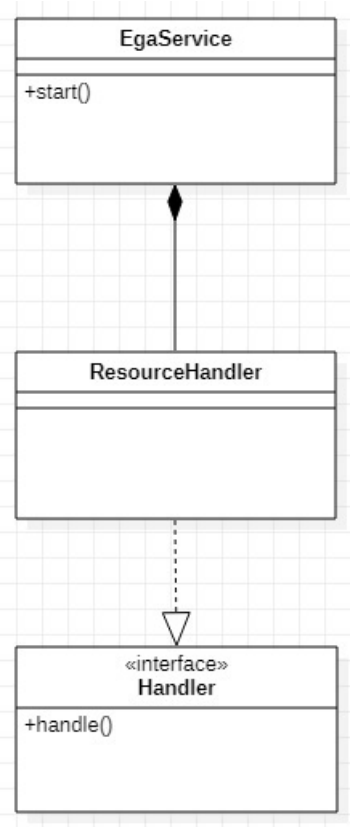


Figura 3.9: Diagramma dell'architettura logica di EGA Server incentrato sull'introduzione della gestione delle risorse

Data l'eterogeneità intrinseca nella gestione delle risorse messe a disposizione da EGA Server, si è scelto di associare a ogni risorsa uno specifico gestore con una classe dedicata mantenendo tuttavia un'interfaccia comune per garantire un corretto e uniforme adempimento delle richieste ricevute dai client. In questo modo è stato infatti possibile strutturare uniformemente l'organizzazione di ogni classe dedita alla gestione di una risorsa, garantendone la modularità e la scalabilità, lasciando al tempo stesso libertà di gestione secondo le specifiche esigenze di ciascuna risorsa.

Considerata la particolare struttura di questa modellazione, per metterne in luce gli aspetti salienti, viene ora presentato un diagramma dell'architettura logica di EGA Server incentrato sulla gestione di tutte le risorse descritte.

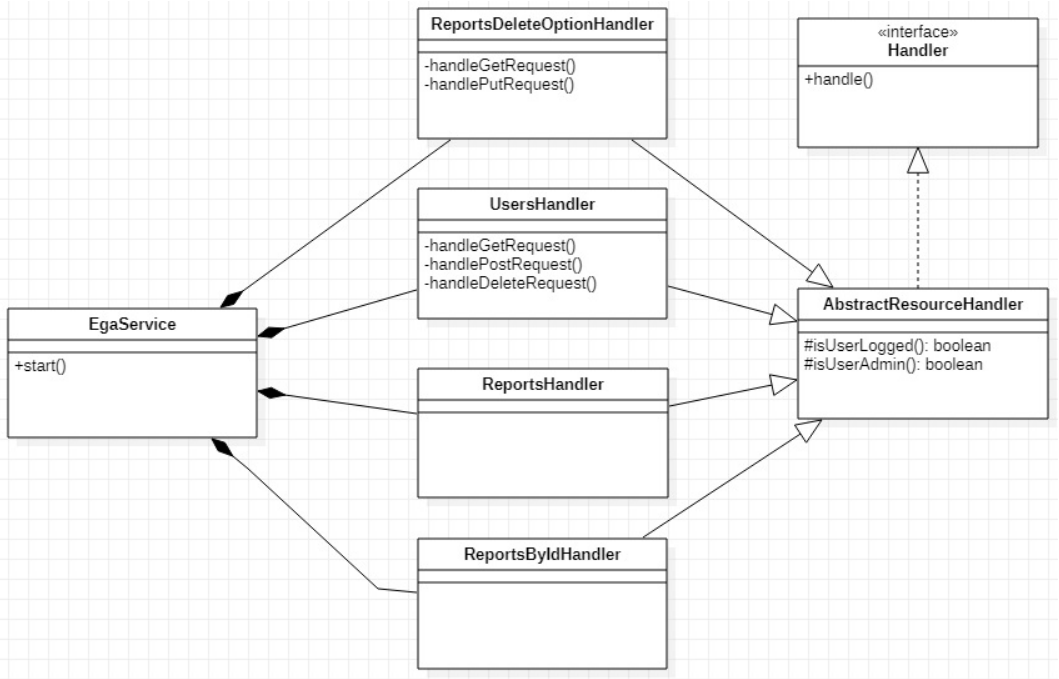


Figura 3.10: Diagramma dell'architettura logica di EGA Server relativo alla gestione delle risorse REST. Ogni gestore di risorsa implementa esclusivamente i metodi necessari all'ottemperanza delle richieste relative alla risorsa stessa

Come si evince dalla figura 3.10, è stata modellata anche un'entità astratta volta a rappresentare un generico gestore di risorsa. Questa classe, oltre ad essere predisposta all'implementazione di tutti i comportamenti che un qualsiasi gestore di risorsa deve avere, fattorizza tutti i metodi necessari a tutelare l'accesso alle risorse da parte di soggetti non autorizzati.

La relazione di composizione tra EgaService e i gestori è stata introdotta in quanto, già nella versione prototipale del sistema, EgaService è la classe designata alla memorizzazione degli URL di tutte le risorse e all'esposizione delle API REST.

Come introdotto precedentemente e in linea con quanto già implementato nel prototipo, la classe EgaService si occupa anche del dispiegamento del server predisposto alla ricezione dei messaggi HL7. La classe OruEndPoint è invece dedita all'estrapolazione di tutte le informazioni significative nei messaggi HL7 ricevuti.

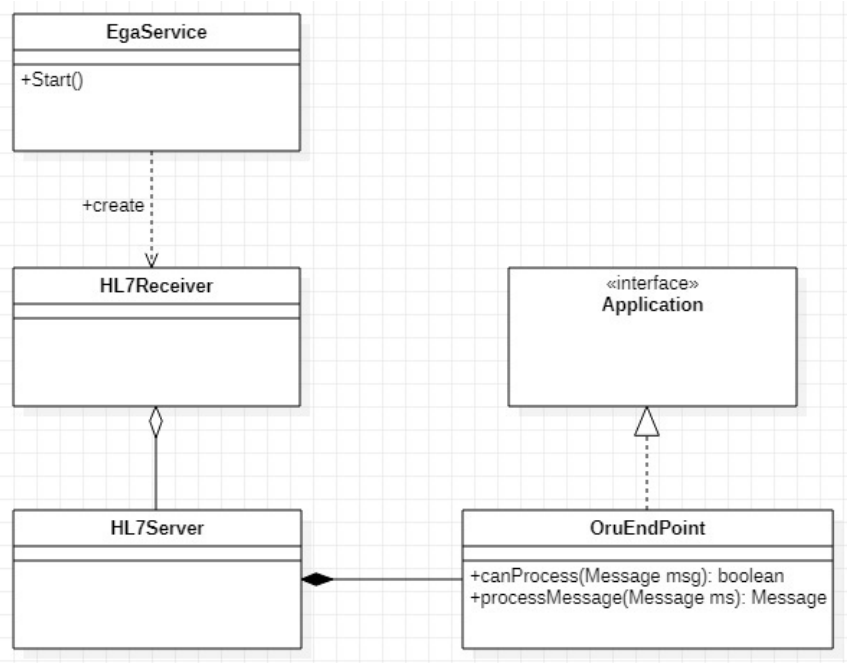


Figura 3.11: Diagramma relativo alla modellazione del server volto alla ricezione dei messaggi HL7

La modellazione illustrata consente quindi di presentare il diagramma completo dell'architettura logica di Ega Server.

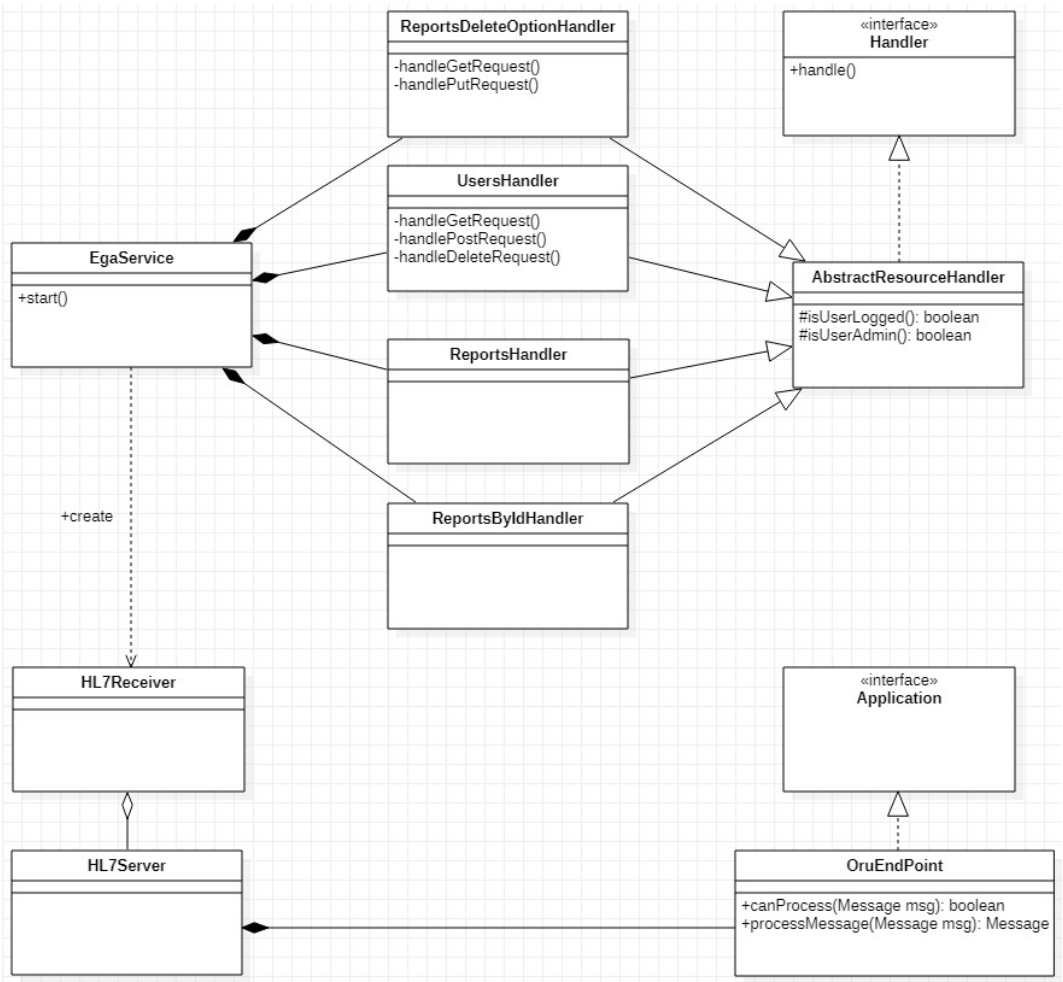


Figura 3.12: Diagramma completo dell'architettura logica di Ega Server

L'introduzione di FHIR è agevolmente realizzabile integrando un server FHIR nel sistema. Le risorse FHIR possono invece essere efficacemente create avvalendosi delle informazioni contenute nei messaggi HL7 ricevuti, è quindi possibile sfruttare le librerie già presenti per il recupero dei dati di interesse e la strutturazione secondo la specifica FHIR. Il server FHIR potrà quindi accedere a queste risorse e renderle disponibili alla consultazione, gestione ed eventuale modifica.

