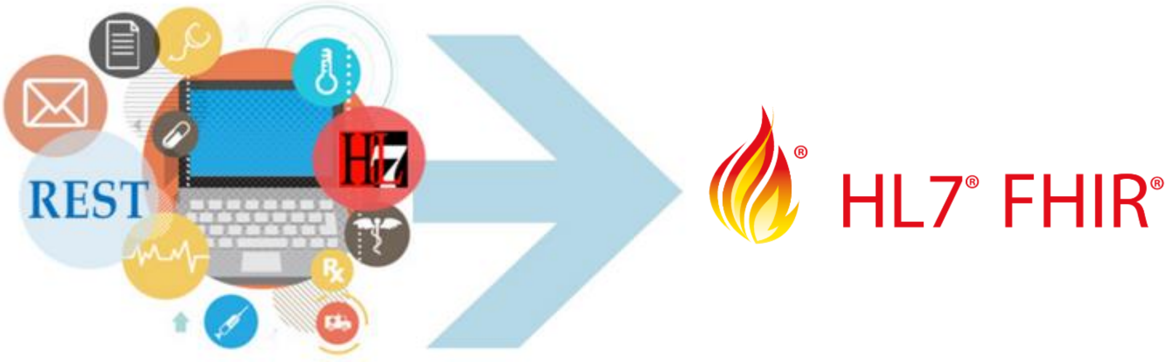
Haute Ecole de Gestion & Tourisme
Hochschule für Wirtschaft & Tourismus

Filière Informatique de gestion
Studiengang Wirtschaftsinformatik

Bachelor Thesis 2017

Web application of physiological data based on FHIR



Student : Camille Pellaton

Professor : Michael Schumacher

Submitted the 2nd of August 2017

Sources of the illustrations of the title page

http://www.techno-soft.com/assets/images/HL7-FHIR.jpg

https://i.embed.ly/1/display/resize?key=1e6a1a1efdb011df84894040444cdc60&url=http%3A%2F%2Fhl7.se%2Fwp-content%2Fuploads%2F2016%2F02%2Ffhirbanner-2.png

Acknowledgments

Thanks to:

- Michael Ignaz Schumacher, professor at the HES-SO Valais-Wallis and collaborator at the Institute of Information Systems of the HES-SO Valais-Wallis, for proposing this Bachelor thesis and for his guidance during the realization of the thesis.

- Fabien Dubosson, collaborator at the Institute of Information Systems of the HES-SO Valais-Wallis for his precious advice, guidance and information.

- My family and friends for their support during this work.

Abstract

This paper works toward implementing a prototype demonstrating some of the capabilities of the FHIR specification. The specification requires a clear understanding of its different components in order to be successfully implemented, therefore the primary concern of this work is to understand and analyse FHIR's concepts.

The research conducted in this work revealed that FHIR is a well-designed specification, based on a powerful data model and technologies. Therefore, it sould help solving the interoperability issues of the healthcare eco-system. It has also been pointed that since FHIR is a recent standard, many of its uses and benefits are still to be discovered. Moreover, FHIR integrates well in the current health information technology context since it can be used in addition to existing standards.

The analysis of the currently used technologies to implement FHIR evidenced that HAPI FHIR java library was the most suited library to develop a FHIR-based application in this work. Indeed, HAPI provides more capability and a better usability than alternative libraries. Implementing FHIR in a web application helped determined that FHIR does enable fast implementation and thus, can potentially drastically decrease the cost of systems integration in the healthcare industry.

The objectives of this Bachelor thesis have been reached in 396 hours using the Scrum framework. Scrum features have facilitated the development of this work, in particular the creation of a product backlog and structuring the development with sprints in which user stories from the product backlog are broken into tasks.

Keywords*: FHIR, Healthcare, Interoperability, Specification, Web Application*

Table of Contents

List of Tables

List of Figures

List of listings

List of Abbreviations

API : Applications Programming Interface

BLOB : Binary Large OBject

CDA : Clinical Document Architecture

CRUD : create, read, update and delete

DB : database

DSTU : Draft Standards for Trial Use

EHR : electronic health record

FHIR : Fast Healthcare Interoperability Resources, : Fast Healthcare Interoperability Resources

HIMSS : Healthcare Information and Management Systems Society

HIT : health information technology

HL7 : Heath Level Seven

HL7 v2 : Health Level Seven version 2

HTTP : HyperText Transfer Protocol

IHE : Integrating the Healthcare Enterprise

ISO : International Organization for Standardization

IT : information technology

JSON : JavaScript Object Notation

MRN : Medical Record Number

MSH : Message Header

NoSQL : Not only SQL

OBR : Observation Request

OBX : Observation

PID : Patient Identification

RDBMS : relational database management system

RDF : Terse Resource Description Framework

REST : representational state transfer

RIM : Reference Information Model

SMART : Substitutable Medical Applications, Reusable Technologies

SOA : service-oriented architecture

SPA : single-page application

SSL : Secure Sockets Layer

STU : Standard for Trial Use, : Standard for trial use

Turtle : Terse RDF Triple Language

UML : Unified Modeling Language

URL : Uniform Resource Locator

v2 : version 2

v3 : version 3

XDS : HE Cross-Enterprise Document Sharing

XML : Extensible Markup Language

Introduction

As healthcare is progressively becoming paperless, the need for the exchange of clinical data between health information systems is increasing. Standards have provided health institutions with the possibility to widely implement the exchange of data within their information systems. For instance, allowing the patients data to be shared amongst different services of a hospital or across hospitals and organizations. As of today, the most implemented interchange standard within health information systems is the Health Level Seven version 2 (HL7 v2) standard. V2 has significantly improved the way medical systems can exchange data and how they can use the exchanged information in multiple innovative manners. Indeed, more suitable care can now be provided and the probability of making diagnostic errors has decreased. Nevertheless, HL7 v2 has shown its limits regarding implementation costs and interoperability resulting in a lack of efficient electronic data exchange across institutions, for example, between private practices and hospitals.

A recent survey (Transcend Insights, 2017) on the patients' expectations of medical information sharing indicates that 64% of patients use a medical device or a mobile application to monitor their health, out of which, 71% think that this data could be helpful to practitioner and should be recorded in their health history. Because of the initiative to include the patients into their own health and because of the need for cheaper and more interoperable solutions, in 2014, the healthcare industry has seen emerged a new HL7 standard called the Fast Healthcare Interoperability Resources (FHIR).

FHIR is full of promises: real-time data access, usage of current web technologies, freely available, a focus set on the implementation to enable the creation of fast and cost-effective solutions and ensure semantic interoperability. FHIR has the capabilities to improve and broaden the existing analytical process of clinical data.

This works aims to understand the capabilities provided by the FHIR specification as well as testing some of those capabilities to evaluate FHIR against its promises. This paper focuses firstly on analyzing and presenting the different concepts of the FHIR specification by analyzing the context amongst which it has been created and the previously developed HL7 standards to understand the reasons behind its conception. In a second phase, existing tools to implement FHIR-based solutions will be analyzed and evaluated in order to determine which technology is the most suited to develop a web application based on FHIR.

The final part of this paper aims to present the application prototype realized and to provide an insight of the creation of a FHIR-based client application. The application prototype illustrates some of the capabilities of FHIR, evidenced by the research conducted in this work, with the practical implementation of FHIR in a web application.

1 Entering FHIR's world

The Fast Healthcare Interoperability Resources (FHIR) is a standard for the electronic exchange of health data. It has been designed by the Heath Level Seven (HL7) organization. The purpose of this specification is to enable a better interoperability in healthcare information systems. FHIR is a web-based toolbox, set of rules and frameworks defining how the exchange of clinical data should be excecuted between different healthcare applications and systems (HL7 International, n.d.-v).

FHIR makes interoperability faster and easier to implement than existing standards, which leads to being able to create much cheaper solutions than by using alternatives.

FHIR is based on previous HL7 standards which it combines and improves: HL7 version 2 (v2), version 3 (v3) and the Clinical Document Architecture (CDA) (HL7 International, n.d.-v).

The FHIR specification is entirely based on the concept of resources. A resource is a small piece of information that represents a real-world entity such as a patient, a location or an appointment. For its purpose of exchanging clinical data, FHIR determines four different paradigms to be used:

- The representational state transfer (REST) Applications Programming Interface (API)
- Messaging
- Documents
- Services (HL7 International, n.d.-g).

1.1 FHIR's origin

1.1.1 HL7 International

The HL7 International is a non-profit standards creating organization. It is globally well represented since it has members in more than 50 countries (Rouse & Sutner, 2015). The best way to understand its purpose is to reading its mission statement - "To provide standards that empower global health data interoperability." (HL7 International, n.d.-a) - and its vision: "A world in which everyone can securely access and use the right health data when and where they need it." (HL7 International, n.d.-a).

1.1.2 The interoperability issues

Nowadays, hospital information systems are composed of a variety of systems provided by different vendors. Those systems must work together by exchanging their data so that laboratorians, physicians, nurses and other actors of healthcare can, for example, access the electronic data relative to a patient. Furthermore, clinical data must be shared across institutions such as between a hospital system and a private practice system to enable a practitioner to consult his patient's data. Another example is the transmission of a prescription from a physician to a pharmacist.

Interoperability is a key point in making great sustainable healthcare information systems. In the health informatics context, interoperability means that separate medical systems or software can communicate without compromising the data, and this, without an extreme amount of effort (Datar, 2016). With interoperable health systems, the actors implicated in patient care use systems that can not only exchange data, but can also interpret and use the data in a meaningful manner to provide better medical care. Healthcare Information and Management Systems Society's (HIMSS) definition of interoperability includes "[…] the ability of health information systems to work together within and across organizational boundaries in order to advance the effective delivery of healthcare for individuals and communities." (HIMSS, 2013).

Communication is the key to providing good healthcare. A technical[1] and semantical[2] interoperability guarantee that systems will be able to communicate, but the supreme goal of interoperability is to reach a clinical interoperability level, meaning that clinicians from different teams are able to transfer patients' care and this, by providing seamless care (Benson & Grieve, 2016).

Improving interoperability in healthcare has been the mission of the HL7 international over the past three decades. HL7 has released different standards helping solve the interoperability issue in the healthcare industry: HL7 v2, v3 and FHIR.

[1] Moving data from one system to another, not caring about the content's meaning
[2] Data is understood and can be interpreted by both systems

1.1.3 Why standards are needed

Standards are present in most aspects of the everyday life, to mention one in many, the notable International Organization for Standardization (ISO) that regulates from sanitary conditions of food to formats for time representation. Standards permit defining agreed-upon rules and definitions to ensure, for instance, safety, quality and communication. In health information technology (HIT), standards provide rules to define a common language and way of exchanging health data between and across organizations (Shaver, 2007). Standards enable the exchange of data by providing exchange formats for the data, data elements (the content to be shared) linked in data models and terminologies. Thus, standards are indispensable to achieve healthcare interoperability.

1.1.4 HL7 v2

The most notorious standard HL7 has developed to date is the HL7 v2. Considered as one of the base standards for the HL7 FHIR standard, HL7 v2 is a messaging standard for exchanging health data. Unlike FHIR, v2 is a database query language that allows to send messages to forward or retrieve electronic health information (Rouse & Sutner, 2015). According to HL7, this standard is the most implemented in healthcare worldwide. In HL7 v2, segments represent entities of the real world and hence resemble FHIR’s resources (HL7 International, n.d.-b).

1.1.4.1 The need to integrate

HL7 v2 was designed to facilitate the creation of interfaces between healthcare systems. Before, custom interfaces had to be implemented in order for systems to communicate together. Custom interfaces require plenty of programming on the sending and the receiving applications, therefore these interfaces are expensive. As a result, only few custom interfaces were implemented in hospitals (Shaver, 2007).

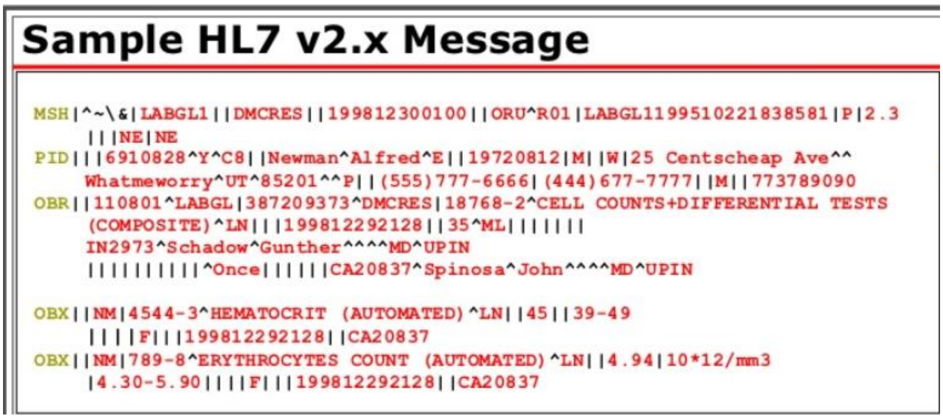
The HL7 approached the integration problematic by creating a standard defining 80 percent of the interfaces between systems and leaving 20 percent up for clients’ customization.

HL7 v2 was thought to reduce integration costs thus enabling a wider range of institutions to build or migrate to integrated information systems. The use cases of the HL7 v2 standard provide an insight into why such integrated systems are needed (Wood, n.d.):

- Doctors need to be able to stay connected to specialists during patient's care
- Enable a connection between doctors and pharmacists in order to reduce errors
- Different hospitals need to connect their hospital information systems(HIS)
- The electronic health record (EHR) of a patient should be retrievable by any health application within the hospital information systems.

1.1.4.2 The messages

V2 determines the structure of the exchanged messages between systems but also the content of the data to be sent. Each of those messages are sent upon a trigger event as, for instance, a patient's appointment (iNTERFACEWARE Inc., 2008).



Sample HL7 v2.x Message

```
MSH|^~\&|LABGL1||DMCRES||199812300100||ORU^R01|LABGL1199510221838581|P|2.3
    |||NE|NE
PID|||6910828^Y^C8||Newman^Alfred^E||19720812|M||W|25 Centscheap Ave^^
    Whatmeworry^UT^85201^^P||(555)777-6666|(444)677-7777||M||773789090
OBR||110801^LABGL|387209373^DMCRES|18768-2^CELL COUNTS+DIFFERENTIAL TESTS
    (COMPOSITE)^LN|||199812292128||35^ML|||||||
    IN2973^Schadow^Gunther^^^^MD^UPIN
    ||||||||||^Once||||||CA20837^Spinosa^John^^^^MD^UPIN

OBX||NM|4544-3^HEMATOCRIT (AUTOMATED)^LN||45||39-49
    ||||F|||199812292128||CA20837
OBX||NM|789-8^ERYTHROCYTES COUNT (AUTOMATED)^LN||4.94|10*12/mm3
    |4.30-5.90||||F|||199812292128||CA20837
```

Figure 1 – HL7 v2 Message (HL7 International, 2010)

The above message sends observations and results to a requiring system. This type of message could be sent from a laboratory system to a hospital system, for example. The content being sent is known by interpreting the type of message contained in the message header segment (MSH), here the code is ORU^R01 which indicates that the content of the message contains observations and results. R01 represents the trigger event.

Some segments are required in this type of messages, including the message header, the patient identification (PID) and the observation request (OBR). Some are optional and repeatable like the observation (OBX) segment. V2 messages are mostly composed of simple primitive types such as strings or numbers (Corepoint Health, n.d.).

As noticeable, v2 messages lack of human readability and the syntax is not simple to apprehend.

1.1.5 HL7 version 3

HL7 v3 messaging standard attempts to improve over the HL7 v2 standard. The main issues of v2 that v3 tried to solve were the lack of a persistent data model, the lack of users and roles definitions and important flexibility of the standard. These issues lead the v2 to be somewhat inconsistent and incomplete. The v3 standard addressed these issues, it is therefore not backward compatible since it is a more detailed and rigid framework.

This newer standard defines a clear data model and precise definitions. It allows less flexibility and thus makes it easier for the implementation of interfaces. Therefore, v3 messaging standard is a cheaper option to implement and to maintain than the v2 standard.

To be more detailed and consistent, HL7 v3 was built as a model-based methodology and established the Reference Information Model (RIM) of the healthcare information domain, a set of data types and a strong domain terminology (Beeler, n.d.). The RIM is an information model providing classes and elements to be used to compose v3 messages.

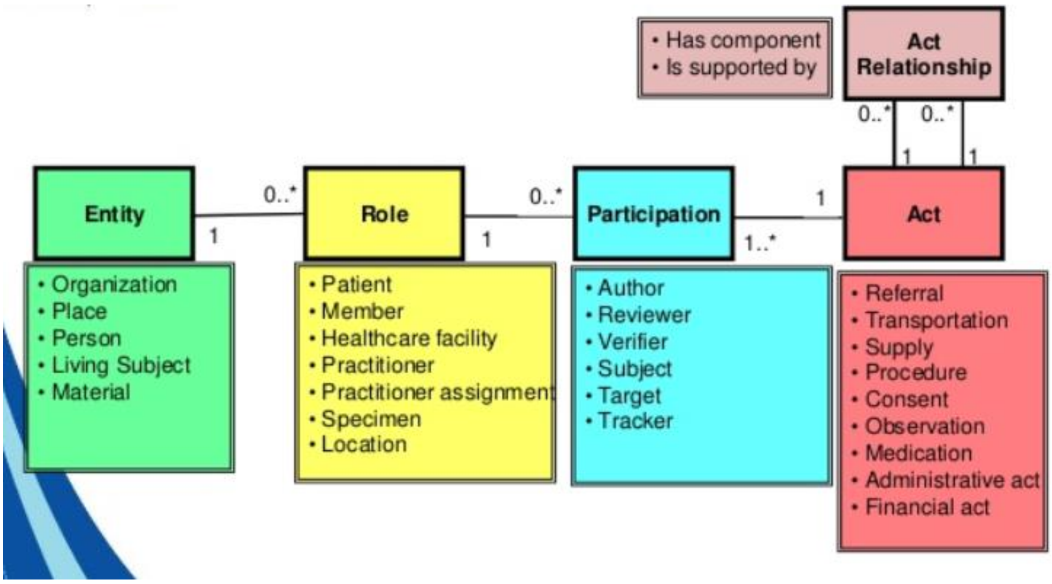


Figure 2 - HL7 v3 RIM (Nawanan Theera-Ampornpunt, 2015, p. 40)

The image above presents the v3 RIM's main classes. It is structured by five key concepts: Entity, Role, Participation, Act and Act Relationship. Data content of HL7 v3 messages are defined by this information model (HL7, 2002, p. 1). The Act class stem from the purpose of the v3 messages of documenting the actions taken in a patient's care.

```
1.  <POLB_IN224200 ITSVersion="XML_1.0" xmlns="urn:hl7-org:v3"
2.  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3.  <id root="2.16.840.1.113883.19.1122.7" extension="CNTRL-3456"/>
4.  <creationTime value="200202150930-0400"/>
5.  <!-- The version of the datatypes/RIM/vocabulary used is that of May
6.  2006 -->
7.  <versionCode code="2006-05"/>
8.  <!-- interaction id= Observation Event Complete, w/o Receiver
9.  Responsibilities -->
10. <interactionId root="2.16.840.1.113883.1.6"
11. extension="POLB_IN224200"/>
12. <processingCode code="P"/>
13. <processingModeCode nullFlavor="OTH"/>
14. <acceptAckCode code="ER"/>
15. <receiver typeCode="RCV">
16. <device classCode="DEV" determinerCode="INSTANCE">
17. <id extension="GHH LAB" root="2.16.840.1.113883.19.1122.1"/>
18. <asLocatedEntity classCode="LOCE">
19.       <location classCode="PLC" determinerCode="INSTANCE">
20.       <id root="2.16.840.1.113883.19.1122.2" extension="ELAB-3"/>
21.       </location>
22. </asLocatedEntity>
23. </device>
24. </receiver>
25. <sender typeCode="SND">
26. <device classCode="DEV" determinerCode="INSTANCE">
27. <id root="2.16.840.1.113883.19.1122.1" extension="GHH OE"/>
28. <asLocatedEntity classCode="LOCE">
29.       <location classCode="PLC" determinerCode="INSTANCE">
30.       <id root="2.16.840.1.113883.19.1122.2" extension="BLDG24"/>
31.       </location>
32. </asLocatedEntity>
33. </device>
34. </sender>
35. <!-- Trigger Event Control Act & Domain Content -->
36. </POLB_IN224200>
```

Listing 1 - V3 message sample, code source: **(Spronk, 2007)**

Messages are sent using the Extensible Markup Language (XML) representation. The code snippet above shows a sample message. Line 19 is a good example of the linkage to the RIM, the *classCode* "PLC" refers to the *Place* entity defined in the RIM.

Even though it has many advantages over the HL7 v2 standard, HL7 v3 will likely not replace it. Since it is not backward compatible, any migration to v3 from v2 will have considerable costs. When creating v3 applications and if a communication with v2 applications is needed, the new v3 applications will need to be able to speak not only v3 but also v2. This double cost of implementation discourages healthcare providers from adopting the HL7 v3 standard (Johnson, 2012, p. 20).

1.2 The need for FHIR

Since HL7 v2 left too much space for customization, a lot of variants exist and caused a major problem for integration of data between health systems and institutions. Additionally, the v2 has a very particular and unique syntax that makes it tedious to implement. Another thing to point at is its lack of security and authentication capabilities, very problematic in healthcare to include distant applications in the exchange (e.g. applications for patients). These reasons cause problems for the v2's interoperability and make it unsuited for mobile use (Datar, 2016).

On the other hand, HL7 v3 has not managed to conquer an important share of the healthcare market. Not only since it is also too expensive to implement but because it is too complex. Yet another problem of the v3 it that it was not thought to be implemented directly:

> *The philosophy is that the models are starter templates that will then be constrained for use by the implementing region or party. This places a large burden of work on the implementer. It also virtually guarantees that no two real-world implementations will be compatible and blocks international software vendors and solution providers from being able to offer consistent solutions across regions and languages.* (Bender & Sartipi, 2013)

HL7 v2's and v3's lack of interoperability, consistency and youth encouraged HL7 to think of a better standard that could benefit from HL7 standards' years of experiences. To the increasing demand for a more modern and easier solution for interoperability, HL7 responded with a trendy web-based specification: FHIR. Moreover, there was a real need for clinical data mobile access that HL7 addressed by offering real-time APIs access.

However, institutions will not likely invest in changing their working information systems to FHIR-based systems in the near future. FHIR has to co-exists with older standard and "[…] will be a standard that can be used to translate between standards and allow those systems that are legacy systems using an older standard to still exchange information within the ecosystem of health information data" (Landi, 2016).

As an example, the figure below shows how FHIR could work along with HL7 v2 standard. The HIS sends a v2 message to the Corepoint Integration Engine, the data is then pushed on the FHIR repository and made available to mobile applications ('Use cases of FHIR', n.d.).

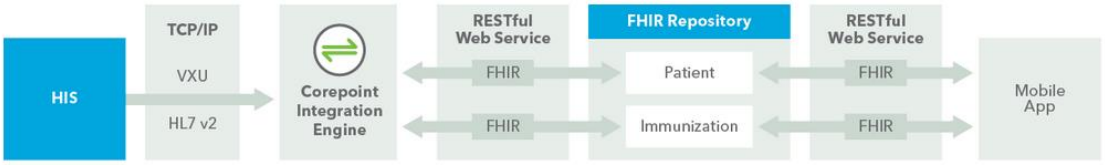


Figure 3 – FHIR and v2 ('Use cases of FHIR', n.d.)

In the medical framework, several different actors that need to exchange confidential data are found. Not only physicians, laboratorian, pharmacists, specialists and administrative members but, more recently, patients tend to be included into their own healthcare. This also led to the need for more flexible and mobile-friendly interoperability solutions (Hay, n.d.).

FHIR gives HL7 a fresher look with its REST architecture. It is also very familiar since it is using well known technologies like JavaScript Object Notation (JSON), XML, HyperText Transfer Protocol (HTTP), Secure Sockets Layer (SSL) and OAuth.

FHIR opens the discussion of the creation of an application platform providing medical applications, like the one existing for Android. This way, FHIR applications created for one system could be made available for millions of others, significantly reducing medical applications development costs. The increasing competitivity would encourage vendors to propose better programs (Landi, 2016).

Another great fact about FHIR is that it is freely available for implementers and numerous open-source implementations can be found, which is not the case for previous HL7 standards. This follows the original idea of FHIR: creating a fast (to implement) specification. It also solves the main problem of healthcare institutions of the cost of interoperability solutions.

