



UNIVERSITÀ DEGLI STUDI DI BARI
“ALDO MORO”

DIPARTIMENTO DI INFORMATICA

Corso di Laurea in Informatica e Tecnologie per la Produzione del Software

TESI DI LAUREA IN SISTEMI MULTIMEDIALI

**Progettazione e sviluppo di un web service e della
portabilità dei dati per un sistema informativo
sanitario conforme allo standard FHIR**

RELATORE:

Chiar.mo Prof. Giovanni Dimauro

CORRELATORE:

Dott. Francesco Girardi

LAUREANDO:

Marzulli Simone

Anno Accademico 2015/2016

Sommario

Introduzione

1.1.	MOTIVAZIONI	5
1.2.	OBIETTIVI	6
1.3.	DESCRIZIONE DEI CAPITOLI	7

Cenni teorici

2.1.	PORTABILITÀ DEI DATI TRA SISTEMI SANITARI ELETTRONICI	8
2.2.	HEALTH LEVEL SEVEN (HL7)	9
2.3.	INTEROPERABILITÀ CON FHIR DI HL7	12
2.4.	ARCHITETTURA REST	13
2.5.	API RESTFUL PER L'IMPLEMENTAZIONE DI FHIR	17
2.6.	LE RISORSE IN FHIR.....	19
2.7.	STRUTTURA DI UNA RISORSA FHIR.....	21
2.8.	RISORSA IN FORMATO XML NEL DETTAGLIO	24

Stato dell'arte

3.1.	PORTABILITÀ SECONDO IL REGOLAMENTO EUROPEO	29
3.2.	SITUAZIONE PREGRESSA DEL SISTEMA	30

Progetto

4.1.	REFACTORING DEL MODULO DELLE API.....	33
4.2.	SVILUPPO DELLA RISORSA FAMILYMEMBERHISTORY	37
4.3.	SVILUPPO DELLA RISORSA ORGANIZATION	44
4.4.	SVILUPPO DELLA RISORSA OPERATIONOUTCOME	47
4.5.	IMPLEMENTAZIONE DI NUOVI SERVIZI REST.....	49

4.5.1.	SERVIZIO POST	50
4.5.2.	SERVIZIO PUT	54
4.5.3.	SERVIZIO DELETE	55
4.6.	PORTABILITÀ DELLE CARTELLE SANITARIE	57
4.6.1.	ESPORTAZIONE DI UNA CARTELLA.....	57
4.6.2.	IMPORTAZIONE DI UNA CARTELLA	62
4.7.	ULTERIORI MIGLIORAMENTI AL WEB SERVICE FHIR	65
4.7.1.	REALIZZAZIONE DEL TIER VIEW	66
4.7.2.	CONTROLLO DEL FORMATO DELLA RISORSA	68
4.7.3.	OPERAZIONI AGGIUNTIVE SULLA RISORSA	68

Conclusioni

5.1.	RISULTATI	71
5.1.1.	PUNTO DI VISTA DEGLI STAKEHOLDER	71
5.1.2.	PUNTO DI VISTA DELLO SVILUPPATORE.....	72
5.1.3.	PUNTO DI VISTA DI UN SISTEMA SANITARIO ESTERNO	73
5.2.	CONSIDERAZIONI FINALI	74
5.3.	SVILUPPI FUTURI.....	76

Riferimenti

6.1.	BIBLIOGRAFIA	78
6.2.	SITOGRAFIA	78

1. Introduzione

Per stare al passo con la sempre maggiore diffusione della tecnologia e migliorare la qualità dell'assistenza ospedaliera, il mondo della sanità ha avuto la necessità di aggiornare strumenti e abitudini introducendo il concetto di **Health Informatics** o informatica medica; quest'ultima è nata come estensione dell'informatica tradizionale e col tempo si è evoluta facendo da ponte tra le due discipline. Questo lavoro di Tesi è incentrato su una descrizione delle odierne tecnologie impiegate per la creazione di sistemi informativi per l'ambito sanitario, oltre che sull'implementazione di alcune tecniche per l'interoperabilità ad essi correlate e basate su appositi standard internazionali. Nei capitoli successivi sarà descritto, con particolari riferimenti ai sistemi sanitari attualmente in uso, l'utilizzo di standard atti alla fruizione di attributi come l'interoperabilità tra registri elettronici ospedalieri e la portabilità dei dati degli utenti in esso memorizzati.

La tecnologia informatica applicata al campo della medicina, nota anche come *e-Health*, è una disciplina sostanzialmente giovane, ma in pochissimi anni è divenuta una tematica fondamentale per organismi sanitari locali e nazionali. Al giorno d'oggi, non tutte le istituzioni sanitarie sono provviste di un sistema informatico per l'archiviazione dei dati e, a causa dell'onnipresente **errore umano**, questo potrebbe costituire un rischio per la salute del paziente qualora il medico curante non dovesse disporre di tutte le informazioni necessarie. Inoltre, ricorrere ad un **framework unificato** per la creazione di software medico da una parte ridurrebbe i costi in tempo e risorse di sviluppo e manutenzione di tali sistemi, dall'altra abbasserebbe la

probabilità di propagazione di errori nella condivisione dei dati tra registri diversi: la vitale importanza del rispetto di standard internazionali è un'ipotesi centrale di questa dissertazione.

1.1. Motivazioni

Le considerazioni appena discusse hanno condotto alla scelta della tematica del lavoro di sviluppo, cioè l'adeguamento di una cartella clinica elettronica ad uno standard di interoperabilità. Queste cartelle elettroniche sanitarie - uno dei motivi dominanti dell'informatica medica - sono chiamate in genere **Electronic Health Record** o con l'acronimo EHR e il loro scopo è supportare il paziente durante il suo periodo di assistenza.

Nel lavoro di Tesi sono state sviluppate alcune componenti per un software sanitario denominato **Registro Elettronico Sanitario Personale** o RESP, ideato dall'Università degli Studi di Bari. Lo scopo di questo registro elettronico, ovvero un EHR, è quello di immagazzinare tutte le più importanti informazioni di carattere clinico di un paziente e di renderle disponibili per ogni dispositivo connesso ad Internet. Il RESP è pensato per essere utilizzato sia dal punto di vista del **paziente** sia del **care provider** (operatore sanitario): l'uso di questa cartella clinica elettronica riduce notevolmente l'impiego di strumentazione analogica, fornisce molte funzionalità per l'assistenza medica e migliora la comunicazione tra paziente e medico curante. Inoltre si può intendere il RESP non solo come un semplice EHR, ma anche come un software PHR (**Personal Health Record**), che a differenza del primo viene gestito anche dal paziente. Tra le varie funzionalità che il RESP mette a disposizione si può ricordare l'implementazione di

moduli per la gestione delle diagnosi, anamnesi, indagini diagnostiche e la presenza di un taccuino digitale per la segnalazione di condizioni di malessere o richiesta di assistenza medica da parte di un paziente.

Posta l'innegabile utilità delle funzionalità di un registro elettronico come il RESP, non va in nessun caso tralasciato l'aspetto di *interoperabilità* e condivisione di informazioni del registro con enti sanitari esterni, come già stabilito in precedenza.

1.2. Obiettivi

Lo scopo del lavoro di tesi era, in primo luogo, quello di realizzare una componente più efficiente ed ingegnerizzata per la gestione delle **API** (*Application Programming Interface*). Successivamente, si sarebbe passati ad aggiungere alcune delle risorse per l'interoperabilità del registro sanitario, in modo aderente allo standard FHIR dell'organismo HL7. Lo sviluppo delle API avrebbe permesso di stabilire una buona base su cui poter svolgere il lavoro più caratteristico di questa Tesi, ovvero la possibilità di esportare ed importare i dati clinici del paziente in RESP. La portabilità si sarebbe ottenuta grazie alla standardizzazione dei dati nel registro elettronico oltre che alla manipolazione degli stessi con il gestore delle API sviluppato nelle fasi iniziali. L'obiettivo finale era quello di permettere al paziente registrato di ottenere un file contenente tutte le informazioni standardizzate del registro e garantire una copia dei dati personali in maniera intuitiva e automatica.

1.3. Descrizione dei capitoli

Sarà esposta di seguito una breve descrizione dei capitoli.

- **Capitolo 2** • *Cenni teorici*, fornisce una base teorica per comprendere il lavoro di Tesi illustrando l'organismo per lo standard sanitario internazionale HL7 e lo sviluppo di sistemi software in ambito sanitario.
- **Capitolo 3** • *Stato dell'arte*, offre una breve panoramica delle attuali leggi in vigore nell'Unione Europea riguardanti la portabilità dei dati nei sistemi informatici e descrive lo stadio di sviluppo del progetto precedente al lavoro di Tesi.
- **Capitolo 4** • *Progetto*, descrive il progetto nelle sue varie fasi, illustrando nel dettaglio le tecnologie e le metodologie utilizzate per arrivare al risultato finale.
- **Capitolo 5** • *Conclusioni*, vengono riassunti i risultati ottenuti e le migliorie apportate attraverso la progettazione del sistema, ripropone le considerazioni discusse nei precedenti capitoli e indica brevemente i possibili sviluppi futuri da continuare nello studio effettuato.

2. Cenni teorici

2.1. Portabilità dei dati tra sistemi sanitari elettronici

Con l'avvento delle nuove tecnologie, sempre più influenti anche nella vita quotidiana, il settore medico ha cercato con gli anni di adeguarsi e di sfruttare tali fattori a suo vantaggio. Tra le tante conseguenze che il progresso tecnologico nel settore medico ha portato, purtroppo vanno citati anche la maggiore dispersione nella gestione delle informazioni e nelle regole di business, lo sviluppo di software eterogeneo quasi mai standardizzato e la ridondanza di dati tra le varie strutture sanitarie.

Il tipico caso d'uso che si può illustrare è quello di un ospedale che utilizza un applicativo software (EHR) per memorizzare e strutturare i dati dei pazienti ricoverati. Quando il paziente viene dimesso, tali dati rimangono molto spesso nelle banche dati dell'ospedale. Successivamente il medico curante che segue quel paziente si troverà ad importare manualmente questi dati basandosi sul referto rilasciato dall'ospedale, all'interno di un diverso EHR e con un diverso database. Similmente, ciò avviene a ruoli invertiti quando un ospedale necessita dei dati privati del paziente, che magari sono già disponibili all'interno del software del medico curante.

Risulta ovvio come questa attività manuale di condivisione dei dati porta spesso ad uno spreco di tempo e risorse sia per il paziente che per i diversi *care provider*. Al fine di minimizzare tali problematiche di comunicazione tra sistemi sanitari elettronici, c'è bisogno di definire un insieme di regole che

consenta ai vari applicativi software di condividere dati e creare meccanismi di **portabilità**.

Si può definire la portabilità dei dati come la possibilità per gli utenti di riutilizzare i propri dati tra diverse applicazioni che interagiscono tra loro; in altre parole, esso è un concetto stabilito per proteggere l'utente dall'archiviazione dei suoi dati in "silos" o "giardini recintati" incompatibili tra loro, cioè piattaforme chiuse e soggette a tecniche di *lock-in*.

Si può subito notare come questa definizione possa essere applicata al caso d'uso illustrato poc'anzi. Per consentire la portabilità dei dati tra due *data controller*, bisogna quindi stabilire degli standard tecnici che promuovano una forma di interoperabilità.

2.2. Health Level Seven (HL7)

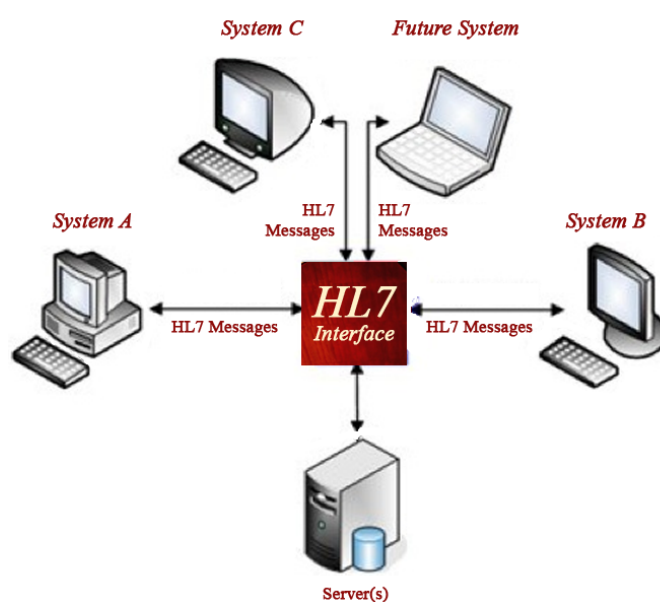
L'organismo internazionale che si occupa da tempo di affrontare questa tematica è l'HL7 o **Health Level-7**. Questa organizzazione mette a disposizione degli standard *de facto*, che consentono di regolare il trasferimento di dati clinici ed amministrativi tra i vari applicativi software usati in campo sanitario, definendo il formato ed il contenuto dei messaggi condivisi. Health Level Seven International, fondata nel 1987, è una associazione non profit ed è l'autorità globale sugli standard per l'interoperabilità e l'intercomunicazione nel campo dell'informatica medica. Comprende membri di più di 55 paesi (inclusa l'Italia con HL7 Italia) e si basa sul modello OSI; nello specifico, rispetto al modello concettuale esso si colloca sul livello 7 (livello Applicazione). Gli standard di HL7 sono prodotti da HL7 International e vengono adottati da altri organismi internazionali

come l'American National Standards Institute (ANSI) e l'International Organization for Standardization (ISO).

I principali obiettivi dell'organismo HL7 sono:

- Facilitare l'interazione fra sistemi sanitari, quasi sempre proprietari ed eterogenei, fornendo delle *interfacce* che favoriscano la condivisione e la comunicazione di informazioni;
- Rendere disponibile un preciso formato ed un protocollo per lo scambio dei dati;
- Standardizzare l'intero complesso di un sistema sanitario dal punto di vista comunicativo.

Il funzionamento degli standard HL7 è strutturato in maniera tale da tradurre i dati dal formato interno dei vari EHR in un formato standardizzato dall'organismo. Questi dati, dopo essere stati scambiati, verranno a loro volta ritradotti nei formati interni comprensibili al sistema EHR specifico.



(Figura 2.1: Scambio tra diversi sistemi sanitari utilizzando l'interfaccia HL7)

Dalla sua nascita, HL7, ha rilasciato differenti standard che sono poi stati adottati ed implementati dal panorama dei sistemi software sanitari. Tra questi possiamo ricordare:

- HL7 Messaging Standard versioni 2 e 3 (HL7 v. 2.x e v.3) – una specifica per l’interoperabilità di transazioni mediche e sanitarie;
- Clinical Document Architecture (CDA) – un modello per lo scambio di documenti clinici, basata sulla terza versione di HL7;
- Structured Product Labelling (SPL) – uno standard per le informazioni riguardanti un certo farmaco;
- Fast Healthcare Interoperability Resources (FHIR) – uno standard per l’interoperabilità tra sistemi sanitari che si scambiano risorse.

Nella lista sopra descritta si può notare l’ultimo standard, ovvero **FHIR**, che verrà ampiamente approfondito nei capitoli successivi e costituisce uno degli argomenti salienti di questo lavoro di Tesi.

Lo standard FHIR nasce inizialmente come *ramo* dello standard CDA, che ha in seguito migliorato e portato a conclusione. Viene utilizzato principalmente per l’identificazione di risorse sanitarie come, ad esempio, pazienti, medici, strutture o terapie. Questo standard prevede che le risorse individuate nei vari EHR vengano codificate in formato XML (eXtensible Markup Language) oppure JSON (JavaScript Object Notation), poiché entrambi i linguaggi sono fortemente utilizzati dalle tecnologie web per la strutturazione di dati, anche di grosse dimensioni. FHIR è stato rilasciato inizialmente nel 2014 con la prima versione DSTU (Draft Standard for Trial Use), quindi come versione di prova, mentre l’ultimo aggiornamento (in base

alla data di scrittura di questa Tesi) risale ad agosto 2016 con la versione *v1.0.2*.