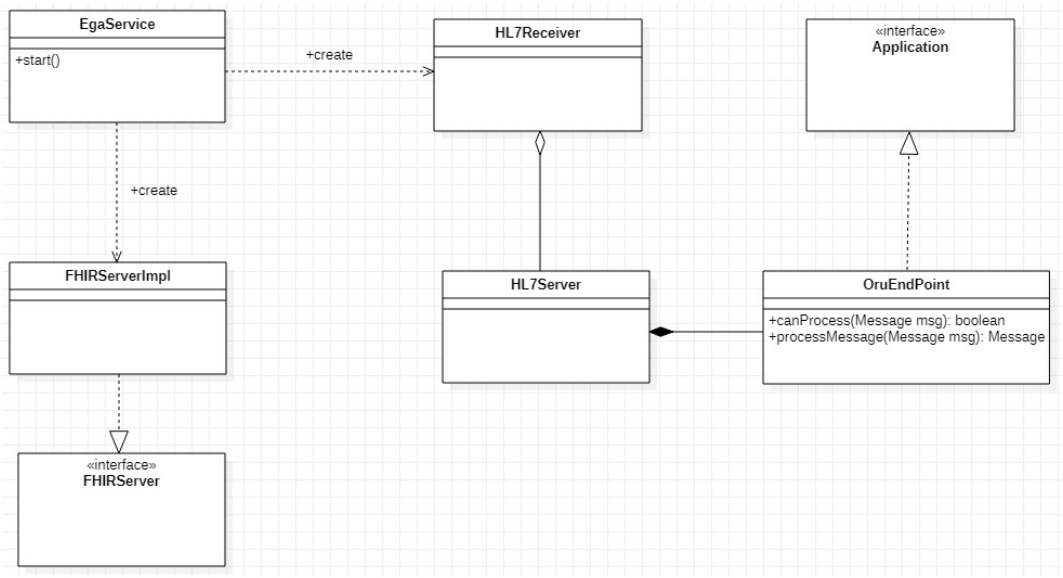


Figura 3.13: Diagramma dell'architettura logica riguardante l'introduzione del server FHIR

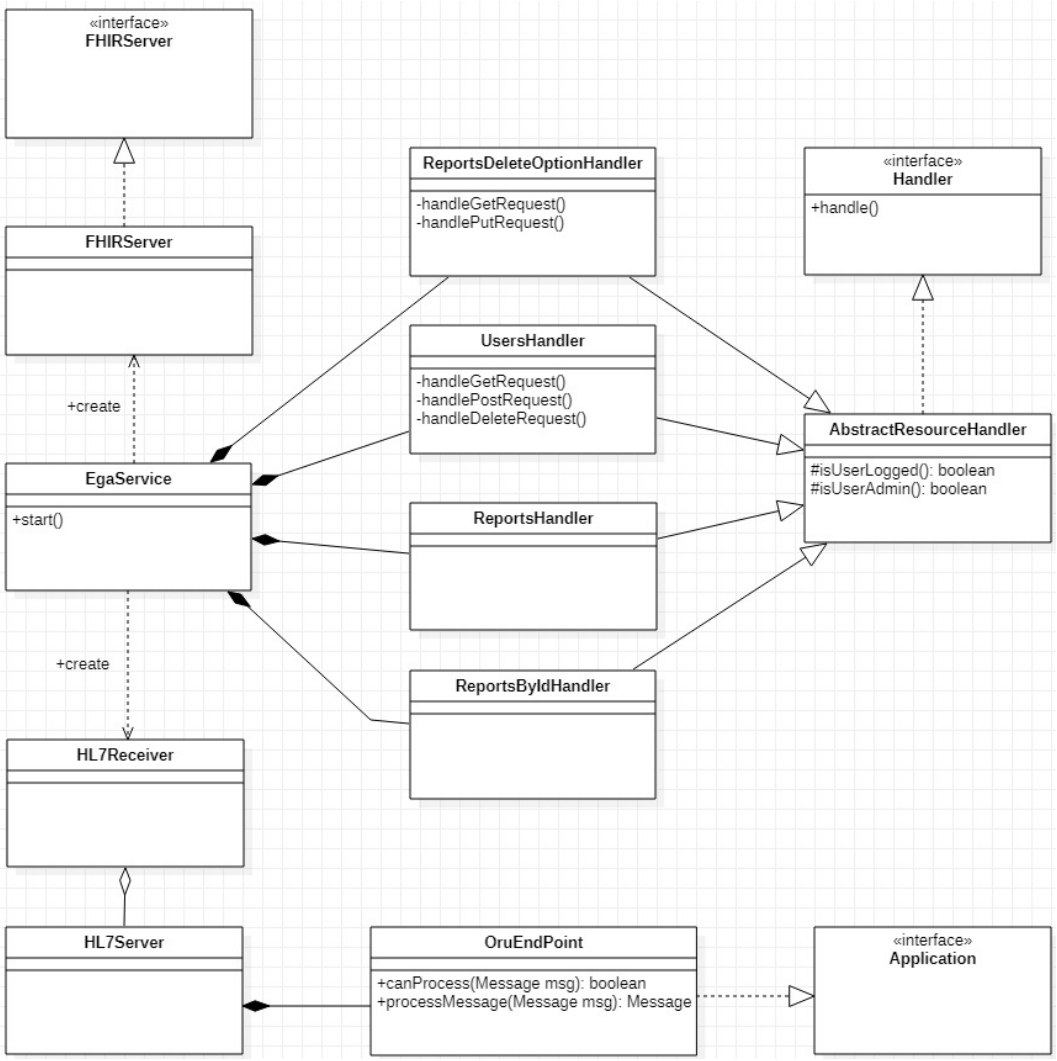


Figura 3.14: Diagramma completo dell'architettura logica del sistema

3.3 Sviluppo

Viene ora approfondito lo sviluppo del software del sistema presentandone le parti più significative, l'implementazione delle funzionalità definite in fase di analisi, dei modelli definiti in fase di progettazione e mostrando i frammenti di codice sorgente più rilevanti. Verrà trattato prima lo sviluppo di EGA Server e successivamente lo sviluppo dell'applicazione web.

3.3.1 Sviluppo di EGA Server

La classe EgaService

La descrizione dello sviluppo di EGA Server inizia dalla classe EgaService, una classe fondamentale in quanto, come definito in fase di progettazione, oltre a contenere gli URL di tutte le risorse ed esporre le API REST associate, si occupa anche di dispiegare il server HTTP che fornirà l'accesso alle risorse e di predisporre le condizioni necessarie alla creazione del server HL7 dedicato alla ricezione dei relativi messaggi. Vert.x mette a disposizione delle librerie che permettono di implementare facilmente, velocemente, adeguatamente e intuitivamente tutte queste funzionalità.

```
public class EgaService extends AbstractVerticle {

    @Override
    public void start() {
      vertx.executeBlocking(handler -> {
         vertx.deployVerticle("t4c.ega.service.hl7.HL7Receiver");
         handler.complete();
      }, result -> {
         if (result.succeeded()) {
            log("HL7 Receiver Deployed");
         }
      });
   }

}
```

Listato 3.1: Dispiegare HL7Receiver con Vert.x

```
final Router router = Router.router(vertx);

this.configureAPI(router);

vertx.createHttpServer().requestHandler(router)
   .listen(Configuration.ServiceUtil.TCP_PORT, result -> {
      if (result.succeeded()) {
         log("Ready.");
      } else {
         log("Failed: " + result.cause());
      }
});
```

Listato 3.2: Dispiegare il server HTTP con Vert.x

Grazie all'entità Router messa a disposizione da Vert.x è possibile definire automaticamente i criteri secondo i quali una determinata richiesta HTTP debba essere affidata a un apposito gestore specificato in fase di implementazione. L'associazione dell'URL di ogni risorsa e la conseguente esposizione delle API REST è contenuta nel metodo configureAPI(router).

```
private void configureAPI(final Router router) {
   router.get(ResourcesPaths.REPORTS_RESOURCE_PATH.getValue())
         .handler(new ReportsHandler()::handle);

   router.get(ResourcesPaths.REPORT_BY_ID_PATH.getValue())
         .handler(new ReportsByIdHandler()::handle);

   router.get(ResourcesPaths.USER_RESOURCE_PATH.getValue())
         .handler(new UsersHandler()::handle);

   router.post(ResourcesPaths.USER_RESOURCE_PATH.getValue())
         .handler(new UsersHandler()::handle);

   router.delete(ResourcesPaths.USER_COOKIES_RESOURCE_PATH.getValue())
         .handler(new UsersHandler()::handle);

   router.get(ResourcesPaths.REPORTS_DELETE_OPTION_PATH.getValue())
         .handler(ReportsDeleteOptionHandler
               .getDeleteReportsHandlerUtil()::handle);

   router.put(ResourcesPaths.REPORTS_DELETE_OPTION_PATH.getValue().
         handler(ReportsDeleteOptionHandler
               .getDeleteReportsHandlerUtil()::handle);

}
```

Listato 3.3: Associazione dei gestori alle relative risorse ed esposizione delle API REST con Vert.x

Come si evince dal listato 3.3, in questo metodo viene associata ad ogni risorsa la classe predisposta alla sua gestione, esattamente come mostrato in fase di progettazione. L'URL delle risorse è contenuto nell'enumerazione privata ResourcesPath innestata nella classe stessa, questo ha permesso di rendere molto pratico, efficace, esplicativo e privo di errori l'utilizzo degli URL senza però esporre inopportunamente questi percorsi alle classi che non ne hanno necessità.