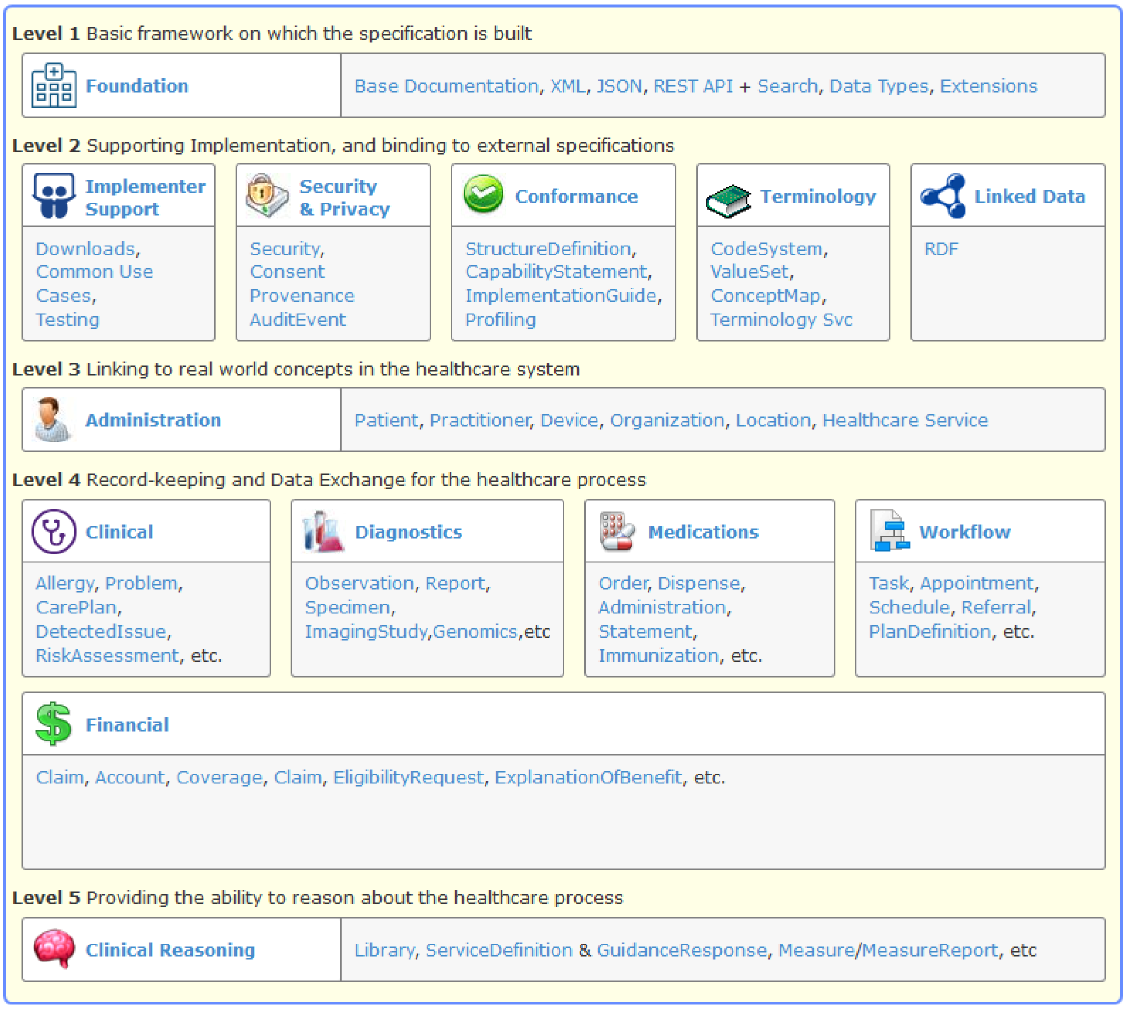
2 The FHIR standard

This chapter focuses on selected FHIR specification's concepts. Since FHIR is a recent standard, it is important to analyze its concepts and capabilities to be able to implement it in a solution and to fully identify its potential benefits. The FHIR specification is vast, it could not be covered entirely in the context of this work, therefore only the main concepts and the one needed to implement FHIR have been selected for this analysis. At first, it might be difficult to build a mental visualization of FHIR, consequently a mind map[3] has been appended to this paper. Since not much documentation is available on the FHIR specification, the main source of documentation for this analysis is the FHIR specification itself.

The current version of the FHIR specification is the version 3.0.1 of the Release 3 Standard for Trial Use (STU). This latest release succeeds to the Draft Standards for Trial Use (DSTU) 2 and the DSTU 1.

HL7 has categorized the specification in distinct modules grouped in five levels, as illustrated in the following figure. The next subchapters introduce some of the most important modules composing FHIR. The modules gather and structure FHIR's features, elements and concepts.

[3] Appendix II



Level 1 Basic framework on which the specification is built

Foundation: Base Documentation, XML, JSON, REST API + Search, Data Types, Extensions

Level 2 Supporting Implementation, and binding to external specifications

Implementer Support: Downloads, Common Use Cases, Testing

Security & Privacy: Security, Consent Provenance AuditEvent

Conformance: StructureDefinition, CapabilityStatement, ImplementationGuide, Profiling

Terminology: CodeSystem, ValueSet, ConceptMap, Terminology Svc

Linked Data: RDF

Level 3 Linking to real world concepts in the healthcare system

Administration: Patient, Practitioner, Device, Organization, Location, Healthcare Service

Level 4 Record-keeping and Data Exchange for the healthcare process

Clinical: Allergy, Problem, CarePlan, DetectedIssue, RiskAssessment, etc.

Diagnostics: Observation, Report, Specimen, ImagingStudy,Genomics,etc

Medications: Order, Dispense, Administration, Statement, Immunization, etc.

Workflow: Task, Appointment, Schedule, Referral, PlanDefinition, etc.

Financial: Claim, Account, Coverage, Claim, EligibilityRequest, ExplanationOfBenefit, etc.

Level 5 Providing the ability to reason about the healthcare process

Clinical Reasoning: Library, ServiceDefinition & GuidanceResponse, Measure/MeasureReport, etc

Figure 4 – FHIR resources structure (HL7 International, n.d.)

The first level, the *foundation* module, explains the specification's key elements: the resources and the four exchange paradigms defined. Level two supports the implementation, the *Implementer Support* module is presented in the work by describing FHIR *Common Use*, the *Security* and the *Conformance* elements. Level three contains administrative resources and is therefore treated in the "Resources" subchapter 2.1. Finally, the *Workflow* module will be presented.

2.1 Resources

FHIR defines several different types of resources to save and exchange data. Resources are the core concept of the FHIR specification. They are the modular representations of clinical and administrative entities. Resources can be grouped together in what is called a "bundle".

HL7 classifies the resources in 6 main sections (HL7 International, n.d.-r):

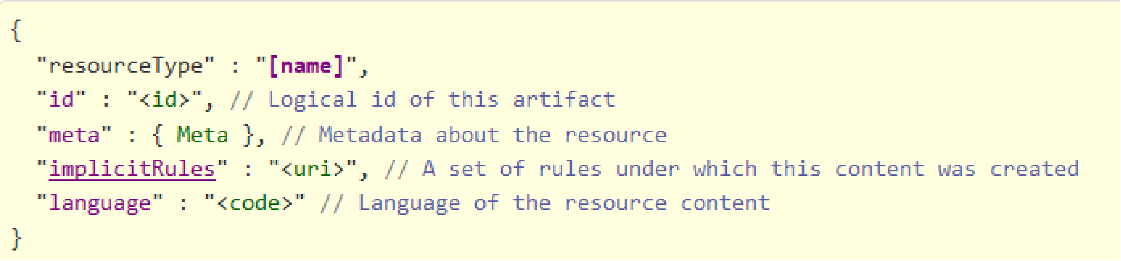
- **Clinical**: containing all the resources representing the content of a clinical record
- **Identification**: comprises the entities involved in the healthcare
- **Workflow**: includes the different tasks of a care process
- **Financial**: concerns the billing and payment ability of the specification
- **Conformance**: encompass the resources used during development, testing and managing FHIR
- **Infrastructure**: belong in the level 1 module "Foundation", it contains resources providing useful base functionalities such as a "binary" resource to store non FHIR resource format, it also defines the "bundle" resource used to group several different resources.

Some resources are typically clinical whilst others are business entities such as the resources of the section "Financial" or "Administration". These are made for the end users to display and manipulate them, whereas some are used by the systems to give specific information such as the resources in the "Conformance" section. An exhaustive list of all the resources can be found on the HL7 FHIR's website[4].

As previously said, FHIR is built on this concept of resource. They can either be represented in XML, in JSON or in Terse Resource Description Framework (RDF) Triple Language (Turtle) format for the exchange (HL7 International, n.d.-f).

Some attributes (also called elements) of a resource are required whereas others are optional. A resource must have a known identity such as a Uniform Resource Locator (URL), a type, structured data items and a version identifier. Some optional elements have been defined for every resource: an identity, meta data, a language and a reference to implicit rules. These, represented in JSON on the figure 5, are just some elements that a resource may contain, it could also contain other resources, text, extensions and many other attributes (HL7 International, n.d.-q).

[4] at https://www.hl7.org/fhir/resourcelist.html



```
{
  "resourceType" : "[name]",
  "id" : "<id>", // Logical id of this artifact
  "meta" : { Meta }, // Metadata about the resource
  "implicitRules" : "<uri>", // A set of rules under which this content was created
  "language" : "<code>" // Language of the resource content
}
```

Figure 5 – base resource (HL7 International, n.d.)

Extending the base resource, FHIR defines a domain resource that is then extended by all the resources except “Bundle”, “Parameter” and “Binary”. Each other resource type respects this domain resource’s structure and is redefined to its specific needs. The domain resource adds the definition of a narrative (human readable text) that indicates the content of the resource. The Domain resource enables resources to include related resources and extensions.

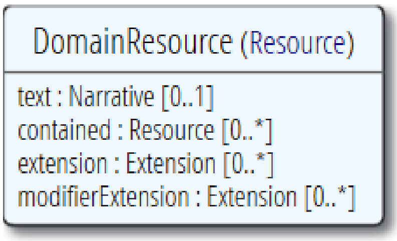

DomainResource (Resource)
text : Narrative [0..1]
contained : Resource [0..*]
extension : Extension [0..*]
modifierExtension : Extension [0..*]

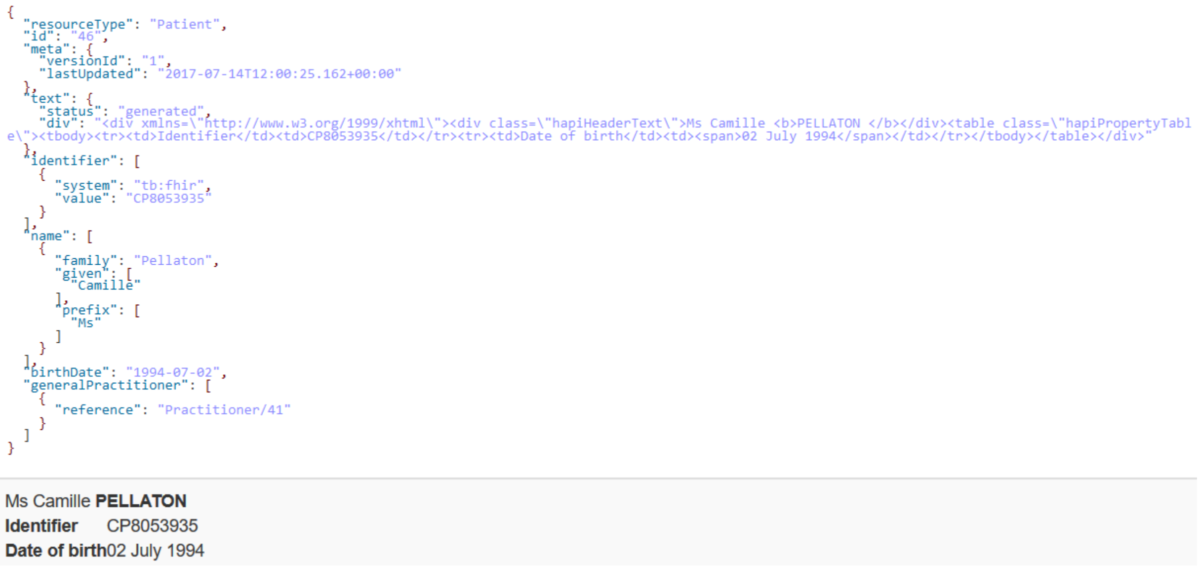
Figure 6 – UML diagram of a Domain Resource (HL7 International, n.d.)

FHIR defines the usage of all the resources contained in the specification, for instance, the Patient resource is used according to HL7 for “Demographics and other administrative information about an individual or animal receiving care or other health-related services.” (HL7 International, n.d.-n).

The figure 7 shows a JSON representation of a Patient resource. This resource includes a reference to a Practitioner resource in order to assign a general practitioner to this patient. Below the code, the narrative representation is displayed. The narrative is contained in the “div” attribute of the Patient resource.

The “id” attribute represents the FHIR id of the resource attributed by the FHIR server upon the resource’s creation. This id must be unique and cannot be changed. On the other hand, the “value” attribute of the “identifier” element represents the Medical Record Number

(MRN) used by medical institutions to identify their patients in their systems. This id can be manipulated in the resource and is therefore used for the MRN which is subject to change (HL7 International, n.d.-n).



```
{
  "resourceType": "Patient",
  "id": "46",
  "meta": {
    "versionId": "1",
    "lastUpdated": "2017-07-14T12:00:25.162+00:00"
  },
  "text": {
    "status": "generated",
    "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\"><div class=\"hapiHeaderText\">Ms Camille <b>PELLATON </b></div><table class=\"hapiPropertyTabl
e\"><tbody><tr><td>Identifier</td><td>CP8053935</td></tr><tr><td>Date of birth</td><td><span>02 July 1994</span></td></tr></tbody></table></div>"
  },
  "identifier": [
    {
      "system": "tb:fhir",
      "value": "CP8053935"
    }
  ],
  "name": [
    {
      "family": "Pellaton",
      "given": [
        "Camille"
      ],
      "prefix": [
        "Ms"
      ]
    }
  ],
  "birthDate": "1994-07-02",
  "generalPractitioner": [
    {
      "reference": "Practitioner/41"
    }
  ]
}
```

Ms Camille **PELLATON**
Identifier CP8053935
Date of birth02 July 1994

Figure 7 - Patient resource in JSON and HTML narrative

2.1.1 Validation

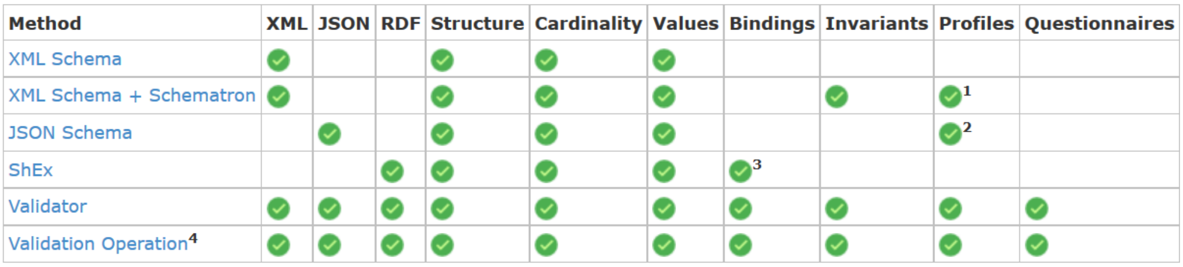
FHIR supports the validation of resources. Validating resources is the process of checking that the resources are valid according to the following aspects (HL7 International, n.d.-x):

- **Structure**: ensure that the content of the resource is defined by the specification
- **Cardinality**: validate the correctness of the cardinality of all properties
- **Values Domains**: ensure that values of properties are conformant to the set of rules defined for the specific types
- **Coding**: check that codes and displays used are valid
- **Invariant**: ensure the invariants[5], constraints are properly followed
- **Profiles**: check that the resource conforms to the profile[6] referenced
- **Questionnaire**: verify that a response to a Questionnaire is valid against the Questionnaire

[5] https://www.hl7.org/fhir/conformance-rules.html#constraints

[6] See chapter 2.4.1

The following table shows the different validation tools suggested by the specification. The aspects supported for the validation are indicated for each tool.



Method	XML	JSON	RDF	Structure	Cardinality	Values	Bindings	Invariants	Profiles	Questionnaires
XML Schema	✓			✓	✓	✓				
XML Schema + Schematron	✓			✓	✓	✓		✓	✓[1]	
JSON Schema		✓		✓	✓	✓			✓[2]	
ShEx			✓	✓	✓	✓	✓[3]			
Validator	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Validation Operation[4]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Figure 8 - validation tools (HL7 International, n.d.-x)

2.2 The four Paradigms

FHIR supports four paradigms to exchange data: RESTful API, Services, Messaging and Documents (HL7 International, n.d.-g). This paper focuses mostly on the RESTful approach since, to this day, it is the most used by implementers and it is also the one chosen to be implemented in the application prototype that has been developed during this work.

2.2.1 RESTful API

FHIR can be referred to as a RESTful specification, indeed it defines "a uniform connector interface" (Fielding, 2000), known as FHIR RESTful API, enabling a stateless client-server communication. FHIR core specification supports the Level 2 of the Richardson REST Maturity model, Level 3 is supported with the usage of extensions.

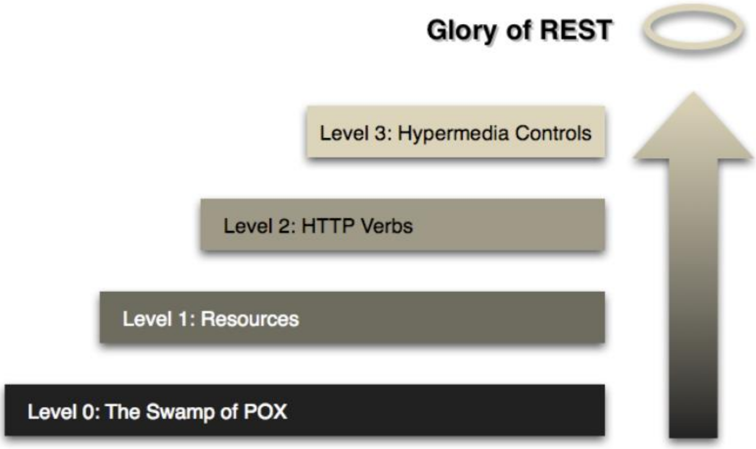


Figure 9 - REST maturity model **(Martin, n.d.)**

FHIR supports Level 2 since it is using HTTP for interactions, it defines resources that can be requested individually and it exploits HTTP verbs (GET, PUT, POST, DELETE) for its operations. The hypermedia controls of Level 3 can be supported by adding extra data in resources containing links to further interactions (HL7 International, n.d.-h).

2.2.1.1 REST

The representational state transfer is an architecture style granting interoperability between systems over the internet. The REST idea is to use a simple way of communicating between machines, which is commonly HTTP. HTTP-based REST uses HTTP commands to perform create, read, update and delete (CRUD) operations (Elkstein, n.d.).

REST is a style of software architecture and not a standard. It defines how networked resources are to be addressed and, on a bigger scale, how the web works. It was first presented by Roy Fielding in his doctoral thesis (cited in Douglas K Barry, n.d.). It is a popular choice for the development of web-services. Applications following REST principles are referred to as RESTful applications whereas APIs adhering to REST principles are called RESTful APIs (Rouse, n.d.).

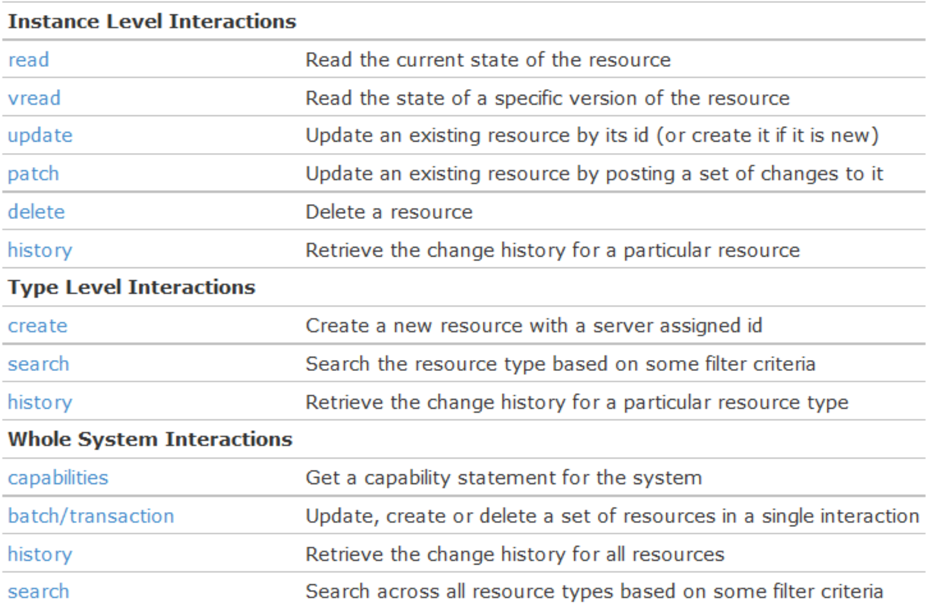
REST defines five main principles:

- **A uniform interface:** the interface between clients and servers uses a well-defined set of methods to manipulate resources. Each resource must be identified in the requests with, for example, an URI. Each message must contain enough information for the server to process it. Resources are separated from the representations send to the clients (XML, JSON…) ('What is REST?', n.d.).
- **Stateless interactions:** the server does not hold the client's state between requests. All the information needed by a request must be contained by it.
- **Cacheable:** responses can be cached by clients
- **Client-server:** the client and the server are separated, the client is responsible for the requests and the server for the response and data storage
- **Layered system:** a client cannot tell whether it is connected to an intermediate or an end server (Servage, 2013).

2.2.1.2 FHIR RESTful API

The API defines a set of operations that can be performed on FHIR resources. Servers should provide a Capability Statement resource describing which operations and resources they support (HL7 International, n.d.-h).

FHIR defines the following interactions:



Instance Level Interactions	
read	Read the current state of the resource
vread	Read the state of a specific version of the resource
update	Update an existing resource by its id (or create it if it is new)
patch	Update an existing resource by posting a set of changes to it
delete	Delete a resource
history	Retrieve the change history for a particular resource
Type Level Interactions	
create	Create a new resource with a server assigned id
search	Search the resource type based on some filter criteria
history	Retrieve the change history for a particular resource type
Whole System Interactions	
capabilities	Get a capability statement for the system
batch/transaction	Update, create or delete a set of resources in a single interaction
history	Retrieve the change history for all resources
search	Search across all resource types based on some filter criteria

Figure 10 - FHIR API interactions (HL7 International, n.d.-h)

Interactions may be used with the following form: **VERB [base]/[type]/[id] {?_format=[mime-type]}**. [base] represents the Service Base URL where all the resources defined by the interface are located. The usage of HTTPS is optional but production exchange of healthcare information shall use SSL (HL7 International, n.d.-h). The following subchapters present the most current API interactions.

2.2.1.2.1 Read

The read interaction is used to request the current content of a resource from the server (HL7 International, n.d.-h). For example, if a client requests the current state of a patient resource with “20207” as id from a FHIR RESTful test server, the read interaction will take this form:

GET https://fhirtest.uhn.ca/baseDstu3/Patient/20207?_format=json

2.2.1.2.2 Vread

The vread interactions allow retrieving a specific state of a resource. It is essentially a read operation which has the version of the resource specified. Vread is performed using a HTTP GET command as follow: **GET [base]/[type]/[id]/_history/[vid] {?_format=[mime-type]}.** Servers supporting the FHIR API should support resources versioning, if not, it should be explicitly declared in their capability statement (HL7 International, n.d.-h).

2.2.1.2.3 Update

Update is performed with: **PUT [base]/[type]/[id] {?_format=[mime-type]}.** It simply creates a new version of an existing resource on the server. The request body is a resource with the same id as the resource in the URL, if not the server will respond with a HTTP 400 error code.

A conditional create has been defined by FHIR allowing clients to update resources based on other criteria than their ids. The command is the following: **PUT [base]/[type]?[search parameters].** HTTP PATCH command can be used to update resources when one does not need or have permission to update an entire resource (HL7 International, n.d.-h).

2.2.1.2.4 Delete

The delete interaction erases a resource. It is performed by the HTTP DELETE command knowing the id of the resource: **DELETE [base]/[type]/[id].** The resource will no longer be found through search operations. It is possible to specify some types of resources that cannot be deleted and it is also possible not to enable deleting resources at all. As for updating, a conditional delete exists and allows deleting resources based on other parameters than their ids (HL7 International, n.d.-h).

2.2.1.2.5 Create

Creating a resource is achieved by an HTTP POST command. The id of the resource will be allocated by the server. The create URL has the form: **POST [base]/[type] {?_format=[mime-type]}.** The request body must contain a FHIR resource. Once the resource has been successfully created, the server returns the new id and version id of the resource.

It is possible to use a conditional create so that a resource is created only if no such resource can be found on the server (HL7 International, n.d.-h).

2.2.1.2.6 Search

The search interactions can be operated with either the GET or the POST command. The GET command is formed as **GET [base]/[type]{?[parameters]{&_format=[mime-type]}}**, whilst the POST command is formed as **POST [base]/[type]/_search{?[parameters]{&_format=[mime-type]}}.** Those two search queries look for all resources of a given type on the server. Servers shall support both versions of the search interaction because of how GET and POST might be handled by proxies and user agents.

Several search parameters apply to all FHIR resources: _content, _id, _lastUpdated, _profile, _query, _security, _tag, _text and _filter. For instance, to search a patient based on its id the following query is used: **GET [base]/Patient?_id=23.** When searching for one resource, a bundle will be returned by the server, therefore when the client is certain of the existence of the resource on the server, a read operation is more appropriate (HL7 International, n.d.-s).

The FHIR specification lists all the search parameters defined for each type of resource directly on each resource's page (e.g. the patient page[7]).

2.2.2 Services

FHIR data exchange can be conceived as a service-oriented architecture (SOA). SOA is a well-known architecture in the software industry. Fundamentally, it can be considered as being a set of exposed services. As in real life, a provider offers a service to a consumer. The consumer obtains the result of the service without further considerations on how the result was achieved behind the scenes. In the information technology (IT) domain, services which represent business functions are made available to other systems through standard protocols (Phelps, 2007).