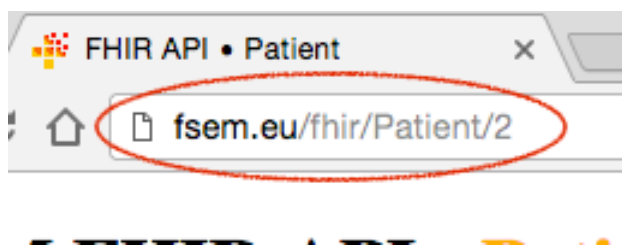
2.3. Interoperabilità con FHIR di HL7

Lo standard FHIR (pronunciato /'faɪə/, “fire” in lingua inglese), è stato progettato per lo scambio di risorse e per l’interoperabilità tra sistemi sanitari elettronici. Nello standard viene illustrato come codificare una risorsa in precisi formati e mette a disposizione un insieme di risorse tipiche che identificano varie informazioni presenti in un sistema sanitario.

Una delle peculiarità di FHIR, rispetto alle vecchie versioni di HL7, è che, oltre ad avere continui aggiornamenti da parte del team di sviluppo, utilizza delle moderne tecnologie web per l’implementazione di **API** (Application Programming Interface). Fra le varie tecnologie impiegate si può citare l’architettura software REST (Representational state transfer) basata sul protocollo web HTTP, l’utilizzo di linguaggi di *markup* (come HTML) per la visualizzazione delle risorse e la sopracitata possibilità di scelta del linguaggio per la codifica delle risorse: XML o JSON.

Proprio grazie all’uso di queste tecnologie, uno dei principali obiettivi dello standard è quello di rendere semplice ed intuitivo il suo impiego in sistemi sanitari *legacy*; si ha di conseguenza una maniera più semplice per l’interfacciamento di applicazioni di terze parti con i sistemi sanitari che adottano lo standard FHIR. Mediante l’uso dello standard, un sistema sanitario permette a sviluppatori esterni di creare applicazioni per computer, smartphone e tablet che interagiscono con esso.

Diversamente da altri protocolli, FHIR non si focalizza su documenti sanitari eterogenei, ma cerca di offrire i dati ospedalieri presi singolarmente, fornendoli come risorse codificate e disponibili sotto forma di servizi web. Un esempio potrebbe essere l'identificazione della risorsa *paziente*: un applicativo software che si interfaccia con il sistema sanitario, potrà fare richiesta delle sole informazioni sul paziente, visitando un URL come quello illustrato in figura 2.2.



(Figura 2.2: esempio di URL per richiedere la risorsa di un paziente)

Le risorse dello standard FHIR sono rese disponibili mediante l'uso di un'architettura REST, che mette a disposizione dell'EHR un insieme di operazioni utilizzabili dai client per la manipolazione delle stesse.

2.4. Architettura REST

Il termine REST sta per *REpresentational State Transfer* ed è un tipo di architettura software che supporta l'interoperabilità tra sistemi informatici nel World Wide Web. REST fu inizialmente concepito da Roy Fielding (uno degli autori del protocollo **HTTP**). Un web service che mette a disposizione un servizio basato su REST si può definire RESTful. L'architettura REST funziona in maniera tale che i client, che richiedono al server delle **risorse**

web identificate da uno specifico URI (Uniform Resource Identifier), effettuino le richieste utilizzando delle operazioni *stateless* messe a disposizione dal servizio web. Per protocollo stateless si intende una sequenza di richieste, da parte dei client, che rappresentano transazioni indipendenti non correlate a richieste precedentemente effettuate; in questa maniera ogni richiesta fatta al server verrà seguita da un'unica risposta.

Le richieste mediante REST sono effettuate utilizzando il protocollo HTTP, evitando l'uso di ulteriori livelli per la trasmissione dei dati, come per il protocollo SOAP (Simple Object Access Protocol) o con la gestione della sessione tramite *cookie*.

Quando un client effettua una richiesta verso il servizio RESTful, la risposta sarà codificata in XML, HTML o JSON.

L'adozione del protocollo HTTP, da parte di REST, permette di usufruire di alcune delle operazioni (definite anche **verbs**) quali GET, POST, PUT e DELETE. I verbs HTTP consentono di manipolare le risorse del servizio web, che possono essere già presenti sul server oppure generate dinamicamente in base alla richiesta; di solito le risorse corrispondono a dei file o degli eseguibili residenti sul server.

Le operazioni del protocollo HTTP, come spiegato poc'anzi, si possono applicare a risorse identificate da URI e sono di seguito illustrate.

L'operazione *GET* consente di fare richiesta di una risorsa al server e la risposta sarà il dato richiesto. Il comportamento di questa operazione non sarà altro che quello di *data retrieval*. Il verb *POST* permette ai client di creare una nuova risorsa sul server. Il dato inviato dal client può essere il

corpo codificato della risorsa che si desidera creare. Il metodo *PUT* è utilizzato per l'aggiornamento e la creazione di una risorsa specificando un preciso URI: così facendo, se la risorsa identificata con l'URI è già esistente sul server, allora questa viene modificata, altrimenti sarà creata con i valori presenti nello stesso URI. Intuitivamente si potrebbe confondere PUT con l'operazione di update di un software CRUD, ma così non è; la differenza tra POST e PUT sta nel fatto che, con il primo metodo, non è specificata la locazione sul server dove la risorsa sarà creata, mentre con il secondo metodo la locazione viene descritta nell'URI. Concludendo il metodo *DELETE* permette la cancellazione di una risorsa dal server.

Il protocollo HTTP mette a disposizione altre operazioni, oltre quelle appena specificate, (consente persino di crearne altre in base alle necessità dello sviluppatore), tra cui:

- HEAD, simile alla GET ma usata per ottenere esclusivamente i metadati di una risorsa;
- OPTIONS, permette di restituire il numero di operazioni che il web service dispone per un particolare URL;
- CONNECT, che converte la richiesta di connessione ad un tunnel TCP/IP trasparente, genericamente usato per rendere crittografata la connessione mediante SSL quando il server non dispone di un canale di comunicazione sicuro.

In base alle specifiche di REST, si può comprendere come questa architettura miri ad avere delle buone prestazioni, affidabilità e semplicità di scalabilità del sistema, anche quando questo è già operativo. Affinché un web service

possa definirsi RESTful, esso necessita di seguire alcuni vincoli architetturali discussi nel paragrafo successivo.

Il meccanismo di funzionamento deve essere *client-server* e viene incoraggiato il ricorso all'**astrazione**. Il vantaggio di questa divisione permette al client di non preoccuparsi dei meccanismi di salvataggio delle informazioni sul server, oltre al fatto che si ha una maggiore portabilità del codice. La comunicazione client-server deve essere *stateless*, pertanto non deve essere memorizzata sul server alcuna informazione sul contesto del client; esso è l'unico dei due attori che dovrà occuparsi di gestire lo stato della sessione. Il web service può inoltre rendere alcune risposte *cacheable*, quindi, quando possibile, il client potrà memorizzare le risposte del server, in modo tale da eliminare parzialmente la comunicazione client-server, migliorando le performance. Un esempio di utilizzo di cache nei client Web si può osservare anche con i moderni browser. Sempre per la logica di trasparenza del funzionamento del server, ad un client non è dato sapere se il server è di basso livello o intermedio, nel qual caso il sistema si definisce *layered* (stratificato). Con la stratificazione aumentano le performance nella connessione tra i server intermedi, le politiche di sicurezza e inoltre migliora la scalabilità. L'ultimo vincolo rappresenta una interfaccia di comunicazione omogenea (*uniform interface*) tra client e server. Questo vincolo è di fondamentale importanza per l'implementazione di servizi REST, dato che **disaccoppia** il client e il server favorendo lo sviluppo dei due in maniera indipendente, nel rispetto del principio ingegneristico della **separazione degli interessi**. Un altro vincolo (opzionale) secondo le specifiche REST è il *code on demand*, che permette una parziale personalizzazione del client da

parte del server inviando del codice eseguibile come Applet Java o linguaggi di scripting client side tipo JavaScript.

2.5. API RESTful per l'implementazione di FHIR

Attraverso una architettura REST, lo standard FHIR permette la manipolazione delle risorse sanitarie con i metodi HTTP visti nel paragrafo precedente.

Per l'implementazione di un web service aderente allo standard di HL7 ci sono alcune accortezze da seguire, al fine di tutelare la privacy e di garantire un certo rigore nello scambio di informazioni sanitarie.

Lo standard suggerisce che lo scambio dei dati sanitari avvenga su connessione SSL (Secure Sockets Layer), quindi in maniera sicura. La comunicazione *dovrebbe* pertanto essere crittografata.

Le operazioni dello standard FHIR applicabili alle risorse, definite come interazioni (*interactions*), sono suddivise in 3 gruppi:

Operazioni a livello di istanza (*Instance Level Interactions*):

- **READ** permette di leggere lo stato corrente di una risorsa (corrisponde all'operazione GET di REST);
- **VREAD** richiede una risorsa specificandone la versione;
- **UPDATE** consente di aggiornare una risorsa specificandone l'id, nel caso in cui la risorsa non dovesse esistere allora viene creata;
- **DELETE** rimuove la risorsa;
- **HISTORY** restituisce tutti i cambiamenti apportati alla risorsa richiesta nel corso del tempo.

Ci sono poi le operazioni a livello di tipo (*Type Level Interactions*):

- **CREATE** consente di creare una nuova risorsa con un id assegnato dal server;
- **SEARCH** consente la ricerca del tipo della risorsa specificando determinati filtri di ricerca;
- **HISTORY** restituisce i cambiamenti di un tipo di risorsa.

Infine sono descritte le *Whole System Interactions*:

- **CONFORMANCE** restituisce uno schema di tutte le operazioni applicabili alle risorse supportate dal web service implementato;
- **BATCH/TRANSACTION** aggiorna, crea o elimina un insieme di risorse effettuando una singola richiesta;
- **HISTORY** ritorna gli aggiornamenti di tutte le risorse presenti sul server;
- **SEARCH** ricerca tra tutti i tipi di risorse presenti sul server specificando i criteri di ricerca.

Le operazioni da applicare alle risorse sono specificate seguendo la sintassi:

```
VERB [base] / [type] / [id] { ? _format=[mime-type] }
```

Dove VERB sta per l'operazione HTTP da utilizzare sulla risorsa, mentre tutto ciò che è racchiuso tra parentesi quadre rappresenta dei campi obbligatori, le parentesi graffe rappresentano campi opzionali. Analizzando la stringa appena illustrata si identifica:

- **base** che rappresenta la root del web service, per esempio `http://fsem.eu/fhir/`;

- `type` indica il nome della risorsa da richiedere, esempio: *Patient*, *Practitioner*, *Organization*, etc;
- `id` specifica l'id logico della risorsa da richiedere.

La stringa racchiusa tra parentesi graffe, invece, rappresenta una variabile che indica il formato con cui richiedere la risorsa al web service (se il sistema supporta diversi formati). Un esempio di specifica del formato può essere descritta nel seguente URL:

```
http://fsem.eu/fhir/Patient/2?_format=json
```

Verrà richiesto al web service la risorsa di *tipo* paziente, con *id* numero 2 e codificata in *formato* JSON.

Fino ad ora, in riferimento all'architettura REST, si è parlato di risorse come la più piccola entità utile al fine di scambiare informazioni tra client e server nello standard FHIR. Nel paragrafo successivo verrà spiegato in cosa consiste effettivamente una risorsa messa a disposizione da un EHR.

2.6. Le risorse in FHIR

Le risorse, componenti fondamentali dello standard FHIR, identificano sia entità sia processi presenti nel settore sanitario e ognuna di esse presenta degli elementi comuni:

- Un URL che permette di accedere univocamente alla risorsa;

- Metadati per fornire informazioni sul contesto tecnico della risorsa nel sistema;
- Delle estensioni, appositamente sviluppate per rendere disponibili informazioni aggiuntive sulla risorsa, non previste inizialmente dallo standard ma utilizzate nello specifico sistema;
- Un documento codificato in formato XHTML, comprensibile da esseri umani;
- Una parte che rappresenta la struttura dati con le informazioni rilevanti della risorsa.

Lo standard FHIR possiede attualmente circa 90 risorse e molte altre verranno rilasciate nei prossimi anni. Queste risorse sono suddivise in base al loro significato semantico nell'ambito sanitario, nei raggruppamenti descritti qui di seguito.

Il gruppo delle risorse cliniche (*Clinical*) contiene quelle che identificano perlopiù le informazioni presenti nelle cartelle cliniche dei pazienti; si possono trovare dati su anamnesi, allergie, intolleranze, farmaci assunti o cure mediche. Le risorse identificative o amministrative (*Identification*) riguardano le entità coinvolte in un sistema sanitario, tra cui pazienti, medici, organizzazioni, ospedali o strumentazione medica. Insieme a queste entità, si possono non solo rappresentare esseri umani, ma anche animali, poiché FHIR mira ad avere un'utenza persino nel campo veterinario. Nelle risorse identificative sono presenti, inoltre, tracciabilità sui luoghi in cui un servizio o un processo medico vengono eseguiti. Ci sono le risorse di *Workflow* per la gestione dei flussi di processo, che modellano i vari processi medici offerti al paziente: ad esempio, esistono risorse che descrivono gli appuntamenti del

paziente con un care-provider, di cui conservano informazioni su date e tempi, oltre che le richieste di particolari servizi e prenotazioni. Le risorse di infrastruttura (*Infrastructure*) servono per il tracciamento dei cambiamenti e la gestione delle informazioni. Ci sono, in aggiunta, risorse che modellano informazioni addizionali al sistema, come questionari con risposte da sottoporre ai pazienti. Le risorse di conformità (*Conformance*), invece, vengono utilizzate per gestire le specifiche e le capacità che un sistema FHIR deve possedere. Per concludere, le risorse finanziarie (*Financial*), ancora in fase di sviluppo, riguardano i pagamenti delle prestazioni mediche e le notifiche di operazioni finanziarie.

2.7. Struttura di una risorsa FHIR

Analizzando nel dettaglio la struttura che ogni risorsa dello standard FHIR possiede, si possono individuare in figura 2.3 le parti fondamentali, già discusse in precedenza: metadati, un testo narrativo, estensioni e una struttura dati ben definita.



(Figura 2.3: Struttura di una risorsa FHIR)

I **metadati** vengono utilizzati per immagazzinare diverse informazioni sulla risorsa in esame, inserite in campi che:

- identificano univocamente la versione della risorsa e necessitano di essere cambiati quando la risorsa è soggetta a variazioni nel tempo;
- specificano quando la versione della risorsa è stata cambiata;
- rappresentano il grado di privacy e sicurezza a cui la risorsa è sottoposta; un esempio potrebbe essere la risorsa *Patient* che contiene informazioni sensibili sulle malattie del paziente, come HIV/AIDS.

I metadati non sono obbligatori, quindi nell'implementazione di una risorsa possono essere omessi.

Il **testo narrativo** deve contenere un documento scritto in linguaggio XHTML. Questa scelta è scaturita dal fatto che la risorsa potrebbe essere visualizzata da un browser web e di conseguenza necessita di essere facilmente letta da un essere umano; oltre al fatto che può risultare difficoltoso, per un utente non esperto, leggere la risorsa codificata direttamente in XML. I dati che il testo narrativo dovrebbe contenere sono scelti a discrezione dello sviluppatore, dovrebbero essere mostrati i più rappresentativi della risorsa. Per esempio, per la risorsa “Paziente” può essere utile conoscere subito i dati anagrafici. Bisogna prestare attenzione nella progettazione della parte narrativa, poiché nei sistemi orientati al trading la condivisione di alcune informazioni potrebbe risultare un rischio per la sicurezza dei dati clinici.