Il server HL7

Per quanto concerne il dispiegamento del server HL7 volto alla ricezione, all'interpretazione e alla memorizzazione nel database locale dei messaggi HL7, è stato implementato sfruttando congiuntamente le librerie messe a disposizione da Vert.x e da HAPI.

```
private final HapiContext context = new DefaultHapiContext();

final HL7Service server =
    context.newServer(Configuration.ServerUtil.TCP_PORT,
      Configuration.ServerUtil.TLS_ACTIVE);

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

try {
   server.startAndWait();
   log("Started.");
} catch (InterruptedException exception) {
   exception.printStackTrace();
}
```

Listato 3.4: Avvio del server HL7 con HAPI nella classe HL7Receiver

L'istanza della classe OruEndPoint si occupa di processare tutti i messaggi HL7 ricevuti e di memorizzare nel database il contenuto dei campi di interesse. La prima funzionalità è efficientemente supportata dalle librerie HAPI mentre la seconda è stata raggiunta grazie all'altrettanto semplice ed efficace implementazione da parte di Vert.x di tutti i metodi necessari alla gestione dei dati in formato JSON e alla creazione di un client MongoDB che permette di salvare facilmente documenti nel database.

```
final JsonObject requiredHL7Fileds = new JsonObject();

try {
   final JsonArray vitalsigns = new JsonArray();

   final ORU_R01 oru = (ORU_R01) msg;

   requiredHL7Fileds.put(ReportsCollectionDocumentsFields
         .CONTROL_ID.getValue(),
               oru.getMSH().getMessageControlID().getValueOrEmpty());
```

```
   requiredHL7Fileds.put(ReportsCollectionDocumentsFields
         .MESSAGE_DATE_AND_TIME.getValue(),
               oru.getMSH().getMsh7_DateTimeOfMessage().encode());

   requiredHL7Fileds.put(ReportsCollectionDocumentsFields
         .MEDICAL_RECORD_NUMBER.getValue(),
               oru.getPATIENT_RESULT().getPATIENT().getPID()
                     .getPid3_PatientIdentifierList(0).getID()
                           .getValueOrEmpty());

   requiredHL7Fileds.put(ReportsCollectionDocumentsFields
      .PATIENT_DATE_OF_BIRTH.getValue(),
          oru.getPATIENT_RESULT().getPATIENT()
         .getPID().getPid7_DateTimeOfBirth().encode());

   for (final ORU_R01_OBSERVATION observation :
      oru.getPATIENT_RESULT().getORDER_OBSERVATION()
         .getOBSERVATIONAll()) {
      vitalsigns.add(retrieveRequiredOBXFields(observation));
   }

} catch (HL7Exception | IOException exception) {
   exception.printStackTrace();
}
```

Listato 3.5: Esempio di utilizzo di HAPI per ricavare le informazioni necessarie da un messaggio HL7. I dati sono inseriti in un'istanza di JSON per prepararli al salvataggio sul database

```
private void saveMessageToDB(final JsonObject document) {
   final MongoClient mongoClient = MongoClient.createShared(vertx,
       MongoDBUtil.mongoAccessConfig());

   mongoClient.insert(Configuration.DbUtil.REPORTS_COLLECTION,
       document, res -> {
       if (res.succeeded()) {
         log("Message saved.");
       } else {
         res.cause().printStackTrace();
       }
   });
}
```

Listato 3.6: Salvataggio di un documento nel database con Vert.x

MongoDBUtil è una classe di utilità implementata appositamente per fattorizzare i metodi comuni necessari alla configurazione del client MongoDB.

La gestione di una risorsa generica

Trattiamo ora le classi più significative volte alla gestione delle richieste di accesso alle risorse messe a disposizione dal server. Per prima cosa, si analizza come la classe AbstractResourceHandler implementi il controllo dell'accesso alle risorse.

```
protected boolean isUserLogged(final RoutingContext routingContext) {
   return routingContext.getCookie(
      CookieNames.USER_ID_COOKIE.getValue()) != null;
}

protected boolean isUserAdmin(final RoutingContext routingContext) {
   return routingContext.getCookie(
      CookieNames.USER_TYPE_COOKIE.getValue()) != null;
}
```

Listato 3.7: Implementazione dei metodi necessari al controllo dell'accesso. Ogni classe volta alla gestione delle risorse ne è provvista

Il primo metodo controlla quindi che il richiedente si sia precedentemente autenticato verificando la presenza del cookie registrato in fase di login. Il secondo invece, serve per garantire l'accesso alle risorse riservate ai soli utenti con privilegi amministrativi.

La gestione della risorsa utenti

La classe UsersHandler è la classe designata alla gestione di tutte le richieste rivolte alla risorsa utenti. Dato che UsersHandler deve prevedere la gestione di diversi tipi di richieste HTTP, il metodo handle() si apre con uno switch volto a redirigere ogni richiesta al metodo di competenza.

```
@Override
public void handle(final RoutingContext routingContext) {
   switch (routingContext.request().method()) {

   case GET:
      this.handleGetRequest(routingContext);
      return;

   case POST:
```

```
      this.handlePostRequest(routingContext);
      return;

   case DELETE:
      this.handleDeleteRequest(routingContext);
      return;

   default:
      break;

   }

}
```

Listato 3.8: Il metodo handle() della classe UsersHandler. Il parametro routingContext è ovviamente fornito da Vert.x per contribuire alla corretta gestione della richiesta

In considerazione di quanto analizzato nella descrizione delle API RESTFul, i metodi presentati implementano solo le funzionalità di login/logout richieste senza tuttavia precludere alla possibilità di aggiungere una gestione diversa in caso dovesse essercene bisogno in futuro.

In linea con quanto appena descritto, le richieste GET e POST (preventivamente esaminate dai metodi indicati) se accettate, garantiranno l'accesso al sistema e alle risorse. In questo contesto, la richiesta POST è stata pensata per supportare la sottomissione dei dati nella form dell'applicazione web.

Le richieste di accesso vengono quindi agilmente gestite all'interno di questa classe. Il recupero dei dati inseriti dall'utente in fase di login è facilmente recuperabile dalla richiesta HTTP.

```
final String username = routingContext.request()
                    .getParam(UsersCollectionDocumentsFields
                       .USERNAME.getValue());
final String password = routingContext.request()
                    .getParam(UsersCollectionDocumentsFields
                       .PASSWORD.getValue());
```

Listato 3.9: Recupero delle credenziali dell'utente tramite Vert.x

Per verificare che le credenziali inserite corrispondano effettivamente a un utente registrato nel database è sufficiente usare il modulo di Vert.x che consente la comunicazione con MongoDB. Verrà quindi effettuata una ricerca dell'utente all'interno database e verrà restituita al client una specifica risposta HTTP (con il corrispettivo messaggio di stato) in base all'esito.

```
final JsonObject query = new JsonObject().put(
   UsersCollectionDocumentsFields.USERNAME.getValue(), username)
      .put(UsersCollectionDocumentsFields.PASSWORD.getValue(),
         password);

final MongoClient mongoClient = MongoClient.createShared(
   routingContext.vertx(), MongoDBUtil.mongoAccessConfig());

mongoClient.find(Configuration.DbUtil.USERS_COLLECTION, query,
    resultHandler -> {
   // Search error
   if (!resultHandler.succeeded()) {
      routingContext.response().setStatusCode(
         HTTPStatusCodes.INTERNAL_SERVER_ERROR.getValue()).end();
      resultHandler.cause().printStackTrace();
   }
   // User found
   else if (!resultHandler.result().isEmpty()) {
      this.setCookies(routingContext, resultHandler.result().get(0));

      routingContext.response().putHeader(
         HttpHeaderNames.CONTENT_TYPE, HttpHeaderValues.CHARSET)
            .setStatusCode(HTTPStatusCodes.OK.getValue())
               .end(resultHandler.result().get(0)
                  .getString(UsersCollectionDocumentsFields.USER_TYPE
                     .getValue())
                        .toString());
   }
   // User not found
   else {
      routingContext.response().setStatusCode(
         HTTPStatusCodes.NOT_FOUND.getValue()).end();
   }
});
```

Listato 3.10: Ricerca dell'utente nel database e invio della risposta HTTP al client. Si noti la struttura asincrona alla base di Vert.x

L'implementazione della classe si conclude con la creazione e la cancellazione dei cookies necessari a mantenere l'accesso dell'utente al sistema.

```
//The cookies last 5 years.
final int cookieAge = 3600 * 24 * 365 * 5;
```

```
//Create cookie
routingContext.addCookie(Cookie.cookie(CookieNames.USER_ID_COOKIE
   .getValue(), String.valueOf(userData
      .getLong(UsersCollectionDocumentsFields.USER_ID.getValue())))
         .setMaxAge(cookieAge));

//Delete every cookie
routingContext.cookieMap().keySet()
   .forEach(cookieName -> routingContext.removeCookie(cookieName));
```

Listato 3.11: Esempio di creazione e cancellazione di cookie di con Vert.x

La risorsa referti

Viene ora approfondita l'implementazione di ReportsHandler e ReportsByIdHandler, le classi predisposte alla gestione delle richieste rivolte alla risorsa referti. Entrambe devono gestire esclusivamente richieste di tipo GET in quanto l'unica azione consentita verso i referti è richiederne il recupero. La cancellazione, come spiegato in seguito, viene infatti gestita in modo diverso. Il comportamento degli handler delle due classi è molto simile perché entrambi devono semplicemente recuperare i referti dal database e, se l'operazione va a buon fine, restituirli al richiedente. L'unica differenza è che, ovviamente, restituire un report specifico richiede preventivamente il recupero del suo codice identificativo dalla richiesta HTTP ricevuta. Gli URL associati alle due classi infatti fanno riferimento allo stesso percorso, in quello di ReportsByIdHandler però, viene anche specificato il codice del referto.

```
final MongoClient mongoClient =
    MongoClient.createShared(routingContext.vertx(),
   MongoDBUtil.mongoAccessConfig());

mongoClient.find(Configuration.DbUtil.REPORTS_COLLECTION, new
    JsonObject(), resultHandler -> {
   if (resultHandler.succeeded()) {
      if (!resultHandler.result().isEmpty()) {
         routingContext.response()
            .putHeader(HttpHeaderNames.CONTENT_TYPE,
               HttpHeaderValues.APPLICATION_JSON)
                  .setStatusCode(HTTPStatusCodes.OK.getValue())
                     .end(new JsonArray(resultHandler.result())
                        .encodePrettily());
      } else {
         routingContext.response().setStatusCode(
```

```
            HTTPStatusCodes.NOT_FOUND.getValue()).end();
      }
   } else {
      routingContext.response().setStatusCode(
         HTTPStatusCodes.INTERNAL_SERVER_ERROR.getValue()).end();
      resultHandler.cause().printStackTrace();
   }
});
```