To do so, the web-services technology is the most commonly used communication protocols. Web-services are exposed on the internet by providing applications, then the consuming applications can send and receive data by invoking those networked services (Doglas, n.d.).

[7] At https://www.hl7.org/fhir/patient.html

Even though FHIR resources are created to best fit the RESTful API approach, it is possible to design an exchange architecture following the services approach. FHIR defines the two following SOA approaches:

- *FHIR + WS* represents FHIR implementations using the web services stack as the communication protocol instead of REST. This would include use of FHIR resources as payload parameters in SOAP calls, for example*
- *FHIR + SOA Pattern illustrates the impact of applying interaction patterns, exception handing and role definition, guided by SOA practices, atop an implementation technology (SOAP, REST, or others).*

  (HL7 International, n.d.-u)

FHIR could benefit from interesting SOA capabilities such as its low-coupling framework, its expertise in transactional integrity within distributed systems and its consistency in exception handling (HL7 International, n.d.-u). FHIR and SOA implementations combined may be a wise choice to make since "SOA provides guidance for how components interact, how to partition responsibilities, and how to manage workflows among different parts of systems, all of which have potential utility in FHIR implementation settings." (HL7 International, n.d.-u).

It is important to bear in mind that FHIR resources have been thought to be exploited by a RESTful implementation and implementing FHIR with SOA could engender a great amount of additional work.

2.2.3 Messaging

FHIR also defines a messaging exchange framework which enables sending routed messages from systems to systems to exchange content. This mechanism can either be implemented on the RESTful API or use an independent messaging system.

This exchange was not designed to complete the RESTful API, but only to allow implementers to choose the way they want to exchange their content. It resembles HL7 v2 exchange mechanism and may be interesting for v2 familiarized developers.

This exchange is event driven, a message is sent when triggered by an event such as a new prescription for a patient or a newly scheduled meeting. A message is in fact a bundle of resources which first resource is of type MessageHeader (HL7 International, n.d.-j). The appendix I of this work contains a FHIR message example to illustrate this point. Indeed, it is easier to understand how messages are formed by going through a concrete instance.

2.2.4 Documents

FHIR delineates a document exchange framework allowing systems to transmit documents. This framework is strongly based on HL7 CDA standard.

Documents are bundles of resources of type “Document” that contain a Composition resource as their first entry, followed by the resources creating the actual content of the document. The resources forming the content of the message are referenced by the Composition. The bundle gathers the whole content of the document into one unique XML or JSON document (HL7 International, n.d.-d). The Composition resource is described by HL7 as:

> *A set of healthcare-related information that is assembled together into a single logical document that provides a single coherent statement of meaning, establishes its own context and that has clinical attestation with regard to who is making the statement. While a Composition defines the structure, it does not actually contain the content: rather the full content of a document is contained in a Bundle, of which the Composition is the first resource contained.* (HL7 International, n.d.-c)

A FHIR document is immutable and has a fixed representation created either by humans, organizations or devices. It can be exchanged and stored by systems. FHIR documents can be signed to ensure the document’s authenticity. A binary resource may be included in the document to contain the stylesheet that will be used to nicely render the document for human readers.

FHIR also provides a document reference resource for documents that are not assembled in FHIR (pre-existing documents) (HL7 International, n.d.-d).

2.3 Security and privacy

FHIR not being a security protocol, it does not provide any security features. Nevertheless, as security and privacy of data are crucial elements in healthcare FHIR has been designed to work with many existing security solutions. FHIR describes how to protect a FHIR server with access controls and authorizations and how to keep tracks of the events performed with audit logging. FHIR advocates that implementations use existing security standards towards providing the encryption of communications, avoidance of information leaks on error occurrence, deactivation of scripts injection into narrative resources, logs records to track abnormal access behaviors and control of data access (HL7 International, n.d.-t).

As a matter of fact, a production FHIR system will require a security system to be established. The following figures presents some integration possibilities.

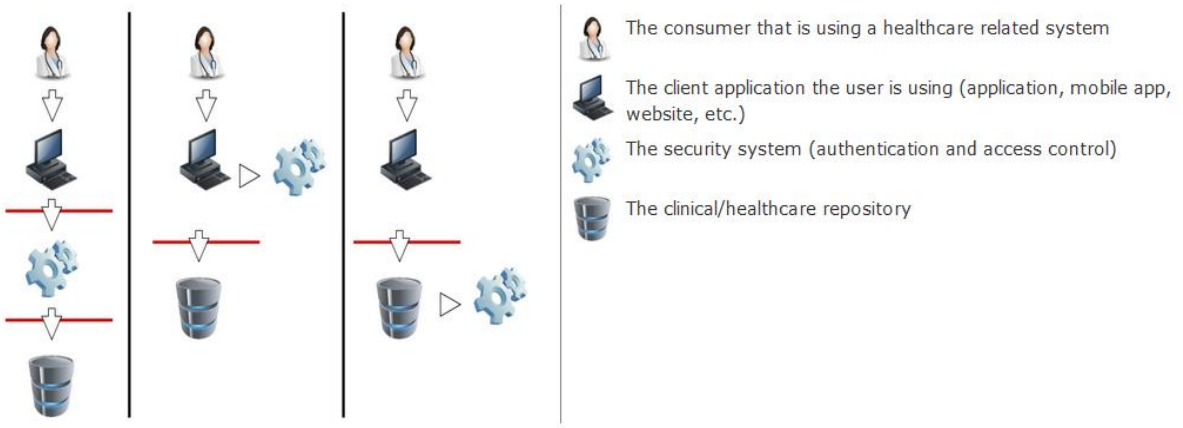

The consumer that is using a healthcare related system
The client application the user is using (application, mobile app, website, etc.)
The security system (authentication and access control)
The clinical/healthcare repository

Figure 11 - FHIR security subsystem (HL7 International, n.d.-t)

2.4 Conformance

2.4.1 Profiles

The FHIR specification defines foundations, on which many different solutions can be created, by providing base resources, frameworks and APIs. Since FHIR is intended to be used by a wide industry, adjustments and adaptations may occur so that the solution fits the implementers' need. An item or package of these adaptations is known as a "profile", while the process of adapting FHIR to specific use case's need is referred to as "profiling". A profile is in fact a "normal" FHIR resource that can be handled by any FHIR server (HL7 International, n.d.-p).

Common adaptations would be defining rules about (HL7 International, n.d.-p):

- The usage of the resources specifying weather they are used or not and what components have been added or restricted
- The API features available and how to use them

Since FHIR is used in the vast industry of healthcare, many similar or overlapping adaptations may exist in different countries, institutions or for different vendors. FHIR implementations and resources shall declare the profiles they conform to. Servers list the profiles they support and check clients' conformances. Resources declare their conformances in their metadata by referencing a profile by its URL (HL7 International, n.d.-p).

2.4.2 Profiling artifacts

FHIR provides a set of artifacts for profiling, they are presented in the table below.

Artifact	Description
Implementation Guide (IG)	A coherent and bounded set of adaptations that are published as a single unit. Validation occurs within the context of the Implementation Guide
Package	A group of related adaptations that are published as a group within an Implementation Guide
Conformance Resource	A single resource in a package that makes rules about how an implementation works. Described in the following point.

Table 1 - FHIR profiling artifacts adapted from (HL7 International, n.d.-p)

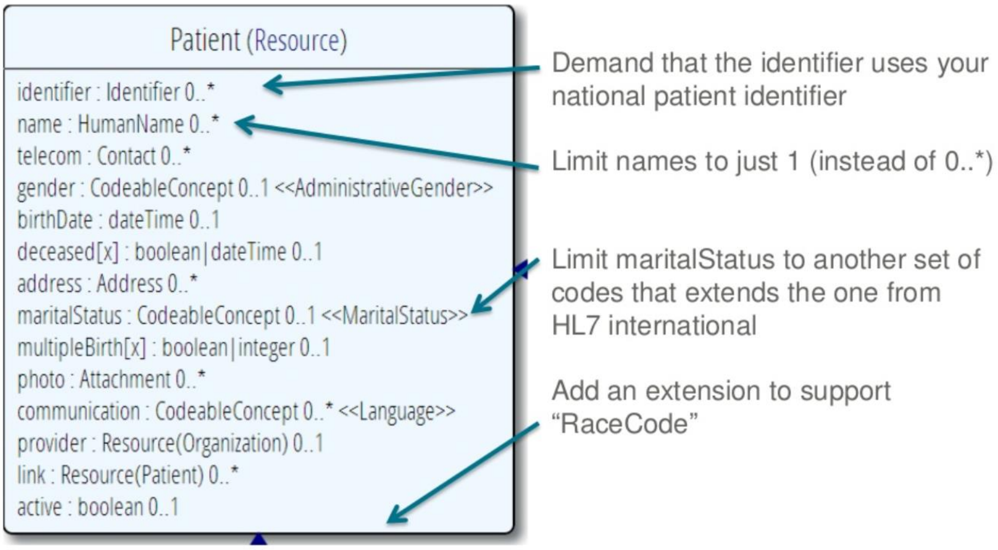
2.4.3 Conformance resources

FHIR defines the need and usage of a conformance resource:

Typically, Implementation Guides both restrict and extend APIs, resources and terminologies. FHIR provides a set of resources that can be used to represent and share the *decisions that have been made, and allows implementers to build useful services from them. These resources are known as the conformance resources.* (HL7 International, n.d.-p)

With these conformance resources, implementers can perform several tasks on the APIs: indicate what calls can be made and in which cases using the *CapabilityStatement* resource, offer additional search operations and search parameters that are not defined by the core specification (*OperationDefinition* and *SearchParameter* resource). They can also define how a resource, extension or datatype is used in the implementation with the *StructureDefinition* resource. The *StructureDefinition* resource allows to describe the elements used, indicate the one that are not and the extensions a resource or datatype supports (HL7 International, n.d.-p). This resource can also be used to describe how a particular resource references a *ValueSet resource* defining the code values used and supported by that particular resource (e.g. LOINC codes).

For instance, a profile (Conformance resource) can be used to restrict a resource element's cardinality or to limit the possible values it might take. It also allows to add new properties to a resource by extending it. The following image shows some interesting usage of a profile.


Patient (Resource)
identifier : Identifier 0..*
name : HumanName 0..*
telecom : Contact 0..*
gender : CodeableConcept 0..1 <<AdministrativeGender>>
birthDate : dateTime 0..1
deceased[x] : boolean | dateTime 0..1
address : Address 0..*
maritalStatus : CodeableConcept 0..1 <<MaritalStatus>>
multipleBirth[x] : boolean | integer 0..1
photo : Attachment 0..*
communication : CodeableConcept 0..* <<Language>>
provider : Resource(Organization) 0..1
link : Resource(Patient) 0..*
active : boolean 0..1

Demand that the identifier uses your national patient identifier

Limit names to just 1 (instead of 0..*)

Limit maritalStatus to another set of codes that extends the one from HL7 international

Add an extension to support "RaceCode"

Figure 12 - Resource profiling **(Hay, 2014, p. 3)**

2.5 Use cases

FHIR describes several use cases that can be implemented with the FHIR standard. The following two use cases are presented: Personal Health Record (PHR) viewer and document repository. It is important to note that FHIR will be implemented to meet many other use cases, therefore the examples provided below are in no way an exhaustive listing of FHIR's uses.

2.5.1 PHR viewer

An EHR system exposes a FHIR server that allows client applications to perform search and read operations on the Patient resource and on a set of clinical resources. This use case is illustrated by the schema below.

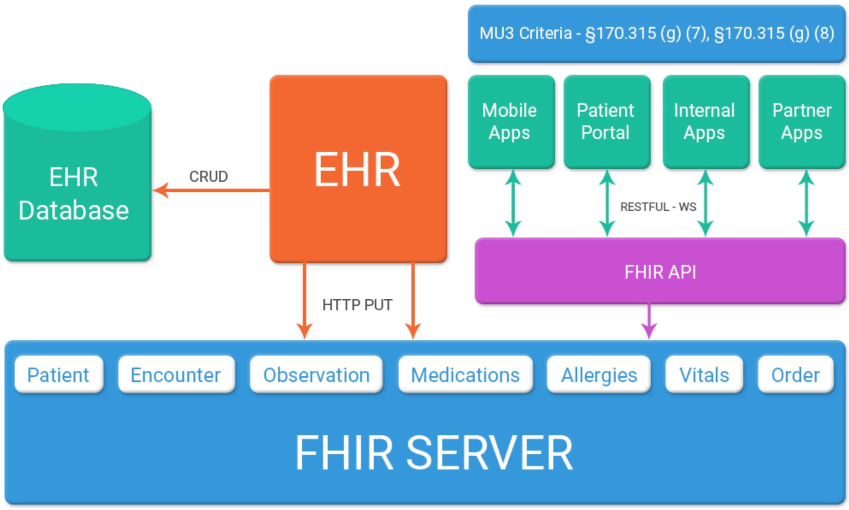


Figure 13 – FHIR integration with EHR (Sangem, 2016)

Further interactions could be allowed to client applications such as update. This use case would typically be defined by a scenario in which patients can have access to their personal health data via a mobile application or a patient portal (HL7 International, n.d.-w).

A simulation of this use case has been implemented in the prototyping phase of this work. The project developed in the context of this work is, indeed, a simulation of the above real-world use case since no EHR system has been integrated in the project. The EHR has been replaced by a web interface allowing the patient medical record's creation, modification and deletion.

2.5.2 Document repository

Integrating medical information from different sources can be done with the build of a patient-centric document repository.

Integrating the Healthcare Enterprise's (IHE) HE Cross-Enterprise Document Sharing (XDS) profile has been adopted as the main framework to create those types of repositories as "XDS allows for a federated system of repositories with a registry to provide coordinated access to the documents" (HL7 International, n.d.-w). FHIR provides the same capabilities as XDS and can be used to create more modern and user-friendly interfaces. Those interfaces can also easily be integrated to existing XDS environment. Furthermore, HL7 and IHE are currently working together to enhance the IHE Mobile access to Health Documents (MHD) profile[8] (IHE, n.d.).

2.6 Workflows

Workflows are the essence of healthcare whether automated or not. Automated workflows improve the medical care in such a manner that many errors can be avoided and simplifies the work of care providers. FHIR is concerned with workflow management and aims to standardize the processes, activities and roles necessary to interoperable workflow executions. FHIR resources can be used in workflows without necessarily using FHIR for the execution although FHIR provides several mechanisms for that purpose. In some cases, the management of the execution of workflows might not even be required, FHIR supports these cases and allows handling them in a simple manner such as shown on Figure 14 (HL7 International, n.d.-y).

FHIR defines a set of resources (activities) that are used within workflows, three categories exist (HL7 International, n.d.-y):

- **Request**: the resource requiring an action to be taken
    - Appointment, Task, MedicationRequest…
- **Events**: resources expressing that an action has been taken
    - Encounter, Observation, SupplyDelivery, DiagnosticReport…
- **Definitions**: resources describing an action that can be done (time-independent)

[8] The profile defines a HTTP interface for XSD-based in order to access medical document from "mobile" applications

- PlanDefinition, Measure, Questionnaire…

2.6.1 Workflow patterns

The simplest workflow architecture using FHIR would be that (HL7 International, n.d.-z):

1. The placer system Perfoms a RESTful create or update interaction on a resource of the receiver RESTful endpoint.
2. The receiver then replies with an HTTP status indicating the state of the request and the response that can be provided in the payload of the HTTP response.

Figure 14 is another example of a workflow architecture. The following steps are defined (HL7 International, n.d.-z):

1. The placer system creates a resource of the request category on the filler FHIR server. This resource possesses a tag element indicating that action has to be taken upon the resource.
2. The filler sends a HTTP status code indicating that the resource has successfully been created.
3. After a certain time, the filler sends an event resource to the placer's resource endpoint. The event resource references the request with a link (basedOn) to identify the resource for which the action has been fulfilled.
4. The placer system replies with a 201 HTTP status code indicating that the resource has been created.

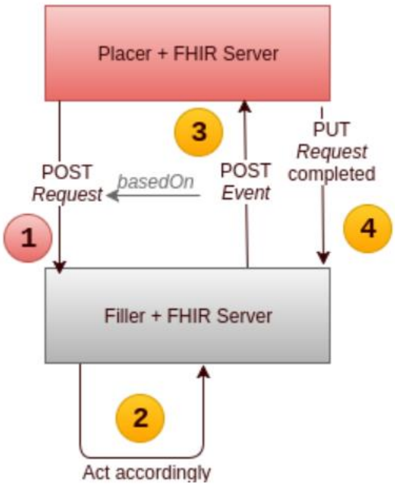


Figure 14 - simple RESTful workflow pattern **(HL7 International, n.d.-z)**

Those two examples illustrate the basic mechanism of FHIR workflow management, in addition, the specification defines more advanced management patterns that manage the execution of workflows. The figure below shows an example of such a pattern. As it can be seen, the state of execution of the workflow is now trackable (tasks completed or not). As the previous example, this pattern manages the workflow using the FHIR API.

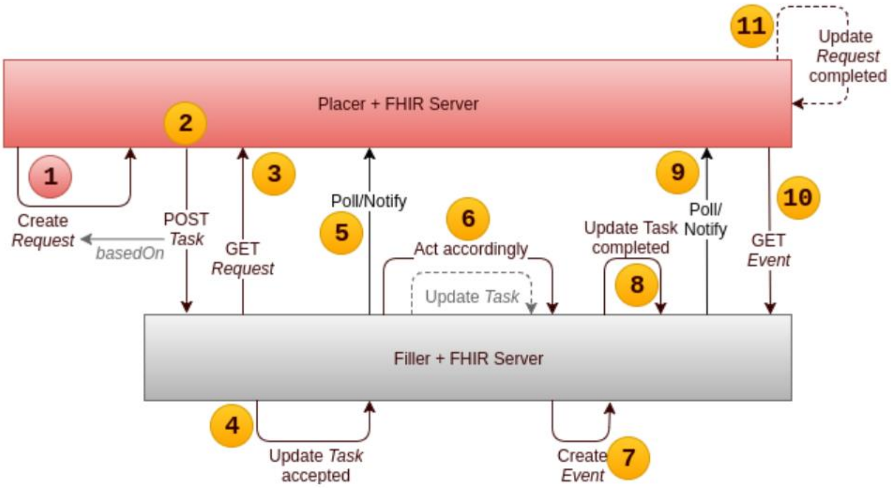


Figure 15 - FHIR worflow management pattern **(HL7 International, n.d.-aa)**

3 Implementations analysis

This chapter treats the current state of FHIR implementations. Some of the databases, libraries and frameworks that can be used to build FHIR systems will be reviewed. This paper does not claim that the tools presented are the only ones available to implement FHIR. The libraries, frameworks and databases presented, found as a result of the research conducted, have been analyzed in order to determine which library or framework use will best fit the creation of a FHIR-based application.

3.1 Databases

3 different approaches exist for FHIR object's storage (McKenzie, 2014):

- Storing the XML or JSON representation directly in a Not only SQL (NoSQL) Database
- Storing the XML or JSON as a Binary Large OBject (BLOB) and save the index data in a table
- Mapping the objects to a classic relational database (DB)

The NoSQL approach is interesting in the FHIR context since most of the storage consist of storing objects in JSON format, a semi-structured type of data. JSON storage is handled natively in most NoSQL databases and thus does not require a lot of transformation in order to be persisted, retrieved and used in the application. NoSQL enables high performances on great volumes of data. It is also often cheaper to use since a lot of the databases are open-source.

To store JSON or XML objects it is also possible to use a relational database management system (RDBMS) and store them as a BLOB. The BLOB has the ability of storing binary data and is therefore often used to store audio files, images or multimedia files. Moreover, using a BLOB avoids having to do character conversions to the database's text format and allows the storage of larger JSON documents (Oracle, n.d.).

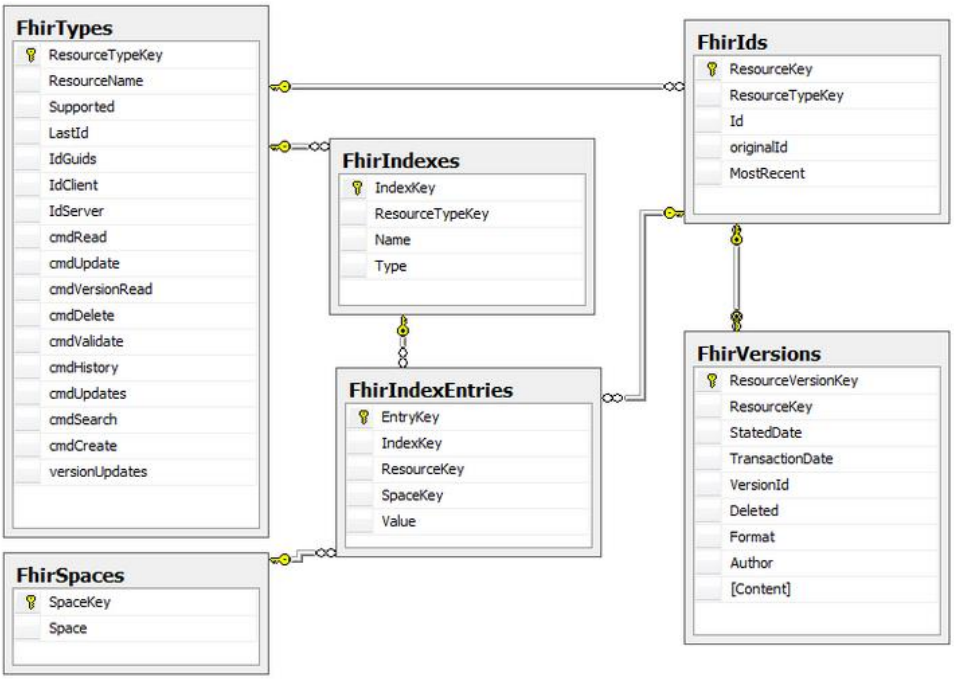


Figure 16 – FHIR server MySQL database schema (Grieve, 2012)

The figure above shows an example of a database schema that could be used to store FHIR resources and build a database supporting FHIR capabilities such as indexing, versioning and advanced searching. The actual content of the resource is in the table *FhirVersions* in the [Content] column.

Supporting FHIR in a complete manner, be fully conformant, requires a well thought database model and might require a lot of different stored procedures in furtherance of keeping conformance to the specification. Building such a database will request special effort in modeling and maintenance since FHIR is still in development and thusly subject to change.

The last option consists of storing the objects in a usual textual way in a relational database like MySQL, Oracle, Derby…, or in an object-oriented database such as PostgreSQL. Some new features of the RDBMS allow storing JSON in a native NoSQL-ish way enabling better search capabilities (Oracle, n.d.).

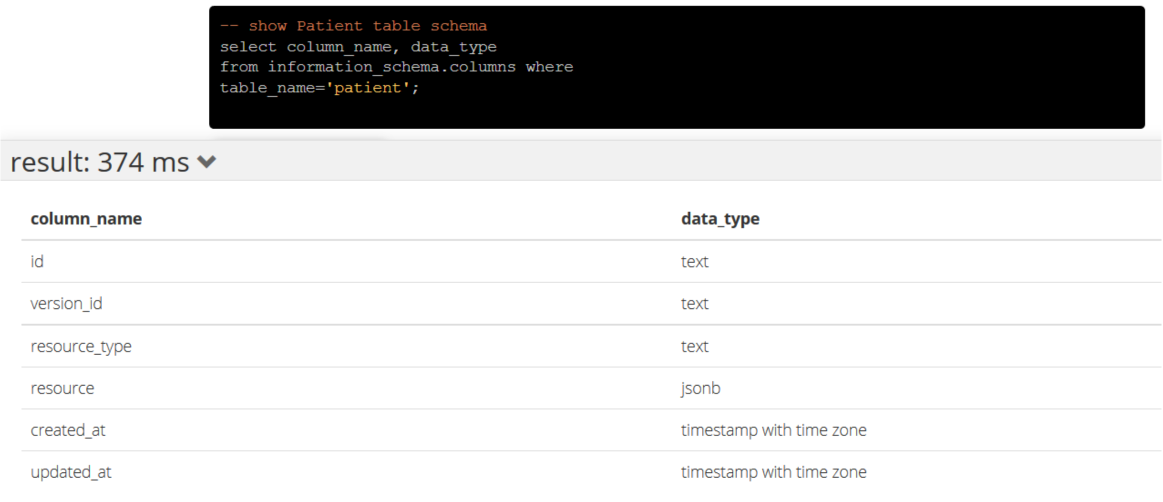
3.1.1 FHIRbase

FHIRbase

Git: https://github.com/fhirbase
License: MIT License
Type: open-source
Logo: http://fhirbase.github.io/

FHIRbase is an open-source database designed by Health Samurai Team to store and retrieve data in FHIR format and is now ready for production use. It is an extension of the well-known PostgreSQL database. The interaction with a FHIRbase DB is done using a PostgreSQL client with SQL syntax. To date, FHIRbase supports the DSTU-2 version of the FHIR specification.