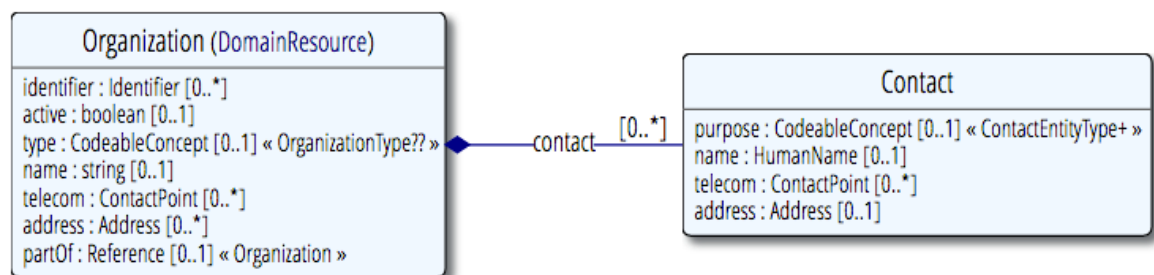
Name	Patricia Wilk
Birth Date	1966-06-08
Contact	339xxxxxxx
City	Bari (BA)
Address	M. Troisi 28
Marital Status	Nessuno

(Figura 2.4: esempio di dati visualizzati nel registro elettronico sanitario personale)

La rappresentazione della risorsa paziente, illustrata in figura 2.4, può essere facilmente costruita in XHTML utilizzando dei semplici *tag* per disegnare tabelle, ciò non toglie che la parte narrativa di una risorsa possa semplicemente essere costituita da un testo libero (ad esempio utilizzando il tag `<p>`) evitando, così, la complessa struttura del linguaggio di markup.

Tra le parti che compongono una risorsa si possono trovare anche le **estensioni**, molto utili nel caso in cui lo standard FHIR non abbia previsto delle informazioni che, invece, un certo EHR necessita di allegare ad una risorsa. Il problema nasce dal fatto che le risorse di FHIR presentano degli elementi comuni a vari sistemi sanitari mondiali, cercando di coprire le varie differenze di giurisdizione e dominio; soddisfare i requisiti specifici di ogni nazione porterebbe ad un accumulo di informazioni superflue. Se un sistema sanitario necessita l'aggiunta di un particolare dato, non previsto nella specifica FHIR, lo può fare mediante l'aggiunta di estensioni. Un esempio di estensione in una risorsa può essere l'inserimento del codice fiscale italiano o in maniera simile il Social Security Number (SSN) per gli Stati Uniti d'America. Infine, la componente più importante di una risorsa è la

struttura dati definita. La struttura dati possiede tutte le informazioni di una risorsa, codificate nel linguaggio scelto dallo sviluppatore, che, come già detto, può essere XML o JSON. Lo standard FHIR, inoltre, è basato su modello ad oggetti, si potrebbe rappresentare la struttura dati anche mediante diagramma UML come in figura 2.5.



(Figura 2.5: Diagramma UML della risorsa Organization)

2.8. Risorsa in formato XML nel dettaglio

In questo lavoro di Tesi si è deciso di codificare le risorse dello standard FHIR in formato XML. L'utilizzo di questo formato implica che il sistema, ad ogni richiesta da parte di un client, fornisca la risorsa codificata in linguaggio XML; analogamente, per la creazione di nuove risorse mediante EHR esterni verrà riconosciuto lo stesso linguaggio. Si entrerà nel dettaglio di come è organizzato il *web service* nei successivi capitoli, per il momento ci si limita, in questo paragrafo, a descrivere la codifica di una risorsa.

Tralasciando i dettagli, XML può definirsi un metalinguaggio; è un formato aperto, sviluppato dal consorzio W3C (World Wide Web Consortium) e, a differenza del linguaggio HTML, consente di creare nuovi linguaggi per la

descrizione di documenti ben strutturati. Lo si può intendere, quindi, come un insieme di regole sintattiche atte a definire il proprio linguaggio di markup. Il suo impiego si può persino trovare in alcuni Data Base Management System (DBMS).

```
<Organization xmlns="http://hl7.org/fhir">
  <!-- from Resource: id, meta, implicitRules, and language -->
  <!-- from DomainResource: text, contained, extension, and modifierExtension -->
  <identifier><!-- 📄 0..* Identifier Identifies this organization across multiple systems --></
  identifier>
  <active value="[boolean]"/><!-- 0..1 Whether the organization's record is still in active use -
  -->
  <type><!-- 0..1 CodeableConcept Kind of organization --></type>
  <name value="[string]"/><!-- 📄 0..1 Name used for the organization -->
  <telecom><!-- 📄 0..* ContactPoint A contact detail for the organization --></telecom>
  <address><!-- 📄 0..* Address An address for the organization --></address>
  <partOf><!-- 0..1 Reference(Organization) The organization of which this organization forms a p
  art --></partOf>
  <contact> <!-- 0..* Contact for the organization for a certain purpose -->
    <purpose><!-- 0..1 CodeableConcept The type of contact --></purpose>
    <name><!-- 0..1 HumanName A name associated with the contact --></name>
    <telecom><!-- 0..* ContactPoint Contact details (telephone, email, etc.) for a contact --></t
    elecom>
    <address><!-- 0..1 Address Visiting or postal addresses for the contact --></address>
  </contact>
</Organization>
```

(Figura 2.6: Descrizione della risorsa Organization in formato XML)

Lo standard FHIR mette a disposizione delle linee guida per la strutturazione di una risorsa. Nell'esempio, in figura 2.6, verrà analizzata la risorsa *Organization* presente nel gruppo delle risorse identificative, illustrate in precedenza. Come si può notare nel documento, la risorsa è composta da diversi tag contenenti i dati delle organizzazioni. I vari tag della risorsa presentano anche una cardinalità identificata nella forma: `0..*` e `0..1`, la prima indica che ci possono essere zero o più elementi (come per il tag `identifier`), mentre la seconda indica che l'elemento può sia essere omesso

o avere una sola occorrenza (un esempio è il tag `active`). Ci sono alcuni accorgimenti importanti da seguire prima di implementare una risorsa:

- il testo del documento è *case sensitive*, quindi, il nome Organization sarà diverso organization;
- gli attributi delle risorse possono contenere dei valori alfanumerici (compresi caratteri speciali) oppure degli URL;
- il *namespace* delle risorse deve essere indicato dalla stringa `xmlns=http://hl7.org/fhir`, presente nel tag *root* del documento; rispettivamente per il testo narrativo si userà il namespace: `xmlns=http://www.w3.org/1999/xhtml`;
- gli elementi delle risorse non possono essere lasciati vuoti; ogni elemento del documento deve possedere almeno un attributo con un valore significativo o un elemento figlio in base alla specifica della risorsa;
- Le risorse per poter essere scambiate necessitano di utilizzare la codifica UTF-8 specificata con l'attributo `encoding` all'inizio del documento.

Di seguito sarà mostrata una risorsa generata dalle API del fascicolo sanitario descritto nel lavoro di Tesi. La prima parte della risorsa sarà raffigurata come in figura 2.7.

```
<?xml version="1.0" encoding="utf-8"?>
<Organization xmlns="http://hl7.org/fhir">
  <id value="2"/>
```

(Figura 2.7: tag iniziali della risorsa Organization)

Nei primi tag viene descritto il formato e la versione con cui il documento XML sarà interpretato; viene poi inserito il tag root del documento, cioè `Organization`, con un attributo che rimanda al sito ufficiale dell'organismo HL7. Successivamente, verrà indicato l'id, con il tag `id`, che servirà per contraddistinguere la risorsa da altre di tipo `Organization`.

Si passa in seguito alla parte narrativa, contenuta nel tag `text`.

```
<text>
  <status value="generated"/>
  <div xmlns="http://www.w3.org/1999/xhtml">
    <table border="2">
      <tbody>
        <tr>
          <td>Nome studio</td>
          <td>Centro Curatutto</td>
        </tr>
```

(Figura 2.8: Testo narrativo della risorsa `Organization`)

Il testo narrativo è contraddistinto dalla presenza di tag XHTML come nella figura 2.8. I tag `div` e `table` consentiranno di visualizzare nel browser, che interpreta la risorsa, le varie informazioni nella tabella, in maniera simile a come è stato già illustrato in figura 2.4.

```
<extension url="http://fsem.eu/extensions/practitioner-id.xml">
  <valueString value="95"/>
</extension>
<extension url="http://fsem.eu/extensions/organization-type.xml">
  <valueString value="Studio specialistico"/>
</extension>
```

(Figura 2.9: Estensioni della risorsa `Organization`)

Le estensioni, descritte dal tag `extension`, sono presenti nel documento per l'aggiunta di ulteriori informazioni sulla risorsa, come l'id del care provider che lavora nello studio ed il tipo di organizzazione in esame, come in figura 2.9, ovvero uno studio specialistico.

La parte riguardante la struttura dati sarà inserita subito dopo i tag per le estensioni. La struttura dati deve essere modellata in base alle specifiche descritte sulla documentazione ufficiale di FHIR. In figura 2.10 sono illustrati alcuni campi che compongono la parte sulla struttura dati e si può inoltre notare il tag di chiusura del documento per la risorsa Organization.

```
<address>
  <line value="panacea, 13"/>
  <city value="Castelluccio dei Sauri (FG)"/>
  <country value="IT"/>
</address>
<contact>
  <name>
    <family value="Kelso"/>
    <given value="Bob"/>
    <prefix value="Dott."/>
  </name>
</contact>
</Organization>
```

(Figura 2.10: struttura dati e chiusura del documento)

L'organismo HL7 mette a disposizione online un *tool* di validazione delle risorse per controllare se l'EHR genera risorse ben formate e conformi allo standard. In questo lavoro di Tesi si è fatto largo uso di questo strumento, per ogni risorsa sviluppata presente nel registro sanitario elettronico.

3. Stato dell'arte

3.1. Portabilità secondo il Regolamento europeo

Data la natura sensibile dei dati che un EHR deve contenere, risulta importante l'argomento della gestione della protezione dei dati personali.

In Italia il 24 maggio 2016 è stato approvato il Regolamento dell'Unione Europea (UE) 2016/679 in merito alla protezione dei dati personali. Gli aspetti trattati in questo regolamento sono il diritto all'oblio, il diritto alla **portabilità dei dati** e l'obbligo di comunicare le violazioni e gli attacchi informatici subiti (*data breach*). Per quanto concerne la portabilità dei dati, il regolamento permette ai cittadini europei di esercitare questo diritto in maniera tale che, qualora una persona decidesse di cambiare il *provider* di un servizio, potrebbe richiedere una copia dei propri dati da riutilizzare compatibilmente con un servizio alternativo. Oltre al diritto di ricevere una copia dei propri dati, l'interessato ha diritto di trasmetterli ad un altro titolare del trattamento, senza che il primo titolare possa porre alcun impedimento al trasferimento. La persona che richiede il cambio del *provider* può, inoltre, richiedere la trasmissione diretta dei dati personali da un titolare del trattamento all'altro, limitatamente alla fattibilità tecnica di questa operazione.

Il **Registro Sanitario Elettronico Personale** presenta già diverse funzionalità che garantiscono un buon livello di sicurezza e che tutelano la privacy dei pazienti che vi sono registrati. Tuttavia, un meccanismo di portabilità non era stato ancora implementato. Grazie agli sviluppi già

intrapresi sull'**interoperabilità** mediante lo standard FHIR, si è potuto introdurre (come esposto nei capitoli successivi), una nuova funzionalità per l'ottenimento di una copia dei dati personali dei pazienti presenti nel registro sanitario elettronico. Questa funzionalità per la portabilità dei dati consente di trasferire la cartella sanitaria di un paziente registrato nel RESP, da un EHR ad un altro, oltre a rispettare il regolamento europeo appena discusso.

3.2. Situazione pregressa del sistema

Prima dell'inizio di questo lavoro di Tesi sul Registro Sanitario Elettronico Personale (RESP), erano già state sviluppate parecchie componenti che permettessero una buona fruizione del servizio. Il RESP è un PHR, quindi permette di memorizzare diverse informazioni di natura sanitaria su pazienti, medici, strutture e altre entità. In seguito all'installazione del registro sanitario su un server, un ospedale può avere accesso al sistema EHR e cominciare ad utilizzarlo per il proprio contesto ospedaliero.

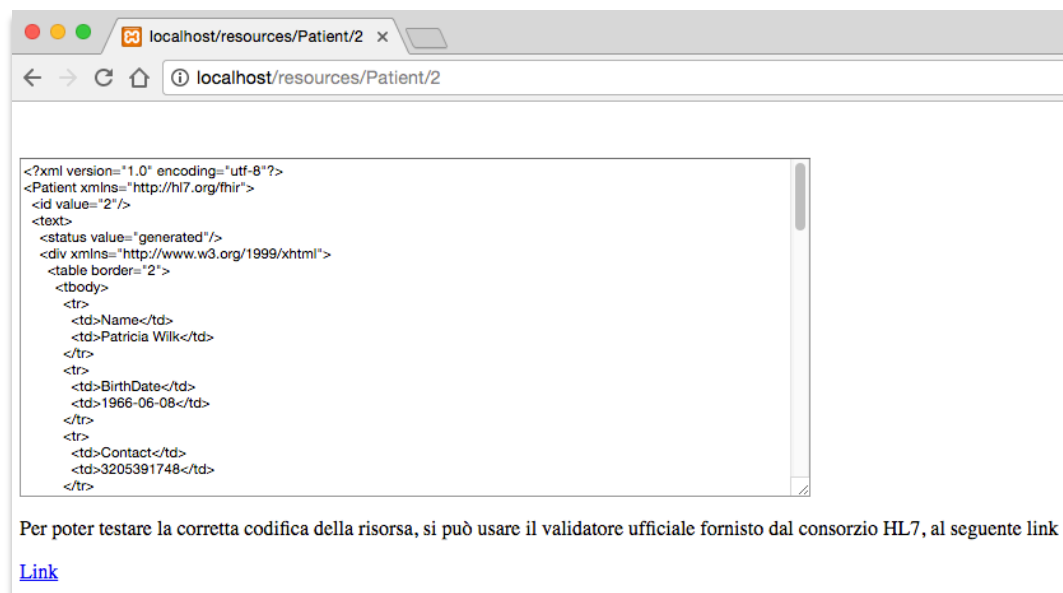
A livello di interoperabilità, il registro sanitario adotta lo standard FHIR e, inizialmente, presentava un web service molto **elementare**, basato su architettura REST, che accettava solamente l'operazione GET sulle risorse del sistema.

Le risorse sviluppate e standardizzate nel rispetto di FHIR erano *Patient*, *Practitioner*, *DiagnosticReport* e *Observation*.

La generazione delle risorse, effettuata dinamicamente in PHP, consisteva nel prelevare i dati direttamente dal database (MySQL nel caso di RESP), successivamente generare il corpo XML in maniera automatica ed infine salvare la risorsa su un file presente in una directory chiamata come il tipo

della risorsa. Ad esempio, se veniva generata la risorsa *Patient* con id numero 2, allora la risorsa era salvata sul server sotto forma di file chiamato `2.xml` nella directory `Patient/`.

L'unica maniera che un sistema esterno aveva per ottenere le risorse sanitarie, tramite richieste GET, era copiare manualmente il corpo della risorsa da una *textbox*, presente nella pagina di visualizzazione delle risorse. Il modulo per la visualizzazione delle richieste GET è illustrato in figura 3.1.



(Figura 3.1: Schermata di visualizzazione della risorsa paziente con id numero 2)

Per simulare il formato degli URL richiesti dallo standard FHIR (vedere figura 2.2), era stato progettato il file `.htaccess` in modo tale da mappare manualmente ogni tipo risorsa. A causa di questa scelta progettuale l'aggiunta di un'ulteriore categoria di risorse avrebbe costretto lo

sviluppatore a cambiare nuovamente il file `.htaccess` per la manipolazione delle nuove risorse.

Non è stato inoltre progettato alcun supporto per la codifica JSON, anche se le specifiche FHIR prevedono l'eventuale scelta tra JSON e XML. Tra le API rese disponibili nel registro sanitario elettronico, non è stata implementata alcuna forma di autenticazione per il prelievo dei dati; chiunque dall'esterno aveva accesso ad ogni risorsa del sistema potendo leggere informazioni anche di natura sensibile.

4. Progetto

4.1. Refactoring del modulo delle API

La prima parte su cui si è intervenuti nel sistema riguardava gli script che gestivano le richieste dei client per il controllo delle API.