Listato 3.12: Recupero di tutti i referti dal database

Per facilitare la consultazione dei referti, è stata introdotta la possibilità di recuperare esclusivamente quelli relativi al paziente specificato come parametro della richiesta. Per agevolare l'operazione di ricerca da parte degli utenti, l'esempio seguente mostra come, grazie a uno specifico operatore messo a disposizione da MongoDB, sia possibile predisporre un'interrogazione al database che consenta di recuperare tutti i referti in cui il nome del paziente contenga anche solo in parte il nome passato come parametro della richiesta. I nomi dei pazienti sono memorizzati in maiuscolo per evitare problemi durante la ricerca.

```
final JsonObject query = new JsonObject()
   .put(ReportsCollectionDocumentsFields.PATIENT_NAME.getValue(),
      new JsonObject().put("$regex",
         patientName.toUpperCase(Locale.ITALIAN)));
```

Listato 3.13: Interrogazione che consente il recupero dei referti relativi al nome del paziente passato come parametro. Il resto del codice è analogo a quello mostrato negli esempi precedenti

La cancellazione dei referti

L'ultima trattazione dello sviluppo di EGA Server riguarda la cancellazione dei referti. La classe ReportsDeleteOptionHandler è stata realizzata usufruendo del pattern Singleton in quanto avere più istanze di questa classe avrebbe potuto facilmente portare a farne un uso che comprometterebbe il corretto funzionamento dell'applicazione e l'uniforme gestione dei referti. Inoltre, l'utilizzo di questo pattern non porta problemi di accesso concorrente in quanto le la gestione delle richieste è garantita essere mutualmente esclusiva da Vert.x.

ReportsDeleteOptionHandler risponde a due tipi di richieste HTTP: GET e PUT. La prima permette di ricevere l'attuale impostazione relativa alla cancellazione dei referti, la seconda consente di cambiare questa configurazione e attivare subito la procedura di cancellazione. Data la periodicità intrinseca

della cancellazione dei referti più datati, si è scelto di introdurre un thread che, una volta al giorno, controllerà se e quali referti debbano essere cancellati, procedendo eventualmente con l'eliminazione. ReportsDeleteOptionHandler quindi, dopo aver verificato che il parametro ricevuto insieme alla richiesta PUT sia corretto, risveglia questo thread contenuto nella classe ReportsDeleter.

```
boolean isDeleteOptionValid = false;

for (final DeleteOptions deleteOption : DeleteOptions.values()) {
   if (Objects.equals(deleteOption.getValue(),
       routingContext.request().getParam(DELETE_OPTION_PARAMETER))) {
      this.reportsDeleter.setRoutingContext(routingContext);
      this.reportsDeleter.setDeleteOption(deleteOption);
      isDeleteOptionValid = true;
      break;
   }
}

if (!isDeleteOptionValid) {
   routingContext.response().setStatusCode(
      HTTPStatusCodes.BAD_REQUEST.getValue()).end();
   return;
}

if (!this.reportsDeleter.isAlive()) {
   this.reportsDeleter.start();
}

if (Objects.equals(this.reportsDeleter.getState(),
    State.TIMED_WAITING)) {
   this.reportsDeleter.interrupt();
}
```

Listato 3.14: Validazione dell'impostazione di cancellazione ricevuta e risveglio del thread predisposto alla cancellazione dei referti. DeleteOptions è un'enumerazione privata innestata nella classe ReportsDeleteOptionHandler

ReportsDeleter è una classe innestata all'interno di ReportsDeleteOptionHandler che si occupa dell'effettiva cancellazione dei referti. Considerando che questa classe contiene il thread prima introdotto, è stato necessario predisporre l'implementazione con tutte le misure necessarie a evitare problemi di accesso concorrente per le risorse condivise.

Inoltre, è fondamentale che l'eliminazione automatica dei referti avvenga a intervalli di tempo regolari e ben determinati sia per fare in modo che gli

utenti possano organizzarsi per lavorare sempre con la versione aggiornata dei dati, sia per evitare problemi di sincronizzazione in caso di distribuzione ed esecuzione contemporanea dell'applicazione in molteplici nodi. L'eliminazione dei referti avviene tutti i giorni alle ore 04:00 secondo il fuso orario italiano. Il recupero dell'ora esatta, il calcolo di quanto tempo debba aspettare il thread prima di risvegliarsi e la manipolazione degli istanti di tempo sono stati possibili grazie alle API fornite dal package java.time[1], una libreria moderna, di fondamentale importanza nonché quella di riferimento per quanto riguarda tutte le necessità relative al recupero della data, dell'ora in qualsiasi fuso orario e della manipolazione di intervalli di tempo.

```
public void run() {
   while (true) {
      try {
         this.deleteReports();

         ZonedDateTime nextDeleteTime =
             LocalDateTime.now().atZone(ZoneId.of(ZONE_FOR_TIMEZONE));

         if (nextDeleteTime.getHour() >=
             DELETE_REPORTS_EUROPE_ROME_HOUR) {
            nextDeleteTime = nextDeleteTime.plusDays(1);
         }

         nextDeleteTime = nextDeleteTime
            .withHour(DELETE_REPORTS_EUROPE_ROME_HOUR).withMinute(0)
            .withSecond(0);

         final ZonedDateTime localDateAndTime =
             LocalDateTime.now().atZone(ZoneId.of(ZONE_FOR_TIMEZONE));

         Thread.sleep(localDateAndTime.until(nextDeleteTime,
             ChronoUnit.MILLIS));
      } catch (InterruptedException exception) {
         exception.printStackTrace();
      }

   }
}
```

Listato 3.15: Metodo principale del thread con calcolo del tempo di attesa per la prossima cancellazione

[1] https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html

```
private synchronized void doDelete(final LocalDateTime
    requiredDateAndTime) {
   final MongoClient mongoClient = MongoClient.createShared(
      routingContext.vertx(), MongoDBUtil.mongoAccessConfig());

   final JsonObject query = new JsonObject().put(
      ReportsCollectionDocumentsFields.MESSAGE_DATE_AND_TIME.getValue(),
         new JsonObject().put(MONGODB_LESS_OR_EQUAL,
            this.getRAPIDComm500DateAndTime(requiredDateAndTime)));

   mongoClient.removeDocuments(Configuration.DbUtil.REPORTS_COLLECTION,
       query, res -> {
      if (res.succeeded()) {
          log("Old reports deleted");
          if (!this.routingContext.response().ended()) {
            this.routingContext.response()
               .setStatusCode(HTTPStatusCodes.OK.getValue()).end();
          }
      } else {
          if (!this.routingContext.response().ended()) {
            this.routingContext.response()
               .setStatusCode(HTTPStatusCodes
                  .INTERNAL_SERVER_ERROR.getValue()).end();
          }
          res.cause().printStackTrace();
      }
   });

}
```

Listato 3.16: Cancellazione dei referti. La data e l'ora ricevuti in ingresso vengono convertiti nello stesso formato usato da HL7

3.3.2 Sviluppo dell'applicazione web

La trattazione dello sviluppo dell'applicazione web seguirà il percorso di navigazione dell'utente all'interno delle pagine dell'applicazione stessa. Per ogni pagina verranno presentate le funzionalità offerte con annesso codice implementativo di quelle più significative ovvero, nello specifico, le funzionalità che permettono di interfacciarsi con le risorse messe a disposizione da EGA Server.

Schermata di login

La pagina iniziale dell'applicazione è la schermata di login. In questa pagina l'utente potrà inserire le sue credenziali per ottenere l'accesso al sistema. Sono stati ovviamente implementati tutti i tradizionali controlli volti a verificare che le credenziali fornite siano corrette. Se l'utente spunta l'opzione "Rimani collegato", non sarà più necessario effettuare il login fino a quando non sarà egli stesso a scegliere di effettuare la disconnessione. Se l'opzione non viene selezionata, l'utente rimarrà connesso fino alla chiusura del browser.

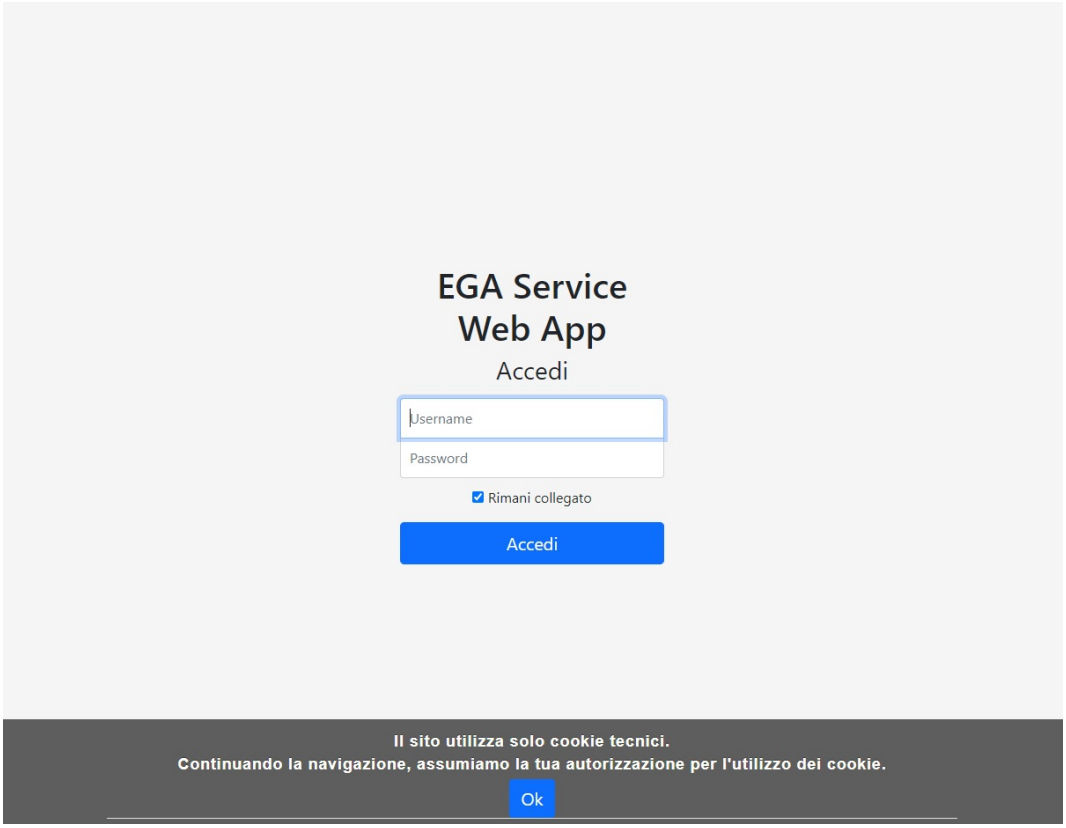


Figura 3.15: La schermata di login

Tutte le funzionalità concernenti l'accesso alle risorse di EGA Server sono state implementate con AJAX[2], in particolare, in questa pagina viene effettuata una richiesta GET per tentare il login automatico al sistema e viene predisposta una richiesta POST che verrà inviata non appena l'utente premerà il bottone di login.