Instead of directly inserting, updating and deleting data inside the FHIRbase, stored procedures are used so that all the different actions required on data changes can be performed. Since PostgreSQL has a native support for JSON format (jsonb type), JSON is the format in which FHIRbase chose to store the FHIR resources. If XML is preferred in an application, it is always possible to convert the JSON with some code in the application directly (Health Samurai Team, n.d.-d).



```
-- show Patient table schema
select column_name, data_type
from information_schema.columns where
table_name='patient';
```

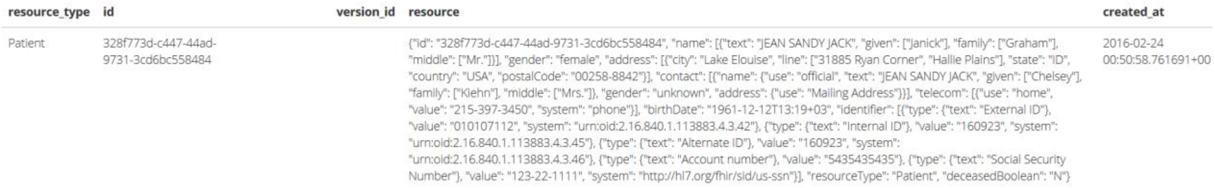
result: 374 ms

column_name	data_type
id	text
version_id	text
resource_type	text
resource	jsonb
created_at	timestamp with time zone
updated_at	timestamp with time zone

Figure 17 – Querying the patient table of FHIRbase (Health Samurai Team, n.d.)

Figure 17 shows the structure and the data types of the patient table as conceived by FHIRbase. The JSON representation of the resource is stored using the native jsonb type, as explained before. The other types are texts and timestamps.

With FHIRbase table structures, a patient resource is stored as following:



resource_type	id	version_id	resource	created_at
Patient	328f773d-c447-44ad-9731-3cd6bc558484		{"id": "328f773d-c447-44ad-9731-3cd6bc558484", "name": [{"text": "JEAN SANDY JACK", "given": ["Janick"], "family": ["Graham"], "middle": ["Mr."]}], "gender": "female", "address": [{"city": "Lake Elouise", "line": ["31885 Ryan Corner", "Hallie Plains"], "state": "ID", "country": "USA", "postalCode": "00258-8842"}], "contact": [{"name": {"use": "official", "text": "JEAN SANDY JACK", "given": ["Chelsey"], "family": ["Klehn"], "middle": ["Mrs."]}, "gender": "unknown", "address": {"use": "Mailing Address"}}], "telecom": [{"use": "home", "value": "215-397-3450", "system": "phone"}], "birthDate": "1961-12-12T13:19+03", "identifier": [{"type": {"text": "External ID"}, "value": "010107112", "system": "urn:oid:2.16.840.1.113883.4.3.42"}, {"type": {"text": "Internal ID"}, "value": "160923", "system": "urn:oid:2.16.840.1.113883.4.3.45"}, {"type": {"text": "Alternate ID"}, "value": "160923", "system": "urn:oid:2.16.840.1.113883.4.3.46"}, {"type": {"text": "Account number"}, "value": "5435435435"}, {"type": {"text": "Social Security Number"}, "value": "123-22-1111", "system": "http://hl7.org/fhir/sid/us-ssn"}], "resourceType": "Patient", "deceasedBoolean": "N"}	2016-02-24 00:50:58.761691+00

Figure 18 - sample patient stored (Health Samurai Team, n.d.)

3.1.1.1 Documentation and analysis

All the documentation on FHIRbase is available online[9]. They have set up a nice working environment with tutorials, technical documentation, a demo, an installation guide and code samples. All information needed can be found very easily which should make working with FHIRbase enjoyable.

The project can be downloaded from GitHub[10]. It obtained 72 stars and has one main contributor niquola who is backed up by three others less committed contributors. Niquola works also on the fhir.js library as well as on Aidbox, which will be introduced later in this work.

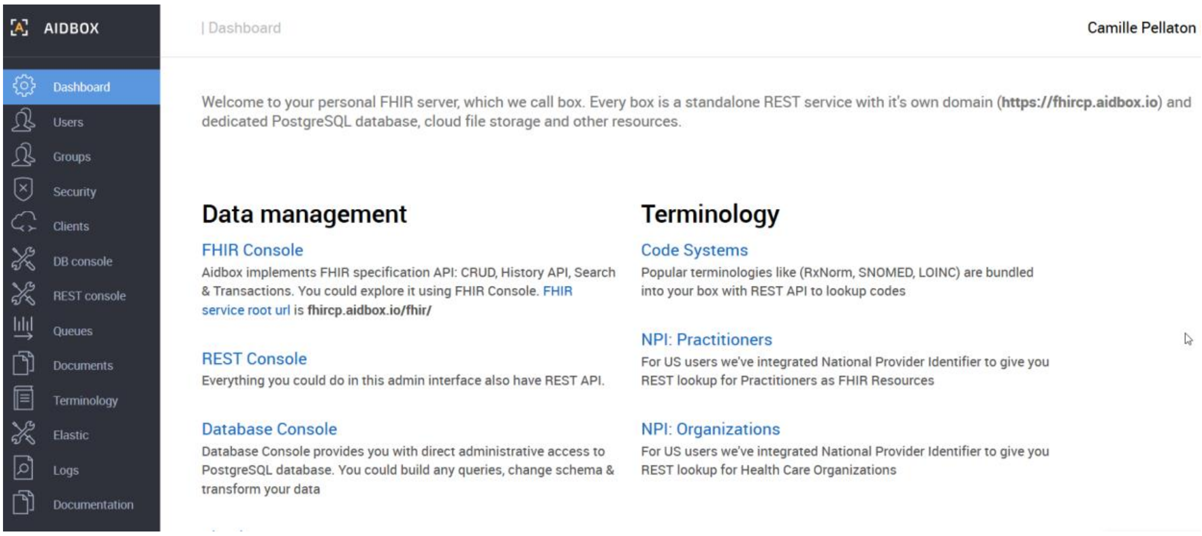
3.2 Framework: Aidbox

Aidbox

Website: https://aidbox.io/index.html
License: Entreprise License
Type: free
Logo: https://hello.aidbox.io/

As well as offering FHIRbase, Health Samurai Team offers free protected FHIR test servers and a cloud backend platform called Aidbox. Once an Aidbox account has been created, it is possible to create a "box" which corresponds roughly to a FHIR server instance. The Figure 19 shows the interface to manage a box with Aidbox.

[9] http://fhirbase.github.io/index.html
[10] https://github.com/fhirbase/fhirbase-plv8



Welcome to your personal FHIR server, which we call box. Every box is a standalone REST service with it's own domain (**https://fhircp.aidbox.io**) and dedicated PostgreSQL database, cloud file storage and other resources.

Data management

FHIR Console

Aidbox implements FHIR specification API: CRUD, History API, Search & Transactions. You could explore it using FHIR Console. FHIR service root url is **fhircp.aidbox.io/fhir/**

REST Console

Everything you could do in this admin interface also have REST API.

Database Console

Database Console provides you with direct administrative access to PostgreSQL database. You could build any queries, change schema & transform your data

Terminology

Code Systems

Popular terminologies like (RxNorm, SNOMED, LOINC) are bundled into your box with REST API to lookup codes

NPI: Practitioners

For US users we've integrated National Provider Identifier to give you REST lookup for Practitioners as FHIR Resources

NPI: Organizations

For US users we've integrated National Provider Identifier to give you REST lookup for Health Care Organizations

Figure 19 - An Aidbox box

Aidbox has several out-of-the-box tools such as users, groups, security and client management. It also includes a DB console to interact with the underlaying FHIRbase and a REST console to operate on the FHIR sever.

Aidbox allows to deploy single-page application (SPA) directly into a dedicated box. More complex applications work with aidbox as well, but require isolated web servers for their hosting.

Aidbox proposes several different usages, one can use it as (Health Samurai Team, n.d.-a):

- An API facade
- A backend
- A clinical data repository
- An integration bus

3.2.1.1 Features

Aidbox provides a database to store health data, a FHIR compliant REST API for data access and exploit. It is also OAuth secured and enables the creation of security policies to restrict access to data. It also includes an interface for the user management and a registration module (Health Samurai Team, n.d.-b).

3.2.1.2 Documentation and analysis

The documentation is well structured. At the same place[11], everything needed to get started with Aidbox and the implementation of an application using it can be found. If not, they have a reactive costumer service that will answer questions.

Technical documentation, Implementation guide, user guide, code samples and even a sample app are available freely on their website. Although the documentation is not yet finished, it is already valuable and usable.

3.3 Libraries

3.3.1 The HAPI-FHIR Java library

HAPI-FHIR

Git: https://github.com/jamesagnew/hapifhir/releases
License: Apache Software version 2.0
Type: open-source
Logo: http://hapifhir.io/

HAPI-FHIR is a java library that enables FHIR capability in an application. It is an open-source implementation of the FHIR specification. It has primarily been developed by the University Health network to build a FHIR RESTful server communicating with various data storages and repositories. The aim of this library is to allow fast, flexible and conformant implementations of FHIR. It provides not only a RESTful FHIR client but the API allows to create a RESTful FHIR server.

[11] https://aidbox.io/docs/Features.html

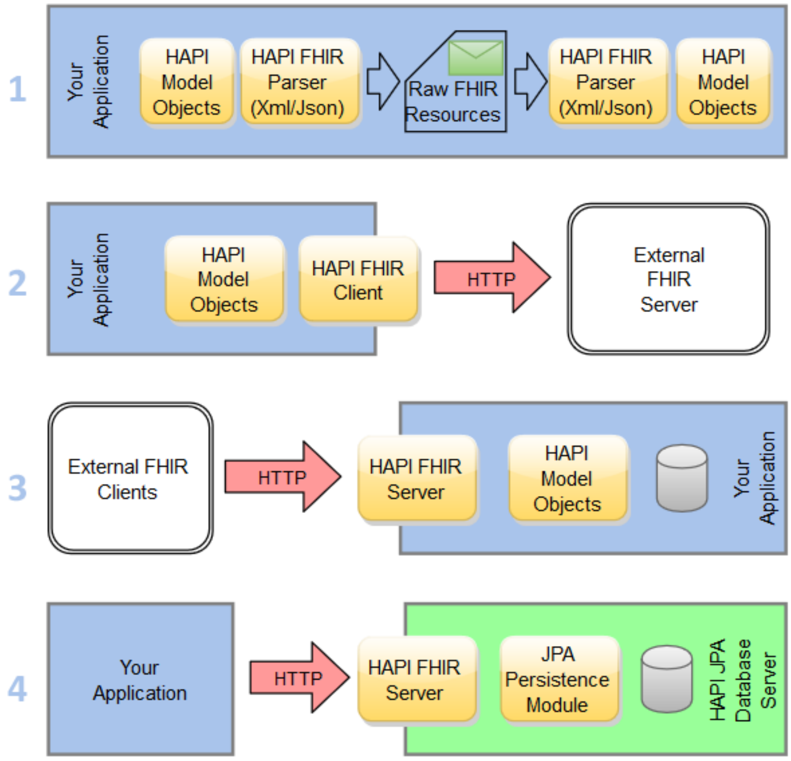


Figure 20 - HAPI-FHIR usage patterns **(University Health Network, n.d.-b)**

HAPI can be utilized mainly in four different manners that are illustrated by the image above. Further explanations on those four manners (University Health Network, n.d.-b):

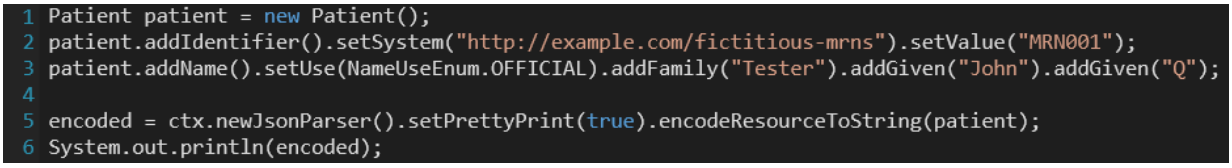
1. Usage of a parser and an encoder to enable a conversion between the application and FHIR's data model.
2. Use the HAPI FHIR client in an application to retrieve or save resources onto an external server.
3. External FHIR client can interact with the application by accessing the HAPI FHIR server.
4. Another way of implementing with HAPI FHIR is to deploy a HAPI JPA/Database server and develop applications to work with it

HAPI API supports both XML and JSON encoding. For a better testability, a public open-source test server built with HAPI-FHIR is available on the internet[12] (University Health Network, n.d.-b).

[12] At http://fhirtest.uhn.ca

3.3.1.1 Data model

The interaction with FHIR model objects is done through a developer friendly Fluent Interface. Every FHIR resource type is defined by HAPI model classes. Since HAPI uses a fluent style, function calls can be done in a very practical way. *Figure 21* illustrates a Patient object's creation, the assignments of its attributes and how it is parsed to be displayed (University Health Network, n.d.-c).



```
Patient patient = new Patient();
patient.addIdentifier().setSystem("http://example.com/fictitious-mrns").setValue("MRN001");
patient.addName().setUse(NameUseEnum.OFFICIAL).addFamily("Tester").addGiven("John").addGiven("Q");

encoded = ctx.newJsonParser().setPrettyPrint(true).encodeResourceToString(patient);
System.out.println(encoded);
```

Figure 21 - Fluent style method calls **(University Health Network, n.d.-c)**

3.3.1.2 RESTful Client

The aim of this HAPI client is to ease the implementation, thus a built-in mechanism allows to connect to FHIR RESTful servers. Two types of clients are available with HAPI: the fluent/generic client and the annotation client. The first one is a simpler version that can be set up fast. The second one requires more effort to be implemented but allows a better compile-time checking against the servers (University Health Network, n.d.-f).

The java client library is well documented and thus seems very practical to use. The Javadoc for FHIR STU 3 is freely available online[13]. All the classes representing fhir resources are listed, their methods and attributes are presented in a useful manner: often with descriptions, with datatypes, parameters and links to other embedded classes.

3.3.1.3 RESTful Server

It is very easy to add server capability to the application with the built-in HAPI Server mechanism. This HAPI RESTful server is based on Servlet and can be deployed in many existing containers. The implementation is almost entirely done using annotations and requires very little time (University Health Network, n.d.-g).

As seen previously, it is also possible to use the HAPI FHIR RESTful server module to create an endpoint to exploit a chosen data source. The data source could be for example a database,

[13] http://hapifhir.io/apidocs-dstu3/index.html

a clinical system or some files. The persistence module that HAPI provides allows to create a full server implementation backed up by a chosen database. (University Health Network, n.d.-d).

3.3.1.4 Documentation and analysis

The HAPI API is well documented. For most features, Client or Server, some examples of code, explanations and implementation advices are available on the HAPI FHIR website[14].

The test server is a nice-to-have tool, it has a friendly user interface that most other test servers lack. Open source implementation examples are easily found, they often reveal themselves to be of great interest. As of now, HAPI's test server is probably the most complete implementation of a fhir server available since it allows complex searches and implements a reference mechanism allowing, for instance, a patient resource to store a reference to its practitioner.

The git project has one main and frequent contributor and three relatively frequent contributors. The main contributor is part of the University Health Network. The project has a 372 stars rank.

3.3.2 FHIR .NET API

FHIR .NET API

Git: https://github.com/ewoutkramer/fhir-net-api
License: BSD 3-clause Revised License
Type: open-source
Logo: https://fhir.furore.com/fhir-api/

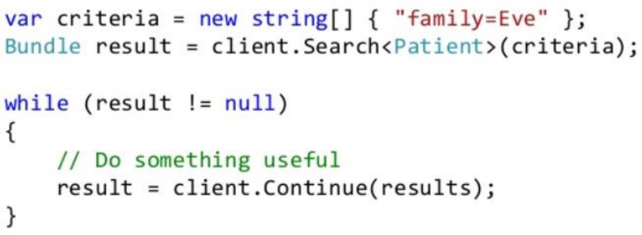
Created by Furore Health Informatics, the .NET API is the reference implementation of the FHIR standard for the Microsoft .NET platform. Along with the library, Furore also introduced spark, an open-source FHIR server and Forge, a FHIR profiling tool.

The library allows to rapidly create a RESTful FHIR client, offers classes for serialization and deserialization into XML and JSON formats as well as parsers, provides the possibility to define server calls with the search commands and class models to work with the data model of the

[14] http://hapifhir.io/

FHIR specification. The API enables to work with extensions and a process of validation of data content, but those features are more complex to set up (Furore, 2017).

The library allows to work efficiently with FHIR resources by defining a class for each resource. The REST operations are addressed with a very practical FHIR client as shown on the Figure below for the search interaction.



```
var criteria = new string[] { "family=Eve" };
Bundle result = client.Search<Patient>(criteria);

while (result != null)
{
    // Do something useful
    result = client.Continue(results);
}
```

Figure 22 - REST Search operation (FHIR Developer Days, 2014)

3.3.2.1 Spark

Spark is the open-source FHIR server implementation of Furore. It has been downloaded and used all around the world, mainly for testing purposes but in production environment as well. Spark offers a wide variety of features including a complete search implementation, Batch, History, XML and JSON support and Validation. Spark works with the OAuth and OpenID authentication protocols.

A FHIR STU3 compliant Spark commercial version is being built to adapt to the production use (Furore, n.d.).

Spark's implementation can be downloaded from GitHub[15].

3.3.2.2 Documentation and analysis

Even though the API has a sufficient amount of documentation, it is a bit short of explanations. The whole documentation can be found online[16].

An element to emphasize is the lack of server capability of the library to build FHIR servers. The Java HAPI API is more complete in that sense. Nevertheless, a proper server

[15] at https://github.com/furore-fhir/spark

[16] http://ewoutkramer.github.io/fhir-net-api/docu-index.html

implementation is available publicly and freely which allows to build complete FHIR solutions using .NET.

3.4 Platform: SMART

SMART on FHIR

Git: https://github.com/smart-on-fhir
License: Licensed under the Apache License, Version 2.0
Type: open-source
Logo: https://smarthealthit.org/

Substitutable Medical Applications, Reusable Technologies (SMART) Health IT is an open and standards based application platform that helps developers create healthcare integrated apps.

The SMART project was launched and is carried out by the Boston Children's Hospital Computational Health Informatics Program and the Harvard Medical School Department for Biomedical Informatics. The idea is to enable and improve exchange around technology in healthcare and ease the development of applications by offering a platform, some libraries, open-source applications and a sandbox for testing. If supporting the SMART standard in an EHR or in a data warehouse, it is then possible to easily integrate existing apps of the platform into the said HIS (SMART, n.d.).

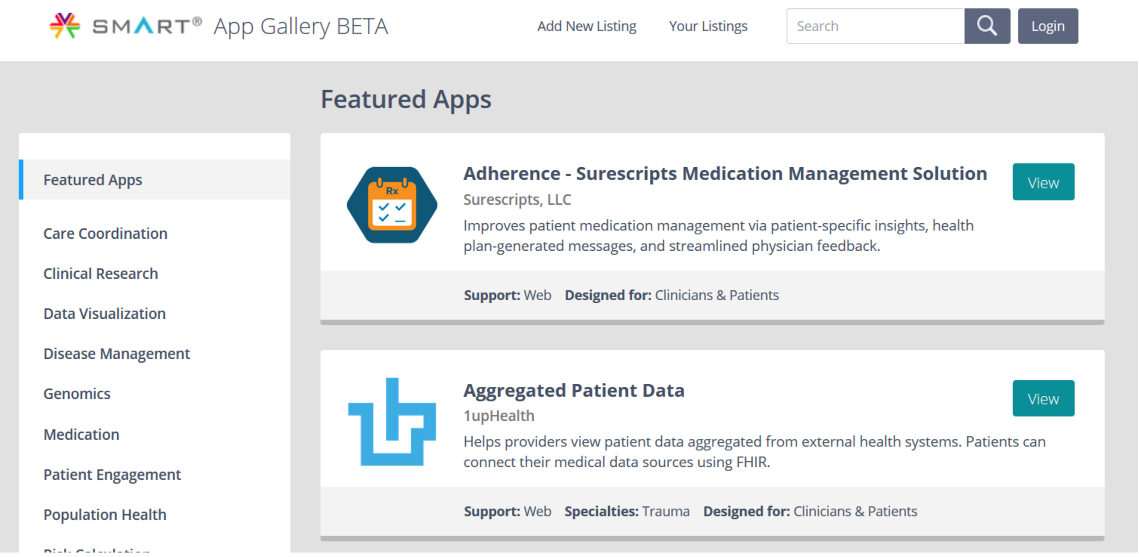
To create SMART client apps whether native, mobile or web apps that can communicate with SMART supporting systems, SMART introduced SMART on FHIR. SMART on FHIR is a collection of open specifications and provides several FHIR client libraries and FHIR testing tools.

To date libraries for three different programming environments can be downloaded: Javascript (SMART JS), Python and IOS (Swift-SMART). The created web apps can be integrated directly in the EHRs' user interfaces. This allows the user to have every functionality needed at the same place (SMART Health IT, 2017a).

In an article of the Journal of the American Medical Informatics Association, they used an analogy that explains clearly what the SMART platform is:

> *We drew an analogy with mobile phone platforms such as iPhone and Android, which lower the barrier to app development by providing a software platform with a published interface to a set of core services such as camera, address book, geo-location, and cell and wireless networks. The platform functionally separates the core system from the apps, and the apps are substitutable.* (Mandl, et al., 2012)

By enabling the creation of substitutable apps in medical systems, SMART enables to change quickly from an unsatisfying app to a better one with the lowest costs possible. This environment also empowers a more competitive market type and once again helps reducing HIS development costs (Mandl, et al., 2012). Those substitutable apps can be found on the platform's App Gallery as illustrated here below.



Adherence - Surescripts Medication Management Solution
Surescripts, LLC
Improves patient medication management via patient-specific insights, health plan-generated messages, and streamlined physician feedback.
Support: Web **Designed for:** Clinicians & Patients

Aggregated Patient Data
1upHealth
Helps providers view patient data aggregated from external health systems. Patients can connect their medical data sources using FHIR.
Support: Web **Specialties:** Trauma **Designed for:** Clinicians & Patients

Figure 23 – The SMART App Gallery ('SMART App Gallery', n.d.)

3.4.1 Data model

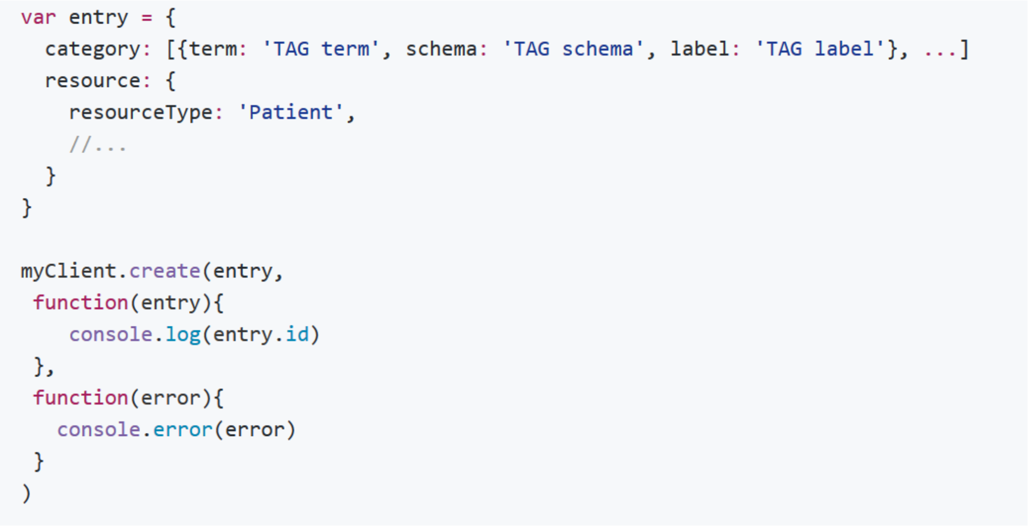
The data layer of the SMART project is built on the FHIR REST API and the FHIR resources definition. In extension to the core capabilities of FHIR, SMART defines a bunch of profiles to better detail and adapt the specification to real world usage ('What Is SMART?', 2012). SMART uses the profiles introduced by The Argonaut Implementation Guide to enlarge FHIR resources capabilities (Argonaut Project, 2017).

3.4.2 SMART JS client

This javascript client library can be downloaded on GitHub[17]. It uses an embedded fhir.js open-source library to interact with FHIR REST API servers. SMART JS adds the ability to work in the context of an EHRs.

Thus, it handles the same API operations as the fhir.js library, the most important of those being:

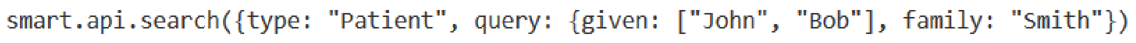
- FHIR CRUD (Create, Read, Update, Delete)
- vread (reading a specific version of a resource)
- search
- fetchAll



```
var entry = {
  category: [{term: 'TAG term', schema: 'TAG schema', label: 'TAG label'}, ...]
  resource: {
    resourceType: 'Patient',
    //...
  }
}

myClient.create(entry,
 function(entry){
    console.log(entry.id)
 },
 function(error){
   console.error(error)
 }
)
```

Figure 24 – FHIR JS resources' creation (niquola, 2014)

The figure above, shows how to create resources with the fhir.js library. It is easy and fast to implement and conforms to the FHIR specification. The other FHIR operations are as easy as the create function to set up. For example, the search operation is coded as simply as the following:



```
smart.api.search({type: "Patient", query: {given: ["John", "Bob"], family: "Smith"}})
```

Figure 25 – SMART js search (SMART Health IT, 2017b)

[17] https://github.com/smart-on-fhir/client-js

To have an idea of what can be built with SMART js, a live demo of a SMART sample client is available online[18].