Quando un client effettuava una richiesta al web service per l'acquisizione di una risorsa (tramite metodo GET), il gestore delle API inizialmente controllava il tipo di richiesta in base alle operazioni HTTP, in seguito instradava la richiesta mediante un controllo con un costrutto **switch-case** sulle risorse disponibili nel sistema, come in figura 4.1.

```
case "GET":
    switch ($tipo_risorsa) {
        case "Patient":
            RecuperaPaziente($id_ris);
            break;
        case "Practitioner":
            RecuperaCarePerovider($id_ris);
            break;
        case "Observation":
            RecuperaObservation($id_ris);
            break;
        case "DiagnosticReport":
            RecuperaDiagnosticReport($id_ris);
            break;
```

(Figura 4.1: costrutto switch-case per la gestione del metodo GET)

Le 4 risorse dello standard FHIR erano contenute in file separati, all'interno di ciascuno dei quali era collocata una funzione specifica che, prendendo in input l'id della risorsa dall'URL fornito dal client, effettuava delle richieste al database **MySQL** per il prelievo dei dati. I dati ottenuti dal database

servivano per costruire il corpo XML della risorsa; questa sarebbe poi stata restituita dalla funzione al gestore principale delle API. Il gestore visualizzava infine la risorsa costruita allegando il codice XML in una textbox della pagina HTML.

Il sistema di gestione delle API presentava una architettura del codice prettamente monolitica, concepita avendo in mente uno stile imperativo del linguaggio PHP. Non essendo stati implementati pattern ingegneristici il riuso e l'estendibilità del codice erano difficilmente ottenibili. Partendo da un percorso di Tesi sullo sviluppo di ulteriori risorse FHIR, il modulo di gestione delle API necessitava di essere migliorato: per questo motivo, sono stati implementati dei pattern ingegneristici per consentire l'aggiunta e la gestione delle risorse in maniera più semplice e si sono utilizzati i principi dell'*object oriented programming*.

Ogni risorsa viene rappresentata da una classe con lo stesso nome della risorsa; queste classi ereditano da una classe astratta chiamata `FHIRResource` (figura 4.2). La classe astratta mette a disposizione dei metodi che rappresentano le funzioni principali da realizzare nel web service FHIR cioè GET, POST, PUT e DELETE e ogni classe che realizza la risorsa implementa i quattro metodi. Così facendo il codice delle risorse è stato reso *object oriented*, sostituendo la singola funzione di generazione del codice XML della risorsa con il metodo `getResource`, presente in ogni classe.

```

<?php
abstract class FHIRResource {
    abstract function getResource($id);
    abstract function createResource($xml);
    abstract function updateResource($id, $xml);
    abstract function deleteResource($id);
}
?>

```

(Figura 4.2: classe astratta da cui deriva ogni risorsa)

Dopo questa modifica, si è astratta la scelta della risorsa da implementare rispetto al client, incapsulandola nella classe `ResourceFactory`. È stato determinato a *run-time* l'oggetto che rappresenta la risorsa, stabilendo la classe da istanziare, e si è delegata a `ResourceFactory` la chiamata al metodo corrispondente al *verb* HTTP. Nel dettaglio, viene passata in input al costruttore di `ResourceFactory` la stringa con il nome della risorsa, in base alla quale esso sceglie l'oggetto corrispondente. Disponendo di questa classe, il gestore delle API può richiamare dinamicamente i metodi per la manipolazione delle risorse semplicemente fornendo a `ResourceFactory` il nome della risorsa e il tipo di operazione HTTP richiesti. È possibile osservare una sinossi della chiamata alla classe nella figura 4.3.

```

$resource_factory = new ResourceFactory("Patient");
$xml_content = $resource_factory->getData('2');

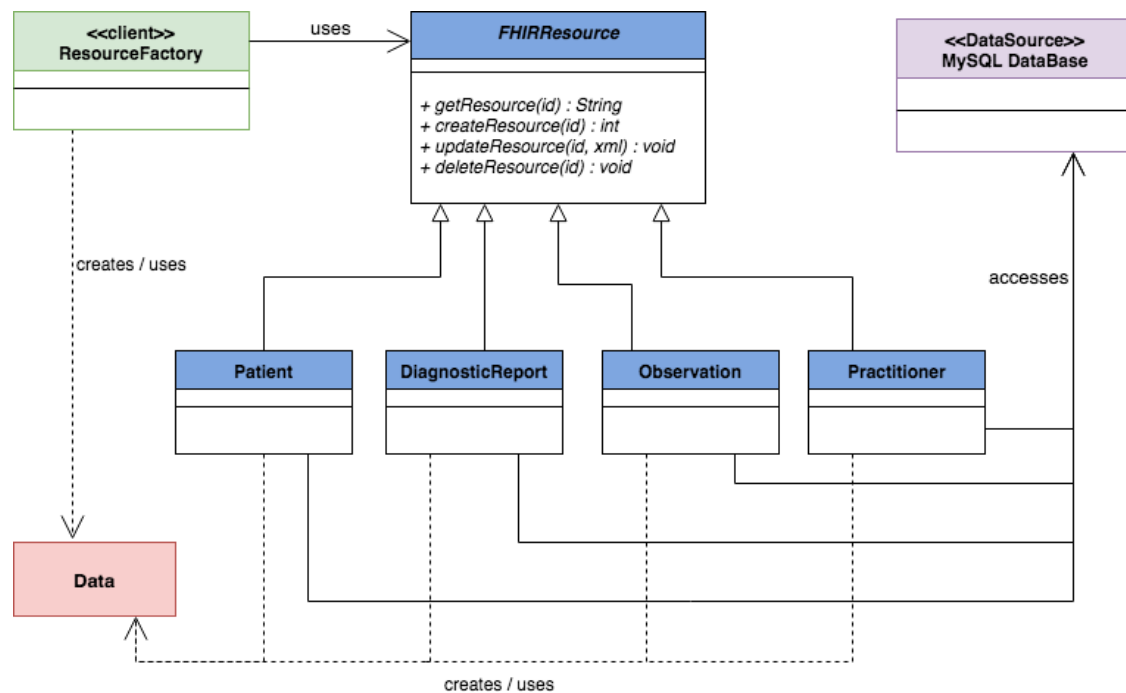
```

(Figura 4.3: creazione dell'oggetto `ResourceFactory` per l'acquisizione della risorsa `Patient` con id numero 2)

Si può facilmente notare come l'utilizzo di questa tecnica non solo abbia permesso di ridurre la complessità ciclomatica del codice esposto in figura

4.1, ma anche di migliorare la riusabilità del codice con un approccio di programmazione orientato agli oggetti.

Passando a `FHIRResource` e le sue sottoclassi, si può osservare che esse realizzano il pattern architetturale **Data Access Object** (DAO), ricordando che esso viene utilizzato per astrarre e incapsulare gli accessi ai *datasource* in un *tier* separato dal resto dei moduli. Infatti, `FHIRResource` fornisce l'interfaccia per la manipolazione dei dati, mentre le classi concrete gestiscono l'effettiva connessione con il database e l'elaborazione dei dati. È possibile osservare il diagramma UML di questo pattern in figura 4.4.



(Figura 4.4: diagramma UML del pattern DAO)

L'utilizzo di questo pattern ha permesso di stabilire una suddivisione dei vari *tier* in maniera ben definita. È possibile, inoltre, notare che il sistema di

gestione delle API, nella sua interezza, è stato progettato in base al pattern architetturale **Model-View-Controller**; le classi *model* possono essere identificate con i vari DAO per le chiamate al database, il *controller* è rappresentato dal gestore delle richieste (situato in `index.php`), infine il *view* è gestito dalla classe `ResourceView` (illustrata nei capitoli successivi).

In seguito a queste modifiche, la creazione di nuove risorse risulta nettamente più semplice, poiché basta creare una classe corrispondente alla nuova risorsa e farle estendere la classe astratta `FHIRResource`. Saranno illustrate, nei paragrafi successivi, le nuove risorse FHIR sviluppate seguendo questo schema.

4.2. Sviluppo della risorsa FamilyMemberHistory

La prima risorsa sviluppata per rendere il registro conforme allo standard FHIR è stata **FamilyMemberHistory**, cioè la pagina sulle anamnesi familiari. La progettazione della risorsa è stata organizzata in maniera tale da modificare l'interfaccia grafica per l'acquisizione dei dati sulle anamnesi, riadattare il database per la memorizzazione di tali campi ed infine codificare la risorsa con le informazioni presenti nel database.

Prima dei cambiamenti apportati al sistema, la sezione sulle anamnesi familiari, nella pagina *Anamnesi* del RESP, era gestita come un campo libero che il *care provider* o il *paziente* potevano riempire arbitrariamente con le relative informazioni. Dato che una analisi del testo libero sarebbe risultata parecchio complessa per l'acquisizione delle varie informazioni da standardizzare, si è preferito creare una nuova schermata che richiedesse esplicitamente i campi della risorsa. La schermata è illustrata in figura 4.5.

(Figura 4.5: finestra con i campi delle anamnesi familiari)

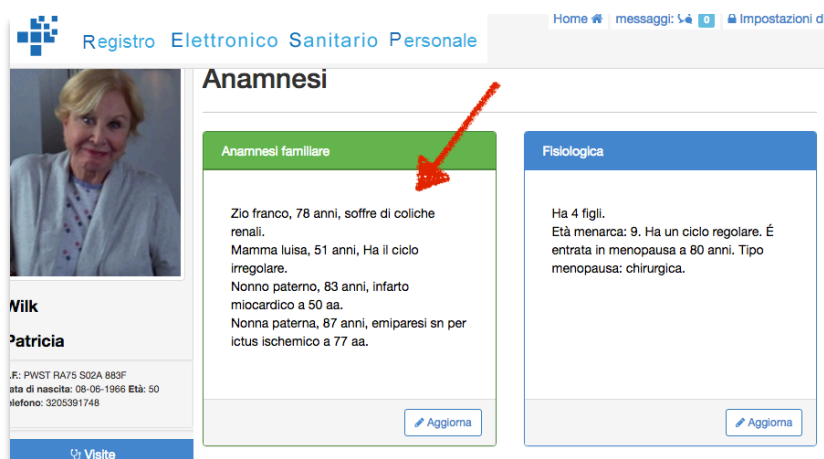
Si è passati poi alla riprogettazione del database in maniera che accettasse i campi presenti nella finestra illustrata poc'anzi. Gli attributi della tabella sulle anamnesi familiari sono esposti di seguito:

- **id**, valore autoincrementato che tiene conto del singolo record sulle anamnesi familiari (sarà inseguito utilizzato come l'id logico da fornire per la risorsa FHIR);
- **idpaziente**, rappresenta l'id dell'utente nel database;
- **idcpp**, memorizza l'id del *care provider*, se questo ha aggiunto o modificato l'anamnesi familiare;
- **dataAggiornamento**, rappresenta la data di aggiunta o modifica dell'anamnesi;
- **componente**, registra un identificativo del familiare (Es: Zio Cesare, Nonna Carolina, Mario Rossi);
- **Sesso**, indica il genere del componente;

- **dataMorte**, registra la data di morte (NULL nel caso in cui è in vita);
- **snomedId** è un campo utilizzato per memorizzare la codifica *SNOMED CT*¹ per lo stato di salute del componente;
- **note** è infine un campo utilizzato per la memorizzazione della condizione medica del membro familiare.

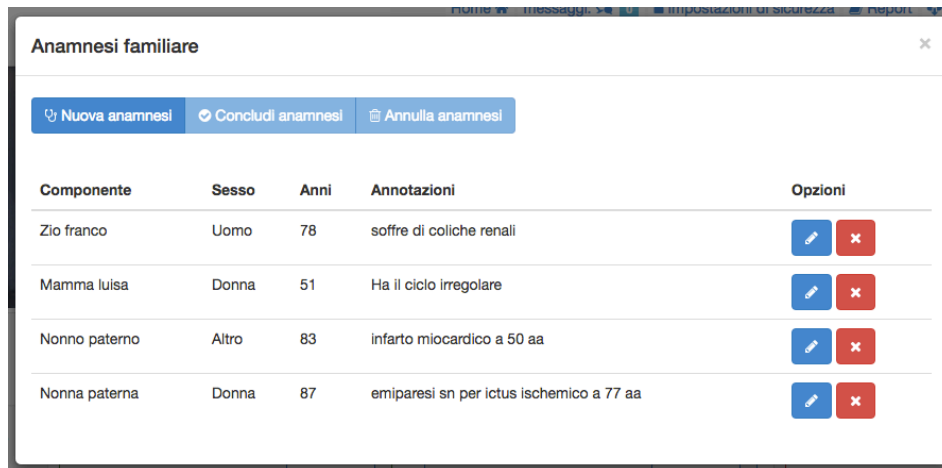
C'è da ricordare che, per motivi di semplicità, non si è progettata l'interfaccia grafica in modo che permettesse l'inserimento della codifica SNOMED CT per la condizione del familiare, ma le API (come esposto in seguito) sono predisposte per accettare tale valore da una richiesta POST effettuata da un sistema esterno (motivo per cui è stato creato il campo **snomedId** nella struttura).

In seguito alla modifica del database, si possono osservare nelle figure 4.6 e 4.7 le schermate di visualizzazione delle anamnesi familiari.



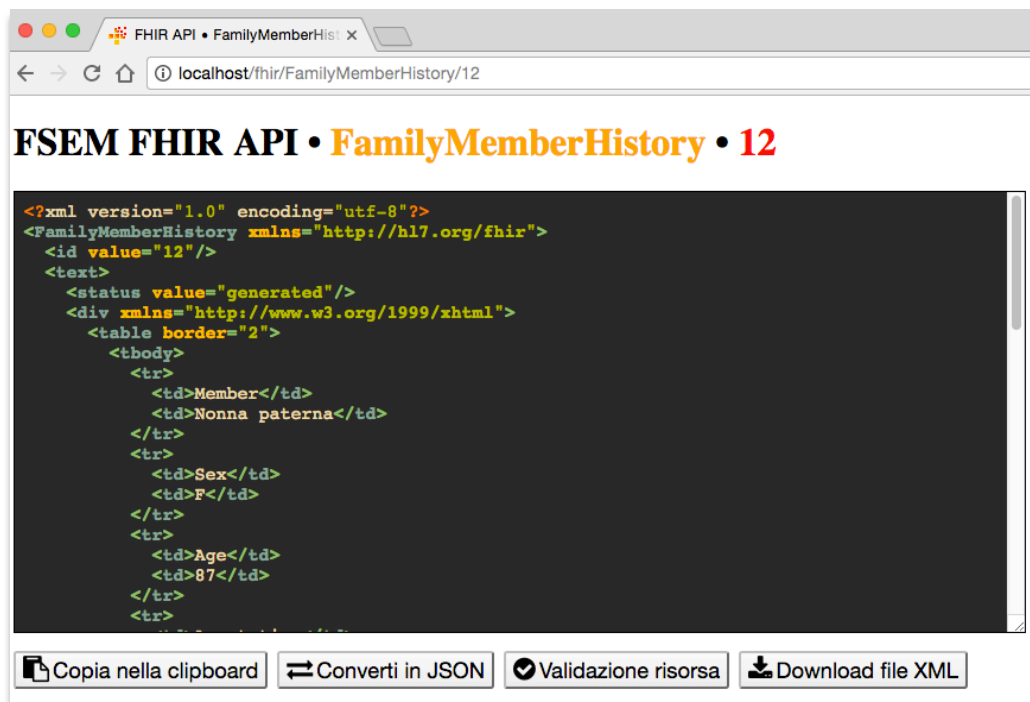
(Figura 4.6: la prima schermata mostrata appena si clicca sulla pagina Anamnesi)

¹ È uno delle più grandi raccolte di terminologia medica processabile da un sistema informatico. Comprende codici, termini e definizioni del settore sanitario.



(Figura 4.7: la schermata mostrata quando si clicca sul pulsante “Aggiorna”, per la modifica, eliminazione o aggiunta di nuove anamnesi familiari.)