[2] https://www.w3schools.com/xml/ajax_intro.asp

```
$.get(usersResourcePath, {isLogin: "true"})
   .done(function(data) {
      location.replace("/reports/reports.html");
   })
   .fail(function(xhr, textStatus, errorThrown) {

   });
```

Listato 3.17: Metodo AJAX per effettuare il login automatico. Il parametro "isLogin" è necessario per comunicare al server che la richiesta riguarda un tentativo di login

```
$.post(usersResourcePath,{isLogin: "true",
   username: $("#username").val(), password: $("#password").val(),
      keepUserLogged: ($("#keepUserLogged").prop("checked") ===
          true) ? "true" : "false"})
         .done(function(userType) {
            if (userType === administratorString) {
               localStorage.setItem(isUserAdmin, true);
            } else {
               localStorage.setItem(isUserAdmin, false);
            }

            localStorage.setItem(username, $("#username").val());

            location.replace("/reports/reports.html");
         });
```

Listato 3.18: Richiesta AJAX POST per effettuare il login previa richiesta dell'utente. Naturalmente, la richiesta viene inviata solo se i dati forniti sono validi e corretti

I dati salvati in locale a seguito dell'esito positivo della richiesta POST sono necessari esclusivamente per il corretto funzionamento dell'applicazione e non compromettono in nessun modo la sicurezza che viene comunque gestita efficacemente anche lato server.

Pagina dei referti

Dopo aver effettuato il login, l'utente viene indirizzato alla pagina contenente l'elenco di tutti i referti relativi all'emogasanalisi (figura figura 3.16). Data la quantità ingente di referti arrivati al sistema, per agevolare la navigazione dell'utente e semplificare notevolmente la consultazione, si è deciso di suddividere la visualizzazione dei referti in pagine (figura 3.17). Attualmente,

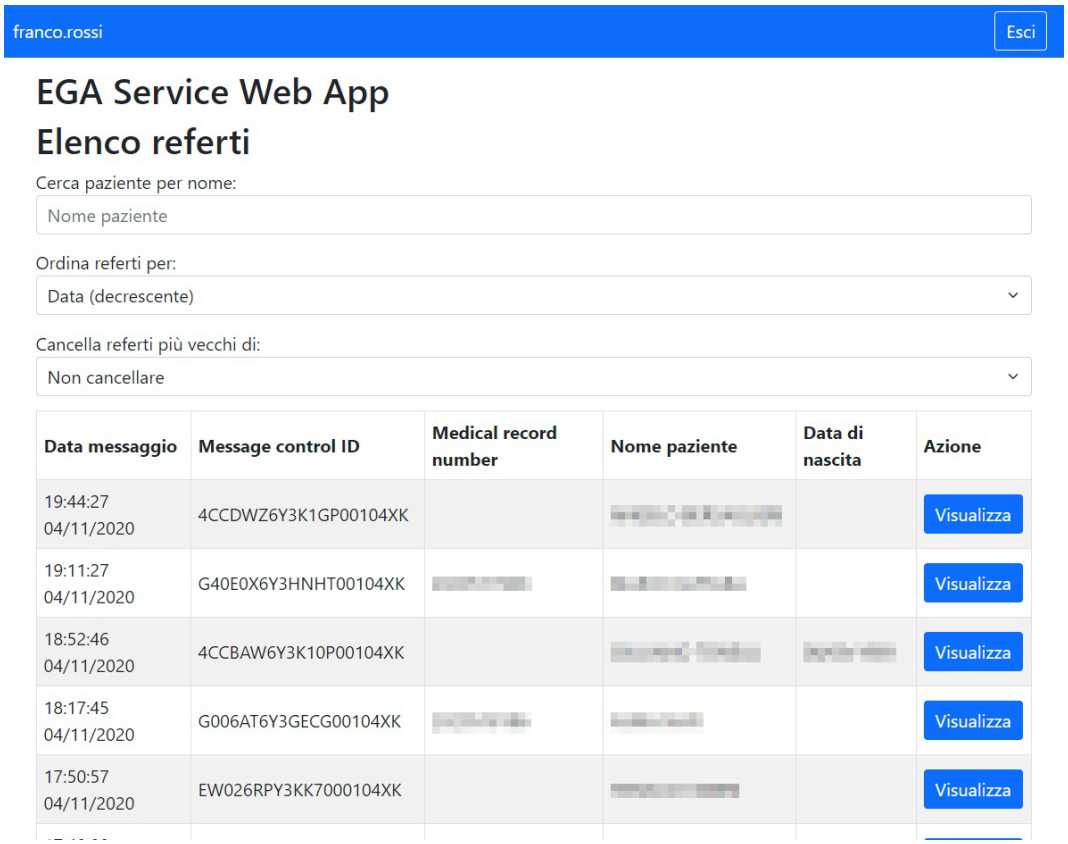


Figura 3.16: Schermata contenente l'elenco dei reports

ogni pagina, tranne eventualmente l'ultima, ne contiene esattamente cinquanta. Il bottone "Visualizza" accanto a ogni referto permette di visualizzare nel dettaglio il risultato dell'emogasanalisi. È stato scelto questo particolare design in quanto permette chiaramente e istantaneamente di capire in che modo sia possibile interagire con ogni referto. In fase di progettazione del design si era anche pensato ad altre soluzioni come, ad esempio, rendere cliccabili le righe della tabella. Tuttavia, questa soluzione, come confermato dall'esperienza di navigazione di diversi utenti, oltre a rendere poco intuitiva la possibilità di visualizzare i referti, sarebbe stata anche molto poco ergonomica, in quanto, utilizzando l'applicazione da un dispositivo mobile come un tablet o uno smartphone sarebbe stato semplice aprire involontariamente un referto durante lo scorrimento dell'elenco. In aggiunta, questa particolare organizzazione permette di aggiungere allo stesso modo altre funzionalità come la modifica o la cancellazione di un referto specifico.

È inoltre possibile ordinare i referti per data (più recente o meno recente), per message control ID o per medical record number. La ricerca dei referti

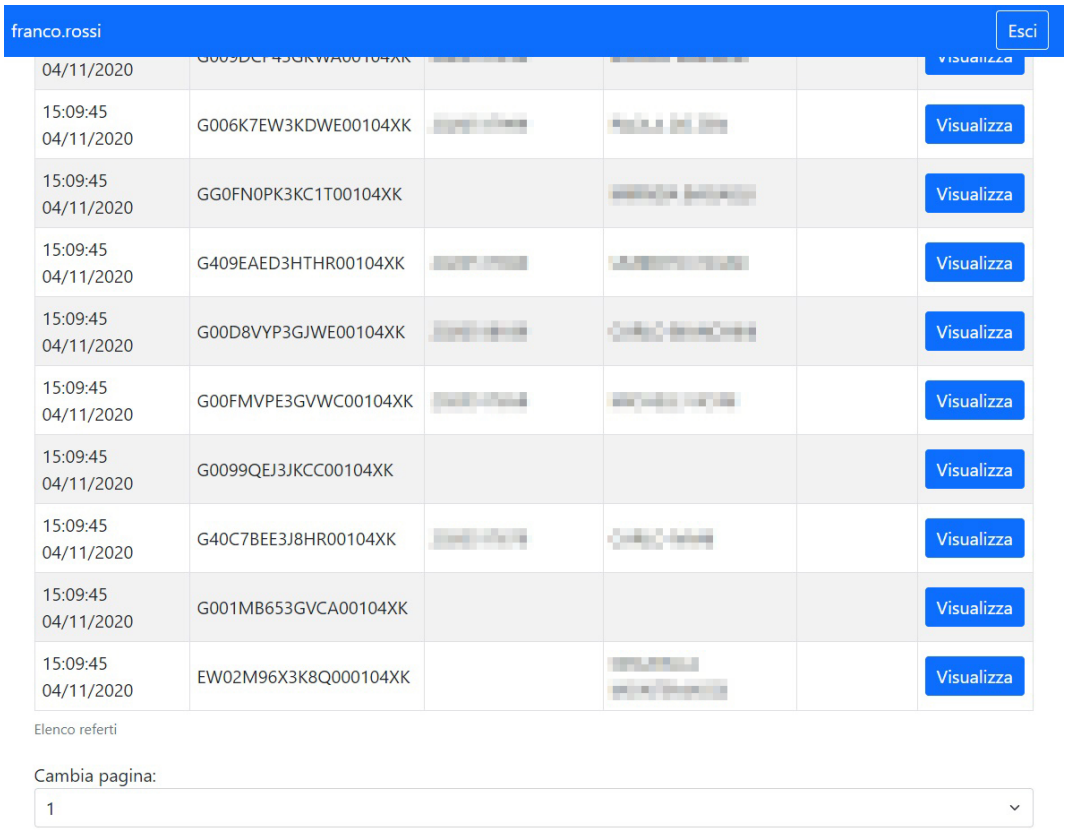


Figura 3.17: L'elenco dei referti è suddiviso in pagine

relativi a uno specifico paziente è stata implementata in modo tale che l'elenco dei referti sia filtrato durante la scrittura stessa del nome.

La richiesta di visualizzazione dei referti avviene tramite una semplice chiamata GET. La tabella non viene visualizzata se la richiesta non dovesse andare a buon fine.

```
$.get(reportsResourcePath, {
      patientName: sessionStorage.getItem(searchPatientByNameText)})
   .done(function(reports) {
      createPageElements(reports);
   });
```

Listato 3.19: Richiesta GET necessaria al recupero dei referti

La pagina consente anche di visualizzare il criterio corrente di cancellazione dei referti e, in caso l'utente sia amministratore, sarà possibile cambiare questa impostazione. Se l'utente sceglie di cancellare i referti, l'elenco sottostante sarà aggiornato automaticamente e verrà mostrato un messaggio contente

informazioni importanti riguardo la periodica cancellazione dei referti.