3.4.2.1 Documentation and analysis

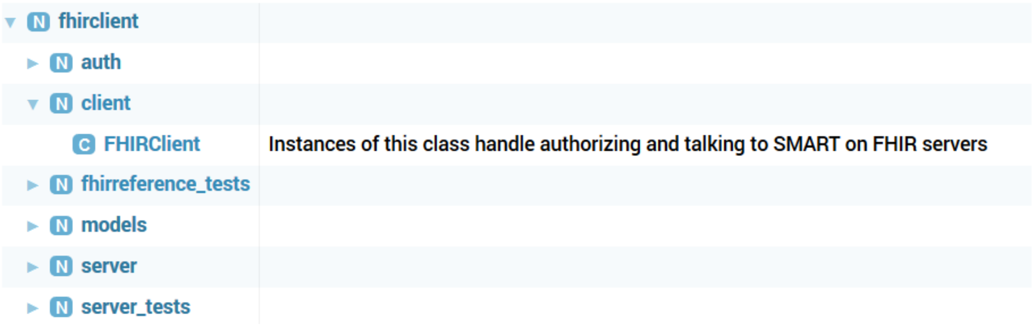
The library is well documented on the SMART on FHIR website[19], they have explanations and tutorials to get implementers started, but also on GitHub since they have extra information and documentation for the basis of the API. The only lack of documentation would be a presentation of the library's structure and a technical guide. It might be difficult to understand how to use all the available classes.

Currently the project has only 47 stars and two consistent contributors on GitHub, both part of the SMART Health IT.

3.4.3 SMART on FHIR Python client

The SMART on FHIR Python client allows setting up easily an interaction to a FHIR server or a SMART on FHIR server. It can be freely downloaded from GitHub[20]. For that mean, an out-of-the-box FHIRClient class is available, it enables the interaction with a FHIRServer class. The client can be used with Python 2.7.10 and 3 (SMART Health IT, 2017f).

The Figure below shows the classes' structure of the Python client. The following elements are listed: the *FHIRClient* class responsible for the server interaction, the authentication class is for the OAuth 2.0 protocol that this client accepts and the *models* folder containing all the classes representing the FHIR resources.



```
▼ N fhirclient
    ► N auth
    ▼ N client
          C FHIRClient           Instances of this class handle authorizing and talking to SMART on FHIR servers
    ► N fhirreference_tests
    ► N models
    ► N server
    ► N server_tests
```

Figure 26 – Python client class structure (SMART Health IT, 2017)

[18] https://plnkr.co/edit/DapXQaTONYSRSyylHdpu?p=preview
[19] http://docs.smarthealthit.org
[20] https://github.com/smart-on-fhir/client-py

The client handles FHIR data within its classes using a parser that manages the serialization and deserialization (SMART Health IT, 2017f). To illustrate this, below, the patient resource is presented from its definition in the FHIR specification to its final class form in the python application.

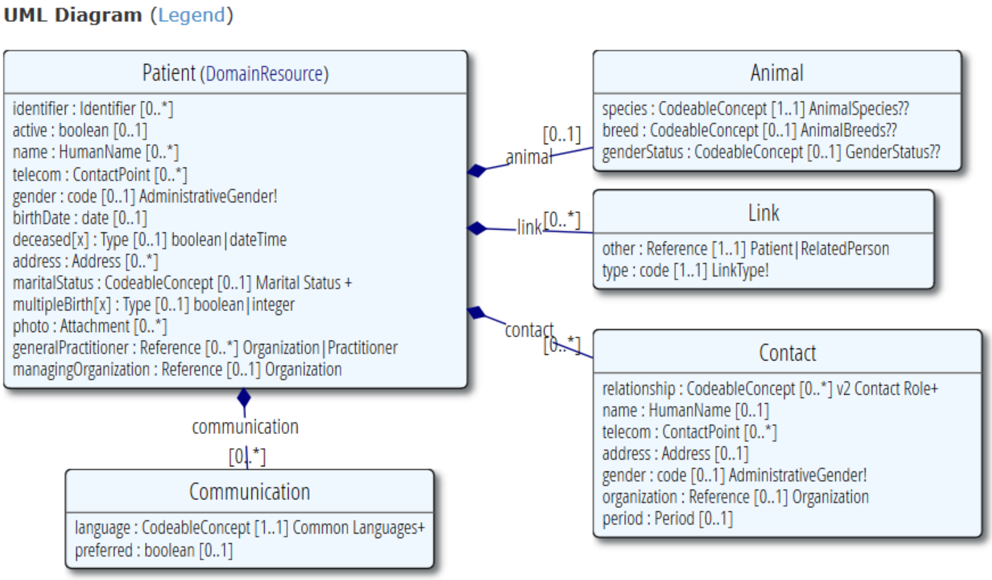
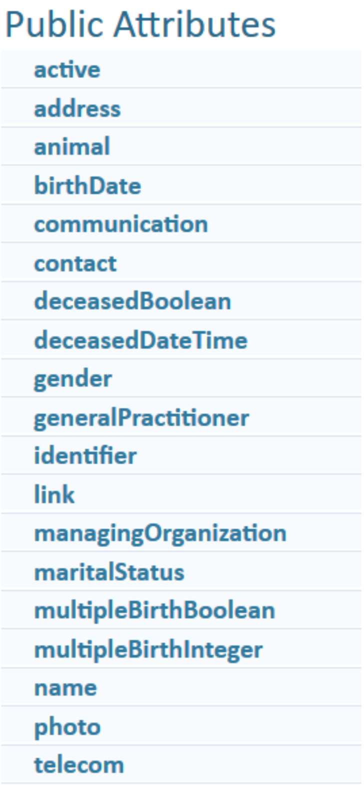

UML Diagram (Legend)
Patient (DomainResource)
identifier : Identifier [0..*]
active : boolean [0..1]
name : HumanName [0..*]
telecom : ContactPoint [0..*]
gender : code [0..1] AdministrativeGender!
birthDate : date [0..1]
deceased[x] : Type [0..1] boolean|dateTime
address : Address [0..*]
maritalStatus : CodeableConcept [0..1] Marital Status +
multipleBirth[x] : Type [0..1] boolean|integer
photo : Attachment [0..*]
generalPractitioner : Reference [0..*] Organization|Practitioner
managingOrganization : Reference [0..1] Organization
[0..1]
animal
Animal
species : CodeableConcept [1..1] AnimalSpecies??
breed : CodeableConcept [0..1] AnimalBreeds??
genderStatus : CodeableConcept [0..1] GenderStatus??
link
[0..*]
Link
other : Reference [1..1] Patient|RelatedPerson
type : code [1..1] LinkType!
contact
[0..*]
Contact
relationship : CodeableConcept [0..*] v2 Contact Role+
name : HumanName [0..1]
telecom : ContactPoint [0..*]
address : Address [0..1]
gender : code [0..1] AdministrativeGender!
organization : Reference [0..1] Organization
period : Period [0..1]
communication
[0..*]
Communication
language : CodeableConcept [1..1] Common Languages+
preferred : boolean [0..1]

Figure 27 – **Patient resource UML** (HL7 International, n.d.-n)

On the figure above the Unified Modeling Language (UML) diagram of the FHIR patient resource is presented. The figure 28 shows the different attributes of the python client's patient class. It can be seen that the patient class follows and conforms to the FHIR patient resource definition. The parser will be used to convert from the FHIR resource received from the server (often in JSON format) to the application resource's object in order to have a functional business object.

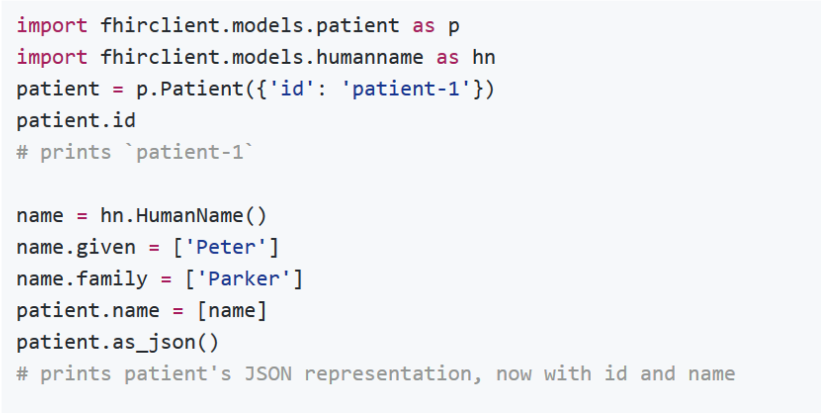


Public Attributes

- active
- address
- animal
- birthDate
- communication
- contact
- deceasedBoolean
- deceasedDateTime
- gender
- generalPractitioner
- identifier
- link
- managingOrganization
- maritalStatus
- multipleBirthBoolean
- multipleBirthInteger
- name
- photo
- telecom

Figure 28 - patient class of the python client (SMART Health IT, 2017e)

Here below an example of how the library works with FHIR resources in its classes. As it can be seen, the library is well-structured, and thus allows fast familiarization and implementation.



```
import fhirclient.models.patient as p
import fhirclient.models.humanname as hn
patient = p.Patient({'id': 'patient-1'})
patient.id
# prints `patient-1`

name = hn.HumanName()
name.given = ['Peter']
name.family = ['Parker']
patient.name = [name]
patient.as_json()
# prints patient's JSON representation, now with id and name
```

Figure 29 – working with FHIR data Python (*client-py*, 27 August 2014/2017)

3.4.3.1 Documentation and analysis

The technical documentation[21] is well organised and thus makes the search for information very pleasant. All the classes and structures are listed and have documentation in attachment. The lack here would be a programming guide containing some tutorials and sample codes to get started faster.

As of end of May 2017, it has 43 stars on GitHub and one main contributor that works for the SMART organization.

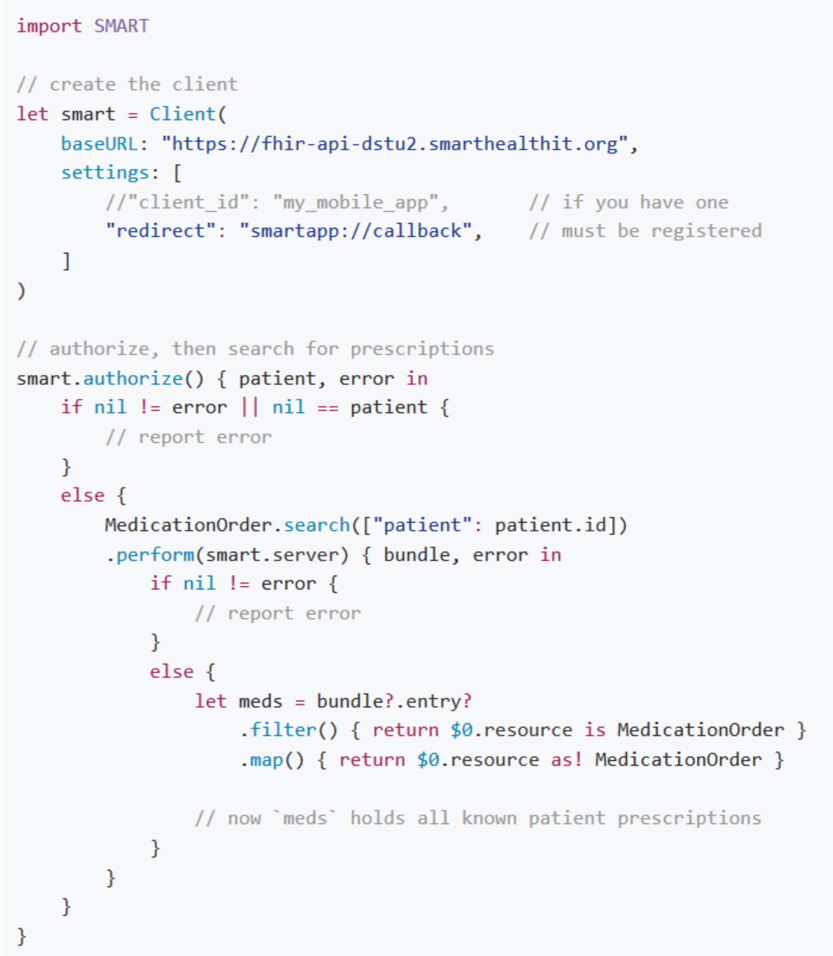
3.4.4 SWIFT-SMART IOS client

This library is a complete FHIR implementation of a client for IOS 8, OS X 10.10 and higher versions. It can be entirely downloaded from GitHub[22].

Working in the same way as the above libraries, SWIFT-SMART provides a FHIR client to set up the interaction with a FHIR server. The instantiation of the client is fast, just like understanding how to work with it. In SWIFT's wiki, a code sample shows that:

[21] http://docs.smarthealthit.org/client-py/

[22] https://github.com/smart-on-fhir/Swift-SMART.

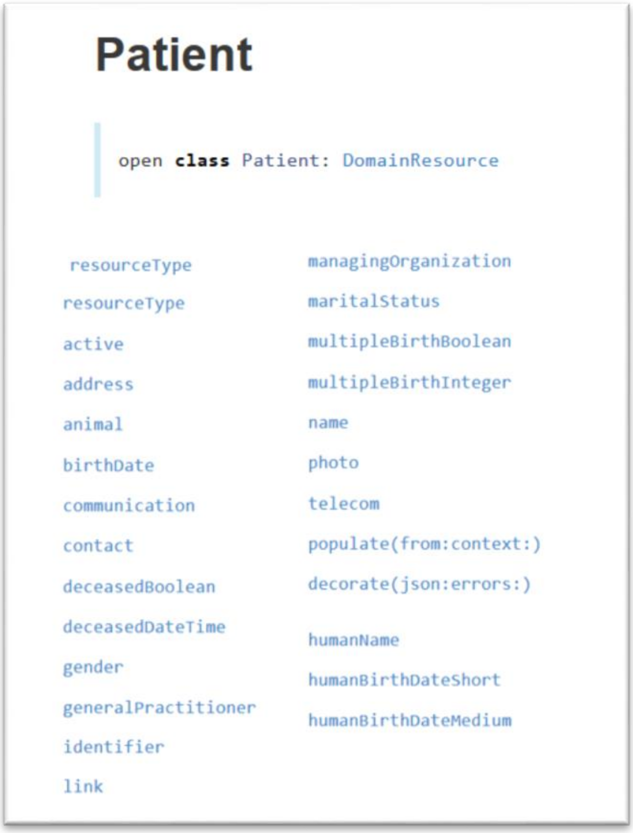


```
import SMART

// create the client
let smart = Client(
    baseURL: "https://fhir-api-dstu2.smarthealthit.org",
    settings: [
        //"client_id": "my_mobile_app",       // if you have one
        "redirect": "smartapp://callback",    // must be registered
    ]
)

// authorize, then search for prescriptions
smart.authorize() { patient, error in
    if nil != error || nil == patient {
        // report error
    }
    else {
        MedicationOrder.search(["patient": patient.id])
        .perform(smart.server) { bundle, error in
            if nil != error {
                // report error
            }
            else {
                let meds = bundle?.entry?
                    .filter() { return $0.resource is MedicationOrder }
                    .map() { return $0.resource as! MedicationOrder }

                // now `meds` holds all known patient prescriptions
            }
        }
    }
}
```

Figure 30 – SWITF client sample code (SMART Health IT, 26 June 2014/2016)

The code of the figure above includes the Client initialization, lets the user log in to select a patient to fetch a patient's *MedicationOrder* resource. The library is robust and practical to use.

The model classes are complete regarding their conformance to the FHIR resource definitions. Once again, taking the same example used for the python library, if the patient resource definition is compared to the patient class provided by the SWIFT library, all necessary attributes are present:



Patient

```
open class Patient: DomainResource
```

```
resourceType
resourceType
active
address
animal
birthDate
communication
contact
deceasedBoolean
deceasedDateTime
gender
generalPractitioner
identifier
link
managingOrganization
maritalStatus
multipleBirthBoolean
multipleBirthInteger
name
photo
telecom
populate(from:context:)
decorate(json:errors:)
humanName
humanBirthDateShort
humanBirthDateMedium
```

Figure 31 – Patient resource's attributes SWIFT (SMART Health IT, 2017c)

3.4.4.1 Documentation and analysis

Of all the different libraries review in this work, SWIFT is the best documented. A programming guide, a full documentation and a sample application are available[23] and make the implementers' jobs much easier.

The project can be download from GitHub[24]. It currently has 45 stars and only one contributor working for the SMART Health IT organization.

3.4.5 Comparative table

To make the best decision regarding which library to use to implement a FHIR client to interact with a FHIR server, several criteria have to be considered. To help comparing between the libraries, a comparative table regrouping the different criteria selected has been realized and is presented below.

[23] At http://docs.smarthealthit.org/Swift-SMART/

[24] https://github.com/smart-on-fhir/Swift-SMART

Each criterion is perused below with a description and explanation for the way it is rated. The appreciation of some criteria uses a rating system on six points. As it is the common grading system in Switzerland, it is more understandable to grade in this fashion. The best grade is six, a sufficient work gets a four and, below that, are the insufficient grades.

- **Simplicity**: how simple it is to use the library. The more the library has been designed for practical use, the more point it will obtain.
- **Documentation**: is how the library is documented. A complete documentation includes a Programming guide, a Technical documentation, explanations and if possible a forum or chat. A complete documentation will receive the grade of six.
- **Exchange supported**: shows the interoperability paradigm provided by the library. As seen previously FHIR thought of four different exchange: REST, messages, documents and services.
- **Server-side API:** the library gets a point if it has an API for FHIR server creation (binary notation).
- **Authentication**: The method supported for authentication, OAuth2 being the most important in the medical environment.
- **Data model:** how the library handles the FHIR resources.
- **FHIR version**: the latest FHIR version supported by the library. As of now, the latest release is STU3.
- **Sample applications/code:** an appreciation on six points of the amount of code and application samples that can be found to help implementing a FHIR application using the library. To get a four the library's documentation must give just enough samples to make the implementation feasible for a junior developer. The appreciation considers tutorials, samples of code and applications found on the internet, weather provided by the library's or not.
- **REST operations:** The FHIR REST API defines a set of base operations to be implemented. Some of the libraries have nicely implemented the operations and even extended their usage with for example some advanced search operation. Comparing the libraries, the one with the most operations will get the best grade and the others will be graded accordingly.

Criteria / Client library	HAPI FHIR	FHIR .NET API	SMART ON FHIR JS	SMART ON FHIR Python	SWIFT-SMART IOS
Simplicity (6pts)	5,5	5	5	4,5	5
Documentation (6pts)	5	4	4,5	5,5	6
Exchange supported	REST: XML & JSON, MESSAGES	REST: XML & JSON	REST: JSON	REST: JSON	REST: JSON
Server-side API (1/0)	1	0	0	0	0
Authentication	HTTP basic / OAuth2	Not found	HTTP basic / OAuth2 / Cookies	HTTP basic / OAuth2	HTTP basic / OAuth2
Data model	Classes for every resource	Classes for every resource	Resources in JSON	Classes for every resource	Classes for every resource
FHIR version	STU3	STU3	Not found	STU3	STU3
Sample applications / code (6pts)	5	3	5,5	4,5	4,5
Rest Operations (6pts)	5,5	4	5	4	6
TOTAL	22	16	20	18,5	21,5

Table 2 - Comparision table, author's data

3.4.6 Discussion

The analysis of the libraries evidenced that they are all well suited for the construction of a FHIR-based application. Documentation is provided for each one of them, although the amount of documentation varies significantly depending on the library. Libraries that lack documentation require a lot of time and effort to use, therefore it was a very important criterion in the context of this work.

HAPI java comes in first place, closely followed by SWIFT. Then come the Javascript, the python and the .NET libraries. The research team collaborating on this project had no particular interest in a IOS client application but a strong preference for the application to be developed in Java or Javascript. Both these languages could easily meet the requirements of this work, nevertheless the java application obtained a better score for all the comparative criteria except the amount of code and application samples. On top of that, a FHIR server implementation is provided by HAPI and is fully compatible with the FHIR client of the library. Java also happens to be the main language taught at the HES-SO Valais-Wallis and thus, is a language I am acquainted with. The different arguments advanced led to the decision of selecting HAPI java library to facilitate the development of a web application based on the FHIR specification.

4 Prototype

The aim of the application prototype that has been developed in the context of this work is to test and evaluate the implementation of a web application based on the FHIR standard using an existing client library. Furthermore, the objective is to demonstrate weather FHIR can really live up to its name by being fast to implement and weather it can become a significant element of the health information systems.

The FHIR standard being under development, test servers available could not guarantee the functioning of the application in the future. In addition, test servers are not reliable since it is not possible to know when the servers are going to be unavailable due to maintenance. Consequently, the hosting of the FHIR server, using a full RESTful FHIR server implementation provided by HAPI-FHIR for the STU 3, is on a private server. The prototype interacts with this private RESTful FHIR test server.

As explained in the previous chapter, to facilitate the implementation, the HAPI-FHIR java library has been selected for this project. The HAPI library provides a RESTful java client that has been designed to ease and accelerate the development of FHIR REST applications.

The source code of this project can be found on the compact disk provided in attachment as well as on GitHub[25].

4.1 Context

This paper was realized within the Scrum framework. The realization of the work has been broken down into six sprints. The development of the application started at the third sprint and was finished during the fifth sprint.

Each sprint had a fixed value of two weeks. At the end of a sprint, a review meeting was scheduled with the professor to share the progress made in the development, to validate the user stories done during the sprint and to obtain feedbacks.

The Scrum documents are provided on the compact disk in attachment.

[25] https://github.com/camouie/fhirtb

4.2 Use-cases

The application supports three different roles with distinct authorizations:

- Administrator
- Doctor
- Patient

The demonstration application serves the following use cases:

The administrator is responsible for the accounts creation for doctors and patients so that they can log into the app. He has access to all the resources on the FHIR server. He can also modify resources and delete them.

The doctor has only access to his own patients' data that he can display and edit. He can also display the height, the weight and the heart rate of each patient. He can see a graph of the evolution of the weight of each patient to analyze his patients' weights tendencies.

The patient has only access to his own medical information. He can display and edit it. Like the doctor, he can display the graph of the evolution of his weight. The patient connects punctually to the application to enter data about his health: his height, his weight and his heart rate. This application is then made available to doctors with the usage of FHIR.

4.3 Disclaimer

The aim of this work not being on the actual construction of a Java web application, the following chapters presenting the client application will focus on the FHIR implementation in the JSF application and will not treat the JSF part of said application.

The codes snippets that are provided in this paper can be applied to a native java program, they do not require the set-up of a web application. The next chapters require the reader to be acquainted with Java programming language or, at a minimum, object-oriented programming languages in order to have a good understanding of the presented information.

4.4 Project Architecture

The overall architecture of the project is illustrated in the figure below. The test application uses HAPI model objects to work with FHIR resources in the java code. The HAPI FHIR client then interacts with the FHIR server by sending and requesting resources via HTTP. The project also contains a MySQL database to store users' information.

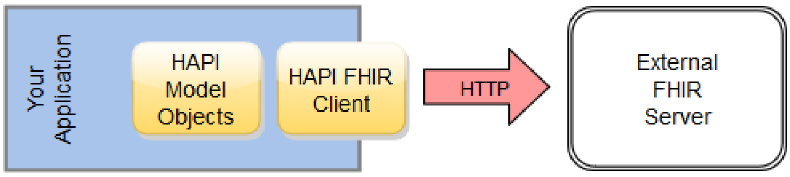
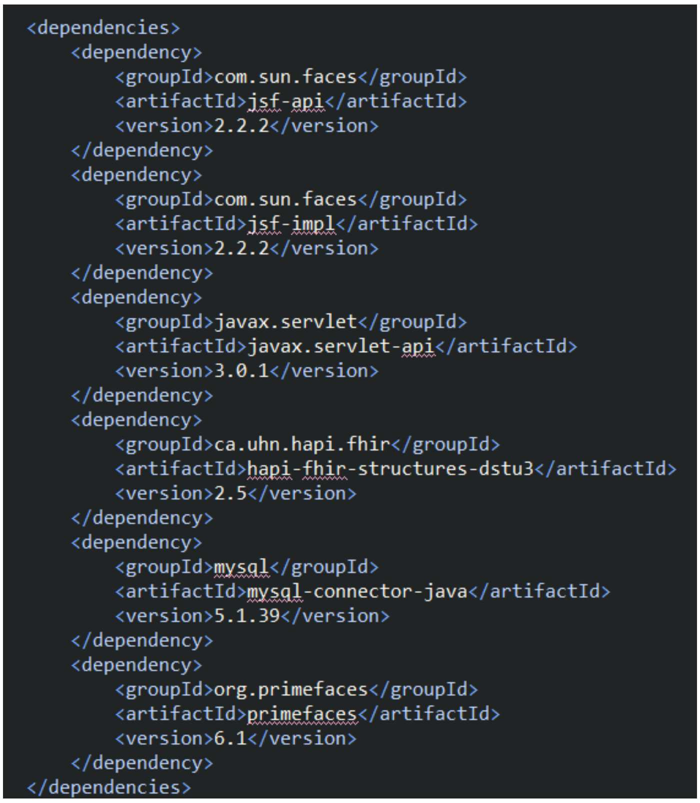


Figure 32 - Architecture of FHIR project (University Health Network, n.d.-b)

4.4.1 Overview of the client application

The client application uses the HAPI-FHIR RESTful client library. HAPI provides two different types of clients. The Generic and the Annotation clients. This application implements the Generic client which is easier and faster to set up. The Annotation client is more powerful when it comes to compile-time checking against servers. It is mainly used when creating an interface for developers to interact with servers having specific operations. For the purpose of this work and the time allotted, the generic client was selected and proved itself more than sufficient (University Health Network, n.d.-f).