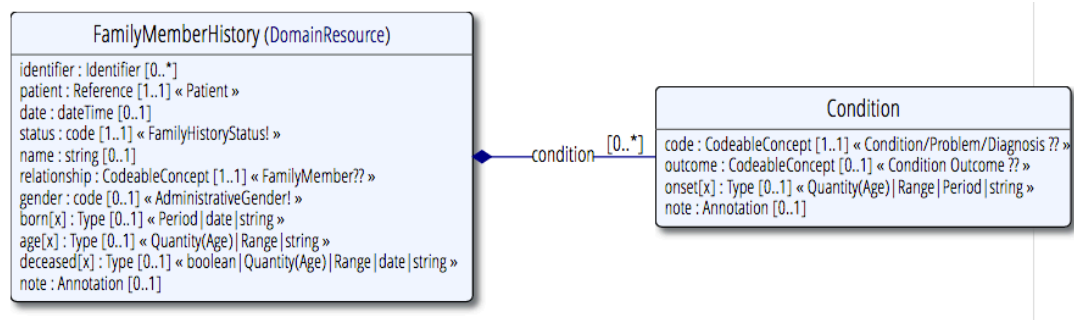
Avendo predisposto il RESP per la nuova gestione delle anamnesi familiari, si è passati all’effettivo sviluppo della risorsa. È stata creata una nuova classe chiamata `FamilyMemberHistory` che estendeva la classe astratta `FHIRResource` ed implementava i metodi della risorsa per la manipolazione dei dati sulle anamnesi. Ad esempio, con il metodo `getResource()`, specificando l’id della risorsa, un client esterno può fare richiesta di una anamnesi del paziente. Il metodo, come già spiegato, genera il corpo XML della risorsa prelevando i dati direttamente dal database; restituisce poi il contenuto generato al gestore delle API. Se il client effettua la richiesta da un browser web la schermata di visualizzazione della risorsa sarà come quella illustrata in figura 4.8.



(Figura 4.8: schermata visualizzazione della risorsa FamilyMemberHistory)

Come si può notare dall'immagine appena esposta, sono state apportate delle migliorie grafiche alla schermata di visualizzazione della risorsa rispetto all'iniziale versione in figura 3.1. Oltre alla colorazione del codice XML (**syntax highlighting**), che aumenta la leggibilità, sono stati introdotti dei pulsanti d'azione per abilitare varie azioni sulla risorsa (saranno spiegati più in seguito).

Spiegando nel dettaglio la struttura della risorsa FamilyMemberHistory, si può osservare dalla figura 4.9 il suo diagramma UML. L'elemento id sarà scelto in base al campo id del record nel database per la singola anamnesi.



(Figura 4.9: diagramma UML della risorsa FamilyMemberHistory)

La parte narrativa visualizza in forma tabellare il nome del membro familiare, il genere, l'età e le note sulla condizione clinica. La struttura dati codifica le varie informazioni prelevate dal database come nelle figure 4.10 e 4.11.

```

<identifier>
  <use value="usual"/>
  <system value="urn:ietf:rhc:3986"/>
  <value value="../fhir/FamilyMemberHistory/12"/>
</identifier>
<patient>
  <reference value="../fhir/Patient/2"/>
</patient>
<date value="2016-11-30T00:00:00+01:00"/>
<status value="completed"/>
<name value="Nonna paterna"/>
<relationship>
  <coding>
    <system value="http://hl7.org/fhir/v3/RoleCode"/>
    <code value="FAMMEMB"/>
  </coding>
</relationship>
<gender value="female"/>
<bornString value="1929-01-01"/>
<ageString value="87"/>
<deceasedBoolean value="false"/>
  
```

(Figura 4.10: Parte struttura dati di FamilyMemberHistory)

```

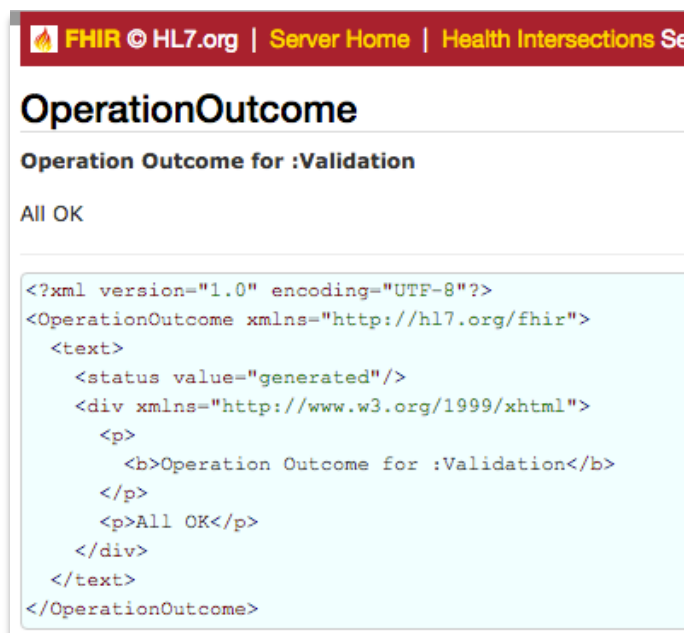
<condition>
  <code>
    <coding>
      <system value="http://snomed.info/sct"/>
      <code value="109006"/>
    </coding>
  </code>
  <note>
    <authorReference>
      <reference value="../fhir/Patient/2"/>
    </authorReference>
    <text value="emiparesi sn per ictus ischemico a 77 aa"/>
  </note>
</condition>

```

(Figura 4.11: continua struttura dati di FamilyMemberHistory)

Alcuni elementi tra cui `gender` sono stati tradotti dalla lingua italiana in inglese per adeguarsi allo standard. Per quanto riguarda l'elemento `condition` con il nodo figlio `coding` (per la codifica SNOMED), si è inserito un valore d'esempio per quando la risorsa non dispone ancora di una codifica SNOMED CT; nel caso in cui il codice dovesse essere presente, allora la corretta codifica verrà visualizzata nella struttura dati della risorsa. Come sviluppo futuro si può prevedere di modificare la schermata di gestione delle anamnesi familiari in modo da registrare anche la codifica SNOMED.

Provando infine a inserire il codice XML della risorsa all'interno del *tool* di validazione, messo a disposizione dall'organismo HL7, si è potuto verificare che la risorsa fosse ben formata da un punto di vista sintattico e conforme allo standard FHIR. In figura 4.12 è possibile osservare l'*OperationOutcome* del validatore.



(Figura 4.12: risposta del tool di validazione dopo aver caricato il codice della risorsa *FamilyMemberHistory*)

Grazie ai cambiamenti apportati nella gestione delle API, la creazione della risorsa sulle anamnesi familiari è stata molto più intuitiva e veloce, in virtù della maggiore ingegnerizzazione dei moduli interessati.

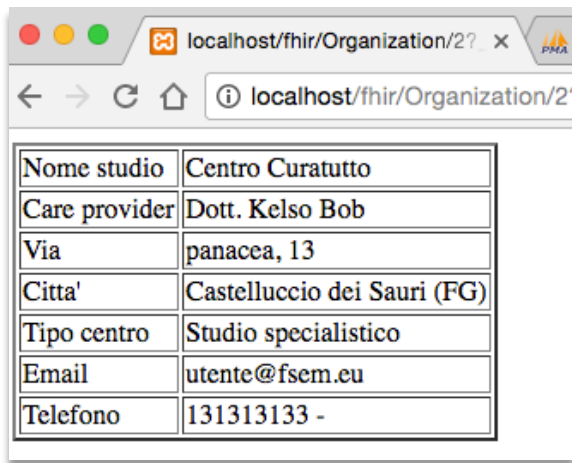
4.3. Sviluppo della risorsa Organization

La risorsa *Organization*, facente parte della categoria delle **Identificative**, consente di rappresentare un gruppo di persone o di organizzazioni - formalmente o informalmente riconosciuto - il cui scopo è quello di raggiungere una forma di azione collettiva. Tra questi gruppi possono rientrare centri ospedalieri, istituzioni sanitarie, studi medici privati, etc. Il fine di avere un'entità standardizzata con tale risorsa risiede nel fatto che questa potrà rendere disponibili dei dati di natura informativa come contatti,

luoghi o indirizzi di domicilio, oltre ad essere utilizzabile come **riferimento** (Reference) in una diversa risorsa FHIR, come ad esempio *HealthcareService*.

A differenza della risorsa illustrata nel precedente paragrafo, per la standardizzazione di questa informazione il RESP non necessitava di essere modificato. Tutti i dati da raccogliere nel documento XML erano già disponibili all'interno del database; sono state utilizzate due tabelle che contenevano le informazioni di cui usufruire, ossia **centriindagini** e **telefonocentriindagini**. Nella prima tabella sono presenti dati sul *care provider*, la sua email e l'indirizzo dello studio in cui esso lavora; all'interno della seconda tabella invece è contenuto il numero di telefono del centro. La risorsa è stata creata in maniera analoga a quella delle anamnesi familiari, creando la classe **Organization**, facendole estendere **FHIRResource** per poi implementare i quattro metodi astratti.

La struttura della risorsa **Organization** (illustrata come esempio di risorsa nel Capitolo 2 con le figure 2.5 e 2.6) è simile a quelle già sviluppate in precedenza, con l'unica differenza che necessitava di contenere ulteriori dati presenti nel database ma non elencati tra i campi suggeriti dallo standard FHIR. Per questo motivo l'utilizzo delle **estensioni** è stata la soluzione che ha permesso di rappresentare tali dati. Le estensioni utilizzate riguardavano la rappresentazione dell'**id del care provider** ed il **tipo di organizzazione** in cui esso lavora. In figura 4.13 si può osservare la parte del testo narrativo della risorsa, in una rappresentazione tabellare dei dati.



A screenshot of a web browser window displaying a table with patient information. The browser's address bar shows 'localhost/fhir/Organization/2?'. The table has two columns and eight rows of data.

Nome studio	Centro Curatutto
Care provider	Dott. Kelso Bob
Via	panacea, 13
Citta'	Castelluccio dei Sauri (FG)
Tipo centro	Studio specialistico
Email	utente@fsem.eu
Telefono	131313133 -

(Figura 4.13: parte narrativa della risorsa Organization)

All'interno della parte della **struttura dati** sono state rappresentate le informazioni del database mediante l'uso di alcuni importanti campi tra cui `telecom` e `contact`, illustrati rispettivamente in figura 4.14 e 4.15.

```
<telecom>
  <system value="email"/>
  <value value="bobkelso@fsem.eu"/>
  <use value="work"/>
</telecom>
```

(Figura 4.14: campo telecom della risorsa Organization)

```
<contact>
  <name>
    <family value="Kelso"/>
    <given value="Bob"/>
    <prefix value="Dott."/>
  </name>
</contact>
```

(Figura 4.15: campo contact della risorsa Organization)

Il campo `telecom` è stato utilizzato per contenere informazioni sui contatti del *care provider*, tra queste ci sono il numero di telefono o l'indirizzo email; nel campo `contact` sono presenti informazioni identificative quali nome, cognome e un titolo della professione (opzionale).

Inserendo nuovamente la risorsa XML, generata dal gestore delle API, nel *validatore* di HL7 si è potuta controllare la conformità del documento allo standard, con un risultato uguale a quello ottenuto in figura 4.12.

4.4. Sviluppo della risorsa *OperationOutcome*

Come ultima risorsa sviluppata nel sistema si è deciso di introdurre *OperationOutcome*, presente nella categoria delle risorse di infrastruttura (**Infrastructure**). Questa risorsa è di fondamentale importanza poiché permette al client che si interfaccia con il web service di analizzare la risposta ad una richiesta che ha scaturito un errore nel sistema. Gli *OperationOutcome* sono una collezione di errori, avvisi e messaggi informativi utili al client al fine di indagare sulla situazione d'errore che si presenta. Tra le tante condizioni d'errore ci possono essere ad esempio problemi riguardanti:

- la mancata implementazione di una risorsa;
- l'id di una risorsa non esistente nel sistema;
- la mancata implementazione di una operazione REST;
- la struttura di una risorsa malformata;
- gli errori nella cancellazione di una risorsa;

OperationOutcome, a differenza delle altre risorse sviluppate nel sistema, non è rappresentata come classe DAO perché non identifica una risorsa che

corrisponde a delle entità nel database; è stata sviluppata piuttosto come una classe a sé stante che fornisce un metodo statico in cui, inserendo un testo di errore, si genera la risorsa XML in output. La classe `OperationOutcome` viene tipicamente utilizzata quando le classi DAO sollevano delle eccezioni e il gestore delle eccezioni passa il messaggio di errore direttamente al metodo `getXML` di questa classe. Un esempio di caso d'uso si può osservare in figura 4.16.

```
try {
    $resource = new ResourceFactory($url_content['type']);
    $resource->deleteData($url_content['id']);

    http_response_code(200);
} catch (UnsupportedOperationException $e) {
    $view->display_raw(OperationOutcome::getXML($e->getMessage()));
    http_response_code(405);
}
```

(Figura 4.16: esempio di utilizzo della classe *OperationOutcome*)

Nella precedente figura, la risorsa generata *OperationOutcome* viene passata direttamente alla classe che gestisce il *tier* delle *view*² nel gestore delle API. Gli *OperationOutcome* possono presentarsi anche quando gli *header* della richiesta del client non corrispondono alle specifiche dello standard FHIR; ad esempio se il campo `content-type` della richiesta del client non ha il valore `application/xml-fhir`, la risorsa *OperationOutcome* conterrà il messaggio di errore “Invalid Content-Type”.

² Il sistema, sviluppato con architettura *Model-View-Controller*, mette a disposizione una classe per la gestione delle viste (ci si è riferiti in precedenza a tali “viste” con il nome *pagina di visualizzazione delle risorse*).



(Figura 4.17: visualizzazione della risorsa *OperationOutcome*)

In figura 4.17 si può notare un altro esempio di errore nella richiesta di una risorsa di tipo *Patient*; il sistema, avendo riconosciuto l'errore tipografico nella formulazione dell'URL, restituisce in output l'*OperationOutcome* con il messaggio di errore "Resource not found!".

4.5. Implementazione di nuovi servizi REST

Per partire da una solida base per il successivo lavoro sulla portabilità, è stato necessario aggiungere nuovi servizi REST al web service del Registro Sanitario Elettronico Personale.

Inizialmente l'unico servizio REST già sviluppato era GET e questo permetteva di ottenere una risorsa specificandone il tipo e l'id. Nel lavoro di

Tesi ci si è concentrati sullo sviluppo di altri tre servizi REST tra cui POST, PUT e DELETE, così da permettere una manipolazione più ampia sulle risorse del registro. I nuovi servizi introdotti corrispondono alle operazioni *create*, *update* e *delete* esposte nella documentazione dello standard FHIR (Paragrafo 2.5). Data la struttura con cui erano state sviluppate le API, per ogni risorsa già esistente nel sistema si sono dovuti sviluppare gli altri metodi per le nuove operazioni introdotte.

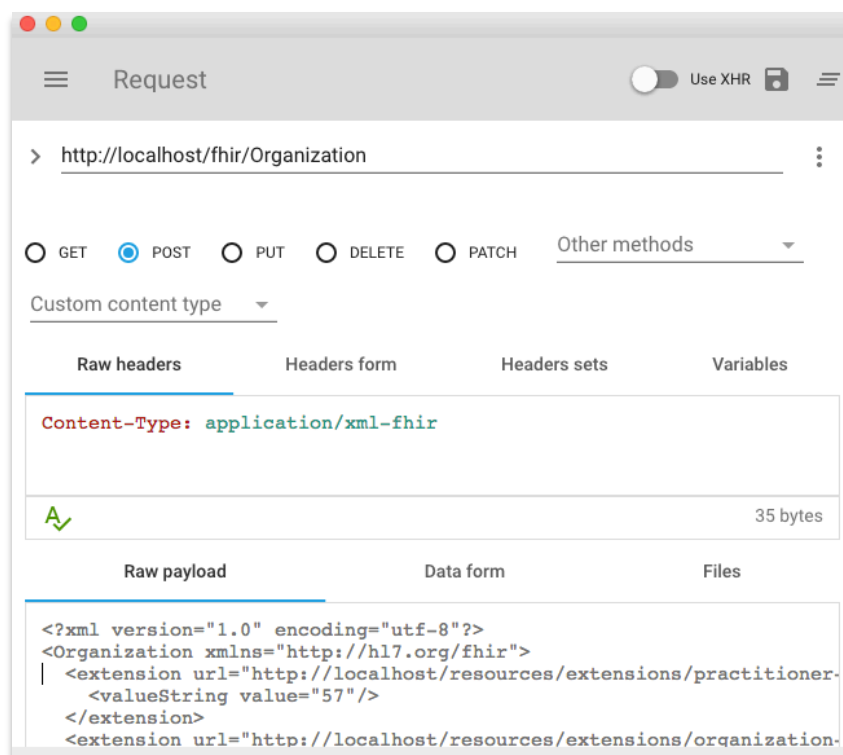
Per quanto riguarda il metodo³ con cui il client effettua la richiesta, il linguaggio PHP permette di acquisirlo attraverso la variabile globale `$_SERVER['REQUEST_METHOD']`. Dopo aver immagazzinato in una variabile il tipo della richiesta, il gestore delle API riuscirà ad eseguire l'operazione adeguata sulla risorsa specificata nell'URL.