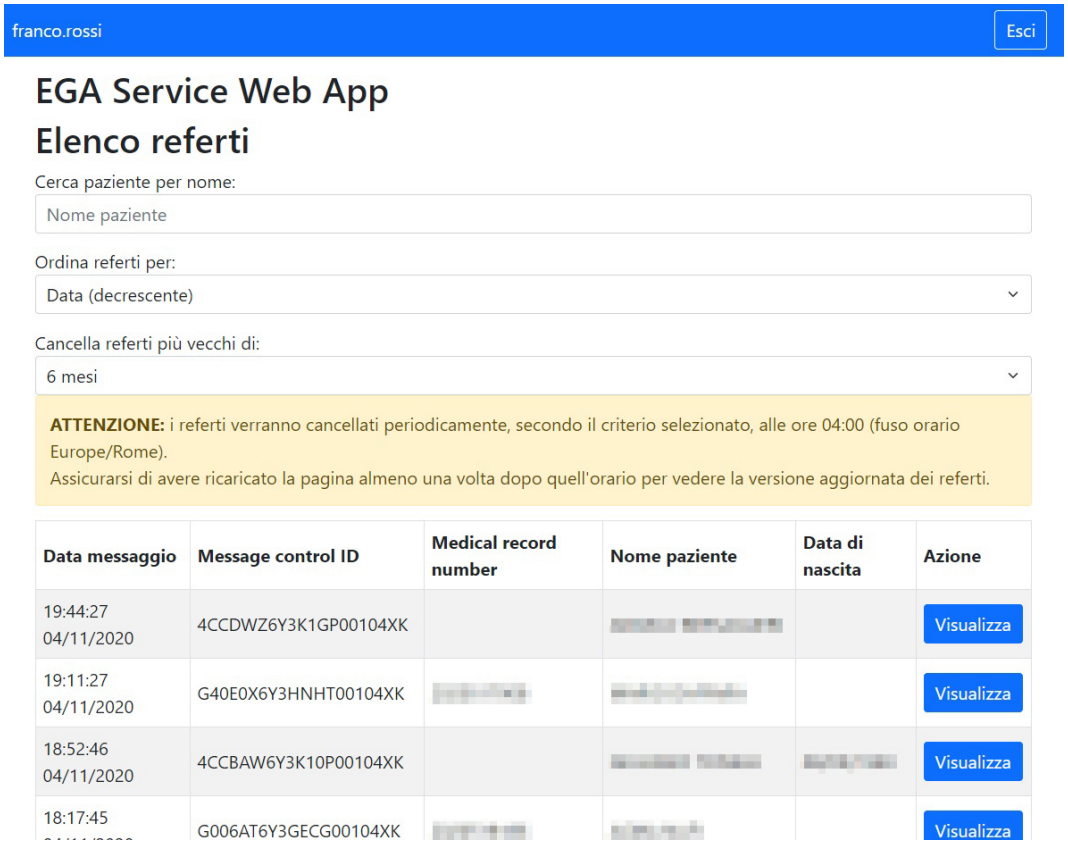


Figura 3.18: Messaggio di avvertenza a seguito della cancellazione dei referti

```
var valueSelected = $("#deleteTimeSelection").val();

$.ajax({
   url: reportsDeleteOptionPath,
   type: "PUT",
   data: {deleteOption: valueSelected},
   success: function(data) {
      $.get(reportsResourcePath)
         .done(function(newReports) {
            $("tbody *").remove();
            $("#changePageLabel").remove();
            $("#selectPage, #selectPage *").remove();
            $("#selectPage").val(1);
            $("#searchPatientByName").val("");
            sessionStorage.setItem(searchPatientByNameText, "");
```

```
            var numPages = getReportsPagesNumber(newReports.length);
            showReports(newReports, 1);
            downloadedReports = newReports;
        });
    }
});

setDeleteWarningMessage();
```

Listato 3.20: Richiesta AJAX per la cancellazione dei referti e conseguente gestione della pagina HTML

Pagina del risultato di un'analisi

Come precedentemente anticipato, cliccando sul bottone "Visualizza" associato a uno specifico referto, si verrà indirizzati verso la pagina che contiene tutte le informazioni dettagliate riguardo al risultato dell'emogasanalisi (figura 3.19). In questa pagina è possibile visualizzare tutti i parametri analizzati con le rispettive unità di misura, la data dell'esame, il modello e il codice del macchinario, completando così la raccolta delle informazioni più significative dal messaggio HL7 proprio come definito nei capitoli precedenti.

```
$.get(reportByIDResourcePath + localStorage.getItem(reportControlId))
    .done(function(report) {
        buildPage(report);
        showNavbar();
    });
```

Listato 3.21: Richiesta AJAX volta al recupero di un referto dato il suo codice identificativo

Cliccando sul pulsante "Torna ai referti" (figura 3.19) l'applicazione porterà l'utente alla pagina precedente, esattamente alla posizione del referto di cui aveva deciso di visualizzare i dettagli

Accesso non autorizzato e disconnessione dal sistema

Qualsiasi utente che dovesse tentare l'accesso non autorizzato alle risorse del sistema o che provi ad accedere a una pagina dell'applicazione senza aver completato correttamente la procedura di login, vedrà comparire a schermo un tradizionale messaggio HTTP 401 Unauthorized eventualmente integrato da apposito un messaggio. È importante sottolineare come non venga mostrato nessun altro componente della pagina o della risorsa richiesta, in quanto

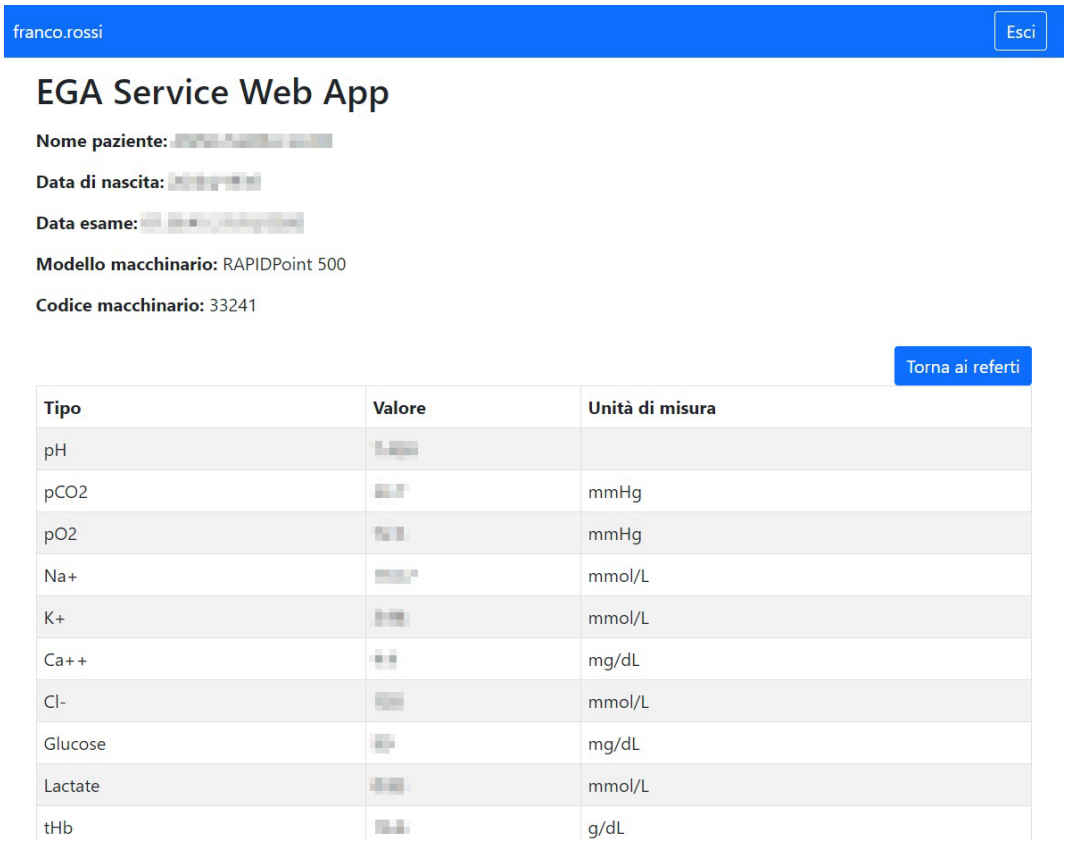


Figura 3.19: Schermata che mostra in dettaglio il referto dell'emogasanalisi di un paziente

non si vuole dare agli utenti non autorizzati nessuna indicazione riguardo alla struttura dell'applicazione.

Infine, l'utente può disconnettersi dal sistema in qualsiasi momento semplicemente cliccando sul bottone esci sempre presente in ogni pagina dell'applicazione web.

Capitolo 4

Deployment

In questo capitolo introduciamo i concetti di virtualizzazione e containerizzazione, analizzandone pregi e difetti. Segue poi la trattazione di Docker sottolineandone la pervasività e l'importanza che questo strumento ha acquisito oggigiorno. Infine, viene spiegato come effettuare il deployment dell'applicazione tramite Dockerfile e Docker Compose.

4.1 Virtualizzazione e containerizzazione

La virtualizzazione[1] è un processo che consente l'astrazione, ovvero la rappresentazione basata su software, delle risorse fisiche di un calcolatore e tramite il quale diventa possibile eseguire applicazioni su hardware virtuale. L'insieme delle componenti fisiche virtualizzate (CPU, RAM, disco fisso, scheda di rete) prende il nome di macchina virtuale. Questa astrazione è possibile grazie a uno strato software chiamato Hypervisor che si occupa anche di gestire l'accesso alle risorse hardware e garantire la protezione e l'isolamento tra le macchine. La virtualizzazione apre quindi alla possibilità di eseguire anche un intero sistema operativo su hardware virtuale e, di conseguenza, di poter installare ed eseguire qualsiasi applicazione al suo interno. Il sistema operativo in cui viene eseguita la macchina virtuale viene detto ospitante (host), la macchina virtuale è chiamata ospite (guest). La virtualizzazione porta con sé diversi pregi, tra i quali la possibilità di eseguire diversi sistemi operativi con diverse applicazioni sfruttando le stesse componenti fisiche, vantaggio che comporta spesso un risparmio notevole dei costi dell'hardware. Inoltre, un sistema operativo virtualizzato implica l'esecuzione di applicativi software in un ambiente isolato, protetto e monitorato, riducendo notevolmente i rischi ai quali potrebbe essere esposta la macchina fisica. Tuttavia, la virtualizzazione ha anche dei difetti in

[1] `https://it.wikipedia.org/wiki/Virtualizzazione`

quanto richiede l'esecuzione di un intero sistema operativo per isolare ed eseguire anche una singola applicazione, la quale continua comunque a dipendere dalla configurazione e dal software installato in quel sistema operativo.