The application is built with the latest JavaServer Faces framework (JSF) 2.2. This API as well as the others used in the project are included by using Apache Maven 3.5.



```xml
<dependencies>
    <dependency>
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-api</artifactId>
        <version>2.2.2</version>
    </dependency>
    <dependency>
        <groupId>com.sun.faces</groupId>
        <artifactId>jsf-impl</artifactId>
        <version>2.2.2</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.0.1</version>
    </dependency>
    <dependency>
        <groupId>ca.uhn.hapi.fhir</groupId>
        <artifactId>hapi-fhir-structures-dstu3</artifactId>
        <version>2.5</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.39</version>
    </dependency>
    <dependency>
        <groupId>org.primefaces</groupId>
        <artifactId>primefaces</artifactId>
        <version>6.1</version>
    </dependency>
</dependencies>
```

Figure 33 - pom.xml dependencies, source: author's data

The pom.xml abstract above presents the different libraries used in the application. The first three dependencies are libraries required to develop a JSF application. The fourth one is the HAPI-FHIR library allowing the use of all objects and models defined by HAPI for the STU 3., the latest release of the FHIR specification. Then the MySQL API to work with a MySQL database is included. The last library used is Primefaces. This JSF user interface (UI) framework allows to easily create various component using third party libraries such as jQuery. The dynamic graph created in the project is rendered using Primefaces. Eclipse neon was used as development environment to for this application.

No authentication protocol has been used in the application, nevertheless a login function has been implemented to ensure that only authenticated users have access to the application. In addition, an authorization mechanism is implemented to limit doctors and patients' rights to access resources.

4.4.2 FHIR Test server

As mentioned previously, a private FHIR server has been created for the purpose of this work. This FHIR server is running and is configured to work with an internal Apache Derby database. The fully contained server has been deployed to an Apache Tomcat server version 8.0.45 and is functioning well[26].

At least a dozen of public FHIR test servers are available online and are very practical to rapidly build an application to test it against. However, most of them do not implement a reference mechanism or do not allow external referencing. This was very inconvenient for the application prototype since Patient resources include a Practitioner reference. Before setting up a private server, public test servers have been used to test the application. The two servers mainly used are the one that HAPI provides[27] and the .NET FHIR server implementation called Spark[28].

Some differences appeared between these two FHIR implementations. Firstly, Spark allows some references but not others, for instance an Observation resource can contain a reference to its subject but Patients cannot hold references to their Practitioners, whereas HAPI has a more complete reference mechanism allowing, in this example, both references. Secondly, Spark does not suffer from the same latencies as the HAPI server. Important delays may occur when uploading a resource, the HTTP POST is executed and the client must wait few minutes for the resource to be available on the server for further queries. This delay happens mainly for resources referencing other resources. Lastly, the HAPI server catches up faster with the FHIR specification since it is already supporting the STU 3, which is not the case for Spark that still supports DSTU2.

The full FHIR server implementation provided by HAPI that is used in this project is freely available for downloads on GitHub[29]. The server implementation is fully written in java using the dedicated FHIR-HAPI server library. The project downloaded can be modified and can be installed via command line. Once the project is installed the .war file can be generated with the command line as well. This .war file is the only needed artifact to deploy the project on a chosen server.

[26] At http://213.136.91.24:8080/hapi-fhir-jpaserver-example/

[27] https://fhirtest.uhn.ca/

[28] http://spark.furore.com/

[29] https://github.com/jamesagnew/hapi-fhir

Very little configurations had to be made to the downloaded FHIR server project. The only two configurations necessary, for the client application to work properly, were to define the FHIR base URL and to enable external references on the server.

```
public DaoConfig daoConfig() {
        DaoConfig retVal = new DaoConfig();
        retVal.setSubscriptionEnabled(true);
        retVal.setSubscriptionPollDelay(5000);
        retVal.setSubscriptionPurgeInactiveAfterMillis(DateUtils.MILLIS_PER_HOUR);

        retVal.setAllowMultipleDelete(true);

        // Allow external references
        retVal.setAllowExternalReferences(true);

        retVal.getTreatBaseUrlsAsLocal().add("http://213.136.91.24:8080/hapi-fhir-jpaserver-example/baseDstu3");

        return retVal;
        }
```

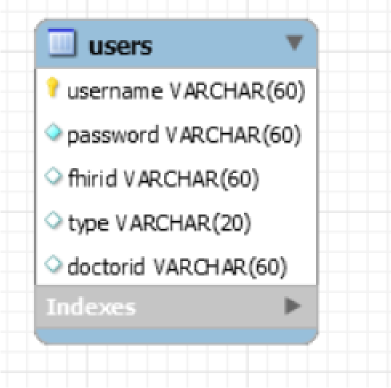
Listing 2 - FHIR server configuration file based on http://hapifhir.io/doc_jpa.html

Setting up a FHIR test server on a chosen server was surprisingly easy. HAPI's open-source implementation is very complete and documentation is available for the configurations to be made. Further configurations could, for example, include setting up an authorization protocol. HAPI provides a very satisfying and nicely designed piece of software.

4.4.3 Database

The open source database server MySQL is used for the database. The database only contains information related to the users of the application. Since the FHIR server provides a storage for the resources it is not necessary for them to be stored in the MySQL database.

The unique table of the database contains five columns. “Username” and “Password” are used for the login functionality. The “fhirid” column stores the FHIR resource's id of a user. “Type” determines if the user is a patient, a doctor or an administrator. The “doctorid” is only used when references mechanism of the server does not allow for Patient resources to reference Practitioner resources.



users

username VARCHAR(60)

password VARCHAR(60)

fhirid VARCHAR(60)

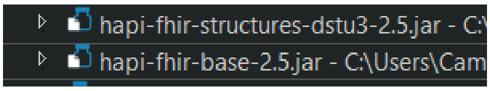
type VARCHAR(20)

doctorid VARCHAR(60)

Figure 34 - MySQL Database schema, source: author's data

4.5 Implementing FHIR client with HAPI-FHIR library

Setting up the library is quickly done, it requires at least the HAPI-FHIR core JAR and the "structures" JAR to be included in the java project. The second one contains all the different classes for the FHIR resources and datatypes definition. The figure bellows shows an abstract of the java project's build path with the two requested JARs.



hapi-fhir-structures-dstu3-2.5.jar - C:\

hapi-fhir-base-2.5.jar - C:\Users\Cam

Figure 35 - HAPI FHIR app Java Build Path, source: author's data

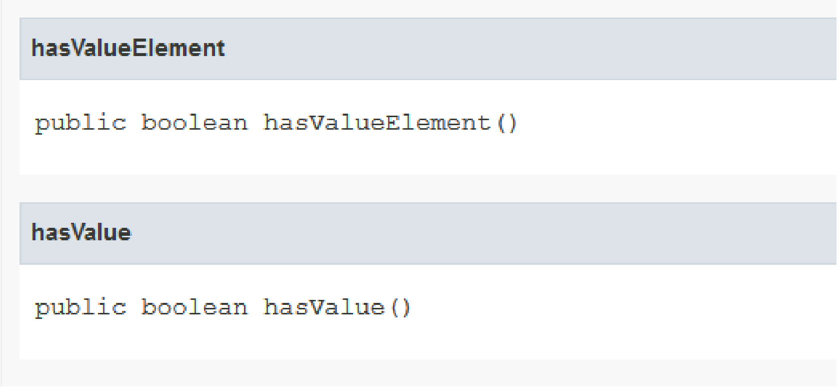
Once the library is included in the project, it can be used for multiple purposes such as connecting to the server or creating java objects to handle FHIR resources.

Implementing a web application communicating with a FHIR server turned out to be easier than expected. It seemed to be a tougher task due to the lack of source code examples for FHIR STU3. Indeed, most of code snippets provided by HAPI's website are only valid for DTSU2, important modifications have been made in terms of datatypes, methods names and parameters in the STU3 supporting library. Fortunately, Javadoc[30] is available for all versions of the library and helped us enormously throughout the implementation.

The methods and attributes of the classes are commented but should be further detailed. It is often quite difficult to understand the purpose of the methods and how they should be used.

[30] http://hapifhir.io/apidocs-dstu3/index.html

As an illustration, the figure bellow shows the Javadoc for two methods of the class "ContactPoint" representing a way of contacting a person (email, telephone…) that is stored in the telecom attribute of a person's resource. It is not explicitly explained what those two methods do and thus, it is difficult to understand the difference between them and to know which one applies when the client application needs to retrieve a phone number or an email address from a Person resource.



hasValueElement

```
public boolean hasValueElement()
```

hasValue

```
public boolean hasValue()
```

Figure 36 - HAPI STU 3 Javadoc ('ContactPoint (HAPI FHIR Structures - DSTU3 2.5 API)', n.d.)

4.5.1 Application server connection

Once the environment is set up, the coding of the interaction with a FHIR server can start. The HAPI library allowed us to execute this task swiftly.

To prevent the application from downloading server's metadata/conformance statement, it is important to disable it before creating a FHIR client. The conformance statement was unreachable on most public test servers and made the connection impossible.

The following snippet shows the class created for the connection to the server. The last two lines of the snippet increase the socket and connection timeouts to ensure a minimum value to enable the application to retrieve resources.

```
private String serverBaseUrl = "http://213.136.91.24:8080/hapi-fhir-
jpaserver-example/baseDstu3";

private FhirContext ctx;

public Fhircontextconnection() {
               this.ctx = FhirContext.forDstu3();

               // Disable server validation
               ctx.getRestfulClientFactory().setServerValidationMode(Server
               ValidationModeEnum.NEVER);

               // increase the timeout
               ctx.getRestfulClientFactory().setConnectTimeout(60 * 1000);
               ctx.getRestfulClientFactory().setSocketTimeout(60 * 1000);

               // create the RESTful client to work with our FHIR server
               IGenericClient client =
               ctx.newRestfulGenericClient(serverBaseUrl);
       }
```

Listing 3 - FHIR connection java class

4.5.2 Understanding the FHIR resources

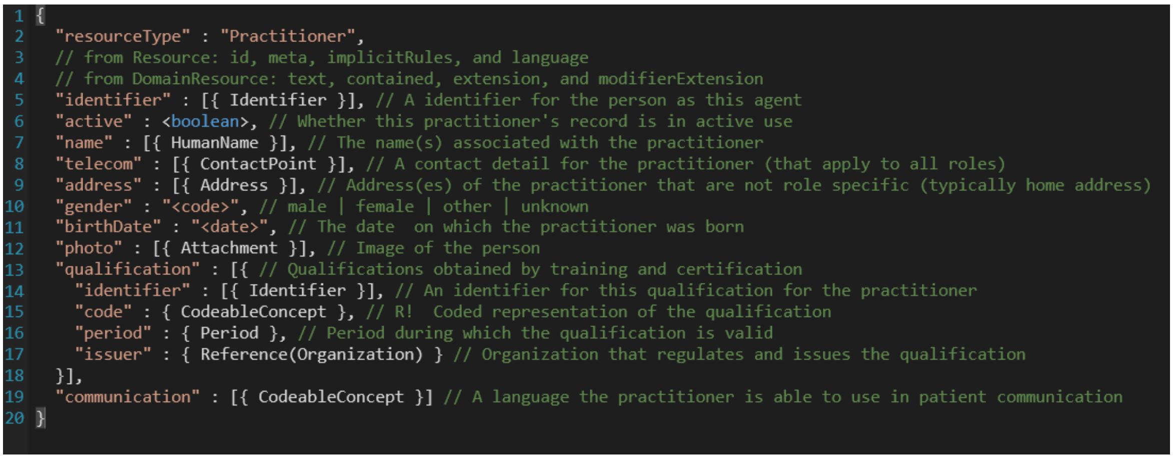
The application use cases require to manipulate three FHIR resources. The Patient, the Practitioner and the Observation resources. The first one is quite self-explanatory: The Patient resource stores patients related information as illustrated in multiples examples of this paper.

The first step towards manipulating the needed resources is to understand what their purposes are and how they should be used. In that sense, the FHIR specification provides descriptions of each resource as well as code samples. The resource attributes are also listed, detailed and their cardinalities are shown to see whether they are required or not. The code samples can be found for all resources formats supported by FHIR. As of this work, the decision has been made to work with JSON, therefore the following examples will be in that format.

The Practitioner resource is detailed in the FHIR specification and is descripted as covering "[…] all individuals who are engaged in the healthcare process and healthcare-related services as part of their formal responsibilities and this Resource is used for attribution of activities and responsibilities to these individuals."(HL7 International, n.d.-o).

In this project, the Practitioner resource will be used to save and exchange data related to the doctors of the application, the general practitioner of a patient. According to FHIR, other example uses could include representing a dentist, a nurse, a medical technician, a pharmacist or any other healthcare worker.

The abstract of code below shows what a Practitioner resource is made of. It does not differ that much from the Patient resource but for the “qualification” attribute.



```
{
  "resourceType" : "Practitioner",
  // from Resource: id, meta, implicitRules, and language
  // from DomainResource: text, contained, extension, and modifierExtension
  "identifier" : [{ Identifier }], // A identifier for the person as this agent
  "active" : <boolean>, // Whether this practitioner's record is in active use
  "name" : [{ HumanName }], // The name(s) associated with the practitioner
  "telecom" : [{ ContactPoint }], // A contact detail for the practitioner (that apply to all roles)
  "address" : [{ Address }], // Address(es) of the practitioner that are not role specific (typically home address)
  "gender" : "<code>", // male | female | other | unknown
  "birthDate" : "<date>", // The date  on which the practitioner was born
  "photo" : [{ Attachment }], // Image of the person
  "qualification" : [{ // Qualifications obtained by training and certification
    "identifier" : [{ Identifier }], // An identifier for this qualification for the practitioner
    "code" : { CodeableConcept }, // R!  Coded representation of the qualification
    "period" : { Period }, // Period during which the qualification is valid
    "issuer" : { Reference(Organization) } // Organization that regulates and issues the qualification
  }],
  "communication" : [{ CodeableConcept }] // A language the practitioner is able to use in patient communication
}
```

Figure 37 - Practitioner JSON example, source: (HL7 International, n.d.-o)

The “Observation” resource is defined by the FHIR specification to be “[…] a central element in healthcare, used to support diagnosis, monitor progress, determine baselines and patterns and even capture demographic characteristics.” (HL7 International, n.d.-l). The FHIR specification presents some uses for the Observation resource among which the usage that will be made of this resource is found: “Vital signs such as body weight, blood pressure, and temperature” (HL7 International, n.d.-l).

“Vital signs” is in fact a core profile defined by FHIR for the Observation resource. It redefines the Observation resource with further required elements: a status, a vital-signs code, a measure value, a patient, a time and a numeric result value.

Three profiles of the Vital signs are important for the purpose of the test application. The data, found in the FHIR specification, relative to the said profiles has been aggregated in following the table. The table indicates the fixed codes to be used for the three Observations.

Profile Name	vital-signs code (LOINC)	LOINC Name and Comments	UCUM Unit Code
Heart rate	8867-4	*Heart rate* - To supplement this vital sign observation, 8887-2 - *Heart rate device type* MAY be used as an additional observation.	/min
Body height	8302-2	*Body height*	cm, [in_i]
Body weight	29463-7	*Body weight* - To supplement this vital sign observation, 8352-7 - *Clothing worn during measure* and 8361-8 - *Body position with respect to gravity* MAY be used as additional observations.	g, kg, [lb_av]

Table 3 - Vital-signs profiles adapted from (HL7 International, n.d.)

4.5.3 Manipulating the resources

Once the concepts attached to each resource that will be manipulated is understood, the implementation can start. In the understanding phase, it is primordial to carefully look at the requirements for each resource so nothing is left out during the implementation. It is also very important to ensure that the correct resources are used for the correct data. In a matter of interoperability, it is necessary to indicate which units, codes and references are used.

The application use cases require that the application performs the following interactions: create, update, search and delete a Patient and an Observation resource. The practitioner resources only require creation and search operations.

Many code examples are available on the HAPI platform[31] for the patient resource since it is the central element of a FHIR implementation. As of the two other types of resources used in the application, only one example of code for the observation resource's creation was found

[31] http://hapifhir.io/

on the platform. However, operations on resources work the same way regardless of their types. The only variable is the attributes of the resources and their datatypes.

In the subchapters below, the different FHIR operations used on the project's resources are presented. The snippets of codes given as examples are extracted from the application's code. Some are based on the example found on the HAPI platform.

4.5.3.1 Creation

The abstract of code of listing 4 shows the patient resource's creation in the application. It is noticeable that the creation is simple and quick to perform. A patient object is created and assigned with the desired attributes. The identifier set is not the id of the resource, it is the simulation of a MRN. After the attributes' assignments, the reference with the Practitioner resource representing the patient's doctor is instantiated and assigned to the patient. When the patient object contains all needed elements, the resource can be created on the server with the *client.create()* method call (*client* being HAPI's restful client).

Once the patient resource is on the server, the three vital-signs observations for the patient are created. Finally, an entry is added to the database with the patient's username and password so the patient (user) will be able to log into the application.

```
this.patient = new Patient();

Random randomGenerator = new Random();
int randomInt = randomGenerator.nextInt(10000000);

//set patient base info
this.patient.addName().addPrefix(prefix).setFamily(lastname)
.addGiven(firstname);
this.patient.setBirthDate(this.birthdate);
this.patient.addIdentifier().setSystem("tb:fhir")
.setValue("CP" + randomInt);

 //add the practitioner resource to the patient
this.getSelectedDoctorbyID();
this.patient.addGeneralPractitioner();
List<Reference> ref = new ArrayList<Reference>();
ref.add(new Reference(this.doctor));
this.patient.setGeneralPractitioner(ref);

MethodOutcome outcome =
client.create().resource(this.patient).prettyPrint().encodedJson().execute();

 IdType id = (IdType) outcome.getId();
this.patientid = id.getIdPart();
 // create the observations for the patients
this.createOBSforPatient();
// create an account for the patient in the DB so he can login later
this.createPatientAccount();
```

Listing 4 - Patient resource creation

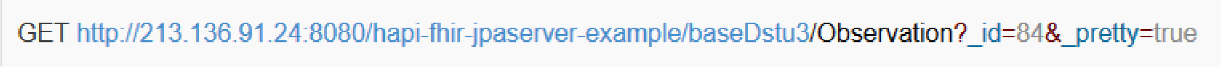
Creating an observation resource works the same way, the method *client.create()* is used and takes the resource as parameter. A noticeable difference is that no narrative attribute will be generated for the observation as it is for the patient resource. Although a HAPI built-in narrative generator exists, the HAPI FHIR server does not take it into account for the Observations. Consequently, a narrative and a status must be added programmatically to the Observation resources as shown in lines 3 to 6 of the following snippet.

As explained earlier, the application deals with three vital-signs Observations. The creation mechanism is the same for all three of them. The attributes' values of the observation object will nonetheless change according to the vital it is recording. The code of listing 5 shows the creation of the "Body weight" vital-sign.

```
Observation observation = new Observation();

observation.setSubject(new Reference(patient));
observation.setStatus(ObservationStatus.FINAL);

Narrative text = new Narrative();
observation.getText().setStatus(Narrative.NarrativeStatus.GENERATED);
observation.getText().setDivAsString("<div>Vital sign Observation<br/> of patient : "+ patient.getNameFirstRep().getFamily() + "</div>");
observation.setIssued(new Date());

// Give the observation a code (what kind of observation is this)
Coding coding = observation.getCode().addCoding();

System.out.println("creating a bodyweight resource");
coding.setCode("29463-7").setSystem("http://loinc.org").setDisplay("Body Weight");

 // Create the quantity
Quantity value = new Quantity();
 value.setValue(bodymeasure)
.setSystem("http://unitsofmeasure.org").setCode("kg");
 observation.setValue(value);

 // Set the reference range
SimpleQuantity low = new SimpleQuantity();
 low.setValue(35)
.setSystem("http://unitsofmeasure.org").setCode("kg");
 observation.getReferenceRangeFirstRep().setLow(low);

SimpleQuantity high = new SimpleQuantity();
 high.setValue(90)
.setSystem("http://unitsofmeasure.org").setCode("kg");
 observation.getReferenceRangeFirstRep().setHigh(high);

 //creation of the resource on the server
 MethodOutcome outcome = client.create().resource(observation)
 .prettyPrint().encodedJson().execute();
 //get the returned ID of newly created resource
 IdType id = (IdType) outcome.getId();
```

Listing 5 - Observation resource creation based on http://hapifhir.io/doc_rest_client_examples.html

Now, if a request is made to the server for the resource created above with the following query:

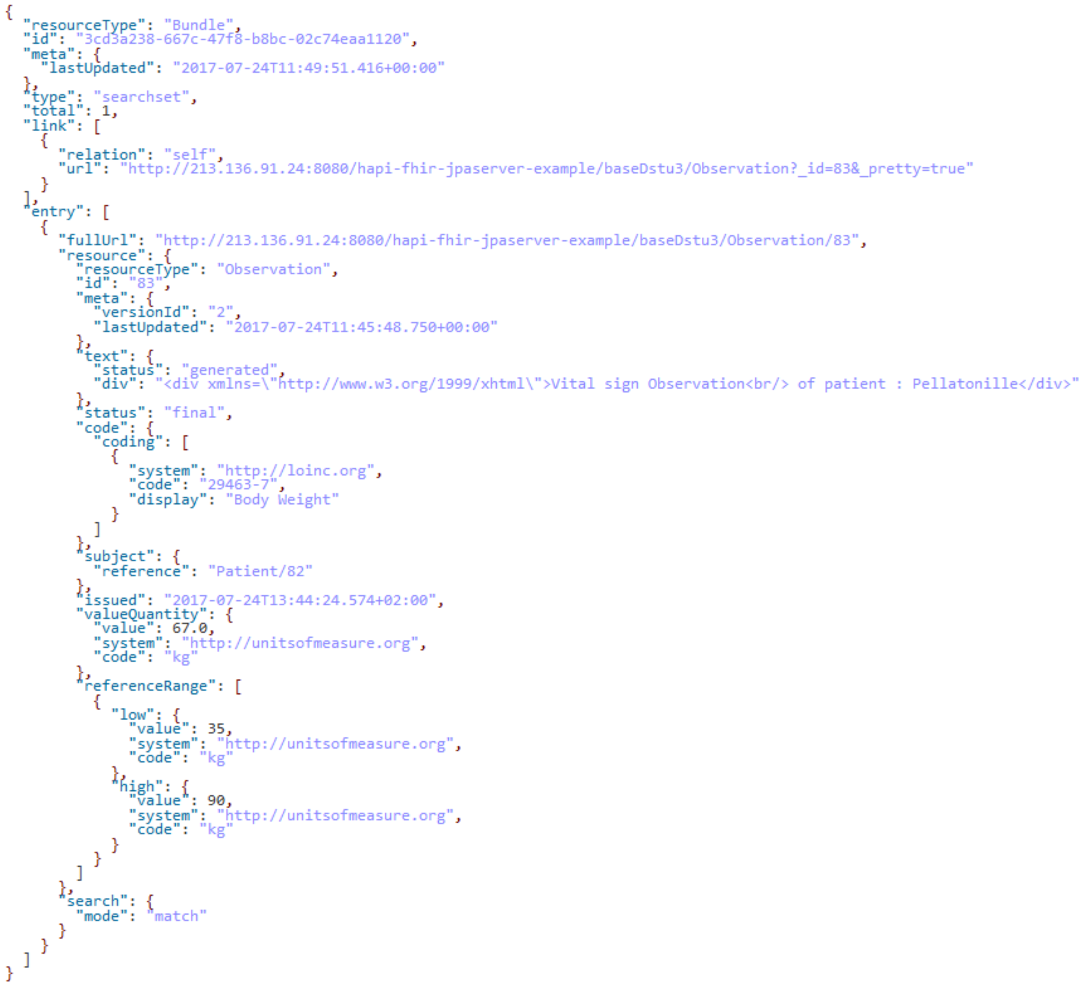


```
GET http://213.136.91.24:8080/hapi-fhir-jpaserver-example/baseDstu3/Observation?_id=84&_pretty=true
```

Figure 38 - FHIR search query

The server will reply with the bundle of figure 39. As figure 39 shows, the resource contains the required FHIR elements for a vital-sign, as explained in chapter 4.5.2: the status that is here “final”, a vital-signs code “29463-7”, a measure value “kg”, a patient referenced by its

URL “Patient/78”, a datetime contained in the “issued” attribute and a numeric result value “67.0” representing the weight observation made on the patient.



```
{
  "resourceType": "Bundle",
  "id": "3cd3a238-667c-47f8-b8bc-02c74eaa1120",
  "meta": {
    "lastUpdated": "2017-07-24T11:49:51.416+00:00"
  },
  "type": "searchset",
  "total": 1,
  "link": [
    {
      "relation": "self",
      "url": "http://213.136.91.24:8080/hapi-fhir-jpaserver-example/baseDstu3/Observation?_id=83&_pretty=true"
    }
  ],
  "entry": [
    {
      "fullUrl": "http://213.136.91.24:8080/hapi-fhir-jpaserver-example/baseDstu3/Observation/83",
      "resource": {
        "resourceType": "Observation",
        "id": "83",
        "meta": {
          "versionId": "2",
          "lastUpdated": "2017-07-24T11:45:48.750+00:00"
        },
        "text": {
          "status": "generated",
          "div": "<div xmlns=\"http://www.w3.org/1999/xhtml\">Vital sign Observation<br/> of patient : Pellatonille</div>"
        },
        "status": "final",
        "code": {
          "coding": [
            {
              "system": "http://loinc.org",
              "code": "29463-7",
              "display": "Body Weight"
            }
          ]
        },
        "subject": {
          "reference": "Patient/82"
        },
        "issued": "2017-07-24T13:44:24.574+02:00",
        "valueQuantity": {
          "value": 67.0,
          "system": "http://unitsofmeasure.org",
          "code": "kg"
        },
        "referenceRange": [
          {
            "low": {
              "value": 35,
              "system": "http://unitsofmeasure.org",
              "code": "kg"
            },
            "high": {
              "value": 90,
              "system": "http://unitsofmeasure.org",
              "code": "kg"
            }
          }
        ]
      },
      "search": {
        "mode": "match"
      }
    }
  ]
}
```