4.5.1. Servizio POST

Le richieste di tipo **POST** effettuate dal client comprendono il riconoscimento del tipo di risorsa indicato nell'URL oltre che il contenuto della risorsa XML da creare sul server. L'URL accettato dal servizio POST sarà nella forma: `https://fsem.eu/fhir/[tipo_risorsa]`. Al posto delle parentesi quadre andrà inserito il tipo effettivo della risorsa che si vuole creare. Il contenuto XML, invece, deve essere aggiunto nella richiesta HTTP come una semplice stringa all'interno del *Payload*. Per simulare il client si è fatto uso del software **Advanced REST Client** illustrato in figura 4.18.

³ Si ricorda che il metodo con cui effettuare la richiesta può essere una delle operazioni HTTP tra cui GET, POST, PUT, DELETE, etc.

Utilizzando questo software si è potuto simulare le richieste POST specificando la risorsa da creare nel campo “Raw payload”. Da notare che tra gli *header* HTTP è previsto l’inserimento manuale del campo `Content-Type` impostandolo come nella figura 4.18. Come suggerito dalle API di FHIR, per l’operazione *create* nell’URL non deve essere specificato l’id della risorsa poiché il server provvederà ad assegnarne uno. Dopo aver effettuato questo controllo il gestore delle API dovrà ottenere il contenuto della stringa caricata nel *payload*; per fare ciò, il linguaggio PHP consente di leggere dallo stream `php://input`.



(Figura 4.18: esempio di richiesta POST con il software Advanced REST Client)

Conoscendo il **tipo** ed il **contenuto** della risorsa, il gestore delle API potrà richiamare la classe `ResourceFactory` per instradare la richiesta del client verso il gestore delle risorse. In figura 4.19 si può osservare la sintassi per la chiamata al gestore delle risorse.

```
$resource = new ResourceFactory($resource_type);  
$new_id = $resource->postData($xml_content);
```

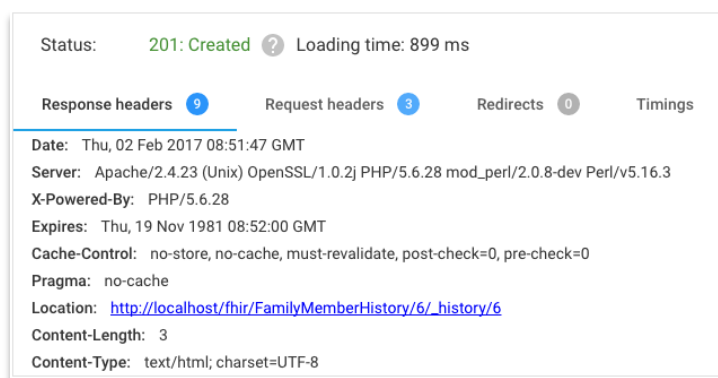
(Figura 4.19: chiamata alla classe `ResourceFactory` per la creazione della risorsa)

La classe `ResourceFactory`, in base al tipo della risorsa specificato, richiamerà il metodo `createResource` della classe DAO specifica (per esempio *Organization*). Come già descritto nei paragrafi precedenti, le classi DAO sviluppate nel registro sanitario implementano tutte il metodo astratto per la creazione della risorsa, ereditandolo dalla classe `FHIRResource`. Questo metodo, cioè `createResource`, prenderà in input il contenuto XML della risorsa ed effettuerà una operazione di *parsing* dei campi presenti nel documento; in seguito questi dati verranno inseriti all'interno del database. Il metodo, prima di terminare, effettua una *query* per controllare se il record è stato creato correttamente, in caso positivo restituisce in output l'id dello stesso. `ResourceFactory`, se tutto è andato a buon fine, restituirà questo id al gestore delle API e quest'ultimo lo inserirà nell'*header* della risposta da fornire al client. Lo standard FHIR suggerisce che nel messaggio di risposta il campo **Location** dell'*header* HTTP debba contenere l'URL con il **nuovo id** della risorsa appena creata.

Introducendo la gestione di **eccezioni personalizzate** all'interno del modulo delle API, si è potuto controllare le situazioni di errore durante le varie operazioni sulle risorse. Se una eccezione si fosse presentata nell'operazione POST, il gestore delle API avrebbe restituito un messaggio di errore in base al contesto, con i seguenti *codici di stato* HTTP:

- **400 Bad request**, se il contenuto XML della risorsa contiene il campo *id* da non specificare per l'operazione *create*;
- **422 Unprocessable entity**, se il formato della risorsa XML non è in forma corretta o se si sono presentati altri problemi come per esempio errori di connessione con il database;
- **404 Not found**, se il tipo della risorsa non è supportato dal gestore delle API.

Se la risorsa viene creata correttamente allora il codice di stato presente nell'*header* sarà **201 Created**. Provando ad effettuare la richiesta POST con il simulatore delle richieste REST si possono osservare, in figura 4.20, gli *header* della risposta del server quando la risorsa viene creata correttamente.



(Figura 4.20: header della risposta del server per la creazione di una risorsa di tipo *FamilyMemberHistory*)

4.5.2. Servizio PUT

Per le richieste di tipo PUT si è proceduto in maniera analoga allo sviluppo dell'operazione *create*. In questo lavoro di Tesi si è sviluppato il servizio PUT come operazione che consentisse esclusivamente l'**aggiornamento** della risorsa. Lo standard FHIR, però, richiede che il web service debba effettivamente creare la risorsa sul server con l'id specificato nell'URL se questa non dovesse già esistere; solo in caso contrario verrebbe effettuata una operazione di aggiornamento. Per uno sviluppo futuro si può migliorare il modulo delle API effettuando questo ulteriore controllo sull'id.

Il servizio PUT del RESP funziona accettando dal client l'URL contenente il tipo di risorsa e l'id, mentre deve essere nuovamente inserito il contenuto XML della risorsa nel *payload* della richiesta. Il documento XML servirà per ottenere i nuovi dati per aggiornare la risorsa specificata. Un esempio di URL per una richiesta di tipo PUT può presentarsi nella seguente forma:

```
https://fsem.eu/fhir/[tipo risorsa]/[id risorsa]
```

Come si può notare, questa volta, data la natura del servizio, l'id della risorsa da creare nel server deve essere specificato. Il gestore delle API, dopo aver controllato il campo id nell'URL, legge dallo *stream* `php://input` ed inoltra il contenuto della risorsa da modificare al gestore delle risorse. Il metodo delle classi DAO, per effettuare l'operazione di *update*, è `updateResource`. Anche il metodo di aggiornamento funziona in maniera tale da ottenere in input il contenuto XML della risorsa e, dopo aver *parsato* i vari campi, effettua una query di aggiornamento al database. Il metodo `updateResource` non restituisce alcun valore alla funzione chiamante, ma

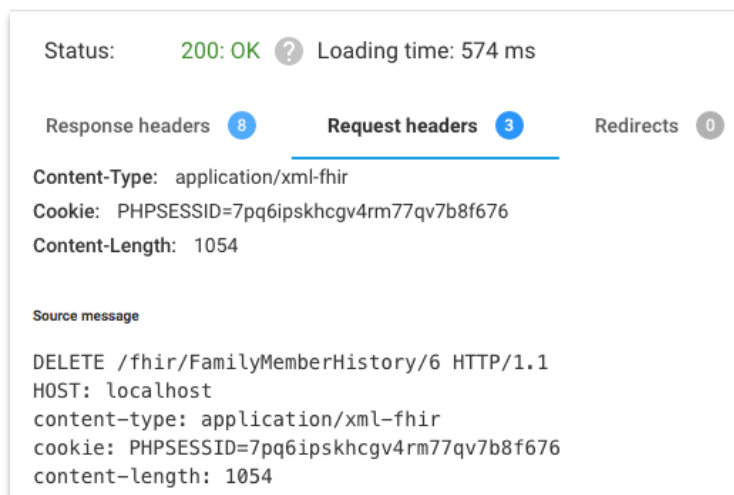
il gestore delle API, se l'operazione dovesse andare a buon fine, restituisce nuovamente al client l'URL della risorsa FHIR aggiornata. In caso di operazione effettuata correttamente il codice di stato HTTP presente negli *header* della risposta sarà **200 OK**.

4.5.3. Servizio DELETE

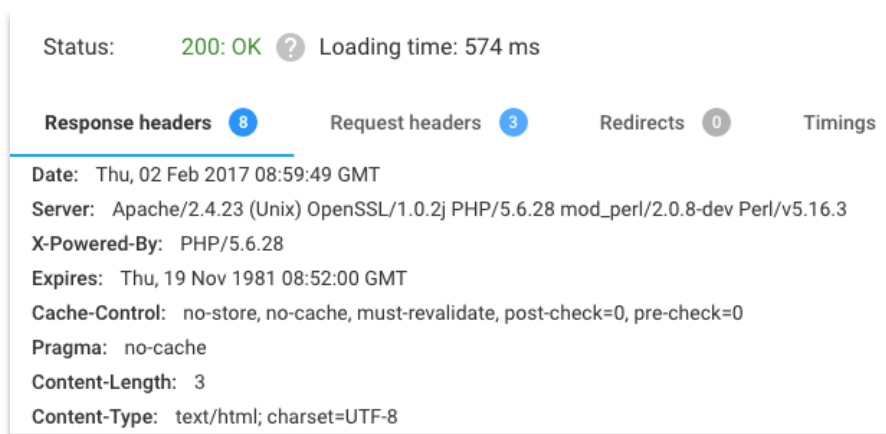
L'ultimo servizio REST implementato nel web service FHIR è quello per la cancellazione di una risorsa mediante l'operazione **DELETE**. Un esempio di URL per richiedere la cancellazione può presentarsi nella forma:

```
https://fsem.eu/fhir/[tipo risorsa]/[id risorsa]
```

A differenza delle altre operazioni, questa volta, non c'è bisogno di definire alcun contenuto XML, quindi il client deve solo specificare il precedente URL nella richiesta. Il gestore delle API, ancora una volta, richiamerà `ResourceFactory` che stabilisce in base al tipo della risorsa l'effettiva classe DAO da istanziare e successivamente chiamerà il metodo `deleteResource`. Il metodo delle classi DAO per la cancellazione di una risorsa accetta come parametro solo l'id stesso della risorsa. Il metodo `deleteResource` effettua una *query* di eliminazione nel database e successivamente controlla se il record è stato effettivamente eliminato. Se la risorsa dovesse essere eliminata senza alcun problema verrà restituito il codice di stato **200 OK**. Utilizzando nuovamente il simulatore per verificare la corretta eliminazione di una risorsa si possono notare gli *header* della richiesta del client in figura 4.21 e quelli della risposta dal web service in figura 4.22.



(Figura 4.21: header della richiesta di eliminazione della risorsa)



(Figura 4.22: header della risposta del web service)

Nel caso in cui dovessero presentarsi dei problemi durante l'operazione di cancellazione, il gestore delle API restituirebbe al client i seguenti stati:

- **405 Method not allowed**, se la classe DAO non riuscisse ad eliminare il record dal database oppure se il metodo astratto `deleteResource` non è stato implementato.

- 404 **Not found**, se l'id della risorsa da eliminare non esistesse nel sistema oppure se il tipo della risorsa non è supportato dalle API.

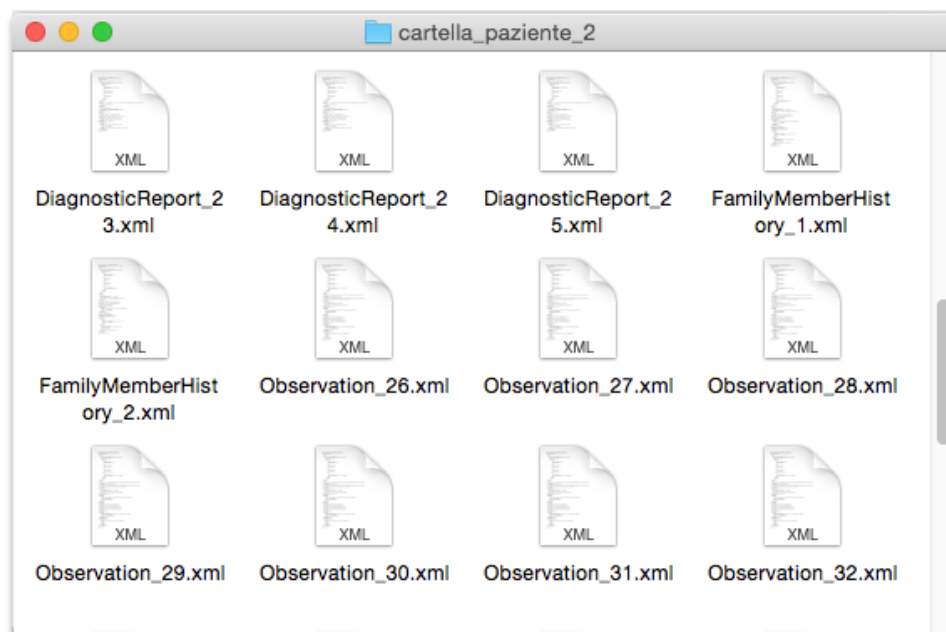
4.6. Portabilità delle cartelle sanitarie

Venendo ora al fulcro della fase di progetto, si discuterà del modulo che ha permesso la **portabilità** delle cartelle sanitarie dei pazienti in RESP. Il concetto di portabilità si è espresso, nello sviluppo di questo registro sanitario, consentendo al paziente di **esportare** in un unico file compresso tutte le risorse *standardizzate* del sistema che lo riguardano; quindi, tra queste il paziente potrà attualmente ottenere le anamnesi familiari, diagnosi, indagini diagnostiche e il profilo con i propri dati anagrafici. Il *care provider* che vorrà **importare** il paziente in un altro ospedale potrà richiedere direttamente il file compresso in modo tale da recuperare tutte le informazioni presenti nel RESP differente usato del paziente.

4.6.1. Esportazione di una cartella

Inizialmente è stato sviluppato lo *script* che permettesse la funzionalità per l'esportazione delle cartelle sanitarie. Questo modulo, sviluppato in linguaggio PHP, permette di effettuare il download del file compresso esclusivamente al paziente possessore della cartella oppure al *care provider* che lo gestisce; quindi un paziente non potrà richiedere la cartella di un altro paziente nel sistema e un *care provider* non potrà ottenere la cartella di un paziente di cui non ha i permessi di lettura. Il file che si potrà scaricare è un file **zip**, rinominato con estensione **.resp** per evitare di accedervi e modificare erroneamente le informazioni al suo interno. Lo *script*, quando

viene eseguito, inserisce automaticamente nell'archivio zip tutte le risorse collegate al paziente in formato XML; se il paziente si è appena registrato in RESP e non ha ancora svolto operazioni sul registro, l'unico file presente nell'archivio zip sarà `Patient.xml`, ovvero la risorsa che conterrà i dati inseriti in fase di registrazione. Illustrando un altro esempio, un file esportato potrà avere il nome `cartella_paziente_2.resp` ed al suo interno ci potrebbero essere le risorse come in figura 4.23.