La containerizzazione[2] è un altro tipo di virtualizzazione in cui viene virtualizzato lo spazio utente del sistema operativo ospitante. Ciò significa che i container realizzano uno spazio utente isolato, con i propri file e le proprie librerie, separando le applicazioni al loro interno dal resto del sistema operativo e fornendo loro le risorse necessarie alla corretta esecuzione. È evidente come la containerizzazione porti diversi vantaggi rispetto alla virtualizzazione tradizionale. Innanzitutto, la creazione di uno spazio utente dedicato non richiede la creazione di un'intera macchina virtuale. Inoltre, le applicazioni saranno dipendenti unicamente dal container e non dalla versione e/o dalla configurazione dell'intero sistema operativo. I container possono essere completamente isolati o condividere risorse, conferendo quindi un elevato grado di personalizzazione e gestione delle stesse.

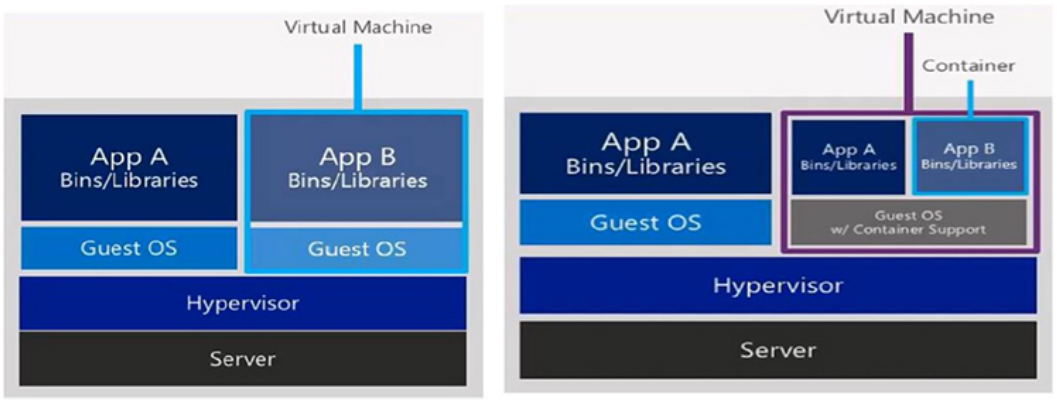


Figura 4.1: Principale differenza tra macchina virtuale e container

I container portano con sé una nuova opportunità, ovvero la possibilità di preconfigurare un'applicazione organizzando tutto ciò di cui necessita e rendendola pronta per essere dispiegata. In aggiunta, una volta costruito e personalizzato un container per una specifica applicazione, è possibile replicare quel container diverse volte, anche sulla stessa macchina virtuale o su macchine virtuali differenti.

[2] `https://en.wikipedia.org/wiki/Virtualization`

4.2 Docker

Docker[3] è una tecnologia che, sfruttando le funzionalità del kernel Linux, consente di automatizzare il dispiegamento (deployment) di applicazioni all'interno di container. È tra gli strumenti di containerizzazione più utilizzati al mondo perché permette di preparare facilmente, agevolmente, efficacemente, velocemente l'ambiente di esecuzione di un'applicazione, preconfigurando tutte le impostazioni e le risorse necessarie per il suo corretto funzionamento.

I container docker sono costruiti partendo dalla loro immagine, il container in esecuzione ne è un'istanza. Un'immagine consiste in uno o più file che rappresentano il contenuto dello spazio utente che si vuole creare, definendo e preimpostando tutto l'occorrente per la corretta esecuzione dell'applicazione. Le immagini possono essere salvate in locale o possono provenire da un ambiente (repository) esterno che mette a disposizione immagini già pronte. In particolare, Docker Hub è il più grande e diffuso repository per salvare, trovare e condividere immagini docker, tanto che, se non diversamente specificato, Docker scarica e utilizza le immagini necessarie direttamente da Docker Hub. La creazione di un immagine avviene tramite il Dockerfile, un file di testo che contiene regole, istruzioni (script) che saranno eseguite automaticamente e in sequenza per costruire l'immagine. Infine, Docker mette a disposizione Docker Compose, uno strumento che semplifica, velocizza e automatizza la creazione delle immagini nonché l'avvio e la gestione di container.

4.3 Deployment con Docker

In un sistema caratterizzato da una struttura a microservizi[4] diventano fondamentali gli aspetti di configurazione, deployment e manutenzione. Queste necessità unite ai vantaggi offerti dalla conteinerizzazione fanno sì che il deployment tramite Docker porti grandi benefici al sistema. Procediamo quindi con la spiegazione di come impostare correttamente il deployment tramite Docker.

4.3.1 Considerazioni preliminari

Il sistema sarà diviso in due container, il primo conterrà l'implementazione di EGA Server e dell'applicazione web (denominati d'ora in avanti EGA Service) mentre il secondo manderà in esecuzione un'istanza di MongoDB server. In questo modo sarà rispettata la filosofia di Docker la quale prevede

[3] `https://www.docker.com/`

[4] `https://en.wikipedia.org/wiki/Microservices`

che ogni container sia progettato per uno specifico microservizio e dedicato a un unico scopo. Inoltre, così facendo si avrà un sistema modulare, con dei container il più possibile indipendenti ma perfettamente funzionanti. I due container potranno comunicare grazie alla creazione di specifiche reti virtuali gestite direttamente da Docker.

La trattazione inizierà quindi dall'implementazione dei Dockerfile, la base necessaria per la creazione di un container.

4.3.2 Dockerfile

EGA Service

Per costruire uno spazio utente che sia in grado di mandare in esecuzione EGA Service è necessario partire dall'immagine di un container in cui sia stato installato il Java Development Kit[5], importare il file JAR[6] contenente tutte le risorse dell'applicazione e preimpostare l'esecuzione di un comando shell che ne consenta l'esecuzione.

```
FROM openjdk:11

WORKDIR /home/

COPY egaservice.jar /home/egaservice.jar

EXPOSE 8080:8080

EXPOSE 49200:49200

ENTRYPOINT ["java", "-jar", "egaservice.jar"]
```

Listato 4.1: Il Dockerfile relativo alla configurazione del container in cui verranno dispiegati EGA Server e l'applicazione web

Il listato 4.1 mostra tutti i comandi necessari alla corretta creazione del container desiderato. Il comando FROM permette di specificare l'immagine di partenza per la creazione del container, in questo caso, è necessaria un'immagine che contenga JDK versione 11, la stessa utilizzata per l'implementazione di EGA Server. Con WORKDIR viene definito il percorso nel quale verranno eseguiti i comandi specificati successivamente e grazie a COPY è possibile copiare dentro dentro il container, nel percorso precisato, tutti i file necessari al corretto funzionamento dell'applicazione.

[5] https://www.java.com/en/download/help/develop.html
[6] https://en.wikipedia.org/wiki/JAR_(file_format)

Per impostazione predefinita, quando Docker crea un container non espone nessuna porta nè al sistema in cui Docker è in esecuzione nè ai container non connessi alla sua stessa rete, questo renderebbe EGA Server irraggiungibile da qualsiasi applicazioni esterna. Tramite il comando EXPOSE è possibile creare una regola nel firewall che associa una porta (una porta TCP se non diversamente specificato) del container a una porta dell'host in cui è in esecuzione Docker. Il container di EGA Service esporrà due porte: la porta 8080 necessaria per la comunicazione con il server HTTP e la porta 49200 per rimanere in ascolto dei messaggi HL7 provenienti dal server di Siemens. In realtà, questo comando non esegue direttamente l'esposizione delle porte ma serve unicamente per indicare all'utente che avvierà la creazione del container quali porte dovranno essere esposte. Per attuare l'effettiva esposizione delle porte sarà necessario specificare l'opzione -p in fase di creazione del container tramite il comando docker run o usare l'apposita istruzione nel file docker-compose.

Infine, con ENTRYPOINT vengono specificati i comandi che saranno eseguiti all'avvio del container nel percorso precedentemente indicato con il comando WORKDIR. Si sarebbe potuto optare per il comando CMD, che offre funzionalità analoghe, tuttavia, si è scelto ENTRYPOINT in quanto non permette di essere sovrascritto da un eventuale comando personalizzato che l'utente può eventualmente specificare in fase di costruzione del container.

MongoDB

La creazione del container in cui verrà dispiegato il database dovrà partire dall'immagine contenente il servizio corrispondente.

```
FROM mongo:latest

WORKDIR /home/

ENTRYPOINT ["sh", "-c", "mongod --bind_ip mongodbserver && exit"]
```

Listato 4.2: Dockerfile necessario alla configurazione del container in cui sarà avviata l'istanza di MongoDB server

Il tag "latest" successivo al nome dell'immagine serve, in fase di creazione del container, per indicare a Docker la volontà di scaricare la versione più recente dell'immagine mongo.

Una volta avviato il container, sarà necessario mandare in esecuzione MongoDB server ed esporre il servizio nella stessa rete di cui fa parte EGA Service. Per ottemperare a questa necessità viene usato il comando ENTRYPOINT per avviare una shell ed eseguire i comandi specificati. Grazie al DNS fornito da Docker per le reti create dagli utenti, è possibile, sin dall'implementazione dei

Dockerfile, utilizzare il nome associato ai container al posto del loro indirizzo IP. In particolare, in questo caso il nome "mongodbserver" si riferisce al nome del servizio specificato con Docker Compose. Così facendo, non risulta necessaria l'associazione della porta 27017 di MongoDB a quella dell'host e si otterrà un ulteriore grado di sicurezza dalle applicazioni esterne.

4.3.3 Docker Compose

Docker Compose si avvale di un file in formato YAML[7] per stabilire i criteri di costruzione delle immagini di partenza dei container, specificare i servizi da mandare in esecuzione e configurare le reti necessarie alla corretta comunicazione tra i container.

```
version: "3.9" # optional since v1.27.0
services:
  egaservice:
    build: ./EgaService/
    ports:
      - 8080:8080
      - 49200:49200
    networks:
      - egaservice_bridge_network

  mongodbserver:
    build: ./MongoDB/
    networks:
      - egaservice_bridge_network

networks:
  egaservice_bridge_network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.21.0.0/16
          gateway: 172.21.0.1
```

Listato 4.3: Configurazione del file docker-compose.yml

Si procede dunque con l'analisi della configurazione mostrata nel listato 4.3.