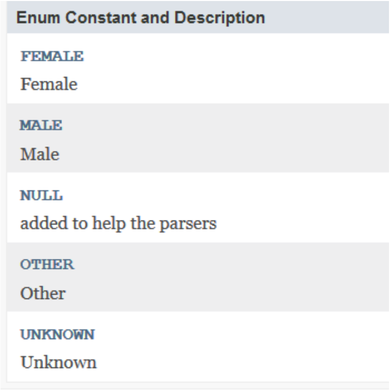
Figure 39 – Bundle of Body weight resource on the server

The last resource type created is the Practitioner. The creation mechanic is once again the same. Attributes unused so far have been added to the practitioner object to test the library’s capabilities. It has four attributes that differ from the Patient, as presented on listing 6. The first one is the gender, then an “active” value telling weather the Practitioner is still working and, finally, some contact information: an email address and a telephone number.

```java
this.practitioner = new Practitioner();

Random randomGenerator = new Random();
int randomInt = randomGenerator.nextInt(10000000);

this.practitioner.addName().addPrefix(this.prefix).setFamily(this.lastname).addGiven(this.firstname);
this.practitioner.setBirthDate(this.birthdate);
this.practitioner.addIdentifier().setSystem("tb:fhir").setValue("CP" + randomInt);

this.practitioner.setActive(true);
if (this.gender.equals("F"))
this.practitioner.setGender(AdministrativeGender.FEMALE);

if (this.gender.equals("M"))
this.practitioner.setGender(AdministrativeGender.MALE);

ContactPoint email = new ContactPoint();
 email.setSystem(ContactPoint.ContactPointSystem.EMAIL);
email.setValue(this.email);

ContactPoint tel = new ContactPoint();
 tel.setSystem(ContactPoint.ContactPointSystem.PHONE);
tel.setValue(this.telephone);

ArrayList<ContactPoint> telecom = new ArrayList<ContactPoint>();
 telecom.add(email);
 telecom.add(tel);

this.practitioner.setTelecom(telecom);

boolean userExists = DAO.userExists(this.email);

if(!userExists){
 //create the resource on the server
MethodOutcome outcome = client.create().resource(this.practitioner)
 .prettyPrint().encodedJson().execute();

 IdType id = (IdType) outcome.getId();
System.out.println("Resource is available at: " + id.getValue());
this.practitioner.setId(outcome.getId());

 //add the doctor in the DB
DAO.addPractitionerAccount(this.email, this.password,
this.practitioner.getIdElement().getIdPart());
 }
```

Listing 6 - Practitioner resource creation, source: application code

To use methods to add attributes to objects, the right information in the right type must be given as arguments. For example, to set the gender of the Practitioner object to female, the documentation defines that a value of type “enum Enumerations.AdministrativeGender” must be given to the *setGender()* method (University Health Network, n.d.-e). Then, by studying the enumeration, the different constants it is holding can be found:



Enum Constant and Description
FEMALE Female
MALE Male
NULL added to help the parsers
OTHER Other
UNKNOWN Unknown

Figure 40 - Constants of gender enum (University Health Network, n.d.-a)

Knowing the constants, the right information can be set to the resource, being here the “FEMALE” value. This way, the attribute gender is protected against false values assignments.

For java developers or developers used to object oriented programming, creating resources should not be a burden.

4.5.3.2 Read

To retrieve the current version of a FHIR resource one can use the read interaction. It is simple to perform:

```
this.patient = client.read()
.resource(Patient.class).withId(this.logicalID)
.execute();
```

Listing 7 - HAPI patient resource read interaction

4.5.3.3 Update

Updating a resource works the same way as creating one. Instantiate a java object with a resource retrieved from the server, then set all its attributes and finally update the resource on the server with the following lines of code:

```
MethodOutcome outcome = client.update()
.resource(this.patient).prettyPrint().encodedJson().execute();
```

Listing 8 - Patient update

4.5.3.4 Deletion

The delete operation is the fastest to implement. A resource cannot be deleted while other resources reference it. Consequently, the deletion of the resources referencing a resource has to be performed prior to deleting the resource itself. Deletion requires very little coding. For instance, the patient deletion:

```
VitalSignsHandler vh = new VitalSignsHandler();
vh.deleteobs(fhirid);

//deletion of the resource
client.delete().resourceById(new IdDt("Patient", fhirid)).execute();
```

Listing 9 - patient deletion

4.5.3.5 Search

As seen previously in this paper, FHIR RESTful API defines many different search operations. HAPI library makes it easy to search for resources with specific parameters. As an example, the code below shows how to retrieve all the patient resources that have the same doctor. When giving an Id as a parameter care should be taken to only use the portion of the id containing the actual numeric value (not the full URL). When requesting resources form the server, the reply will be in the form of a bundle even if only one resource is returned by the search query.

```
Bundle response = client.search().forResource(Patient.class)
                .where(new ReferenceClientParam("general-
                practitioner").hasId(doctorID)).prettyPrint()
                .returnBundle(Bundle.class).execute();
```

Listing 10 – patients search with one parameter

To add a parameter to the search of the figure above, simply add a where clause. For instance, if a client wants to retrieve all the patients of a given doctor that are called a certain name, it can add the following line of code to the query:
.where(Patient.FAMILY.matches().values(this.lastname)).

It is also possible to search using an URL, in the following code a URL is used to retrieve a specific type of vital-sign resource. Two search parameters are specified: the LOINC code and the subject which represents the patient whose observations it is.

```
String searchUrl = "Observation?subject=" +
          patient.getIdElement().getIdPart() + "&code=" + rcode
          + "&_pretty=true";

Bundle response = client.search()
          .byUrl(searchUrl).returnBundle(Bundle.class).execute();
```

Listing 11 - Searching by URL

HAPI has a strong and simple search mechanic, as seen on the previously presented snippets. The only delicate part is giving the right information in the correct data type to the search. To ensure this, the documentation should be thoroughly consulted.

4.5.3.6 Vread

The vread operation retrieves a specific version of a resource. The application performs a vread operation on the body weight Observation to obtain the weight of a patient over time. The demographics retrieved are then displayed in a dynamic graph as shown in appendix III. To achieve this, HAPI provides the method *client.read()* to which the FHIR id and the version of the resource have to be given.

```
Observation bw = this.Obodyweight;

String v = bw.getMeta().getVersionId();
int version = Integer.parseInt(v);

ArrayList <Observation> obs = new ArrayList<Observation>();

for(int i = version; i>0; i--){
   String currentversion = String.valueOf(i);
       Observation ob = client.read()
                         .resource(Observation.class)
                         .withIdAndVersion(bw.getIdElement().getId
                          Part(),currentversion)
                          .execute();

     obs.add(ob);
 }

this.observations = obs;
```

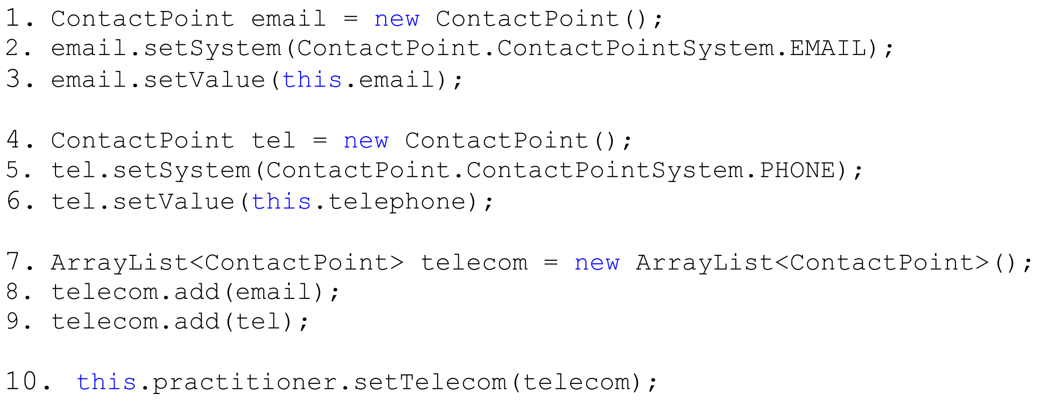
Listing 12 -vread interaction

Like the other HAPI operations, vread is simple to set up. FHIR define a history interaction as follows "*The history interaction retrieves the history of either a particular resource, all* *resources of a given type, or all resources supported by the system*"(HL7 International, n.d.-h). HAPI does provide a *history()* method for its generic client and allows to retrieve the entire history of resources all at once which would have been a better fit for this mean, but it is lacking documentation and could not be correctly set up.

4.5.3.7 HAPI's evaluation

A challenging aspect of working with the HAPI library is finding the methods that retrieve or set the wanted attributes of a resource object. When starting to work with the library it can be a time-consuming task. Once identified, using them often takes further researching. The library defines numerous datatypes that engender difficulties to work with HAPI methods since they require or return unfamiliar datatypes (HAPI objects, enumerations…).

As an illustration, let's consider the assignments of a telephone number and an e-mail address to a practitioner object. Firstly, the documentation says that the *setTelecom*() method should be used. The *setTelecom()* method needs a list of objects of type *ContactPoint* as parameter. Each *ContactPoint* then needs to be assigned with two values: a *system* and a *value* by using the *setSystem()* and the *setValue()* methods. Those methods also have specific datatypes as parameters. Consequently, the simple adding of a telephone number requires quite a lot of searching and trials before it is fully working. The following snippets shows in code the above explanation.

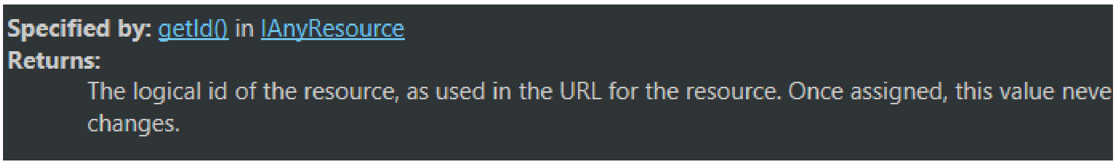


```
ContactPoint email = new ContactPoint();
email.setSystem(ContactPoint.ContactPointSystem.EMAIL);
email.setValue(this.email);

ContactPoint tel = new ContactPoint();
tel.setSystem(ContactPoint.ContactPointSystem.PHONE);
tel.setValue(this.telephone);

ArrayList<ContactPoint> telecom = new ArrayList<ContactPoint>();
telecom.add(email);
telecom.add(tel);

this.practitioner.setTelecom(telecom);
```

Figure 41 - setting a telephone number attribute

For instance, when the application needs to retrieve the usual last name of a patient, the following code will have to be executed: *getNameFirstRep().getFamily().* This is not very intuitive, at first, it is tempting to use instead the *getName()* method of the patient object. However, the *getName()* method returns a list of "HumanName" whilst *getNameFirstRep()* directly returns the first entry. Indeed, FHIR allows patient to have several different given names and lastname.

The same remark applies to the id attribute. When a vread operation is performed, for instance, the id of the FHIR resource to retrieve has to be given as parameter with this code: *getIdElement().getIdPart().* Resources objects have a method called *getId()* that is tempting to be used instead:



Specified by: getId() in IAnyResource
Returns:
The logical id of the resource, as used in the URL for the resource. Once assigned, this value neve
changes.

Figure 42 - getId() method Javadoc

The problem is that the *getId()* method returns a full URL, not only the numerical id used in the URL, and that the Vread operation only needs the numeric portion of the id. The documentation attached to the method is not very clear on this, neither is the requirement for a numerical value of the vread parameters. Therefore, the library is disparaged because of its lack of penetrability[32] for some methods and classes.

The examples above illustrate the main disadvantage of HAPI and its best advantage at the same time: it contains a lot of classes that are full of functionalities. The great number of classes does not reduce the quality of the API since the naming of most classes, methods and properties give a good understanding of their capabilities. Also, it is not the consequence of a poor design pattern, classes are constructed in a hierarchical and modular manner.

Programming with HAPI becomes a lot easier over time, once the classes have been used for one resource object, the mechanism is similar for the others. This factor is an important point in evaluating the usability of a library.

The work done by the University Health network to provide this HAPI API is highly remarkable. Indeed, not only the API is well designed but an incredible amount of FHIR

[32] The simplicity with which the API can be explored, analysed and its components understood (Clarke, n.d.)

features are provided. The FHIR features tested such as the different server’s API interactions or server connection represent a substantial gain in time. Even when using a limited amount of the available capabilities of the API, it still is a valuable addition to a FHIR implementation since it is quickly and easily installed.

Conclusion

The analysis of the current state of the FHIR specification showed how vast FHIR capabilities are. As FHIR uses well-known technologies for the representation of data, FHIR's data model is straightforward to apprehend. However, the complexity and variety of medical use cases engendered the design of a large data model with complex linkage. The application prototype demonstrated that it is not tedious to work with FHIR resources. The use made of FHIR in this work does confirm that FHIR is a fast-implementable technology, indeed with the usage of existing tools, it is possible to rapidly create a valuable and FHIR compatible piece of software. Nevertheless, account must be taken that the FHIR client implemented only makes use of three types of ressources within a testing environment, a real-world use would require much more effort. The prototype was tested against various FHIR RESTful servers. As a result, it was possible to observe that adaptations should be made to an application so it can work with different server implementations. Implementing FHIR requires to carefully look at the server capabilities since available public APIs might not offer the functionalities needed by an application.

The HAPI library has proved itself very usable and useful. The main problems encountered during the implementation were the lack of clarity and explanations of the documentation that slowed down the development on some occasions. Despite this problem, the library greatly helps the development of a FHIR-based application. Indeed, the FHIR data model is already represented in its entirety in the library and many built-in mechanism, such as the implemented FHIR client class, allow to save precious time.

In addition, the analysis showed the interests of the REST API approach that FHIR takes on clinical data exchange. This client-server, query-response, architectural style offers to FHIR-based applications easy access to data exposed either through private intra-hospital interfaces or public interagency interfaces regardless of the vendors of the systems exchanging the data. Moreover, this light-weight communication makes FHIR well suited for mobile use which is an increasing need in the modern healthcare landscape.

FHIR does not attempt to reinvent the wheel by relying on many existing technologies, that have already proven themselves (e.g. OAuth2) adding valuable features to the specification.

A major benefit of FHIR evidenced in this work is its granularity. By providing its XML or JSON resources, FHIR enables care providers to access smaller portion of data than existing standard based upon document exchange, leading to considerable time saving.

FHIR represents a great opportunity for healthcare interoperability. Not only is it easier to implement than existing standards, it is also more inclined to provide semantical interoperability. FHIR has many advantages over HL7 existing standards, it provides real-time data access, allows just about the right amount of flexibility and is not as complicated nor requires as much expertise and money to be implemented.

Since FHIR is recent it is available as a standard for trial use, consequently hindsight is lacking to identify its drawbacks. Due to the standard's recency it has been difficult to find documentation to support the analysis. Today, as the standard is still evolving, it refrains the adoption of the standard. As explained, the FHIR specification is open and freely available which helps creating a community amongst which implementations are shared, significantly helping developers to implement a FHIR solution.

This work could continue by testing further the FHIR specifications, especially by taking more complex use cases in order to identify some of the standard's drawbacks. For instance, by including an authentication protocol to the FHIR application or by implementing a validation mechanism to validate the created resources. Integrating FHIR in an EHR system using SMART on FHIR could also be an interesting extension to this work.

The main difficulty of this work was to synthetize the analysis of the FHIR specification since it is huge and defines numerous elements. It was sometimes difficult to stay focus on the objectives of the work and not get lost in immensity of the FHIR subject.

On a personal note, this work has enriched my knowledge in many domains. I learned a lot on subjects that interest me such as healthcare and its information systems interoperability, java programming and web technologies. This project has also allowed me to improve many of my soft skills, especially self-discipline, perseverance and efficiency.

References

Argonaut Project. (2017, February 15). Argonaut Data Query IG. Retrieved 24 May 2017, from http://www.fhir.org/guides/argonaut/r2/index.html

Beeler, G. W. (n.d.). *Introduction to: HL7 Reference Information Model (RIM) [Microsoft Powerpoint]*.

Bender, D., & Sartipi, K. (2013). HL7 FHIR: An Agile and RESTful approach to healthcare information exchange. In *Proceedings of the 26th IEEE International Symposium on Computer-Based Medical Systems* (pp. 326–331). https://doi.org/10.1109/CBMS.2013.6627810

Benson, T., & Grieve, G. (2016). *Principles of Health Interoperability: SNOMED CT, HL7 and FHIR*. Springer.

Clarke, S. (n.d.). Measuring API Usability. Retrieved 28 July 2017, from http://www.drdobbs.com/windows/measuring-api-usability/184405654

client-py: Python SMART on FHIR client. (2017). Python, SMART on FHIR. Retrieved from https://github.com/smart-on-fhir/client-py (Original work published 27 August 2014)

ContactPoint (HAPI FHIR Structures - DSTU3 2.5 API). (n.d.). Retrieved 16 July 2017, from http://hapifhir.io/apidocs-dstu3/index.html

Corepoint Health. (n.d.). HL7 ORU – HL7 Result Message (Observation Result). Retrieved 25 July 2017, from https://corepointhealth.com/resource-center/hl7-resources/hl7-oru-message

Datar, K. (2016, October 14). Helping Healthcare Catch FHIR - HL7 Standards & the Future of Health Data | ExtraHop. Retrieved 11 June 2017, from https://www.extrahop.com/company/blog/2016/hl7-fhir-hl7-standards-interoperability-future-of-healthcare/

Doglas, B. K. (n.d.). Service-Oriented Architecture (SOA) Definition. Retrieved 26 July 2017, from http://www.service-architecture.com/articles/web-services/service-oriented_architecture_soa_definition.html

Douglas, B. K. (n.d.). Representational State Transfer (REST). Retrieved 26 July 2017, from http://www.service-architecture.com/articles/web-services/representational_state_transfer_rest.html

Elkstein, D. M. (n.d.). 1. What is REST? Retrieved 26 July 2017, from http://rest.elkstein.org/2008/02/what-is-rest.html

FHIR Developer Days. (2014, November). *FHIR API for .Net programmers by Mirjam Baltus*. Santé. Retrieved from https://fr.slideshare.net/DevDays2014/04-b-fhir-api-for-net-mirjam

Fielding, T. R. (2000). *Architectural Styles and the Design of Network-based Software Architectures.* (Doctoral dissertation). University of California, Irvine. Retrieved from https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Furore. (2017, April 11). fhirnetapi.pdf. Retrieved 18 May 2017, from https://media.readthedocs.org/pdf/fhirnetapi/latest/fhirnetapi.pdf

Furore. (n.d.). Spark FHIR Server. Retrieved 18 May 2017, from https://fhir.furore.com/spark-server/

Grieve, G. (2012, December 29). Database schema for my FHIR server. Retrieved 29 May 2017, from http://www.healthintersections.com.au/?p=1299

Hay, D. (2014, March). *FHIR Profiles*. Santé. Retrieved from https://fr.slideshare.net/DavidHay5/fhir-profiles

Hay, D. (n.d.). FHIR is Sparking Innovation in Health Information Sharing. Retrieved 9 June 2017, from https://orionhealth.com/us/knowledge-hub/blogs/fhir-is-sparking-innovation-in-health-information-sharing-how-do-you-become-a-part-of-the-fhir-phenomenon/

Health Samurai Team. (n.d.-a). Aidbox. Retrieved 1 June 2017, from https://aidbox.io/index.html

Health Samurai Team. (n.d.-b). Aidbox features. Retrieved 1 June 2017, from https://aidbox.io/docs/Features.html

Health Samurai Team. (n.d.-c). Demo [demo]. Retrieved 29 May 2017, from http://fhirbase.github.io/demo/index.html#/4-show-patients-table

Health Samurai Team. (n.d.-d). FHIRbase. Retrieved 29 May 2017, from http://fhirbase.github.io/

Health Samurai Team. (n.d.-e). FHIRbase tutorial. Retrieved 29 May 2017, from http://fhirbase.github.io/demo/tutorial.html

HIMSS. (2013). *HIMSS Dictionary of Healthcare Information Technology Terms, Acronyms and Organizations* (3rd ed.).

HL7. (2002). v3 guide [doc]. Retrieved from https://www.hl7.org/documentcenter/public_temp_BC6CF574-1C23-BA17-0CD6BD73DA3AF236/wg/mnm/hdf/v3guide.doc

HL7 International. (2010). hl7 v2 messaging conformance [jpg]. Retrieved 25 July 2017, from https://image.slidesharecdn.com/hl7v2messagingconformancejan2011-110303095217-phpapp02/95/hl7-v2-messaging-conformance-jan-2011-15-728.jpg?cb=1299146151

HL7 International. (n.d.-a). About Health Level Seven International. Retrieved 25 July 2017, from http://www.hl7.org/about/index.cfm?ref=common

HL7 International. (n.d.-b). Comparison-v2 - FHIR v3.0.1. Retrieved 25 July 2017, from https://www.hl7.org/fhir/comparison-v2.html

HL7 International. (n.d.-c). Composition - FHIR v3.0.1. Retrieved 25 July 2017, from https://www.hl7.org/fhir/composition.html

HL7 International. (n.d.-d). Documents - FHIR v3.0.1. Retrieved 25 July 2017, from https://www.hl7.org/fhir/documents.html

HL7 International. (n.d.-e). DomainResource - FHIR v3.0.1. Retrieved 13 June 2017, from https://www.hl7.org/fhir/domainresource.html

HL7 International. (n.d.-f). Formats - FHIR v3.0.1. Retrieved 13 June 2017, from https://www.hl7.org/fhir/formats.html

HL7 International. (n.d.-g). Foundation-module - FHIR v3.0.1. Retrieved 25 July 2017, from https://www.hl7.org/fhir/foundation-module.html#uses

HL7 International. (n.d.-h). Http - FHIR v3.0.1. Retrieved 26 July 2017, from https://www.hl7.org/fhir/http.html

HL7 International. (n.d.-i). Message-request-link.xml - FHIR v3.0.1. Retrieved 25 July 2017, from https://www.hl7.org/fhir/message-request-link.xml.html

HL7 International. (n.d.-j). Messaging - FHIR v3.0.1. Retrieved 25 July 2017, from https://www.hl7.org/fhir/messaging.html

HL7 International. (n.d.-k). Modules - FHIR v3.0.1. Retrieved 13 June 2017, from https://www.hl7.org/fhir/modules.html

HL7 International. (n.d.-l). Observation - FHIR v3.0.1 [Specification]. Retrieved 18 July 2017, from https://www.hl7.org/fhir/observation.html

HL7 International. (n.d.-m). Observation-vitalsigns - FHIR v3.0.1. Retrieved 21 July 2017, from https://www.hl7.org/fhir/observation-vitalsigns.html

HL7 International. (n.d.-n). Patient - FHIR v3.0.1. Retrieved 27 May 2017, from https://www.hl7.org/fhir/patient.html

HL7 International. (n.d.-o). Practitioner - FHIR v3.0.1. Retrieved 21 July 2017, from https://www.hl7.org/fhir/practitioner.html

HL7 International. (n.d.-p). Profiling - FHIR v3.0.1. Retrieved 25 July 2017, from https://www.hl7.org/fhir/profiling.html

HL7 International. (n.d.-q). Resource - FHIR v3.0.1. Retrieved 13 June 2017, from https://www.hl7.org/fhir/resource.html

HL7 International. (n.d.-r). Resourceguide - FHIR v3.0.1. Retrieved 13 June 2017, from https://www.hl7.org/fhir/resourceguide.html#identification

HL7 International. (n.d.-s). Search - FHIR v3.0.1. Retrieved 27 July 2017, from https://www.hl7.org/fhir/search.html

HL7 International. (n.d.-t). Security - FHIR v3.0.1. Retrieved 27 July 2017, from https://www.hl7.org/fhir/security.html

HL7 International. (n.d.-u). Services - FHIR v3.0.1. Retrieved 26 July 2017, from https://www.hl7.org/fhir/services.html

HL7 International. (n.d.-v). Summary - FHIR v3.0.1. Retrieved 25 July 2017, from https://www.hl7.org/fhir/summary.html#2.15

HL7 International. (n.d.-w). Usecases - FHIR v3.0.1. Retrieved 29 July 2017, from https://www.hl7.org/fhir/usecases.html

HL7 International. (n.d.-x). Validation - FHIR v3.0.1. Retrieved 29 July 2017, from https://www.hl7.org/fhir/validation.html

HL7 International. (n.d.-y). Workflow - FHIR v3.0.1. Retrieved 29 July 2017, from https://www.hl7.org/fhir/workflow.html

HL7 International. (n.d.-z). Workflow-ad-hoc - FHIR v3.0.1. Retrieved 29 July 2017, from https://www.hl7.org/fhir/workflow-ad-hoc.html