(Figura 4.23: cartella del paziente contenente le proprie risorse standardizzate)

Per ottenere ogni risorsa illustrata nella figura precedente sono state **riutilizzate** le classi che gestiscono le API del registro sanitario. Lo script di esportazione fa uso della classe `ResourceFactory` eseguendo il metodo `getData` per ottenere il contenuto XML della risorsa. Questa operazione di acquisizione delle risorse sarà fatta per ogni record nel database che

corrisponde ai tipi: *FamilyMemberHistory*, *DiagnosticReport* e *Observation* (la risorsa *Patient* corrisponde ad un solo record). In figura 4.24 si può osservare il frammento di codice che preleva solo la risorsa di tipo *FamilyMemberHistory*.

```
if (!empty(getInfo('id', 'anamnesifamiliare', 'idpaziente = ' .  
$patient_id))) {  
  
    // prelevo l'id di ogni risorsa anamnesi  
    $fmh_ids = getArray('id', 'anamnesifamiliare', 'idpaziente = ' .  
$patient_id);  
  
    $resource = new ResourceFactory('FamilyMemberHistory');  
  
    foreach($fmh_ids as $single_id) {  
        array_push($xml_resource['FamilyMemberHistory'],  
            $resource->getData($single_id));  
    }  
}  
// prelevo altre risorse ...
```

(Figura 4.24: acquisizione di tutte le anamnesi familiari)

Nella figura precedente si può notare il controllo iniziale per verificare, all'interno del database, se ci sono anamnesi familiari con l'id del paziente di cui si vuole scaricare la cartella; in caso positivo, dopo aver prelevato tutti gli id delle anamnesi questi verranno passati come argomento al metodo `getData` per ottenere l'XML di ogni anamnesi familiare. Queste stringhe, che conterranno l'XML delle risorse, verranno inserite in un array associativo (in figura `$xml_resource`), che sarà usato per creare l'archivio zip. Dopo aver ripetuto lo stesso processo per le altre due risorse viene utilizzata la classe `ZipArchive`, presente nelle API di PHP, per la creazione dell'effettivo file zip; questo, come verrà in seguito descritto, sarà automaticamente scaricato al click di un pulsante in una pagina web e per

forzare nel browser il download di questo file è stato utilizzato il seguente frammento di codice:

```
$file_url = '/formscripts/cartella_paziente_' . $patient_id . '.resp';  
  
header('Content-Type: application/octet-stream');  
header("Content-Transfer-Encoding: Binary");  
header("Content-disposition: attachment; filename=\"\" .  
basename($file_url) . \"\"");  
  
readfile($file_url);  
  
// cancello l'archivio zip dopo averlo fatto scaricare  
unlink($zip_filename);
```

(Figura 4.25: modifica dell'header HTTP per il download automatico del file)

Viene modificato l'header HTTP per trasmettere come risposta al client un contenuto di tipo binario arbitrario⁴, specificando poi il nome del file suggerito al browser⁵ che lo scaricherà. Il file, dopo essere stato scaricato dal client, sarà automaticamente eliminato per evitare un accumulo di cartelle sanitarie sul server.

Si è passati successivamente alla progettazione dell'interfaccia grafica per consentire il download del file. È stato modificato il *template* delle pagine visualizzate dal **paziente**, aggiungendo un pulsante “Esporta profilo” nella barra superiore come in figura 4.26. Questo pulsante è sempre visibile in qualsiasi pagina visitata dal paziente e appena cliccato farà partire il download della propria cartella sanitaria.

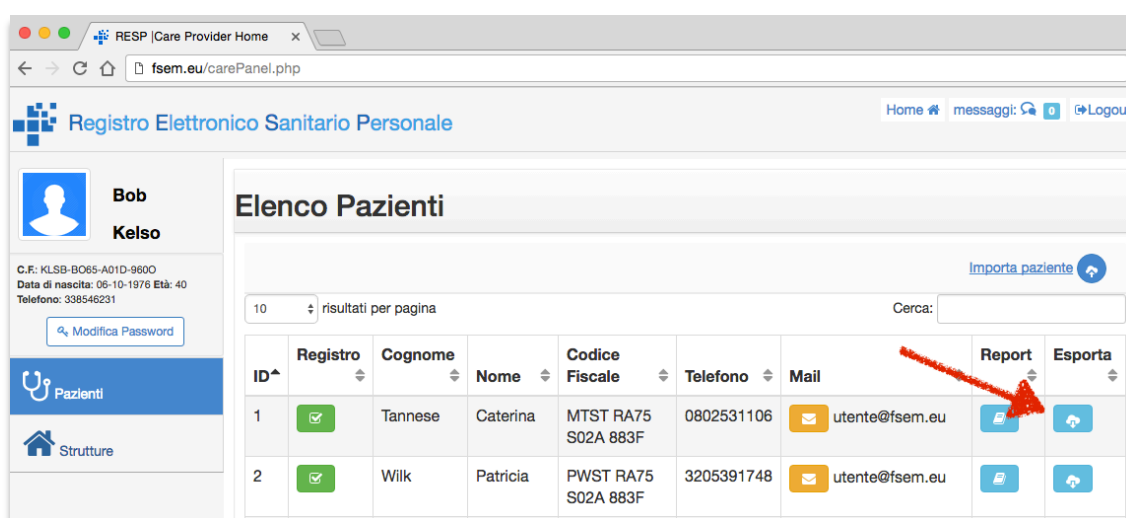
⁴ RFC 2046 Multipurpose Internet Mail Extensions – sezione 4.5.1

⁵ modificando l'header è possibile *suggerire* il nome del file al browser, ma ciò non toglie che il client potrà rinominare l'archivio a suo piacimento.



(Figura 4.26: pulsante di esportazione della cartella sanitaria visualizzato dal paziente)

Dal punto di vista del **care provider**, il pulsante è visualizzato appena esso effettuerà il login nel registro sanitario; è stato posizionato accanto a quello per la generazione del report sanitario nella lista dei pazienti gestiti dal care provider, figura 4.27.



(Figura 4.27: pulsante di esportazione del paziente visualizzato dal care provider)

4.6.2. Importazione di una cartella

Per il processo di importazione di una cartella sanitaria (file zip) si è dovuto procedere in maniera inversa rispetto all'esportazione. Lo script, prendendo in input l'archivio zip, necessita di estrarre tutte le risorse al suo interno e successivamente effettuare delle richieste di creazione mediante le classi che gestiscono le API. L'unico stakeholder che può eseguire l'operazione di importazione di una cartella in RESP è un *care provider*, il paziente invece non può usufruire di questa funzionalità.

Il modulo per l'importazione riceve dall'interfaccia del registro sanitario il **file zip** caricato dall'utente e l'**id del care provider** che sta effettuando l'operazione di importazione. Utilizzando nuovamente la classe `ZipArchive` di PHP vengono letti tutti i file all'interno dell'archivio, per poi immagazzinarli in un array associativo simile a quello illustrato precedentemente (figura 4.24). La prima risorsa che verrà inserita nel sistema è *Patient*, così da creare il paziente all'interno del nuovo sistema. In seguito alla creazione del paziente nel nuovo registro sanitario, lo si lega al care provider che effettua l'importazione della sua cartella nel sistema; per assegnare il paziente al care provider viene utilizzata la tabella del database `careproviderpaziente`. Quest'ultima associa l'id del paziente appena creato con l'id del care provider ottenuto in input nel modulo di importazione. Dopo aver creato correttamente il paziente nel registro sanitario, si passa all'importazione delle altre risorse dell'archivio, l'algoritmo per questa operazione è illustrato nella figura 4.28.

```

foreach($xml_resource as $key => $values) {
    foreach($values as $single_resource) {
        // creo un contenuto xml della risorsa che verra' poi usato
        // per richiamare la POST ed inserire i dati nel sistema
        $xml_content = simplexml_load_string($single_resource);
        unset($xml_content->id);

        $resource = new ResourceFactory($key);
        $resource->postData($xml_content->asxml());
    }
}

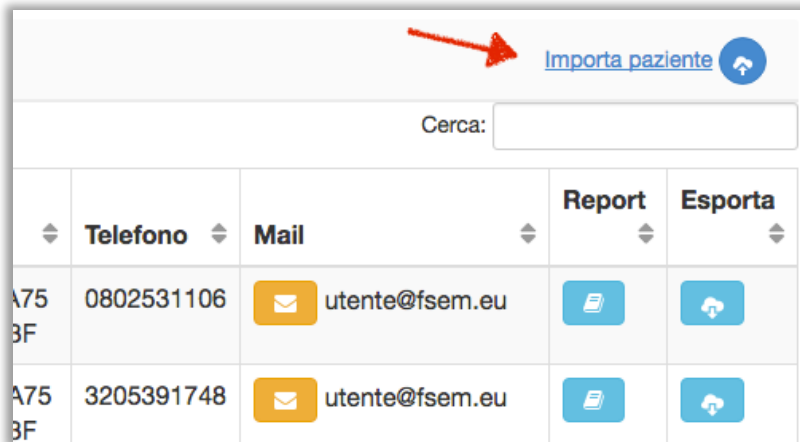
```

(Figura 4.28: frammento di codice per l'importazione delle altre risorse nel sistema)

Con le stringhe ottenute dall'array associativo si costruiscono dei documenti XML, rimuovendo l'elemento `id` dalla risorsa (perché nella POST non deve essere specificato in quanto sarà assegnato dal server). Si istanzia poi la classe `ResourceFactory` passando al costruttore il tipo della risorsa da creare; questo è ottenuto dall'array associativo. Infine si esegue il metodo `postData` per l'effettiva creazione della risorsa nel sistema. Quando lo script termina, il paziente e le risorse standardizzate, presenti all'interno dell'archivio, verranno correttamente inseriti nel registro sanitario.

Per quanto riguarda l'interfaccia grafica, l'unica modifica apportata al registro riguardava ancora una volta la pagina successiva al login del care provider. È stato aggiunto il pulsante “Importa paziente” sopra la lista dei pazienti di un care provider (figura 4.29); quando cliccato, il browser richiede la selezione di un file presente sul computer dell'utente, una volta fatto ciò questo verrà automaticamente caricato nel sistema eseguendo lo script di importazione discusso poc'anzi. Se il file non è un archivio zip verrà sollevata una eccezione dallo script di importazione e sarà mostrato un messaggio d'errore all'utente. Il messaggio di errore potrà comparire anche se l'archivio

zip non presenta al suo interno le effettive risorse esportate da un altro registro sanitario.



(Figura 4.29: pulsante per l'importazione di un paziente)

È stato utilizzato un altro script lato client in linguaggio JavaScript per l'acquisizione dell'archivio zip. Lo script prende in input, dal *form* HTML della pagina, il file zip selezionato dall'utente e l'id del care provider necessari al modulo di importazione nel web service (figura 4.30).

```
$('#upload_patient').on('change', function() {  
    var file_data = $("#upload_patient").prop("files")[0];  
    var form_data = new FormData();  
    form_data.append("patient_file", file_data)  
    form_data.append("careprovider", $('#careprovider_id').val());  
});
```

(Figura 4.30: lettura dal form HTML dei valori del file e l'id del care provider)

Verrà poi effettuata una richiesta POST allo script lato server di importazione della cartella come in figura 4.31.


```

$.ajax({
  url: 'formscripts/importPatient.php',
  type: "POST",
  data: form_data,
  contentType: false,
  cache: false,
  processData: false,
  success: function(data) {
    alert(data);
    window.location.reload();
  }
});
}); // chiusura dell'event handler

```

(Figura 4.31: richiesta POST allo script di importazione mediante ajax)

Per il caricamento della cartella è stata utilizzata la tecnica **AJAX** che ha permesso di effettuare la richiesta al server in maniera *asincrona*. Se l'importazione avviene correttamente, nella pagina attualmente visitata dall'utente saranno mostrati i nuovi dati del paziente appena importato.

Si è potuto notare come, grazie al *refactoring* del codice delle API, sia stato possibile realizzare questo meccanismo di portabilità delle cartelle sanitarie tra diversi RESP.

4.7. Ulteriori miglioramenti al web service FHIR

Concludendo la parte relativa alla progettazione in questo lavoro di Tesi, saranno discusse altre funzionalità introdotte nel web service FHIR del RESP; inoltre, saranno illustrate le classi che hanno permesso lo sviluppo del *tier* di visualizzazione dei dati per il pattern architetturale Model-View-Controller accennato nei precedenti paragrafi. Queste migliorie hanno permesso di adeguare sempre più il web service alle specifiche presenti nello

standard FHIR e hanno favorito una separazione della logica di presentazione dei dati dalla logica di business.

4.7.1. Realizzazione del tier View

Avendo discusso delle classi DAO, che hanno permesso l'interfacciamento con il database, e illustrato le classi come `ResourceFactory` o il gestore delle API (in `index.php`) per la realizzazione della logica di business, verranno mostrate adesso le classi che hanno permesso la creazione del livello di presentazione dei dati.

È stata creata una classe `ResourceView`, contenente due metodi statici, `display_raw` e `display_html`. Questa classe viene utilizzata dal gestore delle API per mostrare al client, che effettua la richiesta al web service, la risorsa presente nel database oppure i messaggi di errore, se dovessero essere sollevate delle eccezioni.

Il primo metodo statico della classe prende in input come parametro il **contenuto** della risorsa e il **formato** con cui è codificata. I formati accettati dal metodo statico sono XML, JSON e HTML; se viene passato al metodo uno dei primi due formati verrà visualizzata al client la risposta modificando l'*header* `Content-Type` rispettivamente in `application/xml+fhir` o `application/json+fhir`. Se la risorsa è in JSON verrà visualizzato un **OperationOutcome** con il messaggio di errore che comunica che il server⁶

⁶ Come verrà illustrato successivamente, è supportata attualmente solo una codifica lato client della risorsa in JSON, effettuata con il linguaggio JavaScript.

non può codificare la risorsa in quel formato. Nel caso in cui il formato della risorsa specificato come parametro è HTML allora:

- se il client effettua la richiesta da **browser web** verrà visualizzata solo la parte narrativa della risorsa quindi in formato tabellare;
- se il client effettua la richiesta HTTP con un linguaggio di programmazione o un *tool* ad esempio **cURL** o **Advanced REST Client**, sarà visualizzata la risorsa in forma XHTML con la relativa struttura dati.

Il secondo metodo statico che fornisce la classe `ResourceView` è `display_html`. Questo metodo prende in input:

- il nome della risorsa (Es. *Patient*, *Organization*, etc.);
- l'id effettivo della risorsa;
- il contenuto codificato della risorsa;
- una variabile *flag* impostata a TRUE o FALSE che specifica se la pagina da visualizzare sarà un errore (**OperationOutcome**).

Il metodo viene richiamato ancora una volta dal gestore delle API (script `index.php`) e permette di migliorare la visualizzazione nel browser della risorsa richiesta del client. Questa schermata è stata già illustrata nella figura 4.8 del paragrafo 4.2 e sotto forma di errore nella figura 4.17 del paragrafo 4.4. Solo attraverso questo metodo saranno disponibili al client dei pulsanti che forniscono delle operazioni aggiuntive da compiere sulla risorsa. Tipicamente questo metodo sarà richiamato quando il client effettua una richiesta di tipo GET senza settare la variabile `_format` nell'URL della risorsa richiesta.

4.7.2. Controllo del formato della risorsa

Secondo il paragrafo RESTful API – Content Types and encodings (2.1.0.6) dello standard FHIR, quando il client richiede la risorsa al web service specificando un URL, oltre al tipo e l'id della risorsa è possibile indicare il formato con cui ricevere la risposta. Nell'URL sarà specificato il formato come segue:

```
http://fsem.eu/fhir/[tipo risorsa]/[id risorsa]?_format=[tipo formato]
```

Al posto delle parentesi quadre potrà essere inserito uno dei formati discussi nel paragrafo precedente (xml, json o html). Riassumendo, nel caso in cui il client effettua una richiesta fornendo un URL come quello appena illustrato allora il gestore delle API richiamerà il metodo `display_raw`; come risposta il client avrà la risorsa con i formati **MIME-type**: `application/xml+fhir` o `application/json+fhir`.

4.7.3. Operazioni aggiuntive sulla risorsa

Come già anticipato, sono state introdotte ulteriori funzionalità nella schermata di visualizzazione della risorsa, sotto forma di pulsanti posizionati in basso rispetto al box dove la risorsa viene visualizzata (figura 4.32).

Il primo pulsante consente di copiare all'interno della *clipboard* tutto il codice della risorsa visibile all'interno nel box. Il client, in questo modo, al posto di selezionare tutto il codice per poi copiarlo e incollarlo, all'occorrenza può cliccare direttamente su questo pulsante.

Il secondo pulsante permette di convertire la risorsa nel formato JSON come in figura 4.33.