Prima di tutto, è possibile specificare la versione di Docker Compose che si intende utilizzare all'interno del file, questo definirà quali parole chiave sarà possibile utilizzare all'interno del file stesso.

[7] https://en.wikipedia.org/wiki/YAML

La sezione “services” è di fondamentale importanza in quanto permette di definire e configurare tutti i servizi che faranno parte del processo di deployment. Naturalmente, in questo caso sono stati definiti due servizi: egaservice e mongodbserver.

L’istruzione “build” avvia la costruzione dell’immagine e ne cerca i file nel percorso specificato. L’istruzione “ports” consente l’effettiva associazione delle porte del container con le porte dell’host, mentre in “networks” viene specificato il nome della rete alla quale il container dovrà essere connesso.

L’ultima configurazione necessaria riguarda proprio la creazione della rete virtuale che permetterà la comunicazione tra i due container. Docker mette a disposizione diversi tipi di rete, ognuna con peculiarità diverse e predisposte per scopi differenti. In questo caso, si è scelto una rete ti tipo “bridge”, la scelta predefinita di Docker, in quanto è l’ideale per consentire una semplice ed efficace comunicazione tra spazi utenti separati mantenendo comunque un buon grado di isolamento tra i container e l’host su cui sono in esecuzione.

Nella configurazione sono stati anche specificati l’intervallo di indirizzi IP forniti alla rete e l’indirizzo del gateway associato. Inoltre, come accennato precedentemente, grazie al servizio di DNS messo a disposizione da Docker per le reti virtuali configurate dall’utente, la comunicazione tra container sarà possibile facendo semplicemente riferimento al nome del servizio definito tramite Docker Compose.

In conclusione, dopo aver implementato i Dockerfile e aver configurato correttamente docker-compose.yml, sarà possibile effettuare il deployment dell’intero sistema usando il solo comando “docker-compose up”.

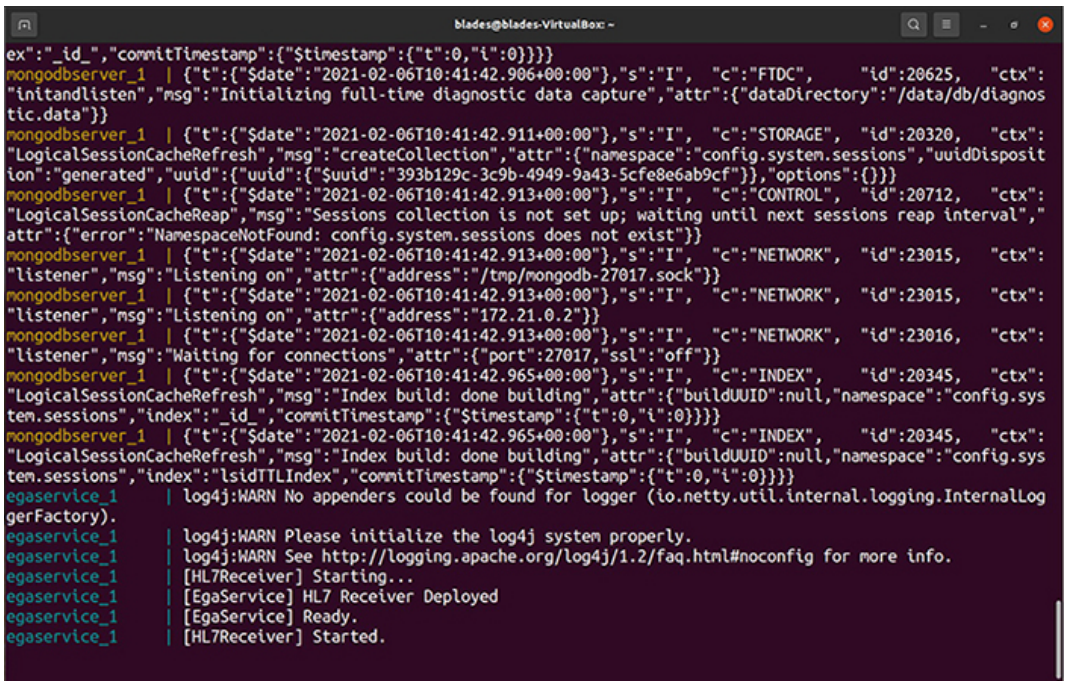


Figura 4.2: Deployment con Docker compose

Conclusioni

L'obiettivo della tesi prevedeva una sostanziale estensione e integrazione di un sistema prototipale che portasse un importante aiuto nell'attività del personale medico concernente la registrazione, consultazione e la gestione dei referti relativi all'emogasanalisi. Il sistema doveva anche rendere possibile l'integrazione dell'emogasanalisi in TraumaTracker e, parallelamente, rendere i referti disponibili anche ad altre eventuali applicazioni del sistema ospedaliero di riferimento.

Un primo punto fondamentale del lavoro svolto è stato l'aver affrontato la realizzazione del sistema adottando un approccio ingegneristico nonostante si partisse da un prototipo preesistente. L'avvalersi dell'analisi dei requisiti e di una corretta progettazione ha permesso non solo di avere una descrizione ben definita, non ambigua e fortemente strutturata di ciò che doveva essere realizzato, ma anche di revisionare, migliorare e ampliare correttamente le poche e spesso inadatte funzionalità che il prototipo metteva a disposizione. Il sistema è ora infatti perfettamente in linea con i requisiti e le specifiche emerse in fase di analisi, lo sviluppo conseguente la precisa progettazione ha conferito una struttura modulare che rende il sistema idoneo a potenziali cambiamenti e/o integrazioni future, eventualità molto frequenti e importanti in ambito sanitario. L'aggiunta di nuove API RESTful, congiunta al miglioramento di quelle già esistenti, ha permesso una perfetta interoperabilità con TraumaTracker e con tutte le applicazioni autorizzate che necessitino di reperire o gestire l'elenco dei referti.

L'applicazione web sviluppata consentirà al personale medico e, più in generale, a tutti gli utenti autorizzati, di usufruire facilmente e agevolmente di tutte le funzionalità offerte dal sistema. Il design responsivo permetterà all'applicazione di funzionare adeguatamente su diversi tipi di dispositivi come portatili, tablet e smartphone. Il codice e l'interfaccia grafica delle pagine web sono stati anche controllati rispettivamente con un validatore di markup e un validatore di accessibilità, aspetti fondamentali nell'ambito dello sviluppo web.

Sviluppi futuri

Naturalmente, visto lo stato prototipale del sistema, vi sono diverse migliorie che potrebbero essere introdotte e aspetti che potrebbero essere raffinati in sviluppi futuri.

Per quanto riguarda EGA Server, oltre agli aspetti già evidenziati sull'introduzione delle tecniche di sicurezza necessarie e il pieno adempimento al regolamento europeo sulla protezione dei dati personali, sarebbe opportuno prevedere l'autenticazione al sistema utilizzando il protocollo OAuth 2.0, in modo da rendere più adeguata ed efficace l'autorizzazione all'accesso da parte di TraumaTracker e di tutte le applicazioni che non sono nativamente predisposte al tradizionale login.

L'effettiva implementazione dello standard FHIR, basata sulle risorse definite in fase di analisi e con le modalità specificate nella progettazione, potenzierà ulteriormente il sistema in termini di interoperabilità e facilità di scambio di informazioni anche tra sistemi differenti.

Per quanto riguarda l'applicazione web, nonostante sia previsto che questa venga usata esclusivamente dal personale medico, sarebbe utile introdurre una tradizionale procedura di iscrizione degli utenti coadiuvata da un'interfaccia che permetta ai soli utenti amministratori di approvare personalmente le richieste di iscrizione al sistema. Si potrebbero inoltre introdurre le consuete tecniche di protezione di un sistema web come la cifratura preventiva delle credenziali dell'utente, il limite di tentativi di accesso consecutivi e fornire agli amministratori la possibilità di gestire, bloccare ed espellere gli utenti. Tutte queste aggiunte arricchirebbero il sistema di un livello di sicurezza aggiuntivo. Un'ulteriore dettaglio potrebbe prevedere la possibilità che ogni utente possa modificare in ogni momento i suoi dati personali o usufruire di una procedura per reimpostare la propria password personale. Per quanto riguarda la gestione dei referti, ulteriori funzionalità implementabili potrebbero essere la modifica e la cancellazione manuale di ogni singolo referto. Queste necessità non sono comparse nell'analisi dei requisiti, il sistema rimane tuttavia pienamente predisposto all'aggiunta di queste e altre funzionalità in caso dovessero essere richieste in futuro.

Infine, durante lo sviluppo del progetto e, in particolare, durante l'ispezione dei messaggi in formato HL7, è emerso che diversi campi contenenti informazioni di interesse sui pazienti (come il medical record number o la data di nascita) non contengono informazioni. Questo accade in quanto, come definito dalla documentazione di Siemens, questi campi sono facoltativi e, di conseguenza, viene spesso omessa la loro compilazione. Tuttavia, come evidenziato durante la trattazione, questi dati possono essere molto utili per l'identificazione del paziente o per ottenere informazioni utili sui trattamenti necessari. Sarebbe

quindi necessario informare il personale medico riguardo al completo e costante inserimento dei dati di interesse ogni qualvolta sia possibile.

Ringraziamenti

Desidero ringraziare vivamente il Professor Ricci e il Professor Croatti per avermi offerto questo importante progetto e per l'infinita disponibilità, pazienza e professionalità con cui mi hanno guidato durante lo svolgimento della tesi.
Vorrei inoltre dedicare un importante ringraziamento a tutti i Professori incontrati durante il corso di laurea che con passione, diligenza e costante impegno mi hanno trasmesso conoscenze e insegnamenti fondamentali per la mia crescita professionale e personale.
Desidero infine rivolgere un ringraziamento speciale a tutte le persone che, anche solo per un istante, mi sono state vicine e mi hanno supportato durante il percorso universitario.

Bibliografia

[1] Siemens Healthcare. Hospital system connection interface manual.

[2] S. Montagna, A. Croatti, A. Ricci, V. Agnoletti, and V. Albarello. Pervasive tracking for time-dependent acute patient flow: A case study in trauma management. In *2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS)*, pages 237–240, 2019.

[3] Tim Benson and Grahame Grieve. *Principles of Health Interoperability.* Springer, 2016.