HL7 International. (n.d.-aa). Workflow-management - FHIR v3.0.1. Retrieved 29 July 2017, from https://www.hl7.org/fhir/workflow-management.html

IHE. (n.d.). Mobile access to Health Documents (MHD) - IHE Wiki. Retrieved 29 July 2017, from http://wiki.ihe.net/index.php/Mobile_access_to_Health_Documents_(MHD)

iNTERFACEWARE Inc. (2008, May 13). Understanding HL7 Messages. Retrieved 25 July 2017, from https://blog.interfaceware.com/understanding-hl7-messages/

Johnson, C. (2012, May). *The HL7 Evolution: Comparing HL7 versions 2 and 3*. Retrieved from https://fr.slideshare.net/Corepoint/comparing-hl7-v3-with-hl7-v2

Landi, H. (2016, July 6). A Clinical Informaticist Shares Why FHIR Won't Extinguish HL7, At Least Not in the Near-Term. Retrieved 27 July 2017, from https://www.linkedin.com/pulse/clinical-informaticist-shares-why-fhir-wont-extinguish-heather-landi

Mandl, K. D., Mandel, J. C., Murphy, S. N., Bernstam, E. V., Ramoni, R. L., Kreda, D. A., … Kohane, I. S. (2012). The SMART Platform: early experience enabling substitutable applications for electronic health records. Retrieved from https://dash.harvard.edu/bitstream/handle/1/10436330/3384120.pdf?sequence=1

Martin, F. (n.d.). Richardson Maturity Model. Retrieved 26 July 2017, from https://martinfowler.com/articles/richardsonMaturityModel.html

McKenzie, L. (2014, November). *Rolling out FHIR - architecture and implementation considerations by …*. Santé. Retrieved from https://fr.slideshare.net/DevDays2014/rolling-out-fhir-architecture-and-implementation-considerations-by-lloyd-mckenzie

Nawanan Theera-Ampornpunt. (2015, November). *HL7 Standards*. Healthcare. Retrieved from https://www.slideshare.net/nawanan/hl7-standards-54853430

niquola. (2014, August 27). fhir.js: JavaScript client for FHIR. Retrieved 24 May 2017, from https://github.com/FHIR/fhir.js

Oracle. (n.d.). Database JSON Developer’s Guide. Retrieved 31 July 2017, from https://docs.oracle.com/database/122/ADJSN/overview-of-storage-and-management-of-JSON-data.htm#ADJSN-GUID-26AB85D2-3277-451B-BFAA-9DD45355FCC7

Phelps, J. (2007, November 3). Service-Oriented Architecture--What Is It, and How Do We Get One? - EQM0739.pdf. University of Wisconsin-Madison. Retrieved from http://www-cdn.educause.edu/ir/library/pdf/EQM0739.pdf

Rouse, M. (n.d.). What is REST (representational state transfer)? - Definition from WhatIs.com. Retrieved 26 July 2017, from http://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer

Rouse, M., & Sutner, S. (2015). What is HL7 (Health Level Seven International)? - Definition from WhatIs.com. Retrieved 25 July 2017, from http://searchhealthit.techtarget.com/definition/Health-Level-7-International-HL7

Sangem, K. (2016, October 28). Nalashaa Solutions. Retrieved 30 May 2017, from http://www.nalashaa.com/part-3-fhir-ehr-follow-fhir-path/

Servage. (2013, April 8). REST principles explained | Servage Magazine. Retrieved 26 July 2017, from https://www.servage.net/blog/2013/04/08/rest-principles-explained/

Shaver, D. (2007, January 22). HL7 101: A Beginner's Guide. Retrieved 29 July 2017, from http://www.fortherecordmag.com/archives/ftr_01082007p22.shtml

SMART. (n.d.). SMART Health IT. Retrieved 23 May 2017, from https://smarthealthit.org/

SMART App Gallery. (n.d.). Retrieved 23 May 2017, from https://apps.smarthealthit.org/

SMART Health IT. (2016, January 27). Swift-SMART: Swift SMART on FHIR framework for iOS and OS X [wiki]. Retrieved 29 May 2017, from https://github.com/smart-on-fhir/Swift-SMART (Original work published 26 June 2014)

SMART Health IT. (2017a). SMART on FHIR. Retrieved 24 May 2017, from http://docs.smarthealthit.org/

SMART Health IT. (2017b). SMART on FHIR -- Clients -- JavaScript. Retrieved 27 May 2017, from http://docs.smarthealthit.org/clients/javascript/

SMART Health IT. (2017c, March 22). Patient Class Reference. Retrieved 27 May 2017, from http://docs.smarthealthit.org/Swift-SMART/Classes/Patient.html

SMART Health IT. (2017d, March 22). SMART on FHIR Python Client: Class List. Retrieved 27 May 2017, from http://docs.smarthealthit.org/client-py/annotated.html

SMART Health IT. (2017e, March 22). SMART on FHIR Python Client: fhirclient.models.patient.Patient Class Reference. Retrieved 27 May 2017, from http://docs.smarthealthit.org/client-py/classfhirclient_1_1models_1_1patient_1_1_patient.html

SMART Health IT. (2017f, March 22). SMART on FHIR Python Client: SMART FHIR Client. Retrieved 27 May 2017, from http://docs.smarthealthit.org/client-py/

Spronk, R. (2007, November 16). HL7 Message examples: version 2 and version 3. Retrieved 25 July 2017, from http://www.ringholm.com/docs/04300_en.htm

Transcend Insights. (2017, February 22). Survey: Patients Regard Open Access to Their Medical Records as Critical to Receiving High Quality Health Care. Retrieved 29 July 2017, from http://www.businesswire.com/news/home/20170222005883/en/Survey-Patients-Regard-Open-Access-Medical-Records

University Health Network. (n.d.-a). Enumerations.AdministrativeGender (HAPI FHIR Structures - DSTU3 2.5 API). Retrieved 24 July 2017, from http://hapifhir.io/apidocs-dstu3/index.html

University Health Network. (n.d.-b). HAPI FHIR - The Open Source FHIR API for Java. Retrieved 14 July 2017, from http://hapifhir.io/

University Health Network. (n.d.-c). Introduction - HAPI FHIR. Retrieved 25 July 2017, from http://hapifhir.io/doc_intro.html

University Health Network. (n.d.-d). JPA Server - HAPI FHIR. Retrieved 25 July 2017, from http://hapifhir.io/doc_jpa.html

University Health Network. (n.d.-e). Practitioner (HAPI FHIR Structures - DSTU3 2.5 API). Retrieved 24 July 2017, from http://hapifhir.io/apidocs-dstu3/index.html

University Health Network. (n.d.-f). RESTful Client - HAPI FHIR. Retrieved 14 July 2017, from http://hapifhir.io/doc_rest_client.html

University Health Network. (n.d.-g). RESTful Server - HAPI FHIR. Retrieved 25 July 2017, from http://hapifhir.io/doc_rest_server.html

Use cases of FHIR. (n.d.). Retrieved 27 July 2017, from https://corepointhealth.com/resource-center/hl7-resources/use-cases-fhir

What is REST? (n.d.). Retrieved 26 July 2017, from http://www.restapitutorial.com/lessons/whatisrest.html#

What Is SMART? (2012, September 3). Retrieved 23 May 2017, from https://smarthealthit.org/an-app-platform-for-healthcare/about/

Wood, G. M. (n.d.). *HIMSS Basic Overview [Microsoft Powerpoint]*. Las Vegas.

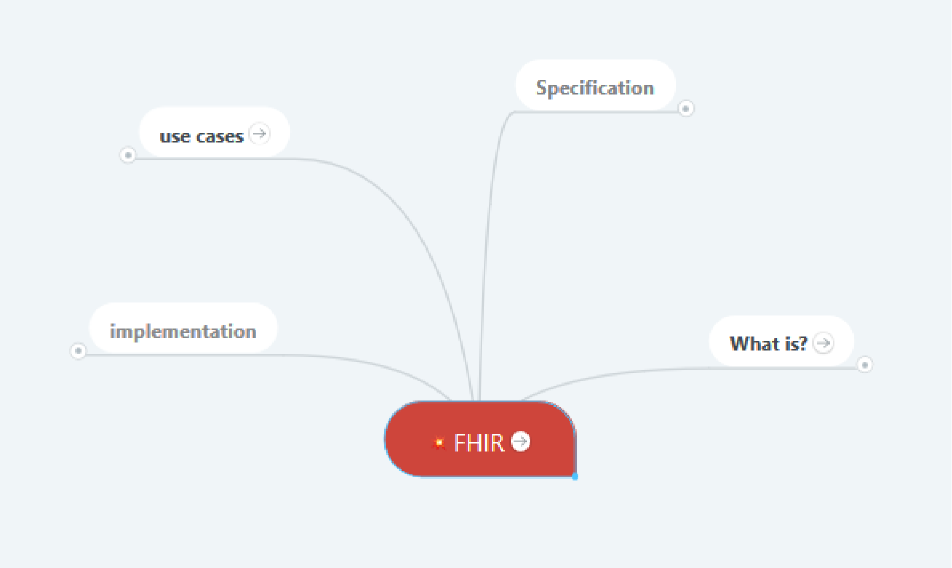
Appendix I: FHIR message example

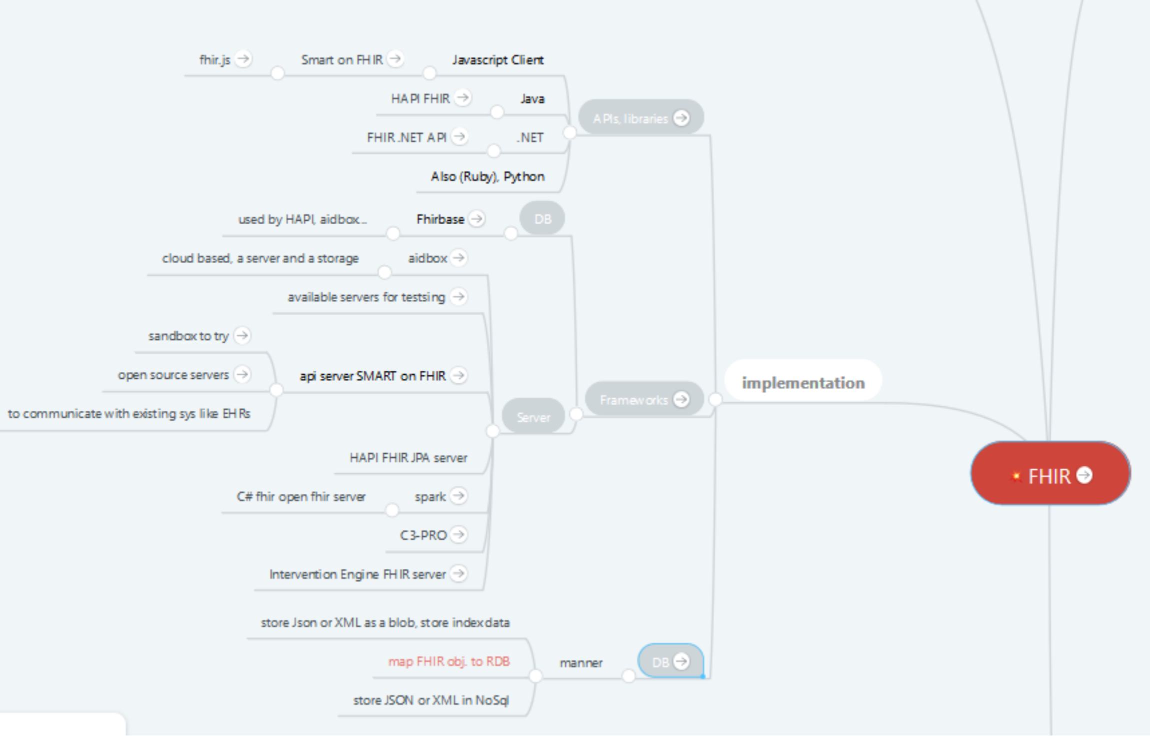
```
<Bundle xmlns="http://hl7.org/fhir">
  <id value="10bb101f-a121-4264-a920-67be9cb82c74"/>
  <type value="message"/>
  <entry>
    <fullUrl value="urn:uuid:267b18ce-3d37-4581-9baa-6fada338038b"/>
    <resource>
      <MessageHeader>
        <id value="efdd254b-0e09-4164-883e-35cf3871715f"/>
        <text>
          <status value="generated"/>
          <div xmlns="http://www.w3.org/1999/xhtml">

            <p> This message is a request to link Patient records 654321
            (Patient Donald DUCK @ Acme Healthcare,
            Inc) and 123456 (Patient Donald D DUCK @ Acme Healthcare,
            Inc)</p>

          </div>
        </text>
        <event>
          <system value="http://hl7.org/fhir/message-events"/>
          <code value="patient-link"/>
        </event>
        <timestamp value="2015-07-14T11:15:33+10:00"/>
        <source>
          <endpoint value="http://example.org/clients/ehr-lite"/>
        </source>
        <responsible>
          <reference value="http://acme.com/ehr/fhir/Practitioner/2323-33-
4"/>
        </responsible>
      <!--    this message is posted to http://acme.com/ehr/fhir,
             with an event 'link' to link 2 patient records, and nominates
             2 patients on the server    -->
        <focus>
          <reference value="http://acme.com/ehr/fhir/Patient/pat1"/>
        </focus>
        <focus>
          <reference value="http://acme.com/ehr/fhir/Patient/pat12"/>
        </focus>
      </MessageHeader>
    </resource>
  </entry>
  <entry>
    <fullUrl value="http://acme.com/ehr/fhir/Patient/pat1"/>
    <resource>
      <Patient>
        <id value="pat1"/>
        <text>
          <status value="generated"/>
          <div xmlns="http://www.w3.org/1999/xhtml">

            <p> Patient Donald DUCK @ Acme Healthcare, Inc. MR = 654321</p>

          </div>
        </text>
        <identifier>
          <use value="usual"/>
          <type>
            <coding>
              <system value="http://hl7.org/fhir/v2/0203"/>
              <code value="MR"/>
            </coding>
          </type>
          <system value="urn:oid:0.1.2.3.4.5.6.7"/>
          <value value="654321"/>
        </identifier>
        <active value="true"/>
        <name>
          <use value="official"/>
          <family value="Donald"/>
          <given value="Duck"/>
        </name>
        <gender value="male"/>
        <contact>
          <relationship>
            <coding>
              <system value="http://hl7.org/fhir/patient-contact-
relationship"/>
              <code value="E"/>
            </coding>
          </relationship>
          <organization>
            <reference value="Organization/1"/>
            <display value="Walt Disney Corporation"/>
          </organization>
        </contact>
        <managingOrganization>
          <reference value="Organization/1"/>
          <display value="ACME Healthcare, Inc"/>
        </managingOrganization>
      </Patient>
    </resource>
  </entry>
  <entry>
    <fullUrl value="http://acme.com/ehr/fhir/Patient/pat12"/>
    <resource>
      <Patient>
        <id value="pat2"/>
        <text>
          <status value="generated"/>
          <div xmlns="http://www.w3.org/1999/xhtml">

            <p> Patient Donald D DUCK @ Acme Healthcare, Inc. MR =
123456</p>

          </div>
        </text>
        <identifier>
          <use value="usual"/>
          <type>
            <coding>
              <system value="http://hl7.org/fhir/v2/0203"/>
              <code value="MR"/>
            </coding>
          </type>
          <system value="urn:oid:0.1.2.3.4.5.6.7"/>
          <value value="123456"/>
        </identifier>
        <active value="true"/>
        <name>
          <use value="official"/>
          <family value="Donald"/>
          <given value="Duck"/>
          <given value="D"/>
        </name>
        <gender value="other">
          <extension url="http://example.org/Profile/administrative-
status">
            <valueCodeableConcept>
              <coding>
                <system value="http://hl7.org/fhir/v2/0001"/>
                <code value="A"/>
                <display value="Ambiguous"/>
              </coding>
            </valueCodeableConcept>
          </extension>
        </gender>
        <managingOrganization>
          <reference value="Organization/1"/>
          <display value="ACME Healthcare, Inc"/>
        </managingOrganization>
      </Patient>
    </resource>
  </entry>
</Bundle>
```

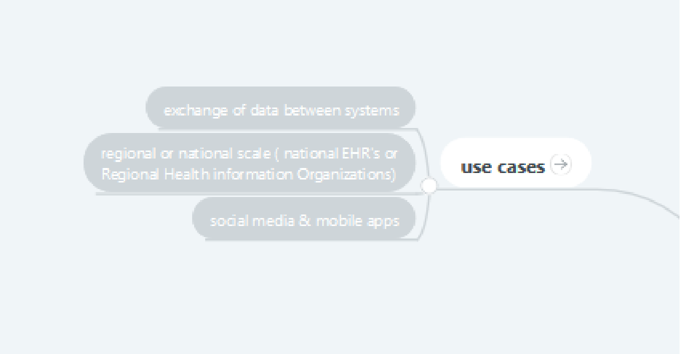
Source: (HL7 International, n.d.-i)

Appendix II: FHIR mind map

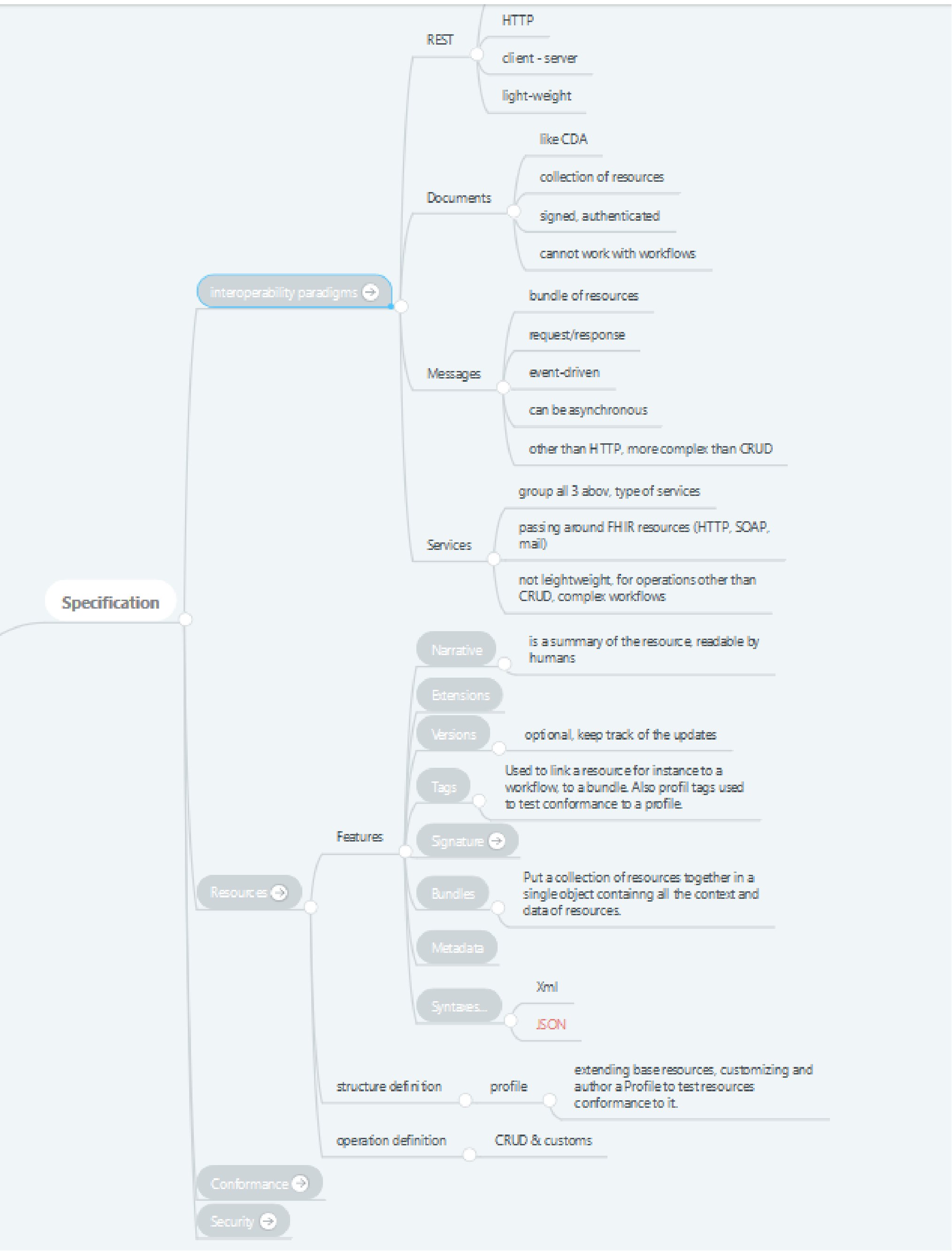
The following image includes the root node of the maps and the main branches. Then the other pictures follow the branches of the map to give better detailed views.



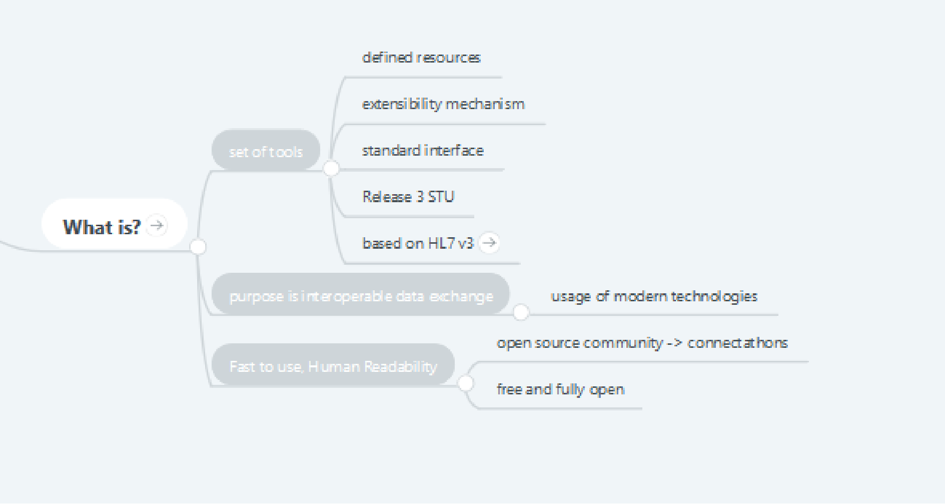




- use cases
  - exchange of data between systems
  - regional or national scale (national EHR's or Regional Health information Organizations)
  - social media & mobile apps

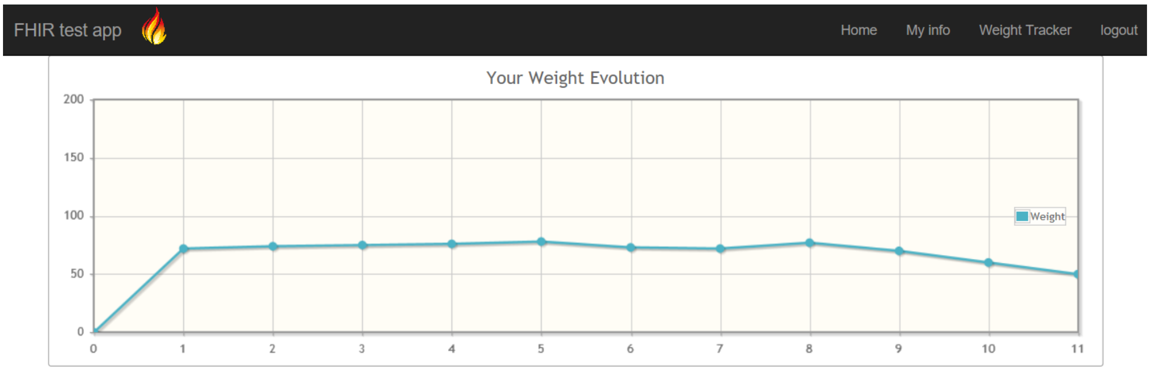


- Specification
  - interoperability paradigms
    - REST
      - HTTP
      - client - server
      - light-weight
    - Documents
      - like CDA
      - collection of resources
      - signed, authenticated
      - cannot work with workflows
    - Messages
      - bundle of resources
      - request/response
      - event-driven
      - can be asynchronous
      - other than HTTP, more complex than CRUD
    - Services
      - group all 3 abov, type of services
      - passing around FHIR resources (HTTP, SOAP, mail)
      - not leightweight, for operations other than CRUD, complex workflows
  - Resources
    - Features
      - Narrative
        - is a summary of the resource, readable by humans
      - Extensions
      - Versions
        - optional, keep track of the updates
      - Tags
        - Used to link a resource for instance to a workflow, to a bundle. Also profil tags used to test conformance to a profile.
      - Signature
      - Bundles
        - Put a collection of resources together in a single object containng all the context and data of resources.
      - Metadata
      - Syntaxes...
        - Xml
        - JSON
    - structure definition
      - profile
        - extending base resources, customizing and author a Profile to test resources conformance to it.
    - operation definition
      - CRUD & customs
  - Conformance
  - Security


What is?
set of tools
defined resources
extensibility mechanism
standard interface
Release 3 STU
based on HL7 v3
purpose is interoperable data exchange
usage of modern technologies
Fast to use, Human Readability
open source community -> connectathons
free and fully open

Source: author’s data

Appendix III: Weight tracker function



source: author's data

Author’s declaration

Je déclare, par ce document, que j'ai effectué le travail de bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après :

- Fabien Dubosson, collaborateur à l’Institut Informatique de gestion.
- Michael Schumacher, professeur HES et collaborateur à l’Institut Informatique de gestion.

Camille Pellaton, 02.08.2017.