FSEM FHIR API • Organization • 2

```
<?xml version="1.0" encoding="utf-8"?>
<Organization xmlns="http://hl7.org/fhir">
  <id value="2"/>
  <text>
    <status value="generated"/>
    <div xmlns="http://www.w3.org/1999/xhtml">
      <table border="2">
        <tbody>
          <tr>
            <td>Nome studio</td>
            <td>Centro Curatutto</td>
          </tr>
          <tr>
            <td>Care provider</td>
            <td>Dott. Kelso Bob</td>
          </tr>
          <tr>
            <td>Via</td>
            <td>panacea, 13</td>
          </tr>
        </tbody>
      </table>
    </div>
  </text>
</Organization>
```



Copia nella clipboard Converti in JSON Validazione risorsa Download file XML

(Figura 4.32: pulsanti per la manipolazione della risorsa)

```
{
  "active": true,
  "resourceType": "Organization",
  "type": {
    "coding": [
      {
        "code": "prov",
        "system": "http://hl7.org/fhir/organization-type"
      }
    ],
    "extension": [
      {
        "valueString": "95",
        "url": "http://localhost/resources/extensions/practitioner-id.xml"
      },
      {
        "valueString": "Studio specialistico",
        "url": "http://localhost/resources/extensions/organization-type.xml"
      }
    ]
  }
}
```

Copia nella clipboard Converti in XML Validazione risorsa Download

(Figura 4.33: risorsa convertita da XML a JSON)

È stato già spiegato che, poiché il web service non supporta la codifica delle risorse in JSON lato server, si è introdotto uno script lato client, sviluppato in linguaggio JavaScript, che prende in input la risorsa nel formato XML e

ne converte il contenuto in JSON direttamente dal browser. Si è scelto di offrire questa funzionalità di conversione perché, se un diverso web service FHIR sfrutta esclusivamente JSON come codifica delle proprie risorse, il client, che utilizza il software RESP, potrà comunque interagire con esso.

Il pulsante “Validazione risorsa” rimanda alla pagina dell’organismo HL7 che contiene il *tool* per la validazione delle risorse FHIR. Questo strumento sarà molto utile per i futuri sviluppatori del fascicolo sanitario che necessitano di testare la creazione di nuove risorse FHIR.

Infine il pulsante “Download file XML” avvia il download della risorsa visualizzata dal browser così da poterla salvare sul computer del client.

5. Conclusioni

5.1. Risultati

L'obiettivo di questo progetto era quello di rendere più ingegnerizzato il sistema per la fruizione delle API, di adeguare il RESP allo standard FHIR per l'interoperabilità con altri sistemi informativi sanitari e infine realizzare una nuova funzionalità per la portabilità dei dati clinici dei pazienti.

5.1.1. Punto di vista degli stakeholder

Data la natura del Registro Sanitario Elettronico Personale, che si ricorda essere un Patient Health Record (cartella clinica personale), la figura del paziente è stato uno dei punti focali di questo lavoro di Tesi. Per fare un esempio, si pensi che, al giorno d'oggi, molte persone continuano ad utilizzare cartelle fisiche piene di referti e analisi, con il conseguente consumo di un ingente quantità di materiale cartaceo, oltre che con il costante rischio di smarrimento. Grazie al meccanismo di **portabilità** realizzato è stata introdotta, per un paziente (utente del RESP), la possibilità portare con sé la propria cartella clinica elettronica, per condividerla successivamente con un *care provider*.

Un caso concreto che si può illustrare è quello del paziente che viene visitato da un medico privato: quest'ultimo potrà effettuare varie operazioni sul RESP, inserendo informazioni nella cartella elettronica sanitaria del paziente. Se il paziente dovesse essere ricoverato in un ospedale, avrà la possibilità di esportare la propria cartella clinica, rappresentata da un file, sui propri

dispositivi per poi copiarla su una memoria di massa (ad esempio una pendrive USB), oppure caricarla in rete sfruttando servizi di *cloud computing*. Il paziente, così facendo, potrà fornire all'ospedale dove è in cura il file esportato dal RESP contenente il suo intero profilo clinico. Se l'ospedale non utilizzasse RESP ma un altro EHR conforme al FHIR, potrebbe comunque interfacciarsi con esso mediante il sistema delle API sviluppato nel lavoro di Tesi; infatti si può ricordare che lo standard *Fast Healthcare Interoperability Resources* dell'organismo HL7 è riconosciuto al livello mondiale. Infine, con le modifiche apportate al RESP e la realizzazione della funzionalità di portabilità, è stato adeguato il sistema al Regolamento europeo discusso nel paragrafo 3.1.

Oltre che il paziente, si è analizzato nel dettaglio anche il comportamento del care provider e ci si è immedesimati nei casi d'uso tipici di un operatore sanitario. Bisogna tenere presente che questi ultimi, in mancanza di un background tecnico informatico approfondito, potrebbero cadere in situazioni d'errore molto più facilmente: per questo motivo, si è fatto in modo di rendere più semplici ed intuitive le interfacce grafiche utilizzate dall'utente finale nel rispetto di alcuni dei più importanti principi di *usabilità*.

5.1.2. Punto di vista dello sviluppatore

Le interazioni con il RESP da parte di sviluppatori sono state rese più chiare e immediate. I miglioramenti apportati alle classi per la gestione delle API hanno permesso di predisporre una solida base di partenza per l'implementazione di **nuove risorse** FHIR. Difatti, come già illustrato nei capitoli precedenti, l'aggiunta delle nuove risorse implica la scrittura di poco

codice oltre ad un riutilizzo di quello già presente. Grazie all'introduzione del meccanismo delle **eccezioni personalizzate** per il controllo degli errori si è potuto sviluppare la risorsa *OperationOutcome*; in questa maniera uno sviluppatore del RESP potrà effettuare operazioni di *debug* molto più velocemente e dettagliatamente. Infine, poiché sono state modificate le interfacce grafiche per la manipolazione delle risorse, lo sviluppo di nuove risulta molto più semplificato ed intuitivo.

5.1.3. Punto di vista di un sistema sanitario esterno

Come già accennato, un sistema elettronico sanitario diverso dal RESP che aderisce allo standard FHIR può interfacciarsi con esso grazie alle *Application Programming Interface* sviluppate. Visitando l'indirizzo `fsem.eu/fhir` e specificando il tipo e l'id della risorsa si potranno ottenere i dati richiesti. Così facendo non si lega il sistema elettronico sanitario al formato utilizzato dal RESP per l'esportazione o l'importazione delle cartelle cliniche.

Si è descritta finora l'operazione per l'acquisizione delle risorse, ma ciò non toglie che un sistema elettronico sanitario, se volesse aggiornare delle informazioni su un paziente o un care provider, lo potrebbe fare grazie alle ulteriori operazioni HTTP implementate cioè: POST, PUT e DELETE rispettivamente per la creazione, l'aggiornamento⁷ e l'eliminazione di nuove risorse.

⁷ PUT, come si è già visto in precedenza, non corrisponde totalmente ad una operazione di aggiornamento, però nel RESP è stata implementata come tale.

Il supporto lato client per la conversione in formato JSON delle risorse permette di avere più compatibilità di formati con altri sistemi sanitari, in quanto un utente del RESP può convertire in JSON il file desiderato e poi passarlo al sistema che fa uso esclusivo di quel formato. Rimane comunque tra gli sviluppi futuri la possibilità supportare il formato JSON da parte del web service così da utilizzarlo per la manipolazione diretta delle risorse se un sistema non utilizzasse il formato XML.

Le tre nuove risorse sviluppate durante la fase di progetto, ovvero *FamilyMemberHistory*, *Organization* e *OperationOutcome*, sono state tutte validate con lo strumento di validazione messo a disposizione dell'organismo HL7. In questo modo si è verificato il risultato dello sviluppo di queste risorse ed è stata controllata la loro adeguatezza allo standard sanitario internazionale.

5.2. Considerazioni finali

L'informatica medica si pone diversi obiettivi, tra cui quello di definire metodi ed architetture per sistemi informativi sanitari, con grande rilevanza data proprio allo sviluppo di una **cartella clinica elettronica**. Dotare le strutture sanitarie di queste tecnologie permetterebbe ai pazienti di avere tutti i dati in un unico posto e gestirli in base alle proprie necessità, mentre un care provider potrebbe disporre di tutte le informazioni adeguate per l'assistenza ad un paziente, anche quando questo viene seguito in diversi ospedali. Sin dalle sue prime versioni, il progetto RESP ha cercato di rispondere alle varie esigenze del settore ospedaliero, soprattutto dal punto di vista della comunicazione tra il paziente e il care provider e della gestione

dei dati clinici. Tuttavia, era necessario proseguire il suo sviluppo concentrandosi sugli aspetti della portabilità dei dati e dell'interoperabilità, nella consapevolezza che la rete sanitaria prevede l'intervento, nel tempo, di entità e organizzazioni diverse. Come si è potuto vedere, affidarsi a standard internazionali risulta una parte imprescindibile della creazione di sistemi atti a questo scopo, specialmente per la compatibilità dei dati e per l'interazione tra diversi sistemi informativi distribuiti. Lo sforzo di questo lavoro di Tesi è stato sia quello di favorire la coesione tra sistemi, sia quello di creare e standardizzare nuove risorse sanitarie riguardanti il paziente, tutto in conformità con lo standard FHIR. Queste migliorie hanno permesso di arricchire la cartella clinica digitale, grazie alla varietà dei servizi e delle funzionalità offerte, e di semplificare il lavoro dei care provider che hanno a disposizione tutte le informazioni necessarie o possono facilmente importarle.

Il raggiungimento dei fini che ci si era prefissati è passato attraverso lo studio e il ricorso ai diversi linguaggi che hanno permesso lo sviluppo delle nuove funzionalità introdotte, tra cui: i linguaggi lato server come PHP per lo sviluppo e le modifiche al web service; i linguaggi lato client, per la validazione e il controllo dei form nelle pagine web, come JavaScript ed il relativo *framework* jQuery; infine i linguaggi di markup per la codifica delle risorse FHIR, tra cui XML e JSON. Si è dovuta studiare l'architettura REST per la realizzazione del web service e per la fruizione dei servizi di elaborazione delle risorse; sono stati inoltre impiegati alcuni dei pattern ingegneristici per rendere il sistema più facile da sviluppare in futuro.

5.3. Sviluppi futuri

Il lavoro per l'adeguamento allo standard FHIR per l'interoperabilità del Registro Sanitario Elettronico Personale non è ancora terminato. Si può inoltre continuare a perfezionare il progetto da un punto di vista ingegneristico per renderlo più efficiente nell'utilizzo e flessibile al cambiamento, in particolare per quanto concerne l'aggiunta di nuove risorse. Saranno illustrate adesso alcune delle funzionalità e degli sviluppi futuri che si possono prevedere per il progetto RESP.

Per quanto concerne il modulo per la portabilità delle risorse, purtroppo al momento l'importazione può avvenire solo se il paziente non è registrato in un registro sanitario, ossia non si possono aggiornare i dati importandoli. Si può, quindi, pensare di effettuare un ulteriore controllo sulla presenza del paziente, in modo da aggiornare solo le informazioni effettivamente modificate nella cartella digitale durante l'importazione.

I vantaggi di avere nuove risorse standardizzate nel RESP sono stati già elencati in precedenza, queste contribuiranno all'arricchimento del profilo sanitario digitale di un paziente. Tra le risorse ancora da standardizzare rimangono: le indagini di laboratorio, le procedure terapeutiche, le visite con un medico curante, etc. Le risorse dello standard FHIR sono circa 93 e molte sono in continuo aggiornamento.

Si potrebbero implementare altri servizi nel web service REST tra cui:

- **HISTORY**, per il tracciamento delle modifiche delle risorse nel tempo;

- **CONFORMANCE**, per l'ottenimento di una lista dei servizi disponibili da parte del web service;
- **SEARCH**, per effettuare delle ricerche tra le risorse attualmente presenti nel registro sanitario.

Data la fondamentale importanza della **privacy nell'ambito sanitario**, durante il lavoro sul progetto RESP sono state introdotte delle tecniche per garantire l'accesso alle API (quindi ai dati sensibili del paziente) solo a chi ne ha diritto. Il RESP possiede già un robusto sistema di sicurezza basato su crittografia **AES** e utilizza algoritmi di *hashing* tra cui **SHA512** per la crittazione dei record all'interno del database. Rimangono però da testare i diversi moduli software realizzati per ridurre la possibilità di ricevere violazioni da parte di personale non autorizzato. Ad esempio per la risorsa standardizzata *Patient* (già sviluppata) si potrebbero adottare tecnologie tra cui **SCIM** (System for Cross-domain Identity Management), che consentirebbero la gestione dell'identità del paziente all'interno del RESP e l'esportazione di questa risorsa. Infine l'utilizzo del protocollo **OAuth** (suggerito persino dall'organismo HL7) consentirebbe di fruire delle API in maniera molto più sicura e standardizzata, grazie all'impiego di un sistema di API *token* e del *logging*⁸ delle richieste. L'implementazione di OAuth è di **fondamentale importanza** e in questo lavoro di Tesi si è fatto in modo da predisporre l'architettura in modo che, per introdurre in futuro questo protocollo, basterebbe modificare solo il modulo delle API, lasciando le altre componenti del RESP inalterate nel loro comportamento.

⁸ The OAuth 1.0 Protocol – RFC 5849 Aprile 2010

6. Riferimenti

6.1. Bibliografia

- Pradeep Sinha, Gaur Sunder, Prashant Bendale, Manisha Mantri, Atreya Dande, *Electronic Health Record: Standards, Coding Systems, Frameworks, and Infrastructures*, ed. John Wiley & Sons (2013)
- Enrico Coiera, *Guide to Health Informatics, Third Edition* (Capitolo 19), ed. CRC Press (gennaio 2013)

6.2. Sitografia

Capitolo 2) Cenni Teorici

- HL7: *FHIR*, <https://hl7.org/fhir/>
- HL7: *RESTful API*, <https://www.hl7.org/fhir/http.html>
- HL7: *Resource Index*, <https://www.hl7.org/fhir/resourcelist.html>
- Wikipedia: *Fast Healthcare Interoperability Resources*,
https://en.wikipedia.org/wiki/Fast_Healthcare_Interoperability_Resources
- Wikipedia: *Health Level 7*,
https://en.wikipedia.org/wiki/Health_Level_7
- Wikipedia: *DataPortability*,
<https://it.wikipedia.org/wiki/DataPortability>
- Wikipedia: *XML*, <https://it.wikipedia.org/wiki/XML>
- Wikipedia: *Representational State Transfer*,
https://it.wikipedia.org/wiki/Representational_State_Transfer

- RKB Healthcare,
http://www.rkb-hc.com/uploads/1/1/3/8/11385341/3234904_orig.png

Capitolo 3) Stato dell'arte

- Garante per la protezione dei dati personali:
<http://www.garanteprivacy.it/regolamentoue>

Capitolo 4) Progetto

- HL7: *Resource FamilyMemberHistory*,
<https://www.hl7.org/fhir/familymemberhistory.html>
- HL7: *Resource Organization*,
<https://www.hl7.org/fhir/organization.html>
- HL7: *Resource OperationOutcome*,
<https://www.hl7.org/fhir/operationoutcome.html>
- HL7: *Content Types and encodings*,
<https://www.hl7.org/fhir/http.html#mime-type>
- Wikipedia: *SNOMED CT*,
https://en.wikipedia.org/wiki/SNOMED_CT
- Wikipedia: *Ajax (programming)*,
[https://en.wikipedia.org/wiki/Ajax_\(programming\)](https://en.wikipedia.org/wiki/Ajax_(programming))
- Wikipedia: *Model-View-Controller*,
<https://it.wikipedia.org/wiki/Model-View-Controller>

- IETF: *Octet-Stream Subtype*,
<https://tools.ietf.org/html/rfc2046#section-4.5.1>
- IETF: *RFC 2183*, <https://www.ietf.org/rfc/rfc2183>
- GitHub: *FHIR XML to/from JSON converter in Lua*,
<https://github.com/vadi2/fhir-formats>

Capitolo 5) Conclusioni

- Wikipedia: *Cloud computing*,
https://it.wikipedia.org/wiki/Cloud_computing
- Wikipedia: *OAuth*,
<https://it.wikipedia.org/wiki/OAuth>
- IETF: *The OAuth 1.0 Protocol*, <https://tools.ietf.org/html/rfc5